UNIT-2 DINESH VASHISHT

Stack

(Non Primitive Linear data structure)

A stack is an ordered group (list) of homogeneous elements. Addition of new element and deletion of existing element are done from only one end, called top of the stack. The item last added will be the first to be removed from the stack. A stack is also called Last-in First out

Basic stack operations :

1. Push : the process of adding a new element to the top of the stack is called push operation.
2. POP : the process of deleting an ~~ele~~ element from top of the stack is called POP operation.

Overflow : This is the situation when the stack becomes full and no more elements can be pushed onto the stack. At this point the top is present at the highest location of the stack.

Underflow : when stack contains no elements. The top is present at the bottom of the stack.

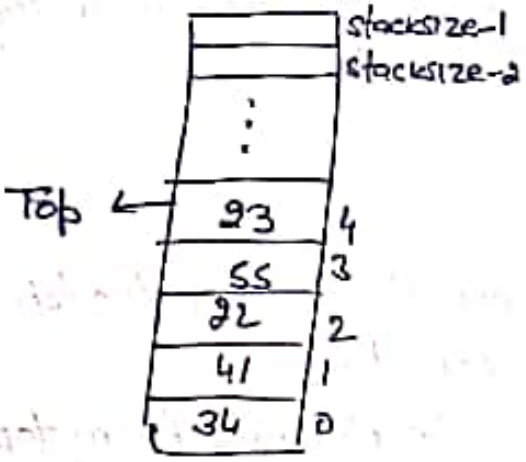
Top : This refers to the top of the stack. It is used to check the overflow and underflow. Initially top stores -1

MAXSIZE : we use this term to refer to the maximum size of the stack.

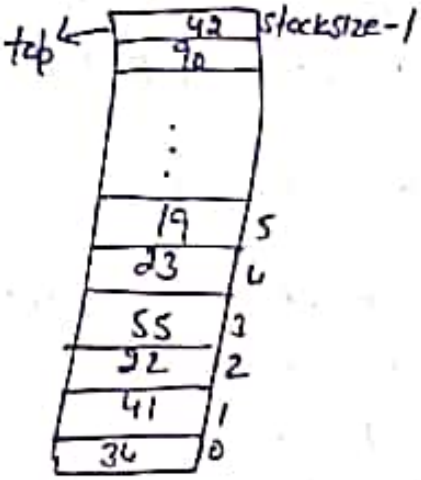
Array Implementation of stack:

The stack can be implemented using arrays and linked lists.

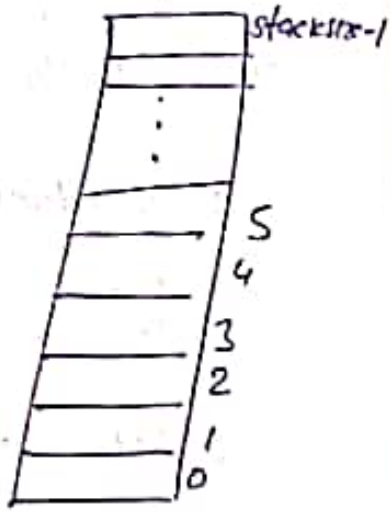
One way to implement stack is to have a data structure where a variable called top keeps the location of elements in the stack.



A stack with 5 elements



A full stack



top=-1

empty stack

Push Operation

```
void push()
{ int item;
  if(top==(MAXSIZE-1))
    printf("overflow");
  else
  { top=top+1;
    printf("enter item to be pushed");
    scanf("%d",&item);
    stack[top]=item;
  }
}
```

Algo

PUSH(stack, size, top, item)

stack is linear array
size is size of stack
top is top of stack
item: value to be inserted

1. [check overflow]
   If(top=size-1)
   print overflow, exit
   otherwise goto step 2
2. top=top+1
3. stack[top]=item
4. exit.

POP

Algo

Pop (stack, top, item)

step 1: [check underflow]
    if (top == -1)
    print underflow and exit
    otherwise goto step 2

step 2: item = stack[top]

step 3: top = top - 1,

step 4: stop

C code

```
void pop (int stack[])
{
  int item,.
  if (top==-1)
  {
    printf("underflow"),.
    exit(),.
  }
  else
  {
    item= stack[top],.
    top--,.
  printf("deleted item=%d", item),.
}
```

Display Contents of stack:

```
void display ()
{
  int i,.
  if (top==-1)
     printf("stack is empty"),.
  else
  {
     for(i=top,.i>=0,.i--)
        printf("%d\t", stack[i]),.
  }
}
```

Applications of stack

A stack is LIFO data structure and is suitable for applications in which information must be saved and later retrieved in reverse order

1. The runtime stack used by a process to keep track of methods in progress.
2. Search Problems
3. Undo, redo, back, forward
4. Recursion
5. Arithmatic expression
   a) Infix notation
   b) Prefix notation or Polish notation
   c) Postfix notation or reverse polish notation.
   d) evaluation of postfix expression.
   → Infix to Postfix conversion.
   → Infix to Prefix conversion.

Types of Notations :

Infix Notation

A + B

Prefix Notation or Polish Notation

+ AB

Postfix or Reverse Polish Notation

AB +

Advantage of Postfix Notation →

- → No Parenthesis required
- → Simple Algo to evaluated Postfix expression
- → No need to ~~remb~~ remember precedence and associativity rules of various operators.

Infix to Postfix Conversion

Q1 A+B/C−D

Sol"

A+(BC/)−D

A+ T −D

(A+T)−D

(AT+)−D

S −D

S D−

AT+D−

ABC/+D−

Any

Q2 (A+B) * C/D

Sol"

(AB+) * C/D

T * C/D

(T*C) /D

(TC*) /D

S/D

SD/

TC*D/

AB+C*D/ Ans

Algorithm To convert infix to Postfix expression.

