**List of Experiments**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Ex. No** | **Name of the experiment** | **Page No.** |
| **1** | **1a** | Implementation of Sorting   1. Bubble sort 2. Insertion Sort | **2**  **4** |
|  | **1b** | Implementation of Searching   1. Linear Search 2. Binary Search | **6**  **8** |
| **2** | **2a** | Singly Linked List Implementation | **10** |
|  | **2b** | Doubly Linked List Implementation | **16** |
|  | **2c** | Circular Linked List Implementation | **23** |
| **3** | **3a** | Implementation of stack using array | **29** |
|  | **3b** | Implementation of stack using Linked list | **33** |
| **4** | **4a** | Implementation of queue using array | **36** |
|  | **4b** | Implementation of queue using Linked list | **40** |
| **5** | **5a** | Applications of Stack  Infix to Postfix | **43** |
|  | **5b** | Applications of Queue  Priority queue | **46** |
| **6** | **6** | Implementation of Binary tree traversal | **52** |
| **7** | **7** | Implementation of Binary Search Tree | **58** |
| **8** | **8** | Minimum spanning tree using prims/kruskals | **62** |
| **9** | **9** | Shortest path algorithm using Dijkstra | **64** |

**Ex. No: 1a(a) SORTING - Bubble Sort**

**Aim**: To write a c program to perform bubble sorting using arrays.

**Algorithm:**

1. Input n numbers of an array A

2. Initialise i = 0 and repeat through step 4 if (i < n)

3. Initialize j = 0 and repeat through step 4 if (j < n – i – 1)

4. If (A[j] > A[j + 1]) (a) Swap = A[j] (b) A[j] = A[j + 1] (c) A[j + 1] = Swap

5. Display the sorted numbers of array A

6. Exit

**Program:**

*#include <stdio.h>*

*int main()*

*{*

*int array[100], n, c, d, swap;*

*printf("Enter number of elements\n");*

*scanf("%d",&n);*

*printf("Enter %d integers\n", n);*

*for(c =0; c < n; c++)*

*scanf("%d",&array[c]);*

*for(c =0; c <( n -1); c++)*

*{*

*for(d =0; d < n - c -1; d++)*

*{*

*if(array[d]> array[d+1])/\* For decreasing order use < \*/*

*{*

*swap = array[d];*

*array[d]= array[d+1];*

*array[d+1]= swap;*

*}*

*}*

*}*

*printf("Sorted list in ascending order:\n");*

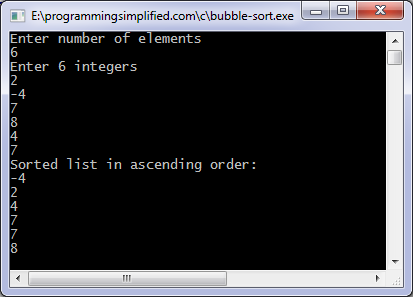
*for( c =0; c < n ; c++)*

*printf("%d\n", array[c]);*

*return0;*

*}*

**Output**

****

**Result:** Thus the program bubble sort written in C language is implemented and executed successfully.

**Ex. No:1a(b) INSERTION SORT**

**Aim:**  To implement insertion sort using array in C

**Algorithm:**

1. Input an array A of *n* numbers

2. Initialize *i* = 1 and repeat through steps 4 by incrementing *i* by one.

(*a*) If (*i* < = *n* – 1)

(*b*) Swap = A [I],

(*c*) Pos = *i* – 1

3. Repeat the step 3 if (Swap < A[Pos] and (Pos >= 0))

(*a*) A [Pos+1] = A [Pos]

(*b*) Pos = Pos-1

4. A [Pos +1] = Swap

5. Exit

**Program:**

*#include<conio.h>*

*#include<stdio.h>*

*#define MAX 20*

*void main()*

*{*

*int arr[MAX],i,j,k,n;*

*clrscr();*

*printf (“\nEnter the number of elements : ”);*

*scanf (“%d”,&n);*

*for (i = 0; i < n; i++)*

*{*

*printf (“\nEnter element %d : ”,i+1);*

*scanf (“%d”, &arr[i]);*

*}*

*printf (“\nUnsorted list is :\n”);*

*for (i = 0; i < n; i++)*

*printf (“%d ”, arr[i]);*

*printf (“\n”);*

*/\*Insertion sort\*/*

*for(j=1;j < n;j++)*

*{*

*k=arr[j]; /\*k is to be inserted at proper place\*/*

*for(i=j–1;i>=0 && k<arr[i];i--)*

*arr[i+1]=arr[i];*

*arr[i+1]=k;*

*printf (“\nPass %d, Element inserted in proper place: %d\n”,j,k);*

*for (i = 0; i < n; i++)*

*printf (“%d ”, arr[i]);*

*printf (“\n”);*

*}*

*printf (“\nSorted list is :\n”);*

*printf (“%d ”, arr[i]);*

*getch();*

*}/\*End of main()\*/*

**Output:**

Enter the number of elements : 5

Enter Element : 7 3 2 9 6

Unsorted list : 7 3 2 9 6

After pass 1 elements are : 3 7 2 9 6

After pass 2 elements are : 2 3 7 6 9

After pass 3 elements are : 2 3 6 7 9

Sorted List : 2 3 6 7 9

**Result:** Thus the program insertion sort written in C language was implemented and executed successfully.

**Ex.No: 1b(a) LINEAR SEARCH**

**Aim:** To Write a C program to implement linear search

**Algorithm:**

1. Input an array A of n elements and “data” to be searched and initialise loc = – 1.

2. Initialise i = 0; and repeat through step 3 if (i < n) by incrementing i by one .

3. If (data = A[i]) (a) loc = i (b) GOTO step 4

4. If (loc > 0) (a) Display “data is found and searching is successful”

5. Else (a) Display “data is not found and searching is unsuccessful”

6. Exit

**Program:**

#include <stdio.h>

 int main()

{

int array[100], search, c, n;

printf("Enter the number of elements in array**\**n");

scanf("%d",&n);

printf("Enter %d integer(s)**\**n", n);

for(c =0; c < n; c++)

scanf("%d",&array[c]);

printf("Enter the number to search**\**n");

scanf("%d",&search);

for(c =0; c < n; c++)

{

if(array[c]== search)*/\* if required element found \*/*

{

printf("%d is present at location %d.**\**n", search, c+1);

break;

}

}

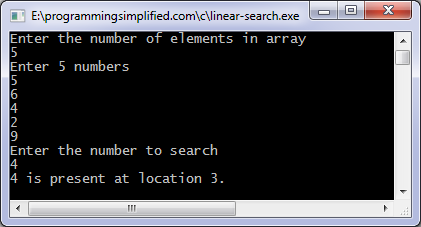
if(c == n)

printf("%d is not present in array.**\n**", search);

return0;

}

**Output:**



**Result:** Thus the program linear search written in C language was implemented and executed successfully.

**Ex. No: 1b(b) BINARY SEARCH**

**Aim:** To Write a C program to implement binary search.

**Algorithm:**

1. Input an array A of n elements and “data” to be sorted

2. LB = 0, UB = n; mid = int ((LB+UB)/2)

3. Repeat step 4 and 5 while (LB <= UB) and (A[mid] ! = data)

4. If (data < A[mid]) (a) UB = mid–1

5. Else (a) LB = mid + 1

6. Mid = int ((LB + UB)/2)

7. If (A[mid]== data) (a) Display “the data found”

8. Else (a) Display “the data is not found”

9. Exit

**Program:**

#include <stdio.h>

 int main()

{

int c, first, last, middle, n, search, array[100];

printf("Enter number of elements**\n**");

scanf("%d",&n);

printf("Enter %d integers**\n**", n);

for(c =0; c < n; c++)

scanf("%d",&array[c]);

printf("Enter value to find**\n**");

scanf("%d",&search);

first =0;

last = n -1;

middle =(first+last)/2;

while(first <= last){

if(array[middle]< search)

first = middle +1;

elseif(array[middle]== search){

printf("%d found at location %d.**\n**", search, middle+1);

break;

}

else

last = middle -1;

middle =(first + last)/2;

}

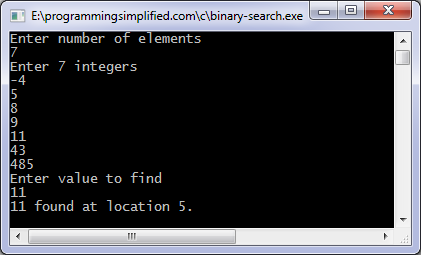
if(first > last)

printf("Not found! %d is not present in the list.**\n**", search);

return0;

}

**Output:**



**Result:**  Thus the program binary search written in C language was implemented and executed successfully.

**Ex.No:2a SINGLY LINKED LIST IMPLEMENTATION**

**Aim:** To write a C program to implement singly linked list operations.

**Algorithm:**

**(i) Insertion at Beginning:-**

Step1: Create a new node..

