CSCI204/MCS9204/CSCI804
Object and Generic Programming in C++
Laboratory Exercise 9 (Week 11)

Task One: String Stream (0.6 marks)

Use **ostringstream** and **istringstream** to implement the following ***two functions*** in a file **stringIO.cpp** by assuming that the operators << and >> have been overloaded for type T.

```
template<class T>
string toString (T value); // convert a value into a string

template <class T>
T toValue(string str); // extract the value from a string
```

Defined a class **Student** in a file **Student.h** and implemented the member functions in a file **Student.cpp**. The class Student can be defined like following

```
class Student {
     private:
          string firstname;
          string lastname;
          int id; // student number
          float gpa;
     public:
          // all necessary functions to be defined here
          … …
};
```

Implement a `main()` function in **stringIO.cpp** to test the two functions in the cases where T is **integer**, **double** and **class Student.**

You need to define and implement the student class properly in order to test the two template functions and assume the input string for the student class is in the following format.

First-name:last-name:student-number:gpa

For example:
John:Anderson:1234567:6.35

Testing:

Compile the program in this task by:
CC –o task1 stringIO.cpp Student.cpp

Run the program (You may use different values)
./task1
Input an integer: 12345678
Integer to string: 12345678
String to integer: 12345678
Input a double: 3214.654

Double to string: 3214.65
String to double: 3214.65
Input a student record (first-name:last-name:number:gpa): David:Smith:1234567:3.65
Student to string:
David:Smith:1234567:3.65
String to Student:
David:Smith:1234567:3.65

Note: The outputs above indicate different types of data.

Task two: Compare two vectors (0.4 marks)

The program **`vectorCompare.cpp`** intends to implement and test a predicate function:

```
bool same_elements(vector<int> a, vector<int> b)
```

that checks whether two vectors have the same elements with the same multiplicities. For example, "`1 4 9 16 9 7 4 9 11`" and "`11 1 4 9 16 9 7 4 9`" would be considered identical, but "`1 4 9 16 9 7 4 9 11`" and "`11 11 7 9 16 4 1`" would not.

And a function that removes duplicates from a vector:

```
void remove_duplicates(vector<int>& a)
```

Complete the implementation of these two functions in the given **`vectorCompare.cpp`** file . You will probably need one or more helper functions for the implementation. Place the helper functions in the specified place. Complete the main function and test your implementation.

Submission:

You should submit the files of tasks to the server by **11:59 PM on Friday, 20 May 2016** via command:

submit –u your-user-name –c CSCI204 –a L9 stringIO.cpp Student.h Student.cpp vectorCompare.cpp

and input your password.

Make sure that you use the correct file names. The UNIX system is case sensitive. You must submit all files in one *submit* command line.

After submit your assignment successfully, please check your email of confirmation. You should keep this email for the reference.

You would receive ZERO of the marks if your program codes could not be compiled correctly.

Later submission will not be accepted. Submission via e-mail is NOT acceptable.

End of Specification

Appendix: PGM Format Specification

pgm

The PGM format is designed to be extremely easy to learn and write programs for. A PGM image represents a grayscale graphic image. For most purposes, a PGM image can just be thought of an array of arbitrary integers, and all the programs in the world that think they're processing a grayscale image can easily be tricked into processing something else.

The name "PGM" is an acronym derived from "Portable Gray Map." Each PGM image consists of the following:

1. A "magic number" for identifying the file type. A pgm image's magic number is the two characters "P5".
2. Whitespace (blanks, TABs, CRs, LFs).
3. A width, formatted as ASCII characters in decimal.
4. Whitespace.
5. A height, again in ASCII decimal.
6. Whitespace.
7. The maximum gray value (Maxval), again in ASCII decimal. Must be less than 65536, and more than zero.
8. Newline or other single whitespace character.
9. A raster of `Height` rows, in order from top to bottom. Each row consists of `Width` gray values, in order from left to right. Each gray value is a number from 0 through Maxval, with 0 being black and Maxval being white. Each gray value is represented in pure binary by either 1 or 2 bytes. If the Maxval is less than 256, it is 1 byte. Otherwise, it is 2 bytes. The most significant byte is first.

   A row of an image is horizontal. A column is vertical. The pixels in the image are square and contiguous.

10. Each gray value is a number proportional to the intensity of the pixel. A value of zero is black. A value of Maxval represents white and the most intense value in the image and any other image to which the image might be compared.
11. Characters from a "#" to the next end-of-line, before the maxval line, are comments and are ignore.