K. J. Somaiya College of Engineering, Mumbai
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: B2 Roll No.: 16010121110
Experiment / assignment / tutorial No.
Grade: AA / AB / BB / BC / CC / CD /DD
Signature of the Staff In-charge with date

Title: Implementation of Stack applications.

Objective: To implement applications of stack

Expected Outcome of Experiment:

CO	Outcome
1	Explain the different data structures used in problem solving

Books/ Journals/ Websites referred:

1. *Fundamentals Of Data Structures In C* – Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications* – Jean Paul Tremblay, Paul G. Sorenson
3. *Data Structures A Pseudo Approach with C* – Richard F. Gilberg & Behrouz A. Forouzan
4. https://www.cprogramming.com/tutorial/computersciencetheory/stack.html
5. https://www.geeksforgeeks.org/stack-data-structure-introduction-program/
6. https://www.thecrazyprogrammer.com/2013/12/c-program-for-array-representation-of-stack-push-pop-display.html

Assigned Stack application:

Parenthesis match using Dynamic queue.

Algorithm:

Struct NodeType{

ElementType Element;

NodeType Next;

}

1. Algorithm StackType CreateStack()

//This Algorithm creates and returns an empty stack- pointed by a pointer-Top

{ createNode(Top);

Top =NULL;

}

2. StackType PushStack(StackType Stack, NodeType

NewNode)

// This Algorithm adds a NewNode at the top of 'stack'. Top is an pointer that

points to the topmost Stack node.

{ if Top ==NULL // first element in stack

NewNode->Next = NULL;

Top=NewNode;

Else NewNode->Next=Top;// General case

Top=NewNode;

}

3. Algorithm ElementType PopStack(StackType stack)

//This algorithm returns value of ElementType stored in topmost node of

stack. Temp is a temproary node used in pop process.

{ if Top==NULL

Print "Underflow"

Else

{

createNode(Temp);

Temp=Top;

Top= Top->next;

Return(temp->Data);

}

}

4. Abstract DestroyStack(StackType Stack)

//This algorithm returns values stored in data structure and free the memory used in data

structure implementation.

{{ if Top==NULL

Print "Underflow"

Else createNode(Temp);

while(NotEmpty(stack))

{

Temp=Top;

Top= Top->next;

Return(temp->Data);

}

}

5. Abstract ElementType Peep(StackType stack)

//This algorithm returns value of ElementType stored

in topmost node of stack.

{ if Top==NULL

Print "Error Message"

Else

Return(Top->Data);

}

6. Abstract DisplayStack(StackType stack)

//This algorithm Prints all the Elements stored in stack. Temp purpose?

{ if Top==NULL

Print "Error Message"

Else {createNode(Temp)

Temp=Top;

While(Temp!=Null)

Print(Temp->Data);

Temp= Temp->next;

}

}

Parentheses matching algorithm

Algorithm Boolean ParenMatch(X,n):

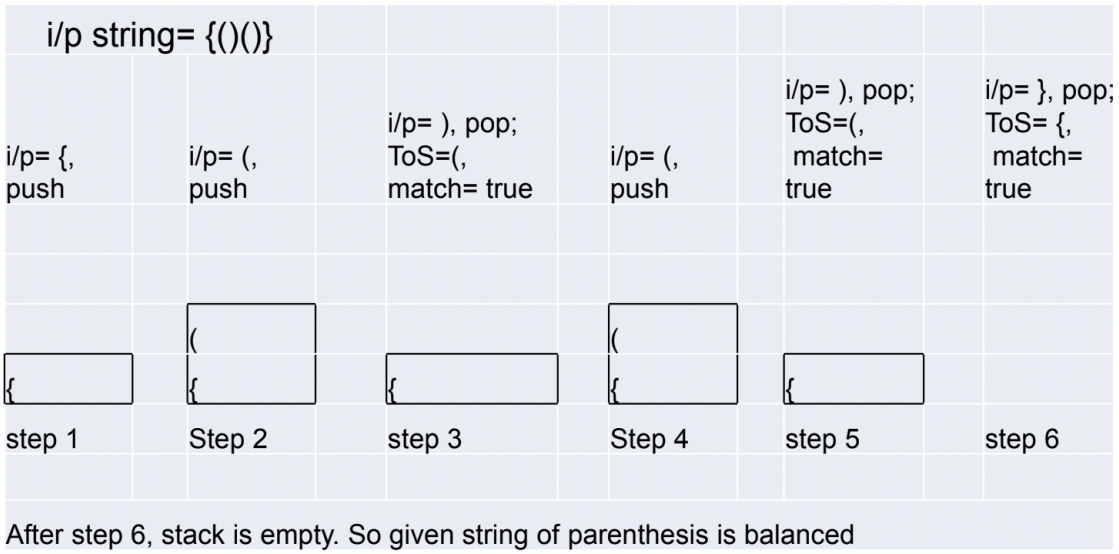
Input: An array X of n tokens, each of which is either a grouping symbol, a variable, an arithmetic operator, or a number