Step2: Assign the data to the data part of new node.

Step3: Now, assign the address part of head to the next part of new node.

Step4: Assign the address of new node to the address of head

**(ii) Insertion at given position:-**

Step1: Create a new node.

Step2: Assign the data to the data part of new node. Specify the position where the element to be inserted. Assume it is 3.

Step3: Now, traverse the list up to position 2 node.

Step4: Assign the next part of node 2 to next part of new node (it it node3).

Step5: Assign the address of new node to next part of node 2.

**(iii) Insertion at end:-**

Step1: Create a new node.

Step2: Assign the data to the data part of new node. Now, traverse the list till the last node

reached.

Step3: The next part of last node is assigned with address of new node.

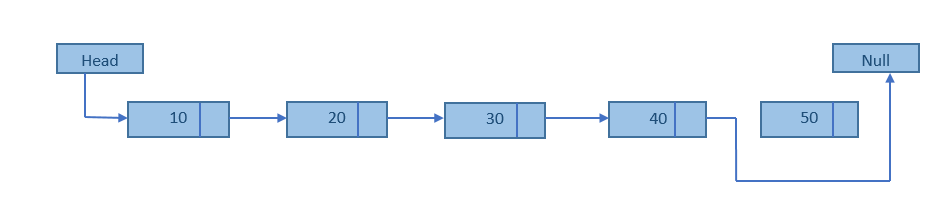
Step5: Assign the next part of new node with NULL.

**(iv) Deletion of node**

Step 1: If list is empty returns no node is present.

Step 2: Adjust the start pointer to point to the next node. Then Remove the original node and Mark the memory it used as free.

Step 3: If the node is present at the end, remove the last node and mark the prior node with null pointer. Free the memory.



Step 4: If we want to delete the node present at the middle, examine the next node using the node pointers to move from node to node until the correct node is identified. Copy the pointer of the removed node into temporary variable. Remove the node from the list and free the memory.Update the previous node’s pointer with the address held in the temporary memory.

**Program:**

*#include*

*struct node* //Each node in list will contain data and next pointer

*{*

*int data;*

*struct node \*next;*

*};*

*struct node \*start;*

*void insertbeg(void)*

*{*

*struct node \*nn;*

*int a;*

*nn=(struct node \*)malloc(sizeof(struct node));*

*printf("enter data:");*

*scanf("%d",&nn->data);*

*a=nn->data;*

*if(start==NULL)* //checking if List is empty

*{*

*nn->next=NULL;*

*start=nn;*

*}*

*else*

*{*

*nn->next=start;*

*start=nn;*

*}*

*printf("%d succ. inserted\n",a);*

*return;*

*}*

*void insertend(void)*

*{*

*struct node \*nn,\*lp;int b;*

*nn=(struct node \*)malloc(sizeof(struct node));*

*printf("enter data:");*

*scanf("%d",&nn->data);*

*b=nn->data;*

*if(start==NULL)*

*{*

*nn->next=NULL;*

*start=nn;*

*}*

*else*

*{*

*lp=start;*

*while(lp->next!=NULL)*

*{*

*lp=lp->next;*

*}*

*lp->next=nn;*

*nn->next=NULL;*

*}*

*printf("%d is succ. inserted\n",b);*

*return;*

*}*

*void insertmid(void)*

*{*

*struct node \*nn,\*temp,\*ptemp;int x,v;*

*nn=(struct node \*)malloc(sizeof(struct node));*

*if(start==NULL)*

*{*

*printf("sll is empty\n"); return;*

*}*

*printf("enter data before which no. is to be inserted:\n");*

*scanf("%d",&x);*

*if(x==start->data)*

*{*

*insertbeg();*

*return;*

*}*

*ptemp=start;*

*temp=start->next;*

*while(temp!=NULL&&temp->data!=x)*

*{*

*ptemp=temp;*

*temp=temp->next;*

*}*

*if(temp==NULL)*

*{*

*printf("%d data does not exist\n",x);*

*}*

*else*

*{*

*printf("enter data:");*

*scanf("%d",&nn->data);*

*v=nn->data;*

*ptemp->next=nn;*

*nn->next=temp;*

*printf("%d succ. inserted\n",v);*

*}*

*return;*

*}*

*void deletion(void)*

*{*

*struct node \*pt,\*t;*

*int x;*

*if(start==NULL)*

*{*

*printf("sll is empty\n");*

*return;*

*}*

*printf("enter data to be deleted:");*

*scanf("%d",&x);*

*if(x==start->data)*

*{*

*t=start;*

*start=start->next;*

*free(t);*

*printf("%d is succ. deleted\n",x);*

*return;*

*}*

*pt=start;*

*t=start->next;*

*while(t!=NULL&&t->data!=x)*

*{*

*pt=t;t=t->next;*

*}*

*if(t==NULL)*

*{*

*printf("%d does not exist\n",x);return;*

*}*

*else*

*{ pt->next=t->next;*

*}*

*printf("%d is succ. deleted\n",x);*

*free(t);*

*return;*

*}*

*void display(void)*

*{*

*struct node \*temp;*

*if(start==NULL)*

*{ printf("sll is empty\n");*

*return;*

*}*

*printf("elements are:\n");*

*temp=start;*

*while(temp!=NULL)*

*{*

*printf("%d\n",temp->data);*

*temp=temp->next;*

*}*

*return;*

*}*

*void main()*

*{*

*int c,a; start=NULL;*

*do*

*{*

*printf("1:insert\n2:delete\n3:display\n4:exit\nenter choice:");*

*scanf("%d",&c);*

*switch(c)*

*{ case 1:*

*printf("1:insertbeg\n2:insert end\n3:insert mid\nenter choice:");*

*scanf("%d",&a);*

*switch(a)*

*{ case 1:insertbeg();break;*

*case 2:insertend();break;*

*case 3:insertmid();break;*

*}*

*break;*

*case 2:deletion();break;*

*case 3:display();break;*

*case 4:printf("program ends\n");break;*

*default:printf("wrong choice\n");*

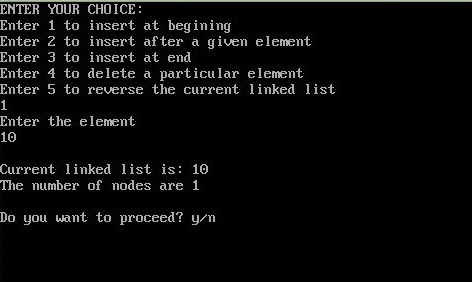
*break;*

*}*

*}while(c!=4);*

*}*

**Output:**



**Result:** Thus the program singly linked list written in C language was implemented and executed successfully.

**Ex.No: 2b DOUBLY LINKED LIST IMPLEMENTATION**

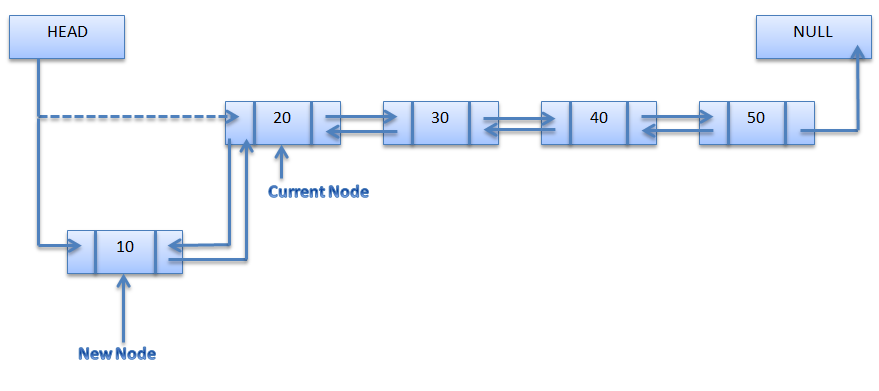
**Aim:** To write a C program to implement doubly linked list.

