Author Identification Block
Author: Ms. Petunia Perfect
Student ID: *00000000
E-Mail: ppfercet@uco.edu
Course: CMSC 2613 – Programming II
CRN: 21641, Spring, 2011
Report: r02
Due: March 23, 2011
Account: tt000

Scoring Block

<table>
<thead>
<tr>
<th>Component</th>
<th>Available</th>
<th>Earned</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cover page</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Interface</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Algorithm</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Source code</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>25</strong></td>
<td><strong>25</strong></td>
<td></td>
</tr>
</tbody>
</table>
1. **Overview**

Project 2 is a program that determines if a string contains a balanced set of brackets. Brackets consist of the pairs (), [], and {}. A string is a sequence of characters containing no white space. White space is a sequence of one or more blank characters, new line characters, or tab characters. A string is balanced if an opening bracket, (, [, or {} is matched by the corresponding closing bracket, ), ], or ). Brackets are matched in a last-in-first-out order. If an opening curly brace, {}, appeared in the string then the next bracket in the string must be a closing curly brace }. Any sequence of characters that are not brackets can appear between the opening and closing brackets.

2. **Interface Description**

Project p02 has an external command line interface and a single internal interface to class Stack. Both interfaces are described below.

1. **Command Line Interface**

Project 2 can be invoked with zero, one, or two program parameters. The first program parameter is the input file name. The second parameter is the output file name. Sample command lines together with corresponding actions by program p02 are shown below. Boldfaced type indicates data entered at the keyboard by the user.

```
$ p02
Enter the input file name: i02.dat
Enter the output file name: o02.dat

$ p02 i02.dat
Enter the output file name: o02.dat

$ p02 i02.dat o02.dat
```

Candidate strings containing brackets are given in figure 1 and corresponding results are shown in figure 2.

<table>
<thead>
<tr>
<th>Candidate</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(((()[[[[]]])))</td>
</tr>
<tr>
<td>2</td>
<td>(((()[[[]])])</td>
</tr>
<tr>
<td>3</td>
<td>(o(t[b]m[s(t(f)s(t)d)f]s)us</td>
</tr>
<tr>
<td>4</td>
<td>(l(l[e]e[a(m(a)h(t)f]j]k]s)x</td>
</tr>
</tbody>
</table>

*Figure 1. Candidate strings*

<table>
<thead>
<tr>
<th>Candidate</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(((()[[[[]]]))) is balanced.</td>
</tr>
<tr>
<td>2</td>
<td>(((()[[[]])]) is not balanced.</td>
</tr>
<tr>
<td>3</td>
<td>(o(t[b]m[s(t(f)s(t)d)f]s)us is balanced.</td>
</tr>
<tr>
<td>4</td>
<td>(l(l[e]e[a(m(a)h(t)f]j]k]s)x is not balanced.</td>
</tr>
</tbody>
</table>

*Figure 2. Project 2 output for the candidates given in figure 1.*
2. **class Stack Interface**
Class Stack is shown in figure 3.

```cpp
struct StackException {
    StackException(char* m)
    {
        cout << endl << "I am the Stack and I am " << m << "." << endl;
    }
};
class Stack {
    int size;                      //Number of available elements
    int tos;                       //Index of the element on top of the stack
    char* S;                       //Points to storage for the stack
public:
    Stack(int sz=100);            //Constructor
    ~Stack();                      //Destructor
    bool IsFull(void);            //Is the Stack full?
    bool IsEmpty(void);           //Is the Stack empty?
    void Push(char v);            //Put v on top of the stack
    char Pop(void);               //Return the character on top of the stack
};
```

**Figure 3. class Stack.**

**Member functions:**
1. `Stack(int sz)` Member function `Stack` is the constructor. The construction is called when an object of type `Stack` is declared.
2. `~Stack()` Member function `~Stack` is the destructor. The destruction is called when control for an object of type `Stack` goes out of the scope where the object was declared.
3. `bool IsFull(void)` Member function `IsFull` determines if the stack is full.
4. `bool IsEmpty(void)` Member function `IsEmpty` determines if the stack is empty
5. `void Push(char v)` Member function `Push` places the value of parameter `v` on top of the stack.
6. `char Pop(void)` Member function `Pop` returns the value on top of the stack and makes the value that was next to the top of the stack the top of stack.

