CS 100 Systems Programming
Homework 2

- Linux (20 pts): make and gdb
  - Create a subdirectory under your hw directory called hw2 and put all files related to this homework in that directory.
  - Define a Makefile to maintain the four programs you are writing for this homework assignment. Each class implementation includes its own header file. Some other .cc files will include .h files they need. Also define a target called `clean` to remove any files created by a make of all these programs (e.g., *.o and all the executables). **Do not use any of make's default rules** and write your Makefile by hand. Type 'touch <filename>' in terminal to modify various files, then give a 'make' command to show that only the dependent files are recompiled. The default make should remake all of the programs correctly.
  - Use the debugger, gdb, while developing your program to help you become more familiar with gdb commands and to help you track down errors. Use all the debugger commands presented in lecture this week.
- C++ (40 pts): Making Change.
  - **Do not use any STL containers**.
  - Write a class, called Coins, that is a holder for a variety of coins (similar to a pocket or coin purse). I wrote a simple main for you to use to test this first part of the homework assignment. Use three files with the names I give in the comments below: Coins.h, Coins.cc, main.cc.

```
// Coins.h          ///  The name of this file.
#include <iostream>
const int CENTS_PER_QUARTER = 25, CENTS_PER_DIME =
10, CENTS_PER_NICKEL = 5;
class Coins
{
public:
  Coins( int q, int d, int n, int p );
  void depositChange( Coins c );
  bool hasSufficientAmount( int amount );
  Coins extractChange( int amount );
  void print( ostream & out );
private:
  int quarters, dimes, nickels, pennies;
};
ostream & operator << ( ostream & out, Coins & c );
// Coins.cc will have definitions of all Coins
methods
// main.cc  /// The name of this file
#include "Coins.h"
const int CENTS_FOR_CANDYBAR = 482;
```

```
int main()
{
///  Creates a Coins object called 'pocket.'
  Coins pocket( 100, 10, 10, 100 );
  cout << "I started with " << pocket << " in my
pocket" << endl;
///  Creates a Coins object called payForCandy and
initializes it.
  Coins payForCandy = pocket.extractChange(
CENTS_FOR_CANDYBAR );
  cout << "I bought a candy bar for " <<
CENTS_FOR_CANDYBAR
     << " cents using " << payForCandy << endl;
  cout << "I have " << pocket << " left in my
pocket" << endl;
  return 0;
}
```

- Write a better main, in a file named betterMain.cc, that does the following scenario and name the executable program betterMain:
    - Create two Coins objects:
      `pocket:  5 q, 3 d, 6 n, 8 p`
      `piggyBank:  50 q, 50 d, 50 n, 50 p`
    - Buy a bag of chips that costs 68 cents taking the coins from your pocket. Display what's left in your pocket. You should take the minimum number of possible coins from your pocket. For example, to buy something worth 20 cents, you should take 2 dimes from your pocket instead of 4 nickels or 20 pennies.
    - Transfer a bunch of coins from your piggyBank to your pocket. Display how much you now have in both.
    - While vacuuming, you find loose change in your sofa. Put it in your piggyBank. Display how much you now have in your piggyBank.
    - To enter the amount found in your sofa just create a new object with the number of coins found.
- Write a third main, in a file called bestMain.cc, that presents a (text-based) menu interface to your program that allows the user to deposit change, extract change, and to print the balance in a single coins object called myCoins. Be sure you don't allow someone to extract more money than they have. (Yes, I know this is probably trivial for most of you, but I want you to get some practice with the C++ while and switch).

- Systems Programming (40 pts): Write a C++ program, called my_ls, that behaves like the Unix ls command. (**You may use STL containers here if you wish**)
    - Use `readdir` and `stat` to write a C++ program, called my_ls, to implement a

subset of `ls`. Files must be listed one name per line indented by the nesting depth of the directory. If there are no arguments to my_ls, list the files in "." (dot). Otherwise, for each command-line argument N, do the following. If N is a directory, print the name of the directory, then indent one tab and do my_ls of each entry in N (i.e., like ls -R). If N is a file, print the following information for file N in the format (like ls -lg):

octal_mode num_links size_in_bytes mod_date name

- Here is a snippet of code that reads the files in dot (the current working directory). Do not copy/paste it, but you can use it for inspiration when you write your program. Note you must not use "scandir" which is a C-lib function that reads a directory, but will leak memory unless you free each file name.

```
#include <sys/types.h>
#include <dirent.h>
#include <errno.h>
{
   char *dirName = ".";
   DIR *dirp;
   if (!(dirp  = opendir(dirName)))
   {
      cerr << "Error(" << errno << ") opening " << dirName << endl;
      return errno;
   }
   dirent *direntp;
   while ((direntp = readdir(dirp)))
      cout << direntp->d_name << endl;  // use stat here to find attributes of file
   closedir(dirp);
```

- Your homework will be graded on *well.cs.ucr.edu*, so make sure it works on that machine.
- For *all* files use the following header:

```
/*
 * Course: CS 100 Fall 2013
 *
 * First Name: <your first name>
 * Last Name: <your last name>
 * Username: <your username>
 * email address: <your UCR e-mail address>
 *
 *
 * Assignment: <assignment type and number, e.g. "Homework #2">
 *
 * I hereby certify that the contents of this file represent
 * my own original individual work. Nowhere herein is there
 * code from any outside resources such as another individual,
 * a website, or publishings unless specifically designated as
 * permissible by the instructor or TA. */
```