Advanced C++ Programming

CIS29

Joseph Bentley

DeAnzaCollege
Computer Information System
December 2022

Contents

Review

Classes, Constructors, and Destructors

Example 1 – Card and Deck class (old code)

```
#include <iostream>
#include <cstdlib>            // needed for rand() function
using namespace std;

const char* const value_name[13] =
 {"two","three","four","five","six","seven","eight","nine","ten",
  "jack","queen","king","ace"};
const char* const suit_name[4] =
 {"clubs","diamonds","hearts","spades"};
const unsigned short DeckSize = 52;

class Card
{
public:
    enum suitType { clubs, diamonds, hearts, spades };
    Card ();
    void assign(unsigned short);
    int get_value(void) const
    {
        return value;
    }
    int get_suit(void) const
    {
        return suit;
    }
    void print(void) const;
private:
    unsigned short value;
    suitType suit;
};

Card::Card() : value(0), suit(clubs)
{}


void Card::assign(unsigned short x)
{
    value = x % 13;
    suit = (suitType) (x % 4);
}

void Card::print(void) const
{
    cout << (value_name[value]) << " of "
<< (suit_name[suit]) << endl;
```

```
}

class Deck
{
public:
    Deck();
    void print(void) const;
private:
    Card    card[DeckSize];
    void shuffle(void);
};

Deck::Deck()
{
    unsigned short i;
    for (i = 0; i < DeckSize; i++) card[i].assign(i);
    shuffle();
}

void Deck::shuffle(void)
{
    unsigned short i;
    unsigned short k;
    Card temp;
    cout << "I am shuffling the Deck\n";
    for (i = 0; i < DeckSize; i++)
    {
        k = rand() % DeckSize;
        temp = card[i];
        card[i] = card[k];
        card[k] = temp;
    }
}

void Deck::print(void) const
{
    for (unsigned short i = 0; i < DeckSize; i++) card[i].print();
}


int main(void)
{
    Deck poker;
    poker.print();
    return 0;
}
```

```
*****  Output  *****

I am shuffling the Deck
four of diamonds
ten of clubs
jack of hearts
jack of diamonds
six of diamonds
```

```
nine of clubs
…
eight of clubs
```

<u>Review questions</u>

Line 5: what does “const char* const” mean?

Line 9: why not **#define DeckSize 52** ?

Line 14: enum suitType{ clubs, diamonds, hearts, spades };
Is this a declaration or a definition?
Does it have to be placed inside the class definition?
What are the implications/constraints/requirements of placing it inside the class definition?

Line 17: What’s this?

Line31: What’s this?

Line 38: Is this a 4-letter word? (suitType)
How else can you write this line?

What is the relationship between Card and Deck?

Lines 57-62: What if you write the Deck constructor as …

```
Deck::Deck()
{
for (unsignedshort i = 0; i < DeckSize; i++) {
     card[i].assign(i);
  }
  shuffle();
}
```

What’s the difference?

Scope?

How many constructor calls take place when line 90 is executed?

Why are there no destructors in this example?

Example 2 – Card and Deck class (revised)

```
1  #include <iostream>
2  #include <cstdlib>             // needed for rand() function
3  #include <string>
4  using namespace std;
5
6  const unsigned short DeckSize = 52;
```

```
class Card
{
public:
    enum suitType { clubs, diamonds, hearts, spades };
    static const string value_name[13];
    static const string suit_name[4];

    Card ();
    Card (int);
    int get_value(void) const
    {
        return value;
    }
    suitType get_suit(void) const
    {
        return suit;
    }
private:
    int value;
    suitType suit;
    static int default_card_initializer;
};

int Card::default_card_initializer = 0;

const string Card::value_name[13] =
   {"two","three","four","five","six","seven",
    "eight","nine","ten","jack","queen","king","ace"};
const string Card::suit_name[4] =
   {"clubs","diamonds","hearts","spades"};

Card::Card()
    : value(default_card_initializer % 13),
      suit(static_cast<suitType>(default_card_initializer % 4))
{
    ++default_card_initializer;
}

Card::Card(int x)
    : value(x % 13),
      suit(static_cast<suitType>(x % 4))
{}

ostream& operator<<(ostream& out, const Card& card)
{
    out << (Card::value_name[card.get_value()])
<< " of "
<< (Card::suit_name[card.get_suit()]);
    return out;
}

class Deck
{
public:
```

```
    Deck();
    const Card* get_card() const
    {
        return card;
    }
    Card get_card(int index) const
    {
        return card[index];
    }
private:
    Card    card[DeckSize];
    void shuffle();
    friend ostream& operator<<(ostream& out, const Deck& deck);
};


Deck::Deck()
{
    shuffle();
}

void Deck::shuffle()
{
    int k;
    Card temp;
    cout << "I am shuffling the Deck\n";
    for (int i = 0; i < DeckSize; i++)
    {
        k = rand() % DeckSize;
        temp = card[i];
        card[i] = card[k];
        card[k] = temp;
    }
}

ostream& operator<<(ostream& out, const Deck& deck)
{
    for (Card c : deck.card)  // range-based for loop
        out << c << endl;
    return out;
}


int main(void)
{
    Deck poker;
    cout << poker << endl;
}
```

Example 3 – Card and Deck class (another revision)

```
#include <iostream>
#include <cstdlib>            // needed for rand() function
```

```cpp
#include <string>
using namespace std;

class Card
{
public:
    enum suitType  { clubs, diamonds, hearts, spades };
    static const string value_name[13];
    static const string suit_name[4];

    Card ();
    Card (int);
    int get_value(void) const
    {
        return value;
    }
    suitType get_suit(void) const
    {
        return suit;
    }
private:
    int value;
    suitType suit;
    static int default_card_initializer;
    friend ostream& operator<<(ostream& out, const Card& card);
};

int Card::default_card_initializer = 0;

const string Card::value_name[13] =
{
    "two","three","four","five","six","seven",
    "eight","nine","ten","jack","queen","king","ace"
};
const string Card::suit_name[4] =
{"clubs","diamonds","hearts","spades"};

Card::Card()
: value(default_card_initializer % 13),
suit(static_cast<suitType>(default_card_initializer % 4))
{
    ++default_card_initializer;
}

Card::Card(int x)
: value(x % 13), suit(static_cast<suitType>(x % 4))
{}

ostream& operator<<(ostream& out, const Card& card)
{
    out << (Card::value_name[card.value])
<< " of "
<< (Card::suit_name[card.suit]);
    return out;
}
```

```

class Deck
{
public:
    Deck();
    Deck(const Deck&);
    ~Deck() { delete [] cards; cards = 0;}
    Deck& operator= (const Deck&);
    const Card* get_cards() const
    {
        return cards;
    }
    Card get_cards(int index) const
    {
        return cards[index];
    }
private:
    static const unsigned short DeckSize;
    Card*    cards;
    void shuffle();
friend ostream& operator<<(ostream& out, const Deck& deck);
};

const unsigned short Deck::DeckSize = 52;

Deck::Deck() : cards(new Card[DeckSize])
{
    shuffle();
}

Deck::Deck(const Deck& anotherDeck)
    : cards(new Card[DeckSize])
{
    for (auto i = 0; i < DeckSize; ++i)
    {
        cards[i] = anotherDeck.cards[i];
    }
}

Deck& Deck::operator=(const Deck& anotherDeck)
{
    if (cards) delete [] cards;
    cards = new Card[DeckSize];
    for (auto i = 0; i < DeckSize; ++i)
    {
        cards[i] = anotherDeck.cards[i];
    }
    return *this;
}


void Deck::shuffle()
{
    int k;
    Card temp;
```

```
113     cout << "I am shuffling the Deck\n";
114     for (auto i = 0; i < DeckSize; i++)
115     {
116         k = rand() % DeckSize;
117         temp = cards[i];
118         cards[i] = cards[k];
119         cards[k] = temp;
120     }
121 }
122
123 ostream& operator<<(ostream& out, const Deck& deck)
124 {
125     for (auto i = 0; i < Deck::DeckSize; ++i)
126         out << deck.cards[i] << endl;
127     return out;
128 }
129
130 int main(void)
131 {
132     Deck poker;
133     cout << poker << endl;
134 }
```

```
*****  Output  *****

I am shuffling the Deck
four of diamonds
ten of clubs
jack of hearts
jack of diamonds
six of diamonds
nine of clubs
ace of diamonds
…
```

Review questions

Lines 63 - 65: copy constructor, destructor, overloaded assignment operator – why?

Line 83: syntax

Line 91: auto

Lines 97-106: how to write an overloaded assignment operator

Lines 27 and 108: Do you have to have friends?

Example 4 – Adding Matrices

```
#include <iomanip>
#include <iostream>
#include <cstdlib>  // for rand()
```

```
using namespace std;

class Matrix
{
private:
    int** element;
    int rows;
    int cols;
    void alloc();
    void release();
public:
    Matrix(int = 0, int = 0);  // also default constructor
    Matrix(const Matrix&); // copy constructor
    ~Matrix();
    Matrix operator+(const Matrix&) const;
    Matrix& operator=(const Matrix&);
    friend ostream& operator<<(ostream&, const Matrix&);
};

int main()
{
    Matrix A(3,4), B(3,4), C;
    cout << A << endl;
    cout << B << endl;
    cout << C << endl;
    C = A + B;
    cout << C << endl;
}

Matrix::Matrix(int r, int c) : rows(r), cols(c)
{
    cout << "Constructor called for object " << this <<endl;
    alloc();

    // initialize Matrix elements with random numbers 0-9
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            element[i][j] = rand()%10;
}

Matrix::Matrix(const Matrix& arg) : rows(arg.rows), cols(arg.cols)
{
    cout << "\nIn copy constructor for object " << this;
    cout << ", argument: " << &arg << endl;

    alloc();
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            element[i][j] = arg.element[i][j];
}

Matrix::~Matrix()
{
    cout << "\n~~ Destructor called for object: " << this << endl;

```

```
    release();
}

void Matrix::alloc()        // allocate heap memory for elements
{
    cout << "Allocate memory for Matrix " << this << " elements\n";

    element = new int*[rows];
    for (int i = 0; i < rows; i++)
        element[i] = new int[cols];
}

void Matrix::release()
{
    cout << "I got rid of Matrix " << this << "'s elements\n";

    for (int i = 0; i < rows; i++)
        delete [] element[i];
    delete [] element;
}

Matrix Matrix::operator+(const Matrix& arg) const
{
    cout << "\nExecuting operator+ for object: " << this;
    cout << ", argument: " << &arg << endl;

    if (rows != arg.rows || cols != arg.cols)
    {
        cerr << "Invalid Matrix addition\n";
        return (*this);
    }

    Matrix temp(rows,cols);

    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            temp.element[i][j] = element[i][j] + arg.element[i][j];

    cout << temp << endl;
    return temp;
}

Matrix& Matrix::operator=(const Matrix& arg)
{
    cout << "\nExecuting operator= for object: " << this;
    cout << ", argument: " << &arg << endl;

    // Make sure rows and cols match the argument
    if (rows != arg.rows || cols != arg.cols)
    {
        release();
        rows = arg.rows;
        cols = arg.cols;
        alloc();
    }
```

```

    for (int i = 0; i < arg.rows; i++)
        for (int j = 0; j < arg.cols; j++)
            element[i][j] = arg.element[i][j];

    return *this;
}

ostream& operator<<(ostream& out, const Matrix& m)
{
    out << "\nMatrix values for object: "<< &m << endl;

    out << "----------------\n";

    for (int i = 0; i < m.rows; i++)
    {
        for (int j = 0; j < m.cols; j++)
            out << setw(4) << m.element[i][j];
        out << endl;
    }
    out << "----------------";
    return out;
}
```

****** Output ******

```
Constructor called for object 0xffffcb80
Allocate memory for Matrix 0xffffcb80 elements
Constructor called for object 0xffffcb70
Allocate memory for Matrix 0xffffcb70 elements
Constructor called for object 0xffffcb60
Allocate memory for Matrix 0xffffcb60 elements

Matrix values for object: 0xffffcb80
----------------
   3   3   2   9
   0   8   2   6
   6   9   1   1
----------------

Matrix values for object: 0xffffcb70
----------------
   3   5   8   3
   0   6   9   2
   7   7   2   8
----------------

Matrix values for object: 0xffffcb60
----------------
----------------

Executing operator+ for object: 0xffffcb80, argument: 0xffffcb70
Constructor called for object 0xffffcb00
Allocate memory for Matrix 0xffffcb00 elements
```

```
Matrix values for object: 0xffffcb00
----------------
   6   8  10  12
   0  14  11   8
  13  16   3   9
----------------

In copy constructor for object 0xffffcb90, argument: 0xffffcb00
Allocate memory for Matrix 0xffffcb90 elements

~~ Destructor called for object: 0xffffcb00
I got rid of Matrix 0xffffcb00's elements

Executing operator= for object: 0xffffcb60, argument: 0xffffcb90
I got rid of Matrix 0xffffcb60's elements
Allocate memory for Matrix 0xffffcb60 elements

~~ Destructor called for object: 0xffffcb90
I got rid of Matrix 0xffffcb90's elements

Matrix values for object: 0xffffcb60
----------------
   6   8  10  12
   0  14  11   8
  13  16   3   9
----------------

~~ Destructor called for object: 0xffffcb60
I got rid of Matrix 0xffffcb60's elements

~~ Destructor called for object: 0xffffcb70
I got rid of Matrix 0xffffcb70's elements

~~ Destructor called for object: 0xffffcb80
I got rid of Matrix 0xffffcb80's elements
```

Maybe You Haven’t Covered This

Conversion Operators

Example 5 - Conversion of a user-defined type to a primitive type

```
#include <iostream>
using namespace std;

class B
{
    int b;
public:
    B(int i) : b(i) {}
```

```
    operator int() const;
};

B::operator int() const
{
    cout << "* B:: operator int() called\n";
    return b;
}

int main()
{
    B eight(8);
    cout << eight << endl;
    cout << eight + 5 << endl;
    cout << 5 + eight << endl;
    cout << (eight > 3) << endl;
}
```

```
******  Output  ******
* B:: operator int() called
8
* B:: operator int() called
13
* B:: operator int() called
13
* B:: operator int() called
1
```

✔ What would happen if operator int() was not defined?

Example 6 - More Conversions of a user-defined type

```
#include <iostream>
#include <string>
using namespace std;

class Day;    // forward declaration

class Number
{
    int n;
public:
    Number(int i) : n(i)
    {
        cout << "Number(int) ctor called\n";
    }
    operator int() const;
    operator Day() const;

};

Number::operator int() const
{
```

```
    cout << "* Number::operator int() called\n";
    return n;
}

const string Days[7] =
{
    "Sunday","Monday","Tuesday","Wednesday","Thursday",
    "Friday","Saturday"
};

class Day
{
    string dow;
public:
    Day(int n) : dow(Days[n%7])
    {
        cout << "Day(int) ctor called\n";
    }
    operator Number() const; // convert Day to Number
    void operator!() const
    {
        cout << "dow = " << dow << endl;
    }
};


Day::operator Number() const
{
    cout << "** Day:: operator Number() called\n";
    for (int i = 0; i < 7; i++)
        if (dow == Days[i]) return Number(i);
    return Number(-1);
}

Number::operator Day() const          // Why is this function here?
{
    cout << "*** Number::operator Day() called\n";
    return n; //Day(n);
}

void somefunction(Day)
{
    cout << "somefunction called\n";
}


int main()
{
    Number N1(65);

    cout << "N1 = " << N1 << endl;

    Day d1(1);
    !d1;

```

```
    //  Day d2(N1);          Why is this an ambiguity?

    Number N2(d1);
    cout << "N2 = " << N2 << endl;
    !Day(Number(d1)+2);

    somefunction(N1);
}
```

```
******  Output  ******

Number(int) ctor called
* Number::operator int() called
N1 = 65
Day(int) ctor called
dow = Monday
** Day:: operator Number() called
Number(int) ctor called
* Number::operator int() called
N2 = 1
** Day:: operator Number() called
Number(int) ctor called
* Number::operator int() called
Day(int) ctor called
dow = Wednesday
*** Number::operator Day() called
Day(int) ctor called
somefunction called
```

Explicit Constructors

The keyword ***explicit*** is used to specify that a constructor may only be used for object instantiation and not for automatic conversion. Here's an example that demonstrates the effect.

Example 7 – Explicit constructors

```
#include <iostream>
using namespace std;

class A
{
public:
    A(int);                      // non-explicit ctor
};


class B
{
public:
    explicit B(int);    // explicit ctor
};

A::A(int)
```

```
{
    cout << "A ctor called for object " << this << endl;
}

B::B(int)                    // do not repeat keyword explicit
{
    cout << "B ctor called for object " << this << endl;
}

void funkA(A object)
{
    cout << "funkA called\n";
}

void funkB(B object)
{
    cout << "funkB called\n";
}

void funkAB(A obj)
{
    cout << "funkAB(A) called\n";
}

void funkAB(B obj)
{
    cout << "funkAB(B) called\n";
}

int main()
{
    A objA(2);          // instantiate an A object
    B objB(3);          // instantiate a B object

    funkA(objA);   // call funkA() with an exact argument match

    funkA(9);           // call funkA() with an non-exact match

    funkB(objB);   // call funkB() with an exact argument match

    //  funkB(16);  // error: cannot convert int to a B object

    funkAB(6);          // compile error if B(int) is not explicit
}
```

****** Output ******

```
A ctor called for object 0x6dfefd
B ctor called for object 0x6dfefc
funkA called
A ctor called for object 0x6dfefe
funkA called
funkB called
A ctor called for object 0x6dfeff
funkAB(A) called
```

typedef and using

The keyword, typedef, originally from C, is used to define a type.

C++ 11 introduced the keyword, using to act like typedef.

typeid operator

The typeid operator returns an identifier of a type, a variable or an expression. The return of the typeid is a class type, called type_info. You can use the name() member function of the type_info class to display a literal description of the type.

Example 8 – typedef, using, typeid

```
#include <iostream>
#include <typeinfo> // for typeid
using namespace std;

int main()
{
    typedef int number;
    number n;

    typedef long long int bignumber;
    bignumber biggie;

    typedef double(*ptr2arrayof10)[10];
    double d[13][10];
    ptr2arrayof10 p = d;

    using Word = unsigned int;
    Word seven = 7U;

    using pint = int*;
    pint addr_n = &n;

    using Int4 = int[4];
    Int4 iota4 = {1,2,3,4};

    cout << "typeid(int).name()=" << typeid(int).name() << endl;
    cout << "typeid(bignumber).name()=" << typeid(bignumber).name()
         << endl;
    cout << "typeid(biggie).name()=" << typeid(biggie).name()
         << endl;
    cout << "typeid(p).name()=" << typeid(p).name() << endl;
    cout << "typeid(ptr2arrayof10).name()="
         << typeid(ptr2arrayof10).name() << endl;
    cout << "typeid(seven).name()=" << typeid(seven).name()
         << endl;
    cout << "typeid(Word).name()=" << typeid(Word).name() << endl;
```

```
    cout << "typeid(pint).name()=" << typeid(pint).name() << endl;
    cout << "typeid(addr_n).name()=" << typeid(addr_n).name()
         << endl;
    cout << "typeid(Int4).name()=" << typeid(Int4).name() << endl;
    cout << "typeid(iota4).name()=" << typeid(iota4).name()
         << endl;
}
```

****** Code::Blocks / NetBeans / Eclipse / Linux / Mac Xcode ******

```
typeid(int).name()=i
typeid(bignumber).name()=x
typeid(biggie).name()=x
typeid(p).name()=PA10_d
typeid(ptr2arrayof10).name()=PA10_d
typeid(seven).name()=j
typeid(Word).name()=j
typeid(pint).name()=Pi
typeid(addr_n).name()=Pi
typeid(Int4).name()=A4_i
typeid(iota4).name()=A4_i
```

****** MS Visual Studio 2019 ******

```
typeid(int).name()=int
typeid(bignumber).name()=__int64
typeid(biggie).name()=__int64
typeid(p).name()=double (*)[10]
typeid(ptr2arrayof10).name()=double (*)[10]
typeid(seven).name()=unsigned int
typeid(Word).name()=unsigned int
typeid(pint).name()=int *
typeid(addr_n).name()=int *
typeid(Int4).name()=int [4]
typeid(iota4).name()=int [4]
```

Some C++ 11/14/17/20 Features

auto type

Using the auto keyword, a variable's type may be automatic assigned. The new usage of the auto keyword negates the former ansi-C storage class meaning.

the decltype operator

The decltype operator is similar to auto, it returns the type of an expression.

Example 1 – auto type and decltype

```
#include <iostream>
#include <typeinfo> // for typeid
using namespace std;

int main()
{
    auto v1 = 7;                              // v1 is type int
    auto mygrade ='a';                        // mygrade is type char
    auto pi = 31.4;                           // pi is type double
     auto cstring = "have a nice day";        // pointer to const char
     auto ptr2char = &mygrade;                // pointer to char
     auto z = "zebra"[0];                     // z is type char

     cout << typeid(v1).name()  << endl;
     cout << typeid(mygrade).name()  << endl;
     cout << typeid(pi).name()  << endl;
     cout << typeid(cstring).name()  << endl;
     cout << typeid(ptr2char).name()  << endl;
     cout << typeid(z).name()  << endl;

     typedef decltype(7) myint;
     myint x;
     cout << typeid(x).name()  << endl;

     decltype(7) y;
     cout << typeid(y).name()  << endl;

     // Somewhat practical
     int array[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
     cout << typeid(array).name() << endl;
     cout << typeid(array[1]).name() << endl;
     cout << typeid(*array).name() << endl;
     cout << typeid(&array).name() << endl;
 }
```

****** Code::Blocks / NetBeans / Linux ******

```
i
```

```
c
d
PKc
Pc
c
i
i
A3_A4_i
A4_i
A4_i
PA3_A4_i
```

****** MS Visual Studio 2017 ******

```
int
char
double
char const *
char *
char
int
int
int [3][4]
int [4]
int [4]
int (*)[3][4]
```

the constexpr specifier

The constexpr specifier declares that a function or variable is const at compile time.

Examples

```
constexpr float pi = 3.14;

constexpr float areaOfCircle(float radius)
{
    return pi * radius * radius;
}

constexpr float area1 = areaOfCircle(1);

const float two = 2.f;
constexpr float area2 = areaOfCircle(two);

float three = 3.f;
constexpr float area32 = areaOfCircle(three);  // ERROR
```

nullptr

nullptr is a pointer constant with conversions to any pointer type. It is used as a replacement for the macro, NULL or a 0 pointer.

char*ptr = nullptr;

void somefunk(type* ptr = nullptr);

if (p == nullptr) …

Uniform initialization/Brace/List initialization

int I{7}; // instead of int I = 7;

int zero{}; // same as int zero = 0;

string s{“apple pie”};

SomeClass object{19}; // instead of SomeClass object(19);

AnotherClass obj{thing,23,2.5,’a’}; // instead of AnotherClass obj(thing,23,2.5,’a’);

Range-based for loop

Example 2 – Range-based for loop

```
#include <iostream>
using namespace std;

int main()
{
    int array[5] = {2,3,5,7,11};
    for (int i : array)
        cout << i << "  ";
    cout << endl;

     for (auto i : array)
         cout << i << "  ";
     cout << endl;

     for (auto i : array)
         i = 13;

     for (auto i : array)
         cout << i << "  ";
     cout << endl;

     for (auto& i : array)
         i = 13;

     for (auto i : array)
```

```
            cout << i << "  ";
        cout << endl;

        for (auto value : {9,8,7,6} )    // note initializer list
        {
            cout << value << "  ";
        }
        cout << endl;
    }
```

****** Output ******

```
2   3   5   7   11
2   3   5   7   11
2   3   5   7   11
13   13   13   13   13
9   8   7   6
```

Defaulted and deleted constructors

The default specifier with the default constructor causes the compiler to generate it. The delete specifier is used to disable a constructor.

```
class ABC
{
   int a,b,c;
Public:
   ABC() = default;            // same as ABC(){}
   ABC(int, int, int);
   ABC(const ABC&) = delete;   // disable copy constructor
   …
};
```

The override specifier

The keyword override specifier is a way to ensure that a virtual function in a derived class overrides the analogous function in the base class.

```
class Base
{
…
public:
   virtual void funk1(int);
   virtual void funk2(float);
   virtual void funk3(string);
…
};

class Derived : public Base
{
```

```
…
public:
   virtual void funk1(int);   // overrides funk1 in Base class
                              // funk2 is not overridden
   virtual void funk3(string) override; // funk3 is overridden
   virtual void funk4(char) override;   // ERROR
…
};
```

R-value references

R-value references permits a reference to bind to an r-value – a temporary or a literal. This is useful for the *move constructor* or the *move assignment operator*, avoiding the expense of copying an object for this purpose.

Example 3 – R-value References

```
#include <iostream>
#include <utility>  // for move
using namespace std;

void increment(int& value)
{
    cout << "increment with lvalue reference argument" << endl;
    ++value;
}

void increment(int&& value)
{
    cout << "increment with rvalue reference argument" << endl;
    ++value;
}

int main()
{
    int i = 1;

    // Increment a variable
    increment(i);
    cout << "i=" << i << endl;

    // Increment an expression
    increment(i + 5);

    // Increment a literal constant
    increment(3);
}
```

****** Output ******

```
increment with lvalue reference argument
i=2
increment with rvalue reference argument
increment with rvalue reference argument
```

Move Semantics

With the use of rvalue references in C++11, the move constructor and the move assignment operator was added as a replacement for the copy constructor and the overloaded assignment operator.

Example 4 – Move Semantics

```
#include <iostream>
#include <cstring>
#include <utility>   // for move
using namespace std;

class Student
{
    char* name;
public:
     Student();                              // default constructor
     Student(const char* n);
     Student(const Student& obj);            // copy constructor
     Student(Student&& obj);                 // move constructor
     ~Student();                             // destructor
     Student& operator=(const Student& obj); // assignment operator
     Student& operator=(Student&& obj);      // move assignment
     const char* getName() const
     {
         return name ? name : "";
     }
 };

 ostream& operator<<(ostream& out, const Student& obj)
 {
     return out << "object=" << &obj << " name=" << obj.getName();
 }

 Student::Student() : name(nullptr)
 {
     cout << "> In default constructor: " << *this << endl;
 }

 Student::Student(const char* n)
 : name(new char[strlen(n)+1])
 {
     strcpy(name,n);
     cout << "> In Student(const char* n) ctor: " << *this << endl;
```

```
}

Student::Student(const Student& obj)
: name(new char[strlen(obj.name+1)])
{
    strcpy(name,obj.name);
    cout << "> In copy constructor: " << *this << endl;
}

Student::Student(Student&& obj)
: name(new char[strlen(obj.name+1)])
{
    strcpy(name,obj.name);
    cout << "> In move constructor: " << *this << endl;
    delete [] obj.name;
    obj.name = nullptr;
}

Student::~Student()
{
    cout << "~ Student destructor " << *this << endl;
    if (name) delete [] name;
    name = nullptr;
}

Student& Student::operator=(const Student& obj)
{
    delete [] name;
    name = new char[strlen(obj.name+1)];
    strcpy(name,obj.name);
    cout << "= In assignment operator: " << *this << endl;
    return *this;
}

Student& Student::operator=(Student&& obj)
{
    delete [] name;
    name = obj.name;
    cout << "= In move assignment operator: " << *this << endl;
    obj.name = nullptr;
    return *this;
}

Student create()
{
    cout << "In create()\n";
    return Student("Temporary");;
}

int main()
{
    cout << "Executing line => Student j(\"Joe\");" << endl;
    Student j("Joe");
    cout << "j = " << j << endl;

```

```
    cout << "\nExecuting line => Student h(j);" << endl;
    Student h(j);

    cout << "\nExecuting line => h = j;" << endl;
    h = j;

    cout << "\nExecuting line => j = create();" << endl;
    j = create();
    cout << "j = " << j << endl;

    cout << "\nExecuting line => Student k(move(j));" << endl;
    Student k(move(j));
    cout << "k = " << k << endl;
    cout << "j = " << j << endl;
    cout << "\nThat's all folks!!!" << endl;
}
```

****** Output ******

```
Executing line => Student j("Joe");
> In Student(const char* n) ctor: object=0x61fe00 name=Joe
j = object=0x61fe00 name=Joe

Executing line => Student h(j);
> In copy constructor: object=0x61fdf8 name=Joe

Executing line => h = j;
= In assignment operator: object=0x61fdf8 name=Joe

Executing line => j = create();
In create()
> In Student(const char* n) ctor: object=0x61fe08 name=Temporary
= In move assignment operator: object=0x61fe00 name=Temporary
~ Student destructor object=0x61fe08 name=
j = object=0x61fe00 name=Temporary

Executing line => Student k(move(j));
> In move constructor: object=0x61fdf0 name=Temporary
k = object=0x61fdf0 name=Temporary
j = object=0x61fe00 name=

That's all folks!!!
~ Student destructor object=0x61fdf0 name=Temporary
~ Student destructor object=0x61fdf8 name=Joe
~ Student destructor object=0x61fe00 name=
```

Default class member initializer

Non-static class data members may contain a default initializer in the class definition. This default initializer can be overridden in a contructor initialization list or in the body of a constructor.

Example 5 –Default class member initializer

```
#include <iostream>
using namespace std;

class DMI
{
    int a = 0;
    int b = 1;
    int c = 2;
public:
    DMI();
    int geta() const { return a; }
    int getb() const { return b; }
    int getc() const { return c; }
};

DMI::DMI() : a(5), b(6) { b = 8; c = 9; }

ostream& operator<<(ostream& out, const DMI& obj)
{
    out << obj.geta() << ' ' << obj.getb() << ' ' << obj.getc();
    return out;
}


int main()
{
    DMI object;
    cout << object << endl;
}
```

****** Output ******

```
5 8 9
```

Explanation

Each member of the DMI class has a default member initializer. Class member initialiations are overridden as follows:

- a is overridden by the constructor initializer
- b is overridden by the constructor initializer, and then overridden in the body of the constructor
- c is overridden in the body of the constructor

The generic size function

The generic size function was introduced in C++ 17. It is used to return the size of an array (number of elements) or a C++ container. It requires the <iterator> header file.

Example 6 – The size function

Note: this example must be compiled using a C++17 compiler.

```
#include <iostream>
#include <iterator>
#include <vector>
using namespace std;

int main()
{
    int a[5];
    int b[] = {1,2,3};
     vector<int> v{3,4,5,6};

     cout << size(a) << endl;
     cout << size(b) << endl;
     cout << size(v) << endl;
 }
```

****** Output ******

```
5
3
4
```

Binary File I/O

istream member functions

read

Read a specified number of characters from an input stream and stores them in a char array. The array is not null-terminated.

```
istream& read (char* s, streamsize[1] n);
```

peek

Returns the next character to be read without extracting it from the input stream.

```
int peek();
```

seekg

Sets the next read position in the input stream.

[1] streamsize is used to represent size and character counts. It is a signed integer type.

```
stream& seekg (streampos[2] pos);
istream& seekg (streamoff[3] offset, ios_base::seekdir way);
```

ios_base::seekdir can be one of three constants

Constant	Meaning
beg	Beginning of the input stream
cur	Current position in the input stream
end	End of the input stream

tellg

Returns the next read position in the input stream.

streampos tellg();

Example 1 – istream member functions

Input file

```
HAVE A NICE DAY
have a nice day
This is line 3.
And that's all folks!!!
```

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

int main()
{
    char buffer[32];
    const char* filename = "c:/temp/ex1data.txt";

     ifstream fin(filename);
     if (!fin) {
         cerr << "Unable to open input file " << filename << endl;
         exit(1);
     }

     fin.read(buffer, 9);     // Read the first 9 bytes of the file
     cout << '/' << buffer << '/' << endl;
     buffer[9] = 0;           // Null terminate the buffer
     cout << '/' << buffer << '/' << endl << endl;

```

[2] streampos is used to represent position in a stream. This type is an integer construction or conversion.
[3] streamoff is used to represents an offset of a position in a stream.

```
    cout << "fin.tellg() = " << fin.tellg() << endl;
    cout << "fin.peek() = " << fin.peek() << endl;
    cout << "static_cast<char>(fin.peek()) = " <<
            static_cast<char>(fin.peek()) << endl << endl;

    // Reposition to byte 1
    // fin.seek(1);   ERROR
    fin.seekg(static_cast<streampos> (1));
    cout << "fin.tellg() = " << fin.tellg() << endl << endl;

    // Create a streampos object
    streampos pos = fin.tellg();
    //  pos++;  ERROR
    //  pos = pos + 5; // throws a warning
    pos = 2;
    fin >> buffer;
    cout << "buffer = " << buffer << endl;
    cout << "fin.tellg() = " << fin.tellg() << endl << endl;

    fin.seekg(-2, ios_base::cur);
    fin.read(buffer, 25);
    buffer[25] = 0;
    cout << "buffer = " << buffer << endl << endl;

    fin.seekg(0, ios_base::beg);
    fin.read(buffer, sizeof (buffer) - 1);
    buffer[sizeof (buffer) - 1] = 0;
    cout << "buffer = " << buffer << endl;
}
```

****** Output: NetBeans on Windows ******

```
/HAVE A NI���/
/HAVE A NI/

fin.tellg() = 9
fin.peek() = 67
static_cast<char>(fin.peek()) = C

fin.tellg() = 1

buffer = AVE
fin.tellg() = 4

buffer = VE A NICE DAY
have a nic

buffer = HAVE A NICE DAY
have a nice da
```

****** Output: MS Visual Studio 2017 ******

```
/HAVE A NI╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠F  çöL ²/
/HAVE A NI/
```

```
fin.tellg() = 9
fin.peek() = 67
static_cast<char>(fin.peek()) = C

fin.tellg() = 1

buffer = AVE
fin.tellg() = 4

buffer = VE A NICE DAY
have a nice

buffer = HAVE A NICE DAY
have a nice day
```

****** Output: Code::Blocks on Windows ******

```
/HAVE A NI/
/HAVE A NI/

fin.tellg() = 13
fin.peek() = 67
static_cast<char>(fin.peek()) = C

fin.tellg() = 1

buffer = AVE
fin.tellg() = 8

buffer =  NICE DAY
have a nice day

buffer = HAVE A NICE DAY
have a nice day
```

ostream member functions

write

Write a specified number of characters to an output stream

```
ostream& write (const char* s, streamsize n);
```

seekp

Sets the next write position in the output stream.

```
ostream& seekp (streampos pos);
ostream& seekp (streamoff off, ios_base::seekdir way);
```

tellp

Returns the next write position in the output stream.

```
streampos tellp();
```

Example 2 – ostream member functions

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cstring>
using namespace std;

int main()
{
    const char* filename = "ex2data.bin";

    ofstream fout(filename);
    if (!fout)
    {
        cerr << "Unable to open output file " << filename << endl;
        exit(1);
    }

    fout.write("Have a nice day",strlen("Have a nice day."));

    int age = 35;
    double gpa = 3.5;

    fout.write(reinterpret_cast<char*>(&age),sizeof(int));
    fout.write(reinterpret_cast<char*>(&gpa),sizeof(gpa));

    cout << fout.tellp() << endl;
    fout.seekp(0,ios::end);
    cout << fout.tellp() << endl;

    fout.seekp(sizeof("Have a ")-1,ios::beg);
    cout << fout.tellp() << endl;
    fout.write("good",4);
    cout << fout.tellp() << endl;
    fout.close();
}
```

****** Output ******

```
28
28
7
11
```

Example 3 – binary file I/O: a practical example

This example demonstrates reading text file, storing each record in a struct and writing it out as a binary file. The "processing" requirement is to read the binary file and give all teachers a 5% raise and give Joe Bentley a 10% raise. The binary file will be updated to reflect the changes.

Input Text File

```
AGUILAR,  RICARDO L    ANIMAL CONTROL OFFICER                 70644.00
ALLISON,  JOHN L       ANIMAL TEACHERCONTROL OFFICER          64392.00
AYALA,  ARTHUR         ANIMAL CONTROL OFFICER                 70644.00
BATINICH,  JACLYN M    VETERINARY ASST                        66948.00
BENTLEY, JOE           TEACHER                                95000.00
CABALLERO,  JORGE      ANIMAL CONTROL OFFICER                 45924.00
CRAYTON,  MARSTINE L   SUPVSR OF ANIMAL CONTROL OFFICERS      73992.00
DEL RIO,  JOSE A       SUPVSR OF ANIMAL CONTROL OFFICERS      89124.00
…
```

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstdlib>
#include <cstring>
using namespace std;

const int NumRecords = 27;
const int SizeOfName = 23;
const int SizeOfJobtitle = 39;

struct SalaryData {
    char name[SizeOfName];
    char jobtitle[SizeOfJobtitle];
    float salary;
};

void printSalaryData(const SalaryData& record);
void rtrim(char* text);
void readAndPrintBinaryFile(const char* binaryfilename);
void processBinaryFile(const char* binaryfilename);
void readTextFileAndWriteToBinaryFile(const char* textfilename,
                                      const char* binaryfilename);

int main()
{
    const char* textfilename = "c:/temp/ex3data.txt";
    const char* binaryfilename = "c:/temp/ex3data.bin";
    readTextFileAndWriteToBinaryFile(textfilename, binaryfilename);
    processBinaryFile(binaryfilename);
    readAndPrintBinaryFile(binaryfilename);
}

```

```
void readTextFileAndWriteToBinaryFile(const char* textfilename,
                                      const char* binaryfilename)
{
    ifstream fin(textfilename);
    if (!fin)
    {
        cerr << "Unable to open input text file " << textfilename
             << endl;
        exit(1);
    }
    ofstream fout(binaryfilename, ios::binary);
    if (!fout)
    {
        cerr << "Unable to open input text file " << textfilename
             << endl;
        exit(2);
    }

    char buffer[80];
    SalaryData temp;

    for (int i = 0; i < NumRecords; ++i)
    {
        fin.getline(buffer, sizeof (buffer));
        strtok(buffer, "\r");
        strncpy(temp.name, buffer, SizeOfName);
        temp.name[SizeOfName - 1] = 0;
        rtrim(temp.name);
        strncpy(temp.jobtitle, buffer + 23, SizeOfJobtitle);
        temp.jobtitle[SizeOfJobtitle - 1] = 0;
        rtrim(temp.jobtitle);
        temp.salary = atof(buffer + 61);
        printSalaryData(temp);
        fout.write(reinterpret_cast<const char*>(&temp),
                   sizeof (SalaryData));
    }
    cout << "--------------------------------------------------\n";
}

void printSalaryData(const SalaryData& record)
{
    cout << fixed << setprecision(2);
    cout << left << setw(SizeOfName + 1) << record.name
         << setw(SizeOfJobtitle + 1) << record.jobtitle
         << right << setw(10) << record.salary << endl;
}

void rtrim(char* text)
{
    size_t size = strlen(text);
    for (int i = size - 1; i > 1; --i)
    {
        if (!isspace(text[i])) break;
        else text[i] = 0;
    }
```

```
}

void readAndPrintBinaryFile(const char* binaryfilename)
{
    ifstream fin(binaryfilename, ios::binary | ios::in);
    SalaryData temp;
    if (fin)
    {
        for (int i = 0; i < NumRecords; ++i)
        {
            fin.read(reinterpret_cast<char*>(&temp),
                     sizeof (temp));
             printSalaryData(temp);
         }
     }
     else
     {
         cerr << "Unable to open binary input file "
              << binaryfilename << endl;
         exit(3);
     }
 }

 // Teachers get a 5% raise
 // Joe Bentley gets a 10% raise
 void processBinaryFile(const char* binaryfilename)
 {
     // open the binary file for read and write
     fstream finfout(binaryfilename, ios::binary|ios::in|ios::out);
     SalaryData temp;
     if (finfout)
     {
         while (!finfout.eof())
         {
             finfout.read(reinterpret_cast<char*>(&temp),
                          sizeof (temp));
             if (strstr(temp.name, "BENTLEY"))
             {
                 temp.salary *= 1.1;
                 // Backup and rewrite the record
                 finfout.seekp(finfout.tellg() -
                     static_cast<streampos>(sizeof (SalaryData)));
                 finfout.write(reinterpret_cast<char*>(&temp),
                               sizeof (temp));
             }
             else if (!strcmp(temp.jobtitle, "TEACHER"))
             {
                 temp.salary *= 1.05;
                 // Backup and rewrite the record
                 finfout.seekp(finfout.tellg() -
                     static_cast<streampos> (sizeof (SalaryData)));
                 finfout.write(reinterpret_cast<char*>(&temp),
                               sizeof (temp));
             }
             else
```

```
            {
            }
        }
    }
    else
    {
        cerr << "Unable to binary file for processing "
             << binaryfilename << endl;
        exit(4);
    }
    if (!finfout.good()) finfout.clear();
    finfout.close();
}
```

****** Output ******

```
AGUILAR,  RICARDO L     ANIMAL CONTROL OFFICER                      70644.00
ALLISON,  JOHN L        ANIMAL TEACHERCONTROL OFFICER               64392.00
AYALA,  ARTHUR          ANIMAL CONTROL OFFICER                      70644.00
BATINICH,  JACLYN M     VETERINARY ASST                             66948.00
BENTLEY, JOE            TEACHER                                     95000.00
CABALLERO,  JORGE       ANIMAL CONTROL OFFICER                      45924.00
CRAYTON,  MARSTINE L    SUPVSR OF ANIMAL CONTROL OFFICERS           73992.00
DEL RIO,  JOSE A        SUPVSR OF ANIMAL CONTROL OFFICERS           89124.00
DIAKHATE,  MAMADOU      OPERATIONS MANAGER - ANIMAL CONTROL         85008.00
DRAKE,  TAURUS L        ANIMAL CONTROL INSPECTOR                    70644.00
EDGECOMBE,  CHERYL K    ANIMAL CONTROL INSPECTOR                    58644.00
FELTON,  DONIELLA M     TEACHER                                     47844.00
FRANCO,  ARTURO         ANIMAL CONTROL OFFICER                      45924.00
GARNER,  LINDSAY        VETERINARIAN                                88080.00
HAMILTON,  ARTHUR       ANIMAL SHELTER MANAGER                      68220.00
HOLCOMB,  ALLEN R       ANIMAL CONTROL INSPECTOR                    77520.00
HOWARD,  MARYANN J      ANIMAL CONTROL INSPECTOR                    64392.00
HUBBS,  CARLA A         SUPERVISING VETERINARY TECHNICIAN           62820.00
JACOB,  VIVISH          SUPVSR OF ANIMAL CARE AIDES                 84420.00
KELLER,  AUDREY A       VETERINARIAN                               124428.00
LOZANO,  RENE P         ANIMAL CONTROL OFFICER                      67464.00
MARTINIS,  JENNIFER     ANIMAL CONTROL OFFICER                      41832.00
RUSSELL,  SUSAN J       TEACHER                                    130008.00
SCHLUETER,  JENNIFER L  EXEC ADMINISTRATIVE ASST II                 59976.00
SILVA,  YVONNE          ANIMAL CONTROL OFFICER                      41832.00
WALTERS,  MICHELLE      TEACHER                                     70092.00
YAMAJI,  PETER S        VETERINARIAN                               128136.00
----------------------------------------------------------------
AGUILAR,  RICARDO L     ANIMAL CONTROL OFFICER                      70644.00
ALLISON,  JOHN L        ANIMAL TEACHERCONTROL OFFICER               64392.00
AYALA,  ARTHUR          ANIMAL CONTROL OFFICER                      70644.00
BATINICH,  JACLYN M     VETERINARY ASST                             66948.00
BENTLEY, JOE            TEACHER                                    104500.00
CABALLERO,  JORGE       ANIMAL CONTROL OFFICER                      45924.00
CRAYTON,  MARSTINE L    SUPVSR OF ANIMAL CONTROL OFFICERS           73992.00
DEL RIO,  JOSE A        SUPVSR OF ANIMAL CONTROL OFFICERS           89124.00
DIAKHATE,  MAMADOU      OPERATIONS MANAGER - ANIMAL CONTROL         85008.00
DRAKE,  TAURUS L        ANIMAL CONTROL INSPECTOR                    70644.00
EDGECOMBE,  CHERYL K    ANIMAL CONTROL INSPECTOR                    58644.00
FELTON,  DONIELLA M     TEACHER                                     50236.20
FRANCO,  ARTURO         ANIMAL CONTROL OFFICER                      45924.00
GARNER,  LINDSAY        VETERINARIAN                                88080.00
```

HAMILTON, ARTHUR	ANIMAL SHELTER MANAGER	68220.00
HOLCOMB, ALLEN R	ANIMAL CONTROL INSPECTOR	77520.00
HOWARD, MARYANN J	ANIMAL CONTROL INSPECTOR	64392.00
HUBBS, CARLA A	SUPERVISING VETERINARY TECHNICIAN	62820.00
JACOB, VIVISH	SUPVSR OF ANIMAL CARE AIDES	84420.00
KELLER, AUDREY A	VETERINARIAN	124428.00
LOZANO, RENE P	ANIMAL CONTROL OFFICER	67464.00
MARTINIS, JENNIFER	ANIMAL CONTROL OFFICER	41832.00
RUSSELL, SUSAN J	TEACHER	136508.41
SCHLUETER, JENNIFER L	EXEC ADMINISTRATIVE ASST II	59976.00
SILVA, YVONNE	ANIMAL CONTROL OFFICER	41832.00
WALTERS, MICHELLE	TEACHER	73596.60
YAMAJI, PETER S	VETERINARIAN	128136.00

Cast operators

Static Cast

A static_cast is used to return a variable or expression as a different type. Static casts are

- Often a cast that would occur automatically
- Usually a replacement for a C-style cast
- Sometimes not necessary, but used to provide visibility to a convesion

Example 1 – static_cast

```
#include <iostream>
using namespace std;

int main()
{
    unsigned ui = 0U;
    unsigned long ul = 123UL;
    int i = 0;

    bool b;
    float f = 3;

    // i = rand() % f;                         // Error
    i = rand() % static_cast<int>(f);

    b = i < ul;                                // Warning
    b = static_cast<unsigned long>(i) < ul;

    f = NULL;                                  // Warning
    f = static_cast<float>(NULL);

    enum color { red, white, blue };

    // Assign int value to enum variable
    // color hue = 1;                          // Error
    color hue = static_cast<color>(1);

    // Assign enum variable to int type
    i = hue;                                   // OK
    // Assign enum value to int type
    ui = white;                                // OK

    int* ptrI;
    // ptrI = &f;                              // Error
    // ptrI = static_cast<int*>(&f);           // Error
    ptrI = reinterpret_cast<int*>(&f);         // OK
}
```

Const Cast

A const_cast is used to add or remove *constness* to an expression. Note, removing constness from a “pointed to” value may result in undefined behavior.

Example 2 – const_cast

```
#include <string>
#include <iostream>
using namespace std;

void foo(string& s) { cout << s << endl; }
void goo(const string& s) { cout << s << endl; }
void delta(string& s) { s = "I am changed"; }

int main()
 {
   string s1 = "I am volatile";
   const string s2 = "I am const";

   foo(s1);
   //    foo(s2);   // Error: cannot convert
   foo(const_cast<string&>(s2));

   goo(s1);
   goo(s2);

   cout << "Before: s1 = " << s1 << endl;
   cout << "Before: s2 = " << s2 << endl;
   delta(s1);
   delta(const_cast<string&>(s2));
   cout << "After: s1 = " << s1 << endl;
   cout << "After: s2 = " << s2 << endl;
 }
```

****** Output ******

```
I am volatile
I am const
I am volatile
I am const
Before: s1 = I am volatile
Before: s2 = I am const
After: s1 = I am changed
After: s2 = I am changed
```

Reinterpret Cast

A reinterpret_cast is used to cast one type to another. It is most commonly used to treat one pointer type as another pointer type, or to treat a pointer type as an integer type and vice versa. Note, this case type may be unsafe and to use it effectively, the sizes of the casted value and the casted type should match.

Example 3 – reinterpret_cast

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   int i = 5;
   double d = 3.14;

   cout << d << ' ' << static_cast<int>(d)  << ' '
        << *(reinterpret_cast<int*>(&d)) << endl;
   cout << "&i=" << &i << ' ' << reinterpret_cast<long long>(&i)
        << endl;

   // write int and double out to a binary file
   ofstream fout("binaryfile");
   //fout.write(static_cast<char*>(&i), sizeof(i));       // ERROR
   fout.write(reinterpret_cast<char*>(&i), sizeof(i));
   fout.write(reinterpret_cast<char*>(&d), sizeof(d));
   fout.close();

   ifstream fin("binaryfile");
   fin.read(reinterpret_cast<char*>(&i), sizeof(i));
   fin.read(reinterpret_cast<char*>(&d), sizeof(d));
   fin.close();

   cout << i << ' ' << d << endl;
 }
```

****** Output (Code::Blocks vers 20.03) ******

```
3.14 3 1374389535
&i=0x61fe0c 6422028
5 3.14
```

Dynamic Cast

A dynamic_cast is used with inheritance to cast a base class pointer or reference to a derived class pointer or references. This is called downcasting. The dynamic_cast is used in conjunction with polymorphism to allow the user to execute a member function of a derived class using a pointer or reference of the base class. In order for this to succeed, the base class must be polymorphic (contains a virtual function).

Reference: http://www.bogotobogo.com/cplusplus/upcasting_downcasting.php

Example 4 – dynamic_cast

```
#include <iostream>
using namespace std;

class Animal
{
public:
    virtual ~Animal() {}  // Initiate polymorphism via virtual dtor
};

class Cat : public Animal
{
};

class Dog : public Animal
{
public:
    void bark() const
    {
        cout << "woof\n";
    }
};

int main()
{
    Cat fred;
    Dog fido;
    fido.bark();
    Animal* ptrAnimal;
    Dog* ptrDog;

    // Call the bark function using an Animal*
    ptrAnimal = &fido;
    // ptrAnimal -> bark();

    // Call the bark function using an Animal* cast to a Dog*
    dynamic_cast<Dog*>(ptrAnimal) -> bark();

    // Testing a dynamic cast
    ptrDog = dynamic_cast<Dog*>(&fido);
    cout << "&fido=" << &fido << " ptrDog = " << ptrDog << endl;

```

```
    ptrDog = dynamic_cast<Dog*>(&fred);
    cout << "&fred=" << &fred << " ptrDog = " << ptrDog << endl;
}
```

***** Output *****

```
woof
woof
&fido=0x61fdf0 ptrDog = 0x61fdf0
&fred=0x61fdf8 ptrDog = 0
```

The string class

The **string** class, part of the C++ “standard”, is an instantiation of the **basic_string** template for type char, or

```
typedef basic_string<char> string;
```

Access to the class requires the inclusion of the <string> header file.

Constructors

```
string();
string(const char* str);
string(const str& str);
string (const string& str, size_t pos, size_t len=npos);
string (const char* s, size_t n);
string (size_t n, char c);
template <class InputIterator>
      string(InputIterator first,InputIterator last);
```

Example 1 – string constructors

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
   // default constructor
   string s1;

    // c-string argument
    string s2a("second string");
    string s2b = "second string";
    string s2c{"second string"};

    // copy constructor
    string s3a(s2a);
    string s3b = s2a;

    // substring
```

```
    string s4(s2a,4,5);

    // c-string buffer
    string s5a("fifth string",5);
    string s5b("fifth string",25);

    // fill constructor
    string s6(10,'A');

    // range using iterators
    string s7(s2a.begin(),s2a.begin()+3);

    // initializer list
    string s8{'W','o','w','!'};

    // move constructor
    string temp("Bye bye");
    string s9(move(temp));

    cout << "s1=" << s1 << endl;
    cout << "s2a=" << s2a << endl;
    cout << "s2b=" << s2b << endl;
    cout << "s2c=" << s2c << endl;
    cout << "s3a=" << s3a << endl;
    cout << "s3b=" << s3b << endl;
    cout << "s4=" << s4 << endl;
    cout << "s5a=" << s5a << endl;
    cout << "s5b=" << s5b << endl;
    cout << "s6=" << s6 << endl;
    cout << "s7=" << s7 << endl;
    cout << "s8=" << s8 << endl;
    cout << "s9=" << s9 << endl;
    cout << "temp=" << temp << endl;
}
```

****** Output ******

```
s1=
s2a=second string
s2b=second string
s2c=second string
s3a=second string
s3b=second string
s4=nd st
s5a=fifth
s5b=fifth stringBye byes1=
s6=AAAAAAAAAA
s7=sec
s8=Wow!
s9=Bye bye
temp=
```

Iterator Functions

begin

Returns an iterator pointing to the first character of the string

```
iterator begin() noexcept[4];
const_iterator begin() const noexcept;
```

end

Returns an iterator pointing to the character beyond the end of the string

```
iterator end() noexcept;
const_iterator end() const noexcept;
```

rbegin

Returns a reverse iterator pointing to the last character of the string

```
reverse_iterator rbegin() noexcept;
const_reverse_iterator rbegin() const noexcept;
```

rend

Returns a reverse iterator pointing to the character in front of the first character of the string

```
reverse_iterator rend() noexcept;
const_reverse_iterator rend() const noexcept;
```

cbegin

Returns a const iterator pointing to the first character of the string

```
const_iterator begin() const noexcept;
```

cend

Returns a const iterator pointing to the character beyond the end of the string

```
const_iterator end() const noexcept;
```

crbegin

Returns a const reverse iterator pointing to the last character of the string

```
const_reverse_iterator rbegin() const noexcept;
```

crend

Returns a const reverse iterator pointing to the character in front of the first character of the string

[4] The noexcept specification means the function will not throw any exceptions.

```
const_reverse_iterator rend() const noexcept;
```

Example 2 – string iterator functions

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string s1("Have a nice day.");

    //  cout << s1.begin() << endl;  ERROR

     cout << *s1.begin() << endl;
     cout << *(s1.begin()+2) << endl;

     cout << '/' << *s1.end() << '/' << endl;     // error on MSVC++
     cout << *(s1.end()-4) << endl;

     cout << "*s1.rbegin()=" << *s1.rbegin() << '/' << endl;
     cout << "*(s1.rbegin()+1)=" << *(s1.rbegin()+1) << '/' << endl;
     cout << "*(s1.rbegin()-1)=" << *(s1.rbegin()-1) << '/' << endl;
     cout << endl;
     cout << "*s1.rend()=" << *s1.rend() << '/' << endl;
     cout << "*(s1.rend()+1)=" << *(s1.rend()+1) << '/' << endl;
     cout << "*(s1.rend()-1)=" << *(s1.rend()-1) << '/' << endl;
     cout << endl;

     *s1.begin() = 'Z';
     cout << s1 << endl;

     // *s1.cbegin() = 'Z';     ERROR

     for (string::const_iterator it = s1.begin(); it != s1.end(); ++it)
         cout << *it << '/';
     cout << endl;

     for (string::const_reverse_iterator it = s1.rbegin(); it != s1.rend(); ++it)
         cout << *it << '/';
 }
```

****** Code::Blocks on Windows ******

```
H
v
/ /
d
*s1.rbegin()=./
*(s1.rbegin()+1)=y/
```

```
*(s1.rbegin()-1)= /

*s1.rend()= /
*(s1.rend()+1)= /
*(s1.rend()-1)=H/

Zave a nice day.
Z/a/v/e/ /a/ /n/i/c/e/ /d/a/y/./
./y/a/d/ /e/c/i/n/ /a/ /e/v/a/Z/
```

****** Linux g++ 4.1.2

```
H
v
//
d
*s1.rbegin()=./
*(s1.rbegin()+1)=y/
*(s1.rbegin()-1)=/

*s1.rend()=/
*(s1.rend()+1)=/
*(s1.rend()-1)=H/

Zave a nice day.
Z/a/v/e/ /a/ /n/i/c/e/ /d/a/y/./
./y/a/d/ /e/c/i/n/ /a/ /e/v/a/Z/
```

Capacity Functions

size

Returns the length of a string

```
size_t size() const noexcept;
```

length

Returns the length of a string

```
size_t length() const noexcept;
```

capacity

Returns the size allocated for the string

```
size_t capacity() const noexcept;
```

max_size

Returns the maximum size for any string

```
size_t max_size() const noexcept;
```

reserve

Change the string's capacity. The function reserves ***at least the size*** requested.

```
void reserve(size_t n = 0);
```

clear

Erases a string. Size becomes 0

```
void clear() noexcept;
```

resize

Resizes a string to n characters

```
void resize (size_t n);
void resize (size_t n, char c);
```

empty

Returns whether the size is empty

```
bool empty() const noexcept;
```

shrink_to_fit

Changes the capacity to the size of the string

```
void shrink_to_fit();
```

Example 3 – capacity functions

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string s1 = "Have an exceptionally nice day";
    cout << s1 << endl;
    cout << "s1.size()=" << s1.size() << endl;
    cout << "s1.capacity()=" << s1.capacity() << endl;
    cout << "s1.max_size()=" << s1.max_size() << endl << endl;

    s1.reserve(50);
    cout << s1 << endl;
    cout << "s1.size()=" << s1.size() << endl;
    cout << "s1.capacity()=" << s1.capacity() << endl << endl;

    s1.reserve(5);
    cout << s1 << endl;
    cout << "s1.size()=" << s1.size() << endl;
```

```
    cout << "s1.capacity()=" << s1.capacity() << endl << endl;

    s1.reserve(75);
    cout << s1 << endl;
    cout << "s1.size()=" << s1.size() << endl;
    cout << "s1.capacity()=" << s1.capacity() << endl << endl;

    s1.resize(19);
    cout << s1 << endl;
    cout << "s1.size()=" << s1.size() << endl;
    cout << "s1.capacity()=" << s1.capacity() << endl << endl;

    s1.shrink_to_fit();
    cout << s1 << endl;
    cout << "s1.size()=" << s1.size() << endl;
    cout << "s1.capacity()=" << s1.capacity() << endl << endl;

    s1.clear();
    cout << s1 << endl;
    cout << "s1.size()=" << s1.size() << endl;
    cout << "s1.capacity()=" << s1.capacity() << endl << endl;

    cout << boolalpha << s1.empty() << endl;
}
```

****** Output ******

```
Have an exceptionally nice day
s1.size()=30
s1.capacity()=30
s1.max_size()=1073741820

Have an exceptionally nice day
s1.size()=30
s1.capacity()=60

Have an exceptionally nice day
s1.size()=30
s1.capacity()=30

Have an exceptionally nice day
s1.size()=30
s1.capacity()=75

Have an exceptional
s1.size()=19
s1.capacity()=75

Have an exceptional
s1.size()=19
s1.capacity()=19


s1.size()=0
s1.capacity()=19
```

```
true
```

Access Functions

at

Returns character at position

```
char& at (size_t pos);
const char& at (size_t pos) const;
```

back

Returns last character in string

```
char& back();
const char& back() const;
```

front

Returns first character in string

```
char& front();
const char& front() const;
```

Example 4 – access functions

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string s = "Have a nice day";
    cout <<s.front() << s.at(3) << s.back() << endl;
}
```

****** Output ******

```
Hey
```

Modifier Functions

assign

Assigns a new value to a string

```
string& assign(const string& str);
string& assign(const string& str,size_t subpos, size_t sublen = npos);
```

```
string& assign(const char* s);
string& assign(const char* s, size_t n);
string& assign(size_t n, char c);
```

append

Appends a value to a string

```
string& append(const string& str);
string& append(const string& str,size_t subpos, size_t sublen = npos);
string& append(const char* s);
string& append(const char* s, size_t n);
string& append(size_t n, char c);
```

erase

Erases part of a string

```
string& erase(size_t pos = 0, size_t len = npos);
iterator erase(const_iterator p);
iterator erase(const_iterator first, const_iterator last);
```

insert

Inserts characters into a string at a specified position

```
string& insert(size_t pos, const string& str);
string& insert(size_t pos, const string& str, size_t subpos,
              size_t sublen = npos);
string& insert(size_t pos,const char* s);
string& insert(size_t pos, const char* s, size_t n);
string& insert(size_t pos, size_t n, char c);
iterator insert(const_iterator p, size_t n, char c);
iterator insert(const_iterator p, char c);
```

push_back

Appends a char to the end of a string

```
void push_back (char c);
```

replace

Replaces part of a string with new contents

```
string& replace(size_t pos, size_t len, const string& str);
string& replace(const_iterator i1, const_iterator i2, const string& str);
string& replace(size_t pos, size_t len, const string& str,size_t subpos,
size_t sublen = npos);
string& replace(size_t pos, size_t len, const char* s);
string& replace(const_iterator i1, const_iterator i2, const char* s);
string& replace(size_t pos, size_t len, const char* s, size_t n);
string& replace(const_iterator i1,const_iterator i2, const char* s,
size_t n);
string& replace(size_t pos, size_t len, size_t n, char c);
```

```
string& replace(const_iterator i1, const_iterator i2, size_t n,char c);
```

swap

Swaps two strings

```
void swap (string& str);
```

pop_back

Erases the last character of a string

```
void pop_back();
```

Example 5 – modifier functions

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string s1 = "Have a nice day";
    string s2, s3, s4, s5, s6;

    s2.assign(s1);
    s3.assign(s1,7,4);
    s4.assign("Hey");
    s5.assign(s1.c_str(),3);
    s6.assign(5,'x');
    cout << s2 << endl << s3 << endl << s4 << endl << s5
         << endl << s6 << endl << endl;

    s2.append(s1);
    s3.append(s1,7,4);
    s4.append("Hey");
    s5.append(s1.c_str(),3);
    s6.append(5,'x');
    cout << s2 << endl << s3 << endl << s4 << endl << s5
         << endl << s6 << endl << endl;

    s2.erase();
    s3.erase(4);
    s4.erase(3,2);
    s5.erase(s5.begin()+1,s5.begin()+4);
    cout << s2 << endl << s3 << endl << s4 << endl << s5
         << endl << endl;

    s2 = s1;
    s3 = "very ";

    s2.insert(7,s3);
    cout << s2 << endl;
    s2.insert(s2.find("nice"),"VERY ");
    cout << s2 << endl << endl;

    s2.push_back('!');
    cout << s2 << endl << endl;

    s2.replace(s2.find("very VERY"),string("excellent").size(),
"excellent");
    cout << s2 << endl << endl;

    s2.replace(s2.find("excellent"),
string("excellent nice").size(),
"swell");
    cout << s2 << endl << endl;

```

```
    s1.swap(s2);
    cout << s1 << endl << s2 << endl << endl;

    s1.pop_back();
    cout << s1 << endl;
}
```

```
******  Output  ******

Have a nice day
nice
Hey
Hav
xxxxx

Have a nice dayHave a nice day
nicenice
HeyHey
HavHav
xxxxxxxxxx


nice
Heyy
Hav

Have a very nice day
Have a very nice day

Have a very nice day!

