Data Structures

It is a way of representing or organizing all data that contains both the data items and their relationships with each other.

To create efficient algorithms,efficient data structures are needed

Types of DS

Linear DS

It contains a linear arrangement of elements in the memory(Eg.STACK,LIST,QUEUE).

Non Linear DS

A DS represents a hierarchical arrangements of elements(Eg.TREE,GRAPH)

Application of DS

OS
Compiler design
Expert System
Network Analysis
DBMS
Statistical and numerical analysis

ABSTRACT DATA TYPE

It is a set of operations

ADT's are mathematical abstractions.(It does not specify how the set of operations is implemented.

It is an extension of **modular design.**

Eg.Objects such as list,sets and graphs, along with their operations

SET ADT

operations (union,intersection,complement,find,insert,delete etc.)

Why ADT?

Implementation of these operations is written once in the program and any other part of the program that need to perform an operation on the ADT can do so by calling the appropriate Function.

Any change if necessary then it can be done at the particular function.

MODULE

In programming each program is breakdown into modules, so that no routine should ever exceed a page. Each module is a logical unit and does a specific job.Size of the module is kept small by calling other modules.

Modularity has several advantages

1. Modules can be compiled separately which makes debugging process easier.
2. Several modules can be implemented and executed simultaneously.
3. Modules can be easily enhanced.

THE LIST ADT

List is an ordered set of elements.

The general form of the list is

$A_1, A_2, A_3, ... ,A_N$

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, 5) - Insert the element X after the position 5.
2. Delete (X) - The element X is deleted
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.

Implementation of List ADT

1. Array Implementation
2. Linked List Implementation
3. Cursor Implementation.

Array Implementation of List

Array is a collection of specific number of data stored in a consecutive memory locations.

Disadvantages of Array implementation

* Insertion and Deletion operation are expensive as it requires more data movement
* Find and Printlist operations takes constant time.
* Even if the array is dynamically allocated, an estimate of the maximum size of the list is required which considerably wastes the memory space.
*Wastage of Memory space.

Linked List Implementation

Linked list consists of series of structures(node) which are not necessarily adjacent in memory.Each node(structure) contains the element and a pointer to its successor node. The pointer of the last node points to NULL.

Insertion and deletion operations are easily performed using linked list.

Data representation :

-

Data in structure representation:

```
Struct node
{
int data
Struct data *link;
}
```

-

Types of Linked List

1. **Singly Linked List**
   A singly linked list is a linked list in which each node contains only one link field pointing to the next node in the list.

-

DECLARATION FOR LINKED LIST

```
Struct node ;
typedef struct Node *List ;
typedef struct Node *Position ;
int IsLast (List L) ;
int IsEmpty (List L) ;
position Find(int X, List L) ;
void Delete(int X, List L) ;
position FindPrevious(int X, List L) ;
position FindNext(int X, List L) ;
void Insert(int X, List L, Position P) ;
void DeleteList(List L) ;
Struct Node
{
int element ;
position Next ;
};
```

ROUTINE TO INSERT AN ELEMENT IN THE LIST

INSERT (25, P, L)

```
void Insert (int X, List L, Position P)
/* Insert after the position P*/
{
position Newnode;
Newnode = malloc (size of (Struct Node));
If (Newnode! = NULL)
{
Newnode ->Element = X;
Newnode ->Next = P->    Next;
P->    Next = Newnode;
}
}
```

ROUTINE TO CHECK WHETHER THE LIST IS EMPTY

```
int IsEmpty (List L) /*Returns 1 if L is empty */
{
if (L ->    Next = = NULL)
return (1);
}
```

ROUTINE TO CHECK WHETHER THE CURRENT POSITION IS LAST

```
int IsLast (position P, List L) /* Returns 1 is P is the last position in L */
{
if (P->Next = = NULL)
return
}
```

FIND ROUTINE

```
position Find (int X, List L)
{
/*Returns the position of X in L; NULL if X is not found */
position P;
P = L->    Next;
while (P! = NULL && P    Element ! = X)
P = P->Next;
return P;
}
}
```

FIND PREVIOUS ROUTINE

```
position FindPrevious (int X, List L)
{
/* Returns the position of the predecessor */
position P;
P = L;
while (P ->    Next ! = Null && P ->Next     Element ! = X)
P = P ->Next;
return P;
}
```

FINDNEXT ROUTINE

```
position FindNext (int X, List L)
{
/*Returns the position of its successor */
P = L ->Next;
while (P    Next! = NULL && P    Element ! = X)
P = P→Next;
return P→Next;
}
```

ROUTINE TO DELETE AN ELEMENT FROM THE LIST

```
void Delete(int X, List L)
{
/* Delete the first occurence of X from the List */
position P, Temp;
P = Findprevious (X,L);
If (!IsLast(P,L))
{
Temp = P→Next;
P →Next = Temp→Next;
Free (Temp);
} }
```

ROUTINE TO DELETE THE LIST

```
void DeleteList (List L)
{
position P, Temp;
P = L →Next;
L→Next = NULL;
```

```
while (P! = NULL)
{
Temp = P→Next
free (P);
P = Temp;
}
}
```

Advantages of Singly Linked List.

It is dynamic memory allocation so that there is no wastage of memory.(memory allocated during run time.)

Simple procedures are used to insert ,delete and to search the element in the list.

Disadvantages of Singly linked list

Extra space is required to store the address of the successor element.

