

|  |
| --- |
| **Title:**  implementation of a data structure for a problem statement |

**Objective:**

.Question no 13:

Write a program to check whether the string has balanced parathesis. Consider different

parenthesis {, [, (, < ‘ in your program.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
|  | The implementation of efficient data structure for a problem |

**Algorithm:**

**step1:delcare stack s and top=-1**

**step2:input string s1 anf flag=0**

**step3:for loop i=0,i<length of string, increment i**

**step4:if s1[i] is an open paranthesis den push in stack**

**step5:if s1[i] is closing paranthesis and the pop value of stack is not the opening bracket for s1[i] then flag=1**

**step6:end of for loop**

**step7:if stack s is not empty by the end of step 6 then flag=1**

**step8:if flag==1 print invalid expression**

**step9::else print valid expression**

**Program:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

char s[1000];

int top=-1;

void push(char a)

{

s[++top]=a;

}

char pop()

{

return s[top--];

}

int main()

{

char s1[1000];

gets(s1);

int f=0;

int i;

for( i=0;i<strlen(s1);i++)

{

if(s1[i]=='(' || s1[i]=='{' ||s1[i]=='['|| s1[i]=='<')

{

push(s1[i]);

}else if(s1[i]==')' || s1[i]=='}'||s1[i]==']' ||s1[i]=='>')

{

if(top==-1)

{

f=1;

}else

{ char temp=pop();

if(s1[i]=='>' && (temp=='(' ||temp=='{'|| temp=='['))

{

f=1;

printf("1");

}else if(s1[i]==')' && (temp=='<' || temp=='{'|| temp=='['))

{

f=1;

printf("2");

}else if(s1[i]=='}' && (temp=='(' ||temp=='<'|| temp=='['))

{ f=1;

printf("3");

}else if(s1[i]==']' && (temp=='(' || temp=='{'|| temp=='<')){

f=1;

printf("4");

}

}

}

}

if(top>0)

{

f=1;

}

if(f==1)

{

printf("invalid expression\n");

}else

{

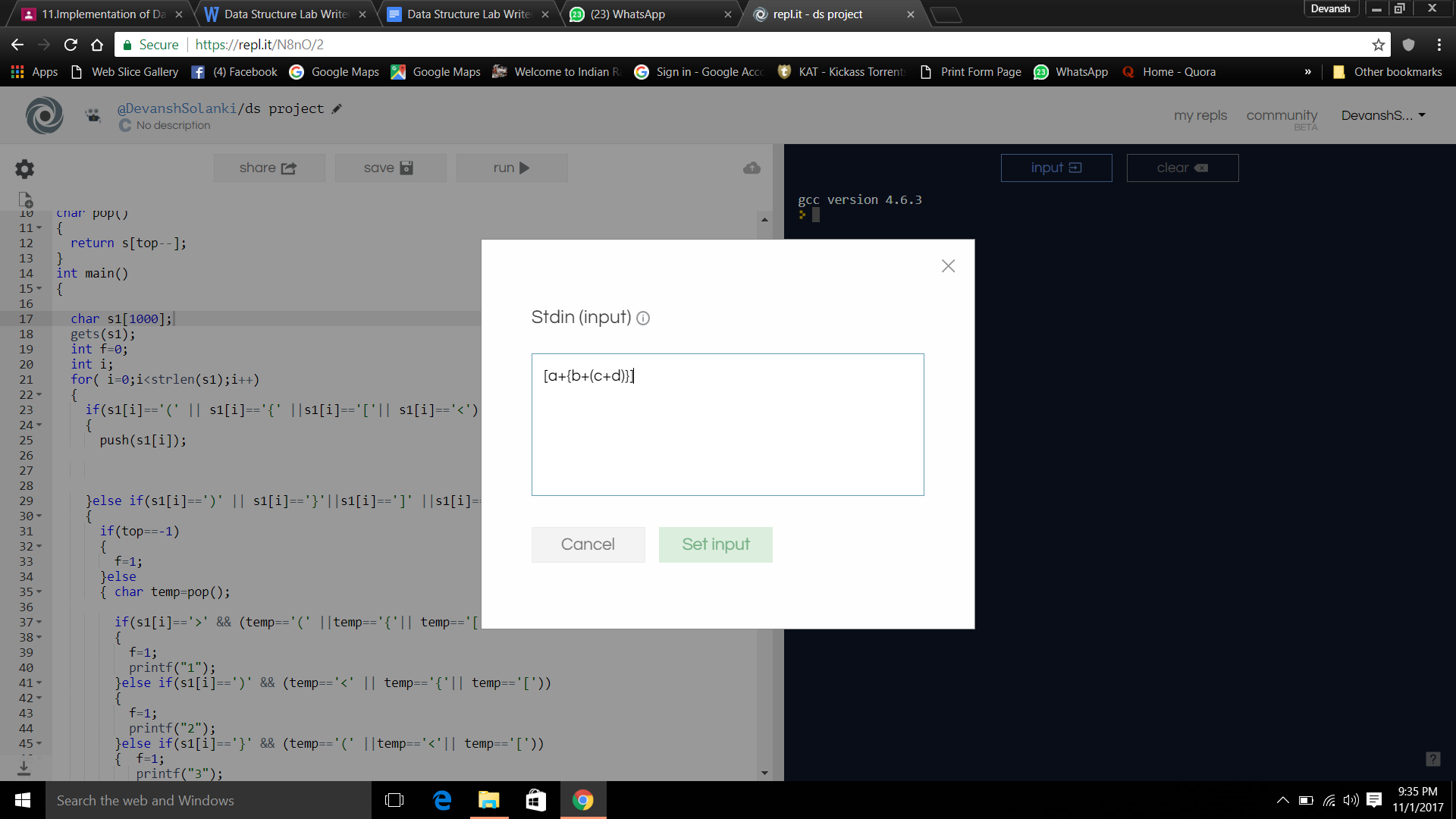
printf("valid expression\n");

