LECTURE 36: ARRAYS

Today

- Reading
  - Weiss Ch. 10, Ch. 11
- Objectives
  - More arrays in C++
  - Iterators in C++

Dynamically allocated arrays

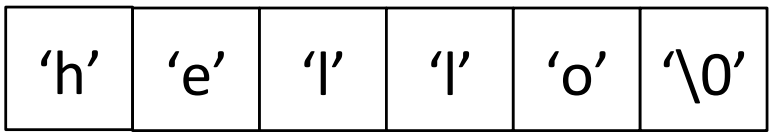
- Use the `new[ ]` operator
  - Just like the new operator but for arrays
  - creates an array of objects on the heap
  - There is a corresponding `delete[ ]` operator

```
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?

Primitive strings

- An array of characters terminated with null terminator '\0'
- Any characters after null terminator ignored by string functions
- Can pass as type (char *)



C++ Standard Template Library

- Contains familiar collections
  - vector
  - deque
  - list
  - map
  - priority_queue
  - pair

Iterators

- Iterator type depends on container and const-ness

- Map from name to mailing address:

```
unordered_map<string, string>  address_book;
```

- Possible iterator declarations:

```
unordered_map<string, string>::iterator itr;
unordered_map<string, string>::const_iterator itr;
```

Iterators

- Operators on iterators
  - itr++ and ++itr
  - *itr returns reference to element being pointed to
- Operators on collection
  - begin() – returns iterator to first element
  - end() – returns iterator pointing just past last element

Iterators

```
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;
}
```