

CMPT 135: Lab Work Week 1

1. Given the Point struct discussed in the lecture notes, what is the output of the following program?

```
void printPoint(Point p)
{
    cout << "(" << p.x << ", " << p.y << ")";
}
void foo(Point &a, Point *b, Point c)
{
    a.x = c.y;
    a.y = b->x;
    c.x = b->y;
    c.y = a.x;
    (*b).x = c.y;
    b->y = a.x;
}
int main()
{
    Point p1, p2, p3;
    p1.x = 1;
    p1.y = 2;
    p2.x = 3;
    p2.y = 4;
    p3.x = 5;
    p3.y = 6;

    foo(p1, &p2, p3);

    cout << "Point p1 is ";
    printPoint(p1);
    cout << endl;

    cout << "Point p2 is ";
    printPoint(p2);
    cout << endl;

    cout << "Point p3 is ";
    printPoint(p3);
    cout << endl;

    system("Pause");
    return 0;
}
```

2. Given the Point struct discussed in the lecture notes, what is the output of the following program?

```
void printPoint(Point p)
{
    cout << "(" << p.x << ", " << p.y << ")";
}
void foo(Point &a, Point *b, Point c)
{
    a = *b;
    *b = c;
    c = a;
    b = &a;
}
```

```

int main()
{
    Point p1, p2, p3;
    p1.x = 1;
    p1.y = 2;
    p2.x = 3;
    p2.y = 4;
    p3.x = 5;
    p3.y = 6;

    foo(p1, &p2, p3);

    cout << "Point p1 is ";
    printPoint(p1);
    cout << endl;

    cout << "Point p2 is ";
    printPoint(p2);
    cout << endl;

    cout << "Point p3 is ";
    printPoint(p3);
    cout << endl;

    system("Pause");
    return 0;
}

```

3. Consider the programs given above. We have provided the **printPoint** function to avoid code duplication in our programs. However even using the **printPoint** function, we still have three lines of code to print every point as follows:

```

cout << "Point p1 is ";
printPoint(p1);
cout << endl;

```

So we ask ourselves "Wouldn't it be better to be able to print point objects as follows?"

```

cout << "Point p1 is " << printPoint(p1) << endl;

```

Of course this will be a syntax error because the **printPoint** function is a void function.

Moreover, we cannot print a point object directly as follows

```

cout << "Point p1 is " << p1 << endl;

```

This will also be a syntax error because the cout statement cannot be used with a struct type (see operator overloading topic to be discussed later).

So what should we do in order to be able to print a point object in one line of code nicely?

One idea would be to call a function that takes a point object and returns a string that contains an opening bracket character, the value of the x-coordinate, a comma character, the value of the y-coordinate, and a closing bracket character. The declaration of the function will look like as follows:

```

string toString(const Point &p)
{
    //Declare a string variable
    //Assign the string variable the concatenation of "(" + p.x + ", " + p.y + ")"
    //Return the string
}

```

Unfortunately this will have a syntax error because C++ concatenation of strings does not concatenate a double value (such as p.x or p.y) with the literal string values "(" or ", " or ")".

So once again, we ask ourselves "How can we concatenate a double value with string value?" The direct answer is to first convert the double value to a string value containing of digits, decimal point character and possibly the negation symbol hyphen character. But this requires a lot of mathematical computations and we wouldn't want to spend lots of time doing that. Instead C++ provides a built-in library that can convert any simple data type such as int, float, double, or bool to string values as described below.

First, we include a library as follows:

```
#include <sstream> //This allows us to use stringstream data type
```

Now, we can declare a variable of data type stringstream as follows

```
stringstream ss; //Useful to convert given data to string
```

Now the ss variable which is a data type of stringstream can be used in the same way we use the cout statement. For example,

```
int a = 2;
float b;
b = -3.0725;
double c;
cout << "Please enter a double value ";
cin >> c; //Assume we enter a value 1.8 here

ss << "a = " << a << ", b = " << b << ", and c = " << c << endl;
```

As you can see the ss uses the exact same syntax as the cout statement. However there is a major difference between ss and cout: *A cout statement automatically prints the message onto the output console window but the ss does not print anything onto the console output window. Instead the ss variable keeps the message inside a memory as a string value.* So there will be the following string value inside memory "a = 2, b = -3.0725, and c = 1.8".

The best part about ss is that we can easily get the message that is in the memory as a string value as follows:

```
string s;
s = ss.str();
```

Now the string variable s contains the message that was printed onto the memory which is "a = 2, b = -3.0725, and c = 1.8".

We can then either print the value of s using a cout statement or even better we can return the value of s from a function to a calling function such as the main function.

