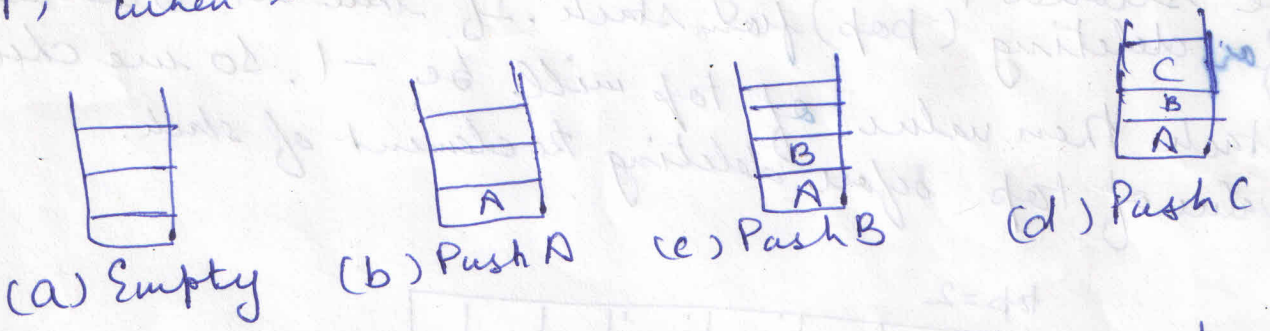
Module - IV

Stack and Queue

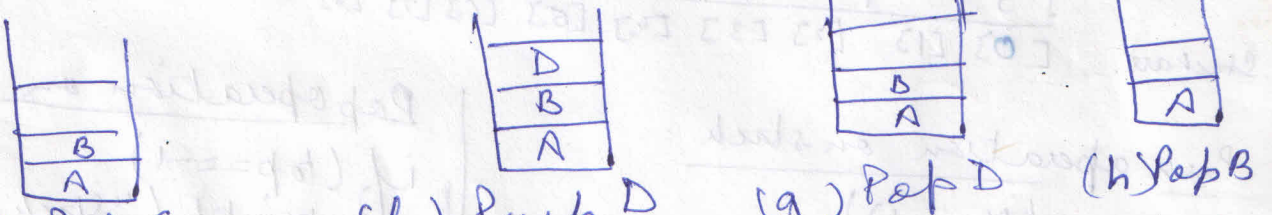
- It may be possible that there is a need of data structure which takes operations on only one end i.e beginning or end of the list.
- In Linked list & arrays ~~use~~ operation can be done at any place of the list.
- Stack & Queue are data structures which take operations only at one end.

Stack

- Stack is defined as a list of elements in which insertion & deletion of the element only at the top of the stack.
- There are two operations on stack:
  1.) Push, when item is added in stack.
  2.) Pop, when item is deleted from stack.



(a) Empty (b) Push A (c) Push B (d) Push C

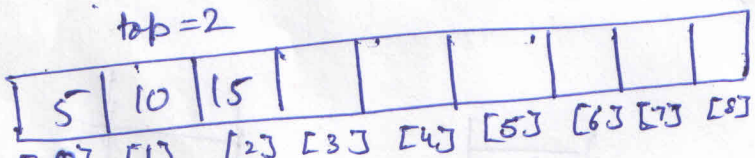


(e) Pop C (f) Push D (g) Pop D (h) Pop B

- The elements are added (pushed) at the last in stack can be deleted (popped) from the stack.
- Its behaviour is like last in first out.
- So stack is also called LIFO DS.

Array Implementation of Stack

- Since stack is a collection of same type of elements, so array can be used for implementing stack.
- In array elements can be pushed one by one from 0th pos. to n−1th pos.
- In array, any element can be added or deleted at any place, but in stack elements are pushed or poped from the top only, therefore 'top' variable is taken, which keeps the position of the top element in array.
- It may be possible that a condition arises when there is no place for adding (push) the element in the array. This is called overflow, hence therefore first of all check the value of top with size of array.
- The second possibility arises when there is no element for deleting (pop) from stack. If there is no element in the stack then value of top will be −1. So we check the value of top before deleting the element of stack.