Have a excellent nice day!

Have a swell day!

Have a swell day!
Have a nice day

Have a swell day
```

Search Functions

find

Locates text in a string. Returns npos if not found

```
size_t find(const string& str, size_t pos = 0) const;
size_t find(const char* s, size_t pos = 0) const;
size_t find(const char* s, size_t pos size_type n) const;
size_t find(char c, size_t pos = 0) const;
```

find_first_of

Locates first occurrence of text in a string

```
size_t find_first_of (const string& str, size_t pos = 0) const noexcept;
size_t find_first_of (const char* s, size_t pos = 0) const;
size_t find_first_of (const char* s, size_t pos, size_t n) const;
size_t find_first_of (char c, size_t pos = 0) const noexcept;
```

find_last_of

Locates last occurrence of text in a string

```
size_t find_last_of (const string& str, size_t pos = 0) const noexcept;
size_t find_last_of (const char* s, size_t pos = 0) const;
size_t find_last_of (const char* s, size_t pos, size_t n) const;
size_t find_last_of (char c, size_t pos = 0) const noexcept;
```

find_first_not_of

Locates first occurrence of any characters not in a string

```
size_t find_first_not_of (const string& str, size_t pos = 0) const noexcept;
size_t find_first_not_of (const char* s, size_t pos = 0) const;
size_t find_first_not_of (const char* s, size_t pos, size_t n) const;
size_t find_first_not_of (char c, size_t pos = 0) const noexcept;
```

find_last_not_of

Locates last occurrence of any characters not in a string

```
size_t find_last_not_of (const string& str, size_t pos = 0) const noexcept;
size_t find_last_not_of (const char* s, size_t pos = 0) const;
size_t find_last_not_of (const char* s, size_t pos, size_t n) const;
size_t find_last_not_of (char c, size_t pos = 0) const noexcept;
```

rfind

Locates text in a string.

```
size_t rfind(const string& str, size_t pos = 0) const;
size_t rfind(const char* s, size_t pos = 0) const;
size_t rfind(const char* s, size_t pos size_type n) const;
size_t rfind(char c, size_t pos = 0) const;
```

Example 6 – search functions

```
#include <iostream>
#include <string>
```

```
using namespace std;

int main()
{
    string hand = "Have a nice day";
    string nice = "nice";
    string Nice = "Nice";

     cout << hand.find(nice) << endl;
     cout << hand.find("nice") << endl;
     cout << hand.find(Nice) << endl;
     cout << nice << " is "
          << (hand.find(nice) == string::npos ? "not " : "")
          << "present" << endl;
     cout << Nice << " is "
          << (hand.find(Nice) == string::npos ? "not " : "")
          << "present" << endl << endl;

     // Find the first 'a'
     cout << hand.find('a') << endl;

     // Find the second 'a'
     cout << hand.find('a',hand.find('a')+1) << endl;

     // Find the third 'a'
     cout << hand.find('a',hand.find('a',hand.find('a')+1)+1)
          << endl;

     // Find the last 'a'
     cout << hand.rfind('a') << endl << endl;

     cout << hand.find_first_of(nice) << endl;
     cout << hand.find_first_of("abcde") << endl;
     cout << hand.find_first_of('v') << endl;
     cout << hand.find_first_of('v',3) << endl << endl;

     cout << hand.find_last_of("abcde") << endl;

     cout << hand.find_first_not_of("abcdefghijklmnopqrstuvwxyz")
          << endl;
     cout << hand.find_last_not_of("abcdefghijklmnopqrstuvwxyz")
         << endl;
 }
```

****** Output ******

```
7
7
4294967295
nice is present
Nice is not present

1
5
13
```

```
13

3
1
2
4294967295

13
0
11
```

Operation Functions

c_str

Returns the null-terminated char array contents of the string.   The c_str and data functions return the same value.

```
const char* c_str() const noexcept;
```

compare

Compares two strings or a string and a cstring

```
int compare (const string& str) const noexcept;
int compare (size_t pos, size_t len, const string& str) const;
int compare (size_t pos, size_t len, const string& str,
             size_t subpos, size_t sublen = npos) const;
int compare (const char* s) const;
int compare (size_t pos, size_t len, const char* s) const;
int compare (size_t pos, size_t len, const char* s, size_t n) const;
```

copy

Copies part of a string into a char array. A null is not added to the char array.

```
size_t copy (char* s, size_t len, size_t pos = 0) const;
```

substr

Returns part of a string

```
string substr (size_t pos = 0, size_t len = npos) const;
```

Example 7 – operation functions

```
#include <iostream>
#include <string>
using namespace std;
```

```

int main()
{
    string Hand = "Have a nice day";
    string hand = "have a nice day";
    string Have = "Have";
    string nice = "nice";

    cout << Hand.compare(Hand) << endl;
    cout << Hand.compare(hand) << endl;
    cout << Hand.compare(Have) << endl;
    cout << string("ABC").compare("ABD") << endl;
    cout << Hand.compare(7,4,nice) << endl;
    cout << Hand.compare(1,string::npos,hand,1,string::npos)<<endl;
    cout << Have.compare(Have.c_str()) << endl << endl;

    char array[16];
    Hand.copy(array,4);
    cout << array << endl;

    cout << Hand.substr(5) << endl;
    cout << Hand.substr(5,6) << endl;
}
```

****** Code::Blocks on Windows ******

```
0
-1
11
-1
0
0
0

Have╟eÆ╓Ç@
a nice day
a nice
```

****** Linux g++ 4.1.2 ******

```
0
-1
11
-1
0
0
0

Have
a nice day
a nice
```

****** Linux g++ 6.4.0 ******

```
0
-32
11
-1
0
0
0

Have
a nice day
a nice
```

Non-member Functions

getline

Extracts from a input stream into a string

```
istream& getline (istream&  is, string& str, char delim);
istream& getline (istream&  is, string& str);
```

swap

Swaps two string

```
void swap (string& x, string& y);
```

Example 8 – Non-member string functions

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
    string filename = __FILE__;                  // What’s this?
    cout << "#1 " << filename << endl << endl;
     ifstream fin(filename);
     if (!fin)
     {
         cerr << "Unable to open " << filename << endl;
         exit(1);
     }
     string buffer1, buffer2;
     getline(fin,buffer1);
     cout << "#2 buffer1 = " << buffer1 << endl;
     getline(fin,buffer2);
     cout << "#3 buffer2 = " << buffer2 << endl << endl;

     swap(buffer1, buffer2);
```

```
    cout << "#4 buffer1 = " << buffer1 << endl;
    cout << "#5 buffer2 = " << buffer2 << endl << endl;

    getline(fin,buffer1,'<');
    cout << "#6 buffer1 = " << buffer1 << '/' << endl;
    getline(fin,buffer2);
    cout << "#7 buffer2 = " << buffer2 << endl << endl;

    getline(fin,buffer1,'_');
    cout << "#8 "  << buffer1 << endl << endl;

    cout << "Life is good? " << boolalpha << fin.good() << endl;
}
```

****** Output ******

```
#1 Z:\deanza\cis29\examples\string_class\ex5-8.cpp

#2 buffer1 = #include <iostream>
#3 buffer2 = #include <fstream>

#4 buffer1 = #include <fstream>
#5 buffer2 = #include <iostream>

#6 buffer1 = #include /
#7 buffer2 = string>

#8 using namespace std;

int main()
{
    string filename =

Life is good? true
```

Member Operators

operator=

Assignment operator: assigns a new value to a string

```
string& operator= (const string& str);
string& operator= (const char* s);
string& operator= (char c);
```

operator[]

Index operator: returns the character at the specified location

```
char& operator[] (size_t pos);
const char& operator[] (size_t pos) const;
```

operator+=

Plus-equal operator: concatenates text to an existing string

```
string& operator+= (const string& str);
string& operator+= (const char* s);
string& operator+= (char c);
```

Non-member Operators

operator+

Operator +: returns, by value, the result of two concatenated strings

```
string operator+ (const string& lhs, const string& rhs);
string operator+ (const string& lhs, const char*   rhs);
string operator+ (const char*   lhs, const string& rhs);
string operator+ (const string& lhs, char          rhs);
string operator+ (char          lhs, const string& rhs);
```

operator<<

Insertion operator: inserts a string into an output stream

```
ostream& operator<< (ostream& os, const string& str);
```

operator>>

Extraction operator: extracts a string from an input stream

```
istream& operator>> (istream& os, const string& str);
```

Example 9 – Member and non-member string operators

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string s = "Have a nice day";
    string s2, s3, s4;

     s2 = s;
     s3 = "Hey";
     s4 = '!';

     cout << s3[1] << endl;
     s3[1] = 'a';
     cout << s3[1] << endl << endl;

```

```
    s2 += s4;
    cout << s2 << endl;
    s2 += '*';
    cout << s2 << endl << endl;

    cout << s3 + s4 << endl;
    cout << s3 + " you" << endl;
    cout << "you " + s3 << endl;
    cout << s3 + '?' << endl;
    cout << '?' + s3 << endl;
}
```

****** Output ******

```
e
a

Have a nice day!
Have a nice day!*

Hay!
Hay you
you Hay
Hay?
?Hay
```

Member Constant

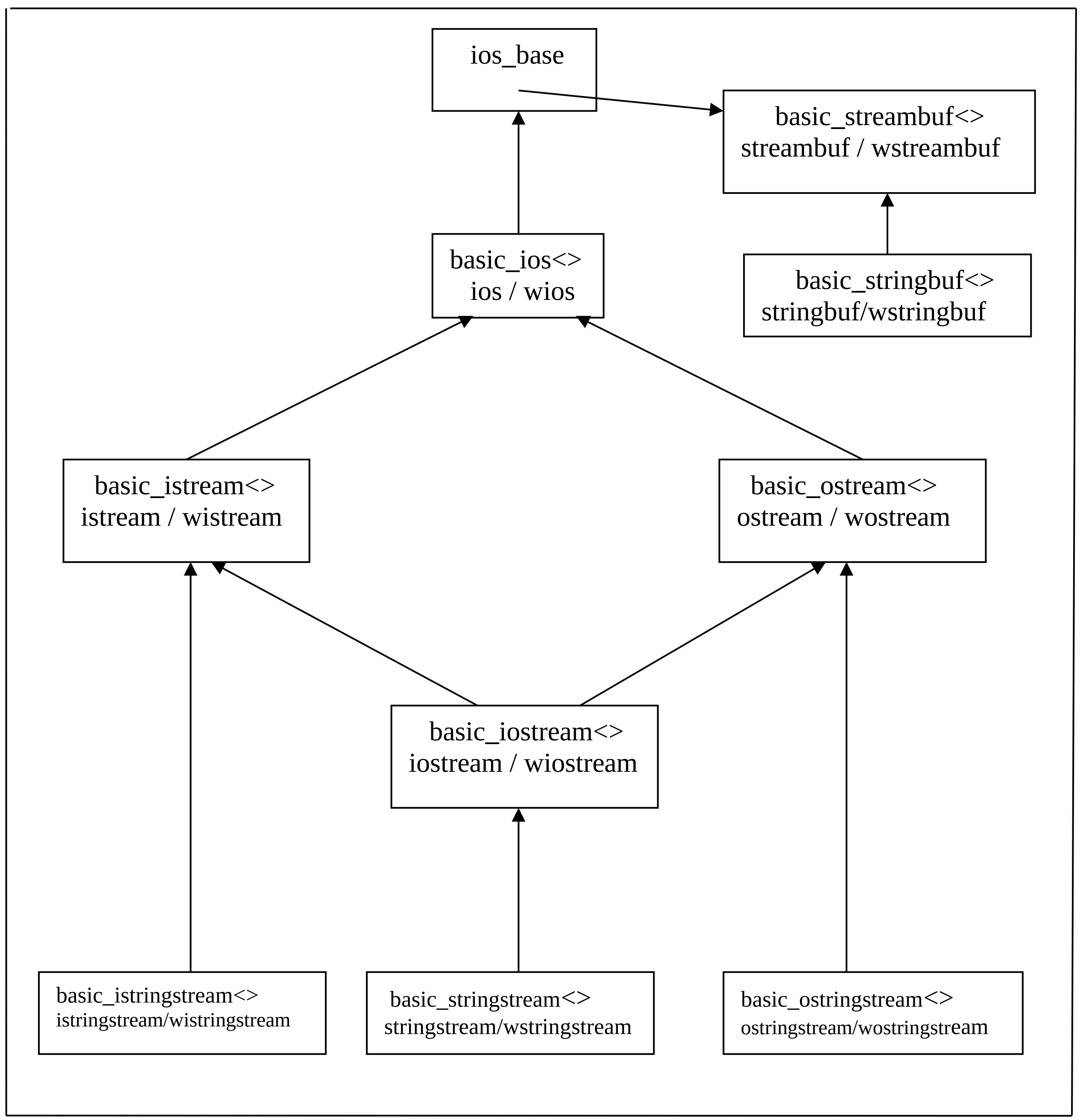
npos

npos is a static member constant, equal to the maximum value for type, size_t. It is used to indicate the location beyond the length of a string, or with use of a find function, the return value, not found.

```
static const size_t npos = -1;
```

The stringstream classes

The stringstream classes, istringstream, ostringstream, and stringstream, are instantiations of the basic_string<> and the basic_istream<> and basic_ostream<> templates. These classes are the results of inheritance of class templates.



The istringstream class

The istringstream class is used to read from a string buffer. A useful technique is to read a string into an istringstream buffer, then use that buffer to parse the input of the entire string.

Example 1 – Using istringstream for parsing input

```
#include <sstream>
#include <iostream>
#include <string>
using namespace std;

int main()
{
   string string1("Have a nice day.");
   string buffer;

   istringstream sin(string1);

     // What is in the istringstream buffer?
     cout << "sin.str()=" << sin.str() << endl;

     // read from the istringstream buffer
     while (sin >> buffer)
     {
         cout << buffer << endl;
     }

     // Let's get a new istringstream buffer
     sin.str("Let's get a new istringstream buffer");
     while (sin >> buffer)
     {
         cout << buffer << endl;
     }

   // Why didn't this work?

   // after reading from the istringstream, what is the "state" of the stream?
   cout << boolalpha << "sin.eof()=" << sin.eof() << endl;
   cout << "sin.rdstate()=" << sin.rdstate()<< endl;

   // clear the eofbit
   sin.clear();
   cout << boolalpha << "sin.eof()=" << sin.eof() << endl;
   cout << "sin.rdstate()=" << sin.rdstate()<< endl;

   cout << "sin.str()="<<sin.str()<<endl;
   cout << "sin.tellg()=" << sin.tellg() << endl;

   sin >> buffer;

   cout << "buffer=" << buffer << " sin.gcount()=" << sin.gcount() << endl;

```

```
    // Why is sin.gcount()= 0?

    char cbuffer[32];
    sin.seekg(0);

    sin.read(cbuffer,4);
    cout << "sin.gcount()=" << sin.gcount() << endl;

    getline(sin,buffer);
    cout << "buffer=" << buffer << " sin.gcount()=" << sin.gcount() << endl;

    sin.seekg(0);
    sin.get(cbuffer,sizeof(cbuffer));
    cout << "cbuffer=" << buffer << " sin.gcount()=" << sin.gcount() << endl;

    sin.seekg(0);
    sin.getline(cbuffer,sizeof(cbuffer));
    cout << "cbuffer=" << buffer << " sin.gcount()=" << sin.gcount() << endl;
}
```

Example 2 - A practical example

```
#include <fstream>
#include <sstream>
#include <iostream>
#include <string>
using namespace std;

int main()
{
    ifstream fin("c:/temp/short_gettysburg_address.txt");
    string buffer, word;
    istringstream sin;

    while (!fin.eof())
    {
        getline(fin,buffer);
        sin.str(buffer);
        while (sin >> word)
        {
            cout << word << endl;
        }
        sin.clear();
    }
}
```

```
Four score and seven years ago our fathers brought forth on this continent, a new
nation, conceived in Liberty, and dedicated to the proposition that all men are
created equal.
```

****** **Output** ******

```
Four
score
and
seven
years
ago
our
fathers
brought
forth
on
this
continent,
a
new
nation,
conceived
in
…
```

The ostringstream class

The ostringstream class is used to write into a string buffer. This is useful for composing a desired output format.

Example 3 – Using ostringstream to compose output

```
// ostringstream example

#include <iostream>
#include <iomanip>
#include <sstream>
#include <string>
using namespace std;

void print(double number);

 int main()
 {
     double array[] =
 {1,1.2,1.23,1.234,123.45,1234.56,12345.67,1234.5678};
     auto numberOfElements = sizeof(array) / sizeof(double);

     for (auto element : array)
         print(element);
     }

 void print(double number)
 {
     ostringstream sout;
     cout << left << setw(12) << setprecision(8) << number;
     sout << setprecision(2) << fixed << '$';
     if (number > 1000)
     {
         int thousands = static_cast<int>(number) / 1000;
         sout << thousands << ',';
         sout << number - thousands*1000;
     }
     else
     {
         sout << number;
     }
     cout << right << setw(16) << sout.str() << endl;
 }
```

****** Output ******

```
1                      $1.00
1.2                    $1.20
1.23                   $1.23
1.234                  $1.23
123.45               $123.45
1234.56            $1,234.56
```

12345.67	$12,345.67
1234.5678	$1,234.57

The stringstream class

Example 4 – Using the stringstream class

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <cctype>
using namespace std;

void rewriteScore(const string&);

int main()
{
    ifstream fin("c:/temp/nfl_scores.txt");
    string buffer;

    while (getline(fin,buffer) && buffer.size())
        rewriteScore(buffer);
}

void rewriteScore(const string& buffer)
{
    string temp, dummy, winner, loser;
    int winnerScore, loserScore;
    stringstream ss;

    ss.str(buffer);

    ss >> dummy >> winner >> temp;
    winner += ' ';
    winner += temp;
    ss >> temp;
    // look for a comma at the end of temp
    if (isalpha(temp[0]) or temp == "49ers")
    {
        winner += ' ';
        winner += temp;
        ss >> temp;
    }

    // remove the comma from the winner's score string
    temp.resize(temp.size()-1);
    winnerScore = stoi(temp);
    ss >> loser >> temp;
    loser += ' ';
    loser += temp;
    ss >> temp;

    if (isalpha(temp[0])or temp == "49ers")
    {
        loser += ' ';
        loser += temp;
        ss >> temp;
```

```
    }
    loserScore = stoi(temp);
    ss.clear();
    ss << winner << " over " << loser << ' ' << winnerScore <<
" to " << loserScore;

    cout << ss.str() << endl;
}
```

Input File

```
8-Sep Denver Broncos 21, Carolina Panthers 20
11-Sep Green Bay Packers 27, Jacksonville Jaguars 23
11-Sep Baltimore Ravens 13, Buffalo Bills 7
11-Sep Cincinnati Bengals 23, New York Jets 22
11-Sep Houston Texans 23, Chicago Bears 14
11-Sep Minnesota Vikings 25, Tennessee Titans 16
11-Sep Philadelphia Eagles 29, Cleveland Browns 10
11-Sep Oakland Raiders 35, New Orleans Saints 34
11-Sep Kansas City Chiefs 33, San Diego Chargers 27
11-Sep Tampa Bay Buccaneers 31, Atlanta Falcons 24
11-Sep Seattle Seahawks 12, Miami Dolphins 10
11-Sep New York Giants 20, Dallas Cowboys 19
…
```

****** Output ******

```
Denver Broncos over Carolina Panthers 21 to 20
Green Bay Packers over Jacksonville Jaguars 27 to 23
Baltimore Ravens over Buffalo Bills 13 to 7
Cincinnati Bengals over New York Jets 23 to 22
Houston Texans over Chicago Bears 23 to 14
Minnesota Vikings over Tennessee Titans 25 to 16
Philadelphia Eagles over Cleveland Browns 29 to 10
Oakland Raiders over New Orleans Saints 35 to 34
Kansas City Chiefs over San Diego Chargers 33 to 27
Tampa Bay Buccaneers over Atlanta Falcons 31 to 24
Seattle Seahawks over Miami Dolphins 12 to 10
New York Giants over Dallas Cowboys 20 to 19
…
```

I/O Manipulators

std manipulators

Manipulators are functions or function-like operators that change the state of the I/O stream.

Manipulator	**I/O**	**Purpose**
Independent Flags		**Turns Setting On**
boolalpha	I/O	sets boolalpha flag
showbase	O	sets showbase flag
showpoint	O	sets showpoint flag
showpos	O	sets showpos flag
skipws	I	sets skipws flag
unitbuf	O	sets unitbuf flag
uppercase	O	sets uppercase flag
Independent Flags		**Turns Setting Off**
noboolalpha	I/O	clears boolalpha flag
noshowbase	O	clears showbase flag
noshowpoint	O	clears showpoint flag
noshowpos	O	clears showpos flag
noskipws	I	clears skipws flag
nounitbuf	O	clears unitbuf flag
nouppercase	O	clears uppercase flag
Numeric Base Flags		
dec	I/O	sets dec flag for i/o of integers, clears oct,hex
hex	I/O	sets hex flag for i/o of integers, clears dec,oct
oct	I/O	sets oct flag for i/o of integers, clears dec,hex
hexfloat (C++11)	I/O	sets hexadecimal floating point formatting
defaultfloat (C++11)	I/O	clears the float field formats
Floating Point Flags		
fixed	O	sets fixed flag
scientific	O	sets scientific flag
Adjustment Flags		
internal	O	sets internal flag
left	O	sets left flag
right	O	sets right flag
Input Only		
ws	I	extracts whitespace
Output Only		
endl	O	inserts a newline **and flushes output stream**
ends	O	inserts a null
flush	O	flushes stream
Parameterized Manipulators(these require the ***iomanip*** header file)		
resetiosflags(ios_base::fmtflags mask)	I/O	clears format flags specified by mask
setbase(int base)	I/O	sets integer base (8, 10, or 16)
setfill(char_type ch)	O	sets the fill character to ch
setiosflags(ios::base::fmtflags mask)	I/O	sets format flags to mask value
setprecision(int p)	O	sets precision of floating point numbers
setw(int w)	O	sets output field width to w
get_money (C++11)	I	parses a monetary value
put_money (C++11)	O	formats and outputs a monetary value
get_time (C++11)	I	parses a date/time value
put_time (C++11)	O	formats and outputs a date/time value
quoted (C++14)	I/O	Allows input/output of quoted text

Example 1 – Input/Output manipulators

The following examples illustrates the use of standard input/output manipulators.

```
#include <iostream>
#include <iomanip>
using namespace std;

void show_fmtflags(ios_base& stream);

int main()
{
    // save the initial cout flags settings
    ios_base::fmtflags cout_fmtflags = cout.flags();

    // Display the cout flags
    show_fmtflags(cin);
    show_fmtflags(cout);
    show_fmtflags(cerr);
    show_fmtflags(clog);
    cout << endl;

    int x = 123;

    // hex, oct, & dec manipulators
    cout << "dec: x = " << dec << x << endl;
    cout << "hex: x = " << hex << x << endl;
    cout << "oct: x = " << oct << x << endl;
    show_fmtflags(cout);
    cout << endl;

    // Turn on showpos, uppercase, showpoint, left, hex
    cout << setiosflags(ios::showpos|ios::uppercase|ios::showpoint|
                        ios::showbase|ios::left|ios::hex);
    show_fmtflags(cout);
    cout << "x = " << x << endl << endl;

    // Clear the oct flag
    cout << resetiosflags(ios::oct) << "x = " << x << endl;
    show_fmtflags(cout);
    cout << endl;

    // Demonstrate the setfill and setw manipulators
    cout << setfill('$') << setw(10) << "x = " << x << endl;
    cout << "x = " << x << endl << endl;

    // Reset cout's flags back to the original settings
    cout.flags(cout_fmtflags);

    // Turn on hex
    cout << hex << "x = " << x << endl;
    show_fmtflags(cout);
    cout << endl;

```

```
    // Turn on octal
    cout << oct << "x = " << x << endl;
    show_fmtflags(cout);
    cout << endl;

    // Demonstrate setprecision
    cout << setprecision(3) << 1.2 << ' ' << 3.14 << ' ' << 35
         << ' ' << 3.14159 << endl;

    // Demonstrate setprecision with showpoint
    cout << showpoint << 1.2 << ' ' << 3.14 << ' ' << 35 << ' '
         << 3.14159 << endl;

    // Demonstrate showpos
    cout << showpos << 1.2 << ' ' << 3.14 << ' ' << 35 << ' '
         << 3.14159 << endl;
    show_fmtflags(cout);
    cout << endl;

    // Back to decimal
    cout << dec <<  1.2 << ' ' << 3.14 << ' ' << 35 << ' '
         << 3.14159 << endl;
    show_fmtflags(cout);
    cout << endl;

    // What is truth?
    cout << true << ' ' << boolalpha << true << endl;
    show_fmtflags(cout);
}


void show_fmtflags(ios_base& stream)
{
    cout << (&stream == &cout ? "cout " : "");
    cout << (&stream == &cerr ? "cerr " : "");
    cout << (&stream == &clog ? "clog " : "");
    cout << (&stream == &cin  ? "cin  " : "");

    cout << "fmtflags set: ";

    cout << (stream.flags() & ios::boolalpha  ? "boolalpha " : "");
    cout << (stream.flags() & ios::dec        ? "dec " : "");
    cout << (stream.flags() & ios::fixed      ? "fixed " : "");
    cout << (stream.flags() & ios::hex        ? "hex " : "");
    cout << (stream.flags() & ios::internal   ? "internal " : "");
    cout << (stream.flags() & ios::left       ? "left " : "");
    cout << (stream.flags() & ios::oct        ? "oct " : "");
    cout << (stream.flags() & ios::right      ? "right " : "");
    cout << (stream.flags() & ios::scientific ? "scientific " :"");
    cout << (stream.flags() & ios::showbase   ? "showbase " : "");
    cout << (stream.flags() & ios::showpoint  ? "showpoint " : "");
    cout << (stream.flags() & ios::showpos    ? "showpos " : "");
    cout << (stream.flags() & ios::skipws     ? "skipws " : "");
    cout << (stream.flags() & ios::unitbuf    ? "unitbuf " : "");
    cout << (stream.flags() & ios::uppercase  ? "uppercase " : "");
```

```
    cout << endl;
}
```

****** Output ******

```
cin  fmtflags set: dec skipws
cout fmtflags set: dec skipws
cerr fmtflags set: dec skipws unitbuf
clog fmtflags set: dec skipws

dec: x = 123
hex: x = 7b
oct: x = 173
cout fmtflags set: oct skipws

cout fmtflags set: hex left oct showbase showpoint showpos skipws uppercase
x = +123

x = 0X7B
cout fmtflags set: hex left showbase showpoint showpos skipws uppercase

x = $$$$$$0X7B
x = 0X7B

x = 7b
cout fmtflags set: hex skipws

x = 173
cout fmtflags set: oct skipws

1.2 3.14 43 3.14
1.20 3.14 43 3.14
+1.20 +3.14 43 +3.14
cout fmtflags set: oct showpoint showpos skipws

+1.20 +3.14 +35 +3.14
cout fmtflags set: dec showpoint showpos skipws

+1 true
cout fmtflags set: boolalpha dec showpoint showpos skipws
```

Example 2 - floatfield manipulators

```
#include <iostream>
#include <sstream>
using namespace std;

int main()
{
    // save the cout format flags
    ios_base::fmtflags originalFlags = cout.flags();

    double f = 1234.5678;
    cout << "Default output: " << f << endl;
    cout << "fixed: " << fixed << f << endl;
    cout << "scientific: " << scientific << f << endl;
    cout << "hexfloat: " << hexfloat << f << endl;
    cout << "default: " << defaultfloat << f << endl;

    // read hexfloat format into a double
    istringstream("0x1P-1022") >> hexfloat >> f;

    // display the double in default format
    cout << "Parsing 0x1P-1022 as hex gives " << f << '\n';

    f = 3.141592654;
    cout << f << " as hexfloat: " << hexfloat << f << endl;

    // save hexfloat value into a string
    ostringstream sout;
    sout << hexfloat << f << endl;

  // save the hexfloat value into an input string buffer
    istringstream sin;
    sin.str(sout.str());

    // read the input string buffer into a double
    sin >> hexfloat >> f;

    // display f
    cout << f << endl;

    // display f in original format
    cout.flags(originalFlags);
    cout << f << endl;
  }
```

****** Output ******

(MS Visual Studio 2017)

```
Default output: 1234.57
fixed: 1234.567800
scientific: 1.234568e+03
```

```
hexfloat: 0x1.34a457p+10
default: 1234.57
Parsing 0x1P-1022 as hex gives 2.22507e-308
3.14159 as hexfloat: 0x1.921fb5p+1
0x1.921fb5p+1
3.14159
```

(MacBook Xcode 8.33)

```
Default output: 1234.57
fixed: 1234.567800
scientific: 1.234568e+03
hexfloat: 0x1.34a456d5cfaadp+10
default: 1234.57
Parsing 0x1P-1022 as hex gives 2.22507e-308
3.14159 as hexfloat: 0x1.921fb5452455p+1
0x1.921fb5452455p+1
3.14159
```

(gnu compiler output)

```
Default output: 1234.57
fixed: 1234.567800
scientific: 1.234568e+03
hexfloat: 0x1.34a456d5cfaadp+10
default: 1234.57
Parsing 0x1P-1022 as hex gives 0← This looks like a bug
3.14159 as hexfloat: 0x1.921fb5452455p+1
0x0p+0← This looks like a bug
0    ← This looks like a bug
```

Example 3 - get_money manipulator

```
#include <iostream>
#include <sstream>
#include <string>
#include <iomanip>
#include <locale>
using namespace std;

int main()
{
   istringstream in("$1,234.56 2.22 USD  3.33");
   locale mylocale("");
   in.imbue(mylocale);

   long double v1, v2;
   string v3;

```

```
   in >> std::get_money(v1) >> std::get_money(v2) >> std::get_money(v3, true);

   cout << quoted(in.str()) << " parsed as: " << v1 << ' ' << v2 << ' ' << v3 << endl;

   in.str("$125 .99");
   in.seekg(0);
   in >> std::get_money(v1) >> std::get_money(v2);
   cout << quoted(in.str()) << " parsed as: " << v1 << ' ' << v2 << endl;
 }
```

(MS Visual Studio 2017, MS Visual Studio 2019 and gnu compiler on Linux and MacBook)
(Does not run on gnu compilers on a PC – 1/28/20)

```
"$1,234.56 2.22 USD  3.33" parsed as: 123456 222 333
"$125 .99" parsed as: 12500 99
```

Note: the quoted() function required compilation with ***std=c++14***.

Example 4 - put_money manipulator

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

int main()
{
   long double value = 123.45;
   std::cout.imbue(std::locale(""));

   cout << put_money(value) << endl;
   cout << put_money(value, true) << endl;  // use international representation

   cout << showbase;
   cout << put_money(value) << endl;
   cout << put_money(value, true) << endl;  // use international representation

   string stringValue = "2345.67";

   cout << noshowbase;
   cout << put_money(stringValue) << endl;
   cout << put_money(stringValue, true) << endl;  // use international representation
   cout << showbase;
   cout << put_money(stringValue) << endl;
```

```
  cout << put_money(stringValue, true) << endl;  // use
international representation
}
```

(MS Visual Studio 2017 / MS Visual Studio 2019)

```
1.23
1.23
$1.23
USD1.23
23.45
23.45
$23.45
USD23.45
```

(g++ 7.2.0 on Linux)

```
1.23
 1.23
$1.23
USD  1.23
23.45
 23.45
$23.45
USD  23.45
```

(g++ on MacBook)

```
1.23
1.23
$1.23
USD 1.23
23.45
23.45
$23.45
USD 23.45
```

This does not work on Windows gnu compilers – 1/28/20

Example 5 - get_time and put_time manipulators

```
#include <iostream>     // cin, cout
#include <iomanip>      // get_time
#include <ctime>        // struct tm
#include <string>
#include <sstream>
#include <locale>
using namespace std;

```

```
int main()
 {
   struct tm when;

   const string monthName[] = {
"January","February","March","April","May","June",
         "July","August","September","October","November","December"
};

   cout << "Please, enter the time (hh:mn): ";
   cin >> get_time(&when, "%R");   // extract time (24H format)

   if (cin.fail()) cout << "Error reading time\n";
   else {
         cout << "The time entered is: ";
         cout << when.tm_hour << " hours and " << when.tm_min << "
minutes\n";
   }

   cout << "Please, enter the date (mm/dd/yy): ";
   cin >> get_time(&when, "%D");   // extract date

   if (cin.fail()) cout << "Error reading date\n";
   else {
         cout << "The date entered is: ";
         cout << monthName[when.tm_mon] << " " << when.tm_mday << ",
";
         cout << when.tm_year + 1900 << endl;
   }

   tm t = {};
   istringstream ss("2011-February-18 23:12:34");

   // imbue cout with the “local” locale
 cout.imbue(locale(""));

   // get the datetime from an istringstream
 ss >> get_time(&t, "%Y-%b-%d %H:%M:%S");
   if (ss.fail()) {
         cout << "Parse failed" << endl;
   }
   else {
         cout << put_time(&t, "%c") << endl;
         cout << put_time(&t, "%D %r") << endl;
   }
 }
```

(MS Visual Studio 2017

```
Please, enter the time (hh:mn): 16:57            ← User input
The time entered is: 16 hours and 57 minutes
Please, enter the date (mm/dd/yy): 09/08/17      ← User input
The date entered is: September 8, 2017
2/18/2011 11:12:34 PM
```

```
02/18/11 11:12:34 PM
```

(g++ on MacBook)

```
Please, enter the time (hh:mn): 14:22            ← User input
The time entered is: 14 hours and 22 minutes
Please, enter the date (mm/dd/yy): 11/15/17      ← User input
The date entered is: November 15, 2017
Sun Feb 18 23:12:34 2011
02/18/11 11:12:34 PM
```

(Cygwin compiler on Windows – g++ 7.4.0): not working 1/28/20

```
Please, enter the time (hh:mn): 16:57            ← User input
The time entered is: 16 hours and 57 minutes
Please, enter the date (mm/dd/yy): 09/08/17      ← User input
The date entered is: September 8, 1917
```

Example 6 – quoted manipulator

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <string>
using namespace std;

int main()
{
   stringstream ss1;
   stringstream ss2;
   string in = "String with spaces, and embedded \"quotes\" too";
   string out;

   // write in to a stringstream object
   ss1 << in;
   cout << "read in     [" << in << "]\n"
        << "stored as   [" << ss1.str() << "]\n";

   // read from a stringstream object
   ss1 >> out;
   cout << "written out [" << out << "]\n";
   cout << "----------------------------------------------" << endl;

   // write in to a stringstream object using quoted
   ss2 << quoted(in);

   cout << "read in     [" << in << "]\n"
        << "stored as   [" << ss2.str() << "]\n";

   // read from a stringstream object using quoted
   ss2 >> quoted(out);
```

```
    cout << "written out [" << out << "]\n";
}
```

****** Output ******

```
read in     [String with spaces, and embedded "quotes" too]
stored as   [String with spaces, and embedded "quotes" too]
written out [String]
----------------------------------------------------
read in     [String with spaces, and embedded "quotes" too]
stored as   ["String with spaces, and embedded \"quotes\" too"]
written out [String with spaces, and embedded "quotes" too]
```

Write your own manipulator

Example 7 - Write your own manipulator with no arguments

Technique: use a function with a stream argument, passed by reference and return the same stream.

```
#include <iostream>
using namespace std;

ostream& spaces3(ostream& os)
{
    return os << "   ";
}

int main()
{
    cout <<"Some" <<spaces3 <<"text" <<endl;
}
```

****** Output ******

```
Some   text
```

Example 8 - Write your own manipulator with one or more arguments

The following example illustrates a technique for creating a parameterized manipulator by creating a class with the same name.

```
#include <iostream>
#include <iomanip>
using namespace std;


struct prec
```

```
{
    prec(int x) : prec_(x) {}
    int prec_;
};

ostream& operator<<(ostream& out, const prec& obj)
{
    out.precision(obj.prec_);
    return out;
}

class dollar
{
    double amount;
public:
    dollar(double amt) : amount(amt) {}
    friend ostream& operator<<(ostream& out, const dollar& obj);
};

ostream& operator<<(ostream& out, const dollar& obj)
{
    out << '$';
    auto currentFlags = out.flags() ;
    auto currentPrecision = out.precision();
    out << fixed << setprecision(2) << obj.amount;
    out.flags(currentFlags);
    out.precision(currentPrecision);
    return out;
}


class format
{
    int width;
    int decimalPlaces;
public:
    format(int arg1, int arg2 = -1);
    friend ostream& operator<<(ostream& out, const format& obj);
};

format::format(int arg1, int arg2)
: width(arg2 == -1 ? 0: arg1),
  decimalPlaces(arg2 == -1 ? arg1: arg2)
{ }

ostream& operator<<(ostream& out, const format& obj)
{
    out << fixed << setw(obj.width)
        << setprecision(obj.decimalPlaces);
    return out;
}

int main( )
{
    double pi = 3.141592654;
```

```
    cout << prec(4) << pi << endl;
    cout << prec(6) << pi << endl;
    cout << prec(0) << pi << endl;
    cout << dollar(pi) << endl;
    cout << pi << endl;
    cout << "-----------------" << endl;

    // print with a width of 5 and 2 decimal places
    cout << '/' << format(5,2) << pi << '/' << endl;

    // print with a width of 12 and 4 decimal places
    cout << '/' << format(12,4) << pi << '/' << endl;

    // print with 1 decimal place
    cout << '/' << format(1) << pi << '/' << endl;
}
```

****** Output ******

```
3.142
3.14159
3
$3.14
3
-----------------
/ 3.14/
/      3.1416/
/3.1/
```

Data at the Bit Level

Data Storage at the bit level

Example 1 – Data storage

The following example shows how data is stored in stack memory. Eleven int variables are declared and initialized. The printVariableValueAndAddress() function displays the value of each variable in decimal and hexadecimal and its memory address in hexadecimal and decimal. The printMemoryContents() function displays the memory contents where the eleven variables are stored.

```
#include <iostream>
#include <iomanip>
using namespace std;

void printVariableValueAndAddress(char ch, const int&);
void printMemoryContents(unsigned char*, unsigned char*);

int main()
{
     int a = 1;
     int b = 12;
     int c = 123;
     int d = 1234;
     int e = 12345;
     int f = 123456;
     int g = 1234567;
     int h = 12345678;
     int i = 123456789;
     int j = 1234567890;
     int k = 12345678901;      // Warning!

     cout << "Var Dec Value   Hex Value   Hex Address  Dec Address"
          << endl;
     printVariableValueAndAddress('a', a);
     printVariableValueAndAddress('b', b);
     printVariableValueAndAddress('c', c);
     printVariableValueAndAddress('d', d);
     printVariableValueAndAddress('e', e);
     printVariableValueAndAddress('f', f);
     printVariableValueAndAddress('g', g);
     printVariableValueAndAddress('h', h);
     printVariableValueAndAddress('i', i);
     printVariableValueAndAddress('j', j);
     printVariableValueAndAddress('k', k);

     unsigned char* addr1 = reinterpret_cast<unsigned char*> (&k);
     unsigned char* addr2 = reinterpret_cast<unsigned char*> (&a)+3;
     printMemoryContents(addr1, addr2);
}

void printVariableValueAndAddress(char ch, const int& i)
```

```
{
    cout << left << showbase;
    cout << ch << " = " << setw(11) << i << ' ' << setw(12) << hex
         << i << dec << &i << "   " << reinterpret_cast<long> (&i)
         << endl;
}

void printMemoryContents(unsigned char* addr1,unsigned char* addr2)
{
    cout << endl << "Addresses / Contents" << endl;
    cout << hex << setfill('0') << noshowbase << right;
    for (unsigned char* addr = addr1; addr <= addr2; addr += 4)
    {
        // Memory addresses are stored in a width of 8 and
        // only the 8 least significant digits are displayed
        cout << setw(8) << reinterpret_cast<long>(addr)%0x100000000
             << ' ';
    }
    cout << noshowbase << left << endl;
    int i = 1;
    for (unsigned char* addr = addr1; addr <= addr2; ++addr, ++i)
    {
        cout << setw(2) << static_cast<int> (*addr);
        if (i && i % 4 == 0)
        {
            cout << ' ';
        }
    }
    cout << endl;
}
```

****** Output – NetBeans 8.2 (Windows) ******

```
Var Dec Value   Hex Value   Hex Address  Dec Address
a = 1           0x1         0xffffcbec   4294953964
b = 12          0xc         0xffffcbe8   4294953960
c = 123         0x7b        0xffffcbe4   4294953956
d = 1234        0x4d2       0xffffcbe0   4294953952
e = 12345       0x3039      0xffffcbdc   4294953948
f = 123456      0x1e240     0xffffcbd8   4294953944
g = 1234567     0x12d687    0xffffcbd4   4294953940
h = 12345678    0xbc614e    0xffffcbd0   4294953936
i = 123456789   0x75bcd15   0xffffcbcc   4294953932
j = 1234567890  0x499602d2  0xffffcbc8   4294953928
k = -539222987  0xdfdc1c35  0xffffcbc4   4294953924

Addresses / Contents
ffffcbc4 ffffcbc8 ffffcbcc ffffcbd0 ffffcbd4 ffffcbd8 ffffcbdc ffffcbe0
ffffcbe4 ffffcbe8 ffffcbec
351cdcdf d2209649 15cd5b70 4e61bc00 87d61200 40e21000 39300000 d2400000
7b000000 c0000000 10000000
```

****** Output – Code::Blocks (Windows) ******

```
Var Dec Value   Hex Value   Hex Address  Dec Address
a = 1           0x1         0x6dfef4   7208692
```

```
b = 12          0xc         0x6dfef0   7208688
c = 123         0x7b        0x6dfeec   7208684
d = 1234        0x4d2       0x6dfee8   7208680
e = 12345       0x3039      0x6dfee4   7208676
f = 123456      0x1e240     0x6dfee0   7208672
g = 1234567     0x12d687    0x6dfedc   7208668
h = 12345678    0xbc614e    0x6dfed8   7208664
i = 123456789   0x75bcd15   0x6dfed4   7208660
j = 1234567890  0x499602d2  0x6dfed0   7208656
k = -539222987  0xdfdc1c35  0x6dfecc   7208652

Addresses / Contents
006dfecc 006dfed0 006dfed4 006dfed8 006dfedc 006dfee0 006dfee4 006dfee8
006dfeec 006dfef0 006dfef4
351cdcdf d2209649 15cd5b70 4e61bc00 87d61200 40e21000 39300000 d2400000
7b000000 c0000000 10000000
```

Note: memory addresses are only 3 bytes in size

****** Output – Linux g++ version 7.3.0 ******

```
Var Dec Value   Hex Value   Hex Address  Dec Address
a = 1           0x1         0x7ffc74fb91ac   140722271130028
b = 12          0xc         0x7ffc74fb91a8   140722271130024
c = 123         0x7b        0x7ffc74fb91a4   140722271130020
d = 1234        0x4d2       0x7ffc74fb91a0   140722271130016
e = 12345       0x3039      0x7ffc74fb919c   140722271130012
f = 123456      0x1e240     0x7ffc74fb9198   140722271130008
g = 1234567     0x12d687    0x7ffc74fb9194   140722271130004
h = 12345678    0xbc614e    0x7ffc74fb9190   140722271130000
i = 123456789   0x75bcd15   0x7ffc74fb918c   140722271129996
j = 1234567890  0x499602d2  0x7ffc74fb9188   140722271129992
k = -539222987  0xdfdc1c35  0x7ffc74fb9184   140722271129988

Addresses / Contents
74fb9184 74fb9188 74fb918c 74fb9190 74fb9194 74fb9198 74fb919c 74fb91a0
74fb91a4 74fb91a8 74fb91ac
351cdcdf d2209649 15cd5b70 4e61bc00 87d61200 40e21000 39300000 d2400000
7b000000 c0000000 10000000
```

Note: memory addresses are 6 bytes in size

****** Output – MS Visual Studio 2017 ******

```
Var Dec Value   Hex Value   Hex Address  Dec Address
a = 1           0x1         001CFACC   1899212
b = 12          0xc         001CFAC0   1899200
c = 123         0x7b        001CFAB4   1899188
d = 1234        0x4d2       001CFAA8   1899176
e = 12345       0x3039      001CFA9C   1899164
f = 123456      0x1e240     001CFA90   1899152
g = 1234567     0x12d687    001CFA84   1899140
h = 12345678    0xbc614e    001CFA78   1899128
i = 123456789   0x75bcd15   001CFA6C   1899116
j = 1234567890  0x499602d2  001CFA60   1899104
k = -539222987  0xdfdc1c35  001CFA54   1899092
```

```
Addresses / Contents
001cfa54 001cfa58 001cfa5c 001cfa60 001cfa64 001cfa68 001cfa6c 001cfa70
001cfa74 001cfa78 001cfa7c 001cfa80 001cfa84 001cfa88 001cfa8c 001cfa90
001cfa94 001cfa98 001cfa9c 001cfaa0 001cfaa4 001cfaa8 001cfaac 001cfab0
001cfab4 001cfab8 001cfabc 001cfac0 001cfac4 001cfac8 001cfacc
351cdcdf cccccccc cccccccc d2209649 cccccccc cccccccc 15cd5b70 cccccccc
cccccccc 4e61bc00 cccccccc cccccccc 87d61200 cccccccc cccccccc 40e21000
cccccccc cccccccc 39300000 cccccccc cccccccc d2400000 cccccccc cccccccc
7b000000 cccccccc cccccccc c0000000 cccccccc cccccccc 10000000
```

Note: memory addresses are 3 bytes in size. The memory address display is in uppercase with no base indicators. The storage locations use 12 bytes of memory (8 bytes of padding).

Example 2 – Storage of negative ints

This example shows how negative int values are stored in memory.

```
#include <iostream>
#include <iomanip>
#include <string>
#include <cmath>
using namespace std;

void print(char ch, const int&);
string printIntInBinary(int arg);
int power(int pow);

 int main()
 {
     int a = 1;
     int b = -1;
     int c = 255;
     int d = -255;
     int e = 256;
     int f = -256;
     int g = 0x7fffffff;
     int h = -0x7fffffff;
     int i = 0x1a2b3c4d;
     int j = -0x1a2b3c4d;
     int k = 0xffffffff;
     int l = 0x00ff00ff;
     int m = -0x00ff00ff;
     cout << "Var Dec Value   Hex Value   Binary Value (4 bytes / 32 bits)" << endl;

     print('a', a);
     print('b', b);
     print('c', c);
     print('d', d);
     print('e', e);
     print('f', f);
     print('g', g);
     print('h', h);
     print('i', i);
```

```
    print('j', j);
    print('k', k);
    print('l', l);
    print('m', m);
}

void print(char ch, const int& i)
{
    cout << showbase;
    cout << setfill(' ') << ch << " = " << setw(11) << i << ' '
         << setw(10) << hex
         << i << dec << "  " << printIntInBinary(i)
         << endl;
}

string printIntInBinary(int arg)
{
    string value;
    for (auto i = 31; i >= 0; --i)
    {
        if (arg & power(i))
            value += '1';
        else
            value += '0';
        if (i%8 == 0)
            value += ' ';
    }
    return value;
}

int power(int pow)
{
    int value = 1;
    for (auto i = 0; i < pow; ++i)
        value *= 2;
    return value;
}
```

****** Output ******

```
Var Dec Value   Hex Value   Binary Value (4 bytes / 32 bits)
a =           1        0x1  00000000 00000000 00000000 00000001
b =          -1 0xffffffff  11111111 11111111 11111111 11111111
c =         255       0xff  00000000 00000000 00000000 11111111
d =        -255 0xffffff01  11111111 11111111 11111111 00000001
e =         256      0x100  00000000 00000000 00000001 00000000
f =        -256 0xffffff00  11111111 11111111 11111111 00000000
g =  2147483647 0x7fffffff  01111111 11111111 11111111 11111111
h = -2147483647 0x80000001  10000000 00000000 00000000 00000001
i =   439041101 0x1a2b3c4d  00011010 00101011 00111100 01001101
j =  -439041101 0xe5d4c3b3  11100101 11010100 11000011 10110011
k =          -1 0xffffffff  11111111 11111111 11111111 11111111
l =    16711935   0xff00ff  00000000 11111111 00000000 11111111
m =   -16711935 0xff00ff01  11111111 00000000 11111111 00000001
```

To convert a positive int value to negative, “flip” the bits and add 1. This is the two’s complement method of storing negative int values. For negative int values, the high order (left-most) bit is 1. This is called the sign bit.

Example 3 – Non-primitive data at the bit level

```
#include <iostream>
#include <iomanip>
#include <climits>
using namespace std;

long address2long(const void* address);
unsigned powerOf2(int exp);
template <typename T> void printBits(T type);

struct Struct1
{
    char c1;
    char c2;
    short s1;
    int i;
};

ostream& operator<<(ostream& out, const Struct1& d)
{
    out << "Address: " << address2long(&d) << "   " << sizeof(d) << " bytes" << endl;
    out << "    &c1: " << address2long(&d.c1);
    printBits(d.c1);
    out << "    &c2: " << address2long(&d.c2);
    printBits(d.c2);
    out << "    &s1: " << address2long(&d.s1);
    printBits(d.s1);
    out << "     &i: " << address2long(&d.i);
    printBits(d.i);
    return out;
}


struct Struct2
{
    char c1;
    int i;
    char c2;
    short s1;
};

ostream& operator<<(ostream& out, const Struct2& d)
{
    out << "Address: " << address2long(&d) << "   " << sizeof(d) << " bytes" << endl;
    out << "    &c1: " << address2long(&d.c1);
    printBits(d.c1);
```

```
    out << "     &i: " << address2long(&d.i);
    printBits(d.i);
    out << "    &c2: " << address2long(&d.c2);
    printBits(d.c2);
    out << "    &s1: " << address2long(&d.s1);
    printBits(d.s1);
    return out;
}

int main()
{
    Struct1 s1 = {'A','B',static_cast<short>(13),55};
    printBits(s1);
    cout << endl;
    Struct2 s2 = {'A',55,'B',static_cast<short>(13)};
    printBits(s2);
}


long address2long(const void* address)
{
    return reinterpret_cast<long>(address);
}

template <typename T>
void printBits(T t)
{
    cout << setw(6) << t << "   ";

    unsigned mask;
    unsigned char* ptr;
    for (size_t i = 0; i < sizeof(T); i++)
    {
        // Advance ptr each byte of the argument
        ptr = reinterpret_cast<unsigned char*>(&t) + i;

        // Print the contents of the byte
        for (int i = 7; i >= 0; --i)
        {
            mask = powerOf2(i);
            cout << (*ptr & mask ? 1 : 0);
        }
        cout << "  ";
    }
    cout << endl;
}

unsigned powerOf2(int exp)
{
    unsigned value = 1;
    for (int i = 0; i < exp; ++i)
    {
        value *= 2;
    }
    return value;
```

```
101  }
```

****** Output ******

```
Address: 4294953904   8 bytes
    &c1: 4294953904     A   01000001
    &c2: 4294953905     B   01000010
    &s1: 4294953906    13   00001101  00000000
     &i: 4294953908    55   00110111  00000000  00000000  00000000
   01000001  01000010  00001101  00000000  00110111  00000000
00000000  00000000

Address: 4294953936   12 bytes
    &c1: 4294953936     A   01000001
     &i: 4294953940    55   00110111  00000000  00000000  00000000
    &c2: 4294953944     B   01000010
    &s1: 4294953946    13   00001101  00000000
   01000001  00000000  00000000  00000000  00110111  00000000
00000000  00000000  01000010  00000000  00001101  00000000
```

Note: The bit representation may vary between big endian and little endian platforms. The contents of “padded” bytes may also vary.

Bitwise Operators

Operator	Symbol Name
&	and
|	or
^	exclusive or
~	not (a unary operator)
<<	left-shift
>>	right-shift
&=	and assignment
|=	or assignment
^=	exclusive or assignment
<<=	left shift assignment
>>=	right shift assignment

& operator

The bitwise and operator returns a 1 only when both bits being compared are 1. For example:

10101110 & 00101010 ➔ 00101010

| operator

The bitwise or operator returns a 1 only when either bits being compared are 1. For example:

10101110 | 00101010 ➔ 10101110

^ operator

The bitwise exclusive or operator returns a 1 only when either, but not both, bits being compared are 1. For example:

10101110 | 00101010 ➔ 10000100

~ operator

The bitwise not, or complement operator is a unary bitwise operator. It returns a 1 when the bit is 0 and returns a 0 when the bit is 1. For example:

~10101110 ➔ 01010001

<< operator

The bitwise left-shift operator shifts bits to left the number of positions as the right-hand operand. Bits on the right are filled with zeros. Bits on the left are lost. The left-shift operator may be used to perform multiplication by integer powers of two. For example,

10101110 << 2 ➔ …10 10111000

>> operator

The bitwise right-shift operator shifts bits to right the number of positions as the right-hand operand. Bits on the left are filled with zeros. Bits on the right are lost. The left-shift operator may be used to perform division by integer powers of two. For example,

10101110 >> 2 ➔ 00101011 10…

The bitwise assignment operators

The bitwise assignment operators: &=, |=, ^=, <<=, and >>= perform the implied operation and assign the resultant value to the left-hand argument.

Example 3 – Bitwise operators

```
#include <iostream>
#include <iomanip>
#include <climits>
using namespace std;
```

```
unsigned powerOf2(int exp);
template <typename T> void printBits(T type);


int main()
{
    unsigned char a = 77;
    unsigned char b = 20;
    cout << "  a =";printBits(a);
    cout << "  b =";printBits(b);
    cout << "a&b =";printBits(a&b);
    cout << "a|b =";printBits(a|b);
    cout << "a^b =";printBits(a^b);
    cout << " ~a =";printBits(~a);
    cout << "a<<1=";printBits(a<<1);
    cout << "a<<2=";printBits(a<<2);
    cout << "a<<8=";printBits(a<<8);
    cout << "a<<9=";printBits(a<<9);
    cout << "a>>1=";printBits(a>>1);
    cout << "a>>2=";printBits(a>>2);
    cout << "a>>9=";printBits(a>>9);
}

template <typename T>
void printBits(T t)
{
    unsigned mask;
    unsigned char* ptr;
    cout << setw(5) << static_cast<int>(t) << " ";
    for (size_t i = 0; i < sizeof(T); i++)
    {
        // Advance ptr each byte of the argument
        ptr = reinterpret_cast<unsigned char*>(&t) + i;

        // Print the contents of the byte
        for (int i = 7; i >= 0; --i)
        {
            mask = powerOf2(i);
            cout << (*ptr & mask ? 1 : 0);
        }
        cout << "  ";
    }
    cout << endl;
}

unsigned powerOf2(int exp)
{
    unsigned value = 1;
    for (int i = 0; i < exp; ++i)
    {
        value *= 2;
    }
    return value;
}
```

****** Output ******

```
  a =   77 01001101
  b =   20 00010100
a&b =    4 00000100  00000000  00000000  00000000
a|b =   93 01011101  00000000  00000000  00000000
a^b =   89 01011001  00000000  00000000  00000000
 ~a =  -78 10110010  11111111  11111111  11111111
a<<1=  154 10011010  00000000  00000000  00000000
a<<2=  308 00110100  00000001  00000000  00000000
a<<8=19712 00000000  01001101  00000000  00000000
a<<9=39424 00000000  10011010  00000000  00000000
a>>1=   38 00100110  00000000  00000000  00000000
a>>2=   19 00010011  00000000  00000000  00000000
a>>9=    0 00000000  00000000  00000000  00000000
```

Bitwise Techniques

Turn a bit on

Use the or assignment bitwise operator to turn a bit on. If the bit is already turned on, the operation has no effect.

Integer_value |= bit

Turn a bit off

Use the and assignment with the not bitwise operators to turn a bit off. If the bit is already turned on, the operation has no effect.

Integer_value &= ~bit

Toggle a bit

Use the exclusive or assignment operator to turn a bit off.

Integer_value ^= bit

Test a bit

Use the and operator to see if a bit is turned on.

Integer_value & bit

Example 4 – Bitwise operator techniques

```
#include <iostream>
#include <iomanip>
using namespace std;

unsigned powerOf2(int exp);
template <typename T> void printBits(T type);

int main()
{
    unsigned char a;
    unsigned char b;

    // turn a bit on
    a = 34;
    cout << " a =";printBits(a);
    b= 4;
    cout << " b =";printBits(b);
    cout << "a|=b"; printBits(a|=b); cout << endl;

    // turn a bit off
    a = 34;
    cout << " a =";printBits(a);
    b= 2;
    cout << " b =";printBits(b);
    cout << "a&~b"; printBits(a&~b); cout << endl;

    // toggle a bit
    a = 34;
    cout << " a =";printBits(a);
    b= 66;
    cout << " b =";printBits(b);
    cout << "a^=b"; printBits(a^=b); cout << endl;

    // test to see if a bit is turned on
    a = 34;
    cout << boolalpha;
    cout << " a =";printBits(a);
    cout << " 2 =";printBits(2);
    cout << "a & 2 = " << static_cast<bool>(a & 2) << endl;
    cout << " 4 =";printBits(4);
    cout << "a & 4 = " << static_cast<bool>(a & 4) << endl;
}

template <typename T>
void printBits(T t)
{
    unsigned mask;
    unsigned char* ptr;
    cout << setw(5) << static_cast<int>(t) << " ";
    for (size_t i = 0; i < sizeof(T); i++)
    {
        // Advance ptr each byte of the argument
```

```
        ptr = reinterpret_cast<unsigned char*>(&t) + i;

        // Print the contents of the byte
        for (int i = 7; i >= 0; --i)
        {
            mask = powerOf2(i);
            cout << (*ptr & mask ? 1 : 0);
        }
        cout << "  ";
    }
    cout << endl;
}

unsigned powerOf2(int exp)
{
    unsigned value = 1;
    for (int i = 0; i < exp; ++i)
    {
        value *= 2;
    }
    return value;
}
```

****** Output ******

```
 a =   34 00100010
 b =    4 00000100
a|=b   38 00100110

 a =   34 00100010
 b =    2 00000010
a&~b   32 00100000  00000000  00000000  00000000

 a =   34 00100010
 b =   66 01000010
a^=b   96 01100000

 a =   34 00100010
 2 =    2 00000010  00000000  00000000  00000000
a & 2 = true
 4 =    4 00000100  00000000  00000000  00000000
a & 4 = false
```

Practical Applications

The following examples illustrate working with binary data.

Example 5 – Bitwise operator techniques

The following example shows how to extract each nibble (4 bits) from a byte.

