



Stack

prepared by Harish Patnaik



Outline

- **Limitation of Linked list**
- **Intro to Stack**
- **Creation of Stack**
- **Stack representation using array**
- **Stack representation using Linked list**



Stack

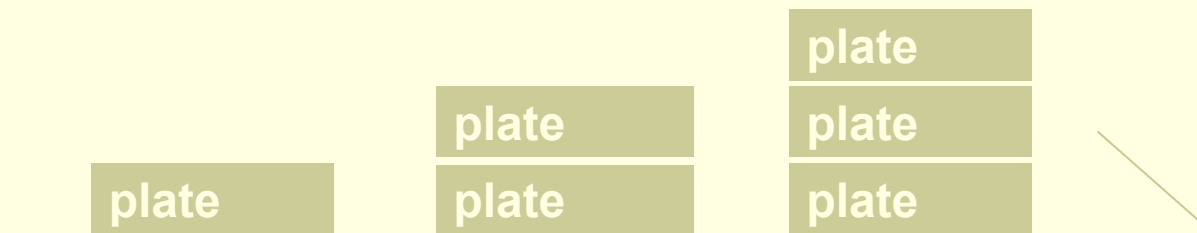
❑ Limitation with Linked list

- insertion & deletion can be done at any node
- not applicable for restricted insertion and deletion operation



Stack

- ✓ Items are added or removed only at one end
- ✓ Items are added or removed only from top of the stack
- ✓ Last In First Out (LIFO)





Stack

- ✓ It is a list in which elements are added or removed from Top of Stack (TOS)
- ✓ Every stack is associated with a pointer TOS
- ✓ Two basic operations of stack are -
 - push - insert an element into stack
 - pop - remove an element from stack



Basic operations of Stack

PUSH

S - stack

size - size of the stack

TOS - top of stack

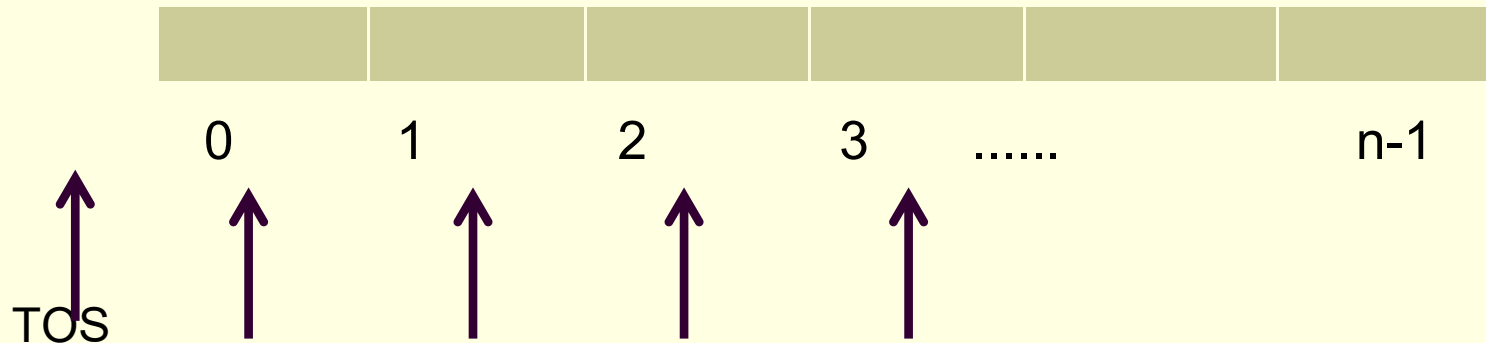
1. if $TOS \geq size$
 print "Stack overflow" and Exit
2. $TOS = TOS + 1$
3. $S[TOS] = value$ //inserting value
4. Exit

POP

1. if $TOS = 0$ //stack is empty
 print "Stack underflow" and Exit
2. $value = S[TOS]$
3. $TOS = TOS - 1$
4. Return value
5. Exit



Array implementation of Stack





```
#define size 50
int tos = -1;
int stack[size];

void push(int s[], int d)
{
    if (tos == size-1)
        printf ("\n Stack overflow ");
    else
        { ++tos ;
          s[tos]=d;
        }
}
```