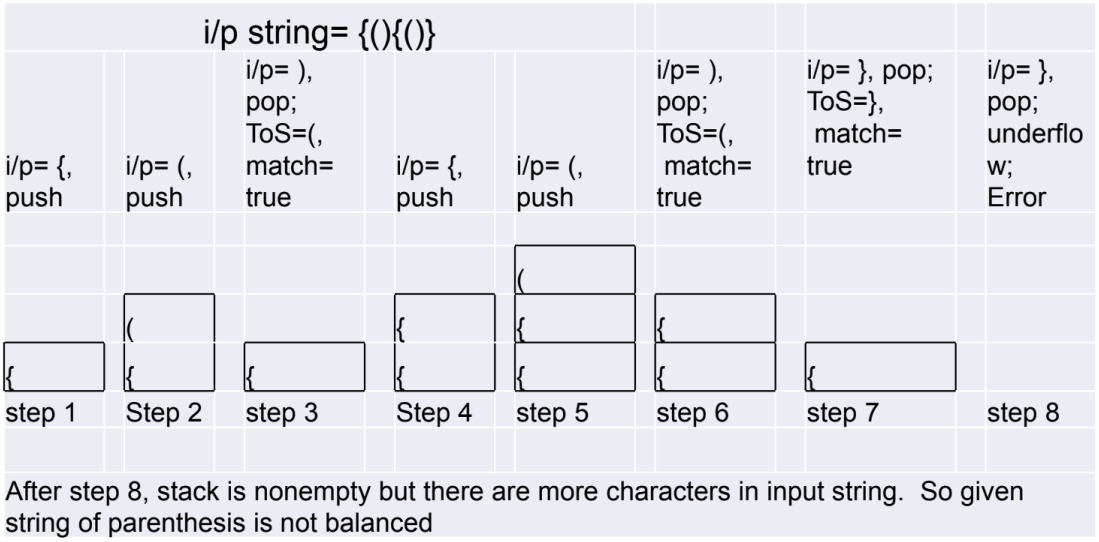
Output: true if and only if all the grouping symbols in X match

Let S be an empty stack

```
for i=0 to n-1 do
if X[i] is an opening grouping symbol then
S.push(X[i])
else if X[i] is a closing grouping symbol then
if S.isEmpty() then

return false {nothing to match with}
if S.pop() does not match the type of X[i] then

return false {wrong type}

if S.isEmpty() then
return true {every symbol matched}
else
return false {some symbols were never matched}
```

Example:


i/p string= {()()}
i/p= {, push
i/p= (, push
i/p=), pop; ToS=(, match= true
i/p= (, push
i/p=), pop; ToS=(, match= true
i/p= }, pop; ToS= {, match= true
step 1
Step 2
step 3
Step 4
step 5
step 6
After step 6, stack is empty. So given string of parenthesis is balanced


i/p string= {(){()}
i/p= {, push
i/p= (, push
i/p=), pop; ToS=(, match= true
i/p= {, push
i/p= (, push
i/p=), pop; ToS=(, match= true
i/p= }, pop; ToS=}, match= true
i/p= }, pop; underflow; Error
step 1
Step 2
step 3
Step 4
step 5
step 6
step 7
step 8
After step 8, stack is nonempty but there are more characters in input string. So given string of parenthesis is not balanced

Sourcecode:

```
/*********************************************************************
*********
Stack- dynamic implementation.

*********************************************************************
*********/
#include <iostream>

using namespace std;


class stack_element{
   public:
   stack_element *next;
   int value;

   stack_element(){

   }
   stack_element(stack_element next_element, int value_){
   next=&next_element;
   value=value_;
   }
   void set_value( int value_){
      value=value_;
   }

   int get_value(){
    //  cout<<"\n"<<value;
      return value;
   }
    stack_element* get_next(){
      return next;
   }
   void point(stack_element* next_element){
   next=next_element;
   }


};

 class stack{
```

```
 public:
stack_element null;

stack_element top;
stack(){

   top.point(&null);

}
bool empty(){
   if(&null==top.get_next()){
      return true;
   }
   return false;
}
void push(int value){

 stack_element *s = new stack_element();

s->set_value(value);
s->point(top.get_next());
top.point(s);

}
int pop(){
   if(empty()==false){
   int temp =top.get_next()->get_value();
   top.point(top.get_next()->get_next());
   return temp;
}
}
int peek(){
   return top.get_next()->get_value();
}
void print(){
    stack_element *p = new stack_element();
    p->point(top.get_next());
    while(&*p->get_next()!=&null){
   int temp2 =p->get_next()->get_value();
   cout<<"\n"<<temp2;
      p->point(p->get_next()->get_next());
    }
    delete p;

}
};
```

```
int main()
{
   stack my_stack;

     string str;
  //BAD QUALITY CODE
    // Taking string input using getline()
    getline(cin, str);
    for (int i = 0; i <= str.length(); i++){
       if(str[i]=='('){
          my_stack.push(0);
       }
       if(str[i]=='{'){
          my_stack.push(1);
       }
       if(str[i]=='['){
          my_stack.push(2);
       }
       if(str[i]==')'){
          if(my_stack.peek()!=0){
             cout<<"wrong";
             return 0;
          }
          else{
             my_stack.pop();
          }
       }
       if(str[i]=='}'){
          if(my_stack.peek()!=1){
             cout<<"wrong";
             return 0;
          }
          else{
             my_stack.pop();
          }
       }
       if(str[i]==']'){
          if(my_stack.peek()!=2){
             cout<<"wrong";
             return 0;
          }
          else{
             my_stack.pop();
          }
       }
```

```
    }
    if(my_stack.empty()==false){
        cout<<"wrong";
        return 0;
    }
    cout<<"correct";
    my_stack.print();



    return 0;
}
```

Output Screenshots:

(3(4((5(5(6())))4)

wrong

(2+3)

correct

(2+(3*4))

correct

2+(3*4(

wrong

(2*3]

wrong

Conclusion:
Thus we have applied stack in a problem statement. This type of problem is found in editors where parentheses come into the picture. If we do not apply data structures in daily life and in computers then those remain abstract concepts but by using them we have transformed them into practical use.

NOTE- All references except code have been taken from PPTs.