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;

string uchar2binary(unsigned char);
unsigned char powerOf2(unsigned char exp);

int main()
{
    unsigned char x;
    cout << showbase;
    for (auto i = 0; i < 10; i++)
    {
       x = rand() % 255;                                // 0-255
       cout << dec << setw(5) << static_cast<int>(x)    // decimal
            << hex << setw(8) << static_cast<int>(x)    // hex
            << setw(12) << uchar2binary(x)              // binary
            << setw(12) << uchar2binary(x >> 4)   // first nibble
            << setw(12) << uchar2binary(x & 0xf)  // second nibble
            << endl;
   }
}

// returns unsigned char as a binary string
string uchar2binary(unsigned char arg)
{
    string out;
    unsigned char mask;
    for (auto i = 7; i >= 0; --i)
    {
        mask = powerOf2(i);
        out += (arg & mask ? '1' : '0');
    }
    return out;
}

// returns 2 raised to exp power
unsigned char powerOf2(unsigned char exp)
{
    unsigned char value = 1u;
    for (auto i = 0u; i < exp; ++i)
    {
       value *= 2u;
```

```
45     }
46     return value;
47  }
```

****** Output ******

```
 41    0x29   00101001   00000010   00001001
107    0x6b   01101011   00000110   00001011
214    0xd6   11010110   00001101   00000110
235    0xeb   11101011   00001110   00001011
 44    0x2c   00101100   00000010   00001100
169    0xa9   10101001   00001010   00001001
  3     0x3   00000011   00000000   00000011
 33    0x21   00100001   00000010   00000001
187    0xbb   10111011   00001011   00001011
239    0xef   11101111   00001110   00001111
```

Explanation

This example makes use of an unsigned char to limit the perspective to just one byte.

Line 19: The first nibble is extracted by shifting the 8 bits to the right by 4. The right shift bitwise operator returns an int (32 bits). That int result is then passed to the uchar2binary function which is converted to an unsigned char.

Line 20: The second nibble is extracted using a 0xf mask with the bitwise ***and*** operator. Keep in mind that mask is 00001111 in binary. With this mask the second nibble bits will be replicated.

Example 6 – Extracting specified bits from a byte

The following example shows how to extract a specified number of bits from a byte. The user specifies the starting bit and the number of bits to extract. The default argument, numbits = 8, allows the user to specify only a starting bit. In that case the function will return all bits from the starting bit to the end of the byte. The problem is solved using the getBitsFromByte function. Note that a byte is returned, not just the specified number of bits. This is because there is no built-in type for less than 8 bits.

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;

string uchar2binary(unsigned char);
unsigned char powerOf2(unsigned char exp);
unsigned char getBitsFromByte(unsigned char byte,
                     unsigned startingBit, unsigned numbits = 8u);

int main()
{
    unsigned char x, sb, nb;
```

```
    cout << showbase;
    for (auto i = 0; i < 15; i++)
    {
        x = rand() % 255;              // unsigned char 0-255
        sb = rand() % 8;               // starting bit 0-7
        nb = rand() % (9-sb);          // number of bits 0-8

        cout << dec << setw(4) << static_cast<int>(x) // decimal
             << hex << setw(6) << static_cast<int>(x) // hex
             << setw(10) << uchar2binary(x);          // binary
        cout << dec;
        if (nb)
        {
            cout << "  sb=" << static_cast<int>(sb)  // start bit
                 << "  nb=" << static_cast<int>(nb)  // num bits
                 << " => "
                 << uchar2binary(getBitsFromByte(x,sb,nb));
        }
        else
        {
            cout << "  sb=" << static_cast<int>(sb) // start bit
                 << "      "
                 << " => "
                 << uchar2binary(getBitsFromByte(x,sb));
        }
        cout << endl;
    }
}

// returns unsigned char as a binary string
string uchar2binary(unsigned char arg)
{
    string out;
    unsigned char mask;
    for (auto i = 7; i >= 0; --i)
    {
        mask = powerOf2(static_cast<unsigned char>(i));
        out += (arg & mask ? '1' : '0');
    }
    return out;
}

unsigned char powerOf2(unsigned char exp)
{
    unsigned char value = 1u;
    for (auto i = 0u; i < exp; ++i)
        value <<= 1;
    return value;
}

// assume bits are numbered 0-7, left-to-right
unsigned char getBitsFromByte(unsigned char byte,
                          unsigned startingBit, unsigned numBits)
{
    byte <<= startingBit;        // shift bits left
```

```
69      byte >>= (8 - numBits);       // shift bits right
70      return byte;
71  }
```

****** Output ******

```
 41  0x29  00101001  sb=3  nb=4 => 00000100
235  0xeb  11101011  sb=1  nb=4 => 00001101
  3   0x3  00000011  sb=6  nb=1 => 00000001
239  0xef  11101111  sb=1  nb=1 => 00000001
 76  0x4c  01001100  sb=3  nb=1 => 00000000
236  0xec  11101100  sb=3  nb=2 => 00000001
237  0xed  11101101  sb=4  nb=1 => 00000001
 69  0x45  01000101  sb=6       => 01000000
 37  0x25  00100101  sb=6       => 01000000
101  0x65  01100101  sb=6  nb=2 => 00000001
 92  0x5c  01011100  sb=6  nb=2 => 00000000
 63  0x3f  00111111  sb=5       => 11100000
167  0xa7  10100111  sb=3  nb=1 => 00000000
204  0xcc  11001100  sb=7  nb=1 => 00000000
212  0xd4  11010100  sb=5  nb=1 => 00000001
```

Explanation

As in the previous example, type unsigned char is used to represent the byte. The method for extraction in the getBitsFromByte function involves shifting the unwanted bits off the left side of the byte, then off the right side of the byte.

Line 68: Bits to the left of the starting bit are shifted off the left side. Notice the use of the <<= operator instead of the << operator. In both cases, an int (32 bits) is returned. With the << operator the unspecified bits to the left of the starting bit would be shift into the next byte. They would then reappear in a right shift. By using <<= the result of the left shift is stored into the unsigned char (one byte), so the is no problem in the subsequent right shift.

Line 69: Bits a shifted to the right so that exactly the number of bits desired are remaining, right justified in the byte.

Multiple Inheritance

Multiple inheritance permits a class to be derived from two (or more) other classes. In this way the derived classes inherits the members and properties of both (or more) base classes.

Example 1 – Multiple Inheritance

```
// Easy multiple inheritance example

#include <iostream>
using namespace std;

class one
{
protected:
    int a,b;
public:
    one(int z,int y) : a(z), b(y)
    { }
    void show() const
    {
        cout << a << ' ' << b << endl;
    }
};

class two
{
protected:
    int c,d;
public:
    two(int z,int y) : c(z), d(y)
    { }
    void show() const
    {
        cout << c << ' ' << d << endl;
    }
};

class three : public one, public two
{
private:
    int e;
public:
    three(int,int,int,int,int);
    void show() const
    {
        cout << a << ' ' << b << ' ' << c << ' ' << d << ' ' << e << endl;
    }
};

three::three(int a1, int a2, int a3, int a4, int a5)
```

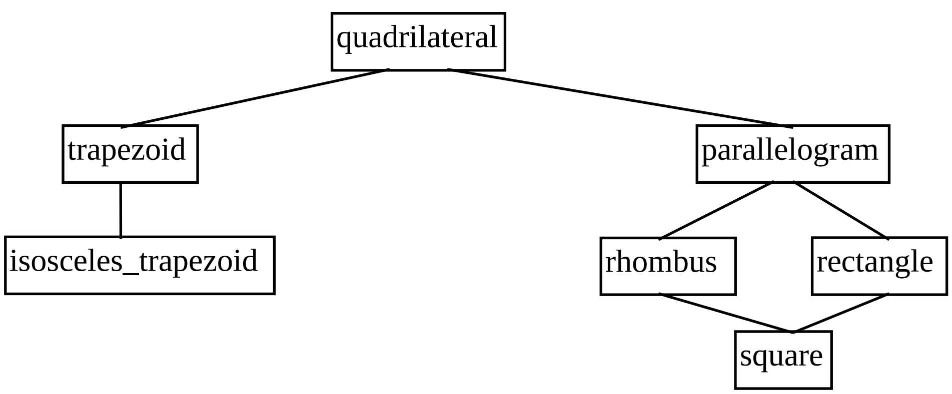
```
     : one(a1,a2),two(a3,a4), e(a5)
{ }

int main()
{
    one abc(5,7);
    abc.show();   // prints 5 7
    two def(8,9);
    def.show();   // prints 8 9
    three ghi(2,4,6,8,10);
    ghi.show();   // prints 2 4 6 8 10
}
```

****** Output ******

```
5 7
8 9
2 4 6 8 10
```

Multiple Inheritance with Virtual Base Classes

The next example illustrates a more complicated inheritance situation. It models the relationship between types of quadrilaterals. This relationship is shown in the following figure:



Note that the parallelogram class will be derived from the quadrilateral class, both the rhombus and rectangle classes will be derived from the parallelogram class. And the square is derived from both the rhombus and the rectangle classes. It's the square class that makes this multiple inheritance.

Example 2 - Multiple Inheritance with Virtual Base classes

```
#include <iostream>
#include <cmath>
using namespace std;

class quadrilateral
{
```

```
protected:
    double a,b,c,d;
public:
    quadrilateral(double s1,double s2,double s3,double s4)
        : a(s1), b(s2), c(s3), d(s4) {}
    quadrilateral() : a(0), b(0), c(0), d(0) {}
    void show()
    {
        cout << "quadrilateral: " << this << "  sides " <<
             a << ' ' << b << ' ' << c << ' ' << d << endl;
    }
};

class trapezoid : public quadrilateral
{
public:
    trapezoid(double base1, double base2, double leg1, double leg2)
        : quadrilateral(base1,leg1,base2,leg2) {}
};

class isosceles_trapezoid : public trapezoid
{
public:
    isosceles_trapezoid(double base1, double base2, double leg)
        : trapezoid(base1,leg,base2,leg) {}
};

class parallelogram : public quadrilateral
{
protected:
    int angle;
public:
    parallelogram(double s1,double s2, int ang)
        : quadrilateral(s1,s2,s1,s2), angle(ang)
    { }
    parallelogram() : angle(0) { }
    void show_angles(void)
    {
        cout << "angles = " << angle << ' ' << (180-angle) << endl;
    }
};

class rectangle : virtual public parallelogram
{
public:
    rectangle(double base, double height)
        : parallelogram(base,height,90) {}
    rectangle() {}
};

class rhombus: virtual public parallelogram
{
public:
    rhombus(double side,int ang) : parallelogram(side,side,ang) {}
    rhombus() {}
```

```
};

class square : public rhombus,public rectangle
{
public:
    square(double side) : parallelogram(side,side,90) {}
};

int main(void)
{
    quadrilateral q1(1,2,3,4);
    q1.show();
    trapezoid q2(22,13,8,15);
    q2.show();
    isosceles_trapezoid q3(18,8,13);
    q3.show();
    parallelogram q4(4,3,45);
    q4.show();
    q4.show_angles();
    rectangle q5(4,3);
    q5.show();
    q5.show_angles();
    rhombus q6(5,45);
    q6.show();
    q6.show_angles();
    cout << endl;
    square q7(5);
    q7.show();
    q7.show_angles();
}
```

Exception Handling

Exception handling in C++ is methodology used to deal with error conditions that usually results in a program failure. These methods are implemented using:

- the try, throw, and catch keywords in C++
- exception class types
- functions, such as set_terminate() and set_unexpected() found in the header files, <stdexcept> and <exception>.

They allow the user to detect specific errors and control the program exit or recover and continue the program. Exception handling is used to handle exceptional situations, not to replace typical error messages.

Exception handling is a standard feature of the language.

Exception handling is designed to provide an alternate means of handling a code situation which would normally abend or abort a program. This mechanism allows transfer of control to another location where the error may be "handled". The transfer is specified by a throw expression. This expression allows the user to pass a value to the "handler". The "handler" catches the thrown expression by matching the type of the throw and deals with the problem as the author desires.

When are Exception Handling Methods Appropriate?

As stated earlier, exception handling is for the exceptional situation, not the common. Consider the following application:

1. A training (relational) database, written in C++, is used to track student training, enrollments, class schedules, etc. How should the following situations be "handled"?

2. A student trying to enroll in a course, but doesn't have the prerequisites for it?

3. A student tries to enroll in a class that is full.

4. A student tries to enroll in a class that is identified as open, but is refused, because the class is really full.

5. A student tries to enroll in a class, but is already enrolled in another section of the same course.

6. A student tries to enroll in a course that is retired.

7. A student tries to enroll in a course in which there are no sections scheduled.

8. A student tries to enroll in a class section, but the schedule record containing the date and number of students is missing or defective.

9. A student tries to enroll in a course, but enters the incorrect course number.

Previous Error Handling Methods

The assert() Macro

A common way of dealing with error conditions is the use of the assert() macro. This macro is most often used in program development to insure that certain conditions are true during the execution of a program. If the assert condition is false, the program aborts displaying an assert diagnostic message. The assert() macro is declared in the <cassert> header file.

Note, the assert macro can be suppressed if the macro, NDEBUG is defined before the <cassert> header file is included, like this:

```
#define NDEBUG
#include <cassert>
```

The following example illustrates its use.

Example 1 - assert

```
#include <iostream>
#include <cassert>
#include <cstdlib>
using namespace std;

class Fraction
{
   int numer, denom;
public:
   Fraction(int n = 0, int d = 1) : numer(n), denom(d)
   {
         assert(denom!=0);     // make sure denom is not 0
   }
   friend ostream& operator<<(ostream& o, const Fraction& f)
   {
         return (o << f.numer << '/' << f.denom);
   }
};

int main()
{
   int i1, i2;
   cout << "Enter two ints => ";
   cin >> i1 >> i2;
   if (cin.good())
   {
         Fraction f(i1,i2);
         cout << f << endl;
   }
   else cerr << "Bad input\n";
   cout << "*** End of Program ***\n";
}
```

****** Sample Run #1 ******

```
Enter two ints => 1 2
1/2
*** End of Program ***
```

****** Sample Run #2 Code::Blocks ******

```
Enter two ints => 2 0
Assertion failed: denom!=0, file ex10-1.cpp, line 13

This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.
```

****** Sample Run #2 Linux ******

```
Enter two ints => 2 0
assertion "denom!=0" failed: file "ex10-1.cpp", line 12, function:
Fraction::Fraction(int, int)
Aborted (core dumped)
```

Note: this approach is used to catch a run-time error. This is not a compile error. Of course, there are other ways of handling this problem. The programmer could put a check in main() to verify that the second int entered is non-zero. Another approach is to put a check for a denom = 0 in the fraction constructor. The problem, of course, could be "handled" not by aborting the program, but maybe by asking the user for another denominator. This may not always be feasible, since the numerator may not always be supplied by the user. Maybe it's a problem that you want to recognize, but continue the program execution. This is known as *fault-tolerant processing*.

The longjmp() function

The longjmp() function is an ANSI C standard function that may be used the jump out of a function containing an error. longjmp() executes after a setjmp() function has be called to capture and store the task state of the program. longjmp() causes a "rollback" of the program state to a previous time. The advantage of this approach is that an error situation may be detected and corrected and the offending code may be rerun.

Example 2 – longjump()

```
#include <iostream>
#include <cstdlib>
using namespace std;
#include <setjmp.h>

jmp_buf jumper;     // declare a jump buffer to save program state

class Fraction
{
     int numer, denom;
 public:
     Fraction(int n = 0, int d = 1) : numer(n), denom(d)
```

```
    {
        cout << "Fraction " << this << " created" << endl;
        if (d == 0)
            longjmp(jumper,1);     // make sure denom is not 0
    }

    ~Fraction()
    {
       cout << "~Fraction " << this << " destroyed" << endl;
    }

    friend ostream& operator<<(ostream& o, const Fraction& f)
    {
        return (o << f.numer << '/' << f.denom);
    }
};

int main()
{
    int i1, i2;
    int state;
    state = setjmp(jumper);
    if (state != 0)
        cout << "** Go back in time with state  " << state << endl;

    cout << "Enter two ints => ";
    cin >> i1 >> i2;

    Fraction f(i1,i2);
    cout << f << endl;

    cout << "*** End of Program ***\n";
}
```

****** Sample Run 1 ******

```
Enter two ints => 2 3
Fraction 0x6dfedc created
2/3
*** End of Program ***
~Fraction 0x6dfedc destroyed
```

****** Sample Run 2 ******

```
Enter two ints => 2 0
Fraction 0x6dfedc created
** Go back in time with state  1
Enter two ints => 2 3
Fraction 0x6dfedc created
2/3
*** End of Program ***
~Fraction 0x6dfedc destroyed
```

- What is wrong with this approach?

Exception Handling Basics

try, throw, and catch

Exception handling is, for the most part, accomplished using three keywords, try, throw, and catch. The try block contains code that may result in an error. The error is detected and you throw an exception-expression. The handling is accomplished by a catch of the expression. The following example illustrates the technique.

Example 3 – try, throw, catch

```
#include <iostream>
#include <cstdlib>
using namespace std;

class Fraction
{
    int numer, denom;
public:
    Fraction(int n = 0, int d = 1) : numer(n), denom(d)
     {
         cout << "Fraction " << this << " created" << endl;
         if (d == 0)
             throw("Error:  denominator = 0");
     }

     ~Fraction()
     {
         cout << "~Fraction " << this << " destroyed" << endl;
     }

     friend ostream& operator<<(ostream& o, const Fraction& f)
     {
         return (o << f.numer << '/' << f.denom);
     }
 };

 int main()
 {
     int i1, i2;

     cout << "Enter two ints => ";
     cin >> i1 >> i2;
     try
     {
         Fraction f(i1,i2);
         cout << f << endl;
     }
     catch (const string& errmsg)
     {
         cerr << errmsg <<endl;
     }
```

```
    cout << "*** End of Program ***\n";
}
```

****** Sample Run 1 ******

```
Enter two ints => 2 3
Fraction 0x6dfedc created
2/3
~Fraction 0x6dfedc destroyed
*** End of Program ***
```

****** Sample Run 2 on Code::Blocks ******

```
Enter two ints => 2 0
Fraction 0x6dfedc created
terminate called after throwing an instance of 'char const*'

This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.
```

****** Sample Run 2 on Linux (voyager) ******

```
Enter two ints => 2 0
Fraction 0x7fffc4477540 created
terminate called after throwing an instance of 'char const*'
Aborted
```

- How is this program an improvement?
- Is there a problem?

Example 4 – Handling a file open error

Here's an example of handling a file open error. The user is given the option to try again.

```
#include <fstream>
#include <iostream>
#include <string>
#include <cstdlib>
using namespace std;

int main()
{
    ifstream fin;
    string filename;
    cout << "Enter filename => ";
    cin >> filename;

    try
    {
        fin.open(filename);
        if (fin.is_open())
        {
            cout << "file " << filename << " opened\n";
```

```
        }
        else
            throw(string("Can't open file ") + filename);
    }
    catch (const string& errmsg)
    {
        cout << errmsg << "\nTry again? ";
        char yn;
        cin >> yn;
        if (yn == 'y')
        {
            fin.clear();
            cout << "Enter filename => ";
            cin >> filename;
            fin.open(filename);
            if (!fin)
            {
                cout << "I quit!  I can't find file " << filename << " either.\n";
            }
            else
            {
                cout << "file " << filename << " opened\n";
            }
        }
        else
        {
            cout << "I didn't think you wanted to open a file anyway!\n";
        }
    }

    cout << "*** End of Program ***\n";
}
```

```
 ******  Sample Run 1 ******

Enter filename => ex10-4.cpp
file ex10-4.cpp opened
*** End of Program ***

******  Sample Run 2 ******

Enter filename => ex10-4.ccp
Can't open file ex10-4.ccp
Try again? n
I didn't think you wanted to open a file anyway!
*** End of Program ***

******  Sample Run 3 ******

Enter filename => ex10-4.ccp
Can't open file ex10-4.ccp
Try again? y
Enter filename => ex10-4.cpc
I quit!  I can't find file ex10-4.cpc either.
*** End of Program ***

******  Sample Run 4 ******

Enter filename => ex10-4.ccp
Can't open file ex10-4.ccp
Try again? y
Enter filename => ex10-4.cpp
file ex10-4.cpp opened
*** End of Program ***
```

Later we'll look at a technique for "re-throwing" the same **throw**.

This next example shows two different styles for throwing exceptions.
The first five exceptions occur in and are handled in main(). The next five occur and are handled in another function called by main().

Example 5 – Where to throw, where to catch

```
#include <iostream>

void funk(int it)
{
    try
    {
        throw it;
    }
    catch(int whatever)
     {
         std::cout << "I caught a " << whatever << std::endl;
     }
 }

```

```
int main()
{
    for (auto up = 1; up <= 5; up++)
    {
        try
        {
            throw up;
        }
        catch(int z)
        {
            std::cout << "You threw me a " << z << std::endl;
        }
    }
    for (auto i = 16; i <= 20; i++)
        funk(i);

    std::cout << "End of program\n";
}
```

****** Output ******

```
You threw me a 1
You threw me a 2
You threw me a 3
You threw me a 4
You threw me a 5
I caught a 16
I caught a 17
I caught a 18
I caught a 19
I caught a 20
End of program
```

Example 6 - Throwing and catching more than one type

It is common to throw more than one type in a program. The following example illustrates shows how this is handled.

Note: When a user-defined type is thrown, the copy constructor is used to create the thrown object.

```
#include <iostream>
#include <string>
using namespace std;

class Dog
{
    string name;
    string breed;
public:
    Dog(const string& n = "Fido", const string& b = "mutt")
    : name(n), breed (b) { }
    friend ostream& operator<<(ostream& o,const Dog& dog)
```

```
    {
        return (o << dog.name << " is a " << dog.breed);
    }
};

void funk(int i)
{
    try
    {
        switch (i)
        {
        case 1:
            throw("Have a nice day");
        case 2:
            throw(5);
        case 3:
            throw(3.14);
        case 4:
            throw(5L);
        case 5:
            throw(&i);
        case 6:
            throw(Dog());
        }
    }
    catch(const char* it)
    {
        cout << "You threw me a const char*: " << it << endl;
    }
    catch (const string& it)
    {
        cout << "You threw me a const string&: " << it << endl;
    }
    catch(int it)
    {
        cout << "You threw me an int: " << it << endl;
    }
    catch(float it)
    {
        cout << "You threw me a float: " << it << endl;
    }
    catch(double it)
    {
        cout << "You threw me a double: " << it << endl;
    }
    catch(long it)
    {
        cout << "You threw me long: " << it << endl;
    }
    catch(int* it)
    {
        cout << "You threw me an int address: " << it << endl;
    }
    catch(Dog it)
    {
```

```
        cout << "You threw me an Dog: " << it << endl;
    }
}

int main()
{
    funk(1);
    funk(2);
    funk(3);
    funk(4);
    funk(5);
    funk(6);
    cout << "End of program\n";
}
```

****** Output ******

```
You threw me a const char*: Have a nice day
You threw me an int: 5
You threw me a double: 3.14
You threw me long: 5
You threw me an int address: 0x6dff00
You threw me an Dog: Fido is a mutt
End of program
```

- ✔ Which catch did not get used?
- ✔ What if you throw a type that you haven't written a catch for?

Example 7 - Unhandled Exceptions

This example shows what happens if you don't write a catch for the type that you throw. This is called an unhandled exception.

```
#include <iostream>
#include <string>
using namespace std;

void funk(int i)
{
    try
    {
        switch (i)
         {
         case 1:
             throw(string("Have a nice day"));
         case 2:
             throw(5);
         case 3:
             throw(3.14);
         }
    }
    catch(const string& it)
    {
```

```
        cerr << "You threw me a string: " << it << endl;
    }

    catch(double it)
    {
        cerr << "You threw me a double: " << it << endl;
    }
}

int main()
{
    funk(1);
    funk(2);
    funk(3);
    cout << "End of program\n";
}
```

******** Output ********

```
You threw me a const char*: Have a nice day
Abnormal program termination
```

Example 8 - How to catch anything

You may use **catch(...)** to catch a throw of a type for which you have not specified a catch.

```
#include <iostream>
#include <string>
using namespace std;

void funk(int i)
{
    try
    {
        switch (i)
        {
        case 0:
            throw(0);
        case 1:
            throw(string("Have a nice day"));
        case 2:
            throw(5);
        case 3:
            throw(3.14);
        }
    }
    catch (const string& it)
    {
        cout << "You threw me a string: " << it << endl;
    }
    catch(const char* it)
    {
        cout << "You threw me a const char*: " << it << endl;
```

```
    }
    catch(double it)
    {
        cout << "You threw me a double: " << it << endl;
    }
    catch(...)
    {
        cout << "You threw me something.  I know not what!\n";
    }
}

int main()
{
    funk(1);
    funk(2);
    funk(3);
    funk(0);
    cout << "End of program\n";
}
```

```
****** Output ******

You threw me a string: Have a nice day
You threw me something.  I know not what!
You threw me a double: 3.14
You threw me something.  I know not what!
End of program
```

Example 9 - Exception Handling Classes

It might be a good idea to create a class to handle the exception.

```
#include <iostream>
#include <string>
using namespace std;

class ZeroDenominator
{
public:
    ZeroDenominator() {}
    friend ostream& operator<<(ostream& out, const ZeroDenominator& error);
};

class Fraction
{
    int numer, denom;
public:
    Fraction(int n = 0, int d = 1) : numer(n), denom(d)
    {
        cout << "Fraction constructor called\n";
        if (denom == 0) throw ZeroDenominator();
    }
```

```
    ~Fraction()
    {
        cout << "Fraction destructor called\n";
    }
    friend ostream& operator<<(ostream& o, const Fraction& f)
    {
        return (o << f.numer << '/' << f.denom);
    }
};

class InputError
{
    string stream;
public:
    InputError(string name) : stream(name) {}
    friend ostream& operator<<(ostream& out, const InputError& error);
};


ostream& operator<<(ostream& out, const InputError& error)
{
    out << "Error in " << error.stream << endl;
    return out;
}


ostream& operator<<(ostream& out, const ZeroDenominator& /*error*/)
{
    out << "ZeroDenominator Error"  << endl;
    return out;
}

int main()
{
    int i1, i2;
    cout << "Enter two ints => ";

    try
    {
        cin >> i1 >> i2;
        if (cin.fail()) throw InputError("cin");
        //   You could also use (!cin) instead of (cin.fail())
        //   cin.bad() did not detect error in cin
        Fraction f(i1,i2);
        cout << f << endl;   // Should this be in the try block?
    }
    catch (const InputError& error)
    {
        cerr << error << endl;
    }
    catch (const ZeroDenominator& errmsg)
    {
        cerr << errmsg << endl;
    }
```

```
    catch (...)
    {
        cerr << "help\n";
    }

    cout << "*** End of Program ***\n";
}
```

****** Sample Run 1 ******

```
Enter two ints => 2 3
Fraction constructor called
2/3
Fraction destructor called
*** End of Program ***
```

****** Sample Run 2 ******

```
Enter two ints => 2 three
Error in cin

*** End of Program ***
```

****** Sample Run 3 ******

```
Enter two ints 2 0
Fraction constructor called
ZeroDenominator Error

*** End of Program ***
```

Example 10 – Use a class to access different values that may be thrown

Another technique is to use a class to access different values that might be thrown.

```
#include <iostream>
#include <cctype>
#include <cfloat>  // for FLT_MAX
using namespace std;

class ErrorStuff
{
public:
    static const int BadInt;
    static const float BadFloat;
    static const char BadChar;

    ErrorStuff(int arg)
    : x(arg), y(BadFloat), z(BadChar)
    {
    }

    ErrorStuff(float arg)
    : x(BadInt), y(arg), z(BadChar)
```

```
    {
    }

    ErrorStuff(char arg)
    : x(BadInt), y(BadFloat), z(arg)
    {
    }

    int get_x() const
    {
        return x;
    }

    float get_y() const
    {
        return y;
    }

    char get_z() const
    {
        return z;
    }
private:
    int x;
    float y;
    char z;
};

const int ErrorStuff::BadInt = 0xffffffff;
const float ErrorStuff::BadFloat = FLT_MAX;
const char ErrorStuff::BadChar = 0;

int main()
{
    int i;
    float f;
    char c;

    try
    {
        cout << "Enter an even int, a positive float, and a alphabetic char => ";
        cin >> i >> f >> c;
        if (cin.fail())
            throw string{"cin"};
        if (i % 2)
            throw ErrorStuff(i);
        else if (f < 0)
            throw ErrorStuff(f);
        else if (!isalpha(c))
            throw ErrorStuff(c);
        else
            cout << "Thanks\n";
    }
    catch (const string& what)
```

```
    {
        if (what == "cin")
        {
            cerr << "*** Can't you type?\n";
            cin.clear();

        }
        else
        {
            cout << "whatever\n";
        }

    }
    catch (const ErrorStuff& e)
    {
        cout << "Hey!!!  ";
        if (e.get_x() != ErrorStuff::BadInt)
            cerr << "You entered an invalid int: " << e.get_x() << endl;
        else if (e.get_y() != ErrorStuff::BadFloat)
            cerr << "You entered an invalid float: " << e.get_y() << endl;
        else
            cerr << "You entered an invalid char: " << e.get_z() << endl;
    }

    cout << "*** End of Program ***\n";
}
```

****** Sample Run 1 ******

```
Enter an even int, a positive float, and a alphabetic char => 2 2.2 A
Thanks
*** End of Program ***
```

****** Sample Run 2 ******

```
Enter an even int, a positive float, and a alphabetic char => two 2.2 A
*** Can't you type?
*** End of Program ***
```

****** Sample Run 3 ******

```
Enter an even int, a positive float, and a alphabetic char => 3 2.2 A
Hey!!!  You entered an invalid int: 3
*** End of Program ***
```

****** Sample Run 4 ******

```
Enter an even int, a positive float, and a alphabetic char => 2 -2.2 A
Hey!!!  You entered an invalid float: -2.2
*** End of Program ***
```

****** Sample Run 5 ******

```
Enter an even int, a positive float, and a alphabetic char => 2 2.2 2
Hey!!!  You entered an invalid char: 2
*** End of Program ***
```

Catching Uncaught Exceptions with set_terminate()

You can name a function to execute using set_terminate() for any unhandled exceptions. The **set_terminate()** function will execute, then the program will abort.

The terminate function has a void argument and void return. By default, an unhandled exception will cause a call to the **terminate()** function, which will, in turn call the **abort()** function. This causes the program to end with a "Abnormal program termination error". The use of **set_terminate()** overrides this default behavior.

set_terminate() returns the previous function assigned.

An uncaught exception will terminate the program. **set_terminate**() cannot override this, so you should not attempt to continue processing by returning to the calling function or jumping to another location. This will result in undefined program behavior.

Further, the **set_terminate**() function, itself, had better not throw an exception!

Syntax

typedef void (*terminate_function)();
terminate_function **set_terminate**(terminate_function fn);

Both the **terminate**() and the **abort**() functions are C++ standard library functions.

Example 11 – set_terminate()

```
#include <iostream>
#include <exception>          // for set_terminate()
#include <string>
using namespace std;

void uncaught()
{
    cerr << "I wasn't able to catch an exception\n";
}

void funk(int i)
{
    try
```

```
    {
        switch (i)
        {
        case 1:
            throw(string("have a nice day"));
        case 2:
            throw(5);
        case 3:
            throw(3.14);
        }
    }
    catch(const string& it)
    {
        cout << "You threw me a string: " << it << endl;
    }
    catch(double it)
    {
        cout << "You threw me a double: " << it << endl;
    }
}

int main()
{
    set_terminate(uncaught);
    funk(1);
    funk(2);
    funk(3);
    cout << "End of program\n";
}
```

```
******  Output  ******

You threw me a const char*: Have a nice day
I wasn't able to catch an exception
Program Aborted
```

Exception Specifications

Dynamic exception specifications **are no longer supported** since C++17.

Examples

```
void funk1() throw (sometype); // Error: not allowed in C++17

void funk2() throw ();         // Error: not allowed in C++17

void funk2() noexcept;         // OK
```

set_unexpected()

The set_unepected() function was removed in C++17.

Example 14 - Re-throwing a throw

Sometimes a catch block is not meant to handle the current error. If this is the case, one option is to re-throw the current throw, so that it is handled by a prior catch block. To do this, just place a **throw;** without an throw-expression in the current catch block. Control is transferred to a higher level catch block. This is illustrated in the following example.

```
#include <iostream>
#include <string>

void funky(void)
{
    try
    {
        throw(std::string("This is a funky booboo"));
    }
    catch(...)
    {
        std::cout << "I don't know how to handle this\n";
        throw;
    }
}

int main()
{
    try
    {
        funky();
    }
    catch(const std::string& x)
    {
        std::cout << "Somebody threw me: " << x << std::endl;
    }
    std::cout << "*** End of Program ***\n";
}
```

```
******  Output  ******

I don't know how to handle this
Somebody threw me: This is a funky booboo
*** End of Program ***
```

Example 15 - Unwinding the stack

When an exception is thrown, destructors are automatically called for automatic objects that were constructed in the try-block. If the exception is thrown during the construction of an object, the destructor is not called for that object. For example, if an array of objects is being constructed when an exception is thrown, destructors will only be called for the array elements which were fully constructed. This process of calling of destructors for automatic objects after an exception is thrown is called **stack unwinding**.

```
#include <iostream>
```

```
#include <cstring>
using namespace std;

class Thing
{
    char* name;
public:
    Thing(const char* arg = nullptr);
    Thing(const Thing& t);         // copy ctor
    ~Thing();
    const char* get_name() const
    {
        return name;
    }
};

Thing::Thing(const char* arg)
    : name(new char[strlen(arg)+1])
{
    if (strcmp(arg,"Satan")==0)
        throw (this);
    else
        strcpy(name,arg);
    cout << ">>> " << name << " successfully constructed\n";
}

Thing::Thing(const Thing& arg) : name(new char[strlen(arg.name)+6])
{
    strcpy(name,arg.name);
    strcat(name, " Clone");
    cout << ">>> " << name << " successfully copy constructed\n";
}

Thing::~Thing()
{
    cout << "<<< destructor called for Thing " << name << endl;
    if (name)
        delete [] name;
    name = nullptr;
}

int main()
{
    Thing* pThing;
    try
    {
        Thing aFriend("Sam");
        Thing aFriendClone(aFriend);
        cout << endl;

        pThing = new Thing("Sarah");
        delete pThing;
        pThing = nullptr;
        cout << endl;

```

```
        Thing satan("Satan");
        Thing harry("Harry");
    }
    catch(const Thing* ptr)
    {
        cerr << "I caught an evil Thing" << endl;
        delete [] ptr->get_name();
    }
    if (pThing) delete pThing;
    cerr << "*** End of Program ***\n";
}

```

****** Output ******

```
>>> Sam successfully constructed
>>> Sam Clone successfully copy constructed

>>> Sarah successfully constructed
<<< destructor called for Thing Sarah

<<< destructor called for Thing Sam Clone
<<< destructor called for Thing Sam
I caught an evil Thing
<<< destructor called for Thing *** End of Program ***
```

Example 16 - Standard Exceptions

```
#include <iostream>
#include <string>
#include <exception>
#include <new>              // for bad_alloc
#include <typeinfo>        // for bad_cast
#include <stdexcept>
using namespace std;

class Base
{
public:
    virtual void funk() {}
    virtual ~Base() {}
};

class Derived : public Base
{
public:
    void funk() {}
};


int main()
{
    // test bad_alloc
    try
```

```
    {
        while (1)
        {
            cout << "Can I have some memory?\n";
            new char[0x7fffffff];
        }
    }
    catch(const bad_alloc& error)
    {
        cerr << "*** I caught a " << error.what() << endl << endl;
    }

    // test bad_cast
    try
    {
        Base      baseObject;
        // try to cast a base object to a derived object
        Derived& ref2Derived = dynamic_cast<Derived&>(baseObject);
    }
    catch(const bad_cast& error)
    {
        cerr << "!!! I caught a " << error.what() << endl << endl;
    }

    // test out_of_range error
    try
    {
        string S = "Hey";
        cout << "S.at(2)=" << S.at(2) << endl;
        cout << "S.at(5)=" << S.at(5) << endl;  // string throws an out_of_range error
    }
    catch (const out_of_range& error)
    {
        cout << "$$$ I caught a " << error.what() << endl << endl;
    }

    cout << "*** End of Program ***\n";
}
```

****** Output ******

```
Can I have some memory?
*** I caught a std::bad_alloc

!!! I caught a std::bad_cast

S.at(2)=y
$$$ I caught a basic_string::at: __n (which is 5) >= this->size() (which is
3)

**** End of Program ***
```

Example 17 - Derive your own exceptions from standard exceptions

```
#include <exception>
#include <stdexcept>
#include <iostream>
#include <cmath>          // for sqrt()
#include <cstring>
#include <cstdlib>
#include <sstream>        // for istreamstream/ostringstream
#include <climits>        // for SHRT_MAX
#include <typeinfo>       // for typeid operator
using namespace std;


ostream& operator<<(ostream& out, const exception& error)
{
    out << "I caught an error of type: " << typeid(error).name()
        << "\nMessage: " << error.what() << endl;
    return out;
}

class my_domain_error : public domain_error
{
public:
    my_domain_error(const char* message) : domain_error(message)
    {}

    // override the virtual what() function
    const char* what() const noexcept override
    {
        static char temp[128];
        strcpy(temp,"my_domain_error: ");
        strcat(temp,domain_error::what());
        return temp;
    }
};

double mysqrt1(double number) throw (domain_error)
{
    if (number < 0)
        throw domain_error("mysqrt1 error: negative argument");
    return sqrt(number);
}

double mysqrt2(double number) throw (my_domain_error)
{
    if (number < 0)
        throw my_domain_error("mysqrt2 error: negative argument");
    return sqrt(number);
}

// Derive the zero_denominator class from invalid_argument
class zero_denominator : public invalid_argument
{
```

```
public:
    zero_denominator()
        : invalid_argument("Error: zero denominator")
    { }
};

class fraction
{
    int numerator, denominator;
public:
    fraction(int n = 0, int d = 1) : numerator(n), denominator(d)
    {
        if (d == 0 )
            throw zero_denominator();
    }
};

// convert a hexadecimal string to unsigned int
unsigned
hex_string_to_unsigned(const string& text) throw (invalid_argument)
{
    if (text.find_first_not_of("0123456789abcdefABCDEF") != string::npos)
    {
        throw invalid_argument(string("Invalid hexadecimal char in: " ) + text);
    }
    istringstream sin(text);
    unsigned number;
    sin >> hex >> number;
    return number;
}

// returns sum of two shorts, make sure sum is valid short
short
add2shorts(short one, short two, bool check_limit = false) throw (overflow_error)
{
    if (check_limit)
    {
        if (static_cast<int>(one) + two > SHRT_MAX)      // SHRT_MAX = 32767
        {
            ostringstream sout;
            sout << "add2shorts failed with arguments " << one << " and " << two;
            throw overflow_error(sout.str());
        }
    }
    return one + two;
}


int main()
{
```

```
    // test throw/catch of domain_error
    try
    {
        cout << "mysqrt1(2.0)=" << mysqrt1(2.0) << endl;
        cout << "mysqrt1(-2.0)=" << mysqrt1(-2.0) << endl;
    }
    catch (const domain_error& error)
    {
        cerr << "Line " << __LINE__ << ": " << error << endl;
    }

    // test throw/catch of logic_error
    try
    {
        cout << "mysqrt1(-2.0)=" << mysqrt1(-2.0) << endl;
    }
    catch (const logic_error& error)
    {
        cerr << "Line " << __LINE__ << ": " << error << endl;
    }

    // test throw/catch of (base class) exception
    try
    {
        cout << "mysqrt1(-2.0)=" << mysqrt1(-2.0) << endl;
    }
    catch (const exception& error)
    {
        cerr << "Line " << __LINE__ << ": " << error << endl;
    }

    // test throw/catch of my_domain_error
    try
    {
        cout << "mysqrt2(-2.0)=" << mysqrt2(-2.0) << endl;
    }
    catch (const my_domain_error& error)
    {
        cerr << "Line " << __LINE__ << ": " << error << endl;
    }

    // test throw/catch of zero_denominator
    try
    {
        fraction F(2,0);
    }
    catch (const zero_denominator& error)
    {
        cerr << "Line " << __LINE__ << ": " << error << endl;
    }

    // test throw/catch of invalid_argument
    try
    {
```

```
157         cout << "hex abc=" << hex_string_to_unsigned(string("abc")) << endl;
158         cout << "hex abz=" << hex_string_to_unsigned(string("abz")) << endl;
159     }
160     catch (const invalid_argument& error)
161     {
162         cerr << "Line " << __LINE__ << ": " << error << endl;
163     }
164
165     // test throw/catch of overflow_error
166     try
167     {
168         cout << "short 31000+32000=" << add2shorts(31000,32000) << endl;
169         cout << "short 31000+32000=" << add2shorts(31000,32000,true) << endl;
170     }
171     catch (const overflow_error& error)
172     {
173         cerr << "Line " << __LINE__ << ": " << error << endl;
174     }
175 }
```

****** Output ******

```
mysqrt1(2.0)=1.41421
Line 111: I caught an error of type: St12domain_error
Message: mysqrt1 error: negative argument

Line 121: I caught an error of type: St12domain_error
Message: mysqrt1 error: negative argument

Line 131: I caught an error of type: St12domain_error
Message: mysqrt1 error: negative argument

Line 141: I caught an error of type: 15my_domain_error
Message: my_domain_error: mysqrt2 error: negative argument

Line 151: I caught an error of type: 16zero_denominator
Message: Error: zero denominator

hex abc=2748
Line 162: I caught an error of type: St16invalid_argument
Message: Invalid hexadecimal char in: abz

short 31000+32000=-2536
Line 173: I caught an error of type: St14overflow_error
Message: add2shorts failed with arguments 31000 and 32000
```

Namespaces

A namespace is a group of types, variables, or objects. This grouping may be used to avoid name clashes. In other words, by using namespaces, an application may reuse a type name or variable name without an ambiguity conflict.

The keyword, namespace, is used to create a namespace and to reference an existing namespace name.

Namespace usage make use of the using directive and the using declaration. A using directive,

is used to qualify all unqualified symbol names of a namespace, such as

```
using namespace std;
```

allows you to write

```
cout << whatever << endl;
```

instead of

```
std::cout << whatever << std::endl;
```

A using declaration allows you to refer to a symbol name without qualifying the entire namespace. For example:

```
using std::cout;
…
cout << whatever << std::end;
```

Example 1 – Create a namespace

```
#include <iostream>
#include <cmath>
#include <cstring>
#include <cstdlib>
#include <cctype>
using namespace std;

// Create a namespace
namespace mystuff
 {
     int cout = 5;
     double sqrt(double x)
     {
         return x / 2.0;
     }
 }

 int main()
```

```
{
    char cout[32] = "This is a bad idea";
    char temp[80];
    std::cout << "hey\n";
    std::cout << "the square root of 2 is " << sqrt(2.) << endl;
    strcpy(temp,"hello");
    strcat(temp," there");
    std::cout << strlen(temp) << temp << endl;
    std::cout << atoi("4") << endl;
    std::cout << toupper('a') << endl;
    std::cout << static_cast<char>(toupper('a')) << endl;

    std::cout << mystuff::cout << ' ' << cout << endl;

    std::cout << sqrt(5.75) << ' ' << mystuff::sqrt(5.75) << endl;
}
```

****** Program Output ******

```
hey
the square root of 2 is 1.41421
11hello there
4
65
A
5 This is a bad idea
2.39792 2.875
```

Example 2 – namespace scope

Note that symbols default to their local definitions first, then to std definitions.

```
#include <iostream>

namespace test
{
    int I = 9;
}

void funk1();
void funk2();
void funk3();

int main()
{
    funk1();
    funk2();
    funk3();
}

void funk1()
{
```

```
    std::cout << test::I << std::endl;  // This is OK
    // std::cout << I << std::endl;  // Compile error
    using namespace test;
    std::cout << I << std::endl;  // OK, now
}

void funk2()
{
    std::cout << test::I << std::endl;  // This is
    // std::cout << I << std::endl;  // Compile error
}

using namespace test;

void funk3()
{
    std::cout << I << std::endl;  // OK, now
}
```

****** Output ******

```
9
9
9
9
```

Example 3 - namespaces and multiple files

This example illustrates the use of namespace in multiple files.

```
// File: node.h

#ifndef NODE_H
#define NODE_H

#include <iostream>

namespace joelinkedlist
{

 class Node
 {
     int data;
     Node*    next;
 public:
     Node(int d,Node* n);
     int get_data() const;
     Node* get_next() const;
     void set_next(Node* ptr);
 };

 std::ostream& operator<<(std::ostream&, const Node&);

```

```
}

#endif
```

```
// File: node.cpp

#include "node.h"
#include <iostream>
using namespace std;

joelinkedlist::Node::Node(int d, Node* n)
: data(d), next(n)
{
 }

 int joelinkedlist::Node::get_data() const
 {
     return data;
 }

 using namespace joelinkedlist;

 Node* Node::get_next() const
 {
     return next;
 }

 void Node::set_next(Node* ptr)
 {
     next = ptr;
 }

 namespace joelinkedlist
 {
     ostream& operator<<(ostream& out, const Node& obj)
     {
         out << obj.get_data();
         return out;
     }
 }
```

```
// File: list.h

#ifndef LIST_H
#define LIST_H

#include "node.h"
#include <iostream>

namespace joelinkedlist
{
    class List
```

```
    {
        Node* top;
    public:
        List();
        ~List();
        void push(int item);
        int pop();
        Node* get_top() const;
        bool remove(int item);
        Node* find(int item) const;
        bool remove_last();
    };

    std::ostream& operator<<(std::ostream&, const List&);

}

#endif
```

```
// File: list.cpp

#include <iostream>
#include <cstdlib>
using namespace std;

#include "list.h"
using joelinkedlist::List;
using joelinkedlist::Node;

List::List() : top(0)
{ }

List::~List()
{
    Node* temp = top;
    while (temp != nullptr) {
        top = top -> get_next();
        delete temp;
        temp = top;
    }
}

void List::push(int item)
{
    Node* temp = new Node(item, top);
    top = temp;
}

int List::pop()
{
    Node* temp = top;
    top = top->get_next();
    int value = temp->get_data();
```

```
    delete temp;
    return value;
}

Node* List::get_top() const
{
    return top;
}

Node* List::find(int item) const
{
    Node* temp = top;
    while (temp != 0) {
        if (temp->get_data() == item) return temp;
        temp = temp -> get_next();
    }
    return 0;
}

bool List::remove(int item)
{
    if (!find(item)) {
        cerr << item << " is not in the List\n";
        return false;
    }
    Node* temp1 = top;
    Node* temp2;
    if (top->get_data() == item) {
        top = top -> get_next();
        delete temp1;
        return true;
    }
    while (temp1->get_next()->get_data() != item) {
        temp1 = temp1 -> get_next();
    }
    temp2 = temp1 -> get_next();
    temp1->set_next(temp2->get_next());
    delete temp2;
    return true;
}

namespace joelinkedlist
{
    ostream& operator<<(ostream& out, const List& object)
    {
        Node* temp = object.get_top();
        while (temp != 0) {
            out << *temp << ' ';
            temp = temp -> get_next();
        }
        return out;
    }
}
```

```cpp
// File: main.cpp

#include <iostream>
using namespace std;

#include "list.h"
using joelinkedlist::List;

int main()
 {
     List L;
     L.push(2);
     L.push(4);
     L.push(6);
     L.push(8);
     L.push(10);
     cout << L << endl;

     cout << "top value is " << L.get_top()->get_data() << endl;

     if (L.find(2)) cout << 2 << " is in the list\n";
     if (L.find(5)) cout << 5 << " is in the list\n";
     if (L.find(6)) cout << 6 << " is in the list\n";
     if (L.find(10)) cout << 10 << " is in the list\n";

     cout << L.pop() << " removed from the list\n";
     cout << L << endl;

     L.remove(3);
     L.remove(6);
     cout << L << endl;

     L.remove(2);
     L.remove(8);
     cout << L << endl;
 }
```

Libraries

Libraries are used to isolate common code that may be used by different applications. By designing and using a library, you do not have to “reinvent the wheel”. You simply “invent the wheel” one time and then you “link it in” to your current application whenever you need it. As part of this process, you also have to tell your current application what the wheel “looks like”. This is typically accomplished by including a heading file.

The use of libraries mandates that the associated libraries files be logically organized in directories that are easily identified and accessed.

Creating a Library

- The library files will usually consist of one or more source files and one or more header files.
- The source files and header files may be located in separate directories. The source file(s) may contain one or (usually) more functions.
- There is no main() function that is usually present in any C++ application.
- Each library source code file is compiled into its own object file.
- The object file(s) are combined together into a library file, sometimes called an archive.
- A library typically contains functions, variables, constants, and types.
- In general, a libraries source file will contain definitions (function definitions and variable definitions). A libraries header file will contain declarations (function prototypes, class declarations, and declarations of other types).

Using a Library

- An application that uses a library must include the libraries header file(s) in order to “see” the libraries declarations. That is required for compilation of the application. When the application file is compiled, it must identify to the compiler the location of the included header file.
- Then the application must “link in” the library. In the “link” step of the application, the location of the library file (or archive) must be identified to the “linker”.

Types of Linking

There are two basic types of linking performed by an application – static and dynamic linking. With static linking the necessary (or referenced) code is inserted into the final executable and becomes part of that binary file. With dynamic linking, the referenced code is not directly inserted into the final executable. The dynamic library “sits out on disk” and the necessary parts are included or accessed as needed during run-time. Applications that use dynamic linking are usually smaller than those that use static linking. Dynamically linking applications will usually run slower than the equivalent statically linked applications, since the dynamically linked library must be loaded into memory at run-time.

Examples

Example 1 – a factorial library

The following example demonstrates a library that is used to calculate factorial. This example makes use of 3 files:

1. A library header file that contains a function prototype
2. A library source file containing the factorial function definition. This file will be compiled and the resulting function will be placed in a library.
3. A test source file containing calls to the factorial function.

Library header file

```
// File: factorial.h

#ifndef FACTORIAL_H
#define FACTORIAL_H

long factorial(long arg);

#endif
```

Library source file

```
// File: factorial.cpp

long factorial(long arg)
{
   long total = 1;
   for (long num = 2; num <= arg; num++)
         total *= num;
   return total;
}
```

Test source file

```
// File: factorial_test.cpp

#include <iostream>
using namespace std;
#include "factorial.h"

int main()
{
    cout << factorial(2) << endl;
     cout << factorial(4) << endl;
     cout << factorial(6) << endl;
     cout << factorial(8) << endl;
     cout << factorial(10) << endl;
}
```

******** Output ********

```
2
24
720
40320
3628800
```

The Process

1. The header file and library source files are first created and compiled as a library (static or dynamic). It is important to give the resulting library an appropriate name and place it in a logical location, probably with other libraries.

2. The test source file must include the library header file for compilation. This means that you must tell the compiler where to find that header file.

3. To link the test application you must "link in" the library. That means telling the compiler where to find the library and what its name is.

Example 2 – a fraction library

This example illustration implementation of a fraction library.

fraction library header file

```
// File: fraction.h

#ifndef FRACTION_H
#define FRACTION_H

class fraction
{
    int numer, denom;
public:
     fraction(int = 0, int = 1);
     void operator!(void) const;          // print the fraction
     fraction& operator~(void);           // reduce the fraction
     fraction operator-(void) const;      // negative of fraction
     fraction operator*(void) const;      // reciprocal of fraction
     fraction& operator+=(const fraction&);
     fraction& operator-=(const fraction&);
     fraction& operator*=(const fraction&);
     fraction& operator/=(const fraction&);
     fraction operator+(int) const;
     fraction operator-(int) const;
     fraction operator*(int) const;
     fraction operator/(int) const;
     int operator>(const fraction&) const;
     int operator<(const fraction&) const;
     int operator>=(const fraction&) const;
     int operator<=(const fraction&) const;
     int operator==(const fraction&) const;
```

```
    int operator!=(const fraction&) const;
    fraction operator+(const fraction&) const;
    fraction operator-(const fraction&) const;
    fraction operator*(const fraction&) const;
    fraction operator/(const fraction&) const;
    fraction& operator++();    // prefix operator returns by ref
    fraction operator++(int); // postfix operator returns by value
};

#endif
```

fraction library source file

```
// File: fraction.cpp

#include "fraction.h"
#include <iostream>

using namespace std;

// member function definitions
fraction::fraction(int n, int d)
{
//  assert(d != 0);
    numer = n;
    denom = d;
}

void fraction::operator!(void) const
{
    cout << numer << '/' << denom << endl;
}

fraction& fraction::operator~(void)
{
    int min;
    // find the minimum of the denom and numer
    min = denom < numer ? denom : numer;
    for (int i = 2; i <= min; i++)
    {
        while ((numer % i == 0) && (denom % i == 0))
        {
            numer /= i;
            denom /= i;
        }
    }
    return *this;
}

fraction fraction::operator-(void) const
{
    return fraction(-numer,denom);
}

fraction fraction::operator*(void) const
```

```
{
    return fraction(denom,numer);
}

fraction& fraction::operator+=(const fraction& f)
{
    numer = numer*f.denom+denom*f.numer;
    denom = denom*f.denom;
    return *this;
}

fraction& fraction::operator-=(const fraction& f)
{
    *this += (-f);
    return *this;
}

fraction& fraction::operator*=(const fraction& f)
{
    numer = numer*f.numer;
    denom = denom*f.denom;
    return *this;
}

fraction& fraction::operator/=(const fraction& f)
{
    *this *= (*f);
    return *this;
}

int fraction::operator>(const fraction& f) const
{
    return (float) numer/denom > (float) f.numer/f.denom;
}

int fraction::operator<(const fraction& f) const
{
    return f>*this;
}

int fraction::operator==(const fraction& f) const
{
    return numer*f.denom == denom*f.numer;
}

int fraction::operator!=(const fraction& f) const
{
    return !(*this == f);
}

int fraction::operator<=(const fraction& f) const
{
    return !(*this > f);
}

```

```
int fraction::operator>=(const fraction& f) const
{
    return !(*this<f);
}

fraction fraction::operator+(const fraction& f) const
{
    return fraction(numer*f.denom+denom*f.numer,denom*f.denom);
}

fraction fraction::operator-(const fraction& f) const
{
    return fraction(numer*f.denom-denom*f.numer,denom*f.denom);
}

fraction fraction::operator*(const fraction& f) const
{
    return fraction(numer*f.numer,denom*f.denom);
}

fraction fraction::operator/(const fraction& f) const
{
    return (*this) * (*f);
}

fraction fraction::operator+(int i) const
{
    return fraction(numer+i*denom,denom);
}

fraction fraction::operator-(int i) const
{
    return (*this) + -i;
}

fraction fraction::operator*(int i) const
{
    return fraction(numer*i,denom);
}

fraction fraction::operator/(int i) const
{
    return fraction(numer,i*denom);
}

// prefix increment operator
fraction& fraction::operator++()
{
    numer += denom;
    return *this;
}

// postfix increment operator
fraction fraction::operator++(int)     // Note dummy int argument
{
```

```
    fraction temp(*this);
    ++*this;                          // call the prefix operator
    return temp;
}
```

fraction library test

```
// File: fraction_main.cpp

#include "fraction.h"
#include <iostream>
using namespace std;

int main(void)
{
    fraction f(3,4);            // initialize fraction f & g
     fraction g(1,2);
     cout << "!f ";
     !f;
     cout << "!g ";
     !g;
     cout << endl;
     cout << "-g ";
     !-g;
     cout << "*g ";
     !*g;
     fraction h = g + f;
     cout << endl;
     cout << "h=g+f " << " !h ";
     !h;
     cout << "!~h ";
     !~h;
     cout << endl;
     cout << "f+g ";
     ! (f + g);
     cout << "f-g ";
     ! (f - g);
     cout << "f*g ";
     ! (f * g);
     cout << "f/g ";
     ! (f / g);
     cout << endl;
     cout << "f+=g ";
     !~(f+=g);
     cout << "f-=g ";
     !~(f-=g);
     cout << "f*=g ";
     !~(f*=g);
     cout << "f/=g ";
     !~(f/=g);
     cout << endl;
     cout << "f<g " << (f<g) << endl;
     cout << "f>g " << (f>g) << endl;
     cout << "f==g " << (f==g) << endl;
     cout << "f!=g " << (f!=g) << endl;
```

```
    cout << "f<=g " << (f<=g) << endl;
    cout << "f>=g " << (f>=g) << endl;
    cout << endl;
    cout << "f+5 ";
    !(f+5);
    cout << "f-5 ";
    !(f-5);
    cout << "f*5 ";
    !(f*5);
    cout << "f/5 ";
    !(f/5);
    cout << endl;
    cout << "f+=5 ";
    f+=5;
    cout << "!~f ";
    !~f;  // How does this work?
    cout << "++f ";
    !++f;
    cout << "f=";
    !f;
    cout << "f++ ";
    !f++;
    cout << "f=";
    !f;
}
```

****** Output ******

```
!f 3/4
!g 1/2

-g -1/2
*g 2/1

h=g+f  !h 10/8
!~h 5/4

f+g 10/8
f-g 2/8
f*g 3/8
f/g 6/4

f+=g 5/4
f-=g 3/4
f*=g 3/8
f/=g 3/4

f<g 0
f>g 1
f==g 0
f!=g 1
f<=g 0
f>=g 1

f+5 23/4
f-5 -17/4
f*5 15/4
```

```
f/5 3/20

f+=5 !~f 23/4
++f 27/4
f=27/4
f++ 27/4
f=31/4
```

Linux compilation

These Linux commands are meant to demonstrate the compilation process.

1) `g++ -Wall -c fraction.cpp`
2) `ar r libfraction.a fraction.o`
3) `g++ -Wall fraction_main.cpp -L. -lfraction -o fraction_test`
4) ls

   ****** Output ******

   ```
   fraction.cpp  fraction.o          fraction_test
   fraction.h    fraction_main.cpp  libfraction.a
   ```

Explanation

Assumption: all files are located in the same directory for this example.

1) The fraction.cpp source file is compiled. The result is an object file, fraction.o. Note, the compiler finds the fraction.h header file in the same directory as the fraction.cpp file.
2) The fraction.o object file is placed in (archived) the library file, libfraction.a.
3) The fraction_main.cpp test file is compiled. The include directory is assumed to be the current directory. The library directory is also the current directory (that's the -L.). The library to *link in* is libfraction.a (that's the -lfraction). The output binary is fraction_test.
4) The ls command lists the 6 files related to this example.

fraction.h – fraction header
fraction.cpp – fraction source
fraction.o – fraction object
libfraction.a – fraction library
fraction_main.cpp – fraction test source
fraction_test – fraction test binary

Example 3 – a linked list library

This example illustration implementation of a linked list library.

Node class header file

```
// File: node.h

#ifndef NODE_H
#define NODE_H

#include <iostream>

class Node
{
     int data;
     Node*    next;
 public:
     Node(int d,Node* n);
     int get_data() const;
     Node* get_next() const;
     void set_next(Node* ptr);
 };

 std::ostream& operator<<(std::ostream&, const Node&);

 #endif
```

Node class source file

```
// File: node.cpp

#include "node.h"
#include <iostream>
using namespace std;

Node::Node(int d,Node* n)
    : data(d), next(n)
{ }

 int Node::get_data() const
 {
     return data;
 }

 Node* Node::get_next() const
 {
     return next;
 }

 void Node::set_next(Node* ptr)
 {
     next = ptr;
 }

 ostream& operator<<(ostream& out, const Node& obj)
 {
     out << obj.get_data();
     return out;
 }
```

List class header file

```
// File: list.h

#ifndef LIST_H
#define LIST_H

#include "node.h"
#include <iostream>

class List
{
    Node*    top;
public:
    List();
    ~List();
    void push(int item);
    int pop();
    Node* get_top() const;
    bool remove(int item);
    Node*    find(int item) const;
    bool remove_last();
};

std::ostream& operator<<(std::ostream&, const List&);

#endif
```

List class source file

```
// File: list.cpp

#include <iostream>
#include <cstdlib>
using namespace std;

#include "list.h"

List::List() : top(0)
{ }

List::~List()
{
    Node* temp = top;
    while (temp != nullptr)
    {
        top = top -> get_next();
        delete temp;
        temp = top;
    }
}

void List::push(int item)
{
    Node* temp = new Node(item,top);
    top = temp;
}
```

```
int List::pop()
{
    Node* temp = top;
    top = top->get_next();
    int value = temp->get_data();
    delete temp;
    return value;
}

Node* List::get_top() const
{
    return top;
}

Node*  List::find(int item) const
{
    Node* temp = top;
    while (temp != 0)
    {
        if (temp->get_data() == item) return temp;
        temp = temp -> get_next();
    }
    return 0;
}

bool List::remove(int item)
{
    if (!find(item))
    {
        cerr << item << " is not in the List\n";
        return false;
    }
    Node* temp1 = top;
    Node* temp2;
    if (top->get_data() == item)
    {
        top = top -> get_next();
        delete temp1;
        return true;
    }
    while (temp1->get_next()->get_data() != item)
    {
        temp1 = temp1 -> get_next();
    }
    temp2 = temp1 -> get_next();
    temp1->set_next(temp2->get_next());
    delete temp2;
    return true;
}

ostream& operator<<(ostream& out, const List& object)
{
    Node* temp = object.get_top();
    while (temp != 0)
```

```
    {
        out << *temp << ' ';
        temp = temp -> get_next();
    }
    return out;
}
```

Library test file

```
File: main.cpp

#include <iostream>
using namespace std;

#include "list.h"

int main (void)
{
    List L;
    L.push(2);
    L.push(4);
    L.push(6);
    L.push(8);
    L.push(10);
    cout << L << endl;

    cout << "top value is " << L.get_top()->get_data() << endl;

    if (L.find(2)) cout << 2 << " is in the list\n";
    if (L.find(5)) cout << 5 << " is in the list\n";
    if (L.find(6)) cout << 6 << " is in the list\n";
    if (L.find(10)) cout << 10 << " is in the list\n";

    cout << L.pop() << " removed from the list\n";
    cout << L << endl;

    L.remove(3);
    L.remove(6);
    cout << L << endl;

    L.remove(2);
    L.remove(8);
    cout << L << endl;
}
```

****** Output ******

```
10 8 6 4 2
top value is 10
2 is in the list
6 is in the list
10 is in the list
10 removed from the list
8 6 4 2
3 is not in the List
8 4 2
```

Linux compilation

These Linux commands are meant to demonstrate the compilation process.

```
1) g++ *.cpp -Wall -c -I.
2) ar r liblinked_list.a *.o

3) g++ main.cpp -Wall -I. -L. -llinked_list -o
   linked_list_test

4) ls
```

****** Output ******

```
liblinked_list.a  list.cpp  list.o    main.o    node.h
linked_list_test  list.h    main.cpp  node.cpp  node.o
```

Explanation

Assumption: all files are located in the same directory for this example.

