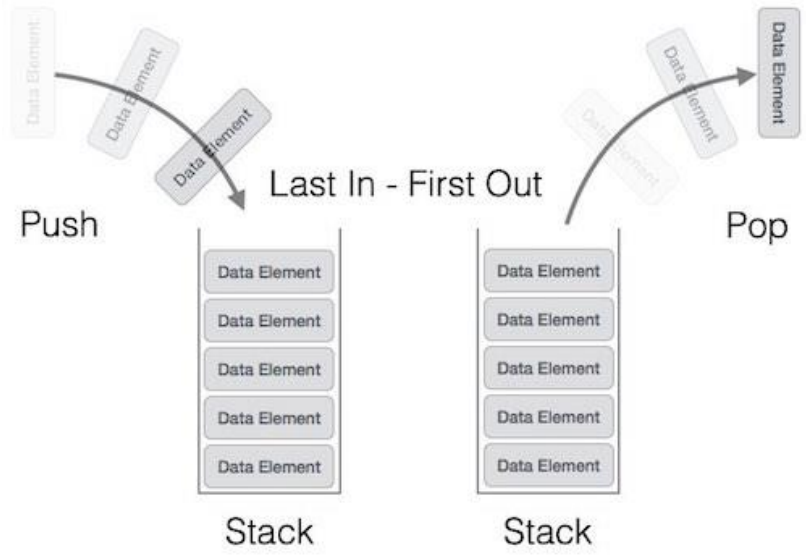
Assignment no	9
Aim	In any language program mostly syntax error occurs due to unbalancing delimiter such as (),{},[]. Write C++ program using stack to check whether given expression is well parenthesized or not
Objective	To understand the concept of Stack data structure To implement Stack data structure and its operations for given application.
Outcome	After executing this assignment, Students will be able to identify and use Stack data structure for given application. Students design and implement stack data structure and its operations
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit) Eclipse

Theory related to assignment:

A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

The following diagram depicts a stack and its operations −



A stack can be implemented by means of Array, Structure, Pointer, and Linked List.

Basic Operations

Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations −

push() − Pushing (storing) an element on the stack.
pop() − Removing (accessing) an element from the stack.

When data is PUSHed onto stack.

To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks −

peek() − get the top data element of the stack, without removing it.
isFull() − check if stack is full.
isEmpty() − check if stack is empty.

Application of Stack:
Expression Evolution
Expression Conversion Infix to Postfix, Infix to Prefix, Postfix to Infix Prefix to Infix Parsing.

Simulation of recursion Function call Algorithm:
1) Declare a character stack S.
2) Now traverse the expression string exp.
a) If the current character is a starting bracket (‘(‘or ‘{‘or ‘[‘) then push it to stack.
b) If the current character is a closing bracket (‘)’ or ‘}’ or ‘]’) then pop from stack
and if the popped character is the matching starting bracket, then fine else parenthesis is not balanced.
3) After complete traversal, if there is some starting bracket left in stack then “not balanced”

Test Cases
1. (A+(B*C) Expression is invalid
2. (A+(B*C)) Expression is valid

Example:
Input:
Enter any expression having number of opening & closing brackets.
Output:
Entered expression is well parenthesized or not.
Algorithm:
Step 1: Declare a character stack S.
Step 2: Now traverse the expression string exp.
If the current character is a starting bracket (‘(‘ or ‘{‘ or ‘[‘)
then
push it to stack.
If the current character is a closing bracket (‘)’ or ‘}’ or ‘]’)
then
pop from stack and
if the popped character is the matching starting bracket
then
fine
else
parenthesis are not balanced

Step 3: After complete trave
rsal, if there is some starting bracket left in stack
then “not balanced”
Step 4: Exit

ADT:

```
class Stack
{
   char s[MAX];
   int top;
   public:
      Stack()
      {
         top=-1;
      }
      void push(char ch);
      char pop();
      bool isEmpty();
      bool isFull();
      bool checkParenthesis(char expr[]);
};
```

Pseudo Code:

1. isEmpty()

```
Algorithm isEmpty() {

   if(top==-1)

      return 1;

   else

      return 0;

}
```

2 isFull()

```
Algorithm isFull() {

   if(top==MAX-1)

      return 1;

   else

      return 0;

}
```

3.push

```
Algorithm push(char ch) {
   if(!isFull())     {
       top++;
       s[top]=ch;
   }
}
```

4. pop

```
 Algorithm pop() {
   if(!isEmpty())     {
       char ch=s[top];
       top--;
       return ch;
   }
   else
       return '\0';
}
```

5.checkParanthesis

```
Algorithm checkParenthesis(char expr[])
{
  // Traversing the Expression
   For i:=0 to length(expr)     {
       if (expr[i]=='('||expr[i]=='['||expr[i]=='{')
       {
           // Push the element in the stack
           push(expr[i]);
```

```
            continue;

        }

    // IF current current character is not opening  bracket, then it must be closing. So stack
//cannot be empty at this point.

        if (isEmpty())

          return false;

         switch (expr[i])        {

        case ')':

            // Store the top element in a

            x = pop();

            if (x=='{' || x=='[')

                return false;

            break;

    case '}':

            // Store the top element in b

            x = pop();

            if (x=='(' || x=='[')

                return false;

            break;

        case ']':

            // Store the top element in c

            x = pop();

              if (x =='(' || x == '{')

                return false;

            break;

        }

    }
```

```
    // Check Empty Stack

    return (isEmpty());

}
```

Conclusion:

Different stack operations are implemented successfully.

Review Questions:

1. Which data structure is required to check whether an expression contains balanced parenthesis?
2. What data structure would you mostly likely see in a non-recursive implementation of a recursive algorithm?
3. Process of removing an element from stack is called….
4. In a stack, if a user tries to remove an element from empty stack it is called---
5. Convert the following infix expressions into its equivalent postfix expressions.
   (A + B -D)/(E – F)+G
6. Consider the following operation performed on a stack of size
   Push(1);Pop();Push(2);Push(3)Pop();Push(4);Pop();Pop();Push(5);
7. If the elements “A”, “B”, “C” and “D” are placed in a stack and are deleted one at a time, what is the order of removal?
8. Convert the following Infix expression to Postfix form using a stack.
   x + y * z + (p * q + r) * s, Follow usual precedence rule and assume that the expression is legal.
9. Define a stack?
10. Stack data structure uses which principle?
11. Stack belongs to which type of data structure?
12. What do you mean by stack underflow?
13. What do you mean by stack overflow?
14. List out the basic operations of a stack? 7. How to implement stack?