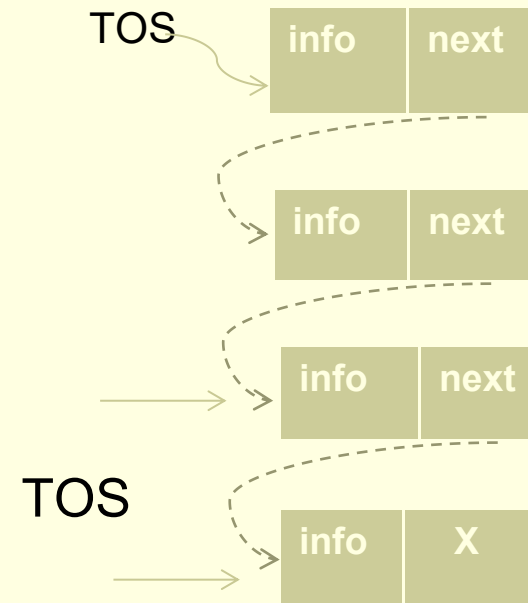
```
int pop(int s[])
{
    int d;
    if (tos == -1)
        { d = 0;
          printf ("\n Stack underflow "); }
    else
        { d = s[tos] ;
          -- tos ; }
    return (d);
}

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




Linked List implementation of Stack

```
struct node
{
    int info;
    struct node * next;
}
struct node *top;
top= NULL;
```





```
void push(struct node *top)
{
    struct node *nnode;
    nnode= (struct node *)
            malloc(sizeof(struct node));
    if (nnode== NULL)
        {printf ("\n Out of memory ");
         exit(0);
        }
    printf ("\nEnter the node value ");
    scanf ("%d",nnode->info);
    nnode->next=top;
    top =nnode;
}
```

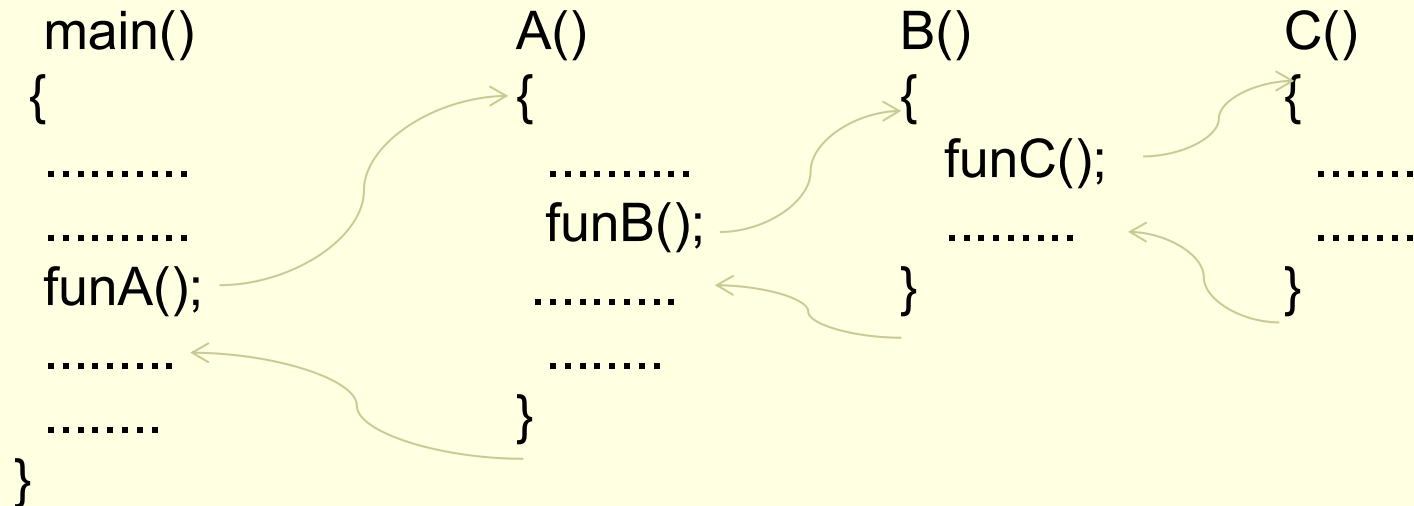
```
int pop(struct node *top)
{
    struct node *temp;    int d;
    if (top==NULL)
        { printf ("\n Stack underflow ");
          exit(0); }
    temp =top;
    d = temp->info;
    top= top->next ;
    free(temp) ;
    return (d);
}