**Algorithm:**

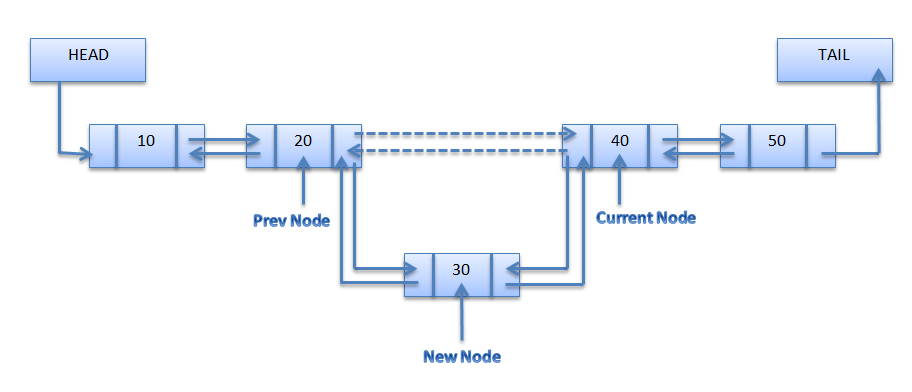
**Insertion**

To insert the new node at the rear of the list, we have to set three pointers: the prev pointer in newNode, the next pointer in the current last node in the list, and tail, which needs to be updated to point to the new last node in the list.

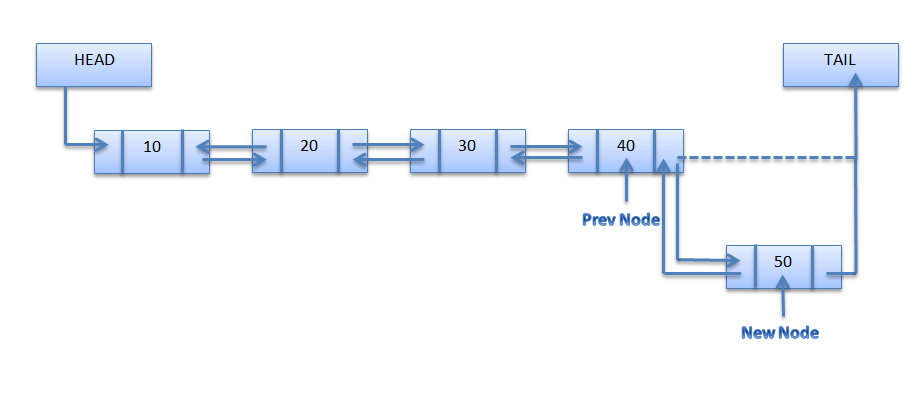
**Insertion at the head**



**Insertion at the middle**



**Insertion at the end**



**Deletion**

1. Input the POS(Position)

2. Initialize TEMP = START; i = 0

3. Repeat the step 4 if (i less than POS) and (TEMP is not equal to NULL)

4. TEMP = TEMP → RPoint; i = i +1

5. If (TEMP not equal to NULL) and (i equal to POS)

(a) Create a New Node

(b) NewNode → DATA = DATA

(c) NewNode → RPoint = TEMP → RPoint

(d) NewNode → LPoint = TEMP

(e) (TEMP → RPoint) → LPoint = NewNode

(f ) TEMP → RPoint = New Node

6. Else (a) Display “Position NOT found”

**Program:**

*#include"stdio.h"*

*#include"alloc.h"*

*struct node*

*{*

*struct node \*prev;*

*int data;*

*struct node \*next;*

*};*

*struct node \*start;*

*void insertbeg(void) // insert element at the beginning of DLL*

*{*

*int a;*

*struct node \*nn;*

*nn=(struct node \*)malloc(sizeof(struct node));*

*printf("enter data:");*

*scanf("%d",&nn->data);*

*a=nn->data;*

*if(start==NULL) //If Linked list is empty*

*{*

*nn->prev=nn->next=NULL;*

*start=nn;*

*}*

*else*

*{*

*nn->next=start;*

*nn->prev=NULL;*

*start->prev=nn;*

*start=nn;*

*}*

*printf("%d succ. inserted\n",a);*

*return;*

*}*

*void insertend(void) // insert element at the End of DLL*

*{*

*int b;*

*struct node \*nn,\*lp;*

*nn=(struct node \*)malloc(sizeof(struct node));*

*printf("enter data:");*

*scanf("%d",&nn->data);*

*b=nn->data;*

*if(start==NULL)*

*{*

*nn->prev=nn->next=NULL;*

*start=nn;*

*}*

*else*

*{*

*lp=start;*

*while(lp->next!=NULL)*

*{*

*lp=lp->next;*

*}*

*nn->prev=lp;*

*lp->next=nn;*

*nn->next=NULL;*

*}*

*printf("%d succ. inserted\n",b);*

*return;*

*}*

*void insertmid(void) // insert element before the given element of DLL*

*{*

*struct node \*nn,\*temp,\*ptemp;int x,c;*

*if(start==NULL)*

*{*

*printf("dll is empty\n"); return;*

*}*

*printf("enter data before which new node is to be insreted\n");*

*scanf("%d",&x);*

*if(x==start->data)*

*{*

*insertbeg();*

*return;*

*}*

*ptemp=start;*

*temp=start->next;*

*while(temp!=NULL&&temp->data!=x)*

*{*

*ptemp=temp;*

*temp=temp->next;*

*}*

*if(temp==NULL)*

*{*

*printf("%d data does not exist\n",x);*

*}*

*else*

*{*

*nn=(struct node \*)malloc(sizeof(struct node));*

*printf("enter data");*

*scanf("%d",&nn->data);*

*c=nn->data;*

*nn->prev=ptemp;*

*nn->next=temp;*

*ptemp->next=nn;*

*temp->prev=nn;*

*printf("%d succ. inserted\n",c);*

*}*

*return;*

*}*

*void deletion(void)*

*{*

*struct node \*pt,\*t,\*nt;*

*int x;*

*if(start==NULL)*

*{*

*printf("dll is empty\n");*

*return;*

*}*

*printf("enter data to be deleted");*

*scanf("%d",&x);*

*if(x==start->data)*

*{*

*t=start;*

*start=start->next;*

*free(t);*

*printf("%d is succ. deleted\n",x);*

*if(start!=NULL)*

*{*

*start->prev=NULL;*

*}*

*return;*

*}*

*pt=start;*

*t=start->next;*

*while(t!=NULL&&t->data!=x)*

*{*

*pt=t;*

*t=t->next;*

*}*

*if(t==NULL)*

*{*

*printf("%d does not exist\n",x);return;*

*}*

*else*

*{*

*pt->next=t->next;*

*if(t->next!=NULL)*

*{*

*nt=t->next;*

*nt->prev=pt;*

*}*

*free(t);*

*}*

*printf("%d is succ. deleted\n",x);*

*return;*

*}*

*void display(void)*

*{*

*struct node \*temp;*

*if(start==NULL)*

*{*

*printf("dll is empty\n");*

*return;*

*}*

*printf("displaying in forword order\n");*

*temp=start;*

*while(temp!=NULL)*

*{*

*printf("%d\n",temp->data);*

*temp=temp->next;*

*}*

*return;*

*}*

*void main()*

*{*

*int c,a;     start=NULL;*

*do*

*{*

*printf("1:insert\n2:delete\n3:display\n4:exit\nenter choice:");*

*scanf("%d",&c);*

*switch(c)*

*{*

*case 1:*

*printf("1:insertbeg\n2:insert end\n3:insert mid\nenter choice:");*

*scanf("%d",&a);*

*switch(a)*

*{*

*case 1:insertbeg();break;*

*case 2:insertend();break;*

*case 3:insertmid();break;*

*}*

*break;*

*case 2:deletion();break;*

*case 3:display();break;*

*case 4:printf("program ends\n");break;*

*default:printf("wrong choice\n");*

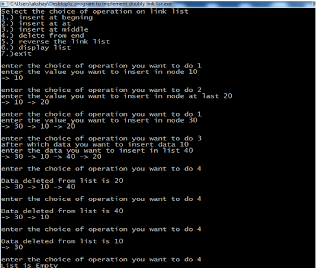
*break;*

*}*

*}while(c!=4) ;*

*}*

**Output:**



**Result:** Thus the program doubly linked list written in c was implemented and executed successfully.

**Ex. No. 2c CIRCULAR LINKED LIST**

**Aim:** To implement circular linked list.

**Algorithm:**

**(i) Insertion at Beginning:-**

Step1: Create a new node..