}

return 0;

}

**Output**:





**Related Theory: -**

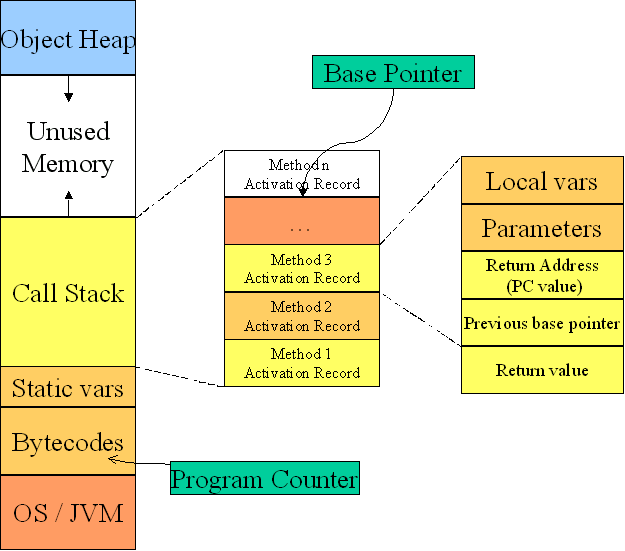
**use of stacks :**

**1)Memory Management**

**Any modern computer environment uses a stack as the primary memory management model for a running program. Whether it's native code (x86, Sun, VAX) or JVM, a stack is at the center of the run-time environment for Java, C++, Ada, FORTRAN, etc.**

**The discussion of JVM in the text is consistent with NT, Solaris, VMS, Unix runtime environments.**

**Each program that is running in a computer system has its own memory allocation containing the typical layout as shown below.**

****

2)**Expression evaluation**

**In particular we will consider arithmetic expressions. Understand that there are boolean and logical expressions that can be evaluated in the same way. Control structures can also be treated similarly in a compiler.**

**This study of arithmetic expression evaluation is an example of problem solving where you solve a simpler problem and then transform the actual problem to the simpler one.**

**Aside: *The NP-Complete problem*. There are a set of apparently intractable problems: finding the shortest route in a graph (Traveling Salesman Problem), bin packing, linear programming, etc. that are similar enough that if a polynomial solution is ever found (exponential solutions abound) for one of these problems, then the solution can be applied to all problems.**

**Conclusion:-The implementation of stack for parenthesis checking in a**