## Dynamic Memory Allocation

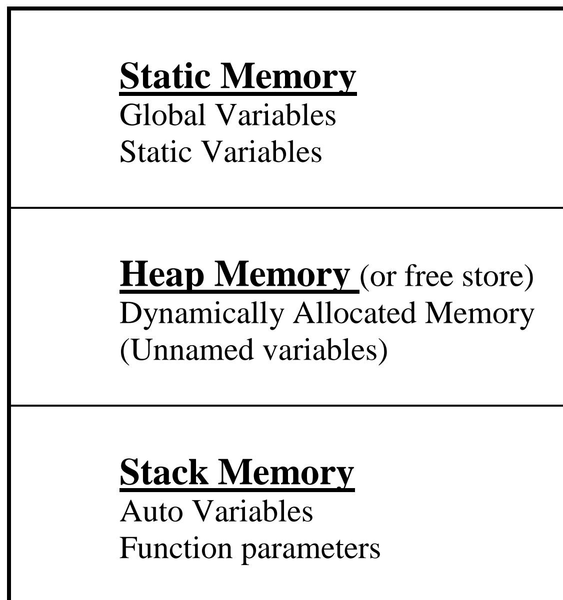
In C and C++ three types of memory are used by programs:

**Static memory** - where global and static variables live

**Stack memory** - "scratch pad" memory that is used by automatic variables.

**Heap memory** - (or free store memory) memory that may be dynamically allocated at execution time. This memory must be "managed". This memory is accessed using pointers.

Computer Memory




In C, the malloc(), calloc(), and realloc() functions are used to dynamically allocate memory from the **Heap**.

In C++, this is accomplished using the **new** and **delete** operators.

Dynamic memory allocation permits the user to create "variable-length" arrays, since only the memory that is needed may be allocated.

### The new operator

**new** is used to allocate memory during execution time. **new** returns a pointer to the address where the object is to be stored. **new** always returns a pointer to the type that follows the **new.**

Example: allocate memory for 1 int

```
int *p;          // declare a pointer to int
p = new int;     // p points to the heap space allocated for the int
```

Example: allocate memory for a float value

```
float *f = new float;      // f points to a float in the heap space
```

More examples:

```
char* ptr_char = new char;

double *trouble = new double;

int** ptr_ptr_int = new int*;



struct employee_record
{
  char empno[7];
  char name[26];
  char orgn[5];
  float salary;
  ...
};


employee_record* harry = new employee_record;
```

✔ What is harry?

```
int *p = new int(6);       // allocated and assigns
```

### Dynamic Memory Allocation for Arrays

Example - allocate memory for 10 ints

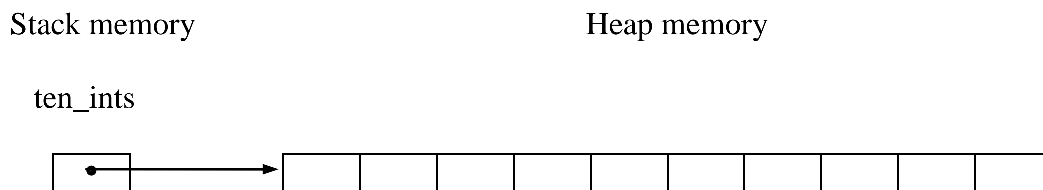
```
int* ten_ints = new int[10];
```

ten_ints is a pointer to the first of 10 ints. They will be stored in contiguous memory, so that you can access the memory like an array. For example, ten_ints[0] is the address of the first int in heap memory, ten_ints[1] is the address of the second int and so on …

It sort of looks like this:

Stack memory　　　　　　　　Heap memory

ten_ints



Type* pType = new Type[25];

Note: Even though you allocate memory for an array of Type with new, it always returns a pointer to the Type.

Example - allocate memory for a two-dimensional array

```
int (*p2d)[4] = new int[3][4];
```

Example - allocate memory for a string

```
char* text = new char[4];
strcpy(text,"hey");
```

If you attempt to dynamically allocate memory and it is not available, new will throw a **bad_alloc exception**. In pre-standard C++ **new** would return a value of 0 (or a null pointer), like malloc() in C, and most C++ programmers would use a test for 0 to check for failure of the allocation. Even though compiler manufacturers were slow to adopt this policy, most now conform to this standard. In this age of vast memory sizes, the failure of **new** is uncommon and more often than not, indicates a problem from a different source. Programmers are advised to adopt exception handling techniques (not covered in this course) for identification of this situation.

Note: **you may not initialize a dynamically allocated array as you do a single value**. Specifically,

```
int* pi = new int[5](0);          // this is illegal
```

### The delete operator

The **delete** operator is used to release the memory that was previously allocated with new. The **delete** operator does not clear the released memory, nor does it change the value of the pointer that holds the address of the allocated memory. It is probably a good idea to set the pointer to the released memory to 0. To release memory for an array that was allocated dynamically, use [] (empty braces) after the **delete** operator.