Doubly Linked List

A Doubly linked list is a linked list in which each node has three fields namely data field, forward link (FLINK) and Backward Link (BLINK). FLINK points to the successor node in the list whereas BLINK points to the predecessor node.

STRUCTURE DECLARATION : -

```
Struct Node
{
int Element;
Struct Node *FLINK;
Struct Node *BLINK
};
```

ROUTINE TO INSERT AN ELEMENT IN A DOUBLY LINKED LIST

```
void Insert (int X, list L, position P)
{
Struct Node * Newnode;
Newnode = malloc (size of (Struct Node));
If (Newnode ! = NULL)
```

```
{
Newnode →Element = X;
Newnode →Flink = P     Flink;
P →Flink →Blink = Newnode;
P →Flink = Newnode ;
Newnode →Blink = P;
}
}
```

ROUTINE TO DELETE AN ELEMENT

```
void Delete (int X, List L)
{
position P;
P = Find (X, L);
If ( IsLast (P, L))
{
Temp = P;
P →Blink →Flink = NULL;
free (Temp);
}
else
{
Temp = P;
P →Blink→     Flink = P→Flink;
P →Flink →Blink = P→Blink;
free (Temp);
}
}
```

Advantage

* Deletion operation is easier.
* Finding the predecessor & Successor of a node is easier.

Disadvantage

* More Memory Space is required since it has two pointers.

2. **Circular Linked List.**

   In circular linked list the pointer of the last node points to the first node.

   Circular linked list can be **implemented** as

   Singly linked list and

   Doubly linked list with or without headers.

Difference between array and LL

Array	Linked List
Static memory allocation	Dynamic Memory allocation
Elements stored in consecutive location	Elements are note stored in consecutive location

Physical order of elements	Logical order of elements

Advantages of Doubly Linked List over Singly Linked List.

Since the doubly linked list has forward and backward pointer,it can traverse the list in the forward and backward direction.

Disadvantage of DLL

Extra space is required to store the address of the previous element and next element.

Advantages of Circular Linked List.

- Allows to traverse the list starting at any position
- Quick access to first and last record.
- Traverse the list in both direction.

Applications of Linked List

1. Polynomial ADT
2. Radix Sort
3. Multilist

Polynomial ADT

- A polynomial object is a homogeneous ordered list of pairs <exponent,coefficient>, where each coefficient is unique.
- Operations include returning the degree, extracting the coefficient for a given exponent, addition, multiplication, evaluation for a given input.
- Eg.

DECLARATION FOR LINKED LIST IMPLEMENTATION OF POLYNOMIAL ADT

```
Struct poly
{
int coeff ;
int power;
Struct poly *Next;
}*list 1, *list 2, *list 3;
```

CREATION OF THE POLYNOMIAL

```
poly create (poly *head1, poly *newnode1)
{
poly *ptr;
```

```
if (head1= =NULL)
{
head1 = newnode1;
return (head1);
}
else
{
ptr = head1;
while (ptr    next ! = NULL)
ptr = ptr→next;
ptr→next = newnode1;
}
return (head1);
}
```

ADDITION OF TWO POLYNOMIALS

```
void add ( )
{
poly *ptr1, *ptr2, *newnode;
ptr1 = list1;
ptr2 = list2;
while (ptr1! = NULL && ptr2! = NULL)
{
newnode = malloc (sizeof (Struct poly));
if (ptr1→power = = ptr2    power)
{
newnode→coeff = ptr1→coeff + ptr2→coeff;
newnode→power = ptr1→power;
newnode→next = NULL;
list 3 = create (list3, newnode);
ptr1 = ptr1→next;
ptr2 = ptr2→next;
}
else
{
if (ptr1→power > ptr2→    power)
{
newnode→coeff = ptr1→coeff;
newnode→power = ptr1→power;
newnode→next = NULL;
list3 = create (list3, newnode);
ptr1 = ptr1→next;
}
else
{
newnode→coeff = ptr2→coeff;
newnode→power = ptr2→power;
```

```
newnode→next = NULL;
list3 = create (list3, newnode);
ptr2 = ptr2→    next;
} } }
```

SUBTRACTION OF TWO POLYNOMIAL

```
void sub ( )
{
poly *ptr1, *ptr2, *newnode;
ptr1 = list1 ;
ptr2 = list 2;
while (ptr1! = NULL && ptr2! = NULL)
{
newnode = malloc (sizeof (Struct poly));
if (ptr1    power = = ptr2    power)
{
newnode→coeff = (ptr1    coeff) - (ptr2    coeff);
newnode→power = ptr1    power;
newnode→next = NULL;
list3 = create (list 3, newnode);
ptr1 = ptr1→next;
ptr2 = ptr2→next;
}
else
{
if (ptr1→power > ptr2→power)
{
newnode→coeff = ptr1→coeff;
newnode→power = ptr1→power;
newnode→next = NULL;
list 3 = create (list 3, newnode);
ptr1 = ptr1→next;
}
else
{
newnode→coeff = - (ptr2    coeff);
newnode→power = ptr2→    power;
newnode→next = NULL;
list 3 = create (list 3, newnode);
ptr2 = ptr2    next;
}
}
}
}
```

POLYNOMIAL DIFFERENTIATION

```
void diff ( )
{
poly *ptr1, *newnode;
```

```
ptr1 = list 1;
while (ptr1 ! = NULL)
{
newnode = malloc (sizeof (Struct poly));
newnode    coeff = ptr1    coeff *ptr1    power;
newnode    power = ptr1    power - 1;
newnode    next = NULL;
list 3 = create (list 3, newnode);
ptr1 = ptr1→next;
}
}
```

Radix Sort : - (Or) Card Sort

Radix Sort is the generalised form of Bucket sort. It can be performed using buckets from 0 to 9.

