UNIT II

STACKS AND QUEUES

STACK ADT – Array and Linked List implementation of stack operations – Balancing Symbols – Infix to postfix Conversion – Evaluating arithmetic expressions – Function calls – Queue ADT – Array and Linked List Implementation of Queue operations – Circular Queue – Applications of Stack and Queues in operating systems

STACK ADT

* It is a linear data structure that follows Last In First out (LIFO) / First In Last out (FILO) principle.
* Insertion and deletion operation is done at only one end named "Top" of the stack.

Ex: pile of coins, stack of trays in cafe.

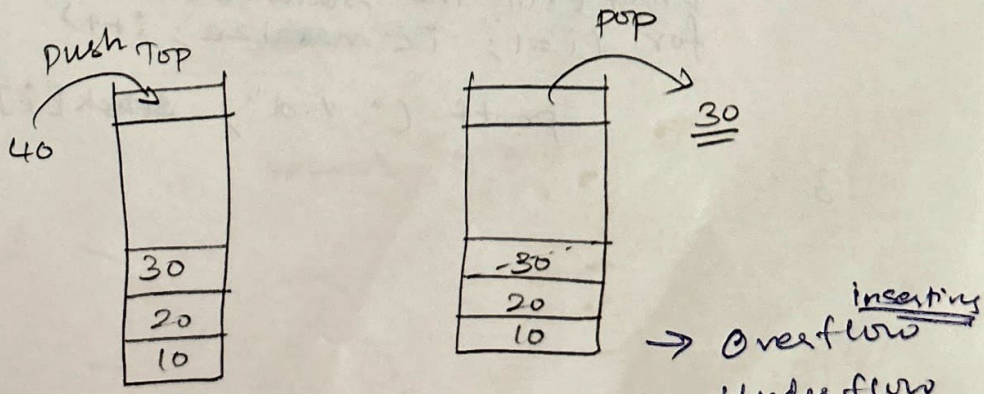
Stack operations.

2 fundamental operations are:

* push – insert a element – top ++
* pop – delete a element. – top --

push operation. (Inserting a element into stack)

- Inserting an element at top of the stack
- Increment the top by 1.



pop operation – delete an element
– decrement top by 1

ARRAY IMPLEMENTATION OF STACK

Routine to push an element to Stack.

```
int maxsize = 8;
int Stack [8];
int top = -1;
int push (int data)
{
    if ( ! isfull ())
    {
        top = top +1;
        Stack [ top ] = data;
    }
    else
    {
        printf ("\n Couldn't insert data ");
    }
}
int isfull ()
{
    if (top == maxsize)
        return 1;
    else
        return 0;
}
int main ()
{
    int i;
    push (10);
    push (20);
    printf ("\n The stack elements are: ");
    for (i=1; i< maxsize; i++)
        printf (" %d", stack[i]);
}
```

→ When we insert an element when stack is full (overflow)

Routine to pop an element from stack.

```
int pop()
{
    if (!isempty())
    {
        top = top - 1;
        item = stack[top];
    }
    else
    {
        printf("\n Underflow"); //No elements in stack
    }
}
int isempty()
{
    if (top == -1)
        return 1;
    else
        return 0;
}
```

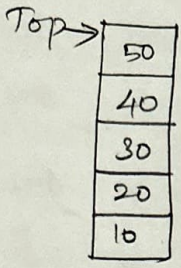
attempt to insert an element when stack is empty - "Underflow"

Peek operation [Visiting each element of the stack].

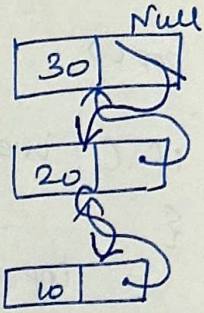
- returns the element which is @ top of the stack.

```
int peek()
{
    if (top == -1)
        printf("Underflow");
        return 0;
    else
        return stack[top];
}
```

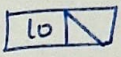
LINKED LIST IMPLEMENTATION OF STACK.