Stack arr.

Push operation on stack.

```
If (top == (Max-1))
    printf ("Stack Overflow\n");
else
{   top = top+1;
    stack_arr[top] = pushed_item;
}
```

(Here top always pts. to last added item of stack)

Pop operation on stack.

```
if (top == -1)
    printf ("Stack Underflow\n");
else
{   printf ("Popped element is:
        %d \n", stack_arr[top]);
    top = top-1;
}
```

Here top pts to last pushed item of stack. After pop operation now it will pt. previous

WAP of stack using array

```
#include <stdio.h>
#define Max 5
int top = -1;
int stack_arr[Max];

main()
{
    int choice;
    while (1)
    {
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Quit\n");
        printf("Enter your choice:");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1: push();
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
            case 4: exit(1);
            default: printf("Wrong choice\n");
        }
    }
}

push()
{
    int pushed_item;
    if (top == (Max-1))
        printf("Stack Overflow\n");
    else
    {
        printf("Enter the item to be pushed:");
        scanf("%d", &pushed_item);
        top = top+1;
        stack_arr[top] = pushed_item;
    }
}

pop()
{
    if (top == -1)
        printf("Stack Underflow\n");
    else
    {
        printf("Popped element is: %d\n", stack_arr[top]);
        top = top-1;
    }
}

display()
{
    int i;
    if (top == -1)
        printf("Stack is empty\n");
    else
    {
        printf("Stack elements:\n");
        for (i = top; i >= 0; i--)
            printf("%d\n", stack_arr[i]);
    }
}
```

Operations on Stacks

- CreateStack(s)
- Push(s,i)
- Pop(s)
- Peek(s) - to access the top element of the stack s without removing it from the stack s.
- IsFull(s)
- IsEmpty(s)

```
VoidCreateStack (Stack *ps)
{
    ps->top = -1;
}

int peek(Stack *ps)
{
    return (ps->element[ps->top]);
```

```
# Define MAX10
Struct stack
{
 int top;
 int stk[MAXJ;
} s;
```

or

```
# Define MAX10
int stack [MAXJ;
int top = -1;
```

Test for stack overflow condition:

- Before we insert new item onto the stack, it is necessary to test whether stack still have some space to accomodate the incoming element i.e to test that whether the stack is full or not.
- If it is not full then push operation can be performed to insert the element at the top of the stack.
- This test is performed by comparing the value of the top with value MAX-1.

```
Int. Isfull ()
{
 if (top == MAX-1)  /* overflow */
    return 1;
 else return 0;
}
```

Test for stack underflow:

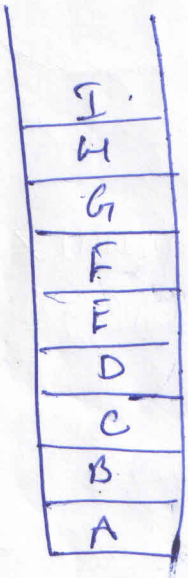
- Before remove on item from a stack, it is necessary to test whether stack still have some element i.e. to test that whether the stack is empty or not.
- If it is not empty then the pop operation can be performed to remove the top element.

- This test is performed by comparing the value of top with sentinel value -1.

```
Int Isempty()
{
if(top == -1)  /* Underflow */
    return 1;
else
    return 0;
}
```

Example of overflow & Underflow:

Max = 9



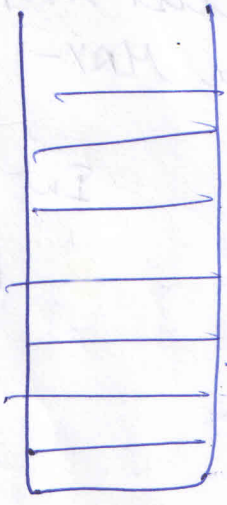
top = 9

Push()

i.e top = max

overflow

we can't insert any more element on to stack



Pop()

i.e underflow

we can't delete any element from stack

top = -1.