In First Pass, all the elements are sorted according to the least significant bit.

In second pass, the numbers are arranged according to the next least significant bit and so on this process is repeated until it reaches the most significant bits of all numbers.

The numbers of passes in a Radix Sort depends upon the number of digits in the numbers given.

PASS 1 :
INPUT : 25, 256, 80, 10, 8, 15, 174, 187
After Pass 1 : 80, 10, 174, 25, 15, 256, 187, 8
PASS 2 :
INPUT : 80, 10, 174, 25, 15, 256, 187, 8
After Pass 2 : 8, 10, 15, 25, 256, 174, 80, 187
PASS 3 :
INPUT : 8, 10, 15, 25, 256, 174, 80, 187
After pass 3 : 8, 10, 15, 25, 80, 175, 187, 256
Maximum number of digits in the given list is 3. Therefore the number of passes required to sort the list of elements is 3.

MultiList

More complicated application of Linked list is Multilist

It is useful to maintain employee involved in different projects.(employee list,project list created which refer each other)

Multilist saves space but takes more time to implement.

UNIT-IV

THE STACK ADT

Stack Model :

A stack is a linear data structure which follows Last In First Out (LIFO) principle, in which both insertion and deletion occur at only one end of the list called the Top.

Example : -

Pile of coins., a stack of trays in cafeteria.

Operations On Stack

The fundamental operations performed on a stack are

1. Push
2. 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 is said to be overflow.

UnderFlow

Attempt to delete an element, when the stack is empty is said to be underflow.

Implementation of Stack

Stack can be implemented using arrays and pointers.

Array Implementation

In this implementation each stack is associated with a pop pointer, which is -1 for an empty stack.

- To push an element X onto the stack, Top Pointer is incremented and then set Stack [Top] = X.
- To pop an element, the stack [Top] value is returned and the top pointer is decremented.
- pop on an empty stack or push on a full stack will exceed the array bounds.

ROUTINE TO PUSH AN ELEMENT ONTO A STACK

```
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)
```

```
return (1);
}
```

ROUTINE TO POP AN ELEMENT FROM THE STACK

```
void pop (Stack S)
{
if (IsEmpty (S))
Error ("Empty Stack");
else
{
X = S [Top];
Top = Top - 1;
}
}
int IsEmpty (Stack S)
{
if (S    Top = = -1)
return (1);
}
```

ROUTINE TO RETURN TOP ELEMENT OF THE STACK

```
int TopElement (Stack S)
{
if (! IsEmpty (s))
return S[Top];
else
Error ("Empty Stack");
return 0;
}
```

LINKED LIST IMPLEMENTATION OF STACK

- Push operation is performed by inserting an element at the front of the list.
- Pop operation is performed by deleting at the front of the list.
- Top operation returns the element at the front of the list.

DECLARATION FOR LINKED LIST IMPLEMENTATION

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

```
};
```

ROUTINE TO CHECK WHETHER THE STACK IS EMPTY

```
int IsEmpty (Stack S)
{
if (S→Next = = NULL)
return (1);
}
```

ROUTINE TO CREATE AN EMPTY STACK

```
Stack CreateStack ( )
{
Stack S;
S = malloc (Sizeof (Struct Node));
if (S = = NULL)
Error (" Outof Space");
MakeEmpty (s);
return S;
}
void MakeEmpty (Stack S)
{
if (S = = NULL)
Error (" Create Stack First");
else
while (! IsEmpty (s))
pop (s);
}
```

ROUTINE TO PUSH AN ELEMENT ONTO A STACK

```
void push (int X, Stack S)
{
Struct Node * 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 RETURN TOP ELEMENT IN A STACK

```
int Top (Stack S)
{
If (! IsEmpty (s))
return S→Next→Element;
Error ("Empty Stack");
```

