## Dynamic Memory allocation

- Every program needs storage to be allocated to it for various purposes.
- When you write a program, you specify what storage it needs by declaring variables and instances of classes, e.g:

```
int a,b,c;
float nums[100];
Circle myCircle(2.0,3,3); etc...
```

- Then when the program executes, it can make use of this memory
- It is not possible for variables or objects to be added during the execution of a program

## Programs and Memory

- In order to execute a program, it must be loaded into RAM (Random Access Memory).
- A certain amount of RAM is allocated as the permanent storage for your program, and this is where the global variables are stored.
- A dynamic region of memory called the stack is where local variables are stored.
- This storage is fixed at compile time.

*Remember, global variables hold their value for the lifetime of the program, while local variables only hold their value for the lifetime of the function they are declared in.*

- However, there are times when it is necessary for a program to make use of variable amounts of storage.
- For example, you may want to write a program that reads in a list of numbers (until 999 is entered) and finds their average.
- You may want to store the numbers in an array, but of what size?
- Each time the program is run, the user may enter any number of numbers, maybe 5, 50 or 5000.
- One solution is to declare an array of the maximum size that it could ever be. E.g int nums[10000].
- However, this is wasteful of memory, as you may very often use only a fraction of the space allocated.

- The solution is to use dynamic allocation of memory.
- This is the means by which a program can obtain memory while it is running.
- Powerful support for dynamic memory allocation is provided in C++, with the operator **new**.
- Memory allocated by C++'s dynamic allocation function is obtained from the heap.
- The heap is a region of free memory area that lies between your program and its permanent storage area and the stack.
- C++ also provides the operator delete, which releases the memory once the program doesn't need it anymore.

## The **new** operator

- The **new** operator returns a pointer to allocated memory from the heap.
- It returns a **NULL** pointer if there is insufficient memory to fulfill the allocation request.
- The **delete** operator frees memory previously allocated using new.
- Example of usage:

```
int *ptr; //Pointer that can point to an integer
ptr = new int;  //Now it points to allocated memory
```

## Example: new

```
int *ptr; //Pointer that can point to an integer
ptr = new int;  //Now it points to allocated memory

if(!ptr)        //NULL pointer returned
    {
    cout << "Allocation error\n";
    }
else
    {
    *p = 100;
    cout <<"Memory location: "<< ptr;
    cout <<" contains the int value: "<< *ptr <<endl;
    delete ptr;       //deallocate the memory

    }
```

## Example: new with initialisation

```
int *ptr; //Pointer that can point to an integer
ptr = new int(87);
          //Allocate memory and initialize to 87

if(!ptr)        //NULL pointer returned
    {
    cout << "Allocation error\n";
    }
else
    {
    cout <<"Memory location: "<< ptr;
    cout <<" contains the int value: "<< *ptr <<endl;
    delete ptr;       //deallocate the memory

    }
```

## Allocating arrays

- You can allocate one-dimensional arrays using new as follows:

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

- You may then use the pointer as you would a normal array.
- You may not specify an initialiser when allocating arrays.
- When an array allocated by using new is released, delete must be told that it is an array. E.g:

```
delete [] ptr;
```

Example:
**new** with arrays.

```
int *ptr;
int inum;

cout<<"How many numbers will you enter?: ";
cin>> inum;

ptr = new int[inum];

if(!ptr)
        cout<<" Allocation Error!\n";
else
        {
        for(int i=0;i<inum;i++)
                {
                cout<< "Enter number: ";
                cin>>ptr[i];
                }
        /*
         * .....Do some processing to the array .....
         */
        delete [ ] ptr;                 //Release 10 elements
        }
}
```