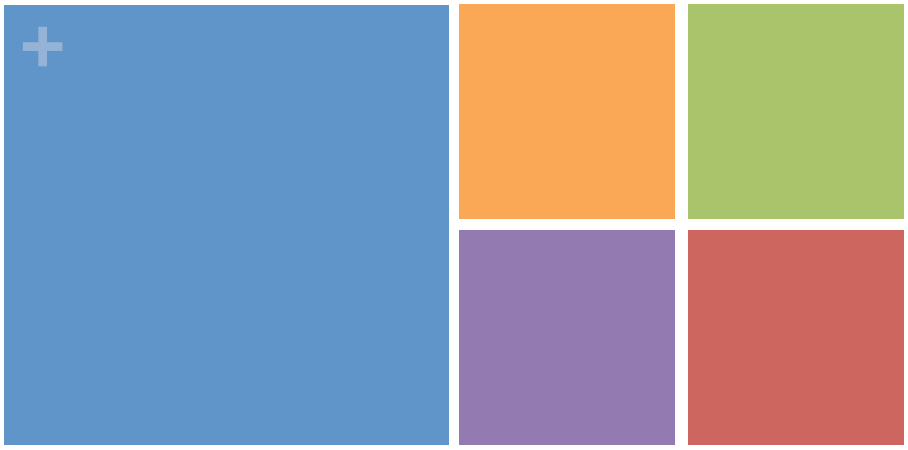

Lecture 36: More C++

Today

- Reading
  - Weiss Ch. 11.1-11.2, 11.5-11.6 (Primitive arrays)
- Objectives
  - Primitive arrays in C++
  - Iterators in C++

Recap: Exceptions

- Use try-catch block just like Java
- In C++, you can throw a variable of any type!
- The C++ standard library only throws types derived from the `exception` class
- Throw lists are deprecated as of C++11
- File I/O
  - Call the exceptions() member function on a file to tell compiler "if these bits are set, alert me by throwing an exception"

Arrays in C++

- Prefer `vector` over using a primitive array
- Prefer `string` over using an array of characters
- Still, it's useful to understand primitive arrays in C++

Declaring an array

- Declare the type and size of the array

```
int arr[3];        // notice where the brackets go
```

- Compiler allocates enough memory
  - The size of the type (in bytes) times the size of the array

4 bytes per integer * 3 integers = 12 bytes allocated

- Can use the `sizeof()` function to get the size of a type in bytes

Behind the scenes

- The name of an array is a *constant* pointer to the beginning of the allocated memory for that array
- The pointer `arr` is always guaranteed to equal `&arr[0]`

```
int main() {
   int SIZE = 3;
   int arr[SIZE];
   return 0;
}
```

Pointer Arithmetic

- You can perform addition on a pointer!
- What is the value of ptr+1?
  - If ptr is an integer pointer, then adds 4 bytes to ptr
  - If ptr is a char pointer, then adds 1 byte to ptr
- Thus, arr[i] is equivalent to arr + 4*i
- Explains why arrays start with 0 instead of 1 in C-based languages!

Implications

- The following is illegal in C++. Why?

```
int array1[3];
int array2[3];
array2 = array1;
```

- Saying array2 == array1 tests memory equality
- What happens when I pass an array as an input argument?

```
int main() {
    int arr[5];
    my_function(arr);
    return 0;
}
```

```
my_function(int array[] ) {
    // do something
}
```

Other differences

- There is no length instance variable associated with array

```
arr.length;        // doesn't work in C++
```

- Must keep track of the length of the array yourself
- No bounds checking in C++
  - Accessing beyond bounds of the array may result in segmentation fault…or may not

Dynamically allocated arrays

- Use the `new[]` operator!
  - Just like the new operator but for arrays
  - creates an array of objects on the heap
  - There is a corresponding `delete[]` operator

```
void my_function() {
    int SIZE = 3;
    int array1[SIZE];              // allocated on the stack
    int *array2 = new int[SIZE];   // allocated on the heap
}
```

After my_function returns, what memory is freed and what is not?

Iterators

- In Java, an iterator is a class that allows you to iterate over the elements of a collection
  - Implements the Iterator interface
  - What are the methods specified in the Iterator interface?
- In C++, every class has its own iterator type

```
vector<int>::iterator   // iterator for a vector
vector<int>::const_iterator // can't modify vector
map<int,int>::iterator // iterator for a map
```

Iterators

- Iterators act like pointers to values but they're not really pointers

```
vector<int> vec;
// add some integers to vec

// itr is an iterator over vec
vector<int>::iterator itr = vec.begin();

// use itr in a for-loop to loop over vec
for(; itr != vec.end(); ++itr) {
    cout << *itr << endl;
}
```