STACKS IN C++

INTRODUCTION TO STACK

DEFINITION:

- ❑ A stack is a data structure that provides temporary storage of data in such a way that the element stored last will be retrieved first.
- ❑ This method is also called LIFO – Last In First Out.

EXAMPLE:

In real life we can think of stack as a stack of copies, stack of plates etc. The first copy put in the stack is the last one to be removed. Similarly last copy to put in the stack is the first one to be removed.

OPERATIONS ON STACK

- A stack is a linear data structure.
- It is controlled by two operations: **push** and **pop**.
- Both the operations take place from one end of the stack, usually called top. Top points to the current top position of the stack.
- Push operation adds an element to the top of the stack.
- Pop operation removes an element from the top of the stack.
- These two operations implements the LIFO method .

IMPLEMENTATION OF STACK

Stack can be implemented in two ways:

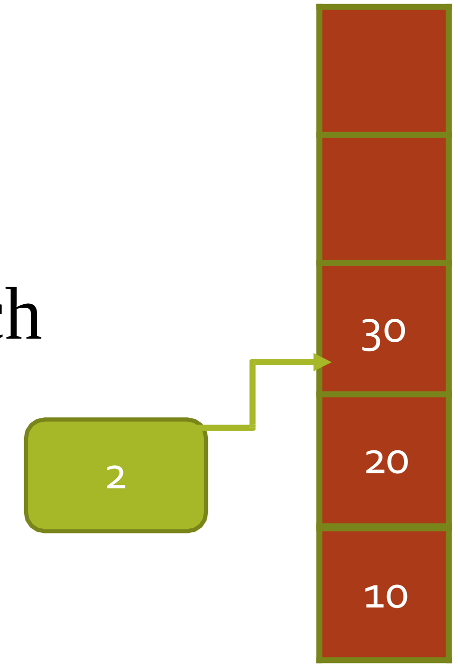
- ❑ As an Array
- ❑ As a Linked List

STACK AS AN ARRAY

Array implementation of stack uses

- ❑ an array to store data
- ❑ an integer type variable usually called the top, which points to the top of the stack.