See the following program for demonstration purposes. Analyze the program and determine its output.

```
#include <iostream>
#include <string>
#include <sstream> //This allows us to use stringstream data type
using namespace std;

string toString(const Point &p)
{
    stringstream ss; //Useful to convert given data to string
    ss << "(" << p.x << ", " << p.y << ")";
```

```

        string s = ss.str();
        return s;
    }
    int main()
    {
        //Create and print a point
        Point p1;
        p1.x = 1;
        p1.y = 2;
        cout << "p1 is " << toString(p1) << endl;

        //Create and print another point
        Point *p2Ptr;
        p2Ptr = new Point;
        p2Ptr->x = 3;
        p2Ptr->y = 4;
        cout << "*p2Ptr is " << toString(*p2Ptr) << endl;

        //Assign the value of p1 to *p2Ptr
        *p2Ptr = p1;

        //Print both p1 and *p2Ptr
        cout << "After assigning p1 to *p2Ptr, p1 is " << toString(p1) << endl;
        cout << "After assigning p1 to *p2Ptr, *p2Ptr is " << toString(*p2Ptr) << endl;

        //Modify *p2Ptr
        (*p2Ptr).x = 5;
        p2Ptr->y = 6;

        //Print both p1 and *p2Ptr
        cout << "After modifying *p2Ptr, the point p1 is " << toString(p1) << endl;
        cout << "After modifying *p2Ptr, the point *p2Ptr is " << toString(*p2Ptr) << endl;

        //Delete all the dynamically allocated memories
        delete p2Ptr;

        system("Pause");
        return 0;
    }
}

```

4. Consider the Date struct declaration provided below designed to represent the day, month and year of a calendar date.

```

struct Date
{
    int day;
    int month;
    int year;
};

```

For simplicity, we assume that the day member variable contains values in the range [1, 30], the month member variable contains values in the range [1, 12] and the year member variable contains any non-negative integer. That is we assume that every month has exactly 30 days and every year has exactly 12 months (which is equal to 360 days). Now consider the following test program and get it to work correctly by implementing any missing functions.

```

int main()
{
    //Create two Date objects
    Date d1, d2;

    //Read input values for d1 and d2. Assume user inputs are valid inputs
    cout << "Please enter the day, month and year of the first Date ";
    cin >> d1.day >> d1.month >> d1.year;
    cout << "Please enter the day, month and year of the second Date ";
    cin >> d2.day >> d2.month >> d2.year;

    //Compute the number of days from d1 to d2. The result can be negative, positive or zero
    int diff1 = getDays(d1, d2);
    cout << "There are " << diff1 << " days from d1 to d2." << endl;

    //Compute the number of days between the two Dates. The result must be non-negative integer
    int diff2 = getDaysBetween(d1, d2);
    cout << "There are " << diff2 << " days between the two dates." << endl;

    //Compute the number of days, month and years between the two Dates.
    //The result must contain a day in the range [0, 29], month in the range [0, 11] and
    //any non negative integer for the year
    Date diff3 = getDateBetween(d1, d2);
    cout << "There are " << diff3.days << " days, " << diff3.month << " months, and "
        << diff3.year << " years between the two dates" << endl;

    system("Pause");
    return 0;
}

```

5. Declare a struct called **RationalNumber** that has two integer member variables. Call the member variables **a** (to mean the numerator) and **b** (to mean the denominator). Then write a function named **toDouble** that takes one **RationalNumber** argument and returns the rational number as double. Write a main program to test your function.
6. Write a function named **standardize** that takes one **RationalNumber** argument by reference and then makes sure that denominator is always kept positive and the numerator modified accordingly. Example if the argument contains a=2, b=-5 then the function must modify **a** and **b** so that a = -2 and b = 5 which is the same rational number as the original but written in a standard way. Discuss with your colleagues what the function should return and what the function should do if
 - Both a and b are positive,
 - Both a and b are negative,
 - a is negative and b is positive, and
 - a is positive and b is negative.

Write a main program to test your function.

7. Write a function named **reduce** that takes one **RationalNumber** argument by reference and reduces the rational number. Reduction of rational number means to write the rational number in the simplest form possible. Example 7/21 is reduced to 1/3; 2/4 is reduced to 1/2; 15/5 is reduced to 3/1 and so on so forth. Don't jump to write program. Instead think carefully how do we do reduction by hand. Hint:- What is the definition of GCD? What does it have to do with reduction of rational numbers? What does the function return? Write a main program to test your function.