**Example:** A sample program that employs class Stack is shown in figure 4 and the output produced by the program is shown in figure 5.
3. Algorithm Description

We discuss two algorithms. First we discuss the algorithm that determines if a candidate string contains balanced brackets in section 3.1. Next, we discuss how the stack is implemented in section 3.2.

3.1. Algorithm that determines if a candidate contains balanced brackets

1. Start with the first character in the candidate string.
2. If the character is an opening or left bracket consisting of one of the three characters \( ( \), [ or \{, then push it on the stack.
3. If the character is a closing or right bracket consisting of one of the three characters \( ) \), ] or \}, then pop a character \( b \) from the stack. If \( b \) is not the opening or left bracket matching the current character a closing or right bracket then the candidate is unbalanced.
4. Go on to the next character.
5. If the stack is empty after processing all characters in the candidate then the candidate is balanced otherwise the candidate is unbalanced.

Refer to the diagram in figure 6 depicting a candidate containing balanced brackets.

```
#include <iostream>
using namespace std;
#include "Stack02.h"
int main()
{
  char V[]="aeiou";
  Stack S(10);
  for (int a=0;a<5;a++) S.Push(F[a]);
  while (!S.IsEmpty()) cout << setw(5) << S.Pop();
  cout << endl;
  return 0;
}
```

Figure 6. Balanced bracket candidate

The candidate is shown in the upper part of the diagram and snapshots of the stack are shown in the lower part of figure 6. Initially, the stack is empty. A curly brace, \( \{ \), is put on the stack as the character in index position 1 is processed. The stack remains unchanged...
until encountering a left parenthesis in index position 3 that is pushed on the stack. The very next character in position 4, a right parenthesis, ), matches the corresponding bracket on top of the stack that is immediately popped. A right curly brace is found in position 7 that matches the left curly brace put on in position 1. Left and right square braces are found and matched in positions 8 and 9. At the conclusion of the string the stack is empty and the candidate is determined to have balanced brackets.

Refer to the diagram in figure 7 containing a set of unbalanced brackets.

```
( ( [ { a a } ] ) b )
```

![Figure 7. Unbalanced bracket candidate.](image)

Two left parentheses, a left square bracket, and a left curly brace are found in the first four characters of the candidate. A right curly brace found in position 6 matches the left curly brace on top of the stack that is then popped from the stack. The right parenthesis in position 7 does not, however, match the left square brace on top of the stack. No more characters are processed because the candidate is unbalanced.

### 3.2. Implementation of class Stack.

The stack employed in project 2 is implemented by dynamically allocating an array of characters as shown in figure 8.

```
Stack
size  S  tos
int  char*  int
sz   -1
```

![Figure 8. Structure of class Stack](image)

To manage the stack, three variables are required.
1. **size**: Variable `size` retains the maximum number of elements available for the stack.
2. **tos**: Variable `tos` records the index of the element on top of the stack.
3. **S**: Variable `S` points to an array allocated to store elements on the stack.

Implementations of member functions are discussed below. Refer to the actual code in section 5 of this report.
**Stack(int sz)**  Member function *Stack* is the constructor. The construction is called when an object of type *Stack* is declared.

  Initialize member *size* to the value stored in parameter *sz*.
  Initialize member *tos* to −1. Negative one is chosen as a value that when incremented will yield zero, the index of the first available element in array *S*.
  Allocate storage referenced by member *S*.

**~Stack()**  Member function ~*Stack* is the destructor. The destruction is called when control for an object of type *Stack* goes out of the scope where the object was declared.

  Reclaim storage for the stack if, indeed, it was allocated.

**bool IsFull(void)**  Member function *IsFull* determines if the stack is full. Element index values *i* are: \(0 \leq i \leq *size* − 1\). The stack is full when *tos* \(\geq *size* − 1\).