```
return 0;
}
```

ROUTINE TO POP FROM A STACK

```
void pop (Stack S)
{
Struct Node *Tempcell;
If (IsEmpty (S))
Error ("Empty Stack");
else
{
Tempcell = S→Next;
S→Next = S→Next→Next;
Free (Tempcell);
}
}
```

APPLICATIONS OF STACK

Some of the applications of stack are :
(i) Evaluating arithmetic expression
(ii) Balancing the symbols
(iii) Towers of Hannoi
(iv) Function Calls.
(v) 8 Queen Problem.

Different Types of Notations To Represent Arithmetic Expression

There are 3 different ways of representing the algebraic expression.
They are
* INFIX NOTATION
* POSTFIX NOTATION
* PREFIX NOTATION

INFIX

In Infix notation, The arithmetic operator appears between the two operands to which it is being applied.
For example : - A / B + C

POSTFIX

The arithmetic operator appears directly after the two operands to which it applies. Also called reverse polish notation. ((A/B) + C)
For example : - AB / C +

PREFIX

The arithmetic operator is placed before the two operands to which it applies. Also called as polish notation. ((A/B) + C)
For example : - +/ABC

INFIX PREFIX (or) POLISH POSTFIX (or) REVERSE POLISH

1. (A + B) / (C - D) /+AB - CD AB + CD - /
2. A + B*(C - D) +A*B - CD ABCD - * +
3. X * A / B - D - / * XABD X A*B/D-

4. X + Y * (A - B) / +X/*Y - AB - CD XYAB - *CD - / +
(C - D)
5. A * B/C + D + / * ABCD AB * C / D +

1. Evaluating Arithmetic Expression

To evaluate an arithmetic expressions, first convert the given infix expression to postfix expression and then evaluate the postfix expression using stack.

Infix to Postfix Conversion

Read the infix expression one character at a time until it encounters the delimiter. "#"
Step 1 : If the character is an operand, place it on to the output.
Step 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.
Step 3 : If the character is a left parenthesis, push it onto the stack.
Step 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.

Evaluating Postfix Expression

Read the postfix expression one character at a time until it encounters the delimiter `#'.
Step 1 : - If the character is an operand, push its associated value onto the stack.
Step 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.

Function Calls

When a call is made to a new function all the variables local to the calling routine need to be saved, otherwise the new function will overwrite the calling routine variables. Similarly the current location address in the routine must be saved so that the new function knows where to go after it is completed.

BALANCING THE SYMBOLS

Step 1: Make an empty stack
Step 2: Read the characters until it reach the end of file
Step 3: If the character is an opening symbol, push it onto the stack.
Step 4: If the character is a closing symbol and if the stack is empty report an error as missing opening symbol.
Step 5: If the character is a closing symbol and if it has corresponding opening symbol in the stack pop it from the stack. Otherwise report an error as mismatched symbol.
Step 6: At the end of the file, if the stack is not empty ,report an error as missing closing symbol. Otherwise report as balanced symbols.

The Queue ADT

Queue Model

A Queue 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.
Example : Waiting Line in Reservation Counter,

Operations on Queue

The fundamental operations performed on queue are

1. Enqueue
2. Dequeue

Enqueue :

The process of inserting an element in the queue.

Dequeue :

The process of deleting an element from the queue.

Exception Conditions

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

Implementation of Queue

Queue can be implemented using arrays and pointers.

Array Implementation

In this implementation queue Q is associated with two pointers namely rear pointer and front pointer.
To insert an element X onto the Queue Q, the rear pointer is incremented by 1 and then set Queue [Rear] = X
To delete an element, the Queue [Front] is returned and the Front Pointer is incremented by 1.

ROUTINE TO ENQUEUE

```
void Enqueue (int X)
{
if (rear > = max _ Arraysize)
print (" Queue overflow");
else
{
Rear = Rear + 1;
Queue [Rear] = X;
}
}
```

ROUTINE FOR DEQUEUE

```
void delete ( )
{
if (Front < 0)
print (" Queue Underflow");
else
{
X = Queue [Front];
if (Front = = Rear)
{
Front = 0;
Rear = -1;
}
else
Front = Front + 1 ;
}
}
```

In Dequeue operation, if Front = Rear, then reset both
the pointers to their initial values. (i.e. F = 0, R = -1)

Linked List Implementation of Queue

Enqueue operation is performed at the end of the list. (rear)
Dequeue operation is performed at the front of the list. (front)

DECLARATION FOR LINKED LIST IMPLEMENTATION OF QUEUE ADT

```
Struct Node;
typedef Struct Node * Queue;
int IsEmpty (Queue Q);
Queue CreateQueue (void);
void MakeEmpty (Queue Q);
void Enqueue (int X, Queue Q);
void Dequeue (Queue Q);
Struct Node
{
int Element;
Struct Node *Next;
}* Front = NULL, *Rear = NULL;
```

ROUTINE TO CHECK WHETHER THE QUEUE IS EMPTY

```
int IsEmpty (Queue Q) // returns boolean value /
{ // if Q is empty
if (Q→Next = = NULL) // else returns 0
return (1);
}
```

ROUTINE TO CHECK AN EMPTY QUEUE

```
Struct CreateQueue ( )
{
Queue Q;
Q = Malloc (Sizeof (Struct Node));
if (Q = = NULL)
Error ("Out of Space");
MakeEmpty (Q);
return Q;
}
void MakeEmpty (Queue Q)
{
if (Q = = NULL)
Error ("Create Queue First");
else
while (! IsEmpty (Q)
Dequeue (Q);
}
```

ROUTINE TO ENQUEUE AN ELEMENT IN QUEUE

```
void Enqueue (int X)
{
Struct node *newnode;
newnode = Malloc (sizeof (Struct node));
if (Rear = = NULL)
{
```

```
newnode →data = X;
newnode →Next = NULL;
Front = newnode;
Rear = newnode;
}
else
{
newnode →    data = X;
newnode →    Next = NULL;
Rear →next = newnode;
Rear = newnode;
}
}
```

ROUTINE TO DEQUEUE AN ELEMENT FROM THE QUEUE