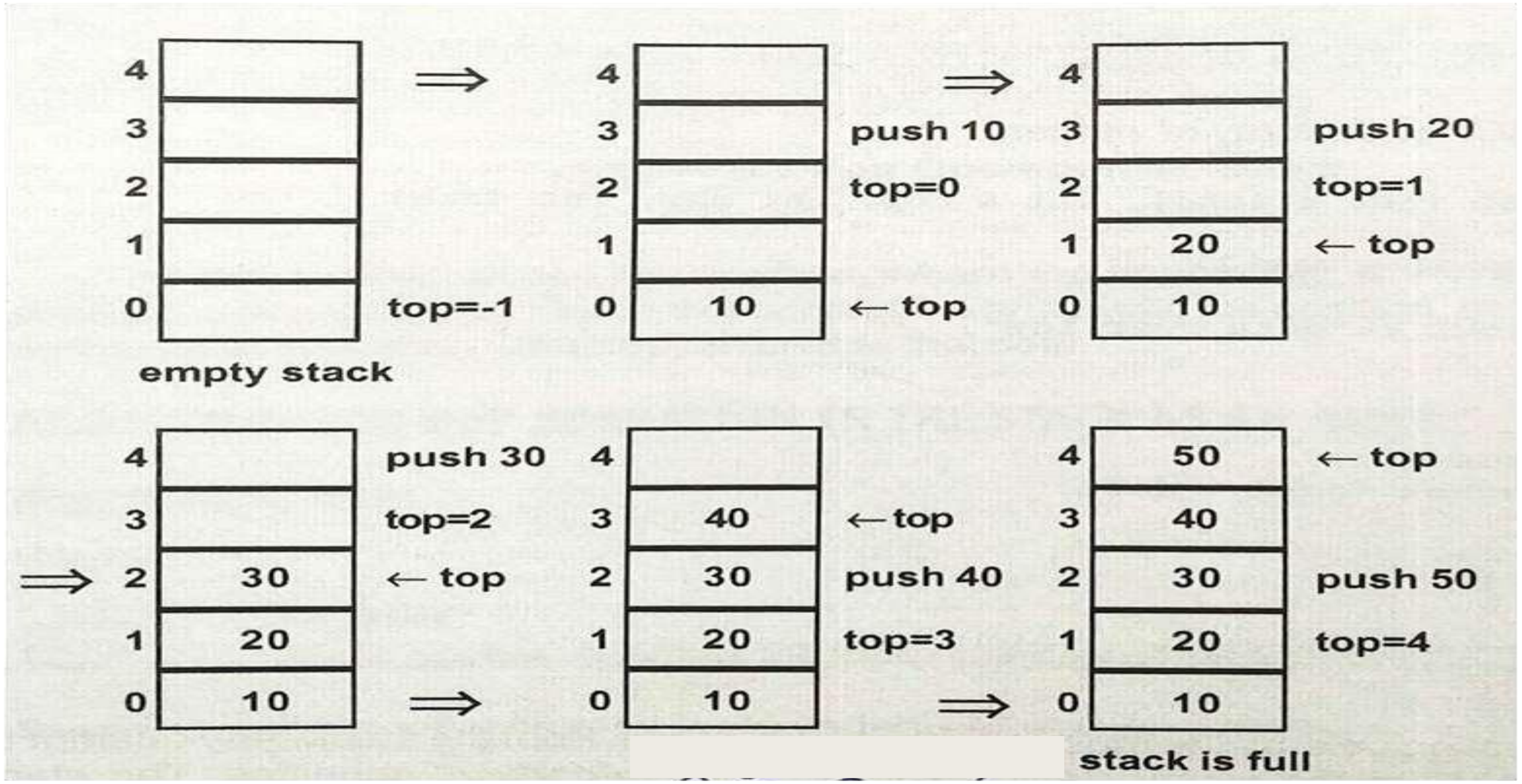
NOTE: The variable top contains the index number of the top most filled element of the array.



PUSH OPERATION

- ❑ Initially when the stack is empty, TOP can have any integer value other than any valid index number of the array. Let TOP =
  -1;
- ❑ The first element in the empty stack goes to the 0th position of the array and the Top is initialized to value 0.
- ❑ After this every push operation increase the TOP by 1 and inserts new data at that particular position.
- ❑ As arrays are fixed in length, elements can not be inserted beyond the maximum size of the array. Pushing data beyond the maximum size of the stack results in “data overflow”.