  Return true when member *tos* \(\geq *size* − 1\); otherwise return false.

**bool IsEmpty(void)**  Member function *IsEmpty* determines if the stack is empty. Element index values *i* are: \(0 \leq i \leq *size* − 1\). The stack is empty when *tos* < 0.

  Return true when member *tos* < 0; otherwise return false.

**void Push(char v)**  Member function *Push* places the value of parameter *v* on top of the stack.

  Throw exception *StackFullException* if the stack is full.
  At least one element is available to store the value of parameter *v*; otherwise control would have passed to the calling function.
  Increment member *tos* to the index of the next available element.
  Store the value of parameter *v* in array *S* using *tos* as the index of the next available element.

**char Pop(void)**  Member function *Pop* returns the value on top of the stack.

  Throw exception *StackEmptyException* if the stack is empty.
  At least one element can be removed from the stack.
  Obtain a copy of the element on top of the stack.
  Decrement member *tos* to the index of the previous element stored on the stack.
  Return the copy of the element on top of the stack.
4. Test Description

The balanced and unbalanced candidates were employed to test project 2. Figures 10 and 11 show test input values and corresponding test output values.

```
({()}  
{((()))})  
{(()((())))})  
(o[t(b)m[s{t(f)s(t)d)f}s)p]us  
(ll[e][a(m(a)h(t)f]]j]k]s]x  
[]  
[])
```

Figure 10. Test input for project 2.
Figure 11. Test output for project p02
5. Source Code

//File p02.cpp exercises class Stack by determining which of the input strings
//are balanced

//Author: Ms. Petunia Perfect
//E-Mail: pperfect@uco.edu
//Student ID: *00000000
//Course: CS 2613, Programming II
//CRN: 10435, Autumn, 2007
//Project: p02
//Due: September, 2007
//Account: tt000

#include <iostream>
#include <iomanip>
#include <string>
#include <fstream>
using namespace std;

//Application include files
#include "Stack02.h"

bool IsBalanced(const string& c) {
    Stack S;
    for (int i=0;i<c.length();i++) {
        if (c[i]==0) break;
        switch (c[i]) {
            case '(':
            case '[':
            case '{':
                S.Push(c[i]);
                break;
            case ')':
                if (S.Pop()!='(') return false;
                break;
            case ']':
                if (S.Pop()!='[') return false;
                break;
            case '}':
                if (S.Pop()!='{') return false;
                break;
        }
    }
    return true;
}
case ']}':
    if (S.Pop()!='{') return false;
    break;
  case '}':
    if (S.Pop()!='{') return false;
    break;
  default:
    break;
  }
  return true;
}

//Function BracketMgr scans the input stream i for candidate strings,
//determines if they are palindromes, and prints a message characterizing
//the candidates in the output stream o.
void BracketMgr(istream& i,ostream& o)
{
    for (;;) {
        string c;
        i >> c;
        if (i.eof()) break;
        o << " \n";
        cout << endl;
        o << c  << " is";  
        cout << c << " is";
        if (!IsBalanced(c)) { o << " not"; cout << " not";} 
        o << " balanced.";  
        cout << " balanced.";
    }
    o << endl;  
    cout << endl;
}

//FileException is thrown when too many arguments appear on the command
//line.
struct FileException {
    FileException(char* fn)
    {   cout << endl;
        cout << "File " << fn << "could not be opened."; 
        cout << endl;
    }
};

//CommandLineException is thrown when too many arguments appear
//on the command line.
struct CommandLineException {
    CommandLineException(int m, int a)
    { cout << endl;
        cout << "Too many arguments on the command line.";
        cout << endl;
    }
};