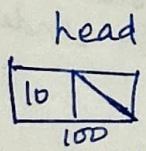


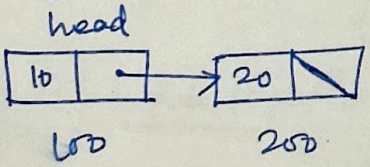
Top = -1



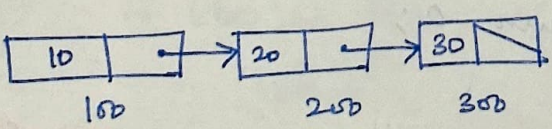
* Linked list allocates memory dynamically.
* Top most nodes holds Null in the address field (if it is the first node → only one node)







Inserting at tail in linked list → O(n) we have to traverse till last node and then insert/delete.

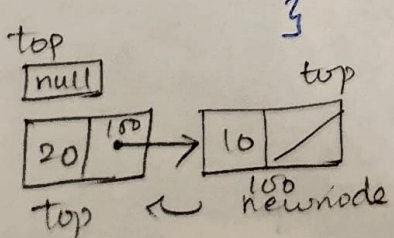


* So, when we insert a node at head position than it takes O(1) time Complexity.

* push & pop operation.

push operation. (Insertion at begin)

```
Void push (int x)
{
    Struct node *newnode = malloc (sizeof (Struct n
    newnode -> data = x;
    newnode -> link = top;
    top = newnode;
}
```

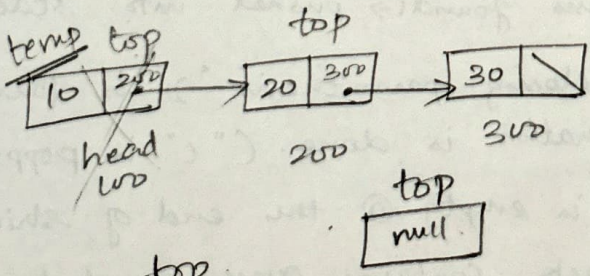


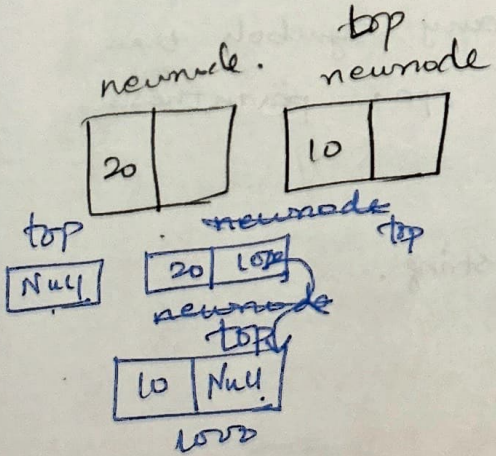
To create a node for stack:

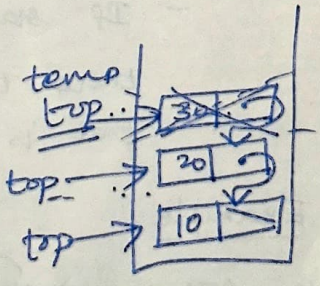
```
Struct node
{
    int data;
    Struct node *ptr;
};
Struct node *top = NOLL;
```

pop operation. (Deletion at beginning)

```
Void pop()
{
    Struct node *temp;
    if (top == NOLL)
        return;
    temp = top;
    top = top -> ptr;
    free (temp);
}
```







Applications of stack. (ToH, 8 Queens Problem, Reverse the string)

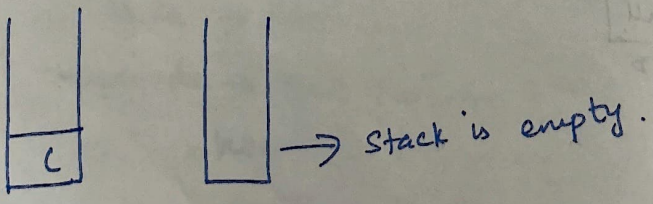
1. Balancing Symbols
2. Infix to Postfix Conversion.
3. Evaluating arithmetic Expression.
4. Function Calls.

