CS 162C++
Static vs. Dynamic Arrays

When you are creating an array, you can either have it statically allocate memory or dynamically allocate memory. The difference is that the size of a statically allocated array must be decided at compile time and the array goes away when it goes out of scope. A dynamically allocated array has its memory allocated on the heap, the size is determined at run time, and the memory remains until you choose to release it with the delete operator. Static allocation is what we have been doing up until this week.

The two types of arrays are identical in how they are used with subscripts.

So, the choice is whether you want your array to last longer than the existence of the function that it is defined in and if you want to define the size when you run the program rather than when you compile the program.

Static arrays

Static arrays are what we have been using up to now in the course. When you define them, you specify what size they will be and they cannot be changed. You cannot return them from functions since their memory is no longer reserved after they go out of scope.

Some examples are:

```
const int SIZE = 10;
int values[SIZE] = { 1, 3, 5, 7, 9}; // defined with length of 10, first five values initialized,
rest set to 0
```

```
char letters[] = {‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’}; // defined with length equal to the length of
initialization list
```

```
int mults[SIZE][SIZE]; // two dimensional array, not initialized
```

In all of these examples, the length of the array is specified. In the case of letters; the length of the associated initialization array defined the array length.

You must be using a constant or literal to set the size of the array, since the compiler must know how much space to allocate when it is compiling the program.

Dynamic arrays

When you define an array dynamically, you use a pointer to save the address of the array and you allocate the space for it on the heap. Since it is allocated at run time (when the program is running), you can use a variable to specify how much space to allocate.

Some examples are:

```
int * values = new int[length]; // defines an array of integers of length length
```

```
char **board = new int[row][col]; // defines a two dimensional array of chars of size row x col
```

```
int * primes = new int[number] {2, 3, 5, 7, 11}; // defines an integer array of length number and
sets the first locations to 2, 3, 5, 7, 11 (Note this only works with C++ 2011 and later.)
```

You can use a variable to define the size, since it is set at compile time.

You must use delete to free up the space on the heap when you are done.

Example program

In this program x1 is a statically allocated array and x2 is dynamically allocated. Note that x2 is freed up with the delete at the end of the program.

```
#include <iostream>

using namespace std;

int main()
{
      const int SIZE = 6;
      int x1[SIZE];

      for(int i = 0; i < SIZE; i++)
            x1[i] = i * i;

      for(int i = 0; i < SIZE; i++)
            cout << x1[i] << endl;

      int theSize;
      cout << " how big should it be? ";
      cin >> theSize;
      int *x2 = new int[theSize];

      for(int i = 0; i < theSize; i++)
            x2[i] = 2 * i;

      for(int i = 0; i < theSize; i++)
            cout << x2[i] << endl;

      delete [] x2;


    return 0;
}
```