PUSH OPERATION EXPLAINED

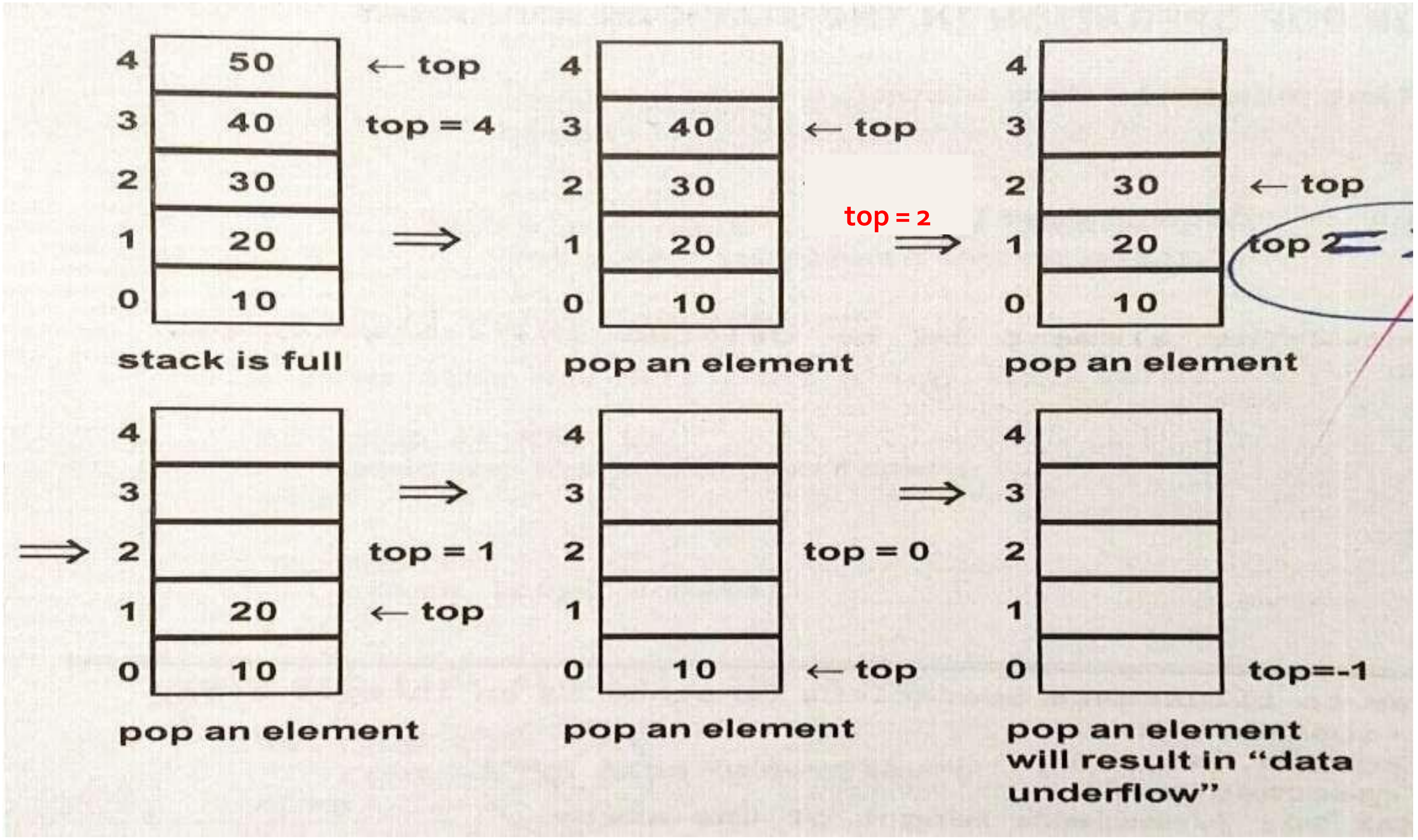


NOTE: Pushing one more element into the stack will result in overflow.

POP OPERATION

- ❑ The pop operation deletes the most recently entered item from the stack.
- ❑ Any attempt to delete an element from the empty stack results in “data underflow” condition.
- ❑ The variable TOP contains the index number of the current top position of the stack.
- ❑ Each time the pop operation is performed, the TOP variable is decremented by 1.

POP OPERATION EXPLAINED



PROGRAM TO ILLUSTRATE OPERATIONS ON STACK AS AN INTEGER ARRAY

```
#include<iostream
.h>
#include<process.
h>  const int size=5
class stack
{     int arr[size];
      int top; //top will point to the last item
            //pushed onto the stack
  public:
      stack() { top=-1; } //constructor to
      create an
                  //empty stack, top=-1
                  indicate
               //that no item is present in the
      array  void push(int item)
      {     if(top==size-1)
            cout<<"stack is full, given item
                  cannot be added";
            else
            {     top++;
                  arr[top]=item
                  ;}
```

```
void  pop()
{ if (top== -1)
      cout<<"Stack is
 empty "; else
      {
      cout<<arr[top
      ]; top - -; }
}
void display()
{
      for(int
            i=top;i>=0;i--)
            cout<<arr[to
            p];
}
void main()
{     stack  s1;
      int
      ch,data;
      while(1)
      {
```

```
cout<<"1.push"<<"\n
2.pop"<<"\n3.display"
<<"\n 4.exit";
cout<<"Enter choice :";
cin>>c
h;  if
(ch==1)
{      cout<<"enter item to
      be pushed in the
      stack"; cin>>data;
      s1.push(data);
}
else if (ch == 2)
      {
s1.pop(); }
else if (ch==3)
      { s1.display(); }
else
      exit(-1);
} }
```

