DATA STRUCTURE

Data structure is a collection of data element, whose arrangement is characterized by accessing function , that are used to store and retrieve the individual data element.

TYPES:
1. Linear data structures
2. Non linear data structures

LINEAR DATA STRUCTURES
1. Arrays
2. Linked list
3. Stacks
4. Queues

NON LINEAR DATA STRUCTURES
1. Trees
2. Graphs

An *abstract data type* (ADT) is a set of operations
(mathematical abstractions)

LIST ADT

- List is an ordered set of elements.
- The general form of the list is

$$A_1, A_2, A_3, \ldots, A_N$$

- A_1 - First element of the list
- A_N - Last element of the list
- N - Size of the list
- If the element at position i is A_i then its successor is A_{i+1} and its predecessor is A_{i-1}.

VARIOUS OPERATIONS PERFORMED ON LIST:

1. **Insert (X, 2)** - Insert the element X after the position 2.
2. **Delete (X)** – This will delete the element X.
3. **Find (X)** - Returns the position of X.
4. **Next (i)** - Returns the position of its successor element i+1.
5. **Previous (i)** - Returns the position of its predecessor i-1.
6. **Print list** - Contents of the list is displayed.
7. **Makeempty** - Makes the list empty.

www.ashairun.blogspot.com

Implementation of List ADT

1. Array 2. Linked List 3. Cursor

- **Array** is a collection of similar data item under a common name, stored in a consecutive memory locations.
- **Linked list** consist of serious of nodes. It contains the element and a pointer used to store next node address.

Types of Linked list

– Singly linked list **(Each node has two fields namely data field, and Address field)**

– Doubly linked list **(Each node has three fields namely data field, forward link(FLINK) and backward link(BLINK))**

– Circular linked list **(the pointer of the last node points to the first node)**

www.asharish.blogspot.com

```
struct node
{
element_type element;
node_ptr next;
};
typedef struct node *node_ptr;
typedef node_ptr list;
typedef node_ptr position;
int is_empty( list l );
int is_last( position p, list l );
void insert(element_type x,list l,position p);
void delete( element_type x, list l);
position find(element_type x, list l ) ;
position find_previous( element_type x, list l );
void print_list(list l);

int is_empty( list l )
{
return( l->next == null );
}

int is_last( position p, list l )
{
return( p->next == null );
}

void insert(element_type x,list
l,position p)
{
position tmp_cell;
tmp_cell = new node_ptr;
if( tmp_cell == null )
fatal_error("out of space!!!");
else
{
tmp_cell->element = x;
tmp_cell->next = p->next;
p->next = tmp_cell;
}
}

void delete( element_type x, list l )
{
position p, tmp_cell;
p = find_previous( x, l );
if( p->next != null )
{
tmp_cell = p->next;
p->next = tmp_cell->next;
Delete tmp_cell;
}
}
```

```
position find(element_type x, list l )
{
position p;
p = l->next;
while( (p != null) && (p->element != x) )
p = p->next;
return p;
}
position find_previous( element_type x, list l )
{
position p;
p = l;
while( (p->next != null) && (p->next ->element != x)
)
p = p->next;
return p;
}
```

Applications of linked list

1. Polynomial ADT
2. Radix Sort
3. Multi set

CURSOR IMPLEMENTATION OF LIST

Creating The Linked List Without The Use Of Pointers.
Memory Allocation – Cursoralloc()
Memory De-allocation – Cursorfree()