Step2: Assign the data to the data part of new node.

Step3: Now, assign the address part of head to the next part of new node.

Step4: Assign the address of new node to the address of head

**(ii) Insertion at given position:-**

Step1: Create a new node.

Step2: Assign the data to the data part of new node. Specify the position where the element to be inserted. Assume it is 3.

Step3: Now, traverse the list up to position 2 node.

Step4: Assign the next part of node 2 to next part of new node (it it node3).

Step5: Assign the address of new node to next part of node 2.

**(iii) Insertion at end:-**

Step1: Create a new node.

Step2: Assign the data to the data part of new node. Now, traverse the list till the last node

reached.

Step3: The next part of last node is assigned with address of new node.

Step5: Assign the next part of new node with head node..

***Program:***

*#include<stdio.h>*

*#include<conio.h>*

*struct node*

*{*

*int info;*

*struct node \*link;*

*};*

*struct node \*start,\*newptr;*

*}*

*void insertend(struct node \*n);*

*void insertmid(struct node \*o);*

*void insertbeg(struct node \*m);*

*void del(int d);*

*void display(struct node \*x);*

*int count(struct node \*c);*

*void main()*

*{*

*int info;*

*int a;*

*int c;*

*struct node \*np;*

*char ch;*

*start=NULL;*

*clrscr();*

*do*

*{*

*printf("ENTER YOUR CHOICE:\n");*

*printf("Enter 1 to insert at begining\n");*

*printf("Enter 2 to insert after a given element\n");*

*printf("Enter 3 to insert at end\n"):*

*printf("Enter 4 to delete a particular element\n");*

*scanf("%d",&a);*

*if(a==1)*

*{*

*printf("Enter the element\n");*

*scanf("%d",&info);*

*np=create\_new\_node(info);*

*insertbeg(np);*

*}*

*else if(a==2)*

*{*

*printf("Enter the element\n");*

*scanf("%d",&info);*

*np=create\_new\_node(info);*

*insertmid(np);*

*}*

*else if(a==3)*

*{*

*printf("Enter the element\n");*

*scanf("%d",&info);*

*np=create\_new\_node(info);*

*insertend(np);*

*}*

*else if(a==4)*

*{*

*printf("Enter the element\n");*

*scanf("%d",&info);*

*np=create\_new\_node(info);*

*del(info);*

*}*

*display(start);*

*c=count(start);*

*printf("The number of nodes are %d\n",c);*

*printf("\nDo you want to proceed? y/n\n");*

*scanf(" %c",&ch);*

*}*

*while(ch=='y');*

*getch();*

*}*

*/\*CREATION OF NEW NODE\*/*

*struct node \*create\_new\_node(int i)*

*{*

*newptr=(struct node \*)malloc(sizeof(struct node));*

*newptr->info=i;*

*newptr->link=NULL;*

*return newptr;*

*}*

*/\*INSERTION AT BEGINING\*/*

*void insertbeg(struct node \*m)*

*{*

*struct node \*temp;*

*if(start==NULL)*

*{*

*start=m;*

*m->link=start;*

*}*

*else*

*{*

*m->link=start;*

*temp=start;*

*while(temp->link!=start)*

*temp=temp->link;*

*temp->link=m;*

*start=m;*

*}*

*}*

*/\*INSERTION AT END\*/*

*void insertend(struct node \*n)*

*{*

*struct node \*temp;*

*if(start==NULL)*

*{*

*start=n;*

*n->link=start;*

*}*

*else*

*{*

*temp=start;*

*while(temp->link!=start)*

*temp=temp->link;*

*temp->link=n;*

*n->link=start;*

*}*

*}*

*/\*INSERTION AFTER A GIVEN ELEMENT\*/*

*void insertmid(struct node \*o)*

*{*

*int value;*

*struct node \*temp;*

*temp=start;*

*printf("Enter no after which new no has to be inserted:\n");*

*scanf("%d",&value);*

*while(temp!=start)*

*{*

*if(temp->info==value)*

*{*

*if(temp->link==start)*

*{*

*o->link=start;*

*temp->link=o;*

*}*

*else*

*{*

*o->link=temp->link;*

*temp->link=o;*

*}*

*}*

*temp=temp->link;*

*}*

*}*

*void del(int d)*

*{*

*struct node \*temp,\*ptr;*

*temp=start;*

*while(temp!=NULL)*

*{*

*if(temp->link->info==d)*

*{*

*ptr=temp->link;*

*temp->link=ptr->link;*

*free(ptr);*

*}*

*temp=temp->link;*

*}*

*}*

*void display(struct node \*x)*

*{*

*printf("\nCurrent linked list is:\t");*

*do*

*{*

*printf("%d\t",x->info);*

*x=x->link;*

*}*

*while(x!=start);*

*printf("\n");*

*}*

*int count(struct node \*c)*

*{*

*int co=0;*

*do*

*{*

*co++;*

*c=c->link;*

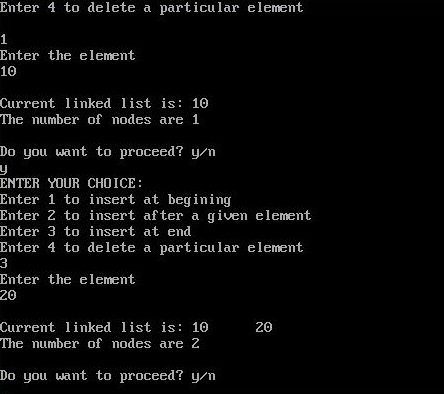
*}*

*while(c!=start);*

*return co;*

*}*

**Output:**



**Result:** Thus the circular linked list was implemented successfully.

**EX. NO. 3a Stack Using ARRAY**

**Aim:** To perform all stacks operation using array.

**Algorithm:**

Step 1: Initialize the integer variables.

Step 2: In a switch case, in case 1 the number to be pushed is got

Step 3:Top is now equal to top+1

Step 4: Else the stack is full

Step 5: In case 2 the number to be deleted is got

Step 6: Top is now equal to top-1

Step 7: Else the stack is empty

Step 8: In case 3 if the top is less than 0 the stack is full

Step 9: Else the stack is printed

Step 10: Default is no such choice

**Program:**

*#include<stdio.h>*

*main()*

*{*

*int a[10],top,add,delete,operation,display,i;*

*l1:*

*printf("Enter the operation to be performed:\n 1) push \n 2) pop \n 3) display \n 4) exit\n\n");*

*scanf("%d",&operation);*

*switch(operation)*

*{*

*case 1:*

*if(top<=9)*

*{*

*printf("Enter the number to be Added:\n");*

*scanf("%d",&add);*

*a[top]=add;*

*top=top+1;*

*}*

*else*

*{*

*printf("stack is full");*

*}*

*break;*

*case 2:*

*if(top>0)*

*{*

*printf("the number to be deleted is %d\n",a[top-1]);*

*top=top-1;*

*}*

*else*

*{*

*printf("stack is empty");*

*}*

*break;*

*case 3:*

*if(top<0)*

*{*

*printf("stack is full");*

*}*

*else*

*{*

*printf("The stack is\n");*

*for(i=0;i<top;i++)*

*{*

*printf("%d\n",a[i]);*

*}*

*}*

*break;*

*case 4:*

*goto l2;*

*break;*

*default:*

*printf("no such choice\n");*

*}*

*goto l1;*

*l2:printf("");}*

**Output:**

Enter the operation to be performed: 1) push 2) pop 3) display 4) exit

1

Enter the number to be Added: 11

Enter the operation to be performed: 1) push 2) pop 3) display 4) exit

1

Enter the number to be Added: 22

Enter the operation to be performed: 1) push 2) pop 3) display 4) exit

1

Enter the number to be Added: 33

Enter the operation to be performed: 1) push 2) pop 3) display 4) exit

1

Enter the number to be Added: 44

Enter the operation to be performed: 1) push 2) pop 3) display 4) exit

2

Enter the number to be Added: 44

Enter the operation to be performed: 1) push 2) pop 3) display 4) exit

3

The stack is 11 22 33

Enter the operation to be performed: 1) push 2) pop 3) display 4) exit

4

**Result:**

Stack using array was implemented successfully.

