GeeksMan
Data Structure
Lesson 2
Introduction to STL

Trilok Kaushik

Founder of GeeksMan

Chirag Soni

Nupur Pahuja

Jessica Mishra

Team Coordinators

Balanced Brackets

In this section , we will see how stack can be used to solve an interesting programming problem quite easily called Balanced Brackets.

Description of the Problem:-

You are given a string containing a combination made up of 3 types of brackets { , } , [,] , (and).Now given a string made from the combination of these brackets ,we have to tell whether the string is balanced or not.

Example of Balanced String:- { ([ ]) }

Example of Unbalanced String:- { ( ]) }

Logic :-

We will create a stack and will start parsing strings from the left hand side. If we found any opening bracket, we will push that bracket into the stack. But if we found a closing bracket, we will look at the topmost bracket in the stack. If the topmost bracket matches the current bracket, we will pop that bracket from the stack and will move to the next bracket in the string (if any).But if the top bracket does not match the current bracket or if the stack is empty, then it is an **UNBALANCED String**. After the whole string is parsed, we look at the stack one final time, if the stack is empty, then we will finally print out that the given string is **BALANCED** but if there are some brackets left in the stack, then we print out that it is an unbalanced string.

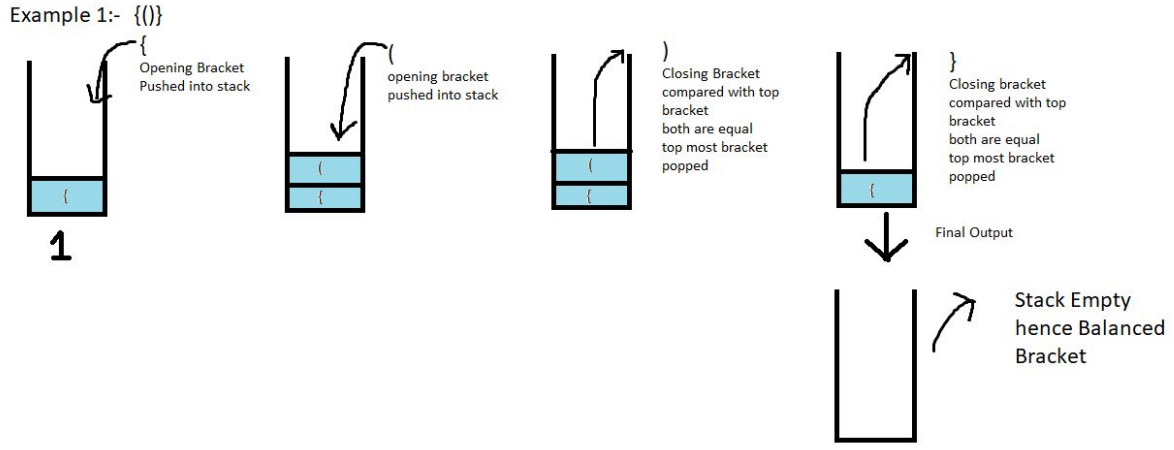

Fig. 1.2(c)

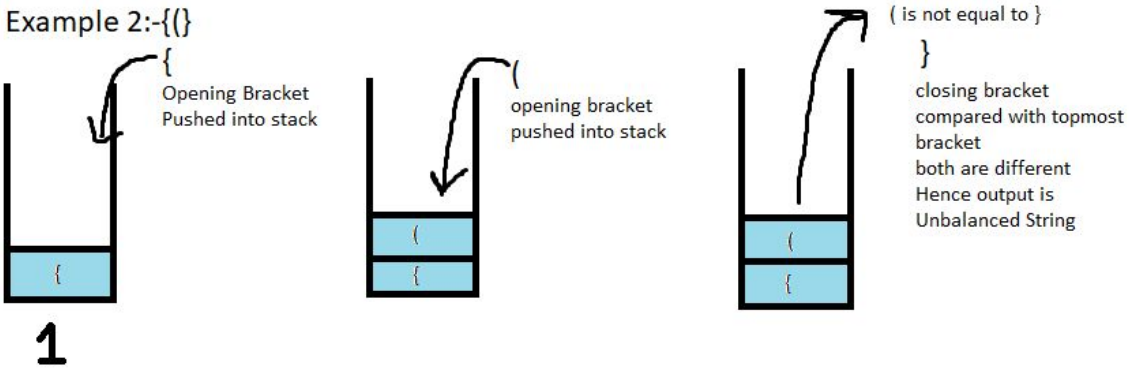

Fig. 1.2(d)

