

## \* Stack

Works on LIFO: last in first out

ADT Stack: It contains Data representation & operations on stack.

Data

- 1) Space for storing elements (qot + t2)
- 2) Top pointer

(Data)

Operations: (x + t1, t2 + 21018) data block t

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

size

## \* Implementation of stack using Array

```
struct Stack
```

```
{
```

```
int size
```

```
int top;
```

```
int *s;
```

```
};
```

```
int main()
```

```
{
```

```
(t2 + 21018) stack st; ~
```

```
printf("Enter size");
```

```
(1 - 2 = scanf("%d", &st.size);
```

```
st.s = new int[st.size];
```

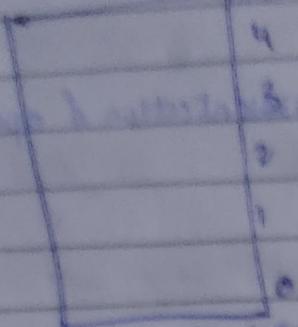
```
st.top = -1;
```

```
; [qot + t2] 2 + t2 = x
```

```
(-- qot + t2
```

```
; r neutral
```

Stack



$-1 \leftarrow \text{top}$  elements printed at stack[0]

### • push()

$\rightarrow$  void push (Stack \*st, int x)

{

if ( $st \rightarrow \text{Top} = st \rightarrow \text{size} - 1$ )  
 print ("stack overflow");

else

$st \rightarrow \text{Top} + 1;$

$st \rightarrow s[st \rightarrow \text{Top}] = x;$

cout << "push done";

}

Union tri / 2nd part true

### • pop()

$\rightarrow$  void <sup>int</sup> pop (Stack \*st)

( $st \rightarrow \text{size} = 0$ )

else if ( $st \rightarrow \text{Top} == -1$ )

cout <<

"stack underflow";

else =  $st \rightarrow \text{Top}$ ;

{

$x = st \rightarrow s[st \rightarrow \text{Top}];$

$st \rightarrow \text{Top} --;$

}

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

20	3 ← Top = 3
18	2
15	1
10	0

• peek

int peek (stack st, int pos) { when == + \* choice  
(when == - ) if

if (Top - pos + 1 < 0) { if i >= 2  
    printf("Invalid position");

else

x = st.s[st.Top - pos + 1]; now b/w

return x; symbol with = + \* symbol

}

(when == + ) if

("without → needed" of 1 call by reference)

• Stack Top (stack st)

if (St. Top == -1)

    return -1; x = null; ↑

else

    x = st.s[st.Top]; ↑

return st.s[st.Top]; ↑

{

• ~~Stack~~ int isEmpty (stack st)

if (st.Top == -1)

    return 1; ↑ now b/w

else

    return 0; ↑ now b/w

1 == r true

(\* printf "is empty" ) printing

• print isfull (stack st)

if (St.Size - St.Top == 1) == 0

    return 1; now b/w opt = qst

else

    return 0; now b/w qst

## \* Stack using linked list

- Insert from left side for using LL as stack

FULL

```
→ Node *t = new Node; // to make link
if (t == NULL)
    stack is full (overflow)
; ("overflow detected") message
```

→ void push(int x)

{

```
Node *t = new Node(x); // new node
```

```
if (t == NULL)
```

    overflow; printf("Stack overflow")

else

(top->next) = t; top = t;

{

    t->data = x; t->next =

    t->next = top;

    top = t; t->next =

}

(+) wrote p190721 in chapter 2

(t == NULL) ??

• void pop()

{ int x = -1

if (TOP == NULL) // empty

    printf("Stack is Empty")

else

(13. hint2) 110721 t-1

    P = TOP; // p190721 = 9510721

    TOP = TOP->next; // p190721

    x = P->data;

    free(P); // p190721

return x; (9x3+ 200) available = 701

3

200 20012 turns

• int Peek (int pos) { node \*nolat = 952 + h  
{ }

int x = 150; i; t27node C9N 32:16

Node \* p = top

for (i=0, p!=NULL & i< pos+1; i++)  
(+1)p = p->next; i = 0; } else

if (p!=NULL)

{ qx, t2 & } else { n = p->data; } if

else { 'c' = [i] qx; } if qx

else noutor ((return qx[i]; i) } i }

}

(t28) qqq

{

\* Parenthesis matching - noutor {

• 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

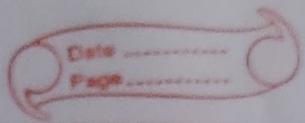
1 struct stack

{ int size;

int Top;

char \*s;

3



```
int isBalance (char *exp) {
```

Struct Stack st;

st.size = strlen(exp);

st.top = -1;

st.s = new char[st.size];

q = q + sizeof

(++i) initialize & stack = (st, q = 0) go

for (i=0; exp[i] != '0'; i++)

{

(i = 0) if

if (exp[i] == '(') push (&st, exp[i]);

else if (exp[i] == ')')

{ if (isEmpty(st)) return false;

pop (&st);

}

} return isEmpty(st) ? true : false;

→ reading input from user & output to file

\* Parenthesis with extended problems

di differentiating brackets with ci segnati fi e  
ci sottosignificati si segnati fi e  
ci sottosignificati si segnati fi e