**EX.No. 3b STACK IMPLEMENTATION USING LINKED LIST**

**Aim:** To implement stack using linked list.

**Algorithm:**

Step 1: Initialize the integer variables and create a link.

Step 2: Top is now equal to top+1

Step 4: Else the stack is full

Step 5: In case 2 the number to be deleted is got

Step 6: Top is now equal to top-1

Step 7: Else the stack is empty

Step 8: In case 3 if the top is less than 0 the stack is full

Step 9: Else the stack is printed

Step 10: Default is no such choice

**Program:**

*#include <stdio.h>*

*#include <malloc.h>*

*#include<process.h>*

*typedef struct link\_tag*

*{*

*int data;*

*struct link\_tag \*link;*

*}node;*

*/\*\*\*\*\*\*\*\*\*\* Function Declaration begins \*\*\*\*\*\*\*\*\*\*/*

*node \*push(node \*);*

*node \*pop(node \*);*

*void display(node \*);*

*/\*\*\*\*\*\*\*\*\*\* Function Declaration ends \*\*\*\*\*\*\*\*\*\*/*

*void main()*

*{*

*node \*start=NULL;*

*int ch;*

*printf("\n\t\t Program of stack using linked list");*

*do*

*{*

*printf("\n\t\tMenu");*

*printf("\n\t\t1.Push");*

*printf("\n\t\t2.Pop");*

*printf("\n\t\t3.Display");*

*printf("\n\t\t4.Exit");*

*printf("\n\t\tEnter choice : ");*

*scanf("%d",&ch);*

*switch(ch)*

*{*

*case 1:*

*start = push(start);*

*break;*

*case 2:*

*start = pop(start);*

*break;*

*case 3:*

*printf("\n\t\*\*\*\* Stack \*\*\*\*\*\n");*

*display(start);*

*break;*

*case 4:*

*exit(0);*

*default:*

*printf("\nwrong choice:");*

*}*

*}*

*while (ch!=4);*

*printf("\n");*

*}*

*/\*\*\*\*\*\*\*\*\*\* Pushing an element in stack \*\*\*\*\*\*\*\*\*\*/*

*/\*\*\*\*\*\*\*\*\*\* Function Definition begins \*\*\*\*\*\*\*\*\*\*/*

*node \*push(node \*temp)*

*{*

*node \*new\_node;*

*int item;*

*printf("Enter an data to be pushed : ");*

*scanf("%d",&item);*

*new\_node = ( node \*)malloc(sizeof( node));*

*new\_node->data = item;*

*new\_node->link = temp;*

*temp = new\_node;*

*return(temp);*

*}*

*node \*pop(node \*p)*

*{*

*node \*temp;*

*if(p == NULL)*

*printf("\n\*\*\*\*\* Empty \*\*\*\*\*\n");*

*else*

*{*

*printf("Popped data = %d\n",p->data);*

*temp = p->link;*

*free(p);*

*p = temp;*

*if (p == NULL)*

*printf("\n\*\*\*\*\* Empty \*\*\*\*\*\n");*

*}*

*return(p);*

*}*

*void display(node \*seek)*

*{*

*printf("\nTop");*

*while (seek != NULL)*

*{*

*printf("-> %d",seek->data);*

*seek = seek->link;*

*}*

*printf("->NULL\n");*

*return;*

*}*

**Result:** Thus the program was executed successfully.

**Ex No:4a QUEUE IMPLEMENTATION USING ARRAYS**

**Aim:** To Write a C program to implement queue using arrays.

**Algorithm:**

**Inserting An Element Into The Queue**

1. Initialize front=0 rear = –1

2. Input the value to be inserted and assign to variable “data”

3. If (rear >= SIZE)

(a) Display “Queue overflow”

(b) Exit

4. Else (a) Rear = rear +1

5. Q[rear] = data

6. Exit

**Deleting An Element From Queue**

1. If (rear< front)

(a) Front = 0, rear = –1

(b) Display “The queue is empty”

(c) Exit

2. Else

(a) Data = Q[front]

3. Front = front +1

4. Exit

**Program:**

***/****\* Implentation of queue using arrays \*/*

*# include <stdio.h>*

*# define SIZE 10*

*int arr[ SIZE ], front = -1, rear = -1, i ;*

*void enqueue() ;*

*void dequeue() ;*

*void display() ;*

*int main()*

*{*

*int ch ;*

*do*

*{*

*printf( "\n[1].ENQUEUE [2].DEQUEUE [3].Display [4].Exit\n" ) ;*

*printf( "Enter your choice [1-4] : " ) ;*

*scanf( "%d", &ch ) ;*

*switch ( ch )*

*{*

*case 1 :*

*enqueue() ;*

*break ;*

*case 2 :*

*dequeue() ;*

*break ;*

*case 3 :*

*display() ;*

*break ;*

*case 4 :*

*break ;*

*default :*

*printf( "Invalid option\n" ) ;*

*}*

*}*

*while ( ch != 4 ) ;*

*}*

*void enqueue()*

*{*

*if ( rear == SIZE – 1 )*

*{*

*printf( "Queue is full (overflow)\n" ) ;*

*return ;*

*}*

*rear++ ;*

*printf( "Enter the element to ENQUEUE : " ) ;*

*scanf( "%d", &arr[ rear ] ) ;*

*if ( front == -1 )*

*front++ ;*

*}*

*void dequeue()*

*{*

*if ( front == -1 )*

*{*

*printf( "Queue is empty (underflow)\n" );*

*return ;*

*}*

*printf( "The DEQUEUE element is : %d\n", arr[ front ] ) ;*

*if ( front == rear )*

*front = rear = -1 ;*

*else*

*front++ ;*

*}*

*void display()*

*{*

*if ( front == -1 )*

*{*

*printf( "Queue is empty (underflow)\n" ) ;*

*return ;*

*} printf( "The elements in queue are : FRONT -> " ) ;*

*for ( i = front ; i <= rear ; i++ )*

*printf( " … %d", arr[ i ] ) ;*

*printf( " … <- REAR\n" ) ;*

*}*

**Output:**

1 Enqueue

2 Dequeue

3 Display

4 Exit

Enter the choice[1-4]: 2

Queue is empty (underflow)

Enter the choice[1-4]: 1

Enter the element to enqueue : 7

Enter the choice[1-4]: 1

Enter the element to enqueue : 6

Enter the choice[1-4]: 1

Enter the element to enqueue : 4

Enter the choice[1-4]: 3

The elements in queue are : front->7 ->6->4-> rear

Enter the choice[1-4]: 4

Exit

**Result:** Thus the Queue implementation using arrays written in C was successfully executed.

**Ex.No. 4b QUEUE IMPLEMENTATION USING LINKED LISTS**

**Aim:** To implement queue using linked list.

**Algorithm:**

**enQueue(q, x)**

1) While stack1 is not empty, push everything from satck1 to stack2.

2) Push x to stack1 (assuming size of stacks is unlimited).

3) Push everything back to stack1.