Applications of stack

1. Arithmetic expression evaluation.
2. Evaluation of a postfix & prefix expression
3. Reversing string
4. Implementation of recursive procedure
5. Back tracking.
6. Stacks are used to pass parameters b/w functions.
7. High level programming languages like C, etc. provides support for recursion used of stack for bookkeeping.

Arithmetic Expression

An arithematic expression is a combination of variables & connected by arithemetic operators & parenthesis.

$$X - Y/Z + (A+B) * C$$

Notation for expression

- There are three notations to represent an arithemetic expression :-

a) Infix
b) Prefix
c) Postfix

(a) Infix Notation.

<operand> <operator> <operand>.

A + B

X / Z

This is called infix coz the operator comes in b/w the operands:

operands : A, B, X, Y.
operator : +, /, −, +, *

(b) Prefix Notation

<operator> <operand> <operand>

(A + B)	+	A	B (+AB)
(X / Y)	/	X	Y (/XY)

- Here operator comes before the operands.
- It is also called POLISH Notation.

(c) Postfix Notation

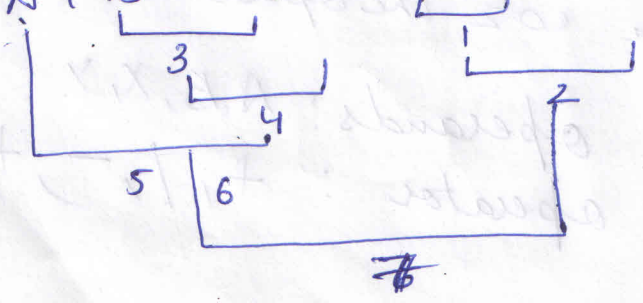
<operand> <operand> <operator>

A	B	+
X	Y	/

- It is also called reverse POLISH Notation.

Operators	Priority/Precedence	Associativity
^ (Multiplication)	2	Right to left
/ (Division)	3	Left to right
% (Modulus (Remainder))	3	"
+ (Plus)	4	"
- (Substraction)	4	"
( )	1	
{ }	1	
[ ]	1	

e.g A + B * C / D - E ^ F * G



Parenthesis Checker: that checks whether mathematical expression is properly parenthesised.

— Three sets of grouping symbols: Standard parentheses '( )'
braces '{ }'
brackets '[ ]'

Steps Involved to convert the infix expression into postfix expression.

(1) Add the unique symbol '#' into stack & at the end of array infix
(2) Scan the symbol of array infix one by one from left to right.
(3) If symbol is left parenthesis '(' then add it to the stack.
(4) If symbol is operand then add it to array postfix.
(5) (i) If symbol is operator then pop the operators which have same precedence or higher precedence than the operator which occured.
(ii) Add the popped operator to array postfix
(iii) Add the scanned symbol operator into stack.
(6) (i) If symbol is right parenthesis ')' then pop all the operators from stack until left parenthesis '(' in stack.
(ii) Remove left parenthesis '(' from stack.
(7) If symbol is '#' then pop all the symbols from stack add them to array postfix except '#'.
(8) Do same process until '#' comes in scanning array infix

[top of stack will always be high priority operator

Infix

A*(B+C^D)-E^F*(G/H)

Initially '#' will be added in stack & at the end of infix Expression.

A*(B+C^D)-E^F*(G/H)#

Step	Symbol	Operator in stack	Postfix Expression
1	A	#	A
2	*	#*	A
3	(	#*(	A
4	B	#*(	AB
5	+	#*(+	AB
6	C	#*(+	ABC
7	^	#*(+^	ABC
8	D	#*(+^	ABCD
9	)	#*	ABCD^+
10	-	#-	ABCD^+*
11	E	#-	ABCD^+*E
12	^	#-^	ABCD^+*E
13	F	#-^	ABCD^+*EF
14	*	#-*	ABCD^+*EF^
15	(	#-*(	ABCD^+*EF^
16	G	#-*(	ABCD^+*EF^G
17	/	#-*(/	ABCD^+*EF^G
18	H	#-*(/	ABCD^+*EF^GH
19	)	#-*	ABCD^+*EF^GH/
20	#		ABCD^+*EF^GH/*-