```
void Dequeue ( )
{
Struct node *temp;
if (Front = = NULL)
Error("Queue is underflow");
else
{
temp = Front;
if (Front = = Rear)
{
Front = NULL;
Rear = NULL;
}
else
Front = Front →Next;
Print (temp→data);
free (temp);
}
}
```

Double Ended Queue (DEQUE)

In Double Ended Queue, insertion and deletion operations are performed at both the ends.

Circular Queue

In Circular Queue, the insertion of a new element is performed at the very first location of the queue if the last location of the queue is full, in which the first element comes just after the last element.

Advantages of circular queue

It overcomes the problem of unutilized space in linear queues, when it is implemented as arrays. To perform the insertion of an element to the queue, the position of the element is calculated by the relation as

Rear = (Rear + 1) % Maxsize.

and then set

Queue [Rear] = value.

ROUTINE TO INSERT AN ELEMENT IN CIRCULAR QUEUE

```
void CEnqueue (int X)
```

```
{
if (Front = = (rear + 1) % Maxsize)
print ("Queue is overflow");
else
{
if (front = = -1)
front = rear = 0;
else
rear = (rear + 1)% Maxsize;
CQueue [rear] = X;
}
}
```

To perform the deletion, the position of the Front printer is calculated by the relation
Value = CQueue [Front]
Front = (Front + 1) % maxsize.

ROUTINE TO DELETE AN ELEMENT FROM CIRCULAR QUEUE

```
int CDequeue ( )
{
if (front = = -1)
print ("Queue is underflow");
else
{
X = CQueue [Front];
if (Front = = Rear)
Front = Rear = -1;
else
Front = (Front + 1)% maxsize;
}
return (X);
}
```

.

Applications of Queue

* Batch processing in an operating system
* To implement Priority Queues.
* Priority Queues can be used to sort the elements using Heap Sort.
* Simulation.
* Mathematics user Queueing theory.
* Computer networks where the server takes the jobs of the client as per the queue strategy.

Implementation of List using ARRAY: (C Language)

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
int n,list[25];
void create();
int insert();
int del();
void display();
void find();

void main()
{
int ch;
clrscr();
printf("\n\t\t\t\tLIST USING ARRAY");
printf("\n\t\t\t\t**** ***** *****");

do{
printf("\n\nMAIN MENU");
printf("\n1.Create\n2.Insert\n3.Delete\n4.Find\n5.Display\n6.Exit");
printf("\nEnter your choice");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\nenter the number of elements");
scanf("%d",&n);
create();
display();
break;

case 2:
n=insert();
display();
break;

case 3:
n=del();
display();
break;

case 4:
find();
break;
```

```
case 5:
display();
break;

case 6:
exit(0);
}
}while(ch<=6);
}


int insert()
{
int i,ele,pos;
printf("\nEnter the number and position");
scanf("%d %d",&ele,&pos);

if(n==25)
{
printf("\nlist is full");
return n;
}
else if(n==0)
{
printf("\nlist has to be created first");
return 0;
}
else if(pos==n+1)
{
list[pos]=ele;
n=n+1;
return n;
}
else if(pos>n)
{
printf("\nnot valid");
return n;
}
else
{
for(i=n;i>=pos;i--)
list[i+1]=list[i];
list[pos]=ele;
n=n+1;
return n;
}
}
```

```
void create()
{
int i;
printf("\nlist to be created:");
for(i=1;i<=n;i++)
scanf("%d",&list[i]);
}
int del()
{
int i,pos;
printf("enter the position to be deleted:");
scanf("%d",&pos);
for(i=pos;i<=n;i++)
list[i]=list[i+1];
n--;
return n;
}
void find()
{
int i,find;
printf("\nenter the data to be searched:");
scanf("%d",&find);
if(n==0)
{
printf("list is empty");
goto jump;
}
else
{
for(i=1;i<=n;i++)
if(list[i]==find)
{
printf("\nthe element %d is positioned at %d\n",find,i);
getch();
goto jump;
}}
printf("\nelement is not found");
jump:
}
void display()
{
int i;
if(n==0)
printf("\nlist is empty");
else
{for(i=1;i<=n;i++)
printf("\t%d",list[i]);
```

}}

LIST ADT USING Linked List(C Language)

Program

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
void create();
void insert();
void del();
void display();
struct node
{
        int data;
        struct node *link;
};
struct node *first=NULL,*last=NULL,*next,*prev,*cur;
void create()
{
        cur=(struct node *)malloc(sizeof(struct node));
        printf("enter the data:");
        scanf("%d",&cur->data);
        cur->link=NULL;
        first=cur;
        last=cur;
}
void insert()
{
        int pos,c=1;
        cur=(struct node*)malloc(sizeof(struct node));
        printf("enter the data:");
        scanf("%d",&cur->data);
        printf("Enter the position:");
        scanf("%d",&pos);
        if(pos==1&&first!=NULL)
        {
                cur->link=first;
                first=cur;
        }
        else
        {
                next=first;
                while( c<pos)
                {
                        prev=next;
                        next=prev->link;
                        c++;
```

```
                }
                if(prev==NULL)
                {
                        printf("invalid position");
                }
                else
                {
                        cur->link=prev->link;
                        prev->link=cur;
                }
        }
}
void del()
{
        int pos,c=1;
        printf("Enter the position to be deleted");
        scanf("%d",&pos);
        if(first==NULL)
        {
                printf("list is empty");
        }
        else
        if(pos==1 && first->link==NULL)
        {
                printf("Deleted element is %d",first->data);
                free(first);
                first=NULL;
        }
        else
        if(pos==1 && first->link!=NULL)
        {
                cur=first;
                first=first->link;
                cur->link=NULL;
                printf("Deleted element is %d",cur->data);
                free(cur);
        }
        else
        {
                next=first;
                while(c<pos)
                {
                        cur=next;
                        next=next->link;
                        c++;
                }
                cur->link=next->link;
                next->link=NULL;
```

