Some applications of a stack

Agenda
1. Determining whether a given string of parentheses is balanced
2. Determining whether there is a match of delimiters in a given expression
3. Determining whether a given string of characters containing exactly one $ is of the form w$w$
4. Determining whether a given string of a's and b's has an equal number of a's and b's.

Determining whether a string of parentheses is balanced

Among numerous applications of a stack, one that is particularly interesting is its use in determining whether a given string of parentheses is balanced or not. (Every compiler includes this test during syntax analysis.)

Definition: A string $w$ consisting of ('s and )'s constitutes a string of balanced parentheses if and only if the following conditions hold:

1. The number of ('s is equal to the number of )'s, and
2. The number of ('s in any prefix of $w$ is greater than or equal to the number of )'s.

Note that the empty string is (trivially) a string of balanced parentheses. Here are some non-trivial examples.

( ( ) ) ( ) ( ( ( ) ) )
( ( ( ( ) ) ) ) ( ( ) ) ( )

Following are examples of strings of parentheses that are not balanced.

( ( ) ) ( ( ) ) Left parentheses outnumber the right parentheses.
( ) ( ) ( ( ) ) Right parentheses outnumber the left parentheses in a prefix of the string.

Because of the high importance of a stack as a data structure, the standard template library of C++ includes its implementation. (Library header: <stack> which places the definition in the std namespace.) Important member functions are as follows.
Member function

<table>
<thead>
<tr>
<th>Member function</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.size()</td>
<td>Returns the number of elements in s. (function type: int)</td>
</tr>
<tr>
<td>s.empty()</td>
<td>Returns true if s is empty, false otherwise. (function type: bool)</td>
</tr>
<tr>
<td>s.top()</td>
<td>Returns the top element of s. (function type: elementType)</td>
</tr>
<tr>
<td>s.push(el)</td>
<td>Inserts a copy of el at the top of s. (function type: void)</td>
</tr>
<tr>
<td>s.pop()</td>
<td>Removes the top element of s. (function type: void)</td>
</tr>
</tbody>
</table>

The accompanying program makes use of this facility. Here is a sample dialog.

Please enter a string of parentheses: ((((()(())))))()()
This is a string of balanced parentheses.

Want to examine another string? (y/n): y
Please enter a string of parentheses: (())()((()(
This is not a string of balanced parentheses, since left parentheses outnumber the right parentheses.

Want to examine another string? (y/n): y
Please enter a string of parentheses: (())()
This is not a string of balanced parentheses, since right parentheses outnumber the left parentheses in a prefix of the string.

Want to examine another string? (y/n): n

Bye!

Note: User inputs in the preceding dialog are in blue.

// CSCI 301: (Instructor: Pranava K. Jha)
// Program to determine whether a given string of parentheses is balanced or not.
#include <iostream>
#include <stack> // stack template exists in the system library.
using namespace std;
int main()
{
    char ch;  // Variable used to hold an input character.
    char ans; // Variable used in a dialog
    bool good; // Boolean flag
    do // Beginning of the outer loop.
    {
        stack<char> s; // Stack s is necessarily empty at the beginning of each iteration.
        good = true; // flag initialized to true.
        cout<< "Please enter a string of parentheses: ";
        cin.get(ch);
//Only '(' or ')' may appear in the input stream.

while(ch != ' \n') //Here is the heart of the program.
{
    if(ch != '(' && ch != ')')//Disallow other characters
    {
        cout << "Only parentheses are admissible." << endl << "Making an exit." << endl;
        exit(1);
    }
    if (ch == '(')
        s.push(ch);
        //Only left parens may appear on the stack.
    else //If control reaches here, ch is equal to ')'.
    {
        if(!s.empty())
            s.pop();
        else //If control reaches here, stack is empty.
        {
            good = false;
            //Reject the string, since )'s outnumber
            //('s in a prefix of the string.
            break;
        }
    }
    cin.get(ch); // Read the next character.
} // end of while

if(s.empty() && good)
    cout << "This is a string of balanced parentheses.";
else
{
    cout << "This string is not balanced, since";
    if(!s.empty())
        cout << endl << "left (‘s outnumber )’s."
    else // (!good)
    {
        cout <<")’s outnumber (‘s " << endl
        << " in a prefix of the string.";
        cin.ignore(1000, ' \n');
    }
}

    cout << endl << "Want to examine another string? (y/n): ";
    cin >> ans;
    cin.ignore(1000, ' \n'); //Ignore the newline character.
    while(ans != 'n' && ans != 'N' && ans != 'y' && ans != 'Y')
    {
        //Force the user to input n or N or y or Y.
        cout << "Please enter n or N or y or Y: ";
        cin >> ans;
        cin.ignore(1000, ' \n');
    } //end of while
} while (ans == 'y' || ans == 'Y'); // end of do-while.
    cout << endl << "Bye!" << endl << endl;
return 0;
} // main
Determining if there is a match of delimiters in a given expression

Here is a generalization of the preceding problem of determining whether or not a string of parentheses is balanced. The delimiters are: ‘(’, ‘{’, ‘[’, ‘)’, ‘}’ and ‘]’.