8. Consider the struct declaration

```
struct StringStat
{
    string s;
    int lower, upper, digit;
};
```

Our aim is to write a program that creates an array of **StringStat** of size 5 and reads the string member variable of each element of the array from the user input. We assume the user will input strings made up lower case alphabets, upper case alphabets, and digits only. Then compute the values of the three member variables lower, upper and digit for each element of the array by counting how many lower case characters, how many upper case characters and how many digit characters the string member variable contains. Finally print each element of the array and the counts in a nicely formatted output.

9. Repeat Question #8 above but now ask the user how long the array is. Then create a dynamic array of the user specified size. Populate the array, print the statistics and finally delete the array.

10. Consider a **StudentRecord** struct declared as follows:

```
struct StudentRecord
{
    string name;
    float test, midterm, final; //test (20%), midterm (30%) and final (50%)
    char letterGrade;
};
```

Write a program that declares a StudentRecord variable, reads the values for its members (name, test, midterm and final) from the user; and then calculates the letter grade. Use this assignment of letter grades [0, 50) = F, [50, 60) = D, [60, 75) = C, [75, 90) = B, and [90, 100] = A). Also write a function named **viewRecord** that takes a StudentRecord argument and prints the argument (its name, test, midterm, final and letter grade) in a nice format. Finally call the function viewRecord from the main program in order to print the student record.

11. Write a program that creates an array of user desired size of StudentRecords and then reads the name, test, midterm and final member variables of each element from the user input. Finally calculate the letterGrade member variable of each element of the array and finally print the elements of the array using your viewRecord function described in Question 10 above.

12. Write a function named **viewRecord** (remember function overloading) that takes three arguments: a dynamic array of structs of type StudentRecord, its size, and a character and prints the elements of the array whose letter grade matches the character argument.

13. Write a function named **searchName** that takes three arguments: a dynamic array of structs of type StudentRecord, its size, and a string argument and returns the first element of the array you find whose name matches the string argument. If there is no element whose name matches the string argument, please return a StudentRecord value whose name = "Not Found", test = -1, midterm = -1, final = -1, and letterGrade = 'N'.

14. Write a function named **getTopStudentRecord** that takes two arguments: a dynamic array of structs of type StudentRecord and its size and then returns the element of the array whose total mark is the maximum. If there are more than elements with the same maximum marks, return just any one of them.

15. Write a function named **printStudentRecordsSorted** that takes two arguments: a dynamic array of structs of type `StudentRecord` and its size and then that must print the elements of the array sorted by their letter grades.
16. Expand your program in Q11 to test your functions in Questions 12, 13, 14, and 15.
17. Analyze the following program and determine its output.

```

struct NiceArray
{
    float *arr;    //pointer for a dynamic memory on the heap
    int size;      //size of the array
};
string toString(const NiceArray &A)
{
    stringstream ss;
    ss << "[";
    for (int i = 0; i < A.size - 1; i++)
        ss << A.arr[i] << ", ";
    if (A.size > 0)
        ss << A.arr[A.size-1];
    ss << "]";
    return ss.str();
}
int main()
{
    //Create and print a NiceArray object A1
    NiceArray A1;
    A1.size = 0;    //size is zero. Therefore no memory needs to be allocated on the heap
    cout << "The array A1 is " << toString(A1) << endl;

    //Create and print a NiceArray object A2
    NiceArray A2;
    A2.size = 3;
    A2.arr = new float[A2.size];
    for (int i = 0; i < A2.size; i++)
        A2.arr[i] = i+1;
    cout << "The array A2 is " << toString(A2) << endl;

    //Assign A1 the value of A2
    A1 = A2;

    //Now print both A1 and A2
    cout << "After assigning A2 to A1, the NiceArray A1 is " << toString(A1) << endl;
    cout << "After assigning A2 to A1, the NiceArray A2 is " << toString(A2) << endl;

    //Modify some elements of A1
    A1.arr[0] = 5;
    A1.arr[2] = 6;

    //Modify some elements of A2
    A2.arr[1] = -4;

    //Now print both A1 and A2
    cout << "After modifying some elements of A1 and A2, the NiceArray A1 is " << toString(A1) << endl;
    cout << "After modifying some elements of A1 and A2, the NiceArray A2 is " << toString(A2) << endl;

    //Delete all the dynamically allocated memories
    delete[] A1.arr;
    //delete[] A2.arr;    //Commented because this will create a run-time error. Explain.

    system("Pause");
    return 0;
}

```

[]

[1, 2, 3, 4]

After assigning A2 to A1, the NiceArray A1 is [1, 2, 3, 4]
[1, 2, 3, 4]

[5, -4, 6, 4]