PROGRAM TO ILLUSTRATE OPERATIONS ON STACK **AS AN ARRAY OF OBJECTS**

```
#include<iostream.h>
#include<process.h>
const int size=5
struct stu
{
        int roll;
        char name[20];
        float total;
};
class stack
        {       stu arr[size];
                int top;
        public:
                stack()   { top=-1; }
                void push(int r, char
                                n[20],float tot)
                {       if  (top==size-1)
        cout<<"overflow";
```

```
        else
        {       top++;
                arr[top].roll=r;
                strcpy(arr[top].name,n);
                arr[top].total=tot;       }
void  pop()
        {       if (top== -1)
                cout<<"Stack is empty ";
                else
               {       cout<<arr[top].roll<<
                arr[top].name<<arr[top].tot;
                top - -;    }
void display()
{
for(int i=top;i>=0;i--)
        cout<<arr[top].roll<<
        arr[top].name<<arr[top].tot<<"\n";
}
```

```
void main()
{     stack s1;
        int ch,data;
        while(1)
        {  cout<<"1.push"<<"\n
        2.pop"<<"\n3.display"<<"\n
        4.exit";
        cout<<"Enter choice :";
        cin>>ch;
        if (ch==1)
        {       cin>>data;
                s1.push(data); }
        else if (ch == 2)
                {  s1.pop();  }
        else if (ch==3)
                {  s1.display();  }
        else
                exit(-1);    } }
```

A stack is an appropriate data structure on which information is stored and then later retrieved in reverse order. The application of stack are as follows:

- When a program executes, stack is used to store the return address at time of function call. After the execution of the function is over, return address is popped from stack and control is returned back to the calling function.

- Converting an infix expression o postfix operation and to evaluate the postfix expression.

- Reversing an array, converting decimal number into binary number etc.

INFIX AND POSTFIX OPERATIONS

- ❑ The infix expression is a conventional algebraic expression, in which the operator is in between the operands.
  For example: a+b

- ❑ In the postfix expression the operator is placed after the operands.
  For example: ab+

The postfix expressions are special because

- They do not use parenthesis
- The order of expression is uniquely reconstructed from the expression itself.

These expressions are useful in computer applications.

CONVERTING INFIX TO POSTFIX EXPRESSION

CONVERT_I_P(I,P,STACK) Where I = infix expression, P = postfix expression, STACK = stack as an array

Step 1. Push ‘(‘ into STACK and add ‘)’ to the end of I.

Step 2. Scan expression I from left to right and repeat steps 3-6 for each element of I until the stack is empty.

Step 3. If scanned element =‘**(**‘ , push it into the STACK. Step 4. If scanned element = **operand**, add it to P.

Step 5. If scanned element = **operator**, then:

a)Repeatedly pop from STACK and add to P each operator from the top of the stack until the operator at the top of the stack has lower precedence than the operator extracted from I.

b) Add scanned operator to the top of the STACK.

Step 6. If scanned element = ‘**)**’ then:

c)Repeatedly pop from STACK and add to P each operator until the left parenthesis is encountered.

d) Pop the left parenthesis but do not add to P.