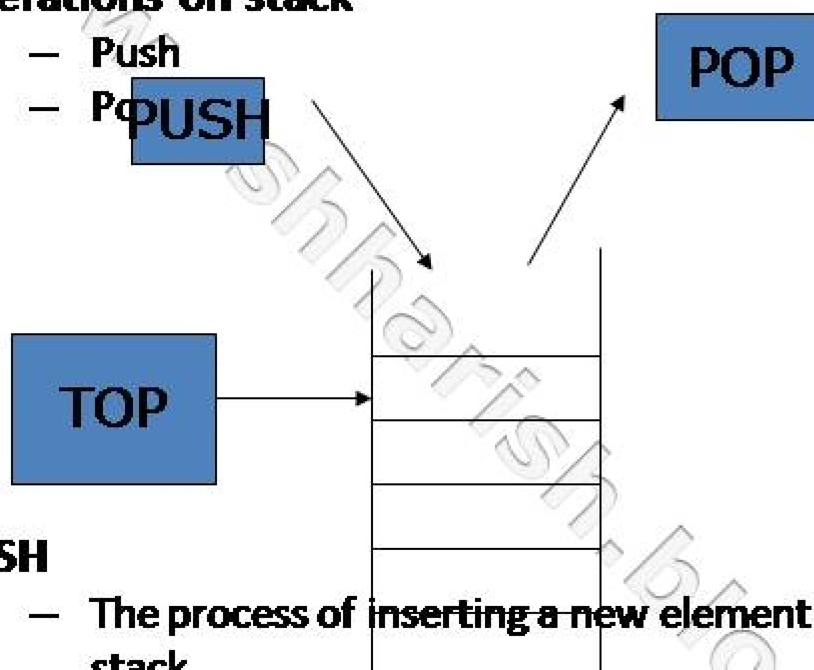
www.asmharish.blog

STACK ADT

- It is a linear data structure which follows **Last In First Out** (**LIFO**) principle
- Both insertion and deletion occur at only one end of the list called the **TOP**

Operations on stack

- Push
- Pop



PUSH

- The process of inserting a new element to the top of the stack
- For every push operation the top is incremented by 1

POP

- The process of deleting an element from the top of stack is called pop operation
- After every pop operation the top pointer is decremented by 1

Exceptional conditions

- Overflow – attempt to insert an element when the stack is full
- Underflow – attempt to delete an element, when the stack is empty

ARRAY IMPLEMENTATION OF STACK

Routine to push an element into a stack (Top=-1 , arraysize =10)

```
void push(int x, stack s)
{
    if(isfull(s))
        error("full stack");
    else
    {
        top = top + 1;
        s[top] = x;
    }
}
```

```
int isfull(stack s)
{
    if(top == arraysize-1)
        return(1);
}
```

Routine to pop an element from a stack

```
void pop (stack s)
{
        if(isempty(s))
        error("empty stack");
        else
        {
                x = s[top];
                top = top – 1;
        }
}
```

```
int isempty(stack s)
{
        if(top == -1)
                return(1);
}
```

www.annaharish.blogspot.com

Linked List Implementation of Stack

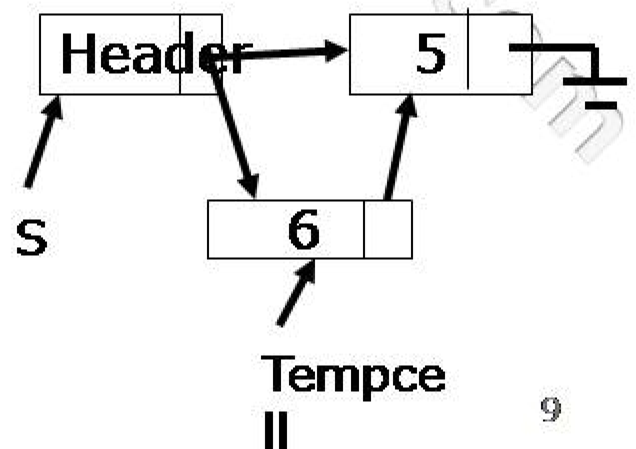
```
Struct Node;
typedef Struct Node *PtrToNode;
typedef PtrToNode Stack;
int IsEmpty (Stack S);
Stack CreateStack ( );
void push (int X, Stack S);
void pop (Stack S);
Struct Node
{
    int Element ;
    PtrToNode Next;
};
```