Balancing Symbols.

* The common mistake, which programmers frequently make is Unbalance of paranthesis.
* To make a Correct representation
  a. There must be equal no. of left and right paranthesis
  b. Each left paranthesis must be balanced right paranthesis
* Stack is used in balancing Symbol.
  - paranthesis found (→ pushed into stack
  - When closing paranthesis ")" is found pop operation is done. ("(" is popped)
  - Stack is empty @ the end of string
  - If stack contains any symbols than there will be any open paranthesis that is not closed.

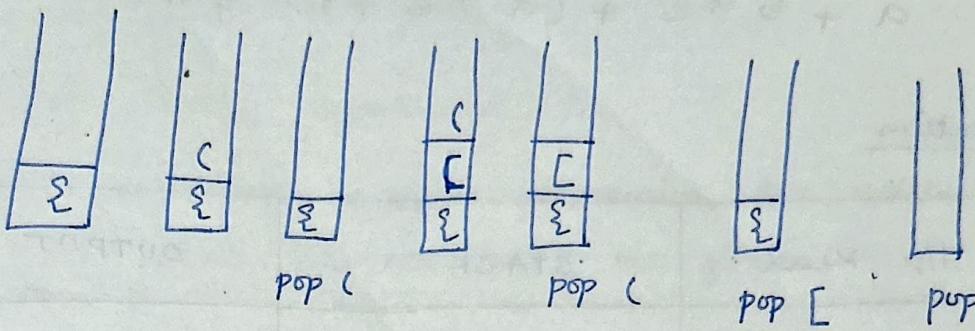
Example :

① $(x + y)\#$ → end of the string.

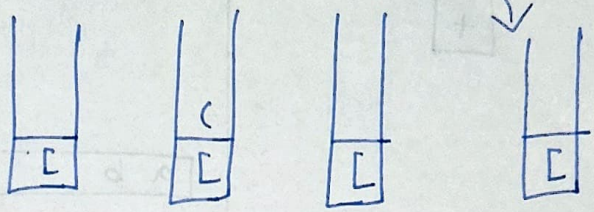


push (When reading) pop (

② {(x+y) − [(x+y+z)]} #



③ [(x+y) #



- reached end of string
- Stack is not empty
- It is Unbalanced.

pop)

Evaluating arithmetic Expression.

Steps for evaluating arithmetic expression are as follows:

① Convert Infix expression to postfix expression

② Evaluate postfix expression with stack.

Infix to postfix expression.

Steps:

1. Read the infix expression one character at a time until reaching the delimiter #
2. If character – operand → place it in output
3. If character – operator → push it into stack.
   Again if a operator → operator inside stack has high priority, pop that operator and place it in output.
4. If character is "(" (left paranthesis) – push into stack.
5. If character is ")" – pop all the operators from stack till it finds "(". Discard both paranthesis in output.

Example 2:

a + b * c + (d * e)

1 2 3 4 2

(i) Infix to postfix expression Conversion

I/p Read	STACK	OUTPUT
a		a
+	+	a
b	+	a b
*	* +	a b
c	* +	a b c
+	+	a b c * +
(	(+	a b c * +
d * e	* (+	a b c * + d
)		a b c * + d e

Evaluating postfix expression.

a b c * + d e * +
1 2 3 4 2

Solution:

Read A, B, C

Read *

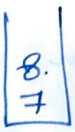
2 * 3 = 6.

Read +

1 + 6 = 7

Read d, e

Read *



4 * 2 = 8

Read +

7 + 8 = 15

Answer: 15

Example 3: A * B + (C C - D / E) #

postfix: A B * C D E / - +
4 5 5 8 2

Answer: 21

Function Calls.

Function that calls itself again is said to be a recursive function.

When a function call is made:

* The arguments, return address is saved using stack. [This information saved is called Stack frame]

* When a function is called → push the stack frame into stack. When return statement is reached, stack frame is popped from the stack.

Example

```
int fact(int n)
{
    if (n == 1)
        return 1;
    else
        return n * fact(n-1);
}
```