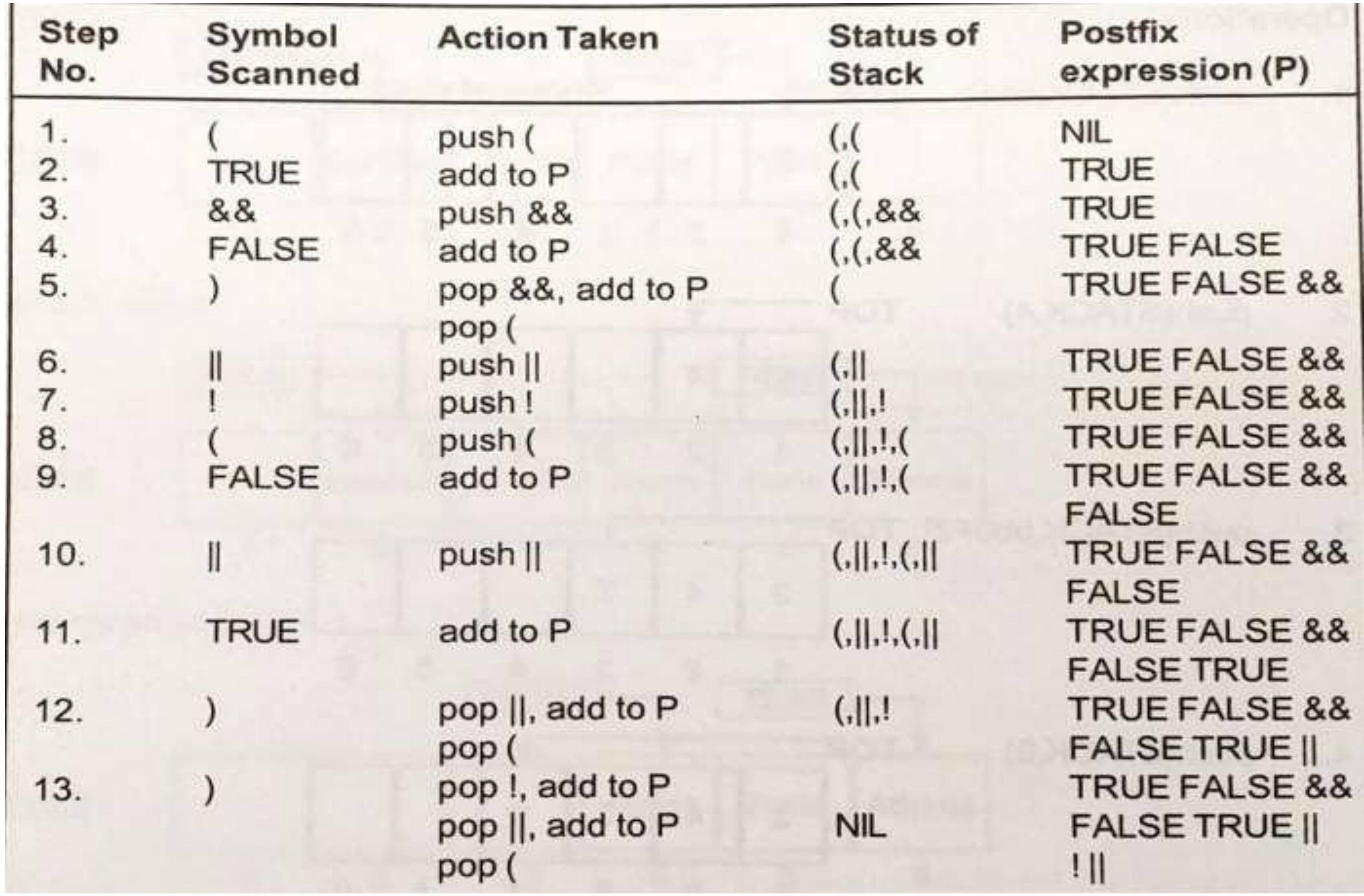
CONVERTING INFIX TO POSTFIX EXPRESSION

Q Convert the following infix operation to
postfix operation.