```
                if(next==NULL)
                {
                        printf("invalid position");
                }
                else
                {
                        printf("Deleted element is %d",next->data);
                        free(next);
                }
        }
}
void display()
{
        cur=first;
        printf("\n The elements in the list are:");
        while(cur!=NULL)
        {
                printf("%d\t",cur->data);
                cur=cur->link;
        }
}
void main()
{
        int ch;
        clrscr();
        printf("singly linked list:");
        do
        {
                printf("\n 1.Create \n 2.Insert \n 3.Delete \n 4.Exit");
                printf("\n\nEnter your option");
                scanf("%d",&ch);
                switch(ch)
                {
                case 1:
                        create();
                        display();
                        break;
                case 2:
                        insert();
                        display();
                        break;
                case 3:
                        del();
                        display();
                        break;
                case 4:
                        exit(0);
```

```
                }
        }
        while(ch<=3);
}
```

Sample Output

```
Singly linked list:
 1.Create
 2.Insert
 3.Delete
 4.Exit
Enter your option1
Enter the data:1
The elements in the list are:9
1.Create
2.Insert
3.Delete
4.Exit
Enter your option2
Enter the data:6
Enter the position:2
The elements in the list are:9 6
```

LIST ADT USING Doubly Linked List(C Language)

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
void create();
void insert();
void del();
void display();
struct node
{
        int data;
        struct node *flink,*blink;
};
struct node *first=NULL,*last=NULL,*next,*prev,*cur;
void create()
{
        cur=(struct node *)malloc(sizeof(struct node));
        printf("Enter the data:");
        scanf("%d",&cur->data);
        cur->flink=NULL;
        cur->blink=NULL;
        first=cur;
        last=cur;
}
void insert()
{
        int pos,c=1;
        cur=(struct node*)malloc(sizeof(struct node));
        printf("Enter the data:");
        scanf("%d",&cur->data);
        printf("Enter the position:");
        scanf("%d",&pos);
        if(pos==1&&first!=NULL)
        {
                cur->flink=first;
                cur->blink=NULL;
                first->blink=cur;
                first=cur;
        }
        else
        {
                next=first;
                while( c<pos)
```

```
                {
                        prev=next;
                        next=prev->flink;
                        c++;
                }
                if(prev==NULL)
                {
                        printf("Invalid position");
                }
                else
                {
                        cur->flink=prev->flink;
                        cur->blink=prev;
                        prev->flink->blink=cur;
                        prev->flink=cur;
                }
        }
}
void del()
{
        int pos,c=1;
        printf("Enter the position to be deleted");
        scanf("%d",&pos);
        if(first==NULL)
        {
                printf("DLL is empty");
        }
        else
        if(pos==1 && first->flink==NULL)
        {
                printf("Deleted element is %d",first->data);
                free(first);
                first=NULL;
                last=NULL;
        }
        else
        if(pos==1 && first->flink!=NULL)
        {
                cur=first;
                first=first->flink;
                cur->flink=NULL;
                first->blink=NULL;
        }
        else
        {
                next=first;
                while(c<pos)
                {
```

```
                cur=next;
                next=next->flink;
                c++;
            }
            if(next==NULL)
            {
                printf("Invalid position");
            }
            else
            {
                cur->flink=next->flink;
                next->flink->blink=cur;
                next->flink=NULL;
                next->blink=NULL;
                printf("Deleted element is %d",next->data);
                free(next);
            }
        }
}
void display()
{
        cur=first;
        printf("\n The elements in the list are:");
        while(cur!=NULL)
        {
            printf("%d\t",cur->data);
            cur=cur->flink;
        }
}
void main()
{
        int ch;
        clrscr();
        printf("Doubly linked list:");
        do
        {
            printf("\n 1.Create \n 2.Insert \n 3.Delete \n 4.Exit");
            printf("\n\nEnter your option");
            scanf("%d",&ch);
            switch(ch)
            {
            case 1:
                create();
                display();
                break;
            case 2:
                insert();
                display();
```

```
                    break;
                case 3:
                    del();
                    display();
                    break;
                case 4:
                    exit(0);

                }
        }
        while(ch<=3);
}
```

Sample Output

```
Doubly linked list:
 1.Create
 2.Insert
 3.Delete
 4.Exit
Enter your option 1
Enter the data:8
The elements in the list are:1
 1.Create
 2.Insert
 3.Delete
 4.Exit
Enter your option2
Enter the data:4
Enter the position:2
 The elements in the list are:8 4
```

