Pointers review

- Let a variable aa be defined as 'int *aa;', what is stored in aa?
- Let a variable aa be defined as 'int ** aa;' what is stored in aa?
- Why we should NOT return a pointer to a local variable?
- What is the pointer form of aa[100]?
- What does the NULL pointer mean?
- What is the relation between pointer and reference?
- What is the output of the following code?

```
int a = 5;
int *ptr = &a;
cout << ptr;
cout << *ptr;
```

More on operator overloading
Dynamic Memory Allocation

More on operator overloading

- All operator can be overloaded
  - Overloading an operator + can be achieved with either a member function or a friend function.
  - Overloading an operator is basically realized with a function – a function does not have to take two parameters of the same type
    - This allows operator to be overloaded in a more flexible manner by operating on different data types.
    - See the overloading of the '+' operator in fraction_overload.

Memory Allocation

- There are essentially two types of memory allocation
  - Static – Done by the compiler automatically (implicitly).
    - Global variables or objects -- memory is allocated at the start of the program, and freed when program exits; alive throughout program execution
      - Can be access anywhere in the program.
    - Local variables (inside a routine) – memory is allocated when the routine starts and freed when the routine returns.
      - A local variable cannot be accessed from another routine.
    - Allocation and free are done implicitly.
    - No need to explicitly manage memory is nice (easy to work with), but has limitations!
      - Using static allocation, the array size must be fixed.
        - Consider the grade roster for the class? What is the number of people in the class?

Memory Allocation

- There are essentially two types of memory allocation
  - Wouldn't it be nice to be able to have an array whose size can be adjusted depending on needs.
    - Dynamic memory allocation deals with this situation.
  - Dynamic – Done explicitly by programmer.
    - Programmer explicitly requests the system to allocate memory and return starting address of memory allocated (what is this?). This address can be used by the programmer to access the allocated memory.
    - When done using memory, it must be **explicitly** freed.

Explicitly allocating memory in C++: The 'new' Operator

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

  - Before the assignment, the pointer may or may not point to a legitimate memory
  - After the assignment, the pointer points to a legitimate memory.

sample1.cpp
sample2.cpp

Explicitly freeing memory in C++: the '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
  - Any new must have a corresponding delete --- if not, the program has memory leak.
  - New and delete may not be in the same routine.

sample3.cpp
(linprog and diablo)
sample4.cpp

The Heap

- Large area of memory controlled by the runtime 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
- 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_1.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?
- If we have M1_ptr = &M1, Perhaps we would use (*(M1_ptr)).Simplify()
- A shorthand for this is the arrow operator
- M1_ptr->Simplifly() is equivelant to(*(M1_ptr)).Simplify()

sample8.cpp

Explicit memroy allocation/free in C (Works also in C++)

- The malloc and free routines
  - Prototype defined in stdlib.h
  - malloc is similar to new except that it specifies the exact memory size
    - Return a (void *) -- needs to convert to the right pointer type
  - free is equivalent to delete (only one form for both single item and many items).

Sample3_malloc.cpp