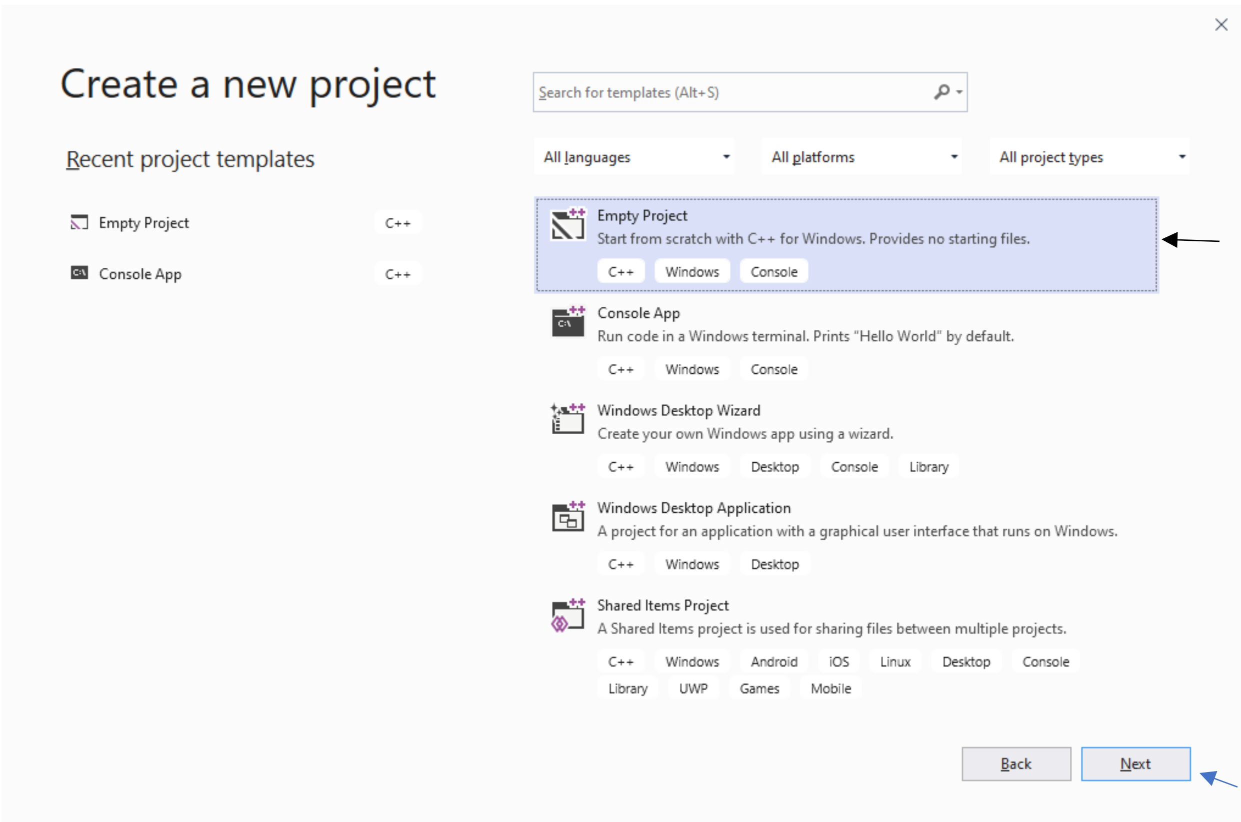
1) The two source files (node.cpp and list.cpp) are compiled. The result is two object files (node.o and list.o). The -c option means to compile only, not produce an executable file. The -I. option means to look in the current directory for include files.
2) Archive all object files into the library file, liblinked_list.a.
3) Compile the test file, main.cpp. Identify the current directory as an include directory. Identify the current directory as a link directory. Link in the library, liblinked_list.a. Name the output file, linked_list_test.
4) The ls command lists the 10 files related to this example.

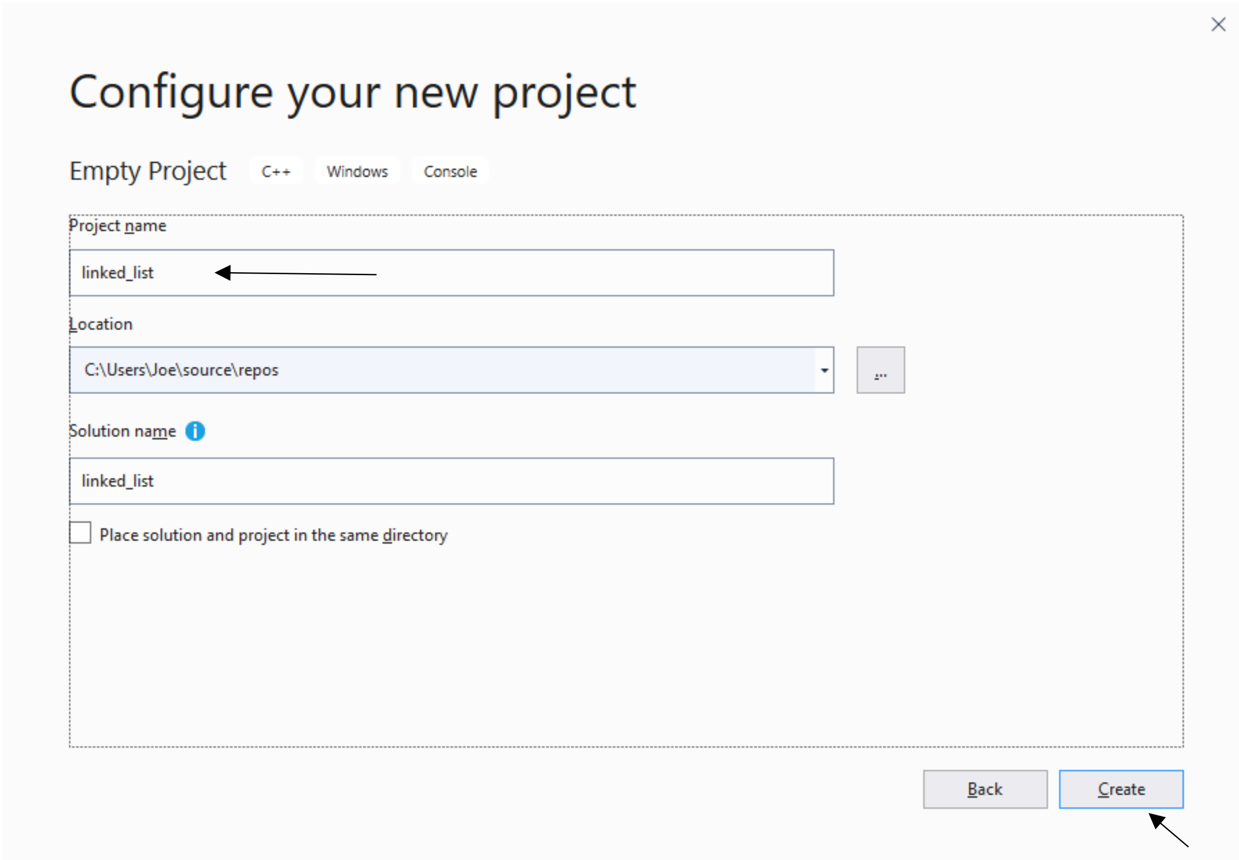
Example 4 - Create a Static Library Using MS Visual Studio 2019

The following example demonstrates building and using a library with Microsoft Visual Studio 2019. In this example, the same files will be used to create the linked list library and to use it. For simplicity, the same directory is used for the source files, header files, the library file, and the application binary.

Create a new project.

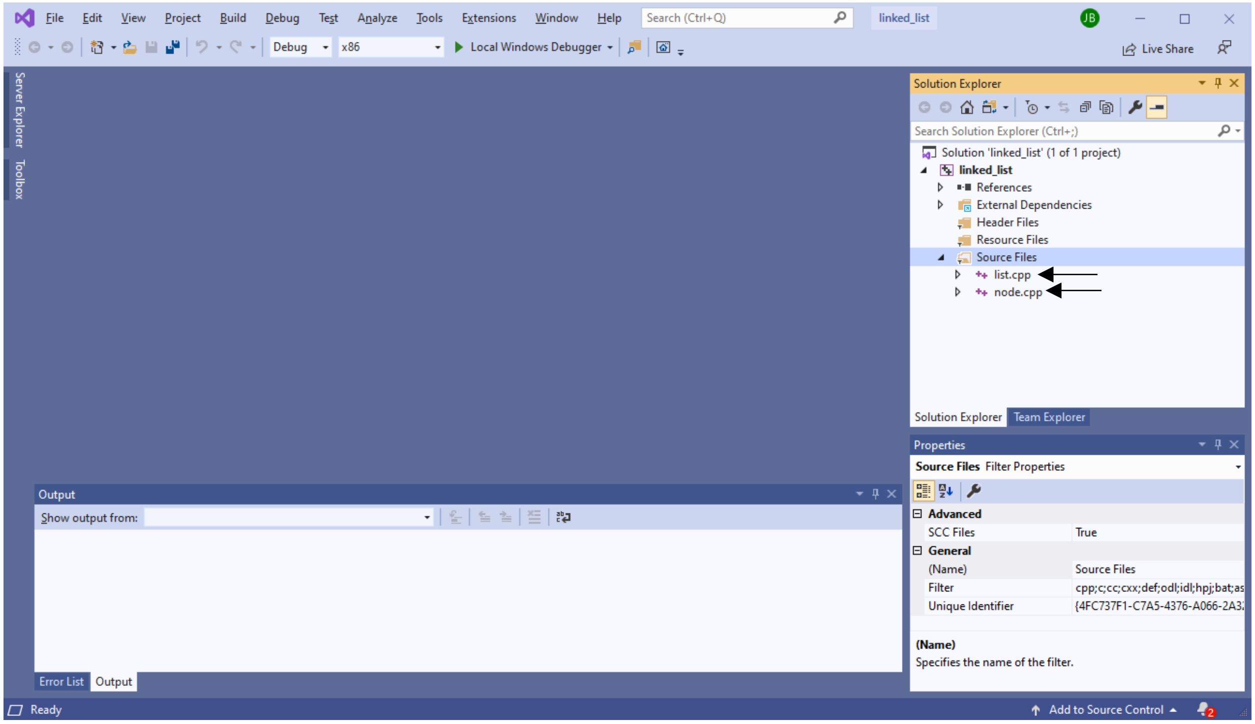
Choose **Empty Project**.




Configure your new project
Empty Project
C++
Windows
Console
Project name
linked_list
Location
C:\Users\Joe\source\repos
...
Solution name
linked_list
Place solution and project in the same directory
Back
Create

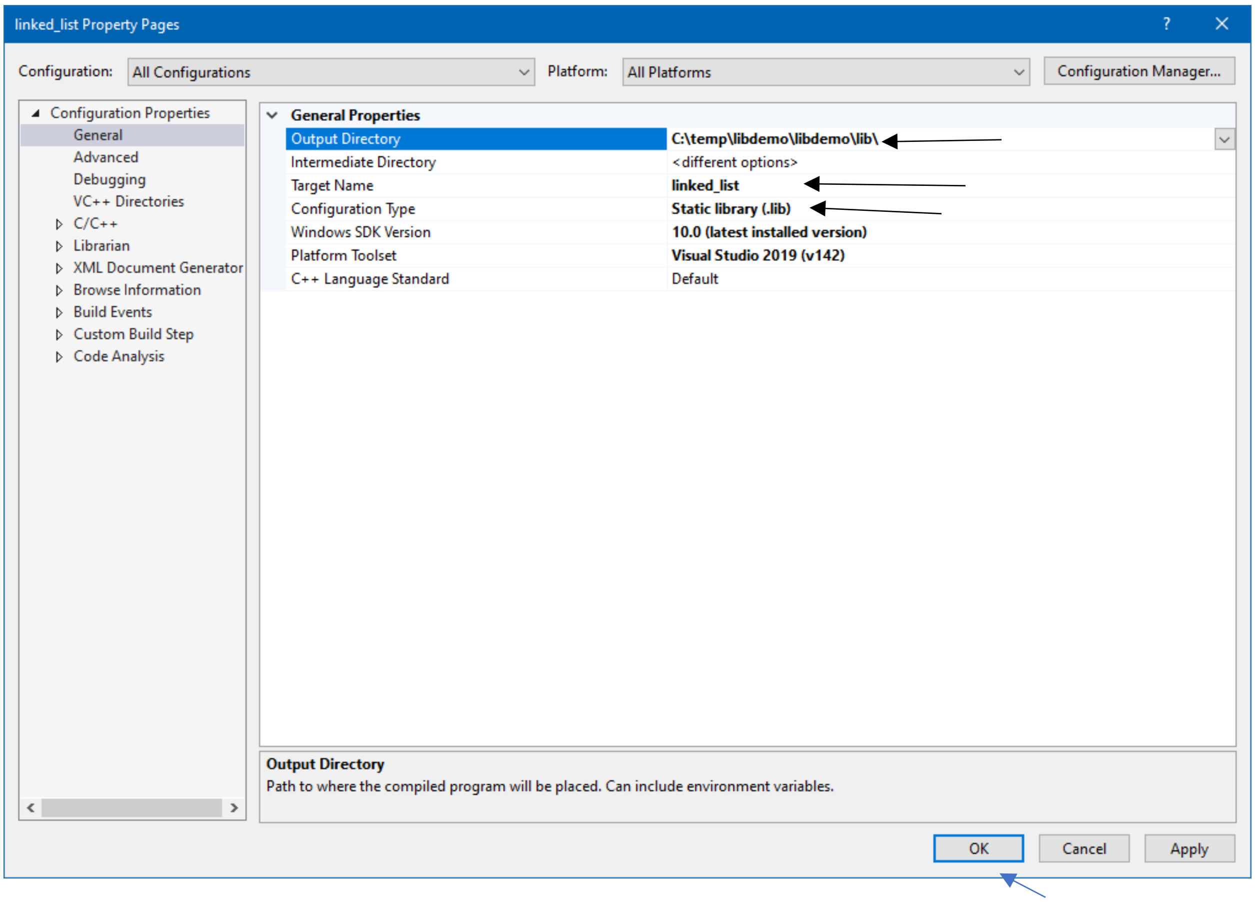
Add the source files for the library

Use a right-mouse click under Source Files in the Solution Explorer.



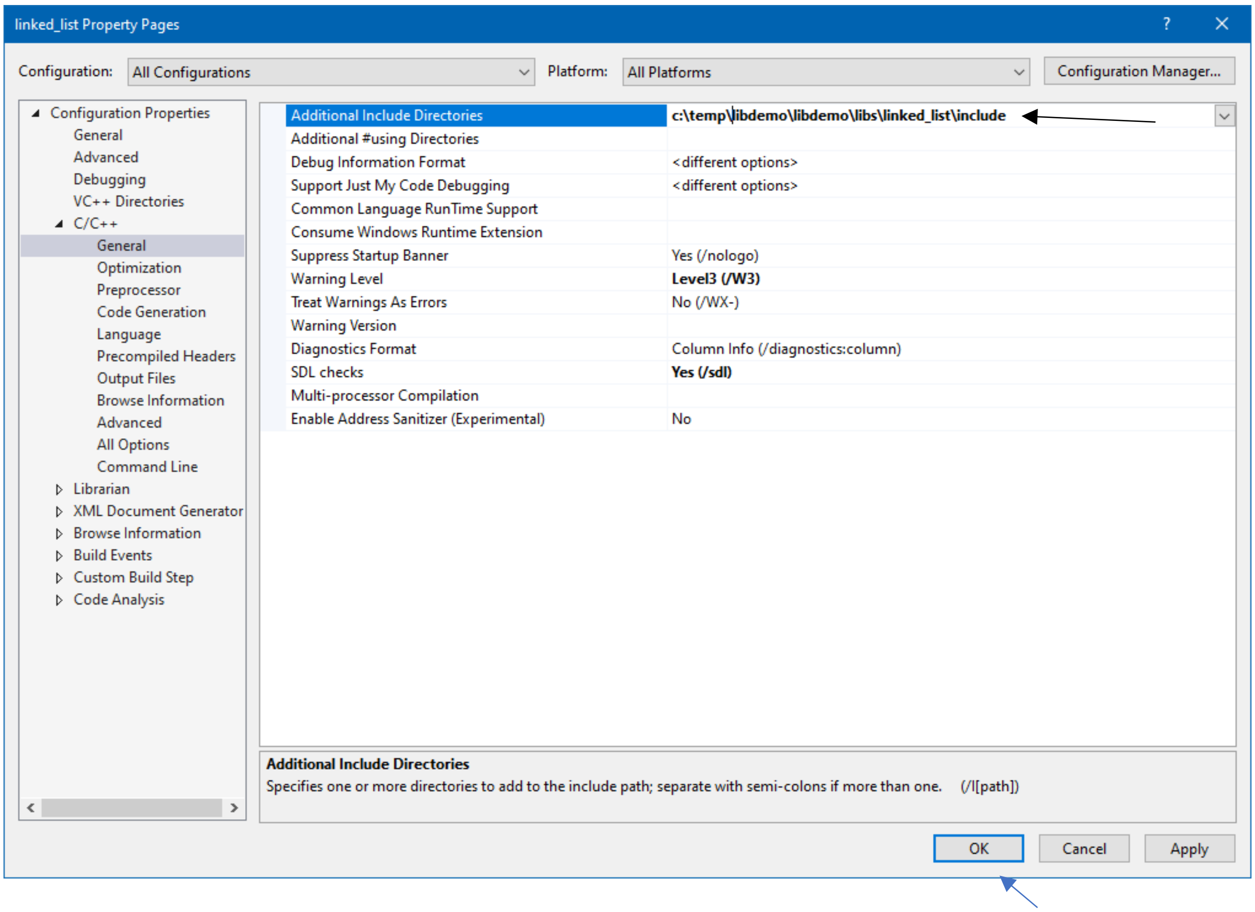
Set the project configuration properties

- Right-mouse click on the project name (linked_list) and select Properties.
- In the Property Pages
    - Enter the name of the Output Directory. End directory path with a \
    - Enter the Target Name (it will default to the project name)
    - Change the Configuration Type to Static library (.lib)



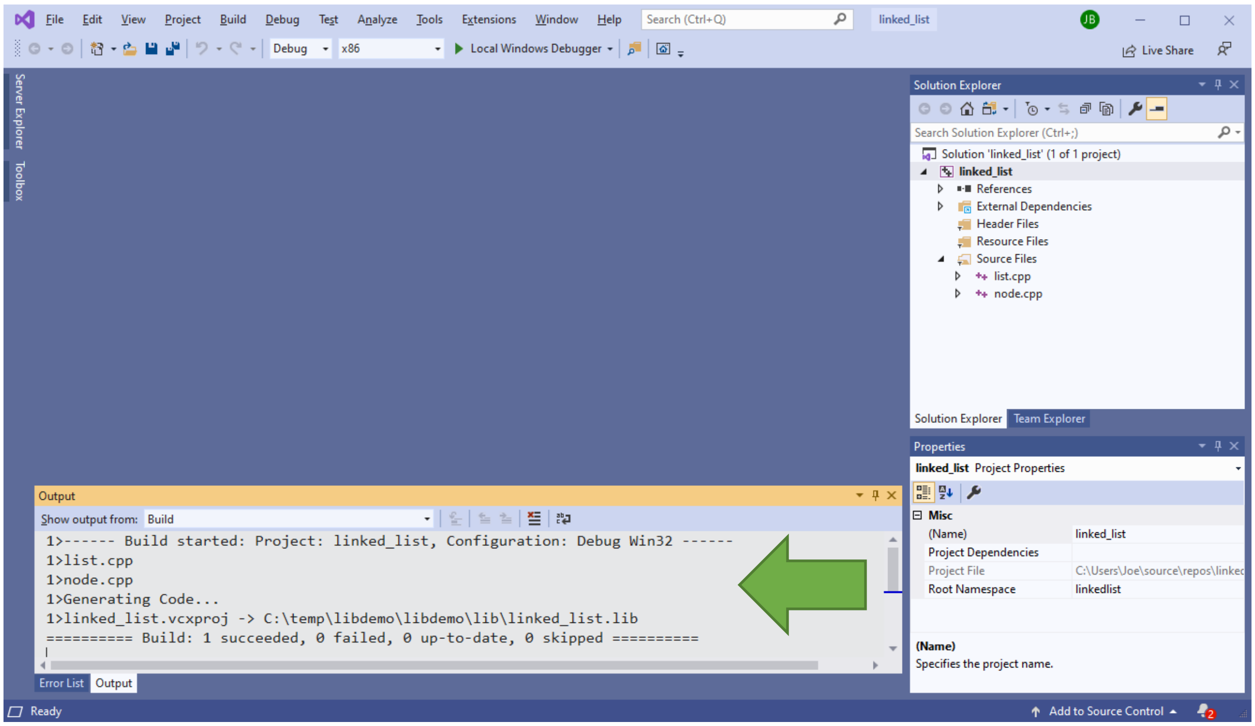
Add the include directories

- Right-mouse click on the project name (linked_list) and select Properties.
- In the Property Pages
  - Under Configuration Properties, expand C/C++ and select the General property
  - Click in the input area to the right of Additional Include Directories
  - Enter the directory path to the header files

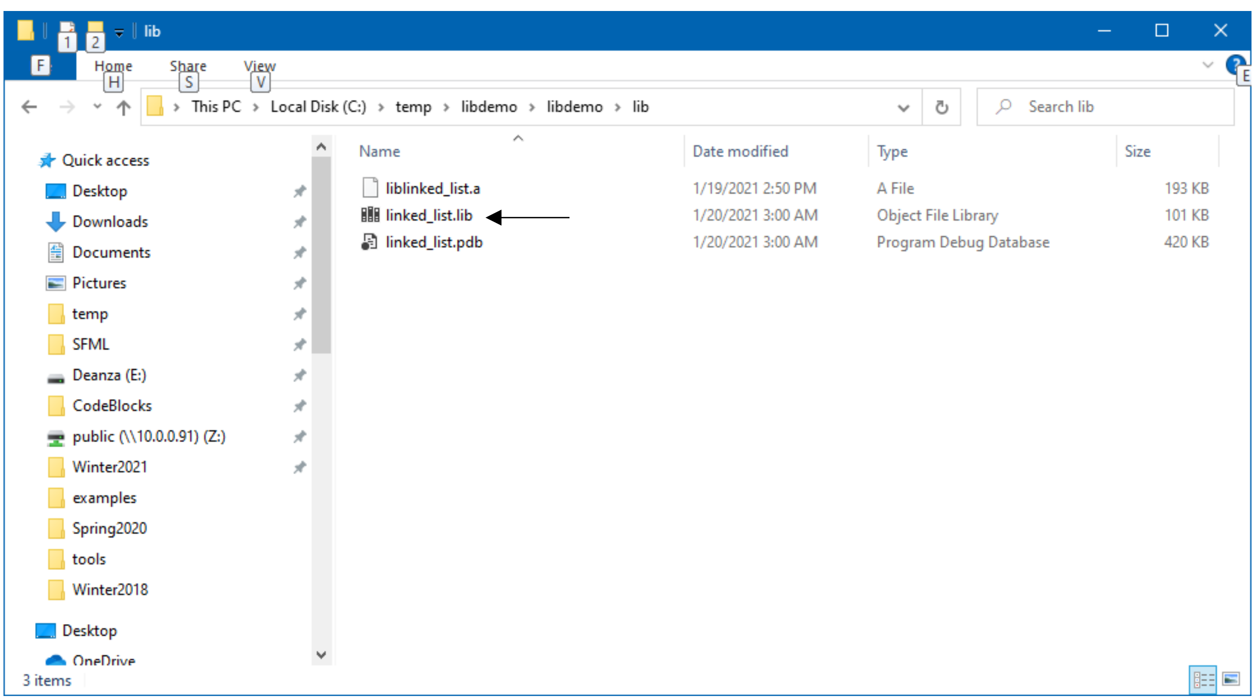


Build the library

Choose Build in the menu, then Build Solution.
You should see messages in the output window indicating success.

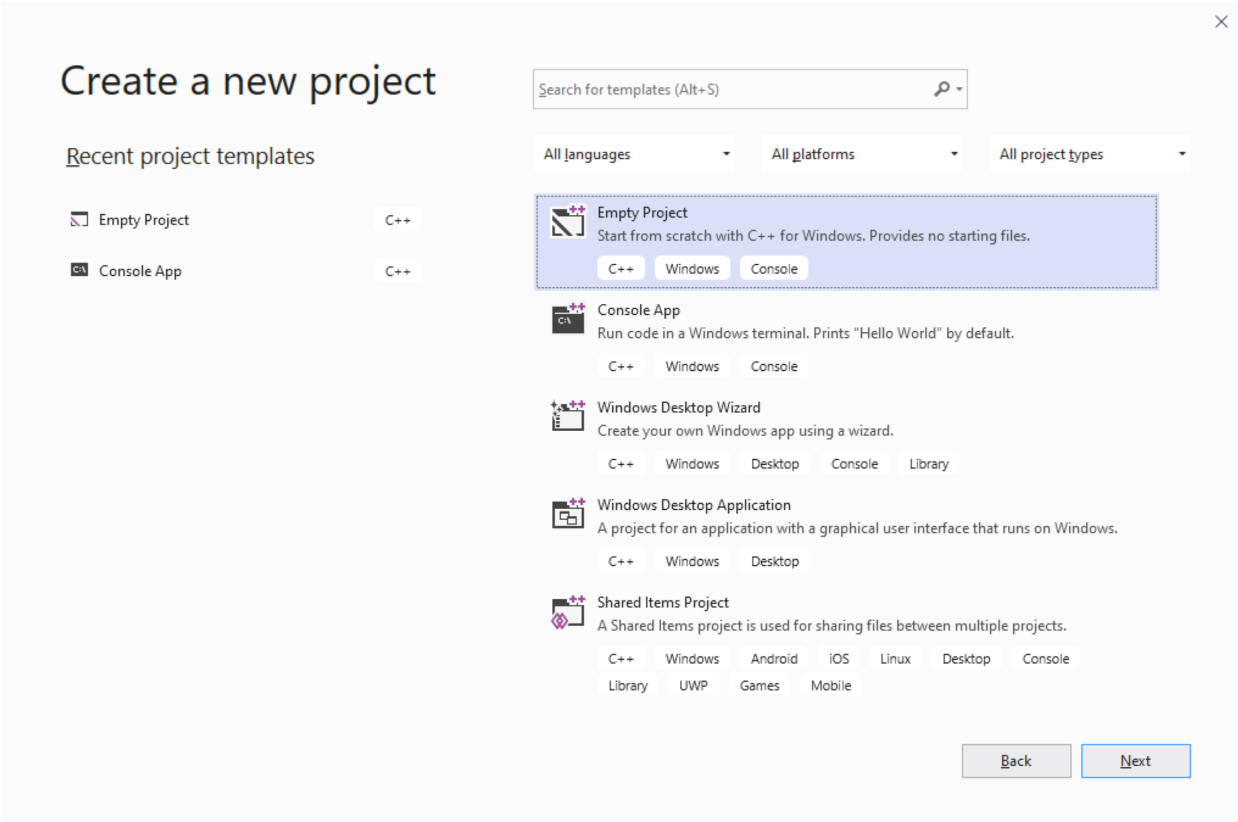


You should see the library now in your Output directory.

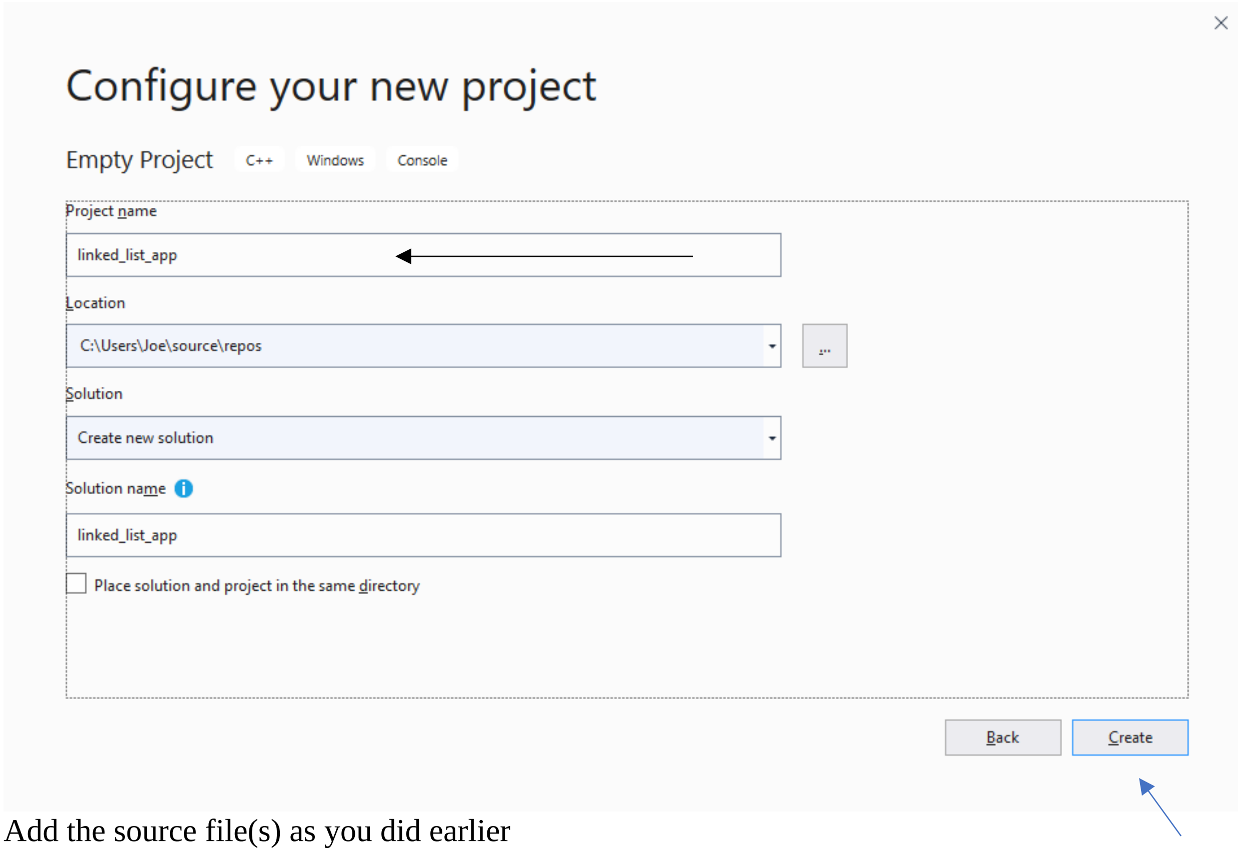


Create the application program project.

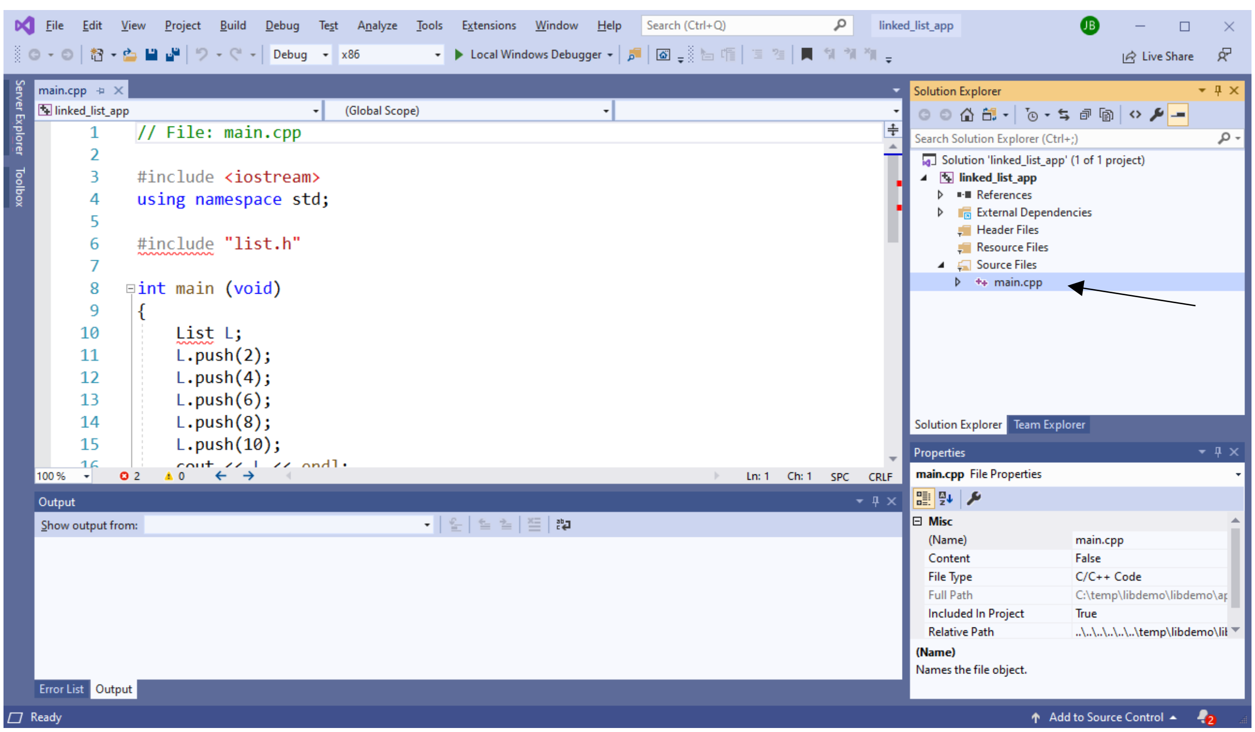
Create a new project just like you did to create the static library.



Name the project



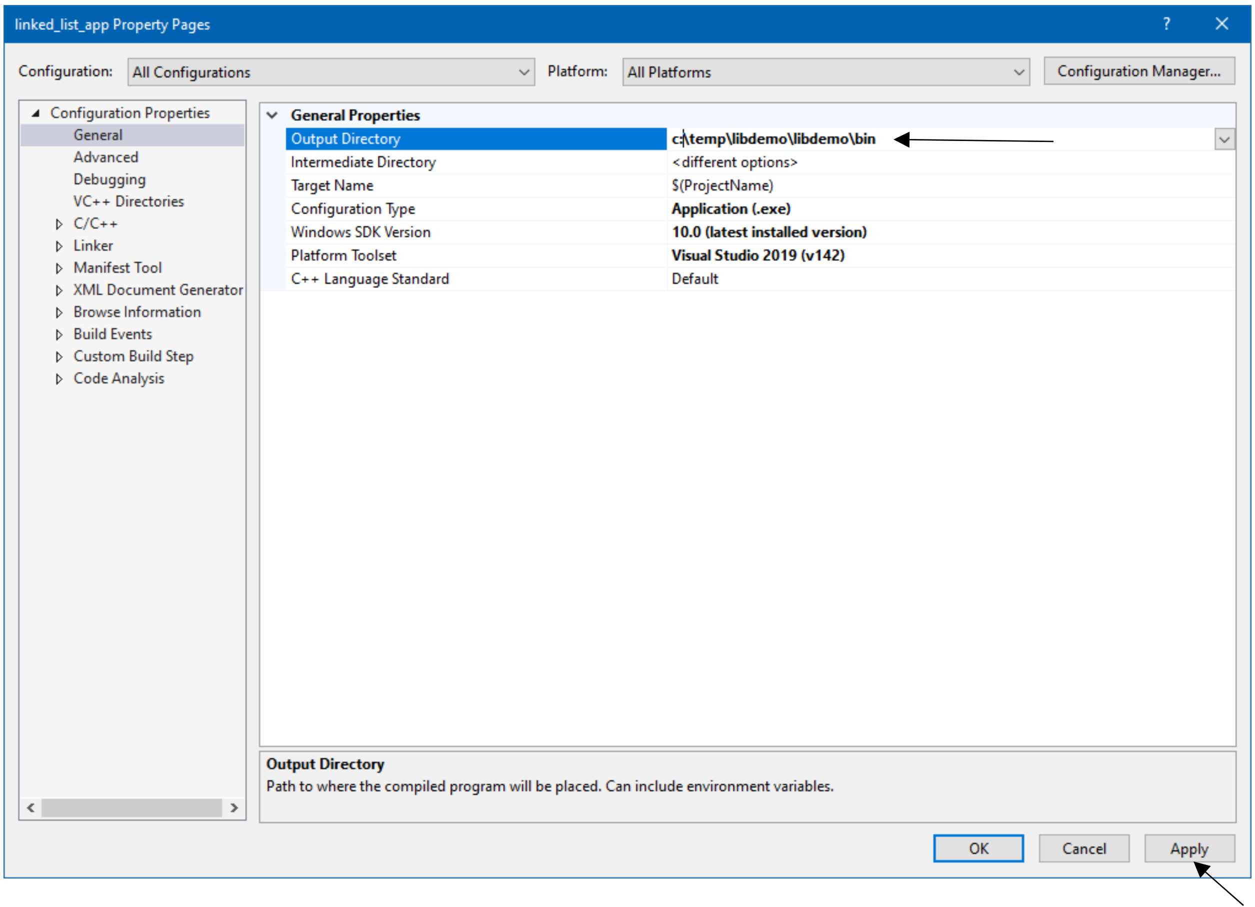
Add the source file(s) as you did earlier



Set the Output Directory

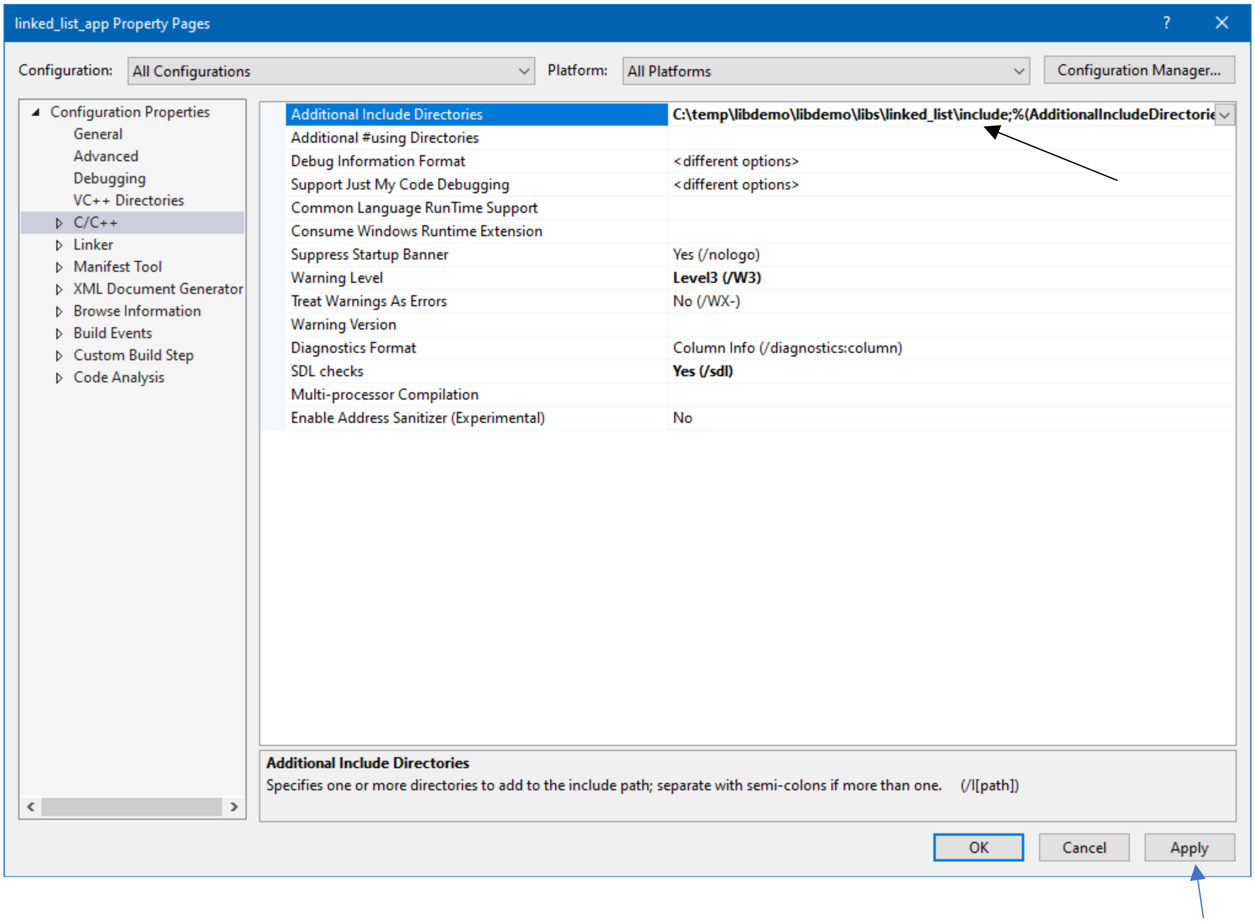
Add the Project Properties (right-mouse click) on the project name and select Properties.

In the Property Pages pop-up window, under General Configuration Properties, change the Output Directory.



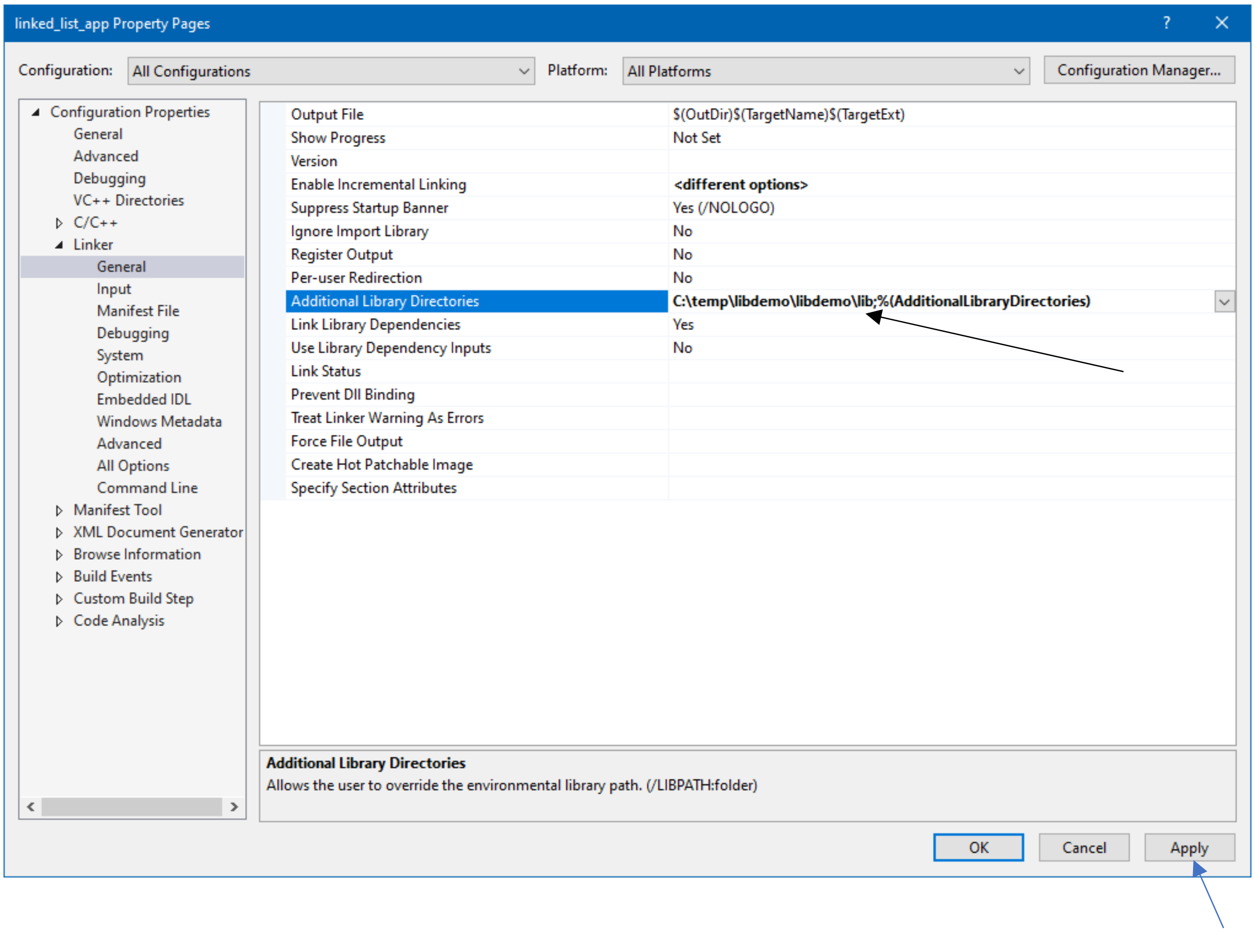
Add Include Directories

Under C/C++, add the Additional Include Directories.



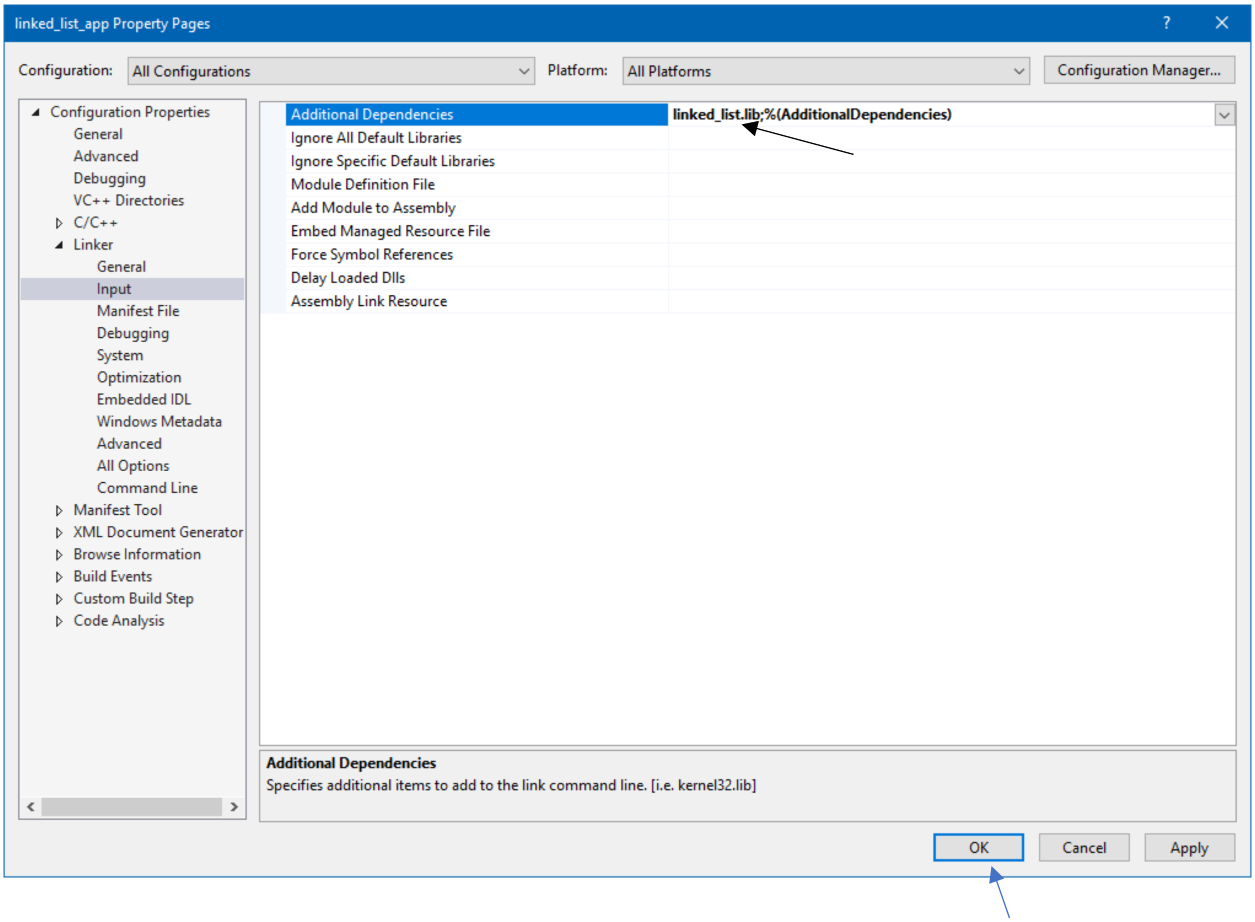
Add the Library Directory to “link in”

Expand the Linker Configuration Properties and select the General page
Under Additional Library Directories, add the path to the libraries to be “linked in”.



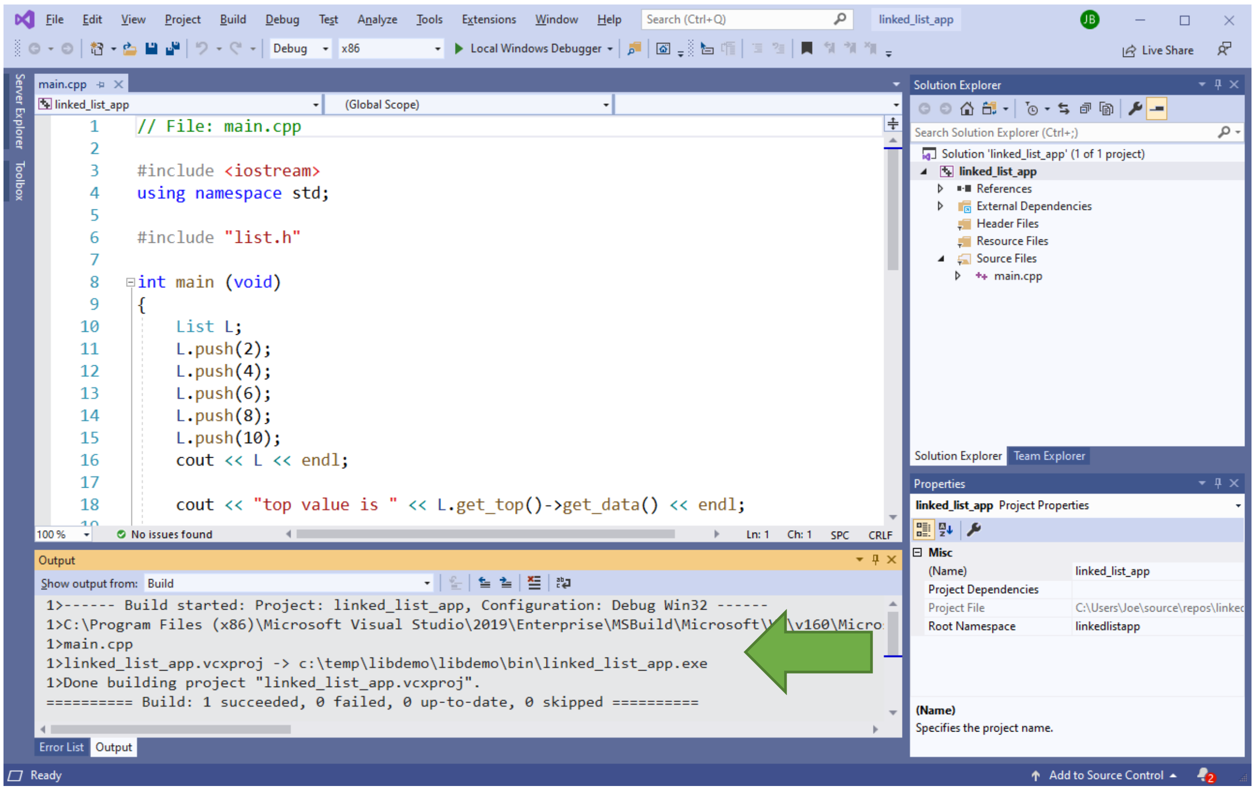
Add the Libraries to be "linked in"

Under the Linker, Input Configuration Properties, enter the Additional Dependencies (library filenames).

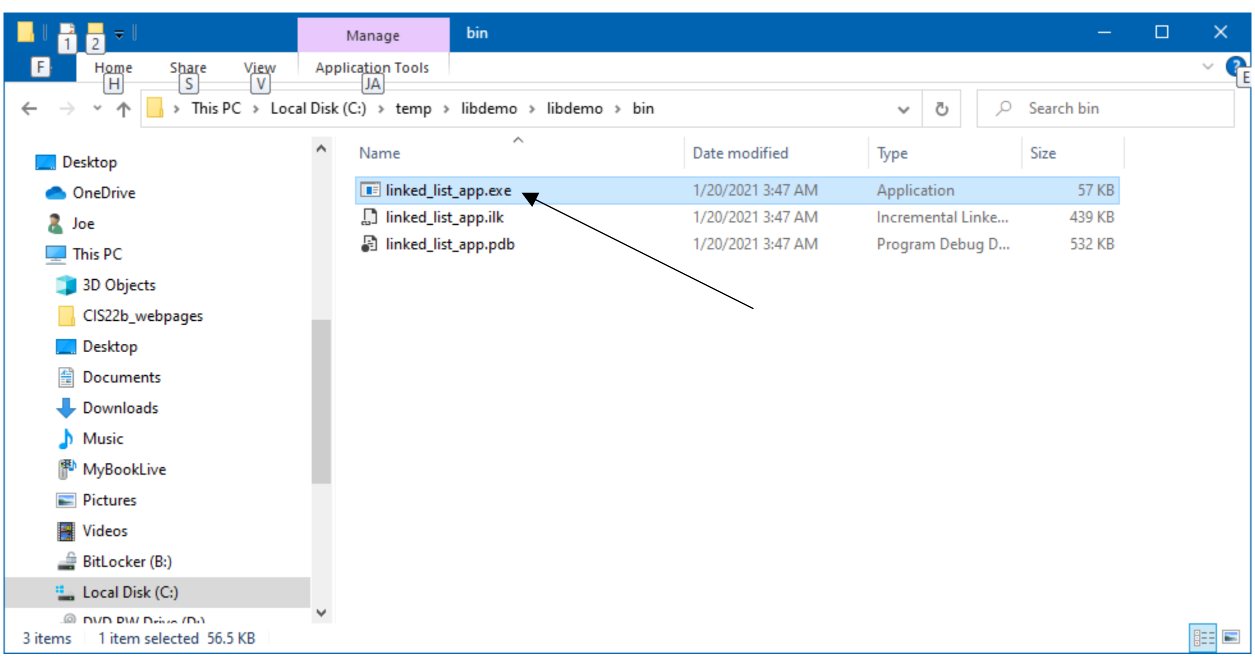


Build and run the application

Choose Build in the menu, then Build Solution.
You should see messages in the output window indicating success.

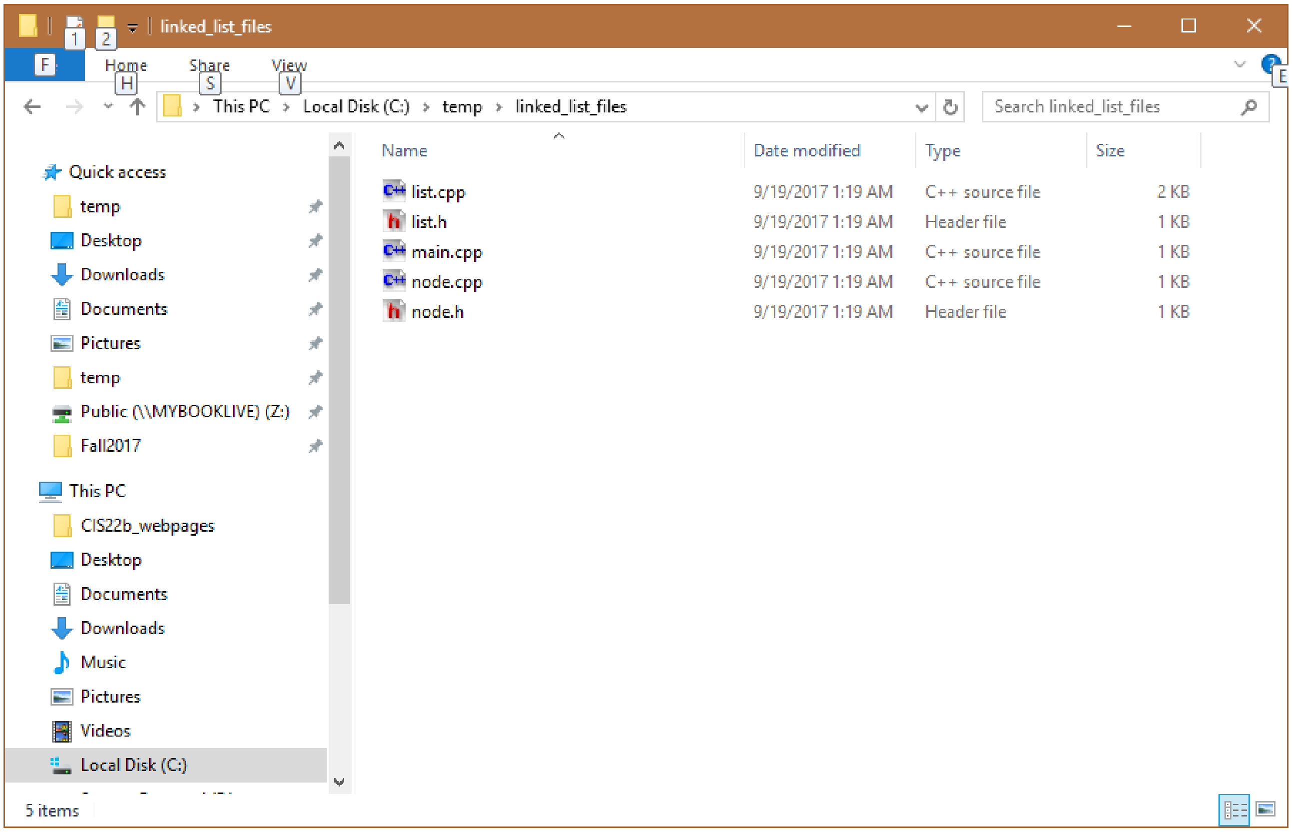


You should see the application executable file in the assigned directory location.

