CS 162C++
Dynamic Memory

Thus far, we have worked with two different types of variables. There are local variables, defined in a function and lasting as long as the function is running, and static variables, which persist outside of the function they are defined in. Dynamic memory is a third type. It is memory that is allocated on the system heap and persists from the time it is allocated until it is released. It can be passed from one function to another.

Memory Allocation and Deallocation

To allocated memory, we use the **new** operator. This operator allocates the amount of memory that we ask for on the system heap and returns its address. Since new returns an address, we have to have a pointer to save it.

For example, the following program allocates space for a single integer on the heap and returns it to the pointer. The program then stores 6 in that location on the heap (using the dereference operator on the pointer) and then outputs it.

```
#include <iostream>

using namespace std;

int main()
{
   int *ptr = new int;

   *ptr = 6;

   cout << *ptr << endl;

   delete ptr;

   return 0;
}
```

Note that we need to use the **delete** operator to release the allocated memory when we are done with it. In other programming languages, like Python or C#, there is a program called garbage collection that cleans up unused memory on the heap when the program is done with it. C and C++ do not have garbage collection by default, although it is an option in the 2020 version of the language. The advantage of manually releasing memory is that the programmer can schedule when it is going to happen instead of having their program halted at some time so that the system can clean things up.

Allocating Arrays

The above example shows allocating a single location on the heap. The new operator can also be used to allocate multiple locations or arrays as shown below.

```
#include <iostream>

using namespace std;

int main()
{
    int length;
    cout << "Enter a length for your array: ";
    cin >> length;

    int *myArray = new int[length];

    for (int i = 0; i < length; i++)
        *(myArray + i) = 2 * i + 1;

    delete [] myArray;

    return 0;
}
```

This shows the advantage of dynamic memory allocation – we can create an array of any size as the program is running. Note that when we allocated multiple locations with new, we have to tell the delete operator that it is an array that is being deleted instead of a single location by using [].