1 Problem Description

In our lab 3, we worked on classes as a review. In particular, we designed two classes Song and MusicLibrary and implemented them partially. This assignment will develop two more classes: Token and Expression. We wish to use them in our last homework assignment. In computer science, tokens represent basic elements or building blocks in a complex statement or expression. The actual token definition varies from applications to applications. C++ string has member functions that support tokenization and C string has library functions such as strtok (which we used in lab two and implemented in homework one) and strtok_r (extra credit problem related to homework one) to support tokenization. Sometimes due to no delimiter between tokens, we have to write code to do that as in this homework.

Let us look at a few examples of expressions and identify the tokens involved. Please pay close attention to the spaces and the lack of spaces in the expressions. In $1+2$, we have three tokens. They are $1$, $+$, $2$. In $a=5$, we have again three tokens. They are $a$, $=$, $5$. In $(a + 123 )*(ab - (3 + 4 ))$, we have fifteen tokens. They are $(, a, +, 123, ), *, (, ab, -, (, 3, +, 4, ), )$. Here is another example of an expression, which is invalid in several ways, and its tokens. In $a12 = 1?ab + - a 0123 c (a + 123 )*(ab - (3 + 4 ))$, we have 23 tokens. They $a12$, $=, 1?ab$, $+, -, a$, $0123$, $c$, $(, a, +, 123, ), *, (, ab, -, (, 3, +, 4, ), )$.

Based on the examples above, we summarize the valid tokens as follows. Close and open parentheses, ( and ), are valid tokens. Operators, $+$, $-$, $*$, $/$, and $=$ are valid tokens. Nonnegative integers, such as 5, 123, are valid tokens. Single letters from English alphabet, such as a, A, w, are valid tokens. In the example above, ab is an invalid token in our definition, but could be a valid token in other situations. Also in the last example, a12, 1?ab, and 0123 are invalid according to the definition above.

Two types of expressions are illustrated in the examples above: assignment expression and arithmetic expression. An assignment expression has three parts: an identifier (in our case a single letter), an equal sign, and an integer. An arithmetic expression involves binary operators for addition, subtraction, multiplication, division, and modulo; operands (nonnegative integers and single character letters); and parentheses for grouping. Arithmetic expressions are evaluated and constructed based on the rules we learned in grade school and high school.
2 Purpose

Understand class, member function and field, constructor. Learn and know how to use string and vector, which may be viewed as our first example of a template class and a container class. Understand the idea of token and expression. Learn how to tokenize an expression.

3 Design

*Token* class is used to store “tokens”, which has three members or fields and a few member functions. They are:

- **type** This field stores the type of the token. Its type is *token_type* whose definition is given later.
- **token** This field stores the value of the token. Its type is string.
- **priority** This field stores the priority of the token when token is an operator or a parenthesis.

**Token()** This default constructor initializes *type* to value invalid, *token* to empty string, and *priority* to -1.

**Token(string s)** This constructor takes in a string and treats it as a token. It sets the fields accordingly.

**void set(string s)** This method takes in a string and treats it as a token. It sets the fields based on the new token s.

**int value() const** This method returns the value associated with the token when token type is a integer or letter. In this assignment, if the type is integer, the integer represented in string should be converted to int and returned; if the type is letter, it returns -1. In the future assignment, we will be able to process assignment statement such as x=5 and a symbol table is used to associate x with 5. For token x, the function returns 5.

**void display() const** This method output the values of the token fields in three lines. Each line begins with the filed name, followed by “=”, followed by the value. This method is mainly for debugging use.

