Lab 3: C++

Before coming to the lab you should:

1. Read the lab. The most recent version can be found at the URL: www.ee.ryerson.ca/~courses/coe808/
2. Create your project directory for lab 3. (i.e. use **mkdir lab3** within your coe808 directory.)
3. Change to your coe808/lab3, try to prepare any questions you may have about the lab.

In this C++ lab, you must use the linux g++ compiler.

Compile and Run a C++ program

The basics of C++ is pretty easy to learn. C++ is compiled languages. The following is a sample C++ program.

lesson1.cpp

```
#include <iostream.h>

int main(void) {
  int age;
  bool youngness;
  // This is a C++ comment
  cout << "Please enter your age: ";
  cin >> age;
  youngness = (age > 30) ? false : true;
  if (youngness == true) {
    cout << "Being only " << age << ", you sure are young." << endl;
  }
  else {
    cout << "Being already " << age << ", you sure are old." << endl;
  }
  return 0;
}
```

How to compile the thing? at the command line you would just type:

```
g++ lesson1.cpp -o lesson1
```

To run the program in the directory, use:

```
lesson1
```

Exercise 1: Maintaining a bank account

Write a C++ program to maintain a bank account balance. The user is first asked to enter an initial balance.

Then the user is (repeatedly) asked to enter a transaction type:

```
What would you like to do?
1 - deposit
2 - withdrawal
3 - balance inquiry
4 - quit
```

If the user enters anything else than 1,2,3,4 the program should print:

```
This is not a valid option. Try again.
What would you like to do?
1 - deposit
2 - withdrawal
3 - balance inquiry
4 - quit
```

If the user presses 1, the program should further ask for the deposit amount and *call a function to update the balance*.

If the user presses 2, the program should further ask for the withdrawal amount and *call a function to update the balance*.

If the user presses 3, the progam should *call a function to display the current balance*.

If the user presses 4, the program should print the final balance, say Good Bye, and quit. Otherwise it should go back to asking the user to enter a transaction type.

Structure your program so that the main function looks as follows:

```
int main() {
   int option = 0; //some dummy initial value
   float balance;

   //ask for initial balance;
   //here you will call a function to initialize balance

while (option != 4) {
 option = getOption();//this function should make sure the option is valid
     if (option == 1) {
```

```
            //here you will call the function to do a deposit
              //...
              };
      if (option == 2) {...};
      if (option == 3) {...};
      if (option == 4) {...};
    }
    return 0;
}
```

Your program should be dummy-proof. That is, it should handle special cases like entering a negative deposit amount or an withdrawal amount larger than current balance.

Exercise 2:
A computer program is required that reads positive integers, one per line, from the keyboard into a list. The user indicates that they are finished by entering any negative integer (the negative integer itself is not stored). The computer program then prints out the list of numbers, on a single line, separated with blank spaces. Write this program using a linked list.

Adding the following functions:

- return true/false if a given integer is in the list,
- add a node to the end of the list,
- delete the entire list,
- delete a given integer from the list.

You can test these functions by calling them in main().

The following code might come in handy:

Create a user-defined type for the list elements:

```
struct ListElement
{
  int element; // the data
  ListElement * next; // a pointer to the next element in the list
};
```

Create an empty list:

```
  ListElement * list; // a pointer to the head of the list
  list = NULL; // initialise the list (mark the end of the list with NULL).
```

Typical function prototypes:
ListElement * AddtoFront(ListElement * head, int newdata);
/* This function takes a linked list and a piece of data and creates a new node with that data, adding it to the front of the list. */
ListElement * PrintList(ListElement * list);
/* This function prints out the list. It does so by printing out the contents of the head of the list and then repointing the head to the next element in the list. It repeats this until the end of the list (a NULL) is found. The function returns a pointer to the head of the list. */

Exercise 3:

Update the solution to Exercise 3 (above) that uses C++ classes and a linked list.

Exercise 4: Update the solution to Exercise 3 (above) that uses Java classes and a linked list.

Submitting your lab

(1) Zip your lab3 assignment within your coe808 directory

zip –r lab3 lab3

(2) Attach your lab3.zip file and send e-mail to GA, Mr. Naimul Mefraz Khan, subject of your e-mail must be lab3 followed by your name and student id.
e-mail: n77khan@ryerson.ca