CS 368-1 C++ for Java Programmers: Lecture 2 Be sure to sign the attendance sheet!
February 4, 2014

A. Preview

Previously….	**Today**	**Next Time**
• handout – course info • course intro/logistics • historical overview • simpleProg.cpp • bool is equiv to 0 or 1 • constants similar to Java • enumerated types	• structures • arrays • vectors • cardExample.cpp • editing with vim • compile and run with g++	• Start Chapter 3 • references, pointers • parameter passing • pointer basics • pointers to structs • pointers to classes

B. Announcements:

1. Program p1 has been released When a program is released, the first thing you should is read the program. Then, read it again to figure out the input and the output of the program. After that, you should try to hand-simulate the program (without writing any code.) Finally, after all those things, you should start to write code.

2. Eclipse has not been very easy to set up in C++. Since I am not a fan of Eclipse anyways, I would suggest that you simply use the g++ compiler on the linux machines. Our TA Chao recommends CodeBlocks. Since everyone's computer setup is a little bit different, I would stick with g++ if you are having trouble getting an IDE to work for you. **The TA's will be able to help you out with this much more effectively than the instructor can.**

3. Post questions on Piazza. If you post on Piazza, another student, or the TA can answer your question. In addition, other students with a similar question can learn from your question. Students who post several correct (and approved) answers may be given an extra "free day" from class.

C. The Edit-Compile-Run Cycle in with g++ (on the CS Lab machines)

Use an editor to put your code in a file ending in .cpp

To compile

```
g++ simpleprog.cpp          OR          g++ -Wall simpleprog.cpp -o  simple
```

To run

```
a.out                                   simple
```

D. Structures: A contiguous block of memory that acts as a user-defined data type.

1. the name of the struct acts as a data type

2. structs are usually declared outside of a function

3. the variables inside the struct are called **members**

4. all **members** are public so it is:
easy to publicly assign data
easy to publicly access data

5. You can quickly assign values to a struct made only of primitives using { }

6. You can copy one struct into another.

7. Can't insert an entire struct into cout
~~cout << d << endl;~~

8. Can insert a member into cout
cout << d.year << endl;

```
#include <iostream>
using namespace std;

struct Date{
      int year;   // year is a member
      int month;  // month is a member
      int day;    // day is a member
};

int main() {
      Date d1;
      d1.year = 2014;
      d1.month = 1;
      d1.day = 19;
      Date d2 = {2014, 02, 03};
      Date d3 = d2;  // copy a struct
      // next line does not compile
      // cout << d << endl;
      cout << d.year << endl;
      return 0;
}
```

E. Arrays - Similar in concept to Arrays in Java, with slightly different notation.

```
int myArray[10];          // stores space for 10 ints, but really you can have more
myArray[22] = 99;       // will not cause a compiler error, and will not immediately crash program
```

Watch out….C++ allows you to access memory outside of an array's indices!! Lots of hard-to-find errors can result.

The example Lecture02_arrays.cpp surves two purposes. First, it gives you some hints as to how to start your program P1. Secondly, it shows you how its possible to write values into an array beyond its intended use, and how your program can without your knowledge, overwrite memory allocated to other variables.

F. Vectors - Similar in concept to ArrayLists in Java, but the method/function calls are different

vector <type> name ;

Vectors are safer than arrays because they do not allow memory access outside of the array's indices.

```
#include <iostream>
#include <vector>               // like import java.util.List;
using namespace std;

int main() {
      // don't do this!!
      // it compiles, but doesn't declare a vector !!
      vector<int> notAVector();

      vector<int> list1;
      list1.push_back(17);
      list1.push_back(44);
      //cout << list1 << endl;   // does not compile
      cout << list1[2] << endl; // undefined, but will still run

      vector<int> list2;
      list2.push_back(222);
      list2.push_back(333);

      cout << list2[list2.size()-1] << endl;  // last element

}
```

G. Example: cardExample.cpp

Take time to look through this example. It shows how to use Structs with arrays. This example uses 2-dimensional arrays, but you will only need a 1-dimensional array for your program P1.

On your own:

1. (20 min) Practice declaring your own structs, arrays, and vectors. Test them out.
2. (5 min) Take a quick look at the C++ reference for vector. Do not memorize, just skim the list of member functions.
3. (10 min) Practice the compile-run process using the CS Linux machines with cardExample.cpp.
4. (10 min) Edit lecture02_arrays.cpp and cardExample.cpp in some small way. Compile and run it.
5. (10 min) Review Hexadecimal Numbers.