Dynamic Memory Allocation

The 'new' Operator

- Used to allocate dynamic memory
- Can be used to allocate a single variable/object or array of a variables/objects
- The new operator returns pointer to type allocated
- Examples:
  - int *my_char_ptr = new char;
  - int *my_int_array =new int[20];
  - Mixed *m1 = new Mixed(1,1,2);

sample1.cpp
sample2.cpp

‘delete’ Operator

- Used to free memory allocated with new operator
- The delete operator should be called on a pointer to dynamically allocated memory when it is no longer needed
- Can delete a single variable/object or an array
  - delete PointerName;
  - delete [] ArrayName;
- After delete is called on a memory region, that region should no longer be accessed by the program
- Convention is to set pointer to deleted memory to NULL

sample3.cpp
sample4.cpp

The Heap (Freestore)

- Large area of memory controlled by the operating system that is used to grant dynamic memory requests.
- It is possible to allocate memory and “lose” the pointer to that region without freeing it. This is called a memory leak.
- A memory leak can cause the heap to become full (no more memory to grant)
- If an attempt is made to allocate memory from the heap and there is not enough, an exception is generated (error)

sample5.cpp

Why use dynamic memory allocation?

- Allows data (especially arrays) to take on variable sizes (e.g. ask the user how many numbers to store, then generate an array of integers exactly that size).
- Allows locally created variables to live past end of routine.
- Allows us to create many structures used in Data Structures and Algorithms

sample6.cpp
sample7.cpp

The . and -> operators

- The dot operator is used to access an object's members
  - M1.Simplify();
  - M1.num = 5;
- But how do we access an objects members if we only have a pointer to the object?
- Perhaps we would use (*(M1_ptr)).Simplify() (What's at M1_ptr is M1)
- A shorthand for this is the arrow operator
- M1_ptr->Simplifly() is equivelant to(*(M1_ptr)).Simplify()

sample8.cpp