A sample dialog follows.

Please enter an expression: [(a+b)/(c*d){}{]} There is a nice match of delimiters in the given expression.

Want to examine another string? (y/n): y
Please enter a string of parentheses: [[]] The delimiters don’t match in the given expression.

Want to examine another string? (y/n): n

Bye!

Observe that an input expression in this case may as well contain characters other than the delimiters themselves. All such characters are ignored.

Here is the program.

// Instructor: Pranava K. Jha
// Determining if there is a match of delimiters in a given expression.
#include <iostream>
#include <stack> //stack template exists in the system library.
using namespace std;

bool match(char left, char right);
// Function to check if a match exists between a left delimiter and // the corresponding right delimiter.

int main()
{
    char ch; //Variable used to hold an input character.
    char ans; //Variable used in a dialog
    bool good; //Flag to indicate success or failure
    do //Beginning of the outer loop.
    {
        stack<char> s;
        //The stack s is empty at the beginning of each iteration
        good = true; //flag initialized to true.
        cout<< "Please enter an expression: ";
        cin.get(ch);
        // ‘(’, ‘{’, ‘[’, ‘)’, ‘}’ and ‘]’ are the delimiters.
        while(ch != '\n') //This is the main loop.
        {
            if (ch == '(' || ch == '{' || ch == '[')
                s.push(ch); //push left delimiters on to the stack.
            else if (ch == ')' || ch == '}' || ch == ']')
                {
if(s.empty())
    good = false;
else // s is not empty
{
    if (!match(s.top(), ch))
        good = false;
    else
        s.pop();
}
//else consume the input character
if (!good)
    break;
else
    cin.get(ch); // Read the next character
}// end of while

if(s.empty() && good)
    cout << "There is a nice match of delimiters."
else
    cout << "The delimiters don't match."
cout << endl << endl;
cout << "Want to examine another string? (y/n): ";
cin >> ans;
cin.ignore(1000, '\n'); //Ignore the newline character.
while(ans != 'n' && ans != 'N' && ans != 'y' && ans != 'Y')
{
    //Force the user to input n or N or y or Y.
    cout << "Please enter n or N or y or Y: ";
cin >> ans;
cin.ignore(1000, '\n');
    //Ignore the newline character.
}//end of while
} while (ans == 'y' || ans == 'Y'); // end of outer loop.
cout << endl << "Bye!" << endl << endl;
return 0;

bool match(char left, char right)
{
    bool temp = true;
    switch(left)
    {
    case '(': if (right != ')') // if parentheses don't match
        temp = false;
        break;
    case '{': if (right != '}') // if braces don't match
        temp = false;
        break;
    case '[': if (right != ']') // if brackets don't match
        temp = false;
    }//end of switch
    return temp;
}//end of match
Determining whether a given string of characters containing exactly one $ is of the form $wR$

Consider the following set:

$L = \{ wR: w \text{ is a string of characters other than }$, and $wR = \text{reverse}(w) \}.$

For example, the strings $a$, $aBBc$ and $cB$ are in $L$, but AB$AB$ and ABC$BC$ are not. Objective is to write a program that determines membership in the given set.

Here is a sample dialog.

Please enter a string of characters containing exactly one $: ab$ba
This string is in the given set.

Want to examine another string? (y/n): y
Please enter a string of characters containing exactly one $: ab$bac
This string is not in the given set, since its right half is longer than the left half.

Want to examine another string? (y/n): y
Please enter a string of characters containing exactly one $: abb$ba
This string is not in the given set, since its left half is longer than the right half.

Want to examine another string? (y/n): y
Please enter a string of characters containing exactly one $: aBb$Bc
This string is not in the given set, since its right half is not a mirror image of the left half.

