SCSSE, University of Wollongong
CSCI204/MCS9204 Object and Generic Programming in C++
Laboratory Exercise 10 (Week 12)

Task One: String split (0.6)

In this task, you will define a class that stores a string and can split the string object into tokens according to the delimiters that given.
Define a class **stringSplit** in a file **stringSplit.h**. The class contains following data members: a string object; a delimiter object; a suitable (STL) container (such as vector, deque, list) object for tokens; an iterator object for the container which points the current token.
Define proper constructor(s), set functions to set string and delimiters for the class stringSplit.
Define the following member functions in the class stringSplit:

- Constructor(s).
- **setString(const std::string &)**: The function set string for the class.
- **setDelimiters(const std::string &)**: The function set delimiters for the class.
- **void split(bool)**: The function can split the string into tokens according to the delimiters. When the bool value is true, the tokens contain delimiters; when it is false, the tokens only contain non delimiter tokens. The iterator object initially points to the first token.
- **bool hasMore()**: The function will return true if more tokens left; return false when there is no more token.
- **std::string next()**: The function will return the token that the iterator points to, and the iterator points to the next token.

Implement the member functions in a file **stringSplit.cpp**. You can download the file **task1Main.cpp** to test the member functions above in the class stringSplit.

Testing:

You can compile your program
CC –o task1 task1Main.cpp stringSplit.cpp
Then run the program
./task1
The output data can be found in a text file **task1.txt**.

Task Two: Stack (0.8)

Write c++ code in a file **Convert.cpp** include main() and other functions that use a stack to convert infix notations from a given text file and print out their postfix notations.
The main() functions should get input file name from **command line** and read infix notations one by one, then convert an infix expression (notation) into a postfix notation. Assume each line contains one expression.

You may use the following algorithm for this task.

Initialize a stack object *aStack* and a container object (e.g. stack, vector) *sOutput*;
Loop
    Get the next input from the file;
    If the input is an operand, add it to *sOutput*
    Else if the input is a left parentheses, push it on *aStack*
    Else if the input is an operator
        If *aStack* is empty, push the operator on *aStack*
        Else if *aStack* top is left parentheses, push the operator on *aStack*
        Else if the operator's priority is higher than *aStack* top, push it on *aStack*
        Else
            Loop
                Pop the operator from *aStack* to sOutput
            Until *aStack* is empty, or aStack top is left parentheses, or *aStack* top priority is lower than this operator.
            Push the operator on *aStack*
    Else if the input is a right parentheses
        Loop
            Pop operators from *aStack* to sOutput
        Until a left parentheses is encountered or *aStack* is empty.
        If *aStack* is not empty
            Pop and discard left parentheses
        Else
            Output error message
End loop
Loop
    Pop operators from *aStack* to *sOutput*
Until *aStack* is empty

You should use the class *stringSplit* define in the task One to split the tokens from the expressions given in the files. Each line is an expression.

You may download test files **exp1.txt** and **exp2.txt** from web site to test your program.

Testing:
You can compile your program
CC –o task2 Convert.cpp stringSplit.cpp
Then run the program
./task2 exp1.txt
The outputs are
Infix notation=A + B * (C - D * E) / F
Postfix notation=A B C D E * - * F / +
Infix notation=AB * CDE + (RST - UV / XX) * 3 - X5
Postfix notation=AB CDE * RST UV XX / - 3 * + X5 -

Run task2 with another file
./task2 exp2.txt
The outputs are
Infix notation=23 * (8 - 3 * 2) - 5 * (30 - 18)

Postfix notation=23 8 3 2 * - * 5 30 18 - * -
Infix notation=12.53 + 21.2 * (33.2-15.4/2) - 8.5 * (2.6 + 12/2)
Postfix notation=12.53 21.2 33.2 15.4 2 / - * + 8.5 2.6 12 2 / + * -

Task Three: (0.6)

The program **vectorProc.cpp** intends to implement and test a template function:
template<class T>
bool same_elements(const vector<T> &val1, const vector<T> &val2);
that checks whether two vector containers 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 template function that removes duplicates from a vector container:
template<class T>
void remove_duplicates(vector<T> &val);
Complete the implementation of these two functions. You will probably need one or more helper (template) functions for the implementation. Place the helper functions in the file **vectorProc.cpp**. Complete the **main()** function and test your implementation.

You may download test files **input1.txt**, **input2.txt** and **input3.txt** from the web site to test your program.

Testing:
You can compile your program
CC –o task3 vectorProc.cpp
Then run the program
./task3 < input1.txt
Input number of elements for integer v1: 9
Input data: 1
Input data: 4
Input data: 9
Input data: 16
Input data: 9
Input data: 7
Input data: 4
Input data: 9
Input data: 11
Input number of elements for integer v2: 9
Input data: 11
Input data: 1
Input data: 4
Input data: 9
Input data: 16
Input data: 9
Input data: 7
Input data: 4
Input data: 9
v1 and v2 contain same elements.

After remove duplicates from v1, now it contains:
1 4 9 16 7 11

After remove duplicates from v2, now it contains:
11 1 4 9 16 7

Input number of elements for double v1: 5
Input data: 1.5
Input data: 4.3
Input data: 9.6
Input data: 16.2
Input data: 9.6
Input number of elements for double v2: 5
Input data: 9.6
Input data: 16.2
Input data: 1.5
Input data: 9.6
Input data: 4.3
v1 and v2 contain same elements.

After remove duplicates from v1, now it contains:
1.5 4.3 9.6 16.2

After remove duplicates from v2, now it contains:
9.6 16.2 1.5 4.3

Submission:

You should submit the files of task One and Two to the banshee server by 11:59 PM on Wednesday, 30 October 2013 via command:

submit –u your-user-name –c CSCI204 –a L10 stringSplit.h stringSplit.cpp Convert.cpp vectorProc.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