**dnQueue(q)**

1) If stack1 is empty then error

2) Pop an item from stack1 and return it

**Program:**

*#include <stdio.h>*

*#include <malloc.h>*

*#include<process.h>*

*typedef struct queue\_link*

*{*

*int data;*

*struct queue\_link \*link;*

*}node;*

*/\*\*\*\*\*\*\*\*\*\* Function Declaration begins \*\*\*\*\*\*\*\*\*\*/*

*void enqueue(node \*\*, node \*\*, int);*

*void dequeue(node \*\*);*

*void display(node \*);*

*/\*\*\*\*\*\*\*\*\*\* Function Declaration ends \*\*\*\*\*\*\*\*\*\*/*

*void main()*

*{*

*node \*front = NULL, \*rear = NULL;*

*int ch,item;*

*printf("\n\t\t Program of queue using linked list");*

*do*

*{*

*printf("\n\t\tMenu");*

*printf("\n\t\t1.enqueue");*

*printf("\n\t\t2.dequeue");*

*printf("\n\t\t3.display");*

*printf("\n\t\t4.exit");*

*printf("\n\t\tEnter choice : ");*

*scanf("%d",&ch);*

*switch(ch)*

*{*

*case 1:*

*printf("Enter an data to be enqueueed : ");*

*scanf("%d",&item);*

*enqueue(&front,&rear,item);*

*break;*

*case 2:*

*dequeue(&front);*

*break;*

*case 3:*

*printf("\n\t\*\*\*\* Queue \*\*\*\*\*\n");*

*display(front);*

*Structures Using C Lab Manual*

*break;*

*case 4:*

*exit(0);*

*default:*

*printf("\n wrong choice:");*

*}*

*}*

*while (ch!=4);*

*printf("\n");*

*}*

*void enqueue( node \*\*front,node \*\*rear,int item)*

*{*

*node \*new\_node;*

*new\_node = (node \*)malloc(sizeof( node));*

*new\_node->data = item;*

*new\_node->link = NULL;*

*if ((\*front) == NULL)*

*{*

*(\*front) = new\_node;*

*(\*rear) = new\_node;*

*}*

*else*

*{*

*(\*rear)->link = new\_node;*

*(\*rear) = new\_node;*

*}*

*return;*

*}*

*void dequeue(node \*\*front)*

*{*

*node \*temp;*

*if((\*front) != NULL)*

*{*

*temp = \*front;*

*(\*front) = (\*front)->link;*

*free(temp);*

*}*

*return;}*

*void display(node \*record)*

*{*

*printf("\nRoot");*

*while (record != NULL)*

*{*

*printf("-> %d",record->data);*

*record = (record->link);*

*}*

*printf("->NULL\n");*

*return;*

*}*

**Result:** Thus the program was implemented successfully.

**Ex.No. 5a Application of stack - Infix to postfix**

**Aim:**

To implement infix to postfix expression in C.

**Algorithm:**

Let Q be any infix expression and we have to convert it to postfix expression P. For this the following procedure will be followed.

 1. Push left parenthesis onto [STACK](http://www.thecrazyprogrammer.com/2013/12/c-program-for-array-representation-of-stack-push-pop-display.html) and add right parenthesis at the end of Q.

 2. Scan Q from left to right and repeat step 3 to 6 for each element of Q until the STACK is empty.

 3. If an operand is encountered add it to P.

 4. If a left parenthesis is encountered push it onto the STACK.

 5. If an operator is encountered, then

* Repeatedly pop from STACK and add to P each operator  
  which has same precedence as or higher precedence than the operator  
  encountered.
* Push the encountered operator onto the STACK.

6. If a right parenthesis is encountered, then

* Repeatedly pop from the STACK and add to P each operator  
  until a left parenthesis is encountered.
* Remove the left parenthesis; do not add it to P.

7. Exit

**Program:**

*#include<stdio.h>*

*#include<string.h>*

*#include<math.h>*

*int top;*

*char stack[30];*

*int isp(char c)*

*{*

*int t;*

*switch(c)*

*{*

*case '^': t=3; break;*

*case '/':*

*case '\*': t=2; break;*

*case '+':*

*case '-': t=1; break;*

*case '(': t=0; break;*

*default : c=1;*

*}*

*return(t);*

*}*

*int icp(char c)*

*{*

*int t;*

*switch(c)*

*{*

*case '^': t=4; break;*

*case '/':*

*case '\*': t=2; break;*

*case '+':*

*case '-': t=1; break;*

*case '(': t=4; break;*

*}*

*return(t);*

*}*

*void main()*

*{*

*int j=0,i,l;*

*char c,r,p[20]={" "},g[20];*

*clrscr();*

*printf("Enter the InFix Expression : ");*

*gets(g);*

*l=strlen(g);*

*g[l]=')'; g[l+1]='$'; top++;*

*stack[top]='(';*

*for(i=0;g[i]!='$';i++)*

*{*

*c=g[i];*

*if((c>='a' && c<='z') || (c>=0 && c<=9)) { j++; p[j]=c; }*

*else if(c=='(') { top++; stack[top]=c; }*

*else if(c==')')*

*{do{*

*r=stack[top];*

*top--;*

*p[++j]=r;*

*}while(stack[top]!='(');*

*top--;*

*}else*

*{*

*while(icp(c)<=isp(stack[top]))*

*{*

*r=stack[top];*

*top--;*

*p[++j]=r;*

*}*

*stack[++top]=c;*

*}}*

*printf("The PostFix Expression is : ");*

*puts(p);*

*getch();}*

**Output:**

Enter the InFix Expression : a+b/c-d

The PostFix Expression is : abc/+d- \*/

**Result:** Thus the program was executed successfully.

**Ex.No: 5b PRIORITY QUEUE**

**Aim:** To implement priority queue

**Algorithm:**

A prioriy queue P must support the following

**methods:**

**- size():**

Return the number of elements in P

**Input: None; Output: integer**

**- isEmpty():**

Test whether P is empty

**Input: None; Output: boolean**

**- insertItem(k,e):**

Insert a new element e with key k into P

**Input: Objects k, e Output: None**

**- minElement():**

Return (but don’t remove) an element of

P with smallest key; an error occurs if P

is empty.

**Input: None; Output: Object e**

**Program:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 5

void insert\_by\_priority(int);

void delete\_by\_priority(int);

void create();

void check(int);

void display\_pqueue();

int pri\_que[MAX];

int front, rear;

void main()

{

int n, ch;

printf("\n1 - Insert an element into queue");

printf("\n2 - Delete an element from queue");

printf("\n3 - Display queue elements");

printf("\n4 - Exit");

create();

while (1)

{

printf("\nEnter your choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("\nEnter value to be inserted : ");

scanf("%d",&n);

insert\_by\_priority(n);

break;

case 2:

printf("\nEnter value to delete : ");

scanf("%d",&n);

delete\_by\_priority(n);

break;

case 3:

display\_pqueue();

break;

case 4:

exit(0);

default:

printf("\nChoice is incorrect, Enter a correct choice");

}

}

}

/\* Function to create an empty priority queue \*/

void create()

{

front = rear = -1;

}

/\* Function to insert value into priority queue \*/

void insert\_by\_priority(int data)

{

if (rear >= MAX - 1)

{

printf("\nQueue overflow no more elements can be inserted");

return;

}

if ((front == -1) && (rear == -1))

{

front++;

rear++;

pri\_que[rear] = data;

return;

}

else

check(data);

rear++;

}

/\* Function to check priority and place element \*/

void check(int data)

{

int i,j;

for (i = 0; i <= rear; i++)

{

if (data >= pri\_que[i])

{

for (j = rear + 1; j > i; j--)

{

pri\_que[j] = pri\_que[j - 1];

}

pri\_que[i] = data;

return;

}

}

pri\_que[i] = data;

}

/\* Function to delete an element from queue \*/

void delete\_by\_priority(int data)

{

int i;

if ((front==-1) && (rear==-1))

{

printf("\nQueue is empty no elements to delete");

return;

}

for (i = 0; i <= rear; i++)

{

if (data == pri\_que[i])

{

for (; i < rear; i++)

{

pri\_que[i] = pri\_que[i + 1];