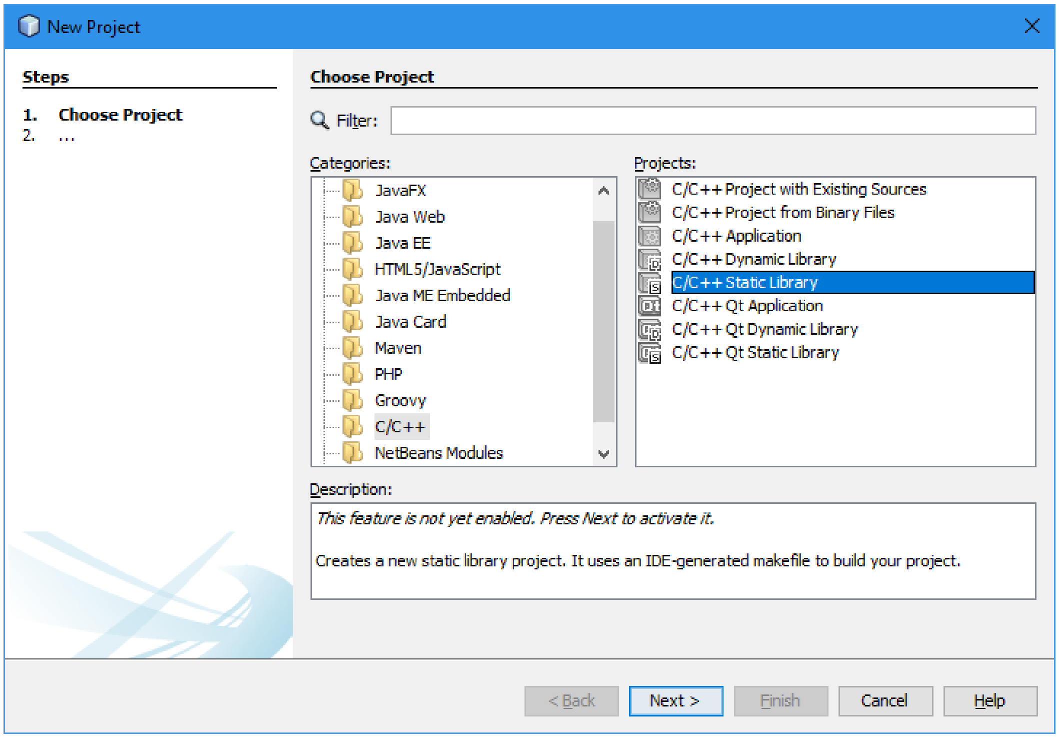


Example 5 - Create Static Library Using NetBeans 8.2

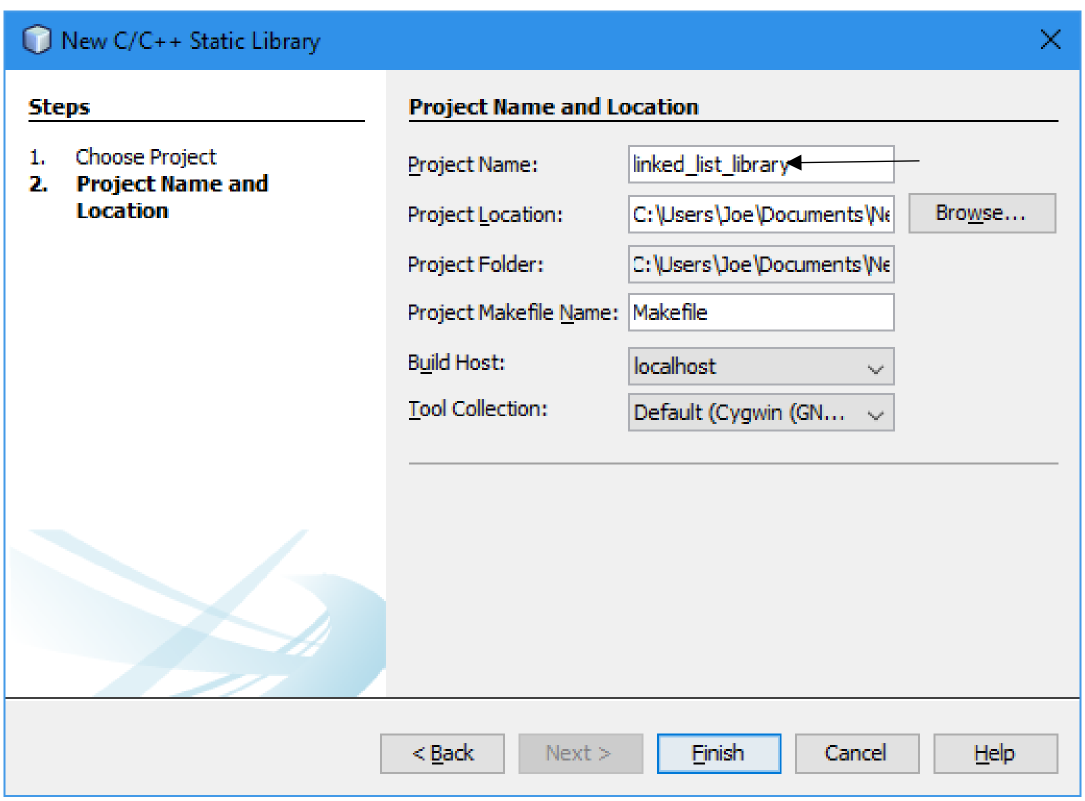
Starting File List



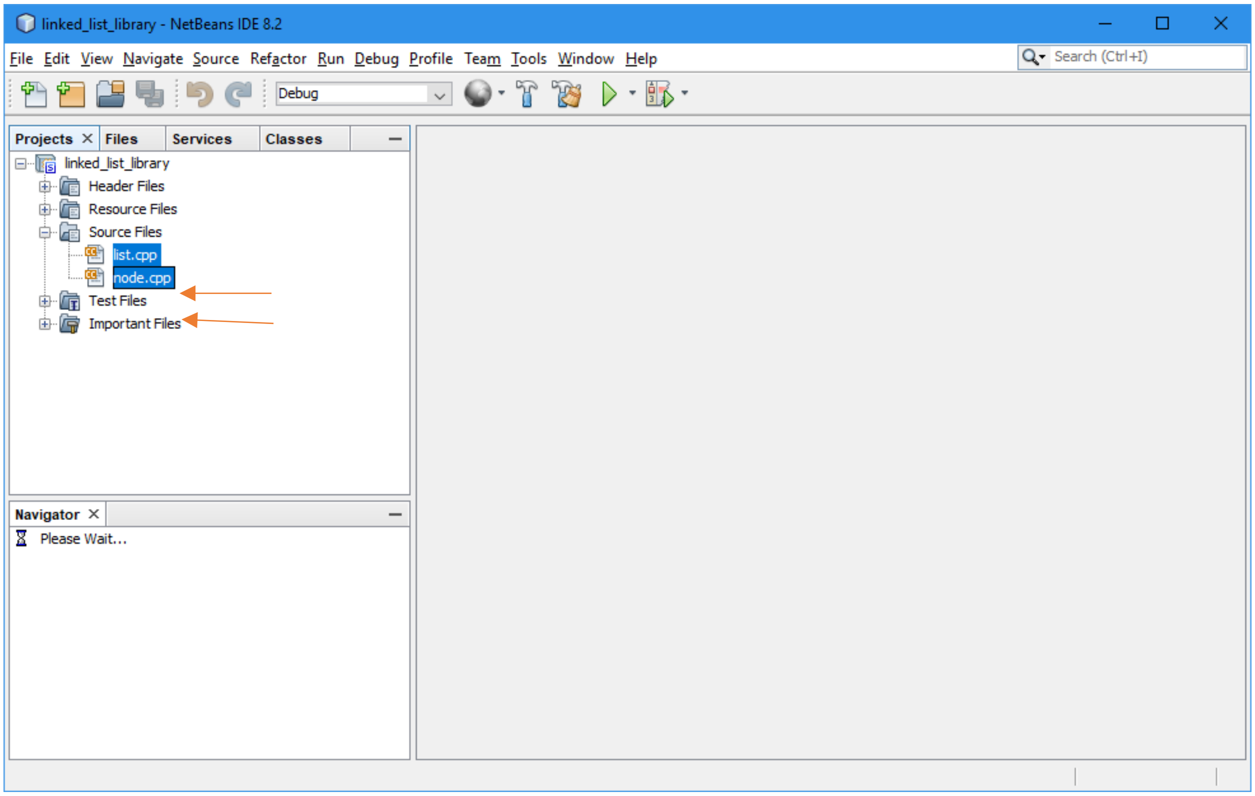
Create a new project. Select File -> New Project … -> C/C++ -> C/C++ Static Library



On the next pop-up, provide a Project Name (recommended). In this example, we will use linked_list_library.

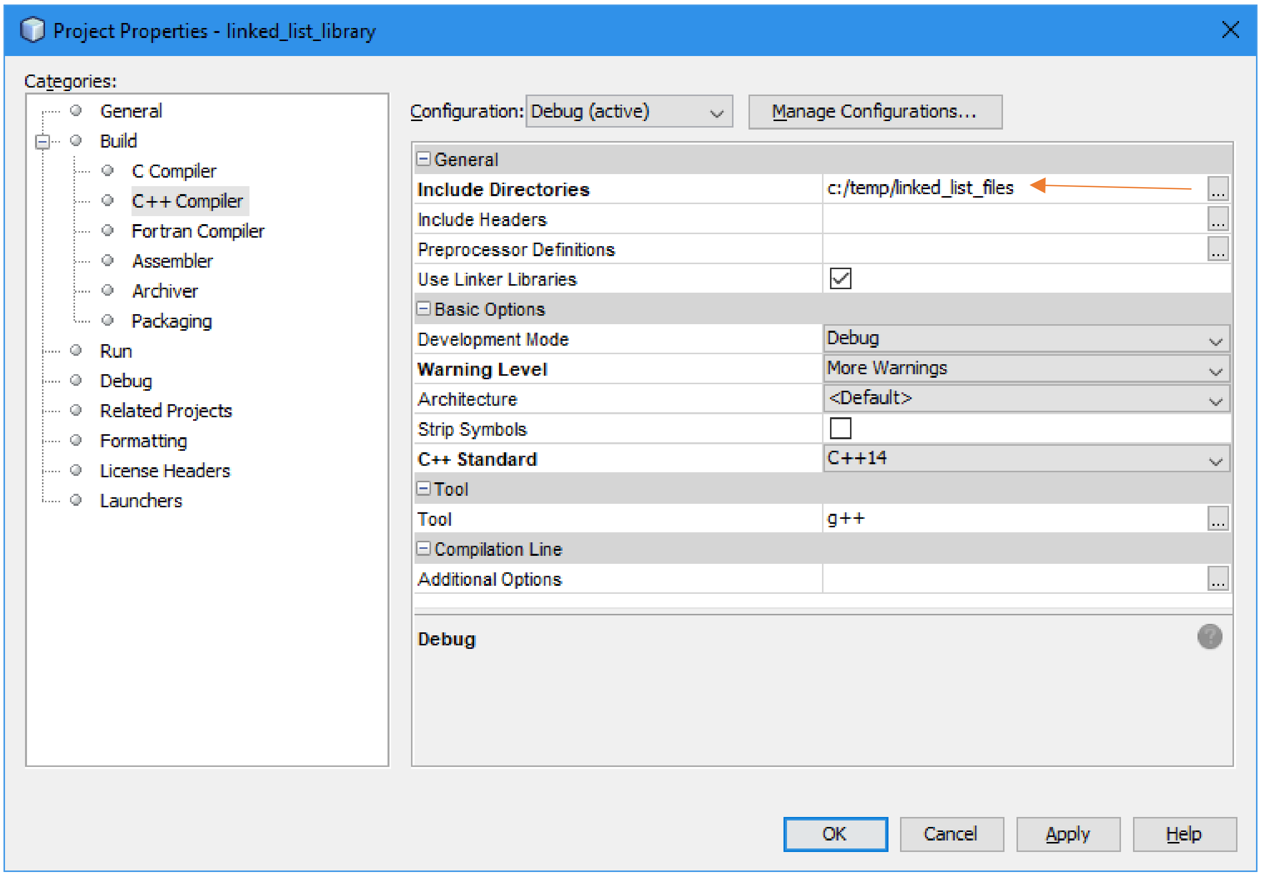


Add the source files for the library. You can use right-mouse click under Source Files in the Project Window.

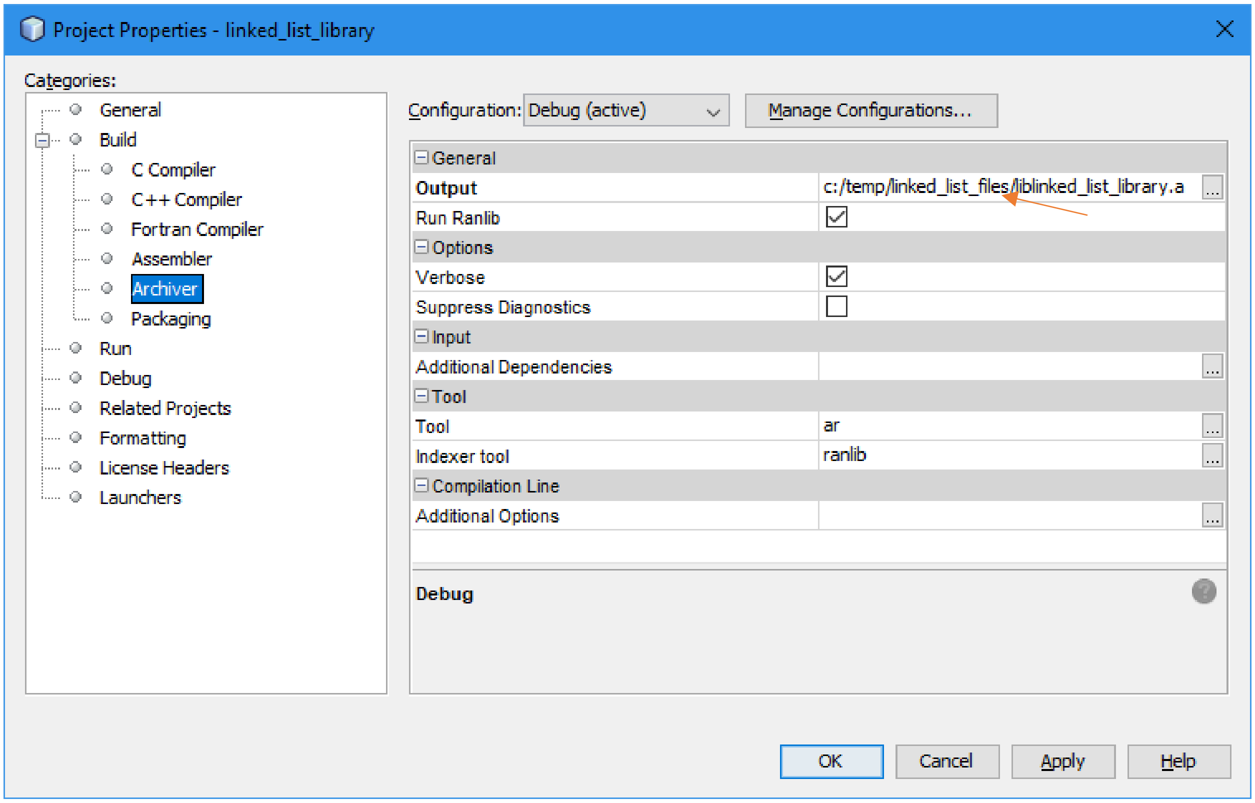

linked_list_library - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Debug
Projects
Files
Services
Classes
linked_list_library
Header Files
Resource Files
Source Files
list.cpp
node.cpp
Test Files
Important Files
Navigator
Please Wait...

Change the project properties.

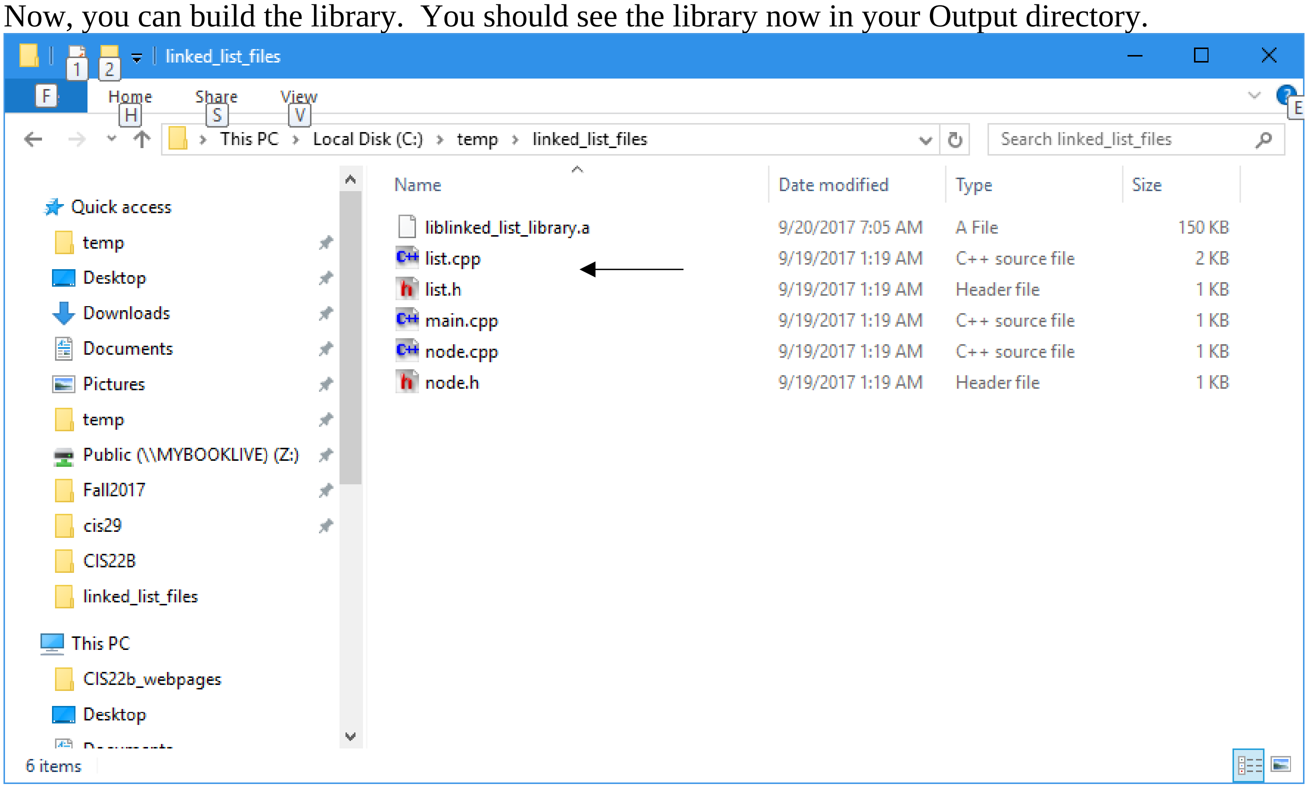
- Right-mouse click on the library name (linked_list_library) and select Properties.
- In the Project Properties pop-up, under Build, C++ Compiler, add the Include Directories.



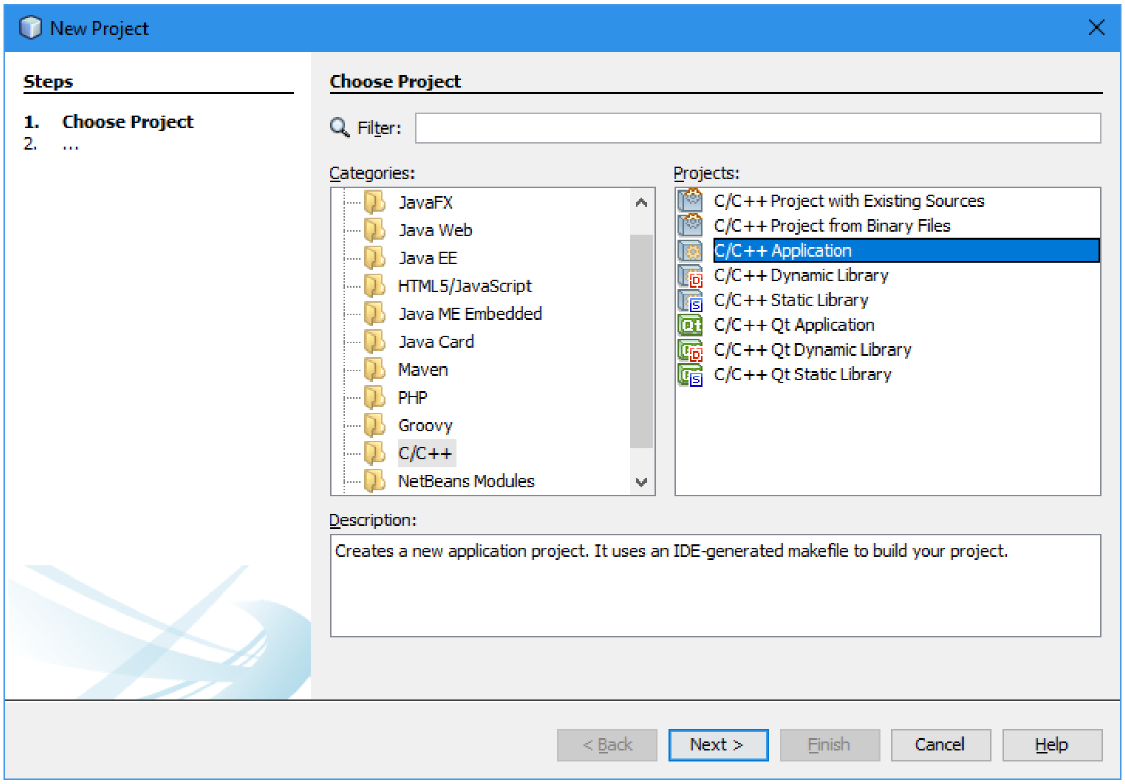
- And under Achiver, change the Output directory.



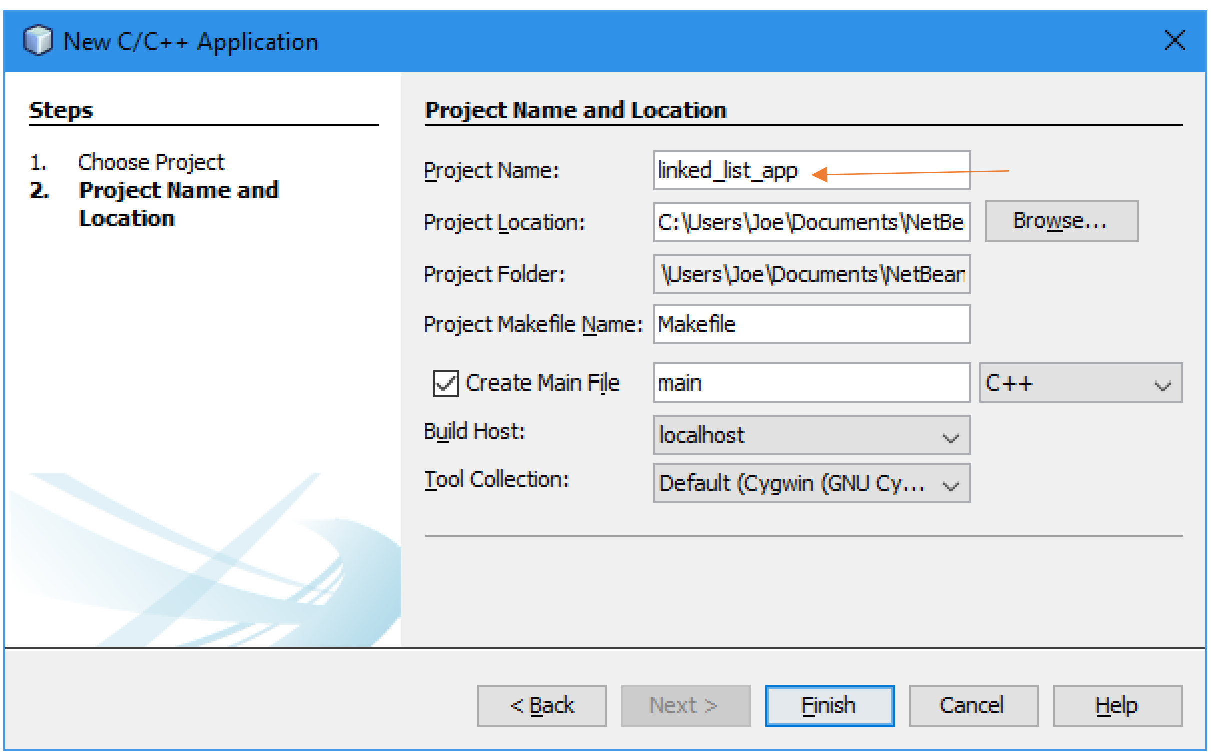
Now, you can build the library. You should see the library now in your Output directory.



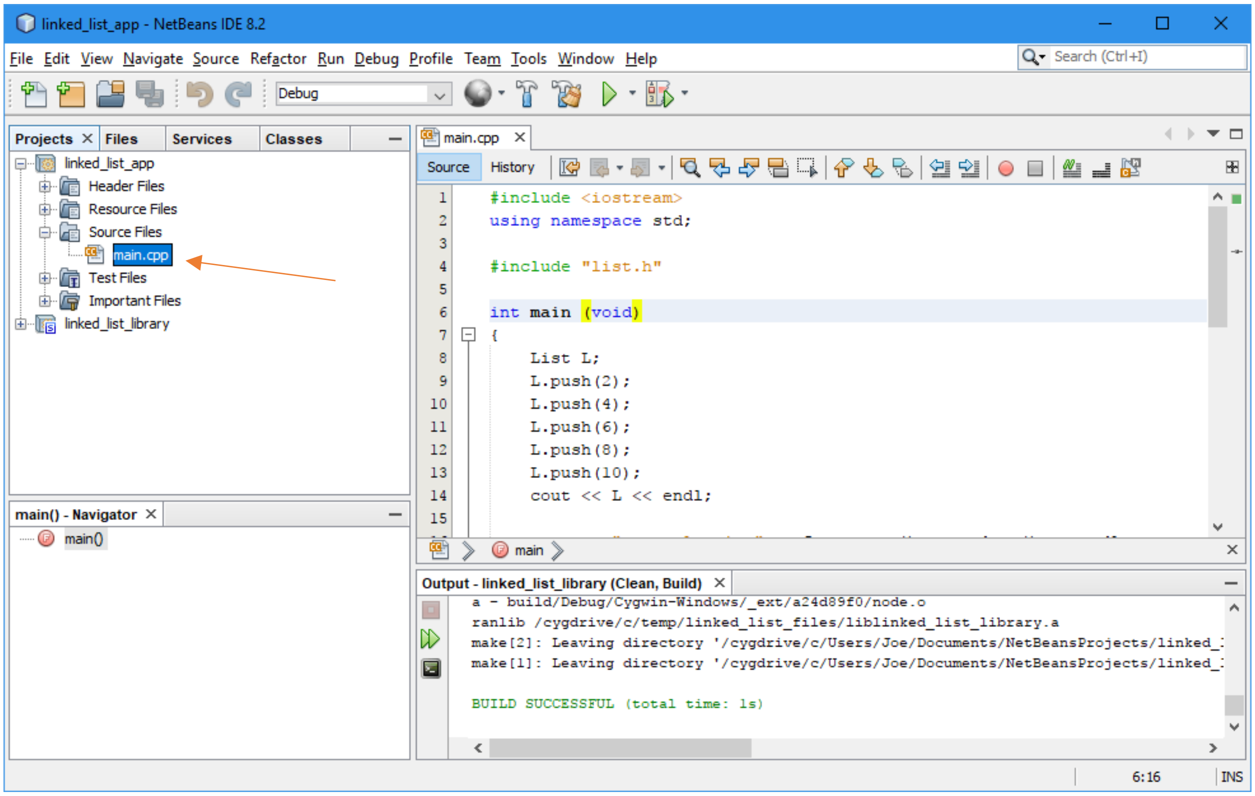
Create the application program project:
File -> New Project … -> C/C++ -> C/C++ Application



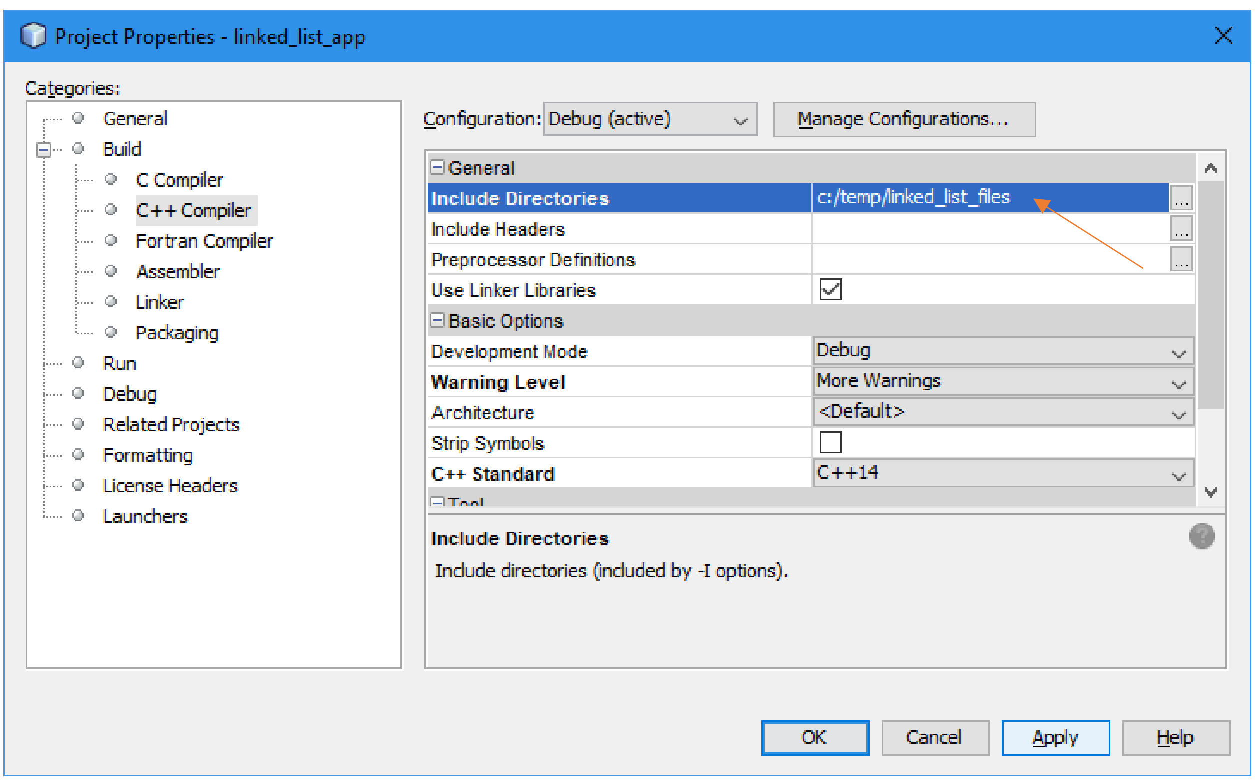
Name the project



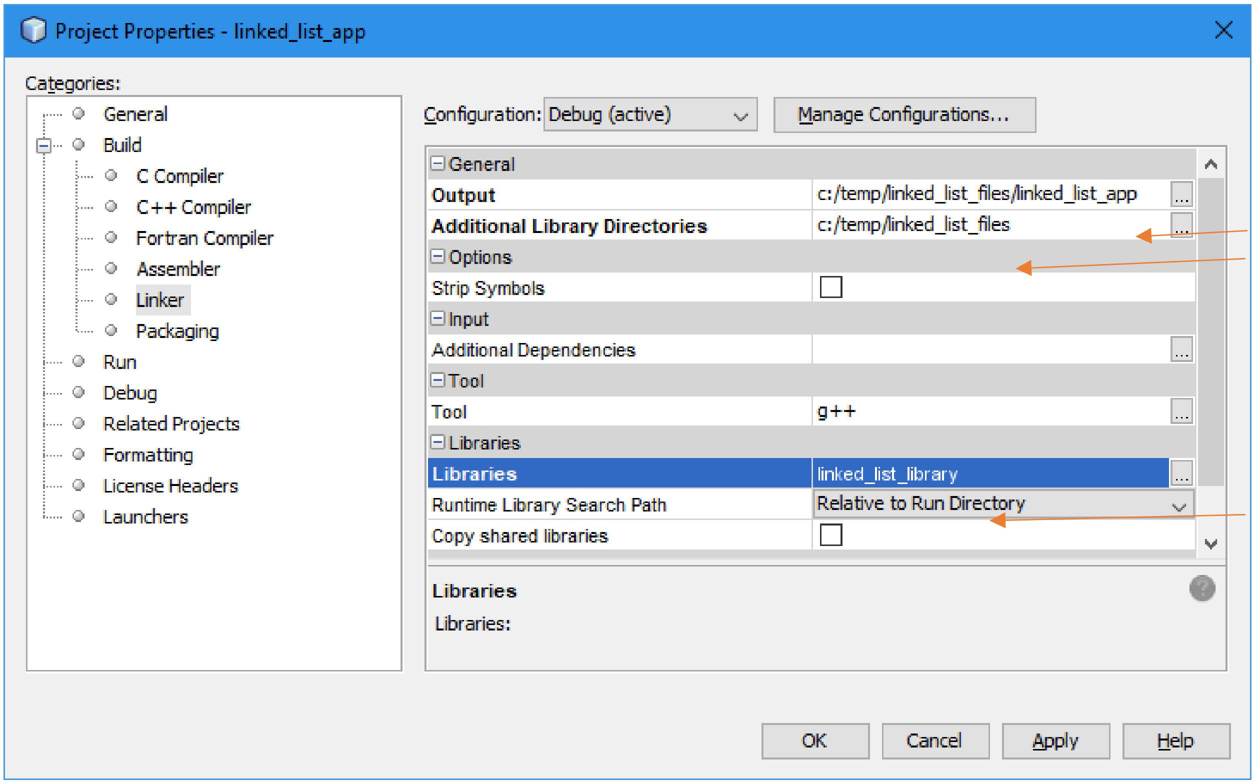
Add the source file(s)



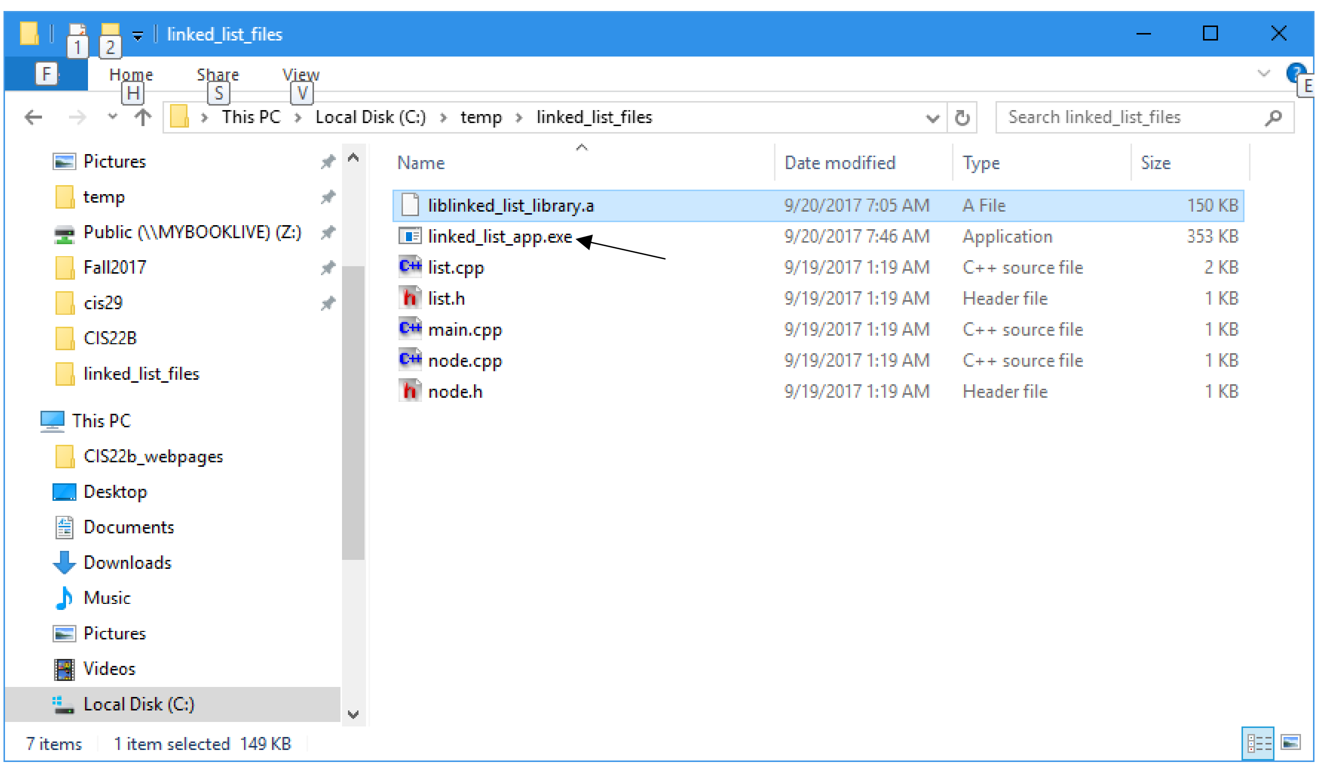
Under Project Properties, add the Include Directories



Under Linker add Output, Additional Library Directories and Libraries.



You should now be able to build and run the application.
Your file list directory should now contain the linked list executable.



What is a shared library?

A shared library is a library file that is linked in at run time, like a dll file. Shared libraries are used on Linux and Unix. Dynamically linked libraries may not have to be present at compile time, and does not have to be present at application startup. Shared libraries must be present at both times.

Library extensions

Library Type	**Extension**
Static	.a
Dynamically linked	.dll
Shared	.so

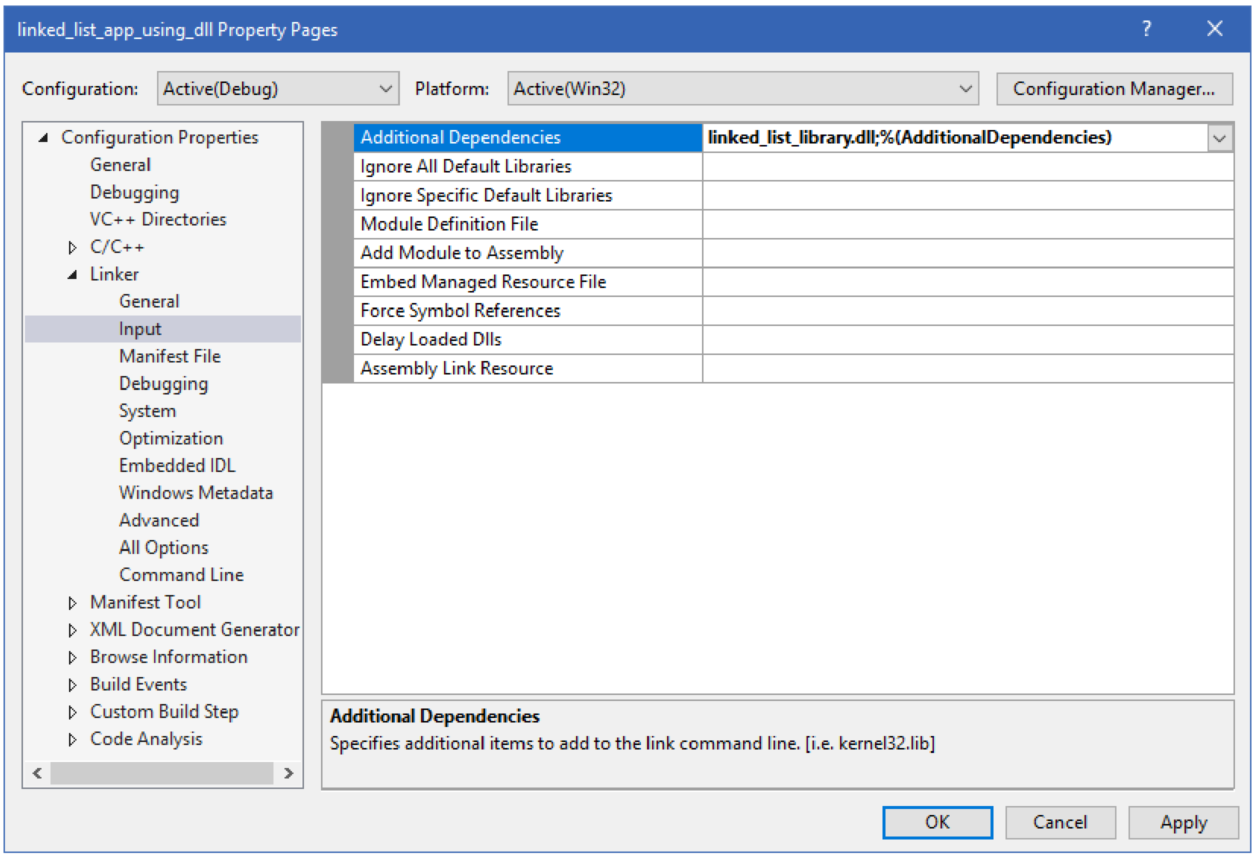
Example 6 - Create a shared library under Linux

1. `$ ls`
```
list.cpp  list.h  main.cpp  node.cpp  node.h
```
2. `$ g++ -I. -shared -fPIC list.cpp node.cpp -o liblinked_list.so`
3. `$ ls`
```
liblinked_list.so  list.cpp  list.h  main.cpp  node.cpp  node.h
```
4. `$ g++ -L. -llinked_list main.cpp -o linked_list_app`
5. `$ ls`
```
liblinked_list.so  linked_list_app  list.cpp  list.h  main.cpp
node.cpp  node.h
```
6. `$ linked_list_app`
```
linked_list_app: error while loading shared libraries:
liblinked_list.so: cannot open shared object file: No such file or
directory
```
7. `[added current directory to LD_LIBRARY_PATH environment variable]`
8. `$ linked_list_app`
```
10 8 6 4 2
top value is 10
2 is in the list
6 is in the list
10 is in the list
10 removed from the list
8 6 4 2
3 is not in the List
8 4 2
4
```

Explanation

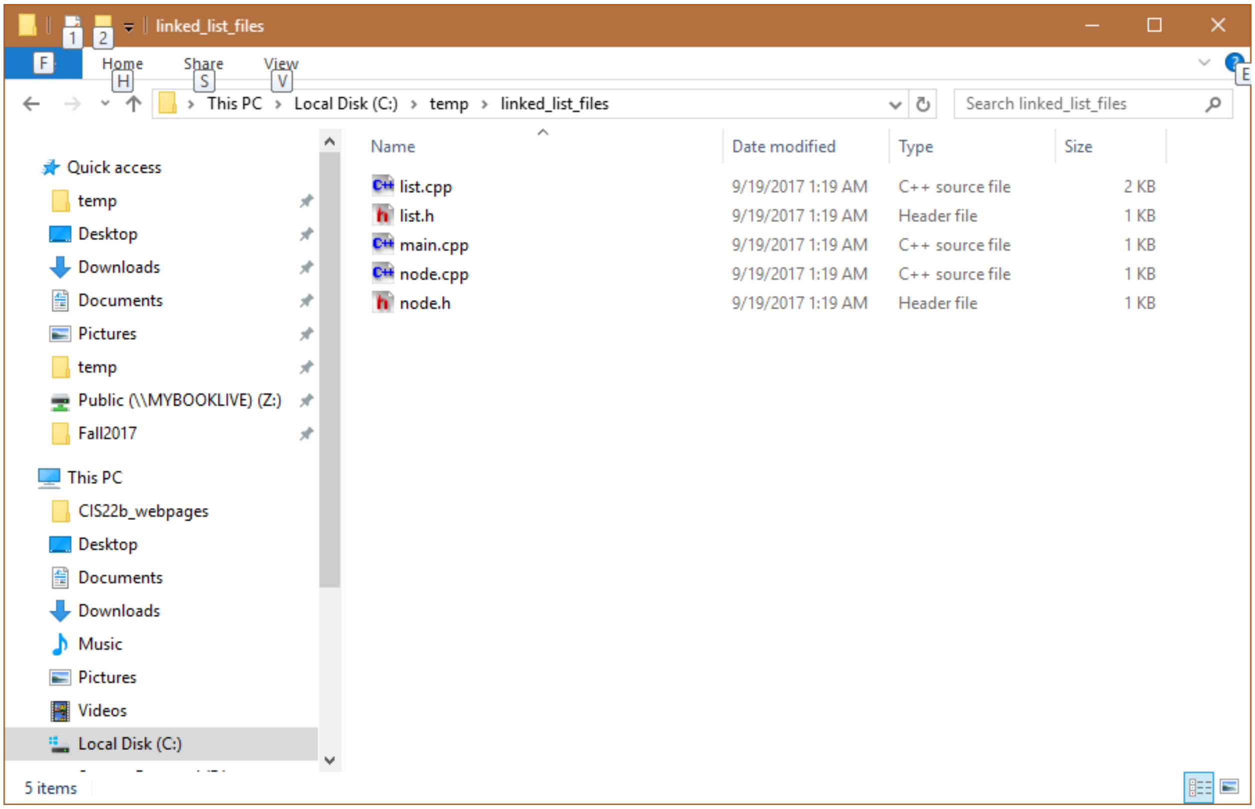
1. List the files in the current directory.

2. Compile list.cpp and node.cpp into a shared library, named liblinked_list.so. -I. means to include the current directory for compilation. The -fPIC option tells the compiler to generate position-independent code (i.e. code that can be loaded at any particular virtual memory address at runtime).
3. List the files in the current directory.
4. Compile main.cpp to the executable name linked_list_app. Link in the library called liblinked_list that is located in the current directory.
5. List the files in the current directory.
6. Attempt to run the linked_list_app executable. The run fails because the shared library is not found.
7. The environment variable, LD_LIBRARY_PATH must be modified so that the current directory is also searched for the shared library.
8. The application now runs.

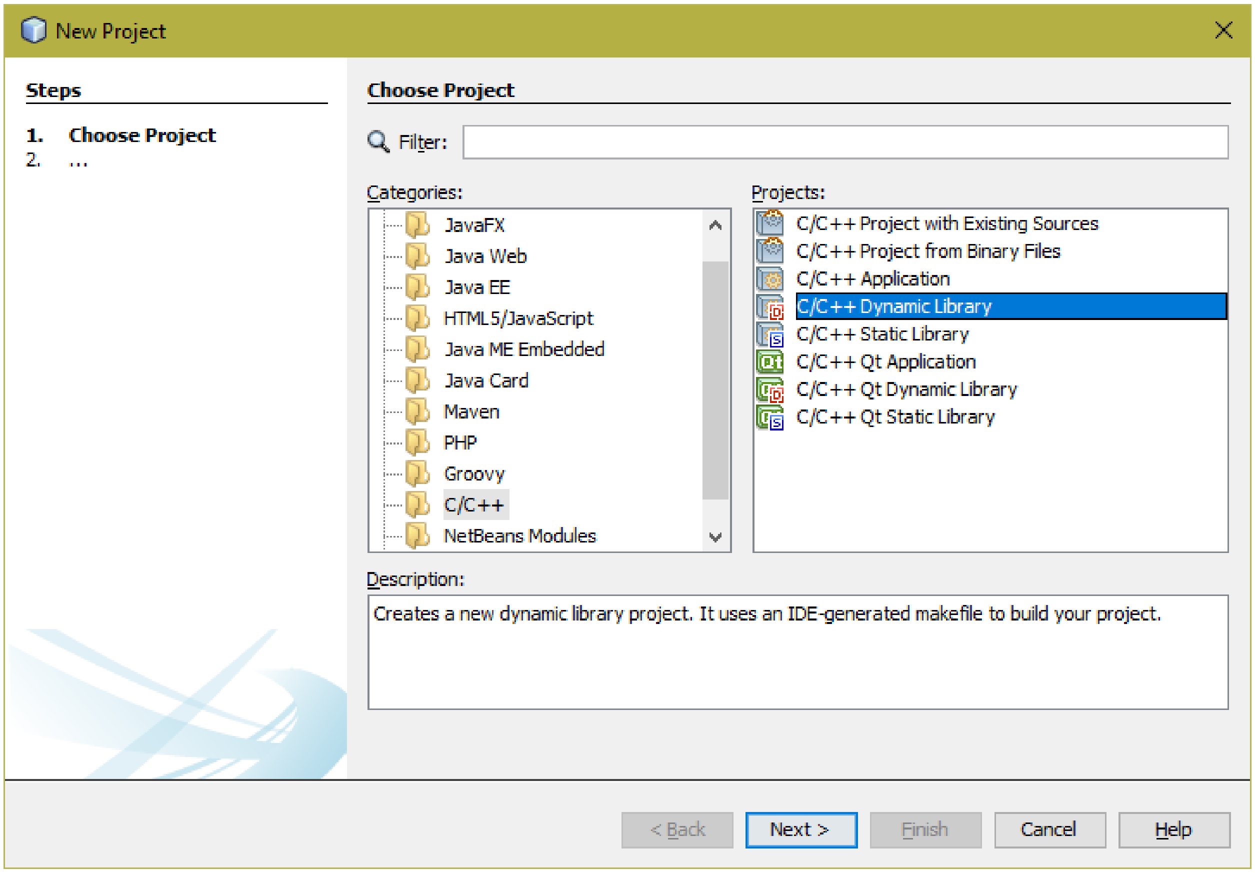


Example 7 - Create Dynamic Library Using NetBeans 8.2 On Windows

Starting File List



Create a new project. Select File -> New Project … -> C/C++ -> C/C++ Dynamic Library

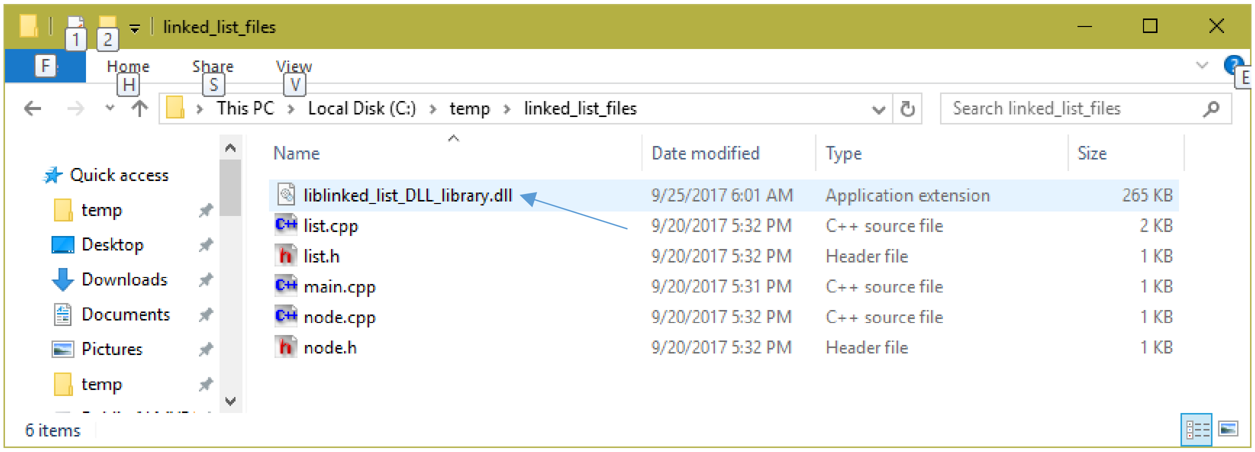


Follow the same steps that was demonstrated in the Static Library Using NetBeans.

Change the project properties.

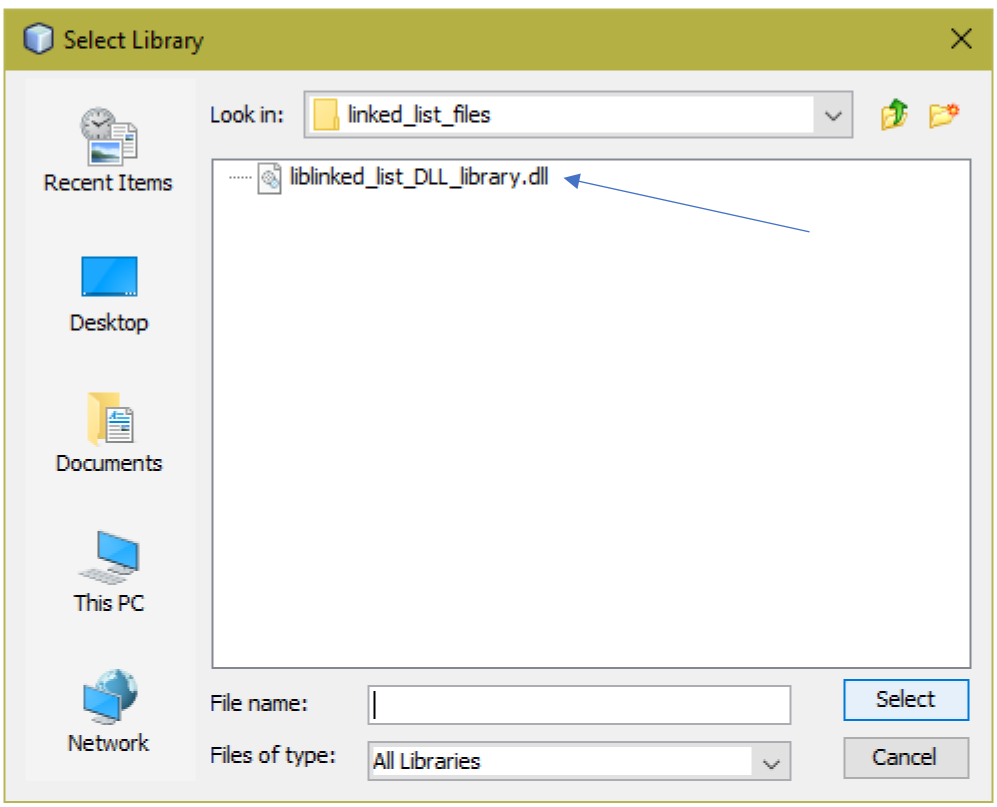
- Right-mouse click on the library name and select Properties.
- In the Project Properties pop-up, under Build, C++ Compiler, add the Include Directories.
- And under ***Linker***, change the Output directory.

Now, you can build the library. You should see the library now in your Output directory.



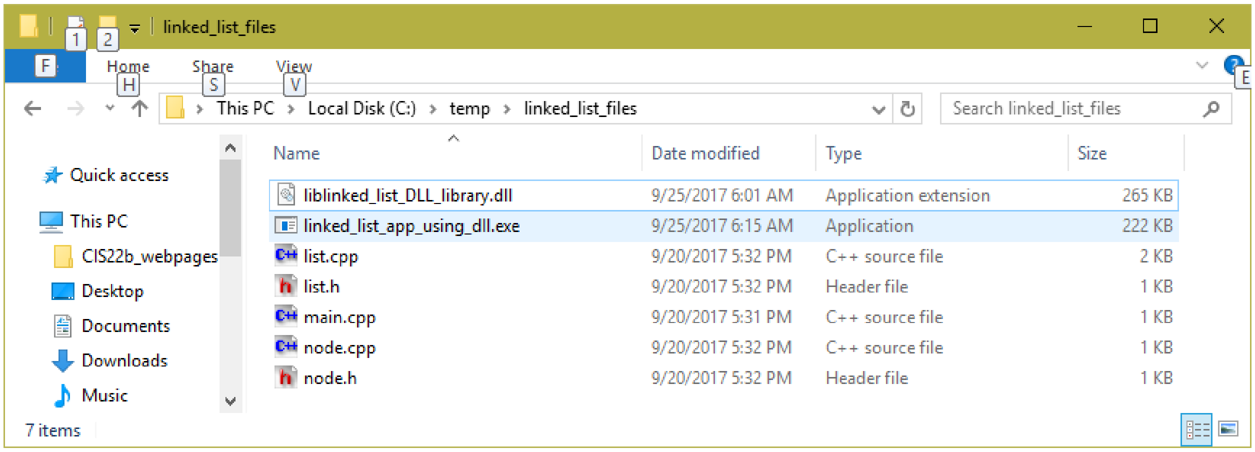
To build an application that uses the DLL library, follow the same steps that you did for an application that uses a static library.

When you select the dynamically linked library from the library directory, you should see it in the list, like this:



When you build the application, NetBeans will automatically link in the DLL library.

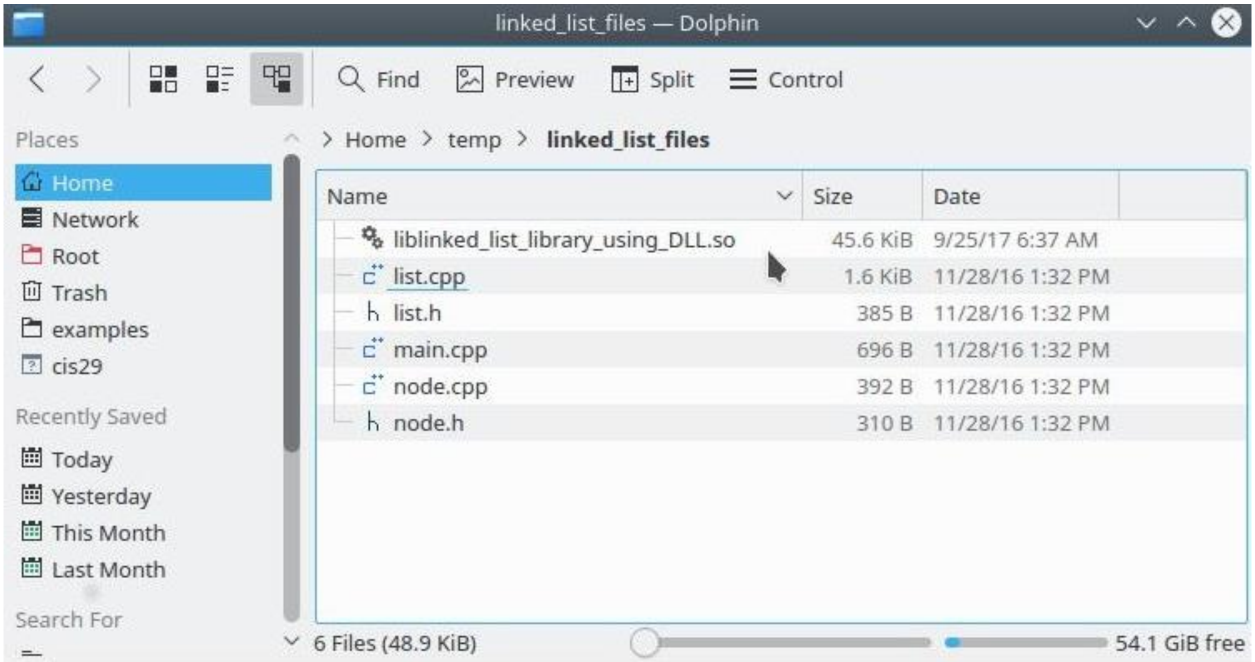
The resultant files are these:



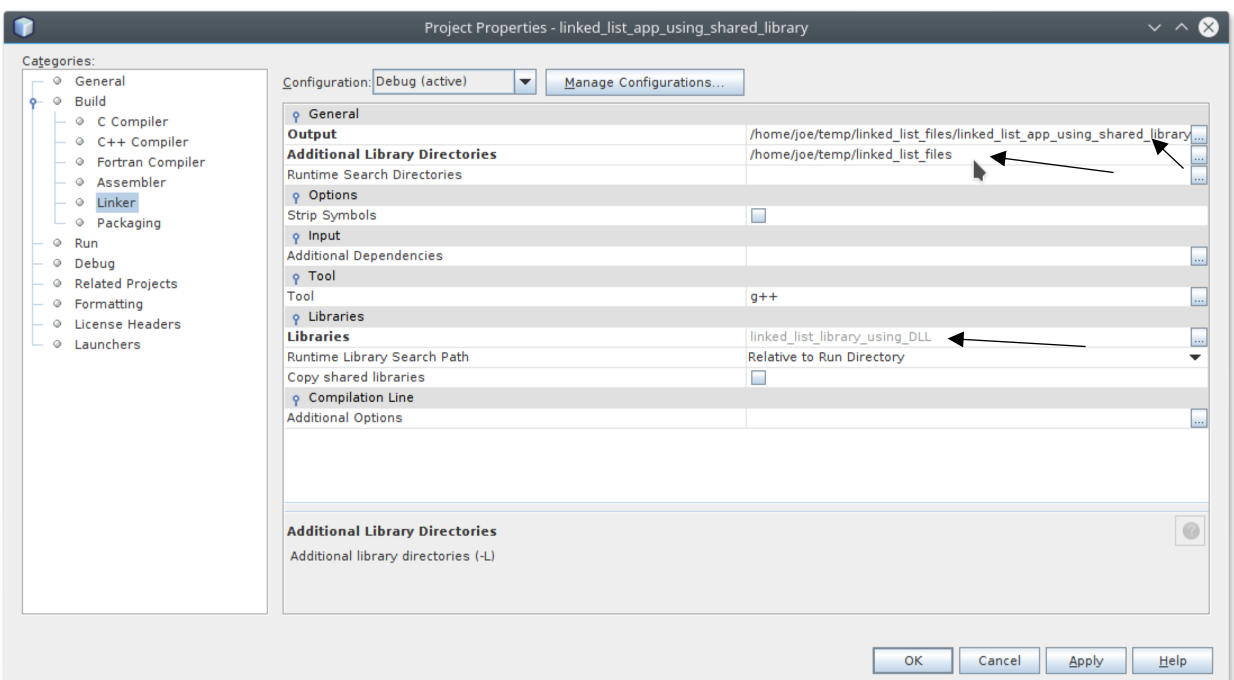
Compare the sizes of the executables of the application using static linking and dynamic linking.