**token_type get_type() const** Getter for type field.

**string get_token() const** Getter for token field.

**int get_priority() const** Getter for priority field.

Typically after we tokenize the input we get a sequence of characters stored in a string, which would be a token. We then use *Token* class to figure out if the token is valid or not and what type of token it is and so on. Since an expression is really a sequence of tokens in the abstract sense, *Token* class is
used by Expression class to be designed next. The token_type, which is an enum type, is defined as follows:

enum token_type {letter, integer, op, eq, openbrace, closebrace, invalid};

If the string s, passed in the constructor or set method, is a letter then the token type is letter; is a sequence of digits then the token type is integer; is one of +, -, *, /, % then the token type is op; is = then the token type is eq; is (, open parenthesis, then the token type is openbrace; is ) close parenthesis, then the token type is closebrace; is none of the above then its token type is invalid.

Since our Token class design does not involve pointers and memory allocation, the default copy constructor and assignment operator should work just fine. We do not need to do anything here.

Expression class is used to store “expressions”, which has five members or fields and a few member functions. They are:

original This field stores the original or not yet processed “expression”. Its type is string.

tokenized This field stores the expression as a sequence of tokens. Its type is vector<Token>.

postfix This field stores the expression as a sequence of tokens in postfix notation. Its type is vector<Token>. It is not used for the current homework assignment but will be used for the last assignment. The set method of this assignment may leave it as is.

valid This field indicates whether the expression is a valid expression (no syntax error). Its type is bool. It is not used for the current homework assignment but will be used for the last assignment. The set method of this assignment may simply set it to false or leave the value as is.

type This field indicates whether the type of the expression is assignment or arithmetic expression or invalid. Its type is exp_type. It is not used for the current homework assignment but will be used for the last assignment. The set method of this assignment may simply set it to invalid or leave the value as is.

Expression() This default constructor initializes all fields to empty or false or illegal.

Expression(const string& s) This constructor takes in a string and treats it as an expression. It tokenizes the input string and sets the fields accordingly.

void set(const string& s) This method takes in a string and treats it as an expression. It tokenizes the input string and sets the fields accordingly.

void display() const This method output the values of the expression fields, one field per line. This method is mainly for debugging use.
string get_original() const Getter for original field.

vector<token> get_tokenized() const Getter for tokenized field.

vector<token> get_postfix() const Getter for postfix field. It is not used for the current assignment.

bool get_valid() const Getter for valid field. It is not used for the current assignment.

double get_type() const Getter for type field. It is not used for the current assignment.

other methods There are other methods such as to evaluate the expression and to transform the expression. Since they are not required, it is omitted for now.

When we receive a line of input, which would be an expression, we use Expression class to figure out if the expression is valid or not and what type of expression it is and so on. In order to do that, inside the Expression class the line gets tokenized. Since an expression-sequence (a portion of our program for example) is really a sequence of expressions, Expression class is used by Expression-sequence class to be designed in our last assignment. The exp_type type is defined as follows:

eenum exp_type {assignment, arithmetic, illegal};

4 Implementation

Please note the similarity between the constructor that takes a string parameter and the set method in Token class. We could implement the set method first. Then the constructor calls the set method to accomplish its tasks. The same can be said for the similarity between constructor and set method in Expression class. You should implement and test the Token class first. Then use it in the development of the Expression class. The most challenging method to implement is the set method of the Expression class in terms of figuring out the right algorithm. One of the reasons is that we cannot simply use space as a delimiter to tokenize. For instance, in (2+3)*4, we have 7 valid tokens. They are (, 2, +, 3, ), *, 4 and no space between them. But if it is given as ( 2 + 3 ) * 4, we could use space. Our method should work for any possible combinations of spaces and tokens.

5 Test and evaluation

Have a main program test the Expression class by sending the constructor and set method many different “expressions” and display the results to see if they are correct or not. You way want to be creative and cover a wide range of possible “expressions” in testing.
6 Report and documentation

A short report about things observed and things learned and understood. The report should also describe the test cases used in the main program and the reasons for each test case selected. Properly document and indent the source code. The source code must include the author name and as well as a synopsis of the file.

7 Homework submission

Use the same website (http://csce.uark.edu/mmoccaro/2014/F13/) and procedure as you have done to submit your homework1.