Now you are ready to write the program for this , Consider the following link for any help.

```
//Balanced Bracket problem...

#include <bits/stdc++.h>
#include <iostream>
using namespace std;

int main()
```

```
{
    string s;
    cin>>s;
    stack<char> st;
    int i,flag=0;
    for(i=0;i<s.length();i++)
    {
        if(s[i]=='('|| s[i]=='{' || s[i]=='[')
            st.push(s[i]);
        else
        {
            if(!st.empty())
            {
                if(s[i]==')'&&st.top()=='(' ||
s[i]=='}'&&st.top()=='{'|| s[i]==']'&&st.top()=='[')
                 st.pop();
             else
             {
                 flag=1;
                 break;
             }
            }
            else
            {
                flag=1;
                break;
            }
        }
    }
    if(st.empty()&&flag==0)
```

```
        cout<<"BALANCED BBRACKET";
    else
        cout<<"unbalanced bracket";
    return 0;

}
```

Balanced bracket:- https://sapphireengine.com/@/dpodt0

INFIX , POSTFIX AND PREFIX CONVERSIONS

In this section we will implement the concept of stack for converting infix expression to postfix expression.
In c++ we already have stack containers in STL(standard template library)and its function are described below:-

- empty() → Returns whether the stack is empty
- size() → Returns the size of the stack
- top() → Returns a reference to the top most element of the stack
- push(g) → Adds the element 'g' at the top of the stack
- pop() → Deletes the top most element of the stack

ques=How to create a stack ???
ans= stack<char> st;
Using this we have created a stack st which can store character element in LIFO order.

Infix expression is an expression in which the operator is in the middle of operands, like **operand operator operand.**

Infix expression = x+y

Postfix expression is an expression in which the operator is after operands, like **operand operand operator.**

Postfix expression= xy+

One important question arises here is why do we need postfix expressions?

This is because the expressions written in postfix form are easily computed by the system as compared to infix notation as parenthesis are not required in postfix. Generally reading and editing by the user is done on infix notations as they are parentheses separated hence easily understandable for humans.

Infix to Postfix Conversion Algorithm:-

1. Print operands as they arrive.

2. If the stack is empty or contains a left parenthesis on top, push the incoming operator onto the stack.

3. If the incoming symbol is a left parenthesis, push it on the stack.

4. If the incoming symbol is a right parenthesis, pop the stack and print the operators until you see a left parenthesis. Discard the pair of parentheses.

5. If the incoming symbol has higher precedence than the top of the stack, push it on the stack.

6. If the incoming symbol has equal precedence with the top of the stack, use association. If the association is left to right, pop and print the top of the stack and then push the incoming operator. If the association is right to left, push the incoming operator.

7. If the incoming symbol has lower precedence than the symbol on the top of the stack, pop the stack and print the top operator. Then test the incoming operator against the new top of stack.

8. At the end of the expression, pop and print all operators on the stack

a - (*b* + *c* * *d*)/*e* to

ch	stack (bottom to top)	postfixExp
a		a
–	–	a
(	–(	a
b	–(	ab
+	–(+	ab
c	–(+	abc
*	–(+*	abc
d	–(+*	abcd
)	–(+	abcd*
	–(	abcd*+
	–	abcd*+
/	–/	abcd*+
e	–/	abcd*+e
		abcd*+e/–

```
//.....................infix to postfix
conversion.................

#include <bits/stdc++.h>
#include <iostream>
using namespace std;
int prec(char c)
{
   if(c == '^')
            return 3;
     else if(c == '*' || c == '/')
            return 2;
   else if(c == '+' || c == '-')
            return 1;
   else
            return -1;
}

void inf_to_post(string s)
{
   string finl;
   int i;
   stack<char> st;
```

```
    for(i=0;i<s.length();i++)
    {
        if((s[i] >= 'a' && s[i] <= 'z')||(s[i] >= 'A' && s[i]
<= 'Z'))
            finl+=s[i];
        else
        {
            if(s[i]=='(')
                st.push(s[i]);
            else if(s[i]==')')
            {
                while(st.top()!='(')
                {
                    finl+=st.top();
                    st.pop();
                }
                st.pop();
            }
            else
            {
                if(st.empty())
                    st.push(s[i]);
                else if(prec(st.top()<s[i]))
                 st.push(s[i]);
                else
                {
                    while(prec(st.top())>=s[i])
                    {
                       finl+=st.top();
                      st.pop();
                    }
                    st.push(s[i]);
                }
            }

        }
    }
    while(!st.empty())
    {
```

```
            finl+=st.top();
                        st.pop();
    }

    cout<<finl;
}



int main()
{
      string s;
    cin>>s;
    inf_to_post(s);
    return 0;
}
```

Link for infix to post code= https://sapphireengine.com/@/blcxul

Question for practice=
1.immediate-smaller-element :
https://practice.geeksforgeeks.org/problems/immediate-smaller-element/0
2.remove-repeated-digits-in-a-given-number :
https://practice.geeksforgeeks.org/problems/remove-repeated-digits-in-a-given-number/0
3.pairwise-consecutive-elements :
https://practice.geeksforgeeks.org/problems/pairwise-consecutive-elements/1
4.reverse-a-string-using-stack :
https://practice.geeksforgeeks.org/problems/reverse-a-string-using-stack/1
5.delete-middle-element-of-a-stack :
https://practice.geeksforgeeks.org/problems/delete-middle-element-of-a-stack/1