Example 8 - Create Dynamic Library Using NetBeans 8.2 On Linux

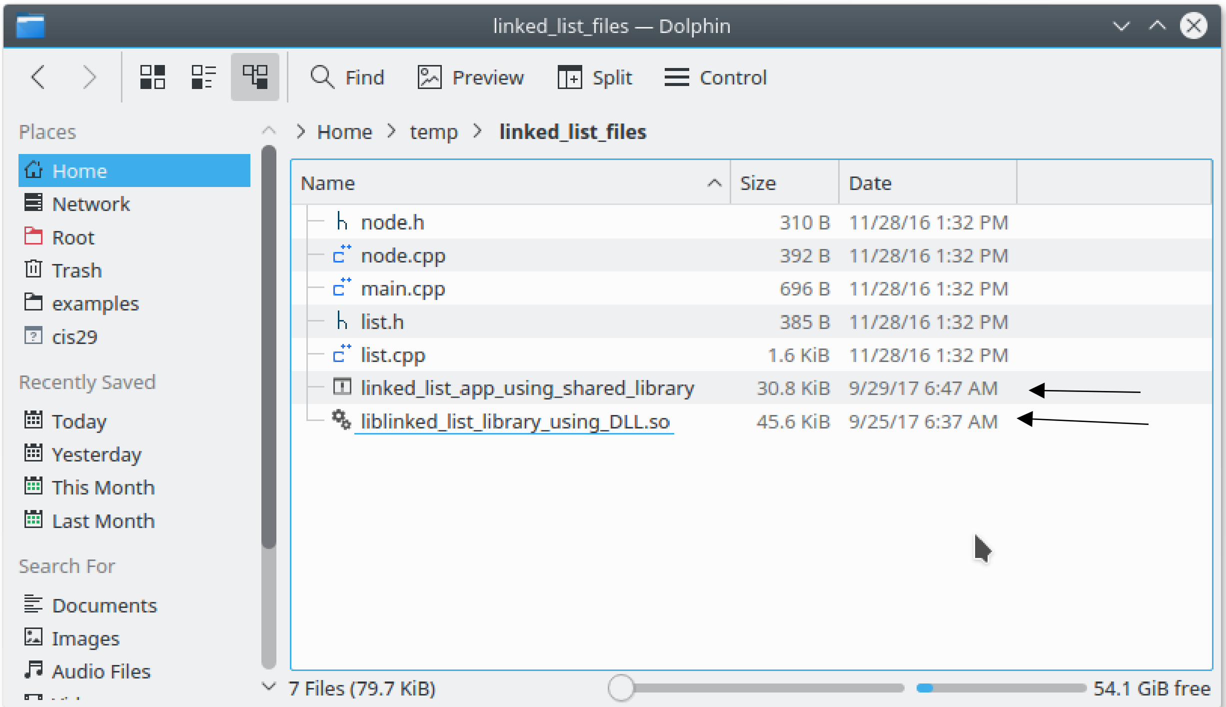
The technique for building a shared library using NetBeans on Linux is the same as building a DLL (dynamically linked library) using NetBeans on Windows. The result is a shared library as shown below.



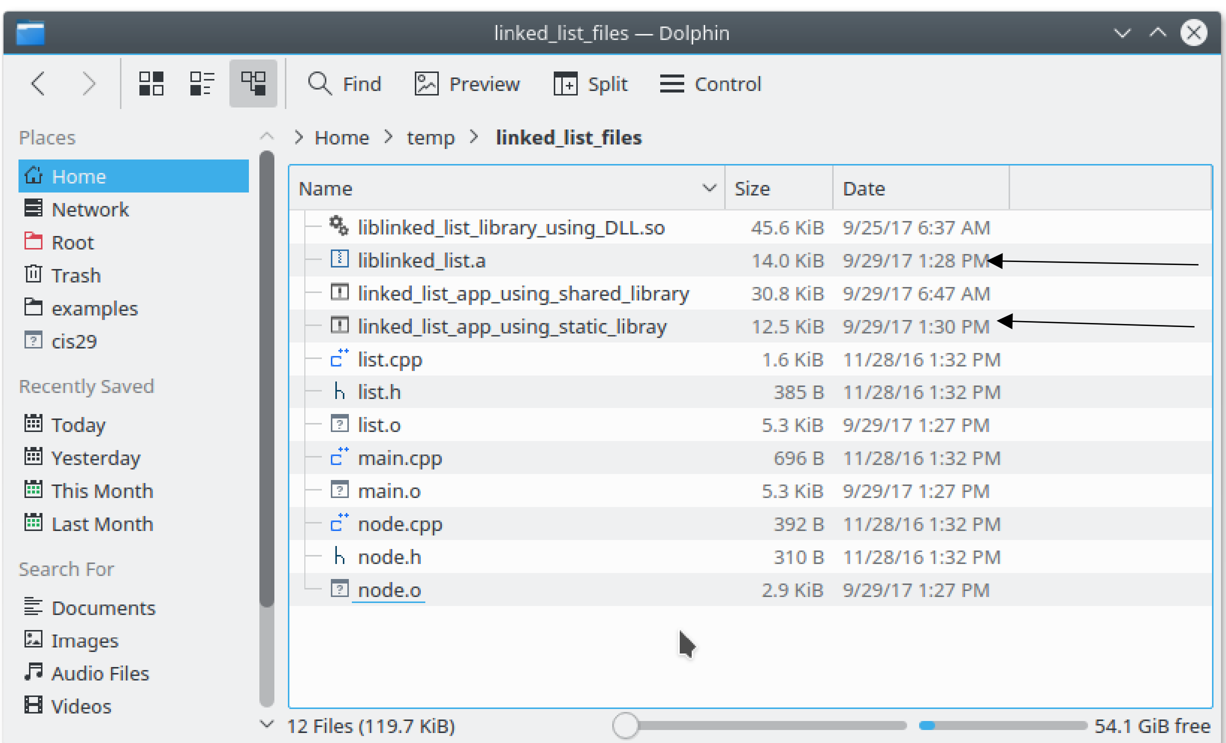
To build the application you have to "link in" the shared library as shown below. Note, the library prefix and extension are not needed here.



The following shows the application after the build using the shared library. Notice the file sizes of the shared library and the executable.



The following shows the same application built using a static library. Again, notice the file sizes.



Using the Curl Library

The curl (or cURL) library is an open source C library that is used for downloading (and uploading internet files). This library may be used to easily retrieve files of almost any type. The library was developed as Linux/Unix as a gnu compatible library. The library is available for Linux/Unix, PC compilers that use a gnu port (Code::Blocks, NetBeans, Eclipse) and Mac IOS. The library may have to be downloaded and installed on your computer.

Example 9 – Using cURL

```
// Command syntax: curl_example [input file] [output file]

#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
#include <cstring>
#include <curl/curl.h>
using namespace std;

ofstream OutFile;
size_t TotalBytesDownloaded = 0;

size_t writeBufferToFile(char *buffer, size_t dummy, size_t numBytes, const char* filename);
void getInternetFile(const char* inputfile, const char* outputfile);

int main(int argc, char* argv[])
{
    char inputFileName[256];
    char outputFileName[256];

    if (argc > 2) // 2 filenames given as arguments
    {
        strcpy(inputFileName, argv[1]);
        strcpy(outputFileName, argv[2]);
    }
    else if (argc > 1) // 1 filename given as an argument
    {
        strcpy(inputFileName, argv[1]);
        cout << "Enter output file => ";
        cin >> outputFileName;
    }
    else
    {
        cout << "Enter input file => ";
        cin >> inputFileName;
        cout << "Enter output file => ";
        cin >> outputFileName;
    }

    OutFile.open(outputFileName);
    if (!OutFile)
    {
```

```
        cerr << "Unable to open output file " << outputFileName << endl;
        exit(EXIT_FAILURE);
    }

    getInternetFile(inputFileName, outputFileName);

    cout << "Total bytes downloaded: " << TotalBytesDownloaded << endl;

    OutFile.close();
}

size_t writeBufferToFile(char *buffer, size_t dummy, size_t numBytes, const char* filename)
{
    cout << "Writing " << numBytes << " bytes to " << filename << endl;
    OutFile.write(buffer, numBytes);
    TotalBytesDownloaded += numBytes;
    return numBytes;
}

void getInternetFile(const char* inputfile, const char* outputfile)
{
    CURL *curl;
    CURLcode res;

    curl_global_init(CURL_GLOBAL_DEFAULT);

    curl = curl_easy_init();
    if (curl)
    {
        curl_easy_setopt(curl, CURLOPT_URL, inputfile);

        /* Define our callback to get called when there's data to be written */
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, writeBufferToFile);

        /* Set a pointer to our struct to pass to the callback */
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, outputfile);

        res = curl_easy_perform(curl);

        /* always cleanup */
        curl_easy_cleanup(curl);

        if (CURLE_OK != res)
        {
            /* we failed */
            cerr << "curl told us " << res << endl;
        }
    }

```

```
    curl_global_cleanup();
}
```

The following execution was performed on Linux (Voyager).

Note, there is a curl include directory under /usr/include. This directory contains the header files for the curl library. If you did your own curl library install, the header files may be found in /usr/local/include.

```
[bentley@voyager cis29_test]$ ls /usr/include/curl
curl.h  curlver.h  easy.h  mprintf.h  multi.h  stdcheaders.h  types.h
```

The curl libraries are in the directory, /usr/lib. If you did your own curl library install, the library files may be found in /usr/local/lib.

```
[bentley@voyager cis29_test]$ ls /usr/lib/*curl*
/usr/lib/libcurl.a  /usr/lib/libcurl.so  /usr/lib/libcurl.so.3
/usr/lib/libcurl.so.3.0.0
```

Here is the compile command

```
[bentley@voyager cis29_test]$ g++ curl_example.cpp -Wall -o
curl_example
```

Notice the link errors

```
/tmp/ccpFuDRi.o: In function `getInternetFile(char const*, char
const*)':
curl_example.cpp:(.text+0xb9): undefined reference to
`curl_global_init'
curl_example.cpp:(.text+0xbe): undefined reference to `curl_easy_init'
curl_example.cpp:(.text+0xe4): undefined reference to
`curl_easy_setopt'
curl_example.cpp:(.text+0xfc): undefined reference to
`curl_easy_setopt'
curl_example.cpp:(.text+0x113): undefined reference to
`curl_easy_setopt'
curl_example.cpp:(.text+0x11c): undefined reference to
`curl_easy_perform'
curl_example.cpp:(.text+0x128): undefined reference to
`curl_easy_cleanup'
curl_example.cpp:(.text+0x15c): undefined reference to
`curl_global_cleanup'
collect2: ld returned 1 exit status
```

The problem is that the linker doesn't' know what library to link in.

```
[bentley@voyager cis29_test]$ g++ curl_example.cpp -Wall -o
curl_example -lcurl
```

Notice that the compiler knew where to find the include files and the library files. That can be facilitated by including the appropriate directories in the $PATH and $LD_LIBRARY_PATH environment variables.

This execution makes use of command-line arguments.

```
[bentley@voyager cis29_test]$ curl_example
http://www.stroustrup.com/glossary.html stroupstrup_glossary.html
Writing 1127 bytes to stroupstrup_glossary.html
Writing 1368 bytes to stroupstrup_glossary.html
Writing 1368 bytes to stroupstrup_glossary.html
Writing 1368 bytes to stroupstrup_glossary.html
Writing 1368 bytes to stroupstrup_glossary.html
Writing 1368 bytes to stroupstrup_glossary.html
Writing 1368 bytes to stroupstrup_glossary.html
Writing 1368 bytes to stroupstrup_glossary.html
…
Writing 1368 bytes to stroupstrup_glossary.html
Writing 1368 bytes to stroupstrup_glossary.html
Writing 1368 bytes to stroupstrup_glossary.html
Writing 1635 bytes to stroupstrup_glossary.html
Total bytes downloaded: 168290
```

Here is the transferred file in the current directory.

```
[bentley@voyager cis29_test]$ ll stroupstrup_glossary.html
-rw-r--r-- 1 bentley cisStaff 168290 Dec 16 16:23
stroupstrup_glossary.html
```

Templates

Function Templates

A function template is a feature in the language that allows the user to define a pattern for a function. Function templates are also called generic functions. The primary reason from writing function templates is to avoid having to write several overloaded versions of a function which performs the same logic on different types. For example, if you needed a function, max to return the maximum value of two numbers, you would have to write a version for int, one for floats, doubles, etc. Not to mention overloaded versions for your own class types. You will end up with:

```
int          max(int n1,int n2);
float        max(float n1,float n2);
double       max(double n1 ,double n2);
long         max(long n1,long n2);
char         max(char n1,char n2);
my_type      max(my_type n1,my_type n2);
```

The logic for each function would be the same:

```
{
  return a > b ? a : b ;
}
```

Example 1 – Function Templates

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

template <typename T> T Max(T a, T b)
{
    return (a > b ? a : b);
}

int main(void)
{
    // Testing primitive types
    cout << Max(3,4) << endl;
    cout << Max(4.55,1.23) << endl;
    cout << Max('a','d') << endl;
    cout << Max('N',Max('H','U')) << endl;
    cout << Max('N',Max('H','U')) << endl;
    // cout << Max(static_cast<short>(2),3) << endl;  // ERROR
    cout << Max(static_cast<short>(2), static_cast<short>(3))
         << endl << endl;

    // Testing strings
    string s1("Dog");
    string s2("Cat");
    string s3("Horse");
    cout << Max(s1,s2) << endl;
```

```
    cout << Max(s2,s3) << endl << endl;

    // Testing char arrays
    char array1[16], array2[16], array3[16];
    strcpy(array1,"dog");
    strcpy(array2,"cat");
    strcpy(array3,"horse");
    cout << Max(array1,array2) << endl;
    cout << Max(array2,array3) << endl;
    cout << reinterpret_cast<long>(array1) << endl;
    cout << reinterpret_cast<long>(array2) << endl;
    cout << reinterpret_cast<long>(array3) << endl;
}
```

****** Output ******

```
4
4.55
d
U
U
3

Dog
Horse

dog
cat
7012024
7012008
7011992
```

Comments

A function template

- begins with the keyword, template.
- This is followed by angle brackets that represent the different types used in the template. The types are identified with the keyword, typename. In the old days, the keyword class was used for this.
- Next comes a *normal-looking* function heading. In place of function argument types and return types, the typename(s) is/are used.
- The rest of the function looks *normal*.

When the function template is called, the compiler instantiates a unique version of the function using the argument types. This instantiation is called a template function.

Example 2 – Function Templates with an overloaded function

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;


template <typename T> T Max(T a, T b)
{
    return (a > b ? a : b);
 }

 char* Max(char* a, char* b)
 {
   return ((strcmp(a,b) > 0) ? a : b);
 }


 int main(void)
 {
     // Testing primitive types
     cout << Max(3,4) << endl;
     cout << Max(4.55,1.23) << endl;
     cout << Max('a','d') << endl;
     cout << Max('N',Max('H','U')) << endl;
     cout << Max('N',Max('H','U')) << endl;
     // cout << Max(static_cast<short>(2),3) << endl;  // ERROR
     cout << Max(static_cast<short>(2), static_cast<short>(3)
          << endl << endl;

     // Testing strings
     string s1("Dog");
     string s2("Cat");
     string s3("Horse");
     cout << Max(s1,s2) << endl;
     cout << Max(s2,s3) << endl << endl;

     // Testing char arrays
     char array1[16], array2[16], array3[16];
     strcpy(array1,"dog");
     strcpy(array2,"cat");
     strcpy(array3,"horse");
     cout << Max(array1,array2) << endl;
     cout << Max(array2,array3) << endl;
     cout << reinterpret_cast<long>(array1) << endl;
     cout << reinterpret_cast<long>(array2) << endl;
     cout << reinterpret_cast<long>(array3) << endl;
 }
```

****** Output ******

```
4
4.55
d
```

```
U
U
3

Dog
Horse

dog
horse
7012024
7012008
7011992
```

Example 3 – A Function Template that always returns a double

```
#include <iostream>
using namespace std;

template <typename Z> double half(Z n)
{
   return static_cast<double>(n/2.);
}

int main(void)
 {
   cout << half(3) << endl;
   cout << half(4.55) << endl;
   cout << half(static_cast<short>(2)) << endl;
   cout << half(static_cast<long>(19)) << endl;
   cout << half(1/2) << endl;
   cout << half('x') << endl;
 }
```

****** Output ******

```
1.5
2.275
1
9.5
0
60
```

Example 4 – A Function Template with an array argument

```
#include <iostream>
#include <cstring>
using namespace std;

template <typename T> double average(T* n,int size)
{
    double sum = 0;
    for (int i = 0; i < size; i++) sum += *(n+i);
```

```
    return sum/size;
}

int main()
{
    int x[5] = {2,4,7,8,9};
    double y[3] = {7.8,9.1,0.9};
    unsigned short z[4] = {2,4,6,8};
    const char cstring[] = "ABCD";
    cout << average(x,5) << endl;
    cout << average(y,3) << endl;
    cout << average(z,4) << endl;
    cout << average(cstring,strlen(cstring));
}
```

****** Output ******

```
6
5.93333
5
66.5
```

Example 5 – A Function Template using two types

```
#include <iostream>
using namespace std;

template <typename X, typename Y> void print_em(X a, Y b)
{
    cout.setf(ios::right,ios::adjustfield);
    cout.width(10);
    cout << static_cast<long>(a);
    cout.precision(2);
    cout.setf(ios::showpoint);
    cout.width(10);
    cout << static_cast<double>(b) << endl;
 }

 int main(void)
 {
   print_em(3,4);
   print_em(3,5.7);
   print_em(5.11,9);
   print_em(static_cast<short>(3),7.777);
   print_em(5,static_cast<float>(3.456));
   print_em('A',5);
   print_em(5,'A');
 }
```

****** Output ******

```
         3       4.0
         3       5.7
```

```
     5       9.0
     3       7.8
     5       3.5
    65       5.0
     5       65.
```

Example 6 – A Function Template with a user defined type

```
#include <iostream>
#include <string>
using namespace std;

class Card
{
private:
    int pips;
    int suit;
public:
    Card(int n = 0) : pips(n % 13), suit(n / 13)
    { }

    bool operator>(const Card& c) const
    {
        return pips > c.pips;
    }
    static const string pips_name[13];
    static const string suit_name[4];
    friend ostream& operator<<(ostream&, const Card&);
};

const string Card::pips_name[13] =
   {"two","three","four","five","six","seven",
    "eight","nine","ten","jack","queen","king","ace"};
const string Card::suit_name[4] =
   {"clubs","diamonds","hearts","spades"};

ostream& operator<<(ostream& out, const Card& card)
{
    out << Card::pips_name[card.pips] << " of " <<
           Card::suit_name[card.suit];
    return out;
}


template <typename T> const T& Max(const T& a, const T& b)
{
    return (a > b) ? a : b;
}

int main(void)
{
    cout << Max(3,4) << endl;
    Card c1(23), c2(9);
    cout << c1 << endl;
    cout << c2 << endl;
```

```
    cout << Max(c1,c2) << endl;
}
```

****** Output ******

```
4
queen of diamonds
jack of clubs
queen of diamonds
```

Example 7 – A Function Template in header files

```
#ifndef FT7_H
#define FT7_H

#include <iostream>

template <typename U> void swap(U& a,U& b)
{
    U temp;
    temp = a;
    a = b;
    b = temp;
}

template <typename T> void sort(T* a,int size)
{
    int i,j;
    for (i = 1; i < size; i++)
        for (j = 0; j < i; j++)
            if ( a[i] < a[j] ) swap(a[i],a[j]);
}

template <typename V> void arrayPrint(const V* a,int size)
{
    int i;
    for (i = 0; i < size; i++) std::cout << a[i] << std::endl;
    std::cout << std::endl;
}

#endif
```

```
#include "ft7.h"

#include <iostream>
using namespace std;

class fraction
{
private:
    int numer,denom;
public:
    fraction(int n = 0, int d = 1) : numer(n), denom(d) {}
```

```
    void assign(int n, int d)
    {
        numer = n;
        denom = d;
    }
    int operator<(fraction& f);
    friend ostream& operator<<(ostream& s, const fraction& f);
};

int fraction::operator<(fraction& f)
{
    return (static_cast<float>(numer)/denom <
static_cast<float>(f.numer)/f.denom);
}

ostream& operator<<(ostream& s,const fraction& f)
{
    s << f.numer << '/' << f.denom;
    return s;
}


class Card
{
protected:
    int pips;
    int suit;
public:
    Card(int n = 0) : pips(n % 13), suit(n / 13)
    { }

    bool operator<(const Card& c) const
    {
        return pips < c.pips;
    }
    static const string pips_name[13];
    static const string suit_name[4];
    friend ostream& operator<<(ostream&, const Card&);
};

const string Card::pips_name[13] = {"two","three","four",”five”,
"six","seven","eight","nine","ten","jack","queen","king","ace"};
const string Card::suit_name[4] =
    {"clubs","diamonds","hearts","spades"};

ostream& operator<<(ostream& out, const Card& card)
{
    out << Card::pips_name[card.pips] << " of " <<
 Card::suit_name[card.suit];
    return out;
}


class PinocleCard : public Card
{
```

```
public:
    PinocleCard(int n = 0) : Card(n)
    {
        pips = n % 6 + 7;
        suit = n / 2 % 4;
    }
    int operator<(PinocleCard&);
};

int PinocleCard::operator<(PinocleCard& c)
{
    if (pips != 8 && c.pips != 8) return (pips < c.pips);
    else if (pips == 8 && c.pips != 12) return 0;
    else if (c.pips == 8 && pips != 12) return 1;
    else return 0;
}

int main()
{
    // array of int
    int a1[5] = { 3, 5, 1, 9, 94};
    arrayPrint(a1,5);
    sort(a1,5);
    arrayPrint(a1,5);

    // array of double
    double a2[4] = { 3.7, 1.5, -1.1,.9};
    arrayPrint(a2,4);
    sort(a2,4);
    arrayPrint(a2,4);

    // array of char
    char a3[4] = {"hey"};
     arrayPrint(a3,3);
     sort(a3,3);
     arrayPrint(a3,3);

     // array of fractions
     fraction a4[4] {{2,3},{1,2},{3,4},{5,9}};
     arrayPrint(a4,4);
     sort(a4,4);
     arrayPrint(a4,4);

     // array of cards
     Card a5[4] = {47,23,43,1};

     arrayPrint(a5,4);
     sort(a5,4);
     arrayPrint(a5,4);

     // array of PinocleCards
     PinocleCard a6[6] = {32,18,41,10,13,27};
     arrayPrint(a6,6);
     sort(a6,6);
     arrayPrint(a6,6);
```

```
}
```

******** Output ********

```
3
5
1
9
94

1
3
5
9
94

3.7
1.5
-1.1
0.9

-1.1
0.9
1.5
3.7

h
e
y

e
h
y

2/3
1/2
3/4
5/9

1/2
5/9
2/3
3/4

ten of spades
queen of diamonds
six of spades
three of clubs

three of clubs
six of spades
ten of spades
queen of diamonds

jack of clubs
nine of diamonds
ace of clubs
king of diamonds
ten of hearts
queen of diamonds
```

```
nine of diamonds
jack of clubs
queen of diamonds
king of diamonds
ten of hearts
ace of clubs
```

Class Templates

A class template is a class definition that contains a generic type, and one or more function templates. Just like function templates, instantiations of a class template are called template classes. Class templates are commonly used with container classes.

Example 8 – class template

```
#include <iostream>
#include <string>
#include <typeinfo>
using namespace std;

template <typename T>
class Thing
{
private:
    T x;
public:
    Thing();
    Thing(T);
    Thing(const Thing<T>&);
    T get() const;
    operator T() const;
};

template <typename T>
Thing<T>::Thing() : x(0) {}

template <typename T>
Thing<T>::Thing(T n) : x(n) {}

template <typename T>
Thing<T>::Thing(const Thing<T>& t) : x(t.x) {}

template <typename T>
T Thing<T>::get() const
{
    return x;
}

template <typename T>
Thing<T>::operator T() const
{
    return x;
```

```
}

template <typename T>
ostream& operator<<(ostream& s, const Thing<T>& t)
{
    return s << t.get();
}

int main(void)
{
    Thing<int> t1;
    cout << "t1=" << t1 << endl;

    Thing<int> t2(18);
    cout << "t2=" << t2 << endl;

    Thing<double> t3(1.28);
    cout << "t3=" << t3 << endl;

    Thing<double> t4(t3);
    cout << "t4=" << t4 << endl;

    cout << "(t2.get() + t3.get()) = " << (t2.get() + t3.get()) <<
            endl;
    cout << "t2 + t3 = " << t2 + t3 << endl;

    Thing<char> t5('z');
    cout << "t5=" << t5 << endl;

    Thing<string> t6("howdy");
    cout << "t6=" << t6 << endl;

    cout << t6.get()[2] << endl;
}
```

******** Output ********

```
t1=0
t2=18
t3=1.28
t4=1.28
(t2.get() + t3.get()) = 19.28
t2 + t3 = 19.28
t5=z
t6=howdy
w
```

Example 9 – class template: a generic array

```
#include <iostream>
#include <cstdlib>
using namespace std;

```

```
template <typename T>
class Array
{
private:
    T* ptrT;
    int size;
public:
    Array():  ptrT(0), size(0) {}
    Array(int);
    T& operator[](int);
};

template <typename T>
Array<T>::Array(int n) : ptrT(new T[n]), size(n)
{
    for (int i = 0; i < size; i++) ptrT[i] = 0;
}

template <typename T>
T& Array<T>::operator[](int index)
{
    if (index < 0 || index >= size)
    {
        cerr << "invalid Array index\n";
        return *ptrT;
    }
    else return ptrT[index];
}

class Fraction
{
private:
    int numer, denom;
public:
    Fraction(int z = 0) : numer(z), denom(0) {}
    Fraction(int n, int d) : numer(n), denom(d) {}
    friend ostream& operator<<(ostream&, const Fraction&);
};

ostream& operator<<(ostream& s, const Fraction& f)
{
    return s << f.numer << '/' << f.denom;
}

int main(void)
{
    int i;
    Array<int> a1(3);
    for (i = 0; i < 3; i++) a1[i] = (2 * i);
    for (i = 0; i < 3; i++) cout << a1[i] << endl;

    Array<float> a2(3);
    for (i = 0; i < 3; i++) a2[i] = (2.7 * i);
    for (i = 0; i < 3; i++) cout << a2[i] << endl;

```

```
    Array<char> a3(6);
    for (i = 0; i < 3; i++) a3[i] = 65+3*i;
    for (i = 0; i < 3; i++) cout << a3[i] << endl;

    Array<Fraction> a4(3);
    a4[0] = Fraction(3,4);
    a4[1] = Fraction(1,2);
    a4[2] = Fraction(5,8);
    for (i = 0; i < 3; i++) cout << a4[i] << endl;
}
```

****** Output ******

```
0
2
4
0
2.7
5.4
A
D
G
3/4
1/2
5/8
```

Example 10 – a container and iterator class template

```
#include <iostream>
#include <string>
using namespace std;

template <typename T, const int size = 7> class Iterator; // Forward declaration

template <typename T, const int size = 7>
class Container
{
    T array[size];
 public:
    friend class Iterator<T, size>;
 };

 template <typename T, const int size>
 class Iterator
 {
    Container<T,size>& ref;
    int index;
 public:
    Iterator(Container<T,size>& cr)
        : ref(cr), index(0)
    {}

    void reset()
```

```
    {
        index = 0;
    }

    // prefix increment operator
    Iterator<T,size>& operator++()
    {
        if(index < size - 1)
            index++;
        else
            index = size;
        return *this;   // indicates end of list
    }

    // dereferencing operator
    T& operator*()
    {
        return ref.array[index];
    }

    // conversion operator
    operator bool() const
    {
        return index < size;
    }
};

class X
{
    int i;
public:
    X(int I = 0) : i(I) {}
    X& operator=(const int& I)
    {
        i = I;
        return *this;
    }

    friend ostream& operator<<(ostream& out, const X& object)
    {
        out << object.i;
        return out;
    }
};
class Fraction
{
    int numer, denom;
public:
    Fraction(int n = 0, int d = 1) : numer(n),denom(d) {}
    Fraction& operator=(const Fraction& f)
    {
        numer = f.numer;
        denom = f.denom;
        return *this;
    }
```

```
    friend ostream& operator<<(ostream& out, const Fraction& object)
    {
        out << object.numer << '/' << object.denom;
        return out;
    }
};


class Card
{
private:
    int pips, suit;
    static const string SuitName[4];
    static const string PipsName[13];
public:
    Card(int n = 0) : pips(n%13), suit(n/13) {}
    Card& operator=(const Card& c)
    {
        pips = c.pips;
        suit = c.suit;
        return *this;
    }
    friend ostream& operator<<(ostream& out, const Card& object)
    {
        out <<PipsName[object.pips] << " of " << SuitName[object.suit];
        return out;
    }
};

const string Card::SuitName[4] =
      {"clubs","diamonds","hearts","spades"};
const string Card::PipsName[13] =
   "two","three","four","five","six","seven",
   "eight","nine","ten","jack","queen","king","ace"};

int main()
{
    Container<X> xC;
    Iterator<X> iX(xC);
    for(auto i = 0; i < 7; i++)
    {
        *iX = i;
        ++iX;
    }
    iX.reset();
    do cout << *iX << endl;
    while(++iX);

    Container<Fraction,3> fractionContainer;
    Iterator<Fraction,3> fractionIterator(fractionContainer);
    for(auto i = 0; i < 3; i++)
    {
        *fractionIterator = Fraction(i+1,i+2);
```

```
        ++fractionIterator;
    }
    fractionIterator.reset();
    do cout << *fractionIterator << endl;
    while(++fractionIterator);

    Container<Card,5> CardC;
    Iterator<Card,5> itCard(CardC);
    for(auto i = 0; i < 5; i++)
    {
        *itCard = Card(3*i+5);
        ++itCard;
    }
    itCard.reset();
    do cout << *itCard << endl;
    while(++itCard);
}
```

****** Output ******

```
0
1
2
3
4
5
6
1/2
2/3
3/4
seven of clubs
ten of clubs
king of clubs
three of diamonds
six of diamonds
```

Example 11 – a generic file I/O class

```
#include <fstream>
#include <iostream>
#include <string>
using namespace std;

template <class T>
class IO
{
private:
    fstream file;
    int eof()
    {
        return file.eof();
    }
 public:
    IO(const string& filename = "temp.bin")
    {
        file.open(filename,ios_base::in | ios_base::out |
```

```
                             ios_base::trunc | ios_base::binary);
    }
    void rewind()
    {
        file.seekg(0L);
        file.seekp(0L);
        file.clear();
    }
    IO& operator>>(T& t);
    IO& operator<<(const T& t);
    operator bool()
    {
        if (!file) return false;
        else return true;
    }
};

template <class T>
IO<T>& IO<T>::operator<<(const T& t)
{
    file.write((char*) &t,sizeof(T));
    return *this;
}

template <class T>
IO<T>& IO<T>::operator>>(T& t)
{
    file.read((char*)&t,sizeof(T));
    return *this;
}

class A
{
    int a;
public:
    friend istream& operator>>(istream& in, A& AA);
    friend ostream& operator<<(ostream& out, A& AA);
};

istream& operator>>(istream& in, A& AA)
{
    cout << "Enter an int for an A object => ";
    return in >> AA.a;
}

ostream& operator<<(ostream& out, A& AA)
{
    return out << AA.a;
}

class B
{
protected:
    double bl;
    char b2[16] ;
```

```
    long b3;
public:
    friend istream& operator>>(istream& in, B& BB);
    friend ostream& operator<<(ostream& out, B& BB);
};

istream& operator>>(istream& in, B& BB)
{
    cout << "Enter double, char* and long for a B object => ";
    return in >> BB.bl >> BB.b2 >> BB.b3;
}

ostream& operator<<(ostream& out, B& BB)
{
    return out << BB.bl << ' ' << BB.b2 << ' ' << BB.b3;
}

int main(void)
{
    A apple;
    IO<A> appleIO("apple.bin");
    cin >> apple;
    appleIO << apple;
    cin >> apple;
    appleIO << apple;

    B banana;
    IO<B> bananaIO("banana.bin");
    cin >> banana;
    bananaIO << banana;
    cin >> banana;
    bananaIO << banana;
    cin >> banana;
    bananaIO << banana;

    int temp;
    IO<int> intIO;
    intIO << rand() % 100;
    intIO << rand() % 100;
    intIO << rand() % 100;
    intIO << rand() % 100;
    intIO << rand() % 100;

    appleIO.rewind();
    while (appleIO >> apple) cout << apple << endl;
    bananaIO.rewind();
    while (bananaIO >> banana) cout << banana << endl;
    intIO.rewind();
    while (intIO >> temp) cout << temp << endl;
}

```

****** Output ******

```
Enter an int for an A object =>123
Enter an int for an A object =>456
Enter double, char* and long for a B object =>1.1 Hey 98765
Enter double, char* and long for a B object =>2.2 you 87654
Enter double, char* and long for a B object =>3.3 guys 76543
123
456
1.1 Hey 98765
2.2 you 87654
3.3 guys 76543
41
67
34
0
69
```

Example 12 – a generic Linked List

```
#include <iostream>
#include <string>
#include <cstdlib>
using namespace std;

template<typename T>
class Node
{
    T  data_;
    Node* next_;
    Node(const Node&) = delete;              // disable copy ctor
    Node& operator=(const Node&) = delete;  // disable ass operator
public:
    Node();
    Node(T d, Node* n);
    const T& data() const;
    T& data();
    Node* next() const;
    Node*& next();
};

template<typename T> Node<T>::Node()
    : data_(), next_(0)
{}

template<typename T> Node<T>::Node(T d, Node* n)
    : data_(d), next_(n)
{}

template<typename T> const T& Node<T>::data() const
{
    return data_;
}

template<typename T> T& Node<T>::data()
{
```

```
    return data_;
}

template<typename T> Node<T>* Node<T>::next() const
{
    return next_;
}

template<typename T> Node<T>*& Node<T>::next()
{
    return next_;
}

template<typename T> ostream& operator<<(ostream& out, const Node<T>& N)
{
    out << N.data();
    return out;
}

template<typename T> class List
{
    Node<T>* top_;
    List(const List&) = delete;              // disable copy ctor
    List& operator=(const List&) = delete;  // disable ass operator
public:
    List();
    ~List();
    void push(T object);
    T pop();
    const Node<T>* top() const;
    bool remove(T object);
    const Node<T>* find(T object) const;
};

template<typename T>
ostream& operator<<(ostream& out, const List<T>& L)
{
    const Node<T>* ptr = L.top();
    while (ptr)
    {
        out << (*ptr) << '\t';
        ptr = ptr -> next();
    }
    return out;
}

template<typename T> List<T>::List()
    : top_(0)
{}

template<typename T> List<T>::~List()
{
    Node<T>* ptr = top_;
    while (ptr)
```

```
    {
        top_ = top_->next();
        delete ptr;
        ptr = top_;
    }
}

template<typename T> void List<T>::push(T object)
{
    Node<T>* ptr = new Node<T>(object, top_);
    top_ = ptr;
}

template<typename T> const Node<T>*  List<T>::top() const
{
    return top_;
}

template<typename T> T List<T>::pop()
{
    Node<T>* ptr = top_;
    top_ = top_ -> next();
    T data = ptr->data();
    delete ptr;
    return data;
}

template<typename T> const Node<T>* List<T>::find(T object) const
{
    const Node<T>* ptr = top();
    while (ptr)
    {
        if (ptr->data() == object)
        {
            return ptr;
        }
        ptr = ptr->next();
    }
    return 0;
}

template<typename T> bool List<T>::remove(T object)
{
    if (!find(object))
    {
        cerr << object << " not found\n";
        return false;
    }
    Node<T>* ptr2current = top_;
    Node<T>* ptr2previous = top_;
    if (top_->data() == object)
    {
        top_ = top_ -> next();
        delete ptr2current;
        return true;
```

```
    }
    while (ptr2current)
    {
        ptr2current = ptr2current->next();
        if (ptr2current->data() == object)
        {
            ptr2previous->next() = ptr2current->next();
            delete ptr2current;
            return true;
        }
        ptr2previous = ptr2current;
    }
    return false;
}

class Card
{
private:
    int pips, suit;
    static const string SuitName[4];
    static const string PipsName[13];
public:
    Card() : pips(rand()%13), suit(rand()%4) {}
    Card(int n) : pips(n%13), suit(n%4) {}
    friend ostream& operator<<(ostream& out, const Card& object)
    {
        out << PipsName[object.pips] << " of "
            << SuitName[object.suit];
        return out;
    }
};

const string Card::SuitName[4] =
   {"clubs","diamonds","hearts","spades"};
const string Card::PipsName[13] =
   {"two","three","four","five","six","seven",
    "eight","nine","ten","jack","queen","king","ace"};


int main()
{
    List<int> Lint;
    Lint.push(2);
    Lint.push(4);
    Lint.push(6);
    Lint.push(8);
    Lint.push(10);
    cout << Lint << endl;
    Lint.pop();
    cout << Lint << endl;

    Card C1;
    Card C2;
    Card C3(25);
    Card C4;
```

```
    Card C5;
    List<Card> LCard;
    LCard.push(C1);
    LCard.push(C2);
    LCard.push(C3);
    LCard.push(C4);
    LCard.push(C5);
    cout << LCard << endl;

    List<string> Lstring;
    Lstring.push("day");
    Lstring.push("nice");
    Lstring.push("very");
    Lstring.push("a");
    Lstring.push("Have");
    cout << Lstring << endl;
    Lstring.remove("very");
    cout << Lstring << endl;
}
```

****** Output ******

```
10      8       6       4       2
8       6       4       2
ace of hearts   nine of clubs   ace of diamonds five of clubs   four of
spades
Have    a       very    nice    day
Have    a       nice    day
```

Hash Tables

A hash table is an abstract data type that uses an array for storage. It makes use of a mapped key as an index. A hash table uses a hash function to translate a value into an index that can you used with an array. The location in the array where the data is stored is referred to as a bucket or slot.

Example 1 – First hash table example

This example demonstrates an array of strings stored in a *hash table*. The *hash table*, itself, is an array of string pointers. The *hash function*, hash, converts each string into an unsigned int value. The unsigned int return value is then used as an index in the array of string pointers. Notice, that some of the string arguments with produce the same return value. This situation is referred to as a *collision*. In this example when a *collision* occurs, the target string is not able to be stored in the *hash table*.

```
#include <iostream>
#include <string>
#include <cctype>
using namespace std;

unsigned hash(const string&);

const unsigned NumberOfBuckets = 10;

int main()
{
    string animals[NumberOfBuckets] =
        {"monkey","dog","cat","horse","pig","goat","hippo",
         "dinosaur","walrus","manatee"};
    string* ptr2strings[NumberOfBuckets] = {nullptr};

    for (auto i = 0u; i < NumberOfBuckets; i++)
    {
        auto index = ::hash(animals[i]);

        // if the index is unused, use it
        if (ptr2strings[index] == nullptr)
        {
            ptr2strings[index] = new string(animals[i]);
        }
        else
        {
            cout << "Can't store " << animals[i] << ". Bucket "
                 << index << " is already taken\n";
        }
    }
    for (auto i = 0u; i < NumberOfBuckets; i++)
    {
        cout << i << ' '
             << (ptr2strings[i] ? *ptr2strings[i] : "" )<< endl;
```

```
    }
}


unsigned hash(const string& str)
{
    static string alphabet = "abcdefghijklmnopqrstuvwxyz";
    size_t pos;
    unsigned sum = 0;
    for (auto i = 0u; i < str.size(); i++)
    {
        pos = alphabet.find(tolower(str[i]));
        sum += pos;
    }

    return sum % NumberOfBuckets;
}
```

****** Output ******

```
Can't store goat. Bucket 9 is already taken
Can't store hippo. Bucket 9 is already taken
Can't store dinosaur. Bucket 3 is already taken
0 horse
1 cat
2 manatee
3 dog
4
5
6
7 monkey
8 walrus
10  pig
```

Example 2 – Use a hash table to store a dictionary

This example simulates an “Unscramble” game in which scrambled words are unscrambled by using a hash table to find the word with the same hashed value. Note, in this solution, *collisions* are also not handled.

```
#include <iostream>
#include <string>
#include <cctype>
#include <fstream>
#include <cstdlib>
#include <stdexcept>
using namespace std;

unsigned hash(const string&);

class Dictionary
{
    string** ptrWords;
public:
```

```
    Dictionary(const string& wordfile);
    ~Dictionary();
    string findScrambledWord(const string& word);
    static const unsigned NumberOfBuckets;
};

const unsigned Dictionary::NumberOfBuckets = 100000;

Dictionary::Dictionary(const string& wordfile)
    : ptrWords(new string*[NumberOfBuckets])
{
    ifstream fin(wordfile.c_str());
    if (!fin)
    {
        throw (invalid_argument(string("Can't find file ") +
                                         wordfile));
    }
    string word;
    unsigned numberOfBucketsUsed = 0;
    unsigned numberOfWordsNotStored = 0;
    unsigned numberOfWords = 0;

    for (auto i = 0u; i < NumberOfBuckets; i++)
    {
        ptrWords[i] = nullptr;
    }

    // create hash table
    while (fin >> word)
    {
        ++numberOfWords;
        auto index = ::hash(word);
        if (ptrWords[index])
        {
            // bucket already taken
            ++numberOfWordsNotStored;
        }
        else
        {
            ptrWords[index] = new string(word);
            numberOfBucketsUsed++;
        }
    }
    cout << "number of buckets used = " << numberOfBucketsUsed
         << endl;
    cout << "number of words not stored = "
         << numberOfWordsNotStored << endl;
    cout << "number of words = " << numberOfWords << endl;
}

Dictionary::~Dictionary()
{
    for (auto i = 0u; i < NumberOfBuckets; i++)
    {
        if (ptrWords[i])
```

```
        {
            delete ptrWords[i];
        }
    }
    delete [] ptrWords;
    ptrWords = nullptr;
}

string Dictionary::findScrambledWord(const string& word)
{
    auto index = ::hash(word);
    if (ptrWords[index])
        return *(ptrWords[index]);
    else
        return string("");
}

int main()
{
    string scrambledWord;
    try
    {
        Dictionary Words("c:/temp/words");

        while (1)
        {
            cout << "Enter a scrambled word (\"quit\" to exit)=> ";
            cin >> scrambledWord;
            if (scrambledWord == "quit")
                return 0;
            else
                 cout << "unscramble = "
                 << Words.findScrambledWord(scrambledWord) << endl;
        }
    }
    catch (const invalid_argument& error)
    {
        cout << error.what() << endl;
        exit(-1);
    }
}

unsigned hash(const string& str)
{
    static unsigned primes[26] = {2, 3, 5, 7, 11, 13, 17, 19, 23,
                      29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,
                      73, 79, 83, 89, 97, 101};
    unsigned product = 1;
    for (auto i = 0u; i < str.size(); i++)
    {
        product *= primes[tolower(str[i])-'a'];
    }
    return product % Dictionary::NumberOfBuckets;
}
```

****** Output ******

```
number of buckets used = 19735
number of words not stored = 4320
number of words = 24055
Enter a scrambled word ("quit" to exit) => ksa
unscramble = ask
Enter a scrambled word ("quit" to exit) => bilrray
unscramble = library
Enter a scrambled word ("quit" to exit) => hsear
unscramble = Asher
Enter a scrambled word ("quit" to exit) => fntcunoi
unscramble = function
Enter a scrambled word ("quit" to exit) => asked
unscramble =
Enter a scrambled word ("quit" to exit) => yranoitcid
unscramble = combatted
Enter a scrambled word ("quit" to exit) => belramcs
unscramble = scramble
Enter a scrambled word ("quit" to exit) => quit
```

Notes

hsear *was supposed to be share*
yranoitcid *was supposed to be dictionary*
belramcs *was supposed to be scramble (but was not found)*

Standard Template Library

The STL consists of

- containers (in the form of class templates),
- iterators - to be used "like" pointers in a container
- function objects (or functors) - A class object that can act like a function.
- algorithms - functions applied to containers.

Containers

Types of containers

Sequential

A sequential container is one in which elements are accessed sequentially. That access is usually performed using an iterator.

Sorted Associative

An associative container is one in which elements are accessed using a key.

Adaptors

Adaptors are adaptations of specific sequential containers for specific purposes.

Unsorted Associative

Unsorted associative containers are implemented using hashing algorithms.

Container	**Type**	**Purpose**
array	sequential	A C-style fixed size replacement
vector	sequential	All-purpose, variable size
list	sequential	Linked-list, double ended
forward_list	sequential	Linked-list, single ended
deque	sequential	Like a vectors with access at ends
queue	Adapter	Implements FIFO
priority_queue	Adapter	Implements FIFO with priority
stack	Adapter	Implements LIFO
set	Sorted associative	Similar to mathematical set
multi_set	Sorted associative	A set with duplicate values
map	Sorted associative	Key-value pairs
multimap	Sorted associative	Key-value pairs with duplicate keys
unordered_set	Unsorted associative	set implemented as hash table
unordered_multiset	Unsorted associative	Multiset implemented as hash table
unordered_map	Unsorted associative	map implemented as hash table
unordered_multimap	Unsorted associative	multimap implemented as hash table
bitset	N/A	Bit manipulators replacement

array

The array container is a replacement for the fixed size C array. This sequence container surfaced in C++ 11. The array container exhibits the indexing behaviors of a C array. To declare an array class object, class template syntax is used and only the default constructor is available. The array container requires the <array> header file.

Examples

array<int,10> object; // instantiates an array of 10 int
array<dog,5> hounds; // instantiates an array of 10 dogs

Iterator Functions

begin

Returns an iterator pointing to the first element of the array

```
iterator begin() noexcept;
const_iterator begin() const noexcept;
```

end

Returns an iterator pointing to the *non-existing* element beyond the end of the array

```
iterator end() noexcept;
const_iterator end() const noexcept;
```

rbegin

Returns a reverse iterator pointing to the last element in the array

```
reverse_iterator rbegin() noexcept;
const_reverse_iterator rbegin() const noexcept;
```

rend

Returns a reverse iterator pointing to the *non-existing* element in front of the first element of the array

```
reverse_iterator rend() noexcept;
const_reverse_iterator rend() const noexcept;
```

cbegin

Returns a const iterator pointing to the first element of the array

```
const_iterator begin() const noexcept;
```

cend

Returns a const iterator pointing to the *non-existing* element beyond the end of the array

```
const_iterator end() const noexcept;
```

crbegin

Returns a const reverse iterator pointing to the last element of the array

```
const_reverse_iterator rbegin() const noexcept;
```

crend

Returns a const reverse iterator pointing to the non-existing element in front of the first element of the array

```
const_reverse_iterator rend() const noexcept;
```

Capacity Functions

size

Returns the number of elements in the array

```
constexpr size_t size() const noexcept;
```

max_size

Returns the maximum number of elements in an array. This is the same as the size.

```
constexpr size_t max_size() const noexcept;
```

empty

Returns whether the array is empty – has size 0.

```
constexpr bool empty() const noexcept;
```

Access Functions

at

Returns element at position

```
value_type& at (size_t position);
const value_type& at (size_t position) const;
```

back

Returns a reference to the last element in the array

```
value_type&  back();
const value_type&  back() const;
```

front

Returns a reference to the first element in the array

```
value_type& front();
const value_type& front() const;
```

data

Returns a pointer to the memory location where a array's first element is stored. Note, array elements are stored in contiguous memory.

```
value_type* data() noexcept;
const value_type* data() const noexcept;
```

Modifier Functions

fill

assigns a value to all elements of an array

```
void fill(const value_type& value);
```

swap

Swaps the contents of two arrays. The arrays must be of the same type and contain the same number of elements.

```
void swap (array& vec);
```

operator[]

Index operator: returns the element at the specified location

```
value_type& operator[] (size_t location);
const value_type& operator[] (size_t location) const;
```

Example 1 – The array container

```
#include <array>
#include <iostream>
#include <cstring>  // for memcpy
using namespace std;

void print_array(const array<int,5>&);
void print_array(const array<char,3>&);

// function template prototype
template <typename T, unsigned long size>
ostream& operator<<(ostream&, const array<T,size>&);

int main()
{
  array<int,5> a1 = {2,3,5,7,11};
  cout << "a1="; print_array(a1);

  array<char,3> a2 = {'h','e','y'};
  cout << "a2="; print_array(a2);

  memcpy(a2.data(),"Wow",a2.size());
  cout << "a2="; print_array(a2);

  array<char,3> a3;
  a3.fill('$');
  a3.swap(a2);
  cout << "a2="; print_array(a2);

  cout << "a1=" << a1 << endl;
}


void print_array(const array<int,5>& arr)
{
  // iterator for loop
  for (auto arrIt = arr.cbegin(); arrIt != arr.cend(); ++arrIt)
       cout << *arrIt << ' ';
  cout << endl;
}

void print_array(const array<char,3>& arr)
{
  // index for loop
  for (auto i = 0u; i < arr.size(); ++i)
       cout << arr[i];
  cout << endl;
}

template <typename T, unsigned long size>
ostream& operator<<(ostream& out, const array<T, size>& object)
{
  // range-based for loop
```

```
   for (const auto& element : object)
         out << element << ' ';
   return out;
}
```

****** Output ******

```
a1=2 3 5 7 11
a2=hey
a2=Wow
a2=$$$
a1=2 3 5 7 11
```

vector

The vector container is a replacement for an array. Unlike an array it has a variable size and can grow and shrink as needed. Further, you may insert new elements into the vector at the beginning or end of the vector . and even in the middle. Vectors may be indexed just like an array. Instead of using pointers to access array elements, iterators are used. The vector container requires the <vector> header file.

Constructors

Default constructor

```
vector();
```

Fill constructors

```
explicit vector(size_type n, const allocator_type& alloc =
                  allocator_type());

vector(size_type n, const value_type& val,
                 const allocator_type& alloc = allocator_type());
```

Range constructor

```
template <class InputIterator>
vector(InputIterator first, InputIterator last,
          const allocator_type& alloc = allocator_type());
```

Copy constructor

```
vector(const vector& x);
```

Move constructor

```
vector(vector&& x);
```

Initializer list constructor

```
vector(initializer_list<value_type> lst,
       const allocator_type& alloc = allocator_type());
```

Iterator Functions

begin

Returns an iterator pointing to the first element of the vector

```
iterator begin() noexcept;
const_iterator begin() const noexcept;
```

end

Returns an iterator pointing to the *non-existing* element beyond the end of the vector

```
iterator end() noexcept;
const_iterator end() const noexcept;
```

rbegin

Returns a reverse iterator pointing to the last element in the vector

```
reverse_iterator rbegin() noexcept;
const_reverse_iterator rbegin() const noexcept;
```

rend

Returns a reverse iterator pointing to the *non-existing* element in front of the first element of the vector

```
reverse_iterator rend() noexcept;
const_reverse_iterator rend() const noexcept;
```

cbegin

Returns a const iterator pointing to the first element of the vector

```
const_iterator begin() const noexcept;
```

cend

Returns a const iterator pointing to the *non-existing* element beyond the end of the vector

```
const_iterator end() const noexcept;
```

crbegin

Returns a const reverse iterator pointing to the last element of the vector

```
const_reverse_iterator rbegin() const noexcept;
```

crend

Returns a const reverse iterator pointing to the non-existing element in front of the first element of the vector

```
const_reverse_iterator rend() const noexcept;
```

Capacity Functions

size

Returns the number of elements in the vector

```
size_t size() const noexcept;
```

capacity

Returns the size allocated for the vector

```
size_t capacity() const noexcept;
```

max_size

Returns the maximum number of elements that a vector can hold

```
size_t max_size() const noexcept;
```

reserve

Change the vector's capacity

```
void reserve(size_t n);
```

resize

Resizes a vector to n elements

```
void resize (size_t n);
void resize (size_t n, const value_type& value);
```

empty

Returns whether the vector is empty

```
bool empty() const noexcept;
```

shrink_to_fit

Changes the capacity to the size of the vector

```
void shrink_to_fit();
```

Access Functions

at

Returns element at position

```
value_type& at (size_t position);
const value_type& at (size_t position) const;
```

back

Returns a reference to the last element in the vector

```
value_type&  back();
const value_type&  back() const;
```

front

Returns a reference to the first element in the vector

```
value_type& front();
const value_type& front() const;
```

data

Returns a pointer to the memory location where a vector’s first element is stored. Note, vector elements are stored in contiguous memory.

```
value_type* data() noexcept;
const value_type* data() const noexcept;
```

Modifier Functions

assign

Assigns new contents to a vector

```
template <class InputIterator>
      void assign(InputIterator beg, InputIterator _end);
void assign(size_type n, const value_type& value);
void assign(initializer_list<value_type> list);
```

clear

Erases a vector. Size becomes 0

```
void clear() noexcept;
```

erase

Erases part of a vector

```
iterator erase(const_iterator p);
iterator erase(const_iterator first, const_iterator last);
```

insert

Inserts elements into a vector at a specified location

```
iterator insert(const_iterator loc, const value_type& value);
iterator insert(const_iterator loc, size_type n, const value_type& value);
template <class InputIterator>
iterator insert(const_iterator loc, InputIterator first, InputIterator last);
iterator insert(const_iterator loc, value_type&& value);
iterator insert(const_iterator loc, initializer_list<value_type> list);
```

push_back

Adds an element to the end of a vector

```
void push_back(const value_type& value);
void push_back(value_type&& value);
```

pop_back

Deletes the last element of a vector

```
void pop_back();
```

swap

Swaps two vectors

```
void swap(vector& vec);
```

Non-member Functions

swap

Swaps two vector

```
void swap(vector& x, vector& y);
```

Member Operators

operator=

The assignment operator: assigns new contents to a vector.

```
vector& operator=(const vector& x);
vector& operator=(vector&& x);
vector& operator=(initializer_list<value_type> list);
```

operator[]

Index operator: returns the element at the specified location

```
value_type& operator[](size_t location);
const value_type& operator[](size_t location) const;
```

Relational operators

`== > < >= <= !=`

Used to compare the contents of two vectors.

Two vectors are equal (==) if their sizes match and each of the corresponding elements match. A less than (<) comparison is made between two vectors by comparing successive elements in order.

Note: these operators, > < >= <= != will be removed in C++20. The <=> operator will be added. More to say about that later.

Example 2 – The vector container

```
#include <vector>
#include <iostream>
using namespace std;

ostream& operator<<(ostream& out, const vector<int>& v);

int main()
{
   // Constructors
   vector<int> v1;
   vector<int> v2(5);
   vector<int> v3(5,19);
   vector<int> v4{2,3,5,7,11,13,17};

   cout << "v2=" << v2 << endl;
   cout << "v3=" << v3 << endl;
   cout << "v4=" << v4 << endl << endl;

   vector<int> v5(v4.begin(),v4.begin()+3);
   vector<int> v6(v4);
   vector<int> v7(move(v4));

   cout << "v4=" << v4 << endl;
   cout << "v5=" << v5 << endl;
   cout << "v6=" << v6 << endl;
   cout << "v7=" << v7 << endl << endl;

   // Capacity functions
   cout << "v7.size()=" << v7.size() << endl;
   cout << "v7.capacity()=" << v7.capacity() << endl;
   cout << "v7.max_size()=" << v7.max_size() << endl;
   v7.reserve(16);
   v7.resize(v7.size()*2);
   cout << "v7.size()=" << v7.size() << endl;
   cout << "v7.capacity()=" << v7.capacity() << endl;
   cout << "v7=" << v7 << endl;
   v7.shrink_to_fit();
```

```
  cout << "v7.size()=" << v7.size() << endl;
  cout << "v7.capacity()=" << v7.capacity() << endl << endl;

  // Access functions
  cout << "v6.front()=" << v6.front() << endl;
  cout << "v6.back()=" << v6.back() << endl;
  cout << "v6.at(3)=" << v6.at(3) << endl;
  int* ptr = v6.data();
  cout << *ptr << ' ' << *(ptr+2) << endl;
  for (auto* p = v6.data(); p < v6.data()+v6.size(); ++p)
        *p *= 2;
  cout << "v6=" << v6 << endl << endl;

  // Modifier functions
  v1.assign({7,6,5,4,3,2,1});
  cout << "v1=" << v1 << endl;
  v2.assign(v1.crbegin(),v1.crend());
  cout << "v2=" << v2 << endl;
  v2.erase(v2.begin()+3);
  cout << "v2=" << v2 << endl;
  v2.insert(v2.begin()+3,15);
  v2.pop_back();
  v2.push_back(30);
  cout << "v2=" << v2 << endl;
  v1.swap(v2);
  cout << "v1=" << v1 << endl;
  cout << "v2=" << v2 << endl << endl;

  // Member operators
  v1[2] = v2[3]*2;
  cout << "v1=" << v1 << endl;
  v1.assign(v2.begin(),v2.begin()+5);
  v1.push_back(13);
  cout << "v1=" << v1 << endl;
  cout << "v2=" << v2 << endl << endl;
  v3 = v1;
  v3.resize(10);
  cout << "v3=" << v3 << endl;
  cout << boolalpha;
  cout << "v1 == v3: " << (v1 == v3) << endl;
  cout << "v1 < v2: " << (v1 < v2) << endl;
  cout << "v1 < v3: " << (v1 < v3) << endl;
  cout << "v2 < v3: " << (v2 < v3) << endl;
}

ostream& operator<<(ostream& out, const vector<int>& v)
{
  for (auto element : v)
        out << element << ' ';
  return out;
}
```

****** Output ******

```
v2=0 0 0 0 0
v3=19 19 19 19 19
v4=2 3 5 7 11 13 17

v4=
v5=2 3 5
v6=2 3 5 7 11 13 17
v7=2 3 5 7 11 13 17

v7.size()=7
v7.capacity()=7
v7.max_size()=2305843009213693951
v7.size()=14
v7.capacity()=16
v7=2 3 5 7 11 13 17 0 0 0 0 0 0 0
v7.size()=14
v7.capacity()=14

v6.front()=2
v6.back()=17
v6.at(3)=7
2 5
v6=4 6 10 14 22 26 34

v1=7 6 5 4 3 2 1
v2=1 2 3 4 5 6 7
v2=1 2 3 5 6 7
v2=1 2 3 15 5 6 30
v1=1 2 3 15 5 6 30
v2=7 6 5 4 3 2 1

v1=1 2 8 15 5 6 30
v1=7 6 5 4 3 13
v2=7 6 5 4 3 2 1

v3=7 6 5 4 3 13 0 0 0 0
v1 == v3: false
v1 < v2: false
v1 < v3: true
v2 < v3: true
```

list

The list container is implemented as a double-ended linked list. It has the advantage of efficient insert and delete operations. The list container requires the <list> header file.