Algo: Postfix (Q, P)

(Let Q is an arithmatic expression in infix notation. This algo finds the equivalent postfix expression P)

1. Push '(' onto the stack and add ')' to the end of expression
2. Scan Q from left to right, repeat step 3 to 6 until stack is empty
3. If OPREND, Add it to P
4. If '(', Push it onto stack
5. If operator then
   a) Repeatedly pop from the stack and add P each operator that has the same precedence or higher precedence than the operator encountered.
   b) Add operator to the stack
6. If ')' then
   a) Pop from the stack and add to P each operator until a left parenthesis is encountered
   b) Remove left parenthesis.
7. exit.

C.f Convert A+B/C−D to postfix expression.

Sr. No	Symbol scanned	stack	Postfix exp.
1		(	
2	A	(	A
3	+	(+	A
4	B	(+	AB
5	/	(+/	AB
6	C	(+/	ABC
7	−	(−	ABC/+
8	D	(−	ABC/+D
9	)		ABC/+D−

Algorithm : [To evaluate Postfix expression]

1. Add a right parenthesis to the end of P
2. Scan P from left to right and repeat step 3 to 6 until ')' is encountered.
3. If an operand is encountered, put it on the stack
4. If an operator is encountered
   a) Pop top two elements from the stack A, B
   b) evaluate A op B
   c) Push the result of (b) on the stack.
   //end of if
   end of step 2 loop
5. Set value equal to the top element of stack
6. exit.

Q Convert the following infix expressions into postfix expressions.

a) $A-(B/C+(D\%E*F)/G)*H$

~~(B) (A-(B/C+(D%E*F~~

Sol^n $A-(B/C+(D\%E*F)/G)*H$

Sr. No	Symbol Scanned.	stack	Postfix expression.
1	—	(	
2	A	(	A
3	−	(−	A
4	(	(−(	A
5	B	(−(	AB
6	/	(−(/	AB
7	C	(−(/	ABC
8	+	(−(+	ABC/
9	(	(−(+(	ABC/
10	D	(−(+(	ABC/D
11	%	(−(+(%	ABC/D
12	E	(−(+(%	ABC/DE
13	*	(−(+(*	ABC/DE%
14	F	(−(+(*	ABC/DE%F
15	)	(−(+	ABC/DE%F*
16	/	(−(+/	ABC/DE%F*
17	G	(−(+/	ABC/DE%F*G
18	)	(−(	ABC/DE%F*G/+
19	*	(−*	ABC/DE%F*G/+
20	H	(−*	ABC/DE%F*G/H
21	)	(	ABC/DE%F*G/H*−

Ans

Q2 Evaluate the postfix expression and show the value of stack at each step

Infix P = 9 − ((3*4)+8)/4

Postfix 9 3 4 * 8 + 4 /

Sol^n

Conversion to Postfix

Step No	Symbol scanned	Stack	Postfix expr
1	—	(	
2	9	(	9
3	−	(−	9
4	(	(−(	9
5	(	(−((	9
6	3	(−((	9 3
7	*	(−((*	9 3
8	4	(−((*	9 3 4
9	)	(−(	9 3 4 *
10	+	(−(+	9 3 4 *
11	8	(−(+	9 3 4 * 8
12	)	(−	9 3 4 * 8 +
13	/	(−/	9 3 4 * 8 +
14	4	(−/	9 3 4 * 8 + 4
15	)		9 3 4 * 8 + 4 / −

Ans.

Now evaluate this postfix expression.

PTO

Evaluate Postfix expression

step	character scanned	stack
1	9	9
2	3	9 3
3	4	9 3 4
4	*	9 12
5	8	9 12 8
6	+	9 20
7	4	9 20 4
8	/	9 5
9	−	4

Ans

Algorithm to Convert Infix Notation to Prefix Notation →

Algo: INFIX-TO-PREFIX (Q, P)

where Q is input expression written in infix notation
P is resultant expression in prefix notation.

1. Scan Q from right to left and repeat step 2 to 5 until Q is empty
2. If it is an operand, Add it to P
3. If it is closing parenthesis ')', Add it to stack
4. If it is an operator (X) then (a) Repeatedly Pop the operator from the stack which has higher precedence than (X)
   (b) Add (X) to stack.
5. If it is an opening parenthesis, Pop operator from the stack and Add them to P until a closing parenthesis is encountered.
6. Reverse P.

eg Give Prefix Form for (A*B+(C/D))-F

Sol^n

(A*B+(/CD))-F
(A*B+T)-F
((A*B)+T)-F
((*AB)+T)-F
(S+T)-F
(+ST)-F
Q-F
-QF
-+STF
-+*ABTF
-+*AB/CDF Ans

Q Convert (A*B+(C/D))-F into Prefix Form and show content of stack at each step.

Soln.

Step No	Symbol Scanned	Stack	P
1	F		F
2	-	-	F
3	)	-)	F
4	)	-))	F
5	D	-))	FD
6	/	-))/	FD
7	C	-))/	FDC
8	(	-)	FDC/
9	+	-)+	FDC/
10	B	-)+	FDC/B
11	*	-)+*	FDC/B
12	A	-)+*	FDC/BA
13	(	-	FDC/BA*+
14			FDC/BA*+-

Reverse P

-+*AB/CDF Ans

Q. Convert A/B↑C-D to prefix Notation and show the contents of the stack.

A