((TRUE && FALSE) || ! (FALSE || TRUE))

Ans. The Postfix expression is:

TRUE FALSE && FALSE TRUE || ! ||



Step No.	Symbol Scanned	Action Taken	Status of Stack	Postfix expression (P)
1.	(	push (	(,(	NIL
2.	TRUE	add to P	(,(	TRUE
3.	&&	push &&	(,(,&&	TRUE
4.	FALSE	add to P	(,(,&&	TRUE FALSE
5.	)	pop &&, add to P pop (	(	TRUE FALSE &&
6.	||	push ||	(,||	TRUE FALSE &&
7.	!	push !	(,||,!	TRUE FALSE &&
8.	(	push (	(,||,!,(	TRUE FALSE &&
9.	FALSE	add to P	(,||,!,(	TRUE FALSE && FALSE
10.	||	push ||	(,||,!,(,||	TRUE FALSE && FALSE
11.	TRUE	add to P	(,||,!,(,||	TRUE FALSE && FALSE TRUE
12.	)	pop ||, add to P pop (	(,||,!	TRUE FALSE && FALSE TRUE ||
13.	)	pop !, add to P pop ||, add to P pop (	NIL	TRUE FALSE && FALSE TRUE || ! ||

EVALUATION OF POSTFIX EXPRESSION

Evaluate(P, result, STACK)

where P=Postfix expression, STACK =stack, result=to hold the result of evaluation

```
Step 1. Scan the postfix expression P from left to right and repeat the steps 2, 3
until the STACK is empty.
Step 2. If the scanned element = operand, push it into the STACK.
Step 3. If  the scanned element = operator, then
        a) Pop the two operands  viz operand A and operand B from the STACK.
        b) Evaluate ((operand B) operator (operand A))
c)      Push the result of evaluation into the STACK.
Step 4. Set result=top most element of the STACK.
```

EVALUATING POSTFIX EXPRESSION

Q Evaluate the following Postfix expression:

TRUE FALSE || FALSE TRUE && ! ||

Ans. TRUE

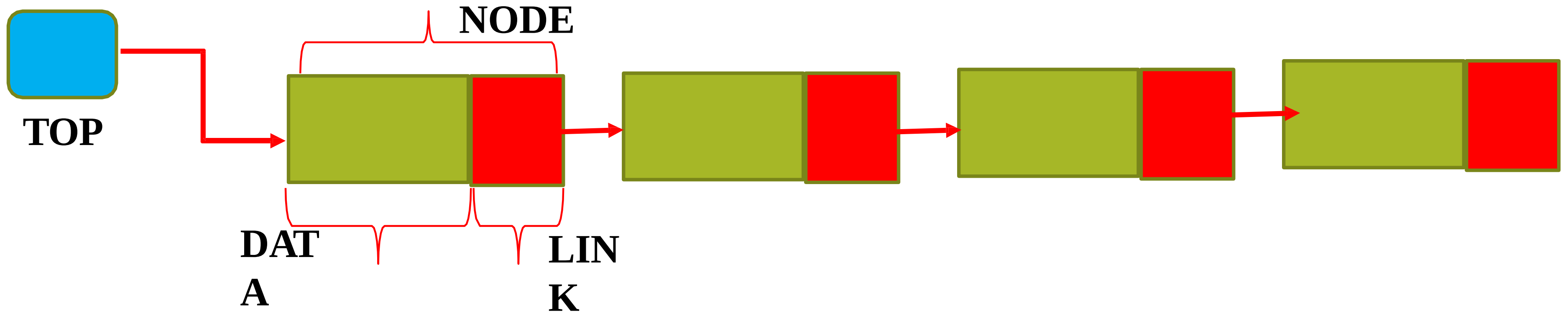
Step No.	Symbol Scanned	Operation Performed	Stack Status
1	TRUE		TRUE
2	FALSE		TRUE FALSE
3	||	TRUE || FALSE =TRUE	TRUE
4	FALSE		TRUE FALSE
5	TRUE		TRUE FALSE TRUE
6	&&	FALSE &&TRUE = FALSE	TRUE FALSE
7	!	! FALSE =TRUE	TRUE TRUE
8	||	TRUE || TRUE	TRUE

STACK AS A LINKED LIST

LINKED LIST IMPLEMENTATION OF STACK USES

- A linked list to store data
- A pointer TOP pointing to the top most position of the stack.