Example

1. A+(B*C) convert into postfix using stack.

Intression	Stack	Output
A+(B*C)	NULL	—
+(B*C)	NULL	A
B*C)	+	A
*C)	+(	AB
C)	+C*	AB
)	+C*	ABC
—	+(	ABC*
—	+	ABC*
	NULL	ABC*+

postfix: ABC*+

2.) (7-5)*(9/2)

Intression	Stack	O/P
(7-5)*(9/2)	NULL	—
7-5)*(9/2)	(	—
-5)*(9/2)	(	7
5)*(9/2)	(-	7
)*(9/2)	(-	75
*(9/2)	(-	75-
(9/2)	NULL	75-
9/2)	*	75-
/2)	*(	75-9
2)	*(	75-92
)	*(	75-92/
—	—	75-92/*
—		

3) A+B*C-D/E

Σ	S	O/P
A+B*C-D/E	NULL	—
+B*C-D/E	NULL	A
B*C-D/E	+	A
*C-D/E	+	AB
C-D/E	+*	AB
-D/E	+*	ABC (^,*,%,+,-)
D/E	+*	ABC*+ (as * & + have high priority than stack will pop 1st *& + & insert -)
/E	-	ABC*+D
E	-	ABC*+D
—	-	ABC*+D/
—	NULL	ABC*+D/-

Q4 ((A+B)^C-(D*E)/F)

Σ	Stract	O/P
((A+B)^C-(D*E)/F)	NULL	—
(A+B)^C-(D*E)/F)	(	—
A+B)^C-(D*E)/F)	((	A
+B)^C-(D*E)/F)	((	A
B)^C-(D*E)/F)	((+	AB
)^C-(D*E)/F)	((	AB+ (bracket close)
^C-(D*E)/F)	((	AB+
C-(D*E)/F)	(^	AB+C
-(D*E)/F)	(^	AB+C^
(D*E)/F)	(-	AB+C^
D*E)/F)	(-(	AB+C^D
*E)/F)	(-(	AB+C^D
E)/F)	(-(*	AB+C^D

En	Stack	O/P
)/F)	C-C*	AB+C^DE
/F)	C-	AB+C^DE* (Backtrack)
F)	C-/	AB+C^DE*
)	C-/	AB+C^DE*F
end	C-	AB+C^DE*F/
—	C*	AB+C^DE*F/-
—	NULL	AB+C^DE*F/-

Q5. (A-B)*(D/E)

En	Stack	O/P
(A-B)*(D/E)	NULL	—
A-B)*(D/E)	(	—
-B)*(D/E)	(	A
B)*(D/E)	(-	AB
)*(D/E)	C-	AB-
*(D/E)	NULL	AB-
(D/E)	*	AB-
D/E)	*(	AB-D
/E)	*(	AB-D
E)	*/	AB-DE
)	*/	AB-DE/
—	~~NULL~~ *	AB-DE/*
—	NULL	AB-DE/*

Q6 (A+B/D) / (E-F)+G

Q7 A*B -(C+D) / (E-F)+G ₹/-.

Converting infix to Postfix using stack:

Note:

1. Stack contains only ~~op~~open parenthesis.
2. Stack contains only low priority operator
3. When get close parenthesis, we pop from the stack.
4. When a higher priority operator is in the stack & comes lower priority operator for push then higher priority operator should be pop & add to push fix expression.

Solved Examples

(1) $2+5*4/2-2\hat{}3*2$

$= 2+20/2-9*2$

$= 2+10-18$

$= 12-18$

$= -16$

Converting infix expression to postfix form without use of stack:

1. $A+B*C = A+BC* = ABC*+$

2. $A+[(B+C)+(D+E)*F]/G$

$= A+[(BC+)+(DE+)*F]/G$

$= A+[BC+ +DE+F*]/G$

$= A+[BC+DE+F*+]/G$

$= A+[BC+DE+F*+G/]$

$= ABC+DE+F*+G/+$

3.) $A+B-C = (A+B)-C = (AB+)-C$

$= AB+C-$

4.) $A*B+C/D = AB*+C/D$

$= AB*+CD/$

$= AB*CD/+$

6.) $A*B+C = AB*+C$

$= AB*C+$