Addition of two polynomial using linked list

```
#include<stdio.h>
#include<malloc.h>
#include<conio.h>
struct link
{
    int coeff;
    int pow;
    struct link *next;
};
struct link *poly1=NULL,*poly2=NULL,*poly=NULL;

void create(struct link *node)
{
    char ch;
    do
    {
        printf("\n Enter coeff:");
        scanf("%d",&node->coeff);
        printf("\n Enter power:");
        scanf("%d",&node->pow);
        node->next=(struct link*)malloc(sizeof(struct link));
        node=node->next;
        node->next=NULL;
        printf("\n Continue(y/n):");
        ch=getch();
    }
    while(ch=='y' || ch=='Y');
}
void show(struct link *node)
{
    while(node->next!=NULL)
    {
        printf("%dx^%d",node->coeff,node->pow);
        node=node->next;
        if(node->next!=NULL)
              printf("+");
    }
}
void polyadd(struct link *poly1,struct link *poly2,struct link *poly)
{
    while(poly1->next &&  poly2->next)
    {
         if(poly1->pow>poly2->pow)
```

```
        {
            poly->pow=poly1->pow;
            poly->coeff=poly1->coeff;
            poly1=poly1->next;
        }
        else if(poly1->pow<poly2->pow)
        {
            poly->pow=poly2->pow;
            poly->coeff=poly2->coeff;
            poly2=poly2->next;
        }
        else
        {
            poly->pow=poly1->pow;
            poly->coeff=poly1->coeff+poly2->coeff;
            poly1=poly1->next;
            poly2=poly2->next;
        }
        poly->next=(struct link *)malloc(sizeof(struct link));
        poly=poly->next;
        poly->next=NULL;
    }
    while(poly1->next || poly2->next)
    {
        if(poly1->next)
        {
            poly->pow=poly1->pow;
            poly->coeff=poly1->coeff;
            poly1=poly1->next;
        }
        if(poly2->next)
        {
            poly->pow=poly2->pow;
            poly->coeff=poly2->coeff;
            poly2=poly2->next;
        }
        poly->next=(struct link *)malloc(sizeof(struct link));
        poly=poly->next;
        poly->next=NULL;
    }
}
main()
{
    char ch;
    do
    {
        poly1=(struct link *)malloc(sizeof(struct link));
        poly2=(struct link *)malloc(sizeof(struct link));
```

```
        poly=(struct link *)malloc(sizeof(struct link));
        printf("\nEnter 1st number:");
        create(poly1);
        printf("\nEnter 2nd number:");
        create(poly2);
        printf("\n1st Number:");
        show(poly1);
        printf("\n2nd Number:");
        show(poly2);
        polyadd(poly1,poly2,poly);
        printf("\nAdded polynomial:");
        show(poly);
        printf("\n Add two more numbers:");
        ch=getch();
    }
    while(ch=='y' || ch=='Y');
}
```

Sample Output

Enter 1st number:
Enter coeff:2
Enter power:2
Continue(y/n):y
Enter coeff:2
Enter power:1
Continue(y/n):y
Enter coeff:2
Enter power:0
Continue(y/n):n
Enter 2nd number:
Enter coeff:1
Enter power:2
Continue(y/n):y
Enter coeff:1
Enter power:1
Continue(y/n):y
Enter coeff:1
Enter power:0
1st Number:6x^2+8x^1+3x^0
2nd Number:1x^2+1x^1+1x^0
Added polynomial:7x^2+9x^1+4x^0
Add two more numbers: n

An array based circular queue and use it to simulate a producer-consumer problem.

Program

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#include<alloc.h>
#define size 3
char cqueue[size];
int front=-1,rear;
void producer_enqueue(int element);
int consumer_dequeue();
void main()
{
        int ch,elem;
        char c;
        clrscr();
        do
                {
                    printf("\n\t\t\t\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");
                    printf("ENTER YOUR CHOICE");
                    scanf("%d",&ch);
                    switch(ch)
                       {
                                case 1:
                                printf("\n enter the element");
                                scanf("%d",&elem);
                                producer_enqueue(elem);
                                break;
                                case 2:
                                elem=consumer_dequeue();
                                if(elem!=-1)
                                printf("\consumer element is %d",elem);
                                break;
                                case 3:
                                exit(0);
                       }
                       printf("\nCONTINUE Y/N ?\n");
                       c=getch();
                }while(c=='y'||c=='Y');
}
void producer_enqueue(int element)
        {
```

```
        if(front==(rear+1)%size)
        {
                printf("\n consumer queue is full\n");
        }
        else
        {
                if(front==-1)
                front=rear=0;
                else
                rear=(rear+1)%size;
                cqueue[rear]=element;
        }
}
int consumer_dequeue()
        {
                int element=-1;
                if(front==-1)
                {
                        printf("\n the consumer queue is empty\n");
                }
                else
                {
                        element=cqueue[front];
                        if(front==rear)
                        front=rear=-1;
                        else
                        front=(front+1)%size;
                }
        return element;
        }
```

Output:

```
1.PRODUCER
2.CONSUMER
3.EXIT
ENTER YOUR CHOICE 1
enter the element8
CONTINUE Y/N ?
1.PRODUCER
2.CONSUMER
3.EXIT
ENTER YOUR CHOICE 1
enter the element7
CONTINUE Y/N ?
1.PRODUCER
2.CONSUMER
3.EXIT
ENTER YOUR CHOICE 2
```

consumer element is 8
CONTINUE Y/N ?

/* IMPLEMENTATION OF STACK USING ARRAY */

```
#include<stdio.h>
#include<conio.h>
int i,top,item,s[10];
int max=10;
void push(int item,int s[]);
void pop(int s[]);
void display(int s[]);
void main()
{
int ch;
clrscr();
top=-1;
do
{
printf("\n\n 1.PUSH \n 2.POP \n 3.EXIT \n");
printf("\n ENTER YOUR CHOICE : ");
scanf("%d",&ch);
switch(ch)
{
case 1:printf("\t PUSH \n");
        if(top>=max-1)
        {
        printf("\n STACK IS FULL\n");
        }
        else
        {
        printf("\n ENTER AN ELEMENT : ");
        scanf("%d",&item);
        push(item,s);
        }
        display(s);
        break;
case 2:printf("\t\n POP \n");
        if(top<0)
        {
        printf("\nSTACK IS EMPTY\n");
        }
        else
        {
        pop(s);
        }
        display(s);
```