Constructors

Default constructor

```
list();
```

Fill constructors

```
explicit list(size_type n, const allocator_type& alloc =
                  allocator_type());

list(size_type n, const value_type& val,
                 const allocator_type& alloc = allocator_type());
```

Range constructor

```
template <class InputIterator>
list(InputIterator first, InputIterator last,
          const allocator_type& alloc = allocator_type());
```

Copy constructor

```
list(const list& x);
```

Move constructor

```
list(list&& x);
```

Initializer list constructor

```
list(initializer_list<value_type> lst,
       const allocator_type& alloc = allocator_type());
```

Iterator Functions

begin

Returns an iterator pointing to the first element of the list

```
iterator begin() noexcept;
const_iterator begin() const noexcept;
```

end

Returns an iterator pointing to the *non-existing* element beyond the end of the list

```
iterator end() noexcept;
const_iterator end() const noexcept;
```

rbegin

Returns a reverse iterator pointing to the last element in the list

```
reverse_iterator rbegin() noexcept;
const_reverse_iterator rbegin() const noexcept;
```

rend

Returns a reverse iterator pointing to the *non-existing* element in front of the first element of the list

```
reverse_iterator rend() noexcept;
const_reverse_iterator rend() const noexcept;
```

cbegin

Returns a const iterator pointing to the first element of the list

```
const_iterator begin() const noexcept;
```

cend

Returns a const iterator pointing to the *non-existing* element beyond the end of the list

```
const_iterator end() const noexcept;
```

crbegin

Returns a const reverse iterator pointing to the last element of the list

```
const_reverse_iterator rbegin() const noexcept;
```

crend

Returns a const reverse iterator pointing to the non-existing element in front of the first element of the list

```
const_reverse_iterator rend() const noexcept;
```

Capacity Functions

size

Returns the number of elements in the list

```
size_t size() const noexcept;
```

max_size

Returns the maximum number of elements that a list can hold

```
size_t max_size() const noexcept;
```

empty

Returns whether the list is empty

```
bool empty() const noexcept;
```

Access Functions

back

Returns a reference to the last element in the list

```
value_type&  back();
const value_type&  back() const;
```

front

Returns a reference to the first element in the list

```
value_type& front();
const value_type& front() const;
```

Modifier Functions

assign

Assigns new contents to a list

```
template <class InputIterator>
      void assign(InputIterator beg, InputIterator _end);
void assign(size_type n, const value_type& value);
void assign(initializer_list<value_type> lst);
```

clear

Erases a list. Size becomes 0

```
void clear() noexcept;
```

erase

Erases part of a list

```
iterator erase(const_iterator p);
iterator erase(const_iterator first, const_iterator last);
```

insert

Inserts elements into a list at a specified location

```
iterator insert(const_iterator loc, const value_type& value);
iterator insert(const_iterator loc, size_type n, const value_type& value);
```

```
template <class InputIterator>
iterator insert(const_iterator loc, InputIterator first, InputIterator last);
iterator insert(const_iterator loc, value_type&& value);
iterator insert(const_iterator loc, initializer_list<value_type> lst);
```

emplace

Constructs and inserts a new element at a specified location in the list

```
template <class Type>   void emplace(const iterator loc, Type&&... args);
```

push_back

Adds an element to the end of a list

```
void push_back(const value_type& value);
void push_back(value_type&& value);
```

push_front

Adds an element to the beginning of a list

```
void push_front(const value_type& value);
void push_front(value_type&& value);
```

pop_back

Deletes the last element of a list

```
void pop_back();
```

pop_front

Deletes the first element of a list

```
void pop_front();
```

swap

Swaps two lists

```
void swap(list& lst);
```

resize

Changes the size of a list.   If the size is smaller, elements are removed. If the size is larger, elements are added to the list.

```
void resize(size_type n);
void resize(size_type n, const value& val);
```

Example 3 – The list container

```
#include <list>
#include <iostream>
using namespace std;

ostream& operator<<(ostream& out, const list<int>& li);

int main()
{
   // Constructors
   list<int> li1;
   list<int> li2(5);
   list<int> li3(5,19);
   list<int> li4{2,3,5,7,11,13,17};

   cout << "li2=" << li2 << endl;
   cout << "li3=" << li3 << endl;
   cout << "li4=" << li4 << endl << endl;

   //    list<int> li5(li4.begin(),li4.begin()+3);  ERROR
   list<int> li5(li4.begin(),++++++li4.begin());   // ???
   list<int> li6(li4);
   list<int> li7(move(li4));

   cout << "li4=" << li4 << endl;
   cout << "li5=" << li5 << endl;
   cout << "li6=" << li6 << endl;
   cout << "li7=" << li7 << endl << endl;

   cout << "capacity functions" << endl;
   cout << li1.size() << ' ' << boolalpha << li1.empty() << endl;

   cout << endl << "access functions" << endl;
   cout << "li6.front()=" << li6.front() << endl;
   cout << "li6.back()=" << li6.back() << endl;

   cout << endl << "iterator functions" << endl;
   cout << "*li6.begin()=" << *li6.begin() << endl;
   cout << "*++li6.begin()=" << *++li6.begin() << endl;
   cout << "*--li6.end()=" << *--li6.end() << endl;
   cout << "*li6.rbegin()=" << *li6.rbegin() << endl;
   cout << "*++li6.rbegin()=" << *++li6.rbegin() << endl;
   cout << "*--li6.rend()=" << *--li6.rend() << endl;

   cout << endl << "assign" << endl;
   li1.assign({7,6,5,4,3,2,1});
   cout << "li1=" << li1 << endl;
   li2.assign(++li1.crbegin(),--li1.crend());
   cout << "li2=" << li2 << endl;
   li3.assign(5,7);
   cout << "li3=" << li3 << endl << endl;

   cout << "erase" << endl;
```

```
  li2.erase(++li2.begin());
  cout << "li2=" << li2 << endl;
  li1.erase(++li1.begin(),--li1.end());
  cout << "li1=" << li1 << endl << endl;

  cout << "insert" << endl;
  li2.insert(++li2.begin(),3);
  cout << "li2=" << li2 << endl;
  li2.insert(++li2.begin(),li3.begin(),li3.end());
  cout << "li2=" << li2 << endl << endl;

  cout << "push_front / pop_back" << endl;
  li1.push_front(1);
  li1.pop_back();
  cout << "li1=" << li1 << endl << endl;

  cout << "swap" << endl;
  li1.swap(li2);
  cout << "li1=" << li1 << endl << endl;

  cout << "resize" << endl;
  li1.resize(5);
  cout << "li1=" << li1 << endl;
  li1.resize(10);
  cout << "li1=" << li1 << endl;
}

ostream& operator<<(ostream& out, const list<int>& li)
{
  for (auto element : li)
        out << element << ' ';
  return out;
}
```

```
***** OUTPUT ******

li2=0 0 0 0 0
li3=19 19 19 19 19
li4=2 3 5 7 11 13 17

li4=
li5=2 3 5
li6=2 3 5 7 11 13 17
li7=2 3 5 7 11 13 17

capacity functions
0 true

access functions
li6.front()=2
li6.back()=17

iterator functions
*li6.begin()=2
*++li6.begin()=3
*--li6.end()=17
*li6.rbegin()=17
```

```
*++li6.rbegin()=13
*--li6.rend()=2

assign
li1=7 6 5 4 3 2 1
li2=2 3 4 5 6
li3=7 7 7 7 7

erase
li2=2 4 5 6
li1=7 1

insert
li2=2 3 4 5 6
li2=2 7 7 7 7 7 3 4 5 6

push_front / pop_back
li1=1 7

swap
li1=2 7 7 7 7 7 3 4 5 6

resize
li1=2 7 7 7 7
li1=2 7 7 7 7 0 0 0 0 0
```

forward_list

The forward_list container is implemented as a single-ended linked list. Because it only uses a forward pointer, it is usually considered more efficient that a list container. The forward_list container requires the <forward_list> header file. The forward_list container was introduced in C++11.

Constructors

Default constructor

```
forward_list();
```

Fill constructors

```
explicit forward_list (size_type n, const allocator_type& alloc =
                  allocator_type());

forward_list (size_type n, const value_type& val,
                 const allocator_type& alloc = allocator_type());
```

Range constructor

```
template <class InputIterator>
forward_list (InputIterator first, InputIterator last,
          const allocator_type& alloc = allocator_type());
```

Copy constructor

```
forward_list (const vector& x);
```

Move constructor

```
forward_list (vector&& x);
```

Initializer list constructor

```
forward_list (initializer_list<value_type> lst,
       const allocator_type& alloc = allocator_type());
```

Iterator Functions

begin

Returns an iterator pointing to the first element of the forward_list

```
iterator begin() noexcept;
const_iterator begin() const noexcept;
```

before_begin

Returns an iterator pointing to the location before first element of the forward_list

```
iterator begin() noexcept;
```

```
const_iterator begin() const noexcept;
```

end

Returns an iterator pointing to the *non-existing* element beyond the end of the forward_list

```
iterator end() noexcept;
const_iterator end() const noexcept;
```

cbegin

Returns a const iterator pointing to the first element of the forward_list

```
const_iterator begin() const noexcept;
```

cbefore_begin

Returns a const iterator pointing to the location before first element of the forward_list

```
const_iterator begin() const noexcept;
```

cend

Returns a const iterator pointing to the *non-existing* element beyond the end of the forward_list

```
const_iterator end() const noexcept;
```

Capacity Functions

max_size

Returns the maximum number of elements that a forward_list can hold

```
size_t max_size() const noexcept;
```

empty

Returns whether the forward_list is empty

```
bool empty() const noexcept;
```

front

Returns a reference to the first element in the forward_list

```
value_type& front();
const value_type& front() const;
```

Modifier Functions

assign

Assigns new contents to a forward_list

```
template <class InputIterator>
      void assign(InputIterator beg, InputIterator _end);
void assign(size_type n, const value_type& value);
void assign(initializer_list<value_type> lst);
```

clear

Erases a forward_list. Size becomes 0

```
void clear() noexcept;
```

erase_after

Erases part of a list

```
iterator erase_after(const_iterator p);
iterator erase_after(const_iterator first, const_iterator last);
```

insert_after

Inserts elements into a forward_list at a specified location

```
iterator insert_after(const_iterator loc, const value_type& value);
iterator insert_after(const_iterator loc, size_type n, const value_type& va);
template <class InputIterator>
iterator insert_after(const_iterator loc, InputIterator f, InputIterator ls);
iterator insert_after(const_iterator loc, value_type&& value);
iterator insert_after(const_iterator loc, initializer_list<value_type> lst);
```

push_front

Adds an element to the beginning of a forward_list

```
void push_front(const value_type& value);
void push_front(value_type&& value);
```

pop_front

Deletes the first element of a forward_list

```
void pop_front();
```

emplace_front

Constructs and inserts a new element in the beginning of the forward list

```
template <class Type>   void emplace_front(Type&&... args);
```

emplace_after

Constructors and inserts a new element in a location in the forward list

```
template <class Type> void emplace_after(const iterator loc, Type&&... args);
```

swap

Swaps two forward_lists

```
void swap(forward_list& lst);
```

resize

Changes the size of a forward_list.   If the size is smaller, elements are removed. If the size is larger, elements are added to the list.

```
void resize(size_type n);
void resize(size_type n, const value& val);
```

Operation Functions

merge

Merge two forward_lists. The merge function assumes both forward_lists are sorted.

```
void merge(forward_list& fwdlst);
void merge(forward_list&& fwdlst);
template <class Compare> void merge(forward_list& fwdlst, Compare comp);
template <class Compare> void merge(forward_list&& fwdlst, Compare comp);
```

remove

Removes all elements with a specified value from the forward_list

```
void remove(const value_type& value);
```

remove_if

Removes elements that meet a specified condition

```
template <class Predicate> void remove_if(Predicate pred);
```

reverse

Reverses the order of elements in a forward_list

```
void reverse() noexcept;
```

sort

Sorts elements in a forward_list

```
void sort();
template <class Compare> void sort(Compare comp);
```

splice_after

Inserts part of another forward_list into a forward_list

```
void splice_after(const_iterator position, forward_list& fwdlst);
void splice_after(const_iterator position, forward_list&& fwdlst);
void splice_after(const_iterator position, forward_list& fwdlst,
                  const_iterator i);
void splice_after(const_iterator position, forward_list&& fwdlst,
                  const_iterator i);
void splice_after(const_iterator position, forward_list& fwdlst,
                  const_iterator first, const_iterator last);
void splice_after(const_iterator position, forward_list&& fwdlst,
                  const_iterator first, const_iterator last);
```

unique

Removes duplicate values from a forward_list

```
void unique();
template <class BinaryPredicate> void unique(BinaryPredicate binary_pred);
```

Example 4 – The forward_list container

```
#include <forward_list>
#include <iostream>
using namespace std;

ostream& operator<<(ostream& out, const forward_list<int>& obj);

int main()
{
    // Constructors
    forward_list<int> f1;
    forward_list<int> f2(5);
    forward_list<int> f3(5,19);
    forward_list<int> f4{2,3,5,7,11,13,17};

    cout << "f2 = "<< f2 << endl;
    cout << "f3 = "<< f3 << endl;
    cout << "f4 = "<< f4 << endl;
    cout << endl;
    forward_list<int> f5(f4);
    forward_list<int> f6(move(f4));
    cout << "f4 = "<< f4 << endl;
    cout << "f5 = "<< f5 << endl;
    cout << "f6 = "<< f6 << endl;
    cout << endl;

    // Capacity functions
```

```
    cout << "f1.max_size() = " << f1.max_size() << ' '
         << boolalpha << "  f1.empty() = " << f1.empty() << endl << endl;

    // Access and Iterator functions
    cout << "f5.front() = " << f5.front() << endl;
    cout << "*f5.begin() = " << *f5.begin() << endl;
    cout << "*++f5.before_begin() = " << *++f5.before_begin() << endl << endl;

    // Modifier functions
    cout << "assign" << endl;
    f1.assign(5,7);
    cout << "f1 = " << f1 << endl;
    f1.assign({7,6,5,4,3,2,1});
    cout << "f1 = " << f1 << endl;
    cout << endl;

    cout << "erase_after" << endl;
    f1.erase_after(f1.begin());
    cout << "f1 = " << f1 << endl << endl;

    cout << "insert_after" << endl;
    f1.insert_after(f1.before_begin(),3);
    cout << "f1 = " << f1 << endl;
    f1.insert_after(f1.begin(),f3.begin(),f3.end());
    cout << "f1 = " << f1 << endl << endl;

    cout << "emplace" << endl;
    f1.emplace_front(1);
    cout << "f1 = " << f1 << endl;
    f1.emplace_after(f1.begin(),2);
    cout << "f1 = " << f1 << endl << endl;

    cout << "push_front" << endl;
    f1.push_front(1);
    cout << "f1 = " << f1 << endl << endl;

    cout << "swap" << endl;
    f1.swap(f6);
    cout << "f1 = " << f1 << endl;
    f1.resize(5);
    cout << "f1 = " << f1 << endl << endl;

    cout << "reverse" << endl;
    f1.reverse();
    cout << "f1 = "<< f1 << endl << endl;

    f1.assign({2,4,7,4,5,9,5});
    f2.assign({1,5,7,3,6,2,5});

    // forward_lists are supposed to be sorted before merge
    cout << "sort" << endl;
    cout << "before sort" << endl;
    cout << "f1 = " << f1 << endl;
```

```
    cout << "f2 = " << f2 << endl;
    f1.sort();
    f2.sort();
    cout << "after sort" << endl;
    cout << "f1 = " << f1 << endl;
    cout << "f2 = " << f2 << endl << endl;

    cout << "merge" << endl;
    cout << "f1.merge(f2);" << endl;
    f1.merge(f2);
    cout << "f1 = " << f1 << endl;
    cout << "f2 = " << f2 << endl << endl;

    cout << "f1.unique();" << endl;
    f1.unique();
    cout << "f1 = " << f1 << endl << endl;

    cout << "splice_after" << endl;
    cout << "f3 = " << f3 << endl;
    f1.splice_after(++f1.begin(),f3);
    cout << "f1 = " << f1 << endl;
}

ostream& operator<<(ostream& out, const forward_list<int>& obj)
{
  for (auto forward_listIt = obj.cbegin(); forward_listIt != obj.cend(); ++forward_listIt)
      out << *forward_listIt << ' ';
  return out;
}
```

****** Output ******

```
f2 = 0 0 0 0 0
f3 = 19 19 19 19 19
f4 = 2 3 5 7 11 13 17

f4 =
f5 = 2 3 5 7 11 13 17
f6 = 2 3 5 7 11 13 17

f1.max_size() = 1152921504606846975   f1.empty() = true

f5.front() = 2
*f5.begin() = 2
*++f5.before_begin() = 2

assign
f1 = 7 7 7 7 7
f1 = 7 6 5 4 3 2 1

erase_after
f1 = 7 5 4 3 2 1

insert_after
f1 = 3 7 5 4 3 2 1
f1 = 3 19 19 19 19 19 7 5 4 3 2 1
```

```
emplace
f1 = 1 3 19 19 19 19 19 7 5 4 3 2 1
f1 = 1 2 3 19 19 19 19 19 7 5 4 3 2 1

push_front
f1 = 1 1 2 3 19 19 19 19 19 7 5 4 3 2 1

swap
f1 = 2 3 5 7 11 13 17
f1 = 2 3 5 7 11

reverse
f1 = 11 7 5 3 2

sort
before sort
f1 = 2 4 7 4 5 9 5
f2 = 1 5 7 3 6 2 5
after sort
f1 = 2 4 4 5 5 7 9
f2 = 1 2 3 5 5 6 7

merge
f1.merge(f2);
f1 = 1 2 2 3 4 4 5 5 5 5 6 7 7 9
f2 =

f1.unique();
f1 = 1 2 3 4 5 6 7 9

splice_after
f3 = 19 19 19 19 19
f1 = 1 2 19 19 19 19 19 3 4 5 6 7 9
```

deque

The deque container is similar to vectors and lists. The deque container provides direct access to elements, like a vector and efficient insertion and deletion at both ends, like a list. Unlike a vector, a deque elements are not stored in contiguous memory. The deque container requires the <deque> header file.

Constructors

Default constructor

```
deque();
```

Fill constructors

```
explicit deque(size_type n, const allocator_type& alloc =
                  allocator_type());

deque(size_type n, const value_type& val,
                 const allocator_type& alloc = allocator_type());
```

Range constructor

```
template <class InputIterator>
deque(InputIterator first, InputIterator last,
          const allocator_type& alloc = allocator_type());
```

Copy constructor

```
deque(const deque& x);
```

Move constructor

```
deque(deque&& x);
```

Initializer list constructor

```
deque(initializer_list<value_type> lst,
       const allocator_type& alloc = allocator_type());
```

Iterator Functions

begin

Returns an iterator pointing to the first element of the deque

```
iterator begin() noexcept;
const_iterator begin() const noexcept;
```

end

Returns an iterator pointing to the *non-existing* element beyond the end of the deque

```
iterator end() noexcept;
const_iterator end() const noexcept;
```

rbegin

Returns a reverse iterator pointing to the last element in the deque

```
reverse_iterator rbegin() noexcept;
const_reverse_iterator rbegin() const noexcept;
```

rend

Returns a reverse iterator pointing to the *non-existing* element in front of the first element of the deque

```
reverse_iterator rend() noexcept;
const_reverse_iterator rend() const noexcept;
```

cbegin

Returns a const iterator pointing to the first element of the deque

```
const_iterator begin() const noexcept;
```

cend

Returns a const iterator pointing to the *non-existing* element beyond the end of the deque

```
const_iterator end() const noexcept;
```

crbegin

Returns a const reverse iterator pointing to the last element of the deque

```
const_reverse_iterator rbegin() const noexcept;
```

crend

Returns a const reverse iterator pointing to the non-existing element in front of the first element of the deque

```
const_reverse_iterator rend() const noexcept;
```

Capacity Functions

size

Returns the number of elements in the deque

```
size_t size() const noexcept;
```

max_size

Returns the maximum number of elements that a deque can hold

```
size_t max_size() const noexcept;
```

resize

Resizes a deque to n elements

```
void resize (size_t n);
void resize (size_t n, const value_type& value);
```

empty

Returns whether the deque is empty

```
bool empty() const noexcept;
```

shrink_to_fit

Changes the capacity to the size of the deque

```
void shrink_to_fit();
```

Access Functions

at

Returns element at position

```
value_type& at(size_t position);
const value_type& at(size_t position) const;
```

back

Returns a reference to the last element in the deque

```
value_type&  back();
const value_type&  back() const;
```

front

Returns a reference to the first element in the deque

```
value_type& front();
const value_type& front() const;
```

Modifier Functions

assign

Assigns new contents to a deque

```
template <class InputIterator>
      void assign(InputIterator beg, InputIterator _end);
void assign(size_type n, const value_type& value);
void assign(initializer_list<value_type> list);
```

clear

Erases a deque. Size becomes 0

```
void clear() noexcept;
```

erase

Erases part of a deque

```
iterator erase(const_iterator p);
iterator erase(const_iterator first, const_iterator last);
```

insert

Inserts elements into a deque at a specified location

```
iterator insert(const_iterator loc, const value_type& value);
iterator insert(const_iterator loc, size_type n, const value_type& value);
template <class InputIterator>
iterator insert(const_iterator loc, InputIterator first, InputIterator last);
iterator insert(const_iterator loc, value_type&& value);
iterator insert(const_iterator loc, initializer_list<value_type> list);
```

push_back

Adds an element to the end of a deque

```
void push_back(const value_type& value);
void push_back(value_type&& value);
```

pop_back

Deletes the last element of a deque

```
void pop_back();
```

push_front

Adds an element to the beginning of a deque

```
void push_front(const value_type& value);
void push_front(value_type&& value);
```

pop_front

Deletes the first element of a deque

```
void pop_front();
```

swap

Swaps two deques

```
void swap(deque& vec);
```

emplace

Constructs and inserts a new element at a specified location in the deque

```
template <class Type>   void emplace(const iterator loc, Type&&... args);
```

emplace_front

Constructs and inserts a new element in the beginning of a deque

```
template <class Type>   void emplace_front(Type&&... args);
```

emplace_back

Constructs and inserts a new element at the end of the deque

```
template <class Type>   void emplace_back(Type&&... args);
```

Member Operators

operator=

The assignment operator: assigns new contents to a deque.

```
deque& operator=(const deque& x);
deque& operator=(deque&& x);
deque& operator=(initializer_list<value_type> lst);
```

operator[]

Index operator: returns the element at the specified location

```
value_type& operator[](size_t location);
const value_type& operator[](size_t location) const;
```

Relational operators

```
== > < >= <= !=
```

Used to compare the contents of two deques.

Two deques are equal (==) if their sizes match and each of the corresponding elements match. A less than (<) comparison is made between two deques by comparing successive elements in order.

Example 5 – The deque container

```
#include <forward_list>
#include <iostream>
using namespace std;

ostream& operator<<(ostream& out, const forward_list<int>& obj);

int main()
{
    // Constructors
     forward_list<int> f1;
     forward_list<int> f2(5);
     forward_list<int> f3(5,19);
     forward_list<int> f4{2,3,5,7,11,13,17};

     cout << "f2 = "<< f2 << endl;
     cout << "f3 = "<< f3 << endl;
     cout << "f4 = "<< f4 << endl;
     cout << endl;
     forward_list<int> f5(f4);
```

```
    forward_list<int> f6(move(f4));
    cout << "f4 = "<< f4 << endl;
    cout << "f5 = "<< f5 << endl;
    cout << "f6 = "<< f6 << endl;
    cout << endl;

    // Capacity functions
    cout << "f1.max_size() = " << f1.max_size() << ' ' << boolalpha
         << "  f1.empty() = " << f1.empty() << endl << endl;

    // Access and Iterator functions
    cout << "f5.front() = " << f5.front() << endl;
    cout << "*f5.begin() = " << *f5.begin() << endl;
    cout << "*++f5.before_begin() = " << *++f5.before_begin()
         << endl << endl;

    // Modifier functions
    cout << "assign" << endl;
    f1.assign(5,7);
    cout << "f1 = " << f1 << endl;
    f1.assign({7,6,5,4,3,2,1});
    cout << "f1 = " << f1 << endl;
    cout << endl;

    cout << "erase_after" << endl;
    f1.erase_after(f1.begin());
    cout << "f1 = " << f1 << endl << endl;

    cout << "insert_after" << endl;
    f1.insert_after(f1.before_begin(),3);
    cout << "f1 = " << f1 << endl;
    f1.insert_after(f1.begin(),f3.begin(),f3.end());
    cout << "f1 = " << f1 << endl << endl;

    cout << "emplace" << endl;
    f1.emplace_front(1);
    cout << "f1 = " << f1 << endl;
    f1.emplace_after(f1.begin(),2);
    cout << "f1 = " << f1 << endl << endl;

    cout << "push_front" << endl;
    f1.push_front(1);
    cout << "f1 = " << f1 << endl << endl;

    cout << "swap" << endl;
    f1.swap(f6);
    cout << "f1 = " << f1 << endl;
    f1.resize(5);
    cout << "f1 = " << f1 << endl << endl;

    cout << "reverse" << endl;
    f1.reverse();
    cout << "f1 = "<< f1 << endl << endl;

    cout << "merge" << endl;
```

```
    f1.assign({2,4,7,4,5,9,5});
    f2.assign({1,5,7,3,6,2,5});
    cout << "before merge: f1 = " << f1 << endl;
    cout << "before merge: f2 = " << f2 << endl;

    cout << "f1.merge(f2);" << endl;
    f1.merge(f2);
    cout << "after merge: f1 = " << f1 << endl;
    cout << "after merge: f2 = " << f2 << endl << endl;

    // forward_lists are supposed to be sorted before merge
    f1.assign({2,4,7,4,5,9,5});
    f2.assign({1,5,7,3,6,2,5});

    cout << "sort" << endl;
    cout << "before sort" << endl;
    cout << "f1 = " << f1 << endl;
    cout << "f2 = " << f2 << endl;
    f1.sort();
    f2.sort();
    cout << "after sort" << endl;
    cout << "f1 = " << f1 << endl;
    cout << "f2 = " << f2 << endl << endl;

    cout << "f1.merge(f2);" << endl;
    f1.merge(f2);
    cout << "f1 = " << f1 << endl;
    cout << "f2 = " << f2 << endl << endl;

    cout << "f1.unique();" << endl;
    f1.unique();
    cout << "f1 = " << f1 << endl << endl;

    cout << "splice_after" << endl;
    cout << "f3 = " << f3 << endl;
    f1.splice_after(++f1.begin(),f3);
    cout << "f1 = " << f1 << endl;
}

ostream& operator<<(ostream& out, const forward_list<int>& obj)
{
    for (auto forward_listIt = obj.cbegin(); forward_listIt != obj.cend(); ++forward_listIt)
        out << *forward_listIt << ' ';
    return out;
}
```

****** Output ******

```
f2 = 0 0 0 0 0
f3 = 19 19 19 19 19
f4 = 2 3 5 7 11 13 17

f4 =
f5 = 2 3 5 7 11 13 17
```

```
f6 = 2 3 5 7 11 13 17

f1.max_size() = 1152921504606846975   f1.empty() = true

f5.front() = 2
*f5.begin() = 2
*++f5.before_begin() = 2

assign
f1 = 7 7 7 7 7
f1 = 7 6 5 4 3 2 1

erase_after
f1 = 7 5 4 3 2 1

insert_after
f1 = 3 7 5 4 3 2 1
f1 = 3 19 19 19 19 19 7 5 4 3 2 1

emplace
f1 = 1 3 19 19 19 19 19 7 5 4 3 2 1
f1 = 1 2 3 19 19 19 19 19 7 5 4 3 2 1

push_front
f1 = 1 1 2 3 19 19 19 19 19 7 5 4 3 2 1

swap
f1 = 2 3 5 7 11 13 17
f1 = 2 3 5 7 11

reverse
f1 = 11 7 5 3 2

merge
before merge: f1 = 2 4 7 4 5 9 5
before merge: f2 = 1 5 7 3 6 2 5
f1.merge(f2);
after merge: f1 = 1 2 4 5 7 4 5 7 3 6 2 5 9 5
after merge: f2 =

sort
before sort
f1 = 2 4 7 4 5 9 5
f2 = 1 5 7 3 6 2 5
after sort
f1 = 2 4 4 5 5 7 9
f2 = 1 2 3 5 5 6 7

f1.merge(f2);
f1 = 1 2 2 3 4 4 5 5 5 5 6 7 7 9
f2 =

f1.unique();
f1 = 1 2 3 4 5 6 7 9

splice_after
f3 = 19 19 19 19 19
f1 = 1 2 19 19 19 19 19 3 4 5 6 7 9
```

queue

The queue container *adaptor* implements a FIFO (first in, first out) container. The queue is an *adaptor*. This means that its data is a container itself. The queue adapter is simply an interface to the underlying container. Elements of a queue are pushed on to the back of the queue and popped off the front of the queue. The queue container requires the <queue> header file.

Constructors

Initialize constructor

```
explicit queue(const container_type& ctnr);
```

Move initialize constructor

```
explicit queue(container_type&& ctnr = container_type());
```

Where is the copy constructor?

Member Functions

size

Returns the number of elements in the queue

```
size_type size() const;
```

empty

Returns whether the queue is empty

```
bool empty() const;
```

back

Returns a reference to the last element added to the queue.

```
value_type&  back();
const value_type&  back() const;
```

front

Returns a reference to the first element in the queue. This is the next element that will be *popped off*.

```
value_type& front();
const value_type& front() const;
```

push

Adds an element to the end of a queue.

```
void push(const value_type& value);
void push(value_type&& value);
```

pop

Removes the first element in the queue. That is, the *oldest* element in the queue.

```
void pop();
```

emplace

Constructs and add a new element to the back of the queue.

```
template <class Type>   void emplace(Type&&... args);
```

swap

Swaps the contents of two queues. The types of the queues must match.

```
void swap(queue& another_queue) noexcept;
```

Relational operators

```
== > < >= <= !=
```

Used to compare the contents of two queues.

Two deques are equal (==) if their sizes match and each of the corresponding elements match. A less than (<) comparison is made between two queues by comparing successive elements in order.

Example 6 – The queue adaptor

```
#include <list>
#include <vector>
#include <queue>
#include <iostream>
using namespace std;

int main()
{
    // Constructors
     queue<int> q1;

     q1.push(10);
     q1.push(20);
     q1.push(30);
     cout << "q1.size() = " << q1.size() << endl;
     cout << "q1.front() = " << q1.front() << endl;
     cout << "q1.back() = " << q1.back() << endl << endl;
```

```
    cout << "\"process q1\"" << endl;
    while (!q1.empty())
    {
        cout << q1.front() << ' ';
        q1.pop();
    }
    cout << endl << endl;

    cout << "Create a queue using an underlying list" << endl;
    list<int> l1{2,3,5,7};
    queue<int, list<int>> q2(l1);
    cout << "q2.size() = " << q2.size() << endl;
    cout << "q2.front() = " << q2.front() << endl;
    cout << "q2.back() = " << q2.back() << endl << endl;
    cout << "\"process q2\"" << endl;
    while (!q2.empty())
    {
        cout << q2.front() << ' ';
        q2.pop();
    }
    cout << endl << endl;

    cout << "emplace" << endl;
    q2.emplace(17);
    q2.emplace(18);
    cout << "q2.front() = " << q2.front() << endl;
    cout << "q2.back() = " << q2.back() << endl;
    cout << endl;

    cout << "Create a queue by moving a vector" << endl;
    vector<double> v1{1.2,3.4,5.6,7.8};
    queue<double, vector<double>> q4(move(v1));
    cout << "q4.size() = " << q4.size() << endl;
    cout << "v1.size() = " << v1.size() << endl;
    cout << endl;

    queue<double> q5;
//    q5.swap(q4);    ERROR
    v1 = {1.1,2.2,3.3};  // reassign vector v1
    cout << "create a queue using an underlying vector of doubles"
<< endl;
    queue<double, vector<double>> q6(v1);

    cout << "swap two queues" << endl;
    q6.swap(q4);
    cout << "q6.size() = " << q6.size() << endl;
}
```

****** Output ******

```
q1.size() = 3
q1.front() = 10
q1.back() = 30

"process q1"
```

```
10 20 30

Create a queue using an underlying list
q2.size() = 4
q2.front() = 2
q2.back() = 7

"process q2"
2 3 5 7

emplace
q2.front() = 17
q2.back() = 18

Create a queue by moving a vector
q4.size() = 4
v1.size() = 0

create a queue using an underlying vector of doubles
swap two queues
q6.size() = 4
```

priority_queue

The priority_queue *adaptor* implements a container in which the first element is always the one that is considered the maximum value. Hence, the maximum value will always be *popped off* first. The determination of the maximum value requires a *binary predicate*[5] to make comparison of the priority_queue values. The priority_queue container requires the **<queue>** header file.

Constructors

Initialize constructor

```
priority_queue (const Compare& comp, const Container& ctnr);
```

Move initialize constructor

```
explicit priority_queue (const Compare& comp = Compare(),
                         Container&& ctnr = Container());
```

Range constructor

```
template <class InputIterator>
  priority_queue (InputIterator first, InputIterator last,
                  const Compare& comp, const Container& ctnr);
```

Move range constructor

```
template <class InputIterator>
  priority_queue (InputIterator first, InputIterator last,
                  const Compare& comp, Container&& ctnr = Container());
```

Member Functions

size

Returns the number of elements in the priority_queue

```
size_type size() const;
```

empty

Returns whether the priority_queue is empty

```
bool empty() const;
```

top

Returns a reference to the top (first to be *popped*) element in the queue.

```
const value_type& top() const;
```

[5] A binary predicate is a function object that requires two arguments and returns a bool.

push

Inserts a new element into the priority_queue.

```
void push(const value_type& value);
void push(value_type&& value);
```

pop

Removes the top element in the priority_queue. This is the element with the maximum *value*.

```
void pop();
```

emplace

Constructs and inserts a new element into the priority_queue.

```
template <class Type>   void emplace(Type&&... args);
```

swap

Swaps the contents of two priority_queues. Both the value types and the comparison functions of the two priority_queues must match.

```
void swap(priority_queue& another_pq) noexcept;
```

Example 7 – The priority_queue adaptor

```
#include <iostream>
#include <queue>
#include <vector>
#include <functional>     // for greater<int>
#include <string>
using namespace std;

// "Non-destructive" print function?
template<typename T> void print_queue(T q)
 {
     while(!q.empty())
     {
         std::cout << q.top() << " ";
         q.pop();
     }
     std::cout << '\n';
 }

 // binary predicate (function object/functor) for comparing strings
 // returns true if first string is shorter than second string
 struct longer
 {
     bool operator()(const string& a, const string& b)
     {
         return a.size() < b.size();
     }
```

```cpp
};

int main ()
{
    int myints[]= {10,60,50,20};
    vector<int> v1{10,20,30,40};
    vector<string> v2{"Have","a","really","very","nice","day","."};

    // pq1, pq2, pq3 uses default < comparison for type int
    priority_queue<int> pq1;
    priority_queue<int> pq2 (v1.begin(), v1.end());
    priority_queue<int> pq3 (myints,myints+4);

    // pq4 uses default > comparison for type int for priority
    priority_queue<int, vector<int>, std::greater<int> > pq4 (myints,myints+4);

    // pq5 uses default < comparison for type string
    priority_queue<string> pq5 (v2.begin(),v2.end());

    // pq6 uses longer binary predicate comparison for type string
    priority_queue<string, vector<string>, longer> pq6 (v2.begin(),v2.end());

    cout << "pq2 = ";    print_queue(pq2);
    cout << "pq3 = ";    print_queue(pq3);
    cout << "pq4 = ";    print_queue(pq4);
    cout << "pq5 = ";    print_queue(pq5);
    cout << "pq6 = ";    print_queue(pq6);

    cout << "pq3.size()=" << pq3.size() << endl;
    cout << "pq4.size()=" << pq4.size() << endl << endl;

    cout << "pq2 and pq3 swapped" << endl;
    pq2.swap(pq3);
    //  pq3.swap(pq4);  ERROR - why?
    cout << "pq2 = ";    print_queue(pq2);

    pq2.push(95);
    pq2.push(5);
    pq2.push(25);
    pq2.emplace(35);
    cout << "pq2 = ";    print_queue(pq2);
}
```

****** Output ******

```
pq2 = 40 30 20 10
pq3 = 60 50 20 10
pq4 = 10 20 50 60
pq5 = very really nice day a Have .
pq6 = really Have nice very day . a
pq3.size()=4
pq4.size()=4

pq2 and pq3 swapped
```

```
pq2 = 60 50 20 10
pq2 = 95 60 50 35 25 20 10 5
```

stack

The stack container *adaptor* implements a LIFO (last in, first out) container. The stack, like a queue and a priority_queue is an *adaptor*, meaning that its data is a container itself. The stack uses a deque, by default as its underlying container. Elements of a stack are pushed on to the top of the stack and popped off the top of the stack. The queue container requires the <stack> header file.

Constructors

Initialize constructor

```
explicit stack(const container_type& ctnr);
```

Move initialize constructor

```
explicit stack(container_type&& ctnr = container_type());
```

Member Functions

size

Returns the number of elements in the stack.

```
size_type size() const;
```

empty

Returns whether the stack is empty

```
bool empty() const;
```

top

Returns a reference to the last element added to the stack.

```
value_type&  top();
const value_type&  top() const;
```

push

Adds an element to the top of the stack.

```
void push(const value_type& value);
void push(value_type&& value);
```

pop

Removes the element on the top of the stack. That is, the *last* element pushed on the stack.

```
void pop();
```

emplace

Constructs and add a new element to the top of the stack.

```
template <class Type>   void emplace(Type&&... args);
```

swap

Swaps the contents of two stacks. The types of the stacks must match. Note, swap swaps the two underlying containers.

```
void swap(stack& another_stack) noexcept;
```

Relational operators

```
== > < >= <= !=
```

Used to compare the contents of two stacks.

Two deques are equal (==) if their sizes match and each of the corresponding elements match. A less than (<) comparison is made between two deques by comparing successive elements in order.

Example 8 – The stack adaptor

```
#include <list>
#include <vector>
#include <stack>
#include <iostream>
using namespace std;

// Why is this a template?
template<typename T> void print_stack(T q)
{
     while(!q.empty())
     {
         cout << q.top() << " ";
         q.pop();
     }
     cout << endl;
 }

 int main()
 {
     // Constructors
```

```
    stack<int> stk1;

    stk1.push(10);
    stk1.push(20);
    stk1.push(30);
    cout << "stk1 = "; print_stack(stk1);
    cout << endl;

    list<int> l1{2,3,5,7};
    stack<int, list<int>> stk2(l1);
    cout << "stk2 = "; print_stack(stk2);
    cout << endl;

    stk2.emplace(17);
    stk2.emplace(18);
    cout << "stk2 = "; print_stack(stk2);
    cout << endl;

    vector<double> v1{1.2,3.4,5.6,7.8};
    stack<double, vector<double>> stk3(move(v1));
    cout << stk3.size() << endl;
    cout << v1.size() << endl;
    cout << "stk3 = "; print_stack(stk3);
    cout << endl;

    stack<double> stk4;
    // stk4.swap(stk3);  ERROR - why?

    v1 = {1.3,2.2,3.3};
    stack<double, vector<double>> stk5(v1);
    stk5.swap(stk3);
    cout << "stk3 = "; print_stack(stk3);
    cout << "stk5 = "; print_stack(stk5);

    stk5.push(3.2);
    cout << "stk5 = "; print_stack(stk5);
    cout << "stk3 > stk5: " << boolalpha << (stk3 > stk5) << endl;
    cout << endl;

    stk3.push(stk3.top());
    stk3.push(stk3.top());
    cout << "stk3 = "; print_stack(stk3);
    cout << "stk5 = "; print_stack(stk5);
    cout << boolalpha << endl;
    cout << "stk3 > stk5: " << (stk3 > stk5) << endl;
    cout << "stk3 < stk5: " << (stk3 < stk5) << endl;
    cout << "stk3 == stk5: " << (stk3 == stk5) << endl;
}
```

****** Output ******

```
stk1 = 30 20 10

stk2 = 7 5 3 2
```

```
stk2 = 18 17 7 5 3 2

4
0
stk3 = 7.8 5.6 3.4 1.2

stk3 = 3.3 2.2 1.3
stk5 = 7.8 5.6 3.4 1.2
stk5 = 3.2 7.8 5.6 3.4 1.2
stk3 > stk5: true

stk3 = 3.3 3.3 3.3 2.2 1.3
stk5 = 3.2 7.8 5.6 3.4 1.2

stk3 > stk5: true
stk3 < stk5: false
stk3 == stk5: false
```

set

The set container is an associative container in which elements are unique and stored in a sorted order. The set container requires the <set> header file.

Constructors

Default constructor

```
set();
```

empty constructor

```
explicit set (const key_compare& comp, const allocator_type& alloc =
allocator_type());
```

range constructor

```
template <class InputIterator>
  set(InputIterator first, InputIterator last,
      const key_compare& comp = key_compare(),
       const allocator_type& = allocator_type());

template <class InputIterator>
  set(InputIterator first, InputIterator last,
      const allocator_type& = allocator_type());
```

copy constructor

```
set(const set& x);
```

move constructor

```
set(set&& x);
```

initializer list constructor

```
set(initializer_list<value_type> lst,
     const key_compare& comp = key_compare(),
     const allocator_type& alloc = allocator_type());
```

Iterator Functions

begin

Returns an iterator pointing to the first element of the set

```
iterator begin() noexcept;
const_iterator begin() const noexcept;
```

end

Returns an iterator pointing to the *non-existing* element beyond the end of the set

```
iterator end() noexcept;
const_iterator end() const noexcept;
```

rbegin

Returns a reverse iterator pointing to the last element in the set

```
reverse_iterator rbegin() noexcept;
const_reverse_iterator rbegin() const noexcept;
```

rend

Returns a reverse iterator pointing to the *non-existing* element in front of the first element of the set

```
reverse_iterator rend() noexcept;
const_reverse_iterator rend() const noexcept;
```

cbegin

Returns a ***const*** iterator pointing to the first element of the set

```
const_iterator begin() const noexcept;
```

cend

Returns a ***const*** iterator pointing to the *non-existing* element beyond the end of the set

```
const_iterator end() const noexcept;
```

crbegin

Returns a ***const*** reverse iterator pointing to the last element of the set

```
const_reverse_iterator rbegin() const noexcept;
```

crend

Returns a ***const*** reverse iterator pointing to the non-existing element in front of the first element of the set

```
const_reverse_iterator rend() const noexcept;
```

Capacity Functions

size

Returns the number of elements in the set

```
size_t size() const noexcept;
```

max_size

Returns the maximum number of elements that a set can hold

```
size_t max_size() const noexcept;
```

empty

Returns whether the set is empty

```
bool empty() const noexcept;
```

Modifier Functions

clear

Erases all elements of a set. Size becomes 0

```
void clear() noexcept;
```

erase

Erases elements in a set

```
iterator erase(const_iterator p);
size_t erase(const value_type& value);
iterator erase(const_iterator first, const_iterator last);
```

insert

Inserts elements into a set at a specified location. Elements must be unique, so duplicate values may not be inserted.

```
pair<iterator,bool> insert(const value_type& value);
pair<iterator,bool> insert(value_type&& value);
iterator insert(const_iterator position, const value_type& value);
iterator insert(const_iterator position, value_type&& value);
```

```
template <class InputIterator>
   void insert(InputIterator first, InputIterator last);
void insert(initializer_list<value_type> lst);
```

swap

Swaps two sets

```
void swap(set& another_set);
```

Operation Functions

count

Returns the number of elements that are equal to a value in the set. Because the elements in a set must be unique, count can only return 1 or 0.

```
size_type count(const value_type& value) const;
```

find

Searches the set for a value. Returns an iterator to the found element, otherwise it returns set::end().

```
const_iterator find(const value_type& value) const;
iterator       find(const value_type& value);
```

lower_bound

Returns an iterator pointing to the first element in the set that is not less than a value. If there are no elements less than the value, then then function returns set::end().

```
iterator lower_bound (const value_type& value);
const_iterator lower_bound (const value_type& value) const;
```

upper_bound

Returns an iterator pointing to the first element in the set that is greater than a value. If there are no elements greater than the value, then then function returns set::end().

```
iterator upper_bound (const value_type& value);
const_iterator upper_bound (const value_type& value) const;
```

Example 9 – The set container

```
#include <iostream>
```

```cpp
#include <set>
using namespace std;

class Student
{
    unsigned id;
    string name;
public:
    Student() = delete;
    Student(unsigned arg1, string arg2 = "") : id(arg1), name(arg2) {}
    Student(const Student&) = default;
    bool operator<(const Student& obj) const
    {
        return id < obj.id;
    }
    bool operator==(const Student& obj) const
    {
        return id == obj.id;
    }
    friend ostream& operator<<(ostream& out, const Student& obj)
    {
        out << obj.id << "   " << obj.name;
        return out;
    }
};

ostream& operator<<(ostream& out, const set<Student>& stu)
{
    for (auto it = stu.cbegin(); it != stu.cend(); ++it)
    {
        out << *it << endl;
    }
    return out;
}

int main()
{
    set<Student> Students;
    Students.insert({117,"John"});
    Students.insert({124,"Paul"});
    Students.insert({102,"George"});
    Students.insert({106,"Ringo"});
    Students.insert({223,"Peter"});
    Students.insert({203,"Paul"});
    Students.insert({243,"Mary"});

    cout << "Students.size() = "  << Students.size() << endl;
    cout << Students << endl;

    bool insertSuccess;
    cout << boolalpha;

    insertSuccess = Students.insert({309,"Mick"}).second;
    cout << "insert 309: " << insertSuccess << endl;
```

```
    insertSuccess = Students.insert({117,"Nobody"}).second;
    cout << "insert 117: " << insertSuccess << endl << endl;

    cout << "find 106: " << *(Students.find(106)) << endl;  // How does this work?
    //  cout << *(Students.find(107)) << endl;  // ERROR

    unsigned id;
    set<Student>::const_iterator it;
    cout << "find 203: " << (Students.find(203) != Students.end()) << endl;
    cout << "find 107: " << (Students.find(107) != Students.end()) << endl << endl;

    cout << "Before erase: Students.size() = "  << Students.size() << endl;
    id = 203;
    Students.erase(Students.find(id));  // Did this work?
    cout << "After erase of 203: Students.size() = "  << Students.size() << endl;
    cout << "Students.erase(102) = " << Students.erase(102) << endl;
    cout << "Students.erase(103) = " << Students.erase(103) << endl;
}
```

****** Output ******

```
Students.size() = 7
102   George
106   Ringo
117   John
124   Paul
203   Paul
223   Peter
243   Mary

insert 309: true
insert 117: false

find 106: 106   Ringo
find 203: true
find 107: false

Before erase: Students.size() = 8
After erase of 203: Students.size() = 7
Students.erase(102) = 1
Students.erase(103) = 0
```

multiset

The multiset container is an associative container in which elements stored in a sorted order, but element values are not unique. The multiset container requires the <set> header file.

Member Functions

The multiset constructors and member functions are essentially the same as the set container. The following illustrates some of the differences.

erase

Erases elements in a multiset

```
iterator erase(const_iterator p);
```

Only a single element of the multiset is erased.

```
size_t erase(const value_type& value);
```

Erases all elements in the multiset with a key equal to the specified value. The function returns the number of elements erased.

insert

```
iterator insert(const value_type& val);
iterator insert(value_type&& val);
```

This version of the insert function returns only an iterator to the element that was inserted. Unlike the set::insert, there is no bool indication of success or failure.

As of C++11, when duplicate values are inserted into the multiset, newly inserted elements are inserted after those with the same value.

count

Like the set::count the function returns the number of elements that are equal to a value in the set. Since the elements in a multiset are not necessarily unique, the count may be greater than 1.

```
size_type count(const value_type& value) const;
```

equal_range

Returns a pair of iterators pointer to the first and last element that is equal to a value in the multiset. If no matches are found, the range returned has a length of zero, with both iterators pointing to the first element that is greater than the value.

```
pair<const_iterator,const_iterator> equal_range(const value_type& value)
const;
pair<iterator,iterator>             equal_range(const value_type& value);
```

Non-member Functions

Note: these operators, > < >= <= != will be removed in C++20. The <=> operator will be added. More to say about that later.

Example 10 – The multiset container

```
#include <iostream>
#include <set>
using namespace std;

class Student
{
    unsigned id;
    string name;
public:
    Student() = delete;
    Student(unsigned arg1, string arg2 = "") : id(arg1), name(arg2) {}
    Student(const Student&) = default;
    bool operator<(const Student& obj) const
    {
        return id < obj.id;
    }
    bool operator==(const Student& obj) const
    {
        return id == obj.id;
    }
    friend ostream& operator<<(ostream& out, const Student& obj)
    {
        out << obj.id << "   " << obj.name;
        return out;
    }
};

ostream& operator<<(ostream& out, const multiset<Student>& stu)
{
    for (auto it = stu.cbegin(); it != stu.cend(); ++it)
    {
        out << *it << endl;
    }
    return out;
}

int main()
{
    multiset<Student> Students;
    Students.insert({117,"John"});
    Students.insert({124,"Paul"});
    Students.insert({102,"George"});
    Students.insert({106,"Ringo"});
    Students.insert({223,"Peter"});
    Students.insert({203,"Paul"});
    Students.insert({243,"Mary"});
```

```

    cout << "Students.size() = "  << Students.size() << endl;
    cout << Students << endl;

    multiset<Student>::iterator msIt;
    msIt = Students.insert({309,"Mick"});
    cout << "New student: " << *msIt << endl;

    msIt = Students.insert({117,"Elvis"});
    cout << "Another new student: " << *msIt << endl << endl;

    cout << Students << endl;

    // Check count
    cout << "count of 117 = " << Students.count(117) << endl;
    // cout << "# of Paul = " << Students.count("Paul") << endl; // ERROR
    cout << endl;

    // check find
    multiset<Student>::const_iterator cMsIt;
    cMsIt = Students.find(124);
    cout << "find 124: " << *cMsIt << endl;
    //  cout << *(Students.find(107)) << endl;  // ERROR
    ++cMsIt;
    cout << *cMsIt << endl;
    ++cMsIt;
    cout << *cMsIt << endl;
    int id = 125;
    cMsIt = Students.find(id);
    // cout << *cMsIt << endl; // CRASH
    if (cMsIt == Students.end())
        cout << "Can't find " << id << endl << endl;

    // equal_range
    cout << "equal_range 117" << endl;
    auto twoIterators = Students.equal_range(117);
    cout << *twoIterators.first << endl <<  *twoIterators.second << endl << endl;
    cout << "equal_range 203" << endl;
    twoIterators = Students.equal_range(203);
    cout << *twoIterators.first << endl <<  *twoIterators.second << endl << endl;
    cout << "equal_range 204" << endl;
    twoIterators = Students.equal_range(204);
    cout << *twoIterators.first << endl <<  *twoIterators.second << endl << endl;
    if (twoIterators.first == twoIterators.second) cout << "204 not found" << endl << endl;

    // erase
    cout << "Erase 117: " << Students.erase(117) << endl;
    cout << "Erase 118: " << Students.erase(118) << endl << endl;
    cout << Students << endl;
}
```

****** Output ******

```
Students.size() = 7
102   George
106   Ringo
117   John
124   Paul
203   Paul
223   Peter
243   Mary

New student: 309   Mick
Another new student: 117   Elvis

102   George
106   Ringo
117   John
117   Elvis
124   Paul
203   Paul
223   Peter
243   Mary
309   Mick

count of 117 = 2

find 124: 124   Paul
203   Paul
223   Peter
Can't find 125

equal_range 117
117   John
124   Paul

equal_range 203
203   Paul
223   Peter

equal_range 204
223   Peter
223   Peter

204 not found

Erase 117: 2
Erase 118: 0

102   George
106   Ringo
124   Paul
203   Paul
223   Peter
243   Mary
309   Mick
```

map

The map container is an associative container in which elements, consisting of a key-mapped value ***pair*** stored in a sorted order by the key. The key value must be unique in the map. The map container requires the <map> header file.

Constructors

Default constructor

```
map();
```

empty constructor

```
explicit map(const key_compare& comp, const allocator_type& alloc =
allocator_type());
```

range constructor

```
template <class InputIterator>
  map(InputIterator first, InputIterator last,
      const key_compare& comp = key_compare(),
       const allocator_type& = allocator_type());

template <class InputIterator>
  map(InputIterator first, InputIterator last,
      const allocator_type& = allocator_type());
```

copy constructor

```
map(const map& x);
```

move constructor

```
map(map&& x);
```

initializer list constructor

```
map(initializer_list<value_type> lst,
     const key_compare& comp = key_compare(),
     const allocator_type& alloc = allocator_type());
```

Iterator Functions

begin

Returns an iterator pointing to the first element of the map

```
iterator begin() noexcept;
const_iterator begin() const noexcept;
```

end

Returns an iterator pointing to the *non-existing* element beyond the end of the map

```
iterator end() noexcept;
const_iterator end() const noexcept;
```

rbegin

Returns a reverse iterator pointing to the last element in the map

```
reverse_iterator rbegin() noexcept;
const_reverse_iterator rbegin() const noexcept;
```

rend

Returns a reverse iterator pointing to the *non-existing* element in front of the first element of the map

```
reverse_iterator rend() noexcept;
const_reverse_iterator rend() const noexcept;
```

cbegin

Returns a ***const*** iterator pointing to the first element of the map

```
const_iterator begin() const noexcept;
```

cend

Returns a ***const*** iterator pointing to the *non-existing* element beyond the end of the map

```
const_iterator end() const noexcept;
```

crbegin

Returns a ***const*** reverse iterator pointing to the last element of the map

```
const_reverse_iterator rbegin() const noexcept;
```

crend

Returns a ***const*** reverse iterator pointing to the non-existing element in front of the first element of the map

```
const_reverse_iterator rend() const noexcept;
```

Capacity Functions

size

Returns the number of elements in the map

```
size_t size() const noexcept;
```

max_size

Returns the maximum number of elements that a map can hold

```
size_t max_size() const noexcept;
```

empty

Returns whether the map is empty

```
bool empty() const noexcept;
```

Modifier Functions

clear

Erases all elements of a map. Size becomes 0

```
void clear() noexcept;
```

erase

Erases elements in a map

```
iterator erase(const_iterator p);
size_t erase(const key_type& value);
iterator erase(const_iterator first, const_iterator last);
```

insert

Inserts elements into a map at a specified location

Note, the value_type is a *key, mapped-value pair*, in which the *key* must be unique.

```
pair<iterator,bool> insert(const value_type& value);
pair<iterator,bool> insert(value_type&& value);
iterator insert(const_iterator position, const value_type& value);
iterator insert(const_iterator position, value_type&& value);
template <class InputIterator>
   void insert(InputIterator first, InputIterator last);
void insert(initializer_list<value_type> lst);
```

swap

Swaps two maps

```
void swap(map & another_ map);
```

Operation Functions

count

Returns the number of elements that are equal to a key in the map. Because the elements in a map must be unique, count can only return 1 or 0.

```
size_type count(const key_type& value) const;
```

find

Searches the map for a key. Returns an iterator to the found element, otherwise it returns map::end().

```
const_iterator find(const key_type& key) const;
iterator       find(const key_type& key);
```

lower_bound

Returns an iterator pointing to the first element in the map that is not less than a key_value. If there are no elements less than the key_value, then then function returns map::end().

```
iterator lower_bound (const key_type& key);
const_iterator lower_bound (const key_type& key) const;
```

upper_bound

Returns an iterator pointing to the first element in the map that is greater than a key_value. If there are no elements greater than the key_value, then then function returns map::end().

```
iterator upper_bound (const key_type& key);
const_iterator upper_bound (const key_type& key) const;
```

Accessor function/operator

operator[]

Returns the mapped-value for a given key-value. If the key-value is not contained in the map, then the operator inserts a new element into the map, with a *default-constructed mapped-value*.

```
mapped_type& operator[] (const key_type& key);
mapped_type& operator[] (key_type&& key);
```

at

Returns the mapped-value for a given key-value. If the key-value is not contained in the map, the function throws an *out_of_range exception*.

```
mapped_type& at(const key_type& key);
const mapped_type& at(const key_type& key) const;
```

Example 11 – The map container

```
#include <iostream>
#include <iomanip>
#include <map>
#include <string>
#include <cstdlib>
using std::cout;
using std::endl;
using std::string;

// Alias declarations
using StudentId = unsigned;
using Name = string;
using Students = std::map<StudentId,Name>;

// function prototypes
unsigned rand100u();
Students::const_iterator
getInteratorForName(Students&, const Name& name);
std::ostream& operator<<(std::ostream&, const Students&);


int main()
{
    Students students;

    // insert 4 Students into the map
    students[rand100u()] = "John Lennon";
    students.insert(std::pair<StudentId,Name>(rand100u(),"Paul McCartney"));
    using Student = std::pair<StudentId,Name>;
    Student george{rand100u(),"George Harrison"};
    students.insert(george);
    StudentId ringoId = rand100u();
    Student ringo{ringoId,"Ringo Star"};
    students.insert(std::move(ringo));

    cout << students << endl;

    // What does this mean?
    students[50];
    cout << students << endl;

    // Correct the spelling of Ringo's name
    students[ringoId] = "Ringo Starr";
    cout << students << endl;
```

```

    // Remove Student 50
    students.erase(students.find(50));
    cout << students << endl;

    // What is John's number?
    cout << "John's number is "
         << getInteratorForName(students,"John Lennon")->first
         << endl << endl;

    auto it = getInteratorForName(students,"Mick Jagger");
    if (it == students.end())
        cout << "Mick Jagger ain't there" << endl << endl;

    // count
    cout << "number of elements with key " << ringoId << " = "
         << students.count(ringoId) << endl;
    cout << "number of elements with key " << ringoId+1 << " = "
         << students.count(ringoId+1) << endl;
}


unsigned rand100u()
{
    return rand() % 100 + 1;
}

std::ostream& operator<<(std::ostream& out, const Students& studs)
{
    out << std::left;
    for (auto it = studs.begin(); it != studs.end(); ++it)
    {
        out << std::setw(5) << it->first << std::setw(10)
            << it->second << endl;
    }
    return out;
}

Students::const_iterator
getInteratorForName(Students& Students, const string& name)
{
    for (auto it = Students.cbegin(); it != Students.cend(); ++it)
    {
        if (it->second == name) return it;
    }
    return Students.end();
}
```

****** Output ******

```
30   Ringo Star
34   John Lennon
44   Paul McCartney
63   George Harrison
```

```
30   Ringo Star
34   John Lennon
44   Paul McCartney
50
63   George Harrison

30   Ringo Starr
34   John Lennon
44   Paul McCartney
50
63   George Harrison

30   Ringo Starr
34   John Lennon
44   Paul McCartney
63   George Harrison

John's number is 34

Mick Jagger ain't there

number of elements with key 30 = 1
number of elements with key 31 = 0
```

multimap

The multimap container is an associative container in which elements stored in a sorted order. Element values in a multimap are pairs of key and mapped values. Unlike the map container, element key values are not unique. The multimap container requires the <map> header file.

Member Functions

The multimap constructors and member functions are essentially the same as the map container. The following illustrates some of the differences.

erase

Erases elements in a multimap

```
iterator erase(const_iterator p);
```

Only a single element of the multimap is erased.

```
size_t erase(const value_type& value);
```

Erases all elements in the multimap with a key equal to the specified value. The function returns the number of elements erased.

insert

```
iterator insert(const value_type& val);
```

```
iterator insert(value_type&& val);
```

This version of the insert function returns only an iterator to the element that was inserted. Unlike the map::insert, there is no bool indication of success or failure. The multimap::insert does not fail like the map::insert when duplicate key values are inserted.

As of C++11, when duplicate values of the key are inserted into the multimap, newly inserted elements are inserted after those with the same key.

count

Like the map::count the function returns the number of elements that are equal to a value in the set. Since the elements in a multimap are not unique, the count may be greater than 1.

```
size_type count(const value_type& value) const;
```

equal_range

Returns a pair of iterators pointer to the first and last element that has a key value equal to the argument value in the multimap. If no matches are found, the range returned has a length of zero, with both iterators pointing to the first element that is greater than the value.

```
pair<const_iterator,const_iterator> equal_range(const value_type& value) const;
pair<iterator,iterator>             equal_range(const value_type& value);
```

Example 12 – The multimap container

```
#include <iostream>
#include <iomanip>
#include <map>
#include <string>
#include <cstdlib>
using namespace std;

using fraction = pair<int,int>;

ostream& operator<<(ostream&, const fraction&);
ostream& operator<<(ostream&, const pair<double,fraction>&);
ostream& operator<<(ostream&, const multimap<double,fraction>&);


int main()
{
    multimap<double,fraction> fractions;

    // insert 7 elements into the multimap
    fractions.insert(pair<double,fraction>(.75,fraction(3,4)));
    fractions.insert(pair<double,fraction>(.75,fraction{6,8}));
    fraction neg_3_4{-3,-4};
    fractions.insert(pair<double,fraction>(.75,neg_3_4));
```

```

    fraction temp_fraction{1,2};
    pair<double,fraction> temp_double_fraction;
    temp_double_fraction = {.5,temp_fraction};

    fractions.insert(temp_double_fraction);
    fractions.insert({.5,{2,4}});
    fractions.insert({.333,{1,3}});
    fractions.insert({.25,{1,4}});
    fractions.insert({.5,{1,2}});
    cout << fractions << endl << endl;

    // fractions[.4] = fraction(2,5);  // Error: no index operator
    multimap<double,fraction>::const_iterator cIt;
    cIt = fractions.find(.333);
    cout << "fractions.find(.333): " << *cIt << endl;
    cout << "fractions.find(.75): " <<*fractions.find(.75) << endl;
    cIt = fractions.find(.55);
    cout << "fractions.find(.55): " <<*cIt << endl;
    if (cIt == fractions.end())
        cout << "Can't find .55" << endl << endl;

    cout << "fractions.count(.5)=" << fractions.count(.5) << endl;
    cout << "fractions.count(.6)=" << fractions.count(.6) << endl << endl;

    cout << "Elements with key = .5" << endl;
    for (cIt = fractions.lower_bound(.5); cIt != fractions.upper_bound(.5); ++cIt)
        cout << *cIt << endl;
}

ostream& operator<<(ostream& out, const fraction& obj)
{
    out << obj.first << '/' << obj.second;
    return out;
}

ostream& operator<<(ostream& out, const pair<double,fraction>& obj)
{
  out << "first: " << obj.first << "  second: " << obj.second;
  return out;
}
ostream& operator<<(ostream& out, const multimap<double,fraction>& obj)
{
  for (auto it = obj.cbegin(); it != obj.cend(); ++it)
      out << "key: " << it->first << "  value: " << it->second << endl;
  return out;
}
```

****** Output ******

```
key: 0.25  value: 1/4
key: 0.333  value: 1/3
key: 0.5  value: 1/2
key: 0.5  value: 2/4
key: 0.5  value: 1/2
key: 0.75  value: 3/4
key: 0.75  value: 6/8
key: 0.75  value: -3/-4


fractions.find(.333): first: 0.333  second: 1/3
fractions.find(.75): first: 0.75  second: 3/4
fractions.find(.55): first: 3.95253e-323  second: 0/1072168960
Can't find .55

fractions.count(.5)=3
fractions.count(.6)=0

Elements with key = .5
first: 0.5  second: 1/2
first: 0.5  second: 2/4
first: 0.5  second: 1/2
```

unordered_set

The unordered_set container stores unique values using a hash algorithm. This allows for fast retrieval of the elements using the key value. This container was introduced in C++ 11. Elements are stored in buckets using the hash value of the elements. Elements in an unordered_set are not stored in any particular order.

