07 - Stacks and Queues

Post-Lab Exercises

Exercise 1

Delimiters are the braces { and }, brackets [and], and parentheses (and). Each opening or left delimiter should be matched by a closing or right delimiter; that is, every '{' should be followed by a matching '}', and so on. Also, opening delimiters that occur later in the string should be closed before those occurring earlier. Here are some examples:

```
c[d] // correct

a{b[c]d}e // correct

a{b(c]d}e // not correct; ] doesn't match (

a[b{c}d]e} // not correct; nothing matches final }

a{b(c) // not correct; Nothing matches opening {
```

Using the BracketChecker class in file 'StackandQueueClass.cpp', write a program that utilizes stacks to match and check brackets/delimiters. Also, describe the complete working of the `check()` member function of this class separately.

Programme

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
class StackX
   {
   private:
      int maxSize;                     //size of stack vector
      vector<double> stackVect;        //stack vector
      int top;                         //top of stack
   public:
//--------------------------------------------------------------
   StackX(int s);
//--------------------------------------------------------------
   void push(double j);                 //insert item
//--------------------------------------------------------------
   double pop();                        //decrement top
//--------------------------------------------------------------
   double peek();
//--------------------------------------------------------------
   bool isEmpty();
//--------------------------------------------------------------
   bool isFull();
//--------------------------------------------------------------
};
```

```
 StackX::StackX(int s) : maxSize(s), top(-1) //constructor
      {
      stackVect.reserve(maxSize);      //size the vector
      }

void StackX::push(double j)                 //put item on top
      {
      stackVect[++top] = j;            //increment top,
      }

double StackX::pop()                        //take item from top
      {
      return stackVect[top--];         //access item,
      }

double StackX::peek()                       //peek at top of stack
      {
      return stackVect[top];
      }

bool StackX::isEmpty()                      //true if stack is empty
      {
      return (top == -1);
      }

bool StackX::isFull()                       //true if stack is full
      {
      return (top == maxSize-1);
      }
//end class StackX
////////////////////////////////////////////////////////////////

// BracketChecker Class
class BracketChecker
   {
   private:
      string input;                        //input string
   public:
//--------------------------------------------------------------
   BracketChecker(string in);
//--------------------------------------------------------------
   void check();
//--------------------------------------------------------------
   };

BracketChecker::BracketChecker(string in) : input(in)   //constructor
      {  }

   void BracketChecker::check()
      {
      int stackSize = input.length();      //get max stack size
      StackX theStack(stackSize);          //make stack
      bool isError = false;                //error flag
      for(int j=0; j<input.length(); j++)  //get chars in turn
         {
         char ch = input[j];               //get char
         switch(ch)
            {
            case '{':                      //opening symbols
            case '[':
            case '(':
               theStack.push(ch);          //push them
               break;

            case '}':                      //closing symbols
            case ']':
            case ')':
               if( !theStack.isEmpty() )   //if stack not empty,
                  {
                  char chx = theStack.pop();  //pop and check
                  if( (ch=='}' && chx!='{') ||
                      (ch==']' && chx!='[') ||
                      (ch==')' && chx!='(') )
                     {
                     isError = true;
                     cout << "Mismatched delimeter: "
                          << ch << " at " << j << endl;
                     }
                  }
               else                        //prematurely empty
                  {
                  isError = true;
                  cout << "Misplaced delimiter: "
                       << ch << " at " << j << endl;
                  }
               break;
            default:    //no action on other characters
               break;
            }  //end switch
         }  //end for
      //at this point, all characters have been processed
      if( !theStack.isEmpty() )
         cout << "Missing right delimiter" << endl;
      else if( !isError )
         cout << "OK" << endl;
      }  //end check()
//end class BracketChecker
////////////////////////////////////////////////////////////////

int main()
{
    cout << "This programme tests the StackX and BracketChecker
classes" << endl;
    cout << "By using them for delimiter matching." << endl;

    string input;
```

```
while(true)
    {
        cout << "Please enter a string with delimiters and without 
whitespace." << endl;
        cin >> input;
        if( input.length() == 1 )
            break;

        BracketChecker theChecker(input);
        theChecker.check();
    }  //end while
    return 0;
}  //end main()
```

Programme Output

```
This programme tests the StackX and BracketChecker classes
By using them for delimiter matching.
Please enter a string with delimiters and without whitespace.
a[b]cd
OK
Please enter a string with delimiters and without whitespace.
c[d]
OK
Please enter a string with delimiters and without whitespace.
a{b[c]d}e
OK
Please enter a string with delimiters and without whitespace.
a{b(c]d}e
Mismatched delimeter: ] at 5
Please enter a string with delimiters and without whitespace.
a[b{c}d]e}
Misplaced delimiter: } at 9
Please enter a string with delimiters and without whitespace.
a{b(c)
Missing right delimiter
Please enter a string with delimiters and without whitespace.
q
Program ended with exit code: 0
```

Explanation

The test programme described in this section uses the StackX and BracketChecker classes defined in StackAndQueueClass.cpp to match braces, brackets, and parentheses entered by the user.

The StackX class is a straightforward vector-based implementation of the Stack ADT, with push, pop, peek, isEmpty, and isFull methods. These methods use the built-in vector push and pop functions to implement stack operations along with a private member variable top to keep track of the index in the vector from which elements can be accessed and removed.

The real use of the StackX class is in the BracketChecker class. This class has only one member variable ‘input’, which is the string an instance of the BracketChecker will check for missing or mismatched delimiters. The class initializes input to the string that has to be checked using its constructors. Its second method, check, performs the actual test for missing or mismatched delimiters.

check()

First, the method gets the total number of characters in string to be checked (input) using the built-in string library function length(). It uses this variable as the upper bound of a ‘for’ loop in which the method parses the entire input string character by character and checks for delimiters. The same value is used to initialize a StackX object called theStack which will be used to store delimiters in the input string. The Boolean variable isError is initialized to false at the beginning of the method and will be used as a flag to signal when and if an error in delimiter matching is found.

The method creates a for loop which parses the input string one character at a time. If the character being parsed is an opening bracket, brace, or parenthesis, it is inserted on top of theStack. This helps the programme keep track of which delimiters appear in the input string as well as the order in which they appear. The function also checks for closing braces, brackets, and parentheses, and if it finds one in the input string, it enters a complex if block.

The if block will first check if the stack is not empty so as to prevent the pop operation from being called on an empty Stack (can’t remove elements from an empty stack). If the stack is not empty, it pops the character on top of the stack and stores it in a variable chx. The if block then attempts to match delimiters. If the last closing delimiter was { then, assuming the user has entered matching delimiters in the right order, the value popped from the stack should be }. The same applies for the () and [ ] delimiter pairs.

The if block tests this condition for all three pairs of delimiters. If the expression evaluates to false, then the user has made an error in entering delimiter pairs. The error flag is set to true and an error message is printed to the console to inform the user about the error. If the error flag is still false at the end of the for loop (when all characters in the string have been processed) then the user entered a syntactically correct expression and is informed of this via the console. If the user did missed a delimiter, then the stack will not be empty (the stack will only be empty if, for every closing delimiter entered, a corresponding opening delimiter is popped).