Week 14 - Friday

CS222

Last time

- What did we talk about last time?
- OOP
- Separating header and implementation files in C++
- Overloading operators

Questions?

Project 6

Quotes

C makes it easy to shoot yourself in the foot. C++ makes it harder, but when you do, it blows away your whole leg.

Bjarne Stroustrup
Developer of C++

What's all that const?

- `const`, of course, means constant in C++
- In class methods, you'll see several different usages
- Const methods make a guarantee that they will not change the members of the object they are called on
  - `int countCabbages() const;`
- Methods can take const arguments
  - `void insert(const Coin money);`
- Methods can take const reference arguments
  - `void photograph(const Castle& fortress);`
- Why take a `const` reference when references are used to change arguments?

Templates

Templates

- Allow classes and functions to be written with a generic type or value parameter, then instantiated later
- Each necessary instantiation is generated at compile time
- Appears to function like generics in Java, but works very differently under the covers
- Most of the time you will **use** templates, not create them

Template method example

```
template<class T> void
  exchange(T& a, T& b )
{
  T temp = a;
  a = b;
  b = temp;
}
```

Template classes

- You can make a class using templates
- The most common use for these is for container classes
  - e.g. you want a **`list`** class that can be a list of anything
- The STL filled with such templates
- Unfortunately, template classes **must** be implemented entirely in the header file
  - C++ allows template classes to be separate from their headers, but no major compiler fully supports it

Template class example

```
template<class T> class Pair {
  private:
      T x;
      T y;
  public:
      Pair( const T& a, const T& b ) {
            x = a;
            y = b;
      }

      T getX() const { return x; }

      T getY() const { return y; }

      void swap() {
            T temp = x;
            x = y;
            y = temp;
      }
};
```

Programming practice

- Let's write an **`ArrayList`** class with templates!
- Methods:
  - **`void add(T element)`**
  - **`T get(int index)`**
  - **`T remove(int index)`**

STL

Standard Template Library

Containers

- `list`
- `map`
  - `multimap`
- `set`
  - `multiset`
- `stack`
- `queue`
  - `deque`
- `priority_queue`
- `vector`

Iterators

- Generalization of pointers
- No iterators for:
  - **`stack`**
  - **`queue`**
  - **`priority_queue`**
- Regular iterator operations:
  - Postfix and prefix increment and decrement
  - Assignment
  - == and !=
  - Dereference
- **`deque`** and **`vector`** iterators also have <, <=, >, >=, +, −, +=, and −=, and these containers also support [ ] access

STL example part 1

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main() {
      int count, i;
      vector<string> words;
      vector<string>::iterator index;
      string word;
```

STL example part 2

```
    cout << "How many words will you enter? ";
    cin >> count;

    for( i = 0; i < count; i++ ) {
        cin >> word;
        words.push_back( word );
    }
    for( index = words.begin(); index !=
        words.end(); index++ )
        cout << *index << endl;
    return 0;
}
```

Algorithms

- Shuffle
- Find
- Sort
- Count
- Always use the ones provided by the container, if available
- Functors provided in **`<functional>`**

Lab 14

Upcoming

Next time…

- Review up to Exam 1

Reminders

- Keep working on Project 6