→ to understand what is taking result of  
so problem ton sti sottosignificati si segnati  
sottosignificati si segnati

misprorg

sintesi trascrizione

→ sottosignificati si

ci segnati si

ci segnati

{}

## Infix to postfix conversion

1) Infix : operand operator Operand sides = D  
eg:  $a + b$

2) Prefix: opt opnd opnd opnd  
eg:  $+ab$

3) Postfix: opnd opnd opt (To perform all operations in just one scan)  
:  $ab+$

- Example

$a + b * c$

~~$(a + (b * c))$~~

prefix

postfix

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

$(a + [b * c])$

$(a + [bc * ])$

$(a * b) + c$

$(abc * +)$

$((D - ) - )$

Associativity: ~~when no extra info is given~~

$a = b = c = s \in \mathbb{Z}$  Sym Pro: Associativity  
 $+,-$   $\in \mathbb{Z} - R$

R-L

$(a = b = (c = s)))$	*	,	1	2	L-R
$b = a = c = s$	tq3	177	R-L		
$a + b + c - d$	-	dot4		R-L	
	( )	5	C-R		

L-R ~~no more than one~~ tqo bags bags : ~~not too~~ (a  
~~and~~ ~~one~~ ~~more~~ ~~than~~ ~~one~~) ~~add~~ :

$((ca+b)+c) = d$

~~(note)~~

~~probabilistic~~ ~~100% 2~~ ~~Ex sample~~

R-L  $a \wedge b \wedge c$   $\rightarrow$   $a \wedge (b \wedge c)$   $\rightarrow$  ~~(dot4)~~  $(a \wedge b) \wedge c$

$\rightarrow$  ~~add~~  $a \wedge (b \wedge c) \rightarrow$  ~~add~~  $(a \wedge b) \wedge c$

$-a$   
 Pre: -a Post:  $a -$   $(a \wedge b) \wedge c$   $(a \wedge b) \wedge c$

$(-(-a))$

$$-a + b * \log n$$

↓ postfix

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

↓

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

$$[a-] + \log * [n! \log]$$

↓

$$[a-] 0 + [b[n! \log] * ] / [a-] + \log ] \text{ x } n!$$

(x  $n!$  \* rands) + rands \* rands

$$a - b n! \log * +$$

it2 > it2 true2

$$0 = (10-9) \text{ true}$$

\* Infix + 0 (postfix x  $n!$ ) glidw

Method 2 (A19) bnoosq029)7i ?

- If it's an operator send it to stack

- If operator its 2nd operand send it to

postfix string

- If operator has high precedence 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.

((+2)pt0m72i!) glidw

(+2)q0g = It + ()x it + 0g

01 = [2]x it + 0g

## \* Method 2

- 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 whose precedence of stack element is lower than present element

$[a+b]*c+d$

infix  $a + b * c + d + (a + b) * c + d$

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])) {

    cout << infix[i];

    else // its an operator {

        if (pre(infix[i]) > pre(st.top)) {

            push(&st, infix[i]);

        else { cout << infix[i];

            pop(&st);

        } }

    } }

    while (!isEmpty(st))

        postfix[j++] = pop(&st);

    postfix[j] = '\0';

return postfix; // it is  $\rightarrow$  left to right \*

{ auto stack; for (int i = 0; i < infix.length(); i++) { char c = infix[i]; if (c == ')') { int j = stack.top(); stack.pop(); while (j != '(') { postfix.push\_back(j); j = stack.top(); stack.pop(); } stack.push(c); } }

\* challenge of Infix to Postfix conversion: associativity  
and parenthesis level. If we do like this, then it will be wrong.

$((a+b)*c)-d \wedge e \wedge f$	Symbol	out stock	in stock
	pre	pre	pre

$([ab]) * c) - d \wedge e \wedge f$       +,-      1      2

$[ab+tc*] - d \wedge e \wedge f$       \*,/      3      4

$[ab+tc*] - d \wedge [ef]$       ^      6      5

$[ab+tc*] - [def \wedge \wedge]$       (      7      0

$[ab+tc*] - [def \wedge \wedge]$

$[ab+tc*def \wedge \wedge -]$

\* 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 precedences are equal pop element of stack & dont add it to postfix string

It's only for single digit number  
it's not able to distinguish if it is single or two digit

### \* Evaluation of postfix expression

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

stack initial value  $3 \times 4 - ( ) + ( d b )$

3 4

S 1  $\times, +$   $3 \times 4 - ( ) + ( d b )$

P 8  $\times, +$

Z 2  $\times$   $1$   $3 \times 4 - [ * ] + ( d b )$

O F  $)$

P 0 C  $( i j ) \times b - [ * ] + ( d b )$

$[ ( i j ) \times b - [ * ] + ( d b ) ]$

$[ - i j \times b + ( d b ) ]$

if  $i < j$  swap  $i$  and  $j$

$i > j$  swap  $i$  and  $j$

and not in triangle performing  $f_i$

triangle stored in  $f_i$  of triangle  $f_{i+1}$   
part of  $f_{i+1}$  at  $f_i$  has been

triangle  $f_{i+1}$  formed by  $f_i$  and  $f_{i+1}$   $f_i$

using  $i+1$  block of  $f_{i+1}$  and  $f_i$

part of