Assignment No : 1

Q1. Write a program to find mean of n numbers using arrays.

Program:

#include<stdio.h> int main()

{

int A[100],n,s=0,i; float m;

printf("\nEnter the size of the array:"); scanf("%d",&n);

printf("\nEnter the elements of the array:"); for(i=0;i<n;i++)

{

scanf("%d",&A[i]); s+=A[i];

}

m = (float)s/(float)n;

## printf("\nThe mean of the elements of the array is: %f",m);

return 0;

}

Output:

Enter the size of the array:5

Enter the elements of the array:15

19

21

45

76

The mean of the elements of the array is: 35.200001

Q2. Write a program to interchange the smallest &amp; largest no of n numbers using arrays.

Program