void display(struct node *top)
{    struct node *curr;
    curr= top;
    if (curr== NULL)
        { printf ("stack is empty"); exit(0); }
    while (curr !=NULL)
        { printf ("\t %d", curr->info);
          curr= curr->next;    }
}
```



Application of Stack

❖ Function call



Returning
addr. of A

Returning
addr. of B

Returning
addr. of C

R A of C

R A of B

R A of A

calling → 1
returning → 3

2
2

3
1



Application of Stack

❖ Arithmetic Expression evaluation

Three notations for writing expression

Infix - operator is between two operands

postfix (Reverse Polish)- operator follows the operands

prefix (Polish) - operator precedes the two operands

Infix - $a+b$

postfix- $ab+$

prefix- $+ab$

To evaluate an expression, the infix expression need to be converted to postfix expression



Application of Stack

- Conversion of infix to postfix

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

$=a+ (bc*-d)$

$=a+ (bc*d-)$

$=a(bc*d-)+$

$=abc*d-+$ postfix

- Precedence of operators is considered
- Expression is evaluated by scanning from left to right using Stack



Postfix expression evaluation

1. clear the stack
2. sym= input next character
3. while sym \neq null do {
4. if sym is an operand
 Push sym
5. else {
6. Pop 2nd operand OP2
7. Pop 1st operand OP1
8. result= op1 sym op2
9. Push result
10. }
11. sym= input next character
12. }
13. return (Pop stack)



Expression evaluation

$(5+6) - \{3 * (8/2)\}$

56+382/*-

Symbol	OP1	OP2	result	Stack
5				5
6				5, 6
+	5	6	11	11
3				11, 3
8				11,3,8
2				11,3,8,2
/	8	2	4	11,3,4
*	3	4	12	11,12
-	11	12	-1	-1



Application of Stack

❖ Infix to Postfix conversion

Let Q be an arithmetic expression in infix notation. This algorithm find the equivalent postfix expression P.

1. Push '(' onto stack and add ')' to the end of Q
2. Scan Q from left to right and repeat steps 3 to 6 for each element of Q until the stack is empty
3. If an operand is found, add it to P
4. if a '(' is found, push it onto stack.
5. If an operator X is found then
 - a) Repeatedly pop operators from stack and add to P each operator which has same or higher precedence than X
 - b) Add X to stack
6. If a ')' is found then
 - a) Repeatedly pop operators from stack and add to P each operator until a '(' is found
 - b) Remove the '(' from stack
7. Exit



Infix to Postfix conversion

(6*12)+(24/4))

Symbol	Stack	Postfix expression
((
6	((6
*	((*	
12		6,12
)	(6,12, *
+	(+	
((+ (
24		6,12, *, 24
/	(+ (/	
4		6,12, *, 24, 4
)	(+	6,12, *, 24, 4, /
)		6,12, *, 24, 4, /, +



Postfix expression evaluation

6, 12, *, 24, 4, /, +

Symbol	OP1	OP2	result	Stack
6				6
12				6, 12
*	6	12	72	72
24				72, 24
4				72, 24, 4
/	24	4	6	72, 6
+	72	6	78	78



Prefix expression evaluation

1. clear the stack
2. sym= input next character (from right to left)
3. while sym \neq null do {
4. if sym is an operand
 Push sym
- else {
5. Pop 1st operand OP1
6. Pop 2nd operand OP2
7. result= op1 sym op2
8. Push result
- }
9. sym= input next character
- }
10. return (Pop stack)



Prefix expression evaluation

-,*,3,+,16,2,/,12,6

Symbol	OP1	OP2	result	Stack
6				6
12				6, 12
/	12	6	2	2
2				2,2
16				2,2,16
+	16	2	18	2,18
3				2,18,3
*	3	18	54	2,54
-	54	2	52	52



Application of Stack

❖ Infix to Prefix conversion

Let Q be an arithmetic expression in infix notation. This algorithm finds the equivalent prefix expression P.

1. Push ')' onto stack and add '(' to the end of Q
2. Scan Q from right to left and repeat steps 3 to 6 for each element of Q until the stack is empty
3. If an operand is found, add it to P
4. if a ')' is found, push it onto stack.
5. If an operator X is found then
 - a) Repeatedly pop operators from stack and add to P each operator which has same or higher precedence than X
 - b) Add X to stack
6. If a '(' is found then
 - a) Repeatedly pop operators from stack and add to P each operator until a ')' is found
 - b) Remove the ')' from stack
7. Exit



Infix to Prefix conversion

((6*12)+(24/4)

Symbol	Stack (s1)	Prefix expression
)	
)))	
4		4
/)) /	
24		4,24
()	4,24, /
+) +	
)) +)	
12		4,24, /,12
*) +) *	
6		4,24, /,12,6
() +	4,24, /,12,6, *
(4,24, /,12,6, *, +



Thank you