Please enter a string of characters containing exactly one $: $
This string is in the given set.
Want to examine another string? (y/n): n
Bye!

```cpp
#include <iostream>
#include <stack>

using namespace std;

int main()
{
    char ch; //Variable used to hold an input character.
    char ans; //Variable used in a dialog
    bool good; //A Boolean flag
    do //Beginning of the do-while loop.
    {
        stack<char> s;
        // s is empty at the beginning of each iteration.
        good = true;

        cout << "Please enter a string of characters" << " containing exactly one $: ";
        cin.get(ch);
```
while(ch != '$')
{
    s.push(ch);
    cin.get(ch);
}
cin.get(ch);
while(ch != '\n')
{
    if(s.empty())
    {
        good = false;
        break;
    }
    else
    {
        if(s.top() != ch)
        {
            good = false;
            break;
        }
        else
        {
            s.pop();
            cin.get(ch);
        }
    }
}
if(s.empty() && good)
    cout << "This string is in the given set."
else
{
    cout << "This string is not in the given set, "
    << "since its" << endl;
    if(s.empty() && !good)
        cout << "right half is longer than left half.";
    if(!s.empty() && good)
        cout << "left half is longer than right half.";
    if(!s.empty() && !good)
        cout << "right half is not a mirror image of "
        << "left half.";
}
cout << endl << endl;
cout << "Want to examine another string? (y/n): "; //
cin >> ans;
cin.ignore(100, '\n'); //Ignore the newline character.
while(ans != 'n' && ans != 'N' && ans != 'y' && ans != 'Y')
{
    //Force the user to input n or N or y or Y.
    cout << "Please enter n or N or y or Y: ";
    cin >> ans;
    cin.ignore(100, '\n'); //Ignore the newline character.
} //end of while
while (ans == 'y' | | ans == 'Y');// end of do-while.
cout << endl << "Bye!" << endl << endl;
return 0;
} // main
Determining whether a given string of a's and b's has an equal number of a's and b's

Method of attack:

1. While end of the input string has not been reached, perform steps (i) – (iv):
   (i) If the present input character is ‘a’ and the stack is empty or top-of-the-stack symbol is ‘a’, then consume the input character and push ‘a’ on to the stack.
   (ii) If the present input character is ‘b’ and top-of-the-stack symbol is ‘a’, then consume the input character and pop ‘a’ off the stack.
   (iii) If the present input character is ‘b’ and the stack is empty or top-of-the-stack symbol is ‘b’, then consume the input character and push ‘b’ on to the stack.
   (iv) If the present input character is ‘a’ and top-of-the-stack symbol is ‘b’, then consume the input character and pop ‘b’ off the stack.

2. If end of the input string has been reached and the stack is empty, then report success, otherwise report failure.

Because of the symmetry between steps (i) – (ii) and steps (iii) – (iv), the foregoing scheme admits a compact description as follows:

1. While end of the input string has not been reached, perform steps (i) – (ii):
   (i) If the stack is empty or top-of-the-stack symbol is same as the present input character, consume the input character and push its copy on to the stack.
   (ii) If the stack is non-empty and top-of-the-stack symbol is different from the present input character, consume the input character and pop off the stack.

2. If end of the input string has been reached and the stack is empty, then report success, otherwise report failure.

A program follows.
```cpp
#include <iostream>    // Determining whether a's and b's are equinumerous
#include <stack>      // stack template exists in the system library.
using namespace std;

int main()
{
    char ch;  // Variable used to hold an input character.
    char ans; // Variable used in a dialog:
    bool good;  // Flag

do // Outer loop.
{
    stack<char> s;  // Declaring the stack variable s here ensures that s is
                    // necessarily empty at the beginning of each iteration.
    good = true;
    cout << "Please enter a string of a's and b's: ";
    cin.get(ch);    // Only 'a' or 'b' may appear in the stream.
    while(ch != '\n') // Here is the heart of the program.
    {
        if(!(ch == 'a' || ch == 'b'))
        {
            cout << "Only a's and b's are allowed."
                << endl << "Making an exit." << endl;
            exit(1);
        }
        if(s.empty() || s.top() == ch)
            s.push(ch);
        else
            s.pop();
        cin.get(ch);
    }  // end of while
    if(s.empty())
        cout << "Accept: a's as b's are equinumerous."
    else
    {
        cout << "Reject: ";
        if(s.top() == 'a')
            cout << "a's outnumber b's in this string."
        else
            cout << "b's outnumber a's in this string."
    }
    cout << endl << endl;
    cout << "Want to examine another string? (y/n): ";   // cin >> ans;
    cin.ignore(100, ' \n');    // Ignore the newline character.
    while(ans != 'n' && ans != 'N' && ans != 'y' && ans != 'Y')
    {
        cout << "Please enter n or N or y or Y."
            << endl << "Making an exit." << endl;
        cin >> ans;
        cin.ignore(100, ' \n');    // Ignore the newline character.
    }  // end of while
}  // end of do-while
    cout << endl << "Bye!" << endl;
    return 0;
}  // main
```