C++ VI

Pass by Value, Pass by Reference,
Structs, File I/O

Pass by Value, Pass by Reference

When calling a method/function and passing in arguments:

- Java: Always passes the *value* of the arguments into the method. That is, a *copy* of the argument is used, rather than the original value
  - Primitive types (ints, longs, doubles, booleans, etc): the value itself is copied
  - Reference types (objects): The reference to the object is copied (pass reference by value)
  - E.g. See the demo
- C++: Passes the value of the arguments into the function by default. The entire value is copied, regardless of the type
  - Shallow copies, unless a copy constructor is defined to do otherwise
  - The default can be overridden by passing in a pointer or *reference variable*

Reference Variables

- Reference variables are aliases to other variables: they point to the same memory location as those other variables
- This allows C++ functions to pass by reference
- Syntax example:
  - `int myfunction(int& x){}`
- See demo

```
1  #include <iostream>
2
3  using namespace std;
4
5  int foo(int& x){
6      x = 39;
7      return x;
8  }
9
10 int main(){
11     int x = 12;
12     cout << "x before: " << x << endl;
13
14     foo(x);
15
16     cout << "x after: " << x << endl;
17
18     return 0;
19 }
```

Structs

- Structs are often used as bags of information
- Commonly used in storing and retrieving entries from databases
- Note syntax in the example on the right
- All members are public (more on this later)

```
#include <iostream>

using namespace std;

struct person {
  string name;
  int age;
};

person makePerson(){
    person x;
    x.name = "freed";
    x.age = 21;
    return x;
}

int main(){
    person p;
    p.name = "fred";
    p.age = 33;

    cout << p.name << " " << p.age << endl;

    person q = makePerson();

    cout << q.name << " " << q.age << endl;
    return 0;
}
```

File I/O

- Two classes for text file I/O:
  - `ifstream`
  - `ofstream`
- Syntax for creating an instance (ifstream, ofstream):
  - `ifstream yourVariable;`
  - `ifstream yrVar ("filename");`
- Both have .open() and .close() functions
  - Files are automatically closed when program ends, but you may need to close it earlier
- << and >> can be used with file streams
- See example on right

```cpp
#include <fstream>
#include <iostream>

using namespace std;

int main(){

    string s = "C++ for president! Make America Code Again!";

    ofstream file1 ("election.txt");

    file1 << s << endl;

    for(int i = 0; i < 100; i++){
        file1 << " caviar!" << endl;
    }

    ifstream in2 ("election.txt");

    string line;
    int counter = 0;
    while(getline(in2, line)){
        cout << "Line: " <<  counter << " " << line << endl;
        counter++;
    }

    return 0;
}
```

More File I/O

- Note: ofstream deletes everything in an existing file, creates a new if the file isn't there
  - To change behavior of ofstream, pass in additional arguments
    - `ios::app  -- Append to the file`

      `ios::ate  -- Set the current position to the end`

      `ios::trunc -- Delete everything in the file`
    - Example: `ofstream a_file ( "test.txt", ios::app );`

- Can check to see if a file can be opened before opening it with:

```
ifstream a_file ( "example.txt" );

if ( !a_file.is_open() ) {
  // The file could not be opened
}
else {
  // Safely use the file stream
}
```