// Function main processes command line arguments

int main(int argc, char* argv[]) {
    try {
        char ifn[255], ofn[255];
        switch (argc) {
            case 1:
                cout << "Enter the input file name. ";
                cin >> ifn;
                cout << "Enter the output file name. ";
                cin >> ofn;
                break;
            case 2:
                strcpy(ifn, argv[1]);
                cout << "Enter the output file name. ";
                cin >> ofn;
                break;
            case 3:
                strcpy(ifn, argv[1]);
                strcpy(ofn, argv[2]);
                break;
            default:
                throw CommandLineException(2, argc-1);
                break;
        }
        ifstream i(ifn); if (!i) throw FileException(ifn);
        ofstream o(ofn); if (!o) throw FileException(ofn);
        BracketMgr(i, o);
        o.close();
        i.close();
    } catch (...) {
        cout << endl;
        cout << "Program terminated!";
        cout << endl;
        exit(EXIT_FAILURE);
    }
    return 0;
}
#ifndef Stack02_h
#define Stack02_h 1
//------------------------------------------------------------------------
------
//File Stack02.h defines class Stack. class Stack is implemented using a
//dynamically allocated array of characters.
//------------------------------------------------------------------------
------
//Author:       Ms. Petunia Perfect
//E-Mail:       pperfect@uco.edu
//Student ID:   *00000000
//Course:       CS 2613, Programming II
//CRN:          10847, Autumn, 2004
//Project:      p02
//Due:          September 17, 2004
//------------------------------------------------------------------------
------
//Copyright September, 2004 by Ms. Petunia Perfect.
//Do not reproduce without permission from Ms. Petunia Perfect.
//------------------------------------------------------------------------
------
//class Stack contains the attributes and member functions of a stack
//implemented using a dynamically allocated array
//------------------------------------------------------------------------
------
class Stack {
    int size;                   //Number of available elements
    int tos;                    //Top Of Stack - index of the element on
                              //the stack
    char* S;                    //Points to storage for a stack of
                              //characters.
public:
    struct StackFullException{
        StackFullException()
        {   cout << endl << "Stack Overflow" << endl;
        }
    };
    struct StackEmptyException{
        StackEmptyException()
        {   cout << endl << "Stack Underflow" << endl;
        }
    };
    Stack(int sz=100);          //Constructor
    -Stack();                   //Destructor
    void Push(char v);          //Push character v on the stack
    bool IsFull(void);          //Is the stack full?
    bool IsEmpty(void);         //Is the stack empty?
    char Pop(void);             //Pop a character from the stack
};
#endif
//File Stack02.cpp implements class Stack

//Author: Ms. Petunia Perfect
//E-Mail: pperfect@uco.edu
//Student ID: *00000000
//Course: CS 2613, Programming II
//CRN: 10847, Autumn, 2004
//Project: p02
//Due: September 17, 2004
//Account: tt000

#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

#include "Stack02.h"

Stack::Stack(int sz):size(sz),tos(-1){S=new char[size];}

Stack::~Stack(){if (S) delete[] S;}

void Stack::Push(char v)
{   if (IsFull()) throw StackFullException();
   S[++tos]=v;
}
//******************************************************************************
//--Member function Pop
//******************************************************************************
char Stack::Pop(void)
{   if (IsEmpty()) throw StackEmptyException();
   return S[tos--];
}
//******************************************************************************
//--Member function IsFull
//******************************************************************************
bool Stack::IsFull(void){return tos>=size-1;}
//******************************************************************************
//--Member function IsEmpty
//******************************************************************************
bool Stack::IsEmpty(void){return tos<0;}
File p02make

# File p02make contains instructions that create executable file
# p02. Program p02 determines if the strings in an input file
# are palindromes.

# Author: Ms. Petunia Perfect
# Student ID: *00000000
# E-Mail: pperfect@uco.edu
# Course: CS 2613, Programming II
# CRN: 10847, Autumn, 2004
# Project: p02
# Due: September 17, 2004

# Object files (no colon)

obj = p02.o Stack02.o

# Bind (Link) the object files and create executable p02

p02: ${obj}
    g++ -o p02 ${obj} -lm

# Compile p02.cpp that exercises class Stack.

p02.o:   p02.cpp Stack02.h
    g++ -g -c p02.cpp

# Compile Stack02.cpp that implements class Stack.

Stack02.o: Stack02.cpp Stack02.h
    g++ -g -c Stack02.cpp