}

pri\_que[i] = -99;

rear--;

if (rear == -1)

front = -1;

return;

}

}

printf("\n%d not found in queue to delete", data);

}

/\* Function to display queue elements \*/

void display\_pqueue()

{

if ((front == -1) && (rear == -1))

{

printf("\nQueue is empty");

return;

}

for (; front <= rear; front++)

{

printf(" %d ", pri\_que[front]);

}

front = 0;

}

**Output:**

1 - Insert an element into queue

2 - Delete an element from queue

3 - Display queue elements

4 - Exit

Enter your choice : 1

Enter value to be inserted : 20

Enter your choice : 1

Enter value to be inserted : 45

Enter your choice : 1

Enter value to be inserted : 89

Enter your choice : 3

89 45 20

Enter your choice : 1

Enter value to be inserted : 56

Enter your choice : 3

89 56 45 20

Enter your choice : 2

Enter value to delete : 45

Enter your choice : 3

89 56 20

Enter your choice : 4

**Result:** Thus the program was implemented successfully.

**Ex.No: 6 BINARY TREE TRAVERSAL**

**Aim:**To write a C program to implementthe traversal of binary tree.

**Algorithm:**

**Step 1:**Start

**Step** 2:Read the value of ch

**Step** 3:If ch=1 read the values as numbers

**Step** 4:While (num!=0) call insert (tree, num)

**Step** 5:If ch=2 call the inorder to prefer inorder traversal

**Step** 6:If ch=3 call the post order to prefer post order traversal

**Step** 7:If ch=4 call the pre order to prefer pre order traversal

**Step** 8:If ch=5 exit the operations

**Step** 9:Repeat while ch!=5)

**Step**10:End

**Algorithm For Insert**

**Step** 1:Start

**Step** 2:If ( tree –null) allocate memory space to tree

**Step** 3:Iassign num to true ->item

**Step** 4:Set null to tree ->child and tree->rchild

**Step** 5:If( num < tree -> item) insert the num on the left side of node by replacing the step

**Step** 6:Else (num > tree -> item) insert the num on the right side of node and repeat the

step 5

**Step** 7:Else write duplicate value

**Step** 8:Return(tee)

**Step** 9:Stop

**Algorithm For In Order**

**Step** 1:Start

**Step** 2:If ( tree!= null)

**Step** 3:Visit the left child

**Step** 4:Write tree ->item

**Step** 5:Visit the right child

**Step** 6:Stop

**Algorithm For Pre - Order**

**Step 1:** Start

**Step** 2:If true!=null

**Step** 3:Write tree ->item

**Step** 4:Visit the left child

**Step** 5:Visit the right child

**Step** 6:Stop

**Algorithm For Post Order**

**Step** 1:Start

**Step** 2:If (true !=null)

**Step** 3Visit the left child

**Step** 4:Visit the right child

**Step** 5:Write tree -> item

**Step** 6:Stop

**Program:**

*#include<stdio.h>*

*#include<conio.h>*

*#include<alloc.h>*

*struct node*

*{*

*int val;*

*struct node \*lptr,\*rptr;*

*};*

*struct node\*str;*

*int ch;*

*void main()*

*{*

*void create(void);*

*void display(void);*

*str=NULL;*

*clrscr();*

*do*

*{*

*printf("\n\t BINARY TREE TRAVERSAL\n");*

*printf("\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");*

*printf("\n 1.CREATE \n");*

*printf("\n 2.DISPLAY\n");*

*printf("\n Enter ur choice:");*

*scanf("%d",&ch);*

*switch(ch)*

*{*

*case 1:*

*create();*

*break;*

*case 2:*

*display();*

*break;*

*}*

*}*

*while(ch!=2);*

*}*

*void create()*

*{*

*struct node \*temp,\*prev;*

*int c,n;*

*printf("\n Enter the number of elements:");*

*scanf("%d",&n);*

*printf("\n Enter the node elements:\n");*

*do*

*{*

*scanf("%d",&c);*

*temp=str;*

*if(temp==NULL)*

*{*

*str=(struct node\*)malloc(sizeof(struct node));*

*temp=str;*

*}*

*else*

*{*

*while(temp!=NULL)*

*{*

*prev=temp;*

*if(c<temp->val)*

*temp=temp->lptr;*

*else*

*temp=temp->rptr;*

*}*

*temp=(struct node\*)malloc(sizeof(struct node));*

*if(c<prev->val)*

*prev->lptr=temp;*

*else*

*prev->rptr=temp;*

*}*

*temp->val=c;*

*temp->lptr=NULL;*

*temp->rptr=NULL;*

*n--;*

*}while(n>0);*

*}*

*void in(struct node \*str)*

*{*

*if(str!=NULL)*

*{*

*in(str->lptr);*

*printf("\t%d",str->val);*

*in(str->rptr);*

*}*

*}*

*void pre(struct node \*str)*

*{*

*if(str!=NULL)*

*{*

*printf("\t%d",str->val);*

*pre(str->lptr);*

*pre(str->rptr);*

*}*

*}*

*void post(struct node \*str)*

*{*

*if(str!=NULL)*

*{*

*post(str->lptr);*

*post(str->rptr);*

*printf("\t%d",str->val);*

*}*

*}*

*void display()*

*{*

*void in(struct node \*p);*

*void pre(struct node \*p);*

*void post(struct node \*p);*

*do*

*{*

*printf("\n\n 1.INORDER \n");*

*printf("\n 2.PREORDER\n");*

*printf("\n 3.POSTORDER\n");*

*printf("\n 4.EXIT\n");*

*printf("\n Enter ur choice:");*

*scanf("%d",&ch);*

*switch(ch)*

*{*

*case 1:*

*in(str);*

*break;*

*case 2:*

*pre(str);*

*break;*

*case 3:*

*post(str);*

*break;*

*case 4:*

*exit(0);*

*}*

*}while(ch!=4);*

*}*

**Output:**

*1.CREATE*

*2.DISPLAY*

*Enter ur choice:1*

*Enter the number of elements:5*

*Enter the node elements:*

*10*

*5*

*3*

*9*

*23*

BINARY TREE TRAVERSAL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1.CREATE

2.DISPLAY

Enter ur choice:2

1.INORDER

2.PREORDER

3.POSTORDER

4.EXIT

Enter ur choice:1

3 5 9 10 23

1.INORDER

2.PREORDER

3.POSTORDER

4.EXIT

Enter ur choice:2

10 5 3 9 23

1.INORDER

2.PREORDER

3.POSTORDER

4.EXIT

Enter ur choice:3

3 9 5 23 10

1.INORDER

2.PREORDER

3.POSTORDER

4.EXIT

Enter ur choice:4

**Result:** Thus the program binary tree traversal written in c was implemented and executed successfully.

**EX. No.7 BINARY SEARCH TREE**

**Aim:** To implement binary search tree.

**Algorithm:**

**Procedure binary search**

A ← sorted array

n ← size of array

x ← value to be searched

Set lowerBound = 1

Set upperBound = n

while x not found

if upperBound < lowerBound

EXIT: x does not exists.

set midPoint = lowerBound + ( upperBound - lowerBound ) / 2

if A[midPoint] < x

set lowerBound = midPoint + 1

if A[midPoint] > x

set upperBound = midPoint - 1

if A[midPoint] = x

EXIT: x found at location midPoint

end while

end procedure

**Program:**

*#include<stdio.h>*

*#include<conio.h>*

*#include<alloc.h>*

*struct node*

*{*

*struct node \*left ;*

*char data ;*

*struct node \*right ;*

*} ;*

*struct node \*root;*

*struct node \* buildtree(int);*

*char arr[ ] = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', '\0', '\0', 'H' } ;*