```
            break;
}}while(ch!=3);
getch();
}
void push(int item,int s[])
{
top=top+1;
s[top]=item;
}
void pop(int s[])
{
item=s[top];
top=top-1;
printf("\nDELETED ELEMENT IS %d\n",item);
}
void display(int s[])
{
printf("\n");
for(i=top;i>=0;i--)
{
printf(" \n %d",s[i]);}}
```

/* IMPLEMENTATION OF STACK USING LINKED LIST */

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<stdlib.h>
struct node
{
int data;
struct node *link;
};
struct node *cur,*first;

void create();
void push();
void pop();
void display();

void create()
{
printf("\nENTER THE FIRST ELEMENT: ");
cur=(struct node *)malloc(sizeof(struct node));
scanf("%d",&cur->data);
cur->link=NULL;
first=cur;
}
```

```
void display()
{
cur=first;
printf("\n");
while(cur!=NULL)
{
printf("%d\n",cur->data);
cur=cur->link;
}
}
void push()
{
printf("\nENTER THE NEXT ELEMENT: ");
cur=(struct node *)malloc(sizeof(struct node));
scanf("%d",&cur->data);
cur->link=first;
first=cur;
}

void pop()
{
if(first==NULL)
{
printf("\nSTACK IS EMPTY\n");
}
else
{
cur=first;
printf("\nDELETED ELEMENT IS %d\n",first->data);
first=first->link;
free(cur);
}
}

void main()
{
int ch;
clrscr();
while(1)
{
printf("\n\n 1.CREATE \n 2.PUSH \n 3.POP \n 4.EXIT \n");
printf("\n ENTER YOUR CHOICE : ");
scanf("%d",&ch);
switch(ch)
{
```

```
case 1:
        create();
        display();
        break;
case 2:
        push();
        display();
        break;
case 3:
        pop();
        display();
        break;
case 4:
        exit(0);
}
}
}
```

/* IMPLEMENTATION OF QUEUE USING ARRAY */

```
#include<stdio.h>
#include<conio.h>
int i,rear,front,item,s[10];
void insert(int item,int s[]);
void del(int s[]);
void display(int s[]);
void main()
{
int ch;
clrscr();
front=0;
rear=-1;
do
{
printf("\n\n 1.INSERTION \n 2.DELETION \n 3.EXIT \n");
printf("\nENTER YOUR CHOICE : ");
scanf("%d",&ch);
switch(ch)
{
case 1:

        printf("\n\t INSERTION \n");
        if(rear>=9)
        {
        printf("\t\nQUEUE IS FULL\n");
        }
        else
        {
```

```
        printf("\nENTER AN ELEMENT : ");
        scanf("%d",&item);
        insert(item,s);
        }
        display(s);
        break;
case 2:
        printf("\n\t DELETION \n");
        if(front>rear)
        {
        printf("\t\nQUEUE IS EMPTY\n");
        }
        else
        {
        del(s);
        }
        display(s);
        break;
}
}while(ch!=3);
getch();
}
void insert(int item,int s[])
{
rear=rear+1;
s[rear]=item;

}
void del(int s[])
{
item=s[front];
front=front+1;
printf("\n DELETED ELEMENT IS %d\n\n",item);
}

void display(int s[])
{
printf("\n");
for(i=front;i<=rear;i++)
{
printf(" \t %d",s[i]);
}
}
```

/* IMPLEMENTATION OF QUEUE USING LINKED LIST */

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<stdlib.h>
struct node
{
int data;
struct node *link;
};
struct node *cur,*first,*last;

void create();
void insert();
void delte();
void display();

void create()
{
printf("\nENTER THE FIRST ELEMENT: ");
cur=(struct node *)malloc(sizeof(struct node));
scanf("%d",&cur->data);
cur->link=NULL;
first=cur;
last=cur;
}

void insert()
{
printf("\nENTER THE NEXT ELEMENT: ");
cur=(struct node *)malloc(sizeof(struct node));
scanf("%d",&cur->data);
cur->link=NULL;
last->link=cur;
last=cur;
}

void delte()
{
if(first==NULL)
{
printf("\t\nQUEUE IS EMPTY\n");
}
```

```
else
{
cur=first;
first=first->link;
cur->link=NULL;
printf("\n DELETED ELEMENT IS %d\n",cur->data);
free(cur);
}
}

void display()
{
cur=first;
printf("\n");
while(cur!=NULL)
{
printf("\t%d",cur->data);
cur=cur->link;
}
}

void main()
{
int ch;
clrscr();
while(1)
{
printf("\n\n 1.CREATE \n 2.INSERT \n 3.DELETE \n 4.EXIT \n");
printf("\nENTER YOUR CHOICE : ");
scanf("%d",&ch);
switch(ch)
{
case 1:
        create();
        display();
        break;
case 2:
        insert();
        display();
        break;
case 3:
        delte();
        display();
        break;
case 4:
        exit(0);
```