* Stack

Works on LIFO: last in first out

ADT stack: It contain Data representation & operations on stack.

Data
1) Space for storing elements
2) Top pointer

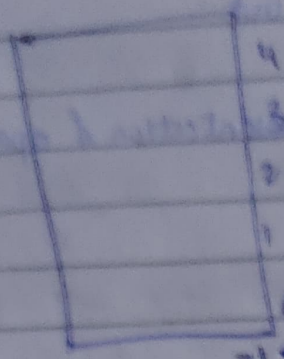
Operations:
1) push(x)
2) pop()
3) peek(index)
4) stackTop()
5) isEmpty()
6) isFull()

- Implementation of stack using Array

```
struct Stack
{
    int size
    int Top;
    int *S;
};
```

```
int main()
{
    struct stack st;
    printf("Enter size");
    scanf("%d", &st.size);
    st.S = new int[size];
    st.Top = -1;
```

Stack



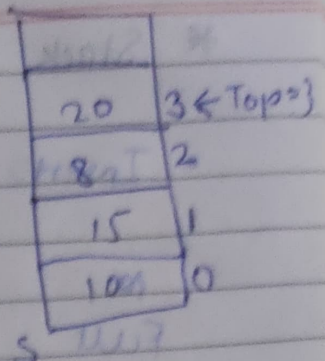
- push()

```
→ void push (stack *st, int x)
{
    if (st->Top = st->size-1)
        printf("Stack overflow");
    else
    {
        st->Top++;
        st->S[st->Top] = x;
    }
}
```

- pop()

```
→ int pop (stack *st)
{
    if (st->Top == -1)
        print ("Stack underflow");
    else
    {
        x = st->S[st->Top];
        st->Top--;
    }
    return x;
```

position	Index = Top - position + 1
1	3 = 3-1+1
2	2 = 3-2+1 = 2
3	1 = 3-3+1 = 1
4	0 = 3-4+1 = 0



- peek

```
int peek (stack st, int pos)
{
    if (Top - pos + 1 < 0)
        printf("Invalid position");
    else
        x = st.s[st.Top - pos + 1];
    return x;
}
```

→ No need of call by reference.

- Stack Top (stack *st)

```
if(St.Top == -1)
    return -1.
else
    return St.s[St.Top]
```

- ~~Stack~~ int isEmpty (stack st)

```
if (st.Top == -1)
    return 1;
else
    return 0;
```

- int isfull (Stack st)

```
if (St.Size - St.Top == -1)
    return
else
    return 0;
```

* Stack using linked list

- Insert from left side for using LL as stack

Full

→ Node *t = new Node;
if (t == Null)
    stack is full

→ void push(int x)
{
    Node *t = new Node;
    if (t == Null)
        print("Stack Overflow")
    else
    {
        t->data = x;
        t->next = top;
        top = t;
    }

- void pop()
{int x = -1
if (Top == Null)
    printf("Stack is Empty")
else
    p = Top;
    Top = Top->next;
    x = p->data;
    free(p);

```
    return x;
}
```

- ```
int Peek(int pos)
{
    int x=-1,i;
    Node *p=Top
    for (i=0, p!=Null && i<pos-1; i++)
        p=p->next;
    if(p!=Null)
        return p->data;
    else
        return -1;
}
```

## * Parenthesis matching -

- Check if there are equal parenthesis.
  - → if there is open parenthesis push it inside stack
  - → if there is closing parenthesis pop it outside of stack
  - → If there exist a open parenthesis at the end in stack its not matching or else its matching

**Program**

```
struct stack
{ int size;
  int Top;
  char *S;
}
```
```

```
int isBalance (char *exp)
{
    Struct Stack st;
    st.size = strlen(exp);
    st.Top = -1
    st.s = new char[st.size];

    // initialize stack
    for(i=0 ; exp[i]!='\0'; i++)
    {
        if (exp[i]='(') push (&st, exp[i]);
        else if (exp[i] == ')')
        { if (isEmpty (st)) return false;
          pop (&st);
        }
    } return isEmpty(st) ? true : false;
```

* Parenthesis with extended prolem different brackets

Infix to postfix conversion

1) Infix : operand operator Operand
   eg: a+b

2) Prefix: opt opnd opnd
   : +ab

3) postfix: opnd opnd opt (To perform all operation in just one scan)
   : ab+

Example

a+b*c

(a+(b*c))

Prefix: (a+[*bc])
(+a*bc)

Postfix: [a+[bc*]]
(abc*+)

Symbol	Precedence
+, -	1
*, /	2
( )	3

Associativity:

$a = b = c = 5$

R-L

$(a = (b = (c = 5)))$

Sym	Pre	Associativity
+, -	1	L-R
*, /	2	L-R
^	3	R-L
-	4	R-L
()	5	L-R

$a + b + c - d$

L-R

$((a+b)+c)-d)$

R-L $a \wedge b \wedge c$

$= (a \wedge (b \wedge c)) \rightarrow$ ~~$(a \wedge b) \wedge c$~~ $(a^{(b^c)})$

$-a$

Pre: $-a$ post $a-$

$(-(-a))$

-a + b * log n!

↓ postfix

-a + b * log[n!]

↓

-a + b * [n! log]

↓

[a -] + b * [n! log]

↓

[a -] [b n! log] * +

↓

a - b n! log * +

* Infix to postfix

Method 1

- If its an operator send it to stack
- If ~~operator~~ its an operand send it to a postfix string
- If operator has high precendence then push it into the stack or else pop it out of the stack into the postfix string.
- At end of expression empty the stack by pop it out of postfix string.

* Method 2

Sym	Pre	Asso
+, -	1	L-R
*, /	2	L-R
a, b, (	3	L-R

- If element of lower precedence is present in stack push element into the stack
  else pop every element of stack until there is no element at all ore the presendence of stack element is lower thant present element

- infix [a|+|b|*|c|-|d|/|e|\0]

```
char * convert(char * infix)
 {
    Struct stack  st;
    char * postfix= new char[strlen(infix)+1]
    int i=0,j=0;
    while (infix[i]!='\0')
    {   if(isOperand(infix[i]))
            postfix[j++]= infix[i++];
        else //its an operator
            { if(pre(infix[i])> pre(stackTop(
              push(&st,infix[i++]);
            else
              postfix[j++]=pop(&st)
    }

    while(!isEmpty(st))
        postfix[j++]=pop(&st)
    postfix[j]='\0';
```

return postfix;
}
}

* Challange of Infix to postfix using associativity and parenthesis

((a+b)*c)-d^e^f

[[ab+]*c]-d^e^f

[ab+c*]-d^e^f

[ab+c*]-d^[ef^]

[ab+c*]-[def^^]

[ab+c*def^^-]

Symbol	out stack Pre	instack Pre
+,-	1	2
*,/	3	4
^	6	5
(	7	0
)	0	9

* If precedence is more than stack element push it inside stack

* If precedence of element is less than stack element pop it outside element and add it to postfix string

* If precendes are equal pop element of stack & dont add it to postfix string

Its only for single digit number or we
we need to be able to distinguish
it is single or two digit

* Evaluation of postfix expression:

i) if operand push it inside stack.
ii) if ~~operand~~ operator pop out two element from stack and perform operation.
iii) push result back in stack
iv) End of expression result is in stack