www.rasikarish.blogspot.com

ROUTINE TO PUSH AN ELEMENT ONTO A STACK

www.annarish.blogspot.com

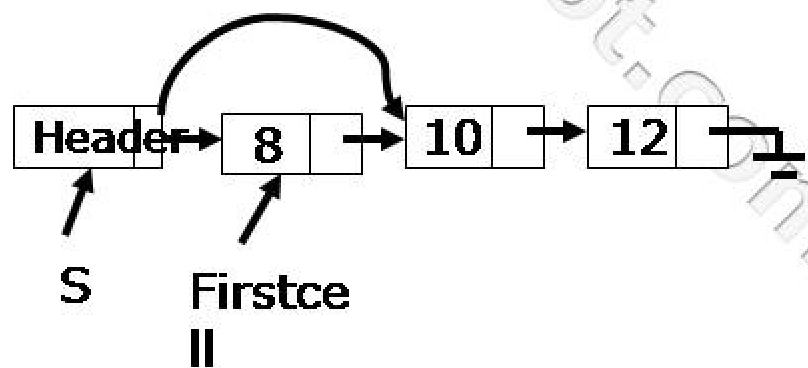
```
void push (int X, Stack S)
{
PtrToNode Tempcell;
Tempcell = malloc (sizeof (struct Node));
    If (Tempcell = = NULL)
        Error ("Out of Space");
    else
    {
        Tempcell→Element = X;
        Tempcell→Next = S→ Next;
        S→ Next = Tempcell;
    }
}
```



ROUTINE TO POP FROM A STACK

```
void pop (Stack S)
{
PtrToNode Firstcell;
    If (IsEmpty (S))
        Error ("Empty Stack");
    else
    {
        Firstcell = S→ Next;
        S→ Next = S→ Next→ Next;
        Free (Firstcell);
    }
}
```

```
int IsEmpty (Stack
S)
{
return(S→Next==
NULL);
}
```



www.acharish.blogspot.com

APPLICATIONS OF STACK:

- Evaluating arithmetic expression
- Balancing the symbols
- Function calls

Evaluating arithmetic expression

Types of notations

- Infix notation
- Postfix notation
- Prefix notation

INFIX

- The arithmetic operator appears between the two operands to which it is being applied
- EX: A+B

POSTFIX

- The arithmetic operator appears directly after the two operands to which it applies
- It is also called reverse polish notation. (A+B)
- EX: AB+

PREFIX

- The arithmetic operator is placed before the two operands to which it applies
- It is also called polish notation (A+B)
- EX: +AB

www.eshiarish.blogspot.com

INFIX TO POSTFIX CONVERSION

- Read the infix expression one character at a time until it encounters the delimiter "#"
  1. If the character is an operand, place it on to the output
  2. If the character is an operator, push it onto the stack. If the stack operator has a higher or equal priority than input operator then pop that operator from the stack and place it onto the output
  3. If the character is a left parenthesis, push it onto the stack
  4. If the character is a right parenthesis, pop all the operators from the stack till it encounters left parenthesis, discard both the parenthesis in the output

INFIX EXPRESSION :

1. (A+B)#
2. (A + B) / (C – D)#
3. (A * B) + (C – (D / E))#

POSTFIX EXPRESSION:

1. AB+
2. A B + C D - /
3. A B * C D E / - +

www.ashwarian.blogspot.com

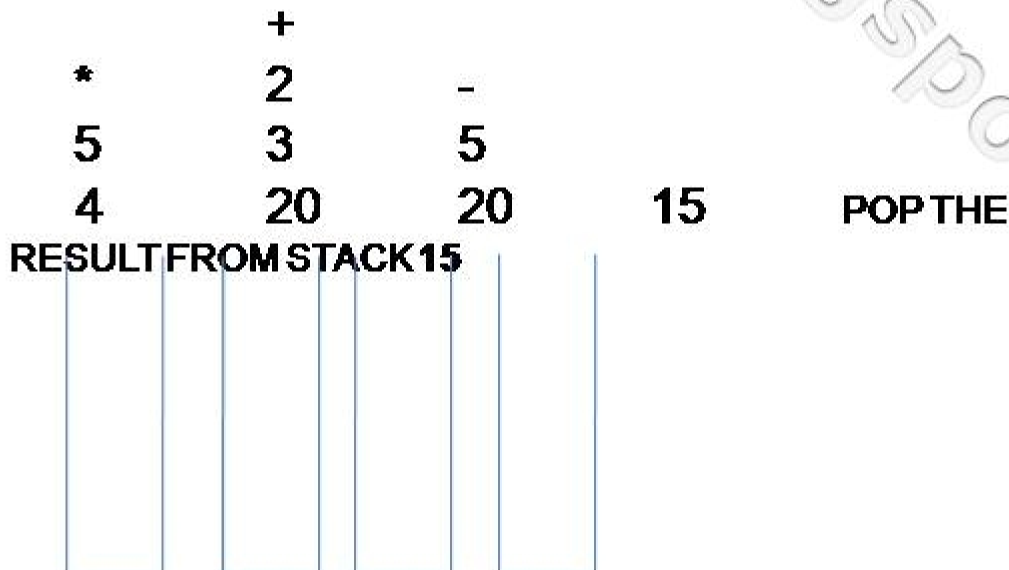
Evaluating postfix expression

- Read the postfix expression one character at a time until it encounters the delimiter "#"

1. If the character is an operand, push its associated value onto the stack
2. If the character is an operator, POP two values from the stack, apply the operator to them and push the result onto the stack

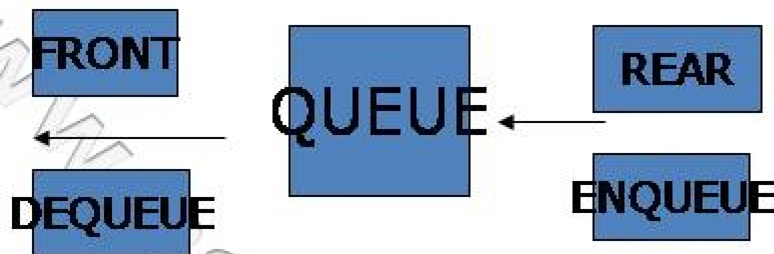
EXAMPLE: (A*B)-(C+D) = **AB*CD+-**

EX, **45*32+-#**



QUEUE

- It is a linear data structure which follows First In First Out (FIFO) principle, in which insertion is performed at rear end and deletion is performed at front end



OPERATION ON QUEUE:

- Enqueue
- Dequeue

ENQUEUE

- The process of inserting an element in the rear end of queue

DEQUEUE

- The process of deleting an element from the front end of queue

Exception conditions

- **Overflow**
  - Attempt to insert an element, when the queue is full is said to overflow condition
- **Underflow**
  - Attempt to delete an element from the queue, when the queue is empty is said to be underflow

www.sliharish.blogspot.com

Array implementation of queue

- To insert an element X on to the queue Q, the rear is incremented by 1
- To delete an element, the queue [front] is returned and the front pointer is incremented by 1
- Initially front =0 and rear end is equal to -1

ROUTINE OF ENQUEUE AND DEQUEUE

```
void enqueue(int x)
{
    if(rear>arraysize-1)
        cout<<"queue overflow";
    else
    {
        rear = rear + 1;
        queue[rear]=x;
    }
}
```

```
void dequeue()
{
if(front<0)
cout<<"queue
underflow";
else
{
x = queue[front];
if(front==rear)
{
        front = 0;
        rear = -1;
}
else
        front = front +
1;
} }
```

```
front=rear=NULL;
```

ROUTINE TO ENQUEUE

```
void enqueue(int x, queue Q)
{
struct node *new1;
new1 =new node;
new1 ->data=x;
new1 ->next=NULL;
if(front==NULL)
front=rear= new1;
else
rear->next= new1;
rear= new1;
}
```

www.rejinpaul.com

APPLICATIONS OF QUEUE

- Batch processing in an operating system
- To implement priority queue

ROUTINE TO DEQUEUE

```
void dequeue(queue Q)
{
if(front==NULL)
{
 cout<<"\nQueue is empty!!";
}
 struct node *temp;
 temp=front;
front=front->next;
delete temp;
}
```

www.ashhariah.blogspot.com