Examples:

```
int *pi = new int;
…
delete pi;
double *pd = new double[100];
…
delete [] pd;
```

Example 2-5 - Dynamic memory allocation

```
1  // File:  ex2-5.cpp
2
3  #include <iostream>
4  #include <cstdlib>
5  #include <new>
6  using namespace std;
7
8  int main(void)
9  {
10
11    int i;
12    int* pint;
13    try {
14         pint = new int[99999];
15         cout << "memory is cheap\n";
16    }
17    // if the dynamic memory allocation fails, new throws a bad_alloc
18    catch (bad_alloc& uhoh) {
19         cerr << uhoh.what() << endl;    //displays "bad allocation"
20    }
21
22    for (i = 0; i < 99999; i++) pint[i] = 0;
23
24    delete [] pint;
25
26    pint = 0;
27  }
```

```
******  Output  ******
memory is cheap
```

Example 2-6 - Dynamic Memory Allocation for char arrays

This example illustrates dynamically allocating memory to store char arrays. Storage for an array of pointers to the char arrays is not (but could be) allocated dynamically. Note each char array (name) can have a different length. Only the space required for each char array is allocated.

```
1  // File: ex2-6.cpp
2
3  #include <iostream>
4  #include <cstring>
5  using namespace std;
6
7  int main(void)
8  {
9     int i;
10    char * names[7];        // declare array of pointers to char
11    char temp[16];
12
13    // read in 7 names and dynamically allocate storage for each
14    for (i = 0; i < 7; i++)
15    {
16          cout << "Enter a name => ";
17          cin >> temp;
18          names[i] = new char[strlen(temp) + 1];
19
20          // copy the name to the newly allocated address
21          strcpy(names[i],temp);
22    }
23
24    // print out the names
25    for (i = 0; i < 7; i ++) cout << names[i] << endl;
26
27    // return the allocated memory for each name
28    for (i = 0; i < 7; i++) delete [] names[i];
29    return 0;
30  }
```

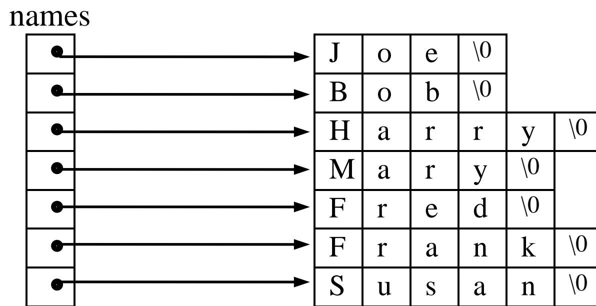
```
******  Sample Run  ******

Enter a name => Joe
Enter a name => Bob
Enter a name => Harry
Enter a name => Mary
Enter a name => Fred
Enter a name => Frank
Enter a name => Susan
Joe
Bob
Harry
```

```
Mary
Fred
Frank
Susan
```

The following illustrates the memory used in the last example:

Stack Memory                              Heap Memory




Here's another solution for the last problem:

Example 2-7 - Dynamic Memory Allocation for char arrays

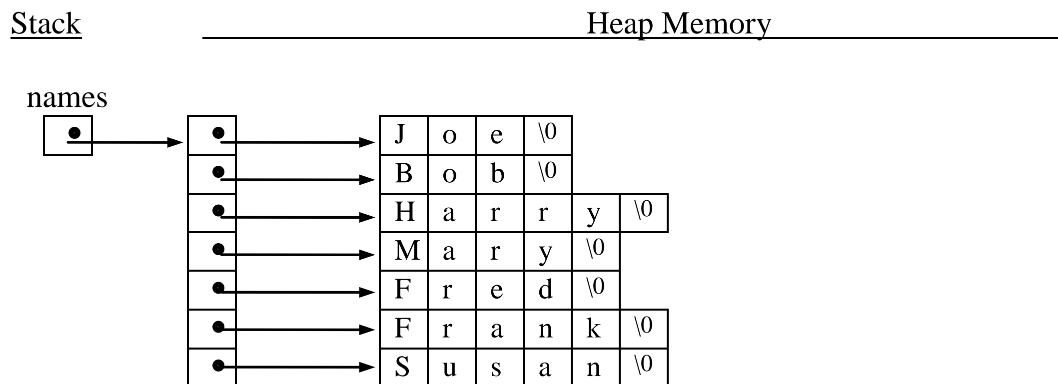
```
1  // File: ex2-7.cpp
2
3  #include <iostream>
4  #include <cstring>
5  using namespace std;
6
7  int main(void)
8  {
9     int i;
10    char ** names;         // declare pointer to pointer to char
11    char temp[16];
12    int NumberOfNames = 7;
13
14    names = new char*[NumberOfNames];
15
16    // read in 7 names and dynamically allocate storage for each
17    for (i = 0; i < NumberOfNames; i++)
18    {
19          cout << "Enter a name => ";
20          cin >> temp;
21          names[i] = new char[strlen(temp) + 1];
22
23          // copy the name to the newly allocated address
24          strcpy(names[i],temp);
25    }
26
27    // print out the names
```

```
28    for (i = 0; i < NumberOfNames; i ++) cout << names[i] << endl;
29
30    // return the allocated memory for each name
31    for (i = 0; i < NumberOfNames; i++) delete [] names[i];
32
33    delete [] names;
34    return 0;
35  }
```

```
******  Sample Run  ******

Enter a name => Joe
Enter a name => Bob
Enter a name => Harry
Enter a name => Mary
Enter a name => Fred
Enter a name => Frank
Enter a name => Susan
Joe
Bob
Harry
Mary
Fred
Frank
Susan
```

Here is what memory looks like for this example:




✔ What happens on line 20 when the user enters a name longer than 16 characters?