NOTE: Each node of a stack as a linked list has two parts : data part and link part and is created with the help of self referential structure.
The data part stores the data and link part stores the address of the next node of the linked list.



PROGRAM TO ILLUSTRATE OPERATIONS ON STACK

AS A LINKED LIST

```
void stack::push()
{
    node *temp;  temp=new
    node;  cout<<"Enter roll
    :";  cin>>temp->roll;
    cout<<"Enter name :";
    cin>>temp->name;
    cout<<"Enter total :";
    cin>>temp->total;  temp-
    >next=top;  top=temp;
}
void stack::pop()
{
    if(top!=NULL)
    {
```

```
#include<iostream.h>
#include<conio.h>
struct node
{
        int roll;
        char name[20];
        float total;
        node *next;
};
class stack
{
    node *top;
public :
    stack()
    { top=NULL;}
     void push();
     void pop();
     void display();
};
```

```
        node *temp=top;
        top=top->next;
        cout<<temp->roll<<temp-
       >name<<temp->total
              <<"deleted
        ";  delete temp;
    }
    else
        cout<<"Stack empty";
}
void stack::display()
{
    node *temp=top;
    while(temp!=NUL
    L)
    {
     cout<<temp->roll<<temp-
      >name<<temp->total<<"
     "; temp=temp->next;
    }  }
```

```
void main()
{  stack  st;
    char ch;
    do
    {     cout<<"stack
       options\nP
       push
       \nO Pop \nD
       Display \nQ
       quit"; cin>>ch;
       switch(ch)
        {  case 'P':
             st.push();break
       ; case 'O':
       st.pop();break;
       case 'D':
       st.display();break;
      }
    }while(ch!='Q');
}
```

PROGRAM TO ILLUSTRATE OPERATIONS ON STACK AS A LINKED LIST : EXPLANATION

```
struct node
{
        int roll;
        char name[20];
        float total;
        node *next;
};
```

Self Referential Structure: These are special structures which contains pointers to themselves.
Here, next is a pointer of type node itself.
Self Referential structures are needed to create Linked Lists.

```
class stack
{
    node *top;
public :
    stack()
    { top=NULL;}
     void push();
     void pop();
     void
     display();
};
```

class stack: It contains pointer TOP which will point to the top of the stack.
The constructor function initializes TOP to NULL.

PROGRAM TO ILLUSTRATE OPERATIONS ON STACK AS A LINKED LIST : EXPLANATION

```
void stack::push()
{
    node *temp;  // pointer  of type node
    temp=new node;  // new operator  will create a new
                             //node  and address of new node
                             // is stored in temp
    cout<<"Enter roll :";  ⎫
    cin>>temp->roll;       ⎪     These lines will input
cout<<"Enter name :";      ⎪      values into  roll , name
    cin>>temp->name;       ⎬      and total of newly
    cout<<"Enter total :"; ⎪      created node.
    cin>>temp->total;      ⎭
    temp->next=top;         //  address stored in TOP gets
                                 //stored in next of newly
                                 //created node.
    top=temp;     //  Top will  contain address of
                       // newly  created node
}
```

```
void stack::pop()
{
    if(top!=NULL) //  if stack is not empty
    {
        node *temp=top;    //  temp is a pointer
                                  //containing address of first node
        top=top->next;     // top will now contain address of
                                  //second node

        cout<<temp->roll<<            ⎫  // This line will
      temp->name<<temp->total        ⎬  display
             <<"deleted";             ⎭   // data stored in
                                             // deleted  node
        delete temp;    // delete operator will delete the node
                             // stored at temp
    }
    else
        cout<<"Stack empty";}
```