*int lc[ ] = { 1, 3, 5, -1, 9, -1, -1, -1, -1, -1 } ;*

*int rc[ ] = { 2, 4, 6, -1, -1, -1, -1, -1, -1, -1 } ;*

*void display();*

*void main( )*

*{*

*int ch;*

*clrscr();*

*do*

*{*

*printf("1. Create.......\n");*

*printf("2. Display......\n");*

*printf("3. Exit.......\n");*

*printf("enter your choice(1..3)\n");*

*scanf("%d",&ch);*

*switch(ch)*

*{*

*case 1:*

*root = buildtree(0);*

*break;*

*case 2:*

*display();*

*break;*

*default :*

*break;*

*}getch();*

*} while(ch != 3);}*

*void display()*

*{void inorder(struct node \*);*

*void preorder(struct node \*);*

*void postorder(struct node \*);*

*int ch;*

*do*

*{*

*printf("1. Inorder.......\n");*

*printf("2. Preorder......\n");*

*printf("3. Postorder.....\n");*

*printf("4. Exit..........\n");*

*printf("enter your choice(1...4)");*

*scanf("%d",&ch);*

*switch(ch)*

*{*

*case 1:*

*inorder(root);*

*break;*

*case 2:*

*preorder(root);*

*break;*

*case 3:*

*postorder(root);*

*break;*

*default :*

*break;*

*}*

*getch();*

*} while(ch !=4);*

*}*

*struct node \*buildtree(int index)*

*{*

*struct node \*temp = NULL;*

*if(index != -1)*

*{*

*temp=(struct node \*)malloc(sizeof(struct node));*

*temp->left=buildtree(lc[index]);*

*temp->data=arr[index];*

*temp->right=buildtree(rc[index]);*

*}*

*return temp;*

*}*

*void inorder(struct node \*root)*

*{*

*if(root != NULL)*

*{*

*inorder(root->left);*

*printf("%c\t", root->data);*

*inorder(root->right);*

*}}*

*void preorder(struct node \*root)*

*{*

*if(root != NULL)*

*{*

*printf("%c\t", root->data);*

*preorder(root->left);*

*preorder(root->right);*

*}}*

*void postorder(struct node \*root)*

*{if(root != NULL){*

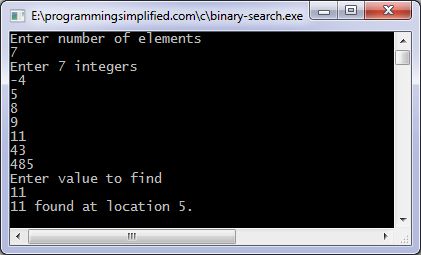
*preorder(root->left);*

*postorder(root->right);*

*printf("%c\t",root->data);*

*}}*

**Output:**



**Result:** Thus the program was implemented successfully.

**Ex.No.8 MINIMUM SPANNING TREE USING PRIMS ALGORITHM**

**Aim:** To implement minimum spanning tree using prims algorithm.

**Algorithm:**

**1)** Create a set mstSet that keeps track of vertices already included in MST.  
**2)** Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.  
**3)** While mstSet doesn’t include all vertices  
**a)** Pick a vertex u which is not there in mstSetand has minimum key value.  
**b)** Include uto mstSet.  
**c)** Update key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if weight of edge u-v is less than the previous key value of v, update the key value as weight of u-v

**Program:**

*#include<stdio.h>*

*#include<conio.h>*

*inta,b,u,v,n,i,j,ne=1;*

*intvisited[10]={0},min,mincost=0,cost[10][10];*

*voidmain()*

*{*

*clrscr();*

*printf("\n Enter the number of nodes:");*

*scanf("%d",&n);*

*printf("\n Enter the adjacency matrix:\n");*

*for(i=1;i<=n;i++)*

*for(j=1;j<=n;j++)*

*{*

*scanf("%d",&cost[i][j]);*

*if(cost[i][j]==0)*

*cost[i][j]=999;*

*}*

*visited[1]=1;*

*printf("\n");*

*while(ne<n)*

*{*

*for(i=1,min=999;i<=n;i++)*

*for(j=1;j<=n;j++)*

*if(cost[i][j]<min)*

*if(visited[i]!=0)*

*{*

*min=cost[i][j];*

*a=u=i;*

*b=v=j;*

*}*

*if(visited[u]==0 || visited[v]==0)*

*{*

*printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);*

*mincost+=min;*

*visited[b]=1;*

*}*

*cost[a][b]=cost[b][a]=999;*

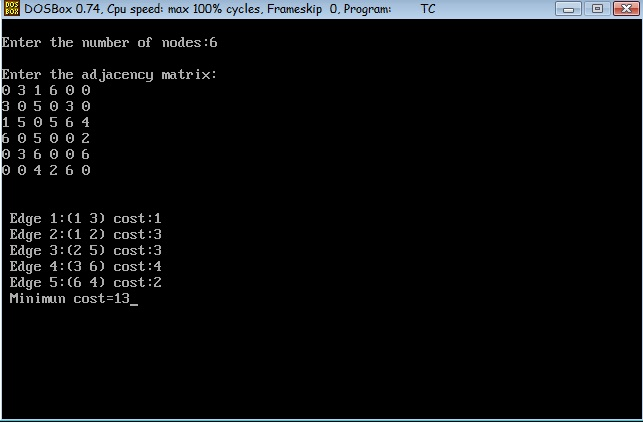
*}*

*printf("\n Minimun cost=%d",mincost);*

*getch();*

*}*

**Output:**



**Result:** Thus the program was implemented successfully.

**Ex.No. 9 SHORTEST PATH - DIJKSTRA ALGORITHM**

**Aim:** To implement shortest path algorithm – Dijkstra

**Algorithm:**

* 1. It maintains a list of unvisited vertices.
  2. It chooses a vertex (the source) and assigns a maximum possible cost (i.e. infinity) to every other vertex.
  3. The cost of the source remains zero as it actually takes nothing to reach from the source vertex to itself.
  4. In every subsequent step of the algorithm it tries to improve(minimize) the cost for each vertex. Here the cost can be distance, money or time taken to reach that vertex from the source vertex. The minimization of cost is a multi-step process.
  5. For each unvisited neighbor (vertex 2, vertex 3, vertex 4) of the current vertex (vertex 1) calculate the new cost from the vertex (vertex 1).
  6. When all the neighbors of the current node are considered, it marks the current node as visited and is removed from the unvisited list.
  7. Select a vertex from the list of unvisited nodes (which has the smallest cost) and repeat step4.

**Program:**

*void dijikstra(int G[MAX][MAX], int n, int startnode)*

*{*

*int cost[MAX][MAX], distance[MAX], pred[MAX];*

*int visited[MAX], count, mindistance, nextnode, i,j;*

*for(i=0;i < n;i++)*

*for(j=0;j < n;j++)*

*if(G[i][j]==0)*

*cost[i][j]=INFINITY;*

*else*

*cost[i][j]=G[i][j];*

*for(i=0;i< n;i++)*

*{*

*distance[i]=cost[startnode][i];*

*pred[i]=startnode;*

*visited[i]=0;*

*}*

*distance[startnode]=0;*

*visited[startnode]=1;*

*count=1;*

*while(count < n-1){*

*mindistance=INFINITY;*

*for(i=0;i < n;i++)*

*if(distance[i] < mindistance&&!visited[i])*

*{*

*mindistance=distance[i];*

*nextnode=i;*

*}*

*visited[nextnode]=1;*

*for(i=0;i < n;i++)*

*if(!visited[i])*

*if(mindistance+cost[nextnode][i] < distance[i])*

*{*

*distance[i]=mindistance+cost[nextnode][i];*

*pred[i]=nextnode;*

*}*

*count++;*

*}*

*for(i=0;i < n;i++)*

*if(i!=startnode)*

*{*

*printf("\nDistance of %d = %d", i, distance[i]);*

*printf("\nPath = %d", i);*

*j=i;*

*do*

*{*

*j=pred[j];*

*printf(" <-%d", j);*

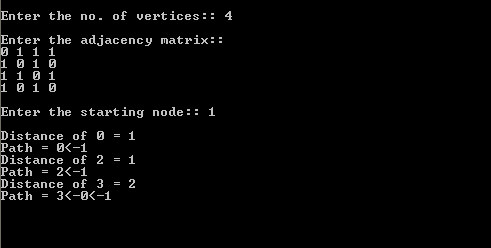
*}*

*while(j!=startnode);*

*}*

*}*

**Output:**



**Result:** Thus the program was executed successfully.