Constructors

default constructor

```
unordered_set();
```

empty constructor

```
explicit unordered_set(size_type minimum_number_of_buckets,
                         const hasher& hf = hasher(),
                         const key_equal& eql = key_equal(),
                         const allocator_type& alloc = allocator_type() );
```

range constructor

```
template <class InputIterator>
         unordered_set(InputIterator first, InputIterator last,
                         size_type n = /* see below */,
                         const hasher& hf = hasher(),
                         const key_equal& eql = key_equal(),
                         const allocator_type& alloc = allocator_type() );
```

copy constructor

```
unordered_set(const unordered_set& ust);
```

move constructor

```
unordered_set(const unordered_set&& ust);
```

initializer list constructor

```
unordered_set(initializer_list<value_type> il,
                size_type n = automatically_determined,
                const hasher& hf = hasher(),
                const key_equal& eql = key_equal(),
                const allocator_type& alloc = allocator_type() );
```

Capacity Functions

size

Returns the number of elements in the unordered_set

```
size_t size() const noexcept;
```

max_size

Returns the maximum number of elements that a unordered_set can hold

```
size_t max_size() const noexcept;
```

empty

Returns whether the unordered_set is empty

```
bool empty() const noexcept;
```

Iterator Functions

begin

Returns an iterator pointing to the first element of the unordered_set

```
iterator begin() noexcept;
const_iterator begin() const noexcept;
```

bucket iterator[6]

```
local_iterator begin(size_type n);
const_local_iterator begin(size_type n) const;
```

end

[6] A bucket iterator allows you to iterate through buckets instead of individual elements

Returns an iterator pointing to the *non-existing* element beyond the end of the unordered_set

```
iterator begin() noexcept;
const_iterator begin() const noexcept;
```

bucket iterator

```
local_iterator end(size_type n);
const_local_iterator end(size_type n) const;
```

cbegin

Returns a ***const*** iterator pointing to the first element of the unordered_set

```
const_iterator cbegin() const noexcept;
const_local_iterator cbegin(size_type n) const;
```

cend

Returns a ***const*** iterator pointing to the *non-existing* element beyond the end of the unordered_set

```
const_iterator cend() const noexcept;
const_local_iterator cend(size_type n) const;
```

Lookup Functions

count

Returns the number of elements that are equal to a value in the unordered_set. Because the elements in an unordered_set must be unique, count can only return 1 or 0.

```
size_type count(const key_type& value) const;
```

find

Searches the unordered_set for a key value. Returns an iterator to the found element, otherwise it returns unordered_set::end().

```
const_iterator find(const key_type& value) const;
iterator       find(const key_type& value);
```

Modifier Functions

clear

Erases the contents of the unordered_set. Destructors are called for each object in the unordered_set.

```
void clear() noexcept;
```

erase

Removes elements from an unordered_set. Destructors are called for each object removed from the unordered_set.

```
iterator erase(const_iterator pos);
size_type erase(const key_type& key);
iterator erase(const_iterator first, const_iterator last);
```

insert

Inserts elements into an unordered_set. unordered_set elements must be unique, so duplicate values may not be inserted.

```
pair<iterator,bool> insert(const value_type& value);
pair<iterator,bool> insert(value_type&& value);
void insert(initializer_list<value_type> lst);
```

Bucket Functions

bucket

Returns a bucket number for a given key value.

```
size_type bucket (const key_type& k) const;
```

bucket_count

Returns the number of buckets in a unordered_set.

```
size_type bucket_count() const noexcept;
```

bucket_size

Returns the number of elements in a given bucket.

```
size_type bucket_size(size_type n) const;
```

Example 13 – The unordered_set container

```
#include <iostream>
#include <unordered_set>
using namespace std;

template<typename T>
ostream& operator<<(ostream& out, const unordered_set<T>& obj);

int main()
{
     unordered_set<float> floats
```

```
    {
        2.3, 6.2, 3.4, 5.6, .78, 5.5, 3.2, 0, 1.7,
        2, 4, 4.7, 6.6, 4, 7.3, 5.6, 2.1, 4.4, 5.5
    };
    cout << "floats.size() = " << floats.size() << endl;
    for (auto it = floats.cbegin(); it != floats.cend(); ++it)
    {
        cout << *it << "  ";
    }
    cout << endl;

    float temp = 2.4;
    cout << temp << " is " << (floats.find(temp) == floats.end() ? "not " : "") << "present\n";
    temp = 3.4;
    cout << temp << " is " << (floats.find(temp) == floats.end() ? "not " : "") << "present\n\n";

    floats.erase(3.4);
    floats.insert(.5);
    cout << floats << endl;

    unordered_set<int> ints;
    for (int i = 0; i < 100; i++)
        ints.insert(rand()%1000+1);
    cout << ints << endl;
}

template<typename T>
ostream& operator<<(ostream& out, const unordered_set<T>& obj)
{
    out << "size = " << obj.size() << endl;
    out << "number of buckets = " << obj.bucket_count() << endl;

    for (size_t i = 0; i < obj.bucket_count(); ++i)
    {
        if (obj.bucket_size(i))
        {
            out << "bucket #" << i << ": ";
            for (auto buckIt = obj.cbegin(i); buckIt != obj.cend(i); ++buckIt)
                out << *buckIt << "  ";
            out << endl;
        }
    }
    return out;
}
```

****** Output ******

```
floats.size() = 16
2.1  6.6  4.7  4  1.7  0  3.2  2  5.5  0.78  5.6  3.4  6.2  4.4  7.3  2.3
2.4 is not present
3.4 is present
```

```
size = 16
number of buckets = 19
bucket #0: 0
bucket #2: 5.6
bucket #3: 0.5  4.7
bucket #7: 0.78
bucket #8: 2.1
bucket #9: 2  5.5
bucket #11: 6.2
bucket #12: 4
bucket #14: 4.4  7.3  2.3
bucket #15: 6.6
bucket #17: 1.7
bucket #18: 3.2

size = 96
number of buckets = 97
bucket #2: 293
bucket #3: 779
bucket #4: 392
bucket #5: 102
bucket #6: 394
bucket #7: 7  492
bucket #9: 300
bucket #10: 107
bucket #16: 501
bucket #18: 309  891
bucket #22: 119  895
…
bucket #85: 85
bucket #86: 377  668
bucket #88: 282
bucket #89: 962
bucket #90: 963
bucket #91: 479
bucket #92: 674  383
bucket #93: 869  772
bucket #94: 967  191  870
bucket #95: 289
```

unordered_multiset

The unordered_multiset container stores values using a hash algorithm. Element values are not necessarily unique as in an unordered_set. This allow for very fast retrieval of the elements using the key value. This container was introduced in C++ 11. Elements are stored in buckets using the hash value of the elements. Elements in an unordered_multiset are not stored in any particular order.

Constructors

default constructor

```
unordered_multiset();
```

empty constructor

```
explicit unordered_multiset(size_type minimum_number_of_buckets,
                         const hasher& hf = hasher(),
                         const key_equal& eql = key_equal(),
                         const allocator_type& alloc = allocator_type() );
```

range constructor

```
template <class InputIterator>
         unordered_multiset(InputIterator first, InputIterator last,
                         size_type n = /* see below */,
                         const hasher& hf = hasher(),
                         const key_equal& eql = key_equal(),
                         const allocator_type& alloc = allocator_type() );
```

copy constructor

```
unordered_multiset(const unordered_multiset& ust);
```

move constructor

```
unordered_multiset(const unordered_multiset&& ust);
```

initializer list constructor

```
unordered_multiset(initializer_list<value_type> il,
                size_type n = automatically_determined,
                const hasher& hf = hasher(),
                const key_equal& eql = key_equal(),
                const allocator_type& alloc = allocator_type() );
```

Capacity Functions

size

Returns the number of elements in the unordered_multiset

```
size_t size() const noexcept;
```

max_size

Returns the maximum number of elements that a unordered_multiset can hold

```
size_t max_size() const noexcept;
```

empty

Returns whether the unordered_multiset is empty

```
bool empty() const noexcept;
```

Iterator Functions

begin

Returns an iterator pointing to the first element of the unordered_multiset

```
iterator begin() noexcept;
const_iterator begin() const noexcept;
```

bucket iterator[7]

```
local_iterator begin(size_type n);
const_local_iterator begin(size_type n) const;
```

end

Returns an iterator pointing to the *non-existing* element beyond the end of the unordered_multiset

```
iterator begin() noexcept;
const_iterator begin() const noexcept;
```

bucket iterator

```
local_iterator end(size_type n);
const_local_iterator end(size_type n) const;
```

cbegin

Returns a ***const*** iterator pointing to the first element of the unordered_multiset

```
const_iterator cbegin() const noexcept;
const_local_iterator cbegin(size_type n) const;
```

cend

Returns a ***const*** iterator pointing to the *non-existing* element beyond the end of the unordered_multiset

```
const_iterator cend() const noexcept;
const_local_iterator cend(size_type n) const;
```

Lookup Functions

count

Returns the number of elements that are equal to a value in the unordered_multiset

```
size_type count(const key_type& value) const;
```

[7] A bucket iterator allows you to iterate through buckets instead of individual elements

find

Searches the unordered_multiset for a key value. Returns an iterator to the found element, otherwise it returns unordered_multiset::end().

```
const_iterator find(const key_type& value) const;
iterator       find(const key_type& value);
```

equal_range

Returns a range (iterators) of elements for a key value. If the key value is not in the unordered_multiset, a pair of unordered_multiset::end() iterators is returned.

```
pair<iterator,iterator> equal_range(const key_type& value);
pair<const_iterator,const_iterator> equal_range(const key_type& value) const;
```

Modifier Functions

clear

Erases the contents of the unordered_multiset. Destructors are called for each object in the unordered_multiset.

```
void clear() noexcept;
```

erase

Removes elements from an unordered_multiset. Destructors are called for each object removed from the unordered_multiset. For the erase function with a key argument, all elements in the unordered_multiset with that key are removed.

```
iterator erase(const_iterator pos);
size_type erase(const key_type& key);
iterator erase(const_iterator first, const_iterator last);
```

insert

Inserts elements into an unordered_multiset. Duplicate values may be inserted, and hence, will be placed in the same bucket.

```
iterator insert(const value_type& value);
iterator insert(value_type&& value);
void insert(initializer_list<value_type> lst);
```

Bucket Functions

bucket

Returns a bucket number for a given key value. Buckets are numbered from 0 to bucket_count-1.

```
size_type bucket(const key_type& k) const;
```

bucket_count

Returns the number of buckets in a unordered_multiset.

```
size_type bucket_count() const noexcept;
```

bucket_size

Returns the number of elements in a given bucket.

```
size_type bucket_size(size_type n) const;
```

Example 14 – The unordered_multiset container

```
#include <iostream>
#include <iostream>
#include <unordered_set>
using namespace std;

template<typename T>
ostream& operator<<(ostream& out, const unordered_multiset<T>& obj);

int main()
 {
    unordered_multiset<int> ints;
    for (int i = 0; i < 50; i++)
    ints.insert(rand()%10+1);
    cout << ints << endl;

    cout << "ints.erase(3) = " << ints.erase(3) << endl;
    cout << "ints.erase(11) = " << ints.erase(11) << endl;
    ints.insert(5);
    cout << "ints.count(7) = " << ints.count(7) << endl;
    cout << ints << endl;
 }

 template<typename T>
 ostream& operator<<(ostream& out, const unordered_multiset<T>& obj)
 {
    out << "size = " << obj.size() << endl;
    out << "number of buckets = " << obj.bucket_count() << endl;

    for (size_t i = 0; i < obj.bucket_count(); ++i)
```

```
    {
        if (obj.bucket_size(i))
        {
            out << "bucket #" << i << ": ";
            for (auto buckIt = obj.cbegin(i); buckIt != obj.cend(i); ++buckIt)
            out << *buckIt << "  ";
            out << endl;
        }
    }
    return out;
}
```

****** Output ******

```
size = 50
number of buckets = 97
bucket #1: 1  1  1  1  1  1  1
bucket #2: 2  2  2  2  2  2
bucket #3: 3  3  3  3  3  3
bucket #4: 4  4  4  4  4  4
bucket #5: 5  5  5  5
bucket #6: 6  6  6
bucket #7: 7  7  7  7  7
bucket #8: 8  8  8  8
bucket #9: 9  9  9
bucket #10: 10  10  10  10  10  10

ints.erase(3) = 6
ints.erase(11) = 0
ints.count(7) = 5
size = 45
number of buckets = 97
bucket #1: 1  1  1  1  1  1  1
bucket #2: 2  2  2  2  2  2
bucket #4: 4  4  4  4  4  4
bucket #5: 5  5  5  5  5
bucket #6: 6  6  6
bucket #7: 7  7  7  7  7
bucket #8: 8  8  8  8
bucket #9: 9  9  9
bucket #10: 10  10  10  10  10  10
```

unordered_map

The unordered_map container implements a map using a hash algorithm. This allows fast retrieval of the elements using the key value. Like the map container, the unordered_map stores data in a key-value pair, with the key being the *look-up*. This container was introduced in C++ 11. Elements are stored in buckets using the hash value of the key. Elements in an unordered_map are not stored in any particular order.

Constructors

default constructor

```
unordered_map();         // C++14
```

empty constructor

```
explicit unordered_ map(size_type minimum_number_of_buckets,
                        const hasher& hf = hasher(),
                        const key_equal& eql = key_equal(),
                        const allocator_type& alloc = allocator_type() );
```

range constructor

```
template <class InputIterator>
         unordered_ map(InputIterator first, InputIterator last,
                         size_type n = /* see below */,
                         const hasher& hf = hasher(),
                         const key_equal& eql = key_equal(),
                         const allocator_type& alloc = allocator_type() );
```

copy constructor

```
unordered_ map(const unordered_map& obj);
```

move constructor

```
unordered_ map(const unordered_map&& obj);
```

initializer list constructor

```
unordered_map(initializer_list<value_type> il,
                size_type n = automatically_determined,
                const hasher& hf = hasher(),
                const key_equal& eql = key_equal(),
                const allocator_type& alloc = allocator_type());
```

Capacity Functions

size

Returns the number of elements in the unordered_map

```
size_t size() const noexcept;
```

max_size

Returns the maximum number of elements that a unordered_map can hold

```
size_t max_size() const noexcept;
```

empty

Returns whether the unordered_map is empty

```
bool empty() const noexcept;
```

Iterator Functions

begin

Returns an iterator pointing to the first element of the unordered_set

```
iterator begin() noexcept;
const_iterator begin() const noexcept;
```

bucket iterator[8]

```
local_iterator begin(size_type n);
const_local_iterator begin(size_type n) const;
```

end

Returns an iterator pointing to the *non-existing* element beyond the last element of the unordered_map

```
iterator begin() noexcept;
const_iterator begin() const noexcept;
```

bucket iterator

```
local_iterator end(size_type n);
const_local_iterator end(size_type n) const;
```

cbegin

Returns a ***const*** iterator pointing to the first element of the unordered_map

```
const_iterator cbegin() const noexcept;
const_local_iterator cbegin(size_type n) const;
```

cend

Returns a ***const*** iterator pointing to the *non-existing* element beyond the last element of the unordered_map

```
const_iterator cend() const noexcept;
const_local_iterator cend(size_type n) const;
```

Lookup Functions

count

Returns the number of elements that are equal to a value in the unordered_map. Because the elements in an unordered_map must be unique, count can only return 1 or 0.

---

[8] A bucket iterator allows you to iterate through buckets instead of individual elements

```
size_type count(const key_type& value) const;
```

find

Searches the unordered_map for a key value. Returns an iterator to the found element, otherwise it returns unordered_map::end().

```
const_iterator find(const key_type& value) const;
iterator       find(const key_type& value);
```

Accessor function/operator

operator[]

Returns the mapped-value for a given key-value. If the key-value is not contained in the unordered_map, then the operator inserts a new element into the map, with a *default-constructed mapped-value*.

```
mapped_type& operator[] (const key_type& key);
mapped_type& operator[] (key_type&& key);
```

at

Returns the mapped-value for a given key-value. If the key-value is not contained in the unordered_map, the function throws an *out_of_range exception*.

```
mapped_type& at(const key_type& key);
const mapped_type& at(const key_type& key) const;
```

Modifier Functions

clear

Erases the contents of the unordered_map. Destructors are called for each object in the unordered_map.

```
void clear() noexcept;
```

erase

Removes elements from an unordered_map. Destructors are called for each object removed from the unordered_map.

```
iterator erase(const_iterator pos);
size_type erase(const key_type& key);
iterator erase(const_iterator first, const_iterator last);
```

insert

Inserts elements into an unordered_map. unordered_map elements must be unique, so duplicate values may not be inserted.

```
pair<iterator,bool> insert(const value_type& value);
pair<iterator,bool> insert(value_type&& value);
void insert(initializer_list<value_type> lst);
```

Bucket Functions

bucket

Returns a bucket number for a given key value.

```
size_type bucket (const key_type& k) const;
```

bucket_count

Returns the number of buckets in a unordered_map

```
size_type bucket_count() const noexcept;
```

bucket_size

Returns the number of elements in a given bucket.

```
size_type bucket_size(size_type n) const;
```

Example 15 – The unordered_map container

```
#include <iostream>
#include <iomanip>
#include <unordered_map>
#include <string>
#include <cstdlib>
using namespace std;


using hashUS = unordered_map<unsigned,string>;

// prototypes
hashUS::iterator getInteratorForName(hashUS&, const string& name);
ostream& operator<<(ostream&, const hashUS&);
unsigned rand100();


int main()
{
    hashUS students;

```

```
    using US = pair<unsigned,string>;

    students[rand100()] = "John";
    students.insert(US(rand100(),"Paul"));
    US george{rand100(),"George"};
    students.insert(george);
    auto ringo_num = rand100();
    US ringo{ringo_num,"Ringo"};
    students.insert(move(ringo));
    cout << students << endl;

    // What does this mean?
    students[50];
    cout << students << endl;

    // Try to insert a new element using Ringo's number
    students[ringo_num] = "Ringo Clone";
    cout << students << endl;

    // What is John's number?
    cout << "John's number is " <<
         getInteratorForName(students,"John")->first << endl;

    auto it = getInteratorForName(students,"maybe");
    if (it == students.end())
        cout << "maybe ain't there" << endl;

    cout << "number of elements with key " << ringo_num << " = "
         << students.count(ringo_num) << endl;
    cout << "number of elements with key " << ringo_num+1 << " = "
         << students.count(ringo_num+1) << endl << endl;

    cout << "students.bucket_count()=" << students.bucket_count() << endl;
}

unsigned rand100()
{
    return rand() % 100 + 1;
}

ostream& operator<<(ostream& out, const hashUS& obj)
{
    out << left;
    for (auto it = obj.begin(); it != obj.end(); ++it)
    {
        out << setw(5) << it->first << setw(10) << it->second << endl;
    }
    return out;
}

hashUS::iterator
getInteratorForName(hashUS& hash_us, const string& name)
{
```

```
    for (auto it = hash_us.begin(); it != hash_us.end(); ++it)
    {
        if (it->second == name)
            return it;
    }
    return hash_us.end();
}
```

****** Output ******

```
30   Ringo
63   George
34   John
44   Paul

50
30   Ringo
63   George
34   John
44   Paul

50
30   Ringo Clone
63   George
34   John
44   Paul

John's number is 34
maybe ain't there
number of elements with key 30 = 1
number of elements with key 31 = 0
```

unordered_multimap

The unordered_map container implements a multimap using a hash algorithm. This allows fast retrieval of the elements using the key value. Element values in a unordered_multimap are pairs of key and mapped values. Unlike the unordered_map container, element key values are not unique. This container was introduced in C++ 11. The unordered_multimap container requires the <unordered_map> header file.

Member Functions

The unordered_multimap constructors and member functions are essentially the same as the unordered_map container. The following illustrates some of the differences.

erase

Erases elements in an unordered_multimap

```
iterator erase(const_iterator p);
```

Only a single element of the multimap is erased.

```
size_t erase(const value_type& value);
```

Erases all elements in the unordered_multimap with a key equal to the specified value. The function returns the number of elements erased.

insert

```
iterator insert(const value_type& val);
iterator insert(value_type&& val);
```

This version of the insert function returns only an iterator to the element that was inserted. Unlike the unordered_map::insert, there is no bool indication of success or failure. The unordered_multimap::insert does not fail like the map::insert when duplicate key values are inserted.

count

Like the unordered_map::count the function returns the number of elements that are equal to a value in the set. Since the elements in an unordered_multimap are not unique, the count may be greater than 1.

```
size_type count(const value_type& value) const;
```

equal_range

Returns a pair of iterators pointer to the first and last element that has a key value equal to the argument value in the unordered_multimap. If no matches are found, the range returned has a length of zero, with both iterators pointing to the end of the unordered_multimap.

```
pair<const_iterator,const_iterator> equal_range(const value_type& val) const;
pair<iterator,iterator>             equal_range(const value_type& value);
```

Example 16 – The unordered_multimap container

```
#include <iostream>
#include <iomanip>
#include <unordered_map>
#include <string>
#include <cstdlib>
using namespace std;

using Fraction = pair<int,int>;

ostream& operator<<(ostream& out, const Fraction& f)
{
    out << f.first << '/' << f.second;
    return out;
}

```

```
//function templates
template <typename F, typename S>
ostream& operator<<(ostream& out, const pair<F,S>& p)
{
   out << "first: " << p.first << "  second: " << p.second;
   return out;
}

template <typename K, typename V>
ostream& operator<<(ostream& out, const unordered_multimap<K,V>& m)
{
   for (auto element : m) out << element << endl;
   return out;
}

int main()
{
   unordered_multimap<double,Fraction> fractions;

   fractions.insert(pair<double,Fraction>(.75,Fraction(3,4)));
   fractions.insert(pair<double,Fraction>(.75,Fraction{6,8}));
   Fraction neg_3_4{-3,-4};
   fractions.insert(pair<double,Fraction>(.75,neg_3_4));

   Fraction temp_fraction;
   pair<double,Fraction> temp_doub_fraction;

   temp_fraction = {1,2};
   temp_doub_fraction = {.5,temp_fraction};
   fractions.insert(temp_doub_fraction);
   fractions.insert({.5,{2,4}});
   fractions.insert({.33,{1,3}});
   fractions.insert({.25,{1,4}});
   fractions.insert({.5,{1,2}});
   cout << fractions << endl;

   // fractions[.4] = fraction(2,5);  // Error: no index operator

   // find
   unordered_multimap<double,Fraction>::const_iterator cIt;
   cout << "fractions.find(.33): ";
   cIt = fractions.find(.33);
   cout << *cIt << endl;
   cout << "fractions.find(.75): " << *fractions.find(.75) << endl;
   cout << "fractions.find(.55):  ";
   cIt = fractions.find(.55);
   // check to make sure find is OK
   if (cIt == fractions.end())
      cout << "Can't find .55" << endl << endl;

   // count
   cout << "fractions.count(.5)=" << fractions.count(.5) << endl;
   cout << "fractions.count(.6)=" << fractions.count(.6) << endl
        << endl;

```

```
    // equal_range
    cout << "equal range(.5): " << endl;
    auto iters = fractions.equal_range(.5);
    cout << *(iters.first) << " / " << *(iters.second) << endl;
    for (auto iter = iters.first; iter != iters.second; ++iter)
       cout <<  *iter << endl;
    cout << endl;

    // erase
    cout << "fractions.erase(.33) = " << fractions.erase(.33)<<endl;
    cout << "fractions.erase(.5) = " << fractions.erase(.5) << endl;
    cout << "fractions.erase(.55) = " << fractions.erase(.55)< endl
         << endl;
    cout << fractions << endl;
}
```

****** Output ******

```
first: 0.25  second: 1/4
first: 0.33  second: 1/3
first: 0.5  second: 1/2
first: 0.5  second: 2/4
first: 0.5  second: 1/2
first: 0.75  second: -3/-4
first: 0.75  second: 6/8
first: 0.75  second: 3/4

fractions.find(.33): first: 0.33  second: 1/3
fractions.find(.75): first: 0.75  second: -3/-4
fractions.find(.55):  Can't find .55

fractions.count(.5)=3
fractions.count(.6)=0

equal range(.5):
first: 0.5  second: 1/2 / first: 0.75  second: -3/-4
first: 0.5  second: 1/2
first: 0.5  second: 2/4
first: 0.5  second: 1/2

fractions.erase(.33) = 1
fractions.erase(.5) = 3
fractions.erase(.55) = 0

first: 0.25  second: 1/4
first: 0.75  second: -3/-4
first: 0.75  second: 6/8
first: 0.75  second: 3/4
```

bitset

A bitset is a class that is used to store bits (binary digits). It is a templatized class in which the template parameter is the size of the sequence or array of bits. bitset is not a true STL container, since it is not templatized on a type, but it is part of the STL. Unlike the STL containers, it does not support iteration. Use of bitset requires the <bitset> header file.

Constructors

default constructor

```
constexpr bitset() noexcept;
```

integer constructor

```
constexpr bitset (unsigned long long val) noexcept;
```

string constructor

```
explicit bitset(const string& str);
```
[9]

c-string constructor

```
explicit bitset(const char* str);
```
[10]

Bit Operation Functions

set

Sets bits to 1

```
bitset& set() noexcept;
```

sets all bits to 1

```
bitset& set(size_t pos, bool val = true);
```

sets a single bit to 1 or 0

flip

flips bits

```
bitset& flip() noexcept;
```

flips all bits

```
bitset& flip(size_t pos);
```

flips a single bit

---

[9] This constructor syntax is an abstraction
[10] This constructor syntax is an abstraction

reset

resets bits to 0

```
bitset& reset() noexcept;
```

resets all bits

```
bitset& reset(size_t pos);
```

resets a single bit

Bit Access Functions

all

Test all bits are set (equal to 1)

```
bool all() const noexcept;
```

any

Test to see if any bits are set

```
bool any() const noexcept;
```

none

Test to see if no bits are set

```
bool none() const noexcept;
```

count

Returns the number of bits that are set

```
size_t count() const noexcept;
```

size

Returns the number of bits in the bitset

```
constexpr size_t size() noexcept;
```

test

Tests to see if a bit is set

```
bool test (size_t pos) const;
```

Conversion Functions

to_string

Returns the bitset as a string

```
string to_string() const;
```
[11]

to_ulong

Returns the bitset as an unsigned long

```
unsigned long to_ulong() const;
```

to_ullong

Returns the bitset as an unsigned long long

```
unsigned long long to_ullong() const;
```

Bitset operators

Member Functions

operator[]   index operator

returns the bit value at a position in the bitset

```
     bool operator[](size_t pos) const;
reference operator[](size_t pos);
```

Bitwise Operators

```
bitset& operator&=(const bitset& rhs) noexcept;

bitset& operator|=(const bitset& rhs) noexcept;

bitset& operator^=(const bitset& rhs) noexcept;

bitset& operator<<=(size_t pos) noexcept;

bitset& operator>>=(size_t pos) noexcept;

bitset operator~() const noexcept;

bitset operator<<(size_t pos) const noexcept;

bitset operator>>(size_t pos) const noexcept;
```

[11] This prototype is an abstraction

```
bool operator== (const bitset& rhs) const noexcept;

bool operator!= (const bitset& rhs) const noexcept;
```

Non-Member Functions

```
template<size_t N>
  bitset<N> operator&(const bitset<N>& lhs, const bitset<N>& rhs) noexcept;

template<size_t N>
  bitset<N> operator|(const bitset<N>& lhs, const bitset<N>& rhs) noexcept;

template<size_t N>
  bitset<N> operator^(const bitset<N>& lhs, const bitset<N>& rhs) noexcept;

template<class charT, class traits, size_t N>
  istream& operator>>(istream& is, bitset<N>& rhs);

template<class charT, class traits, size_t N>
  ostream& operator<<(ostream& os, const bitset<N>& rhs);
```

Example 17 – bitset

```
#include <iostream>
#include <bitset>
using namespace std;


int main()
{
    // Constructor
    bitset<8> b1;
    bitset<16> b2(1234);
    bitset<8> b3("1010");
    string tenten("1010");
    bitset<8> b4(tenten);

    cout << "b1 = " << b1 << endl;
    cout << "b2 = " << b2 << endl;
    cout << "b3 = " << b3 << endl;
    cout << "b4 = " << b4 << endl << endl;

    // set
    b1.set();
    b2.set(15);
    cout << "b1 = " << b1 << endl;
    cout << "b2 = " << b2 << endl << endl;

    // reset, flip
    b1.reset();
    b2.flip();
    b3.flip(0);

    cout << "b1 = " << b1 << endl;
    cout << "b2 = " << b2 << endl;
```

```
    cout << "b3 = " << b3 << endl << endl;

    // all, any, none, count, size, test
    cout << "b2.all() = " << b2.all() << endl;
    cout << "b2.any() = " << b2.any() << endl;
    cout << "b2.none() = " << b2.none() << endl;
    cout << "b2.count() = " << b2.count() << endl;
    cout << "b2.size() = " << b2.size() << endl;
    cout << "b2.test(5) = " << b2.test(5) << endl << endl;

    // to_string, to ulong
    cout << "b3.to_string() = " << b3.to_string() << endl;
    cout << "b3.to_ulong() = " << b3.to_ulong() << endl << endl;

    // index operator
    b1[7] = 1;
    cout << b1[6] << ' ' << b1 << ' ' << b1.to_ulong() << endl
         << endl;

    cout << "b1 = " << b1 << endl;
    cout << "b3 = " << b3 << endl;
    cout << "b4 = " << b4 << endl << endl;

    // bitwise operators
    cout << "b1 | b3 = " << (b1 | b3) << endl;
    cout << "b3 & b4 = " << (b3 & b4) << endl;
    cout << "b3 ^ b4 = " << (b3 ^ b4) << endl;
    cout << "b3 << 2 = " << (b3 << 2) << endl;
    cout << "~b3 = " << (~b3) << endl;
    cout << "b1 |= b3 = " << (b1 |= b3) << endl;
}
```

****** Output ******

```
b1 = 00000000
b2 = 0000010011010010
b3 = 00001010
b4 = 00001010

b1 = 11111111
b2 = 1000010011010010

b1 = 00000000
b2 = 0111101100101101
b3 = 00001011

b2.all() = 0
b2.any() = 1
b2.none() = 0
b2.count() = 10
b2.size() = 16
b2.test(5) = 1

b3.to_string() = 00001011
b3.to_ulong() = 11
```

```
0 10000000 128

b1 = 10000000
b3 = 00001011
b4 = 00001010

b1 | b3 = 10001011
b3 & b4 = 00001010
b3 ^ b4 = 00000001
b3 << 2 = 00101100
~b3 = 11110100
b1 |= b3 = 10001011
```

STL Algorithms

The STL algorithms are function templates that can be applied to STL containers.

This section needs more description and a list of the algorithms.

Example 18 – The algorithm example

```
// algorithm example
#include <iostream>
#include <algorithm>
#include <vector>
#include <list>
#include <deque>
#include <iterator>
using namespace std;


// function generator - void argument function returns container type
int RandomNumber ()
{
    return (rand()%100);
}

// binary function that returns a bool
bool funnyLessThan(const int& a, const int& b)
{
    return a % 10 < b % 10;
}

bool lessthan10(int x)
{
   return x < 10;
}


int main ()
{
```

```
     vector<int> vec(20);
     list<int> lst(20);
     deque<int> deq(20);

     // generate
     generate(vec.begin(), vec.end(), RandomNumber);

     // copy
     copy(vec.begin(), vec.end(),lst.begin());
     copy(vec.begin(), vec.end(),deq.begin());

     cout << "The initial vector of random numbers\n";
     copy(vec.begin(), vec.end(), ostream_iterator<int>(cout," "));
     cout << endl << endl;

     // sort
     sort(vec.begin(), vec.end());
     sort(deq.begin(), deq.end());
     // sort(lst.begin(), lst.end());  // Why doesn't this work?

     cout << "The vector of random numbers after the first sort\n";
     copy(vec.begin(), vec.end(), ostream_iterator<int>(cout," "));
     cout << endl << endl;

     cout << "The deque of random numbers after the sort\n";
     copy(deq.begin(), deq.end(), ostream_iterator<int>(cout," "));
     cout << endl << endl;

     sort(vec.begin(), vec.end(),funnyLessThan);
     cout << "The vector of random numbers after the second sort\n";
     copy(vec.begin(), vec.end(), ostream_iterator<int>(cout," "));
     cout << endl << endl;

     // count
     cout << "count(vec.begin(), vec.end(),8) =  " <<
  count(vec.begin(), vec.end(),8) << endl;
     cout << "count_if(vec.begin(), vec.end(),lessthan10) =  " <<
  count_if(vec.begin(), vec.end(),lessthan10) << endl << endl;

     // the remove algorithm
     string hand{"Have a nice day"};
     remove(hand.begin(),hand.end(),'a');
     cout << hand << endl;
     hand = "Have a nice day";
     string::iterator endit = remove(hand.begin(),hand.end(),'a');
     hand.erase(endit,hand.end());
     cout << hand << endl << endl;
 }
```

```
******  Output  ******

The initial vector of random numbers
41 67 34 0 69 24 78 58 62 64 5 45 81 27 61 91 95 42 27 36

The vector of random numbers after the first sort
0 5 24 27 27 34 36 41 42 45 58 61 62 64 67 69 78 81 91 95
```

```
The deque of random numbers after the sort
0 5 24 27 27 34 36 41 42 45 58 61 62 64 67 69 78 81 91 95

The vector of random numbers after the second sort
0 91 81 41 61 42 62 24 34 64 5 95 45 36 67 27 27 58 78 69

Hve  nice dyday
Hve  nice dy
```

Example 19 – The sort algorithm using compare function pointers, function objects and standard function objects

```
#include <iostream>
#include <iterator>
#include <algorithm>
#include <vector>
using namespace std;

ostream& operator<<(ostream& out, const vector<int>& v)
{
    copy(v.cbegin(),v.cend(),ostream_iterator<int>(out," "));
     out << endl;
     return out;
 }

 bool abs_lt (int i,int j)
 {
     return abs(i) < abs(j);
 }

 class MyLessThan
 {
 public:
     bool operator() (int i,int j)
     {
         return i < j;
     }
 };

 int main()
 {
     int myints[] = {32,-71,12,45,-26,80,-53,33};
     vector<int> myvector (myints, myints+8);
     cout << "1) " << myvector << endl;

     // using default comparison (operator <):
     sort (myvector.begin(), myvector.begin()+4);
     cout << "2) " << myvector << endl;

     // using function as std compare function object
     sort (myvector.begin(), myvector.end(), greater<int>());
     cout << "3) " << myvector << endl;

     // using function
     sort (myvector.begin(), myvector.end(), abs_lt);
     cout << "4) " << myvector << endl;

     // using function object (functor)
     MyLessThan object;
     sort (myvector.begin(), myvector.end(), object);
     cout << "5) " << myvector << endl;
 }
```

****** Output ******

```
1) 32 -71 12 45 -26 80 -53 33

2) -71 12 32 45 -26 80 -53 33

3) 80 45 33 32 12 -26 -53 -71

4) 12 -26 32 33 45 -53 -71 80

5) -71 -53 -26 12 32 33 45 80
```

Example 20 – The transform algorithm

```
#include <iostream>
#include <iterator>
#include <algorithm>
#include <string>
#include <vector>
#include <bitset>
using namespace std;

ostream& operator<<(ostream& out, const vector<char>& v)
 {
     copy(v.cbegin(),v.cend(),ostream_iterator<char>(out," "));
     out << endl;
     return out;
 }

 char encode(char c)
 {
     bitset<8> ch(c);
     ch.flip();
     return static_cast<char>(ch.to_ulong());
 }

 int main()
 {
     string str("HAVE A NICE DAY");
     vector<char> vc(str.size());
     vector<char> vc2(str.size());

     copy(str.cbegin(),str.cend(),vc.begin());
     cout << vc << endl;

     transform(vc.begin(),vc.end(),vc2.begin(),encode);
     cout << vc2 << endl;

     copy(vc2.begin(),vc2.end(),str.begin());
     cout << str << endl;
     transform(vc2.begin(),vc2.end(),vc.begin(),encode);
     copy(vc.begin(),vc.end(),str.begin());
     cout << str << endl;
```

```
40  }
```

****** Output ******

```
H A V E   A   N I C E   D A Y

╖ ╛ ┌ ║ ▀ ╛ ▀ ▒ ╢ ╝ ║ ▀ ╖ ╛ ª

╖╛┌║▀╛▀▒╢╝║▀╖╛ª
HAVE A NICE DAY
```

Lambda Expressions / Functions

A lambda expression allows you to write an anonymous function. This function is used like an inline function. Here's an easy example to get you started.

Lambda Basics

Example 1 – Easy Lambda example

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      auto hand = [](){cout << "Have a nice day\n";};
7      hand();
8  }
```

Explanation

[](){cout << "Have a nice day\n";} is the lambda expression. This expression returns a function. In the example the returned function is assigned to a variable, hand. The hand variable is declared as type auto. Type auto makes is easy so that you don't have to determine the type of hand. In this case, the type is void (*)(). So, you could replace line 6 with

```
void (*hand)() = [](){cout << "Have a nice day\n";};
```

In this example the lambda expression consists of 3 parts

1) The capture list, []. In this case, nothing is captured. More about that later.
2) The lambda arguments, (). In this case, there are no arguments. More about that later.
3) The body of the lambda, between the { }. This is what the lambda does.

And, here, the lambda returns void.

So, hand is a function pointer, and it is called by adding the ().

Example 2 – lambda capture and lambda arguments

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      string whatever = "kinda nice";
8
9      // capture variables (by value) in the same scope
10     auto havd = [=]()
11     {
12         cout << "Have a " << whatever <<" day\n";
13     };
14     havd();
15
16     // capture variables (by reference) in the same scope
17     auto hard = [&]()
18     {
19         whatever = "really nice";
20         cout << "Have a " << whatever <<" day\n";
21     };
22     hard();
23
24     cout << whatever << endl;
25
26     // pass a value to the lambda expression
27     auto argue = [](string arg)
28     {
29         cout << "Have a " << arg << " day\n";
30     };
31
32     argue(whatever);
33     argue("fun");
34 }
```

****** Output ******

```
Have a kinda nice day
Have a really nice day
really nice
Have a really nice day
Have a fun day
```

Explanation

The capture in line 10 is identified as [=]. This means that any variables in the same scope as the lambda expression are available in the lambda. In this case it is as if the variable whatever is passed by value.

The capture in line 17 is identified as [&]. This means that any variables in the same scope as the lambda expression are available in the lambda. In this case it is as if the variable whatever is passed by reference. Notice that whatever is changed in the lambda body.

Line 27 shows a lambda with an argument. This, like any other function argument, makes the argument available in the body of the lambda.

So, in the three cases in this example, the lambda expression creates a function pointer. This pointer is then assigned to an auto variable, and then with parentheses, the function may be called. In the third example, the function call had to provide an argument.

Example 3 – captures, arguments, and returns

```
#include <iostream>
using namespace std;

int main()
{
    int x = 8;
    auto somefunk = [=](int arg)->int { return x + arg; };
    cout << somefunk(7) << endl;

    auto obviousreturntype = [](int arg1, int arg2)
    {
        return arg1 + arg2;
    };
    cout << obviousreturntype(13,4) << endl;

    float f = 3.25;
    double d = 2.0;

    auto anotherfunk = [f,d]()
    {
        //  f = 3.25;  // Error, f is read-only
        return f + d;
    };

    auto ret1 = anotherfunk();
    cout << ret1 << ' ' << sizeof(ret1) << endl;

    auto stillanotherfunk = [f,d]() -> float
    {
        //  f = 3.25;  // Error, f is read-only
        return f + d;
    };

    auto ret2 = stillanotherfunk();
    cout << ret2 << ' ' << sizeof(ret2) << endl;
}
```

****** Output ******

```
15
17
5.25 8
5.25 4
```

Explanation

The lambda expression, on line 7, `[=](int arg)->int { return x + arg; }` captures in scope variables with [=], has an int argument and specifies an int return with `->int`. The int return is optional, since the lambda expression would return an int anyway.

The second lambda, lines 10-13, returns a function pointer that requires two int arguments and assigns it to the auto variable obvious return type. The function pointer is then exercised on line 14.

The third lambda, lines 19-23, captures two local variables, f and d, by value. Note that line 21 is commented out, an error. This illustrates how capture values are different than lambda arguments. A lambda argument, passed by value, is a local copy of some other value and hence, modifiable, locally within the lambda body, and obviously not affecting the source. A capture value is not the same as a lambda argument. The capture, as specified by [=], or in this case [f,d] specifies that variables in the same scope are read only. The exception to this is when the capture is specified as [&], or [&f,&d]. In this case, the capture is by reference and those values are modifiable. This third lambda is used on line 25 and the return from the lambda inspired function is assigned to the auto variable ret1. This ret1 variable is demonstrated using sizeof to be type double.

The fourth lambda, lines 28-32, is the same as the third lambda, except that the return type is specified as float. Hence, the double result for f + d in line 31 is then converted to float. To match the lambda returned specification.

Lambda and the STL

The return power of lambda expressions comes from their use with STL algorithms.

Example 4 – lambda and STL algorithms

```
#include <vector>
#include <algorithm>
#include <iostream>
#include <cstdlib>
#include <climits>     // for INT_MIN
using namespace std;

int main()
{
     vector<int> vec = {1,4,5,8,9,2,6,4,32,7,19};

     // print the vector
     auto printv = [](int i)
```

```
14      {
15          cout << i << "  ";
16      };
17      for_each(vec.begin(),vec.end(), printv);
18      cout << endl;
19
20      // find the maximum value in the vector
21      int max = INT_MIN;
22      for_each(vec.begin(),vec.end(),
23               [&max](int i)
24      {
25          if (i > max) max = i;
26      });
27      cout << "The maximum value is " << max << endl;
28
29      // sort the vector
30      sort(vec.begin(),vec.end(),
31           [](const int& i, const int& j)
32      {
33          return i < j;
34      });
35      for_each(vec.begin(),vec.end(), printv);
36      cout << endl;
37
38      // how many vector values are greater than 10
39      cout << "The are " <<
40           count_if(vec.begin(), vec.end(),[](int i)
41      {
42          return i > 10;
43      })
44              << " values greater than 10" << endl;
45
46      generate(vec.begin(),vec.end(),[] { return rand() % 100;});
47
48      for_each(vec.begin(),vec.end(), printv);
49      cout << endl;
50  }
```

****** Output ******

```
1  4  5  8  9  2  6  4  32  7  19
The maximum value is 32
1  2  4  4  5  6  7  8  9  19  32
The are 2 values greater than 10
1  67  34  0  69  24  78  58  62  64  5
```

Explanation

The first lambda expression, lines 12 -15, is used to display an int. This expression is assigned to the function pointer, printv. That function pointer is then used as the third argument of the for_each algorithm on line 16.

The second lambda expression, lines 22-25, is similarly used as the third argument of the for_each algorithm. In this case, the lambda expression is placed directly *inline* as the third argument.

The third lambda expression, lines 30-33, is the third argument of the sort algorithm.

The fourth lambda expression, on line 45, returns a function pointer of a function that returns a random int.

Example 5 – lambda and function templates

```
#include <vector>
#include <algorithm>
#include <iostream>
#include <iomanip>
using namespace std;

template<typename T>
void printvector(vector<T>& v)
{
    for_each(v.begin(),v.end(), [](T element)
    {
        cout << element << "  ";
    });
    cout << endl;
}

// Generic overloaded insertion operator for a vector
template<typename T>
ostream& operator<<(ostream& out, const vector<T>& v)
{
    for_each(v.begin(),v.end(), [&out](T element)
    {
        out << element << "  ";
    });
    out << endl;

    return out;
}

class Money
{
    unsigned dollars, cents;
public:
    Money(unsigned d, unsigned c)
    : dollars(d + c/100), cents(c%100) {}
    friend ostream& operator<<(ostream& out, const Money& m)
    {
        out << setfill('0');
        out << '$' << m.dollars << '.' << setw(2) << m.cents;
        out << setfill(' ');
        return out;
```

```
    }
};

int main()
{
    vector<int> vec1 = {1,4,5,8,9,2,6,4,32,7,19};
    vector<double> vec2 = {1.4,5.8,9.2,6.4,32.7,19};
    vector<Money> vec3 = {{12,34},{56,78},{910,1112}};

    printvector(vec1);
    printvector(vec2);
    printvector(vec3);
    cout << endl;
    cout << vec1;
    cout << vec2;
    cout << vec3;
}
```

****** Output ******

```
1  4  5  8  9  2  6  4  32  7  19
1.4  5.8  9.2  6.4  32.7  19
$12.34  $56.78  $921.12

1  4  5  8  9  2  6  4  32  7  19
1.4  5.8  9.2  6.4  32.7  19
$12.34  $56.78  $921.12
```

Smart Pointers

Smart pointers are used to manage dynamically allocated memory. Their use will help to avoid memory leaks, calling delete on the same pointer address twice, and assist in avoiding segmentation faults in dereferencing a null pointer. You can think of a smart pointer as a wrapper for a pointer. It is an object stored in stack memory that *owns* a pointer. The obvious advantage is that when the stack memory object goes out of scope its destructor executes and automatically releases dynamically stored memory. There are two primary template classes used for this purpose, unique_ptr and shared_ptr. Both of these were introduced in C++11. Prior to that the auto_ptr template was used for this. The auto_ptr template was deprecated in C++11.

unique_ptr

A unique_ptr is a smart pointer in which a pointer is uniquely owned by one unique_pointer. The unique_ptr template requires the <memory> header file.

Example 1 – unique_ptr example

```
#include <iostream>
#include <memory>
#include <vector>
#include <deque>
#include <iterator>
using namespace std;

class SomeClass
{
    int data_;
public:
    SomeClass(int arg = 0) : data_(arg)
    {
        cout << "SomeClass ctor called: address=" << this << endl;
    }
    ~SomeClass()
    {
        cout << "SomeClass dtor called address=" << this << endl;
    }
    int data() const
    {
        return data_;
    }
    int& data()
    {
        return data_;
    }
};

int main ()
{
```

```
    unique_ptr<int> up1(new int(6));
    cout << "*up1=" << *up1 << endl << endl;

// unique_ptr<int> up2 = new int(7);  // Error
    unique_ptr<int> up2;
//    up2 = new int;   // Error assignment operator does not take pointer argument, except ..
    up2 = nullptr;
    up2 = make_unique<int>(5);  // requires C++14
    cout << "*up2=" << *up2 << endl;
    cout << "up2.get()=" << up2.get() << endl;
    cout << "*up2.get()=" << *up2.get() << endl << endl;

    // If you don't have C++14
    unique_ptr<int> up3 = unique_ptr<int>(new int(4));
    cout << "*up3=" << *up3 << endl << endl;

    // unique_ptrs with class
    auto upS1 = make_unique<SomeClass>(7);
    cout << "upS1->data()=" << upS1->data() << endl;
    upS1->data() *= 3;
    cout << "upS1->data()=" << upS1->data() << endl << endl;

    // unique_ptr with STL container
    auto upV = make_unique<vector<int>>();  // parentheses required
    upV -> push_back(1);
    upV -> push_back(2);
    upV -> push_back(3);
    copy(upV->begin(), upV->end(),ostream_iterator<int>(cout," "));
    cout << endl << endl;

    deque<int> di={3,4,5,6,7};
    auto upDi = make_unique<deque<int>>(di);
    (*upDi)[2] = 77;
    for (auto value : *upDi) cout << value << ' ';
    cout << endl << endl;

    // release
    cout << "up1.get()=" << up1.get() << endl;
    auto ptr4up1 = up1.get();
    cout << "ptr4up1=" << ptr4up1 << endl;
    up1.release();   // Watch out for the leak!
    cout << "up1.get()=" << up1.get() << endl;
    cout << "*ptr4up1=" << *ptr4up1 << endl;
    delete ptr4up1;
    ptr4up1 = nullptr;
    cout << endl;

    // reset
    unique_ptr<int> up4(new int(4));
    cout << "up4.get()=" << up4.get() << endl;
    up4.reset();
    cout << "up4.get()=" << up4.get() << endl;
    up4 = make_unique<int>(44);
    cout << "up4.get()=" << up4.get() << endl;
```

```
    cout << "*up4=" << *up4 << endl;
    up4.reset(new int(444));
    cout << "up4.get()=" << up4.get() << endl;
    cout << "*up4=" << *up4 << endl << endl;

    auto upS2 = make_unique<SomeClass>(77);
    cout << "upS2->data()=" << upS2->data() << endl;
    upS2.reset();
    cout << endl;

    cout << "That's all folks!!!" << endl;
}
```

****** Output ******

```
*up1=6

*up2=5
up2.get()=0x8000128d0
*up2.get()=5

*up3=4

SomeClass ctor called: address=0x800012910
upS1->data()=7
upS1->data()=21

1 2 3

3 4 77 6 7

up1.get()=0x800000400
ptr4up1=0x800000400
up1.get()=0
*ptr4up1=6

up4.get()=0x800000400
up4.get()=0
up4.get()=0x800000400
*up4=44
up4.get()=0x800012970
*up4=444

SomeClass ctor called: address=0x800000400
upS2->data()=77
SomeClass dtor called address=0x800000400

That's all folks!!!
SomeClass dtor called address=0x800012910
```

shared_ptr

A shared_ptr is a smart pointer that is used to manage multiple pointer to the same memory location. The shared_ptr interface is similar to the unique_ptr. It is commonly used in reference counting application.

Example 2 – shared_ptr example

```
#include <iostream>
#include <iomanip>
#include <string>
#include <memory>
#include <vector>
using namespace std;

class Demo
{
 public:
    Demo()
    {
       cout << "default Demo ctor: " << this << endl;
    }
    Demo(const Demo&)
    {
       cout << "copy Demo ctor: " << this << endl;
    }
    ~Demo()
    {
       cout << "Demo dtor: " << this << endl;
    }
};

ostream& operator<<(ostream& out, const Demo&)
{
   out << "Demo object";
   return out;
}

template <typename T>
ostream& operator<<(ostream& out, const shared_ptr<T>& obj);

int main()
{
   shared_ptr<string> sp1;
   shared_ptr<string> sp2(nullptr);
   shared_ptr<string> sp3(new string("carrot"));
   shared_ptr<string> sp4(make_shared<string>("potato"));
   shared_ptr<string> sp5(sp3);

   cout << "sp1: " << sp1 << endl;
   cout << "sp2: " << sp2 << endl;
   cout << "sp3: " << sp3 << endl;
   cout << "sp4: " << sp4 << endl;
   cout << "sp5: " << sp5 << endl << endl;

   cout << "sp1 = sp4;" << endl;
   sp1 = sp4;
   cout << "sp1: " << sp1 << endl;
   cout << "sp4: " << sp4 << endl << endl;

```

```
    cout << "sp2 = sp3;" << endl;
    sp2 = sp3;
    cout << "sp2: " << sp2 << endl;
    cout << "sp3: " << sp3 << endl << endl;

    cout << "sp1.reset();" << endl;
    sp1.reset();
    cout << "sp1: " << sp1 << endl << endl;

    shared_ptr<Demo> sp6(nullptr);  // create "empty" shared pointer
    shared_ptr<Demo> sp7(new Demo);       // calls Demo default ctor
    shared_ptr<Demo> sp8(new Demo(*sp7));    // calls Demo copy ctor
    shared_ptr<Demo> sp9(make_shared<Demo>());  // Demo default ctor
    shared_ptr<Demo> sp10(sp7);        // calls shared_ptr copy ctor
    cout << "sp6: " << sp6 << endl;
    cout << "sp7: " << sp7 << endl;
    cout << "sp8: " << sp8 << endl;
    cout << "sp9: " << sp9 << endl;
    cout << "sp10:" << sp10 << endl << endl;

    cout << "sp6 = move(sp7);" << endl;
    sp6 = move(sp7);
    cout << "sp6: " << sp6 << endl;
    cout << "sp7: " << sp7 << endl << endl;

    cout << "sp6.reset();" << endl;
    sp6.reset();
    cout << "sp6: " << sp6 << endl;
    cout << "sp10: " << sp10 << endl << endl;

    cout << "sp10.reset();" << endl;
    sp10.reset();
    cout << "sp6: " << sp6 << endl;
    cout << "sp7: " << sp7 << endl;
    cout << "sp8: " << sp8 << endl;
    cout << "sp9: " << sp9 << endl;
    cout << "sp10:" << sp10 << endl << endl;

    cout << "That's all folks" << endl;
}

template <typename T>
ostream& operator<<(ostream& out, const shared_ptr<T>& obj)
{
    if (obj.get())
        out << setw(10) << obj.get() << "  " << setw(8) << *obj
         << "  " << obj.use_count();
    else
        out << setw(10) << obj.get();
    return out;
}
```

****** Output ******

```
sp1:         0
```

```
sp2:          0
sp3: 0x800000400    carrot  2
sp4: 0x8000128e0    potato  1
sp5: 0x800000400    carrot  2

sp1 = sp4;
sp1: 0x8000128e0    potato  2
sp4: 0x8000128e0    potato  2

sp2 = sp3;
sp2: 0x800000400    carrot  3
sp3: 0x800000400    carrot  3

sp1.reset();
sp1:          0

default Demo ctor: 0x800012970
copy Demo ctor: 0x8000129b0
default Demo ctor: 0x800012a00
sp6:          0
sp7: 0x800012970  Demo object  2
sp8: 0x8000129b0  Demo object  1
sp9: 0x800012a00  Demo object  1
sp10:0x800012970  Demo object  2

sp6 = move(sp7);
sp6: 0x800012970  Demo object  2
sp7:          0

sp6.reset();
sp6:          0
sp10: 0x800012970  Demo object  1

sp10.reset();
Demo dtor: 0x800012970
sp6:          0
sp7:          0
sp8: 0x8000129b0  Demo object  1
sp9: 0x800012a00  Demo object  1
sp10:         0

That's all folks
Demo dtor: 0x800012a00
Demo dtor: 0x8000129b0
```

Example 3 – shared_ptr solution for CIS22B/Assignment 9

The following example demonstrates a solution for a CIS22B assignment. This is the description of the assignment:

Assignment 9 - Reference Counting and a Linked List

The assignment will give you practice writing constructors and destructors, overloaded operator functions, and implementing a linked list. You will also employ a technique called reference counting.

The Plan

The goal of the assignment is to track a list of various (fruit) "items". You will read and process a transaction file (partially displayed below). The transaction file contains 5 types of transactions. You are to store a count of the items in a sorted linked list.

Details

The transaction file contains slightly over 100 random transaction entries. The five transaction type entries are:

1. ***add*** <item> - add the item to the inventory, or increase the count for that item
2. ***remove*** <item> - remove the item from the inventory, or decrease the count for that item. If the item does not exist, print error message.
3. ***print inventory*** - print the contents of the linked list (in sorted order) as shown below
4. ***misspelled transactions*** (add, remove, or print may be misspelled) - print an error message, including the line number in the file
5. ***blank lines*** - skip over these (but count the lines)

Program Requirements

1. You must write your own linked list. You may not use any STL containers.
2. The linked list **must be maintained in sorted (alphabetical) order** by the item.
3. The linked list node must contain the item name (fruit name) and a count of the number of that item that are added to the list..
4. You must print out the contents of the linked list when a "print list" transaction record appears. See sample output below.
5. You must write at least 2 classes, a "node" class and a "linked list" class. Both classes must contain constructors and the "linked list" class must have a destructor.
6. You must include at least two overloaded operators as member functions.
7. The print function of your "linked list" class must be implemented as an overloaded insertion operator function.

Input File

This is the first 32 records of the input file.

```
add banana
add pear
add orange

add orange
add apple

add peach
add plum
ad plum

remove apple
add watermelon
add pear
add plum
reomve banana
remove pear
add apple
remove orange
remove plum
add watermelon
…
remove potato

add banana
add papaya
remove watermelon
print list
remove banana
remove watermelon
...
```

Partial Program Output

```
Bad transaction: ad in line #10
Bad transaction: reomve in line #16
Unable to remove potato in line #26

Item        Quantity
apple           1
banana          2
orange          1
papaya          3
peach           1
watermelon      1
```

The solution below uses a forward_list (container) of shared pointers. The solution produces the same output that is required in the CIS22B assignment. The assignment description and input file can be found here => http://voyager.deanza.edu/~bentley/cis22b/ass9.html

```
#include <forward_list>
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <algorithm>
#include <memory>
using namespace std;

void processTransactions(const string& filename,
                         forward_list<shared_ptr<string>>&fwdlist);
shared_ptr<string> find(forward_list<shared_ptr<string>>&fwdlist,
                        const string& str);
bool remove(forward_list<shared_ptr<string>>&fwdlist,
            const string& str);
ostream& operator<<(ostream& out,
                    const forward_list<shared_ptr<string>>&lst);
ostream& operator<<(ostream& out, const shared_ptr<string>& obj);

int main()
{
    forward_list<shared_ptr < string>> fruit;
    processTransactions("c:/temp/ass9data.txt", fruit);
}

void processTransactions(const string& filename,
                         forward_list<shared_ptr<string>>&fwdlist)
{
    ifstream fin(filename);
    if (!fin)
    {
        cerr << "Unable to open file " << filename << endl;
        exit(1);
    }
    string buffer, transaction, dummy, numberString;
    string item;
    int lineNumber = 0;
    size_t pos;
    while (!fin.eof())
    {
        lineNumber++;
        getline(fin, buffer);
        if (fin.eof())
            break; // EOF check

        // A gnu/Mac compiler may store \r in the last byte.
        pos = buffer.find('\r');
        if (pos != string::npos)
            buffer.erase(pos);
```

```

        if (buffer.size() < 1)
            continue; // skip over blank line

        // get the first word of the line
        pos = buffer.find(' ');
        transaction = buffer.substr(0, pos);

        // for add or remove, get item
        if (transaction == "add" or transaction == "remove")
            item = buffer.substr(pos + 1);

        if (transaction == "add")
        {
            // Create a shared ptr for the item
            auto sharedPtr = find(fwdlist, item);
            if (!sharedPtr)
                sharedPtr = make_shared<string>(item);

            // Case 1: fwdlist is empty?
            if (fwdlist.empty())
            {
                fwdlist.push_front(sharedPtr);
            }
            // Case 2: item inserted at beginning of fwdlist?
            else if (item <= *(fwdlist.front()))
            {
                fwdlist.push_front(sharedPtr);
            }
            // Case 3: item inserted in fwdlist containing one item
            else if (++(fwdlist.begin()) == fwdlist.end())
            {
                fwdlist.insert_after(fwdlist.begin(), sharedPtr);
            }
            // Case 4: fwdlist containing more than one item
            else
            {
                // find the location to insert the new node
                auto it = fwdlist.begin();
                auto prev = fwdlist.before_begin();
                while (it != fwdlist.end() && **it < item)
                {
                    prev = it;
                    ++it;
                }
                fwdlist.insert_after(prev, sharedPtr);
            }
        }
        else if (transaction == "remove")
        {
             if (!remove(fwdlist, item))
                 cerr << "Unable to remove " << item
                      << " in line #" << lineNumber << endl;
         }
         else if (transaction == "print")
```

```
        {
            cout << fwdlist << endl;
        }
        else
        {
            cout << "Bad transaction: " << transaction
                 << " in line #" << lineNumber << endl;
        }
    }
    fin.close();
}

shared_ptr<string>
find(forward_list<shared_ptr<string>>&fwdlist, const string& str)
{
    for (auto it = fwdlist.cbegin(); it != fwdlist.cend(); ++it)
    {
        if (**it == str)
            return *it;
    }
    return nullptr;
}

bool remove(forward_list<shared_ptr<string>>&fwdlist,
            const string& str)
{
    for (auto it = fwdlist.begin(); it != fwdlist.end(); ++it)
    {
        if (**it == str)
        {
            it->reset();

            // if shared pointer count is 0, remove node
            if (it->use_count() == 0)
                fwdlist.remove(*it);
            return true;
        }
    }
    return false;
}

ostream& operator<<(ostream& out,
                    const forward_list<shared_ptr <string>>&fwdlist)
{
    out << endl << "Item       Quantity" << endl;
    out << left;
    shared_ptr<string> prev_shared_ptr = nullptr;
    for (auto it = fwdlist.cbegin(); it != fwdlist.cend(); ++it)
    {
        if (*it && prev_shared_ptr != *it)
            out << *it << endl;
        prev_shared_ptr = *it;

    }
    return out;
```

```
}

ostream& operator<<(ostream& out, const shared_ptr<string>& obj)
{
    out << left << setw(12) << *obj;
    out << right << setw(4) << obj.use_count();
    return out;
}
```

****** Output ******

```
Bad transaction: ad in line #10
Bad transaction: reomve in line #16
Unable to remove potato in line #26

Item       Quantity
apple          1
banana         2
orange         1
papaya         3
peach          1
watermelon     1

Bad transaction: prlnt in line #50

Item       Quantity
apple          2
apricot        2
banana         7
orange         1
papaya         4
peach          2
plum           1
tangarine      1

Bad transaction: aad in line #62
Unable to remove cabbage in line #81

Item       Quantity
apple          2
apricot        2
banana         7
orange         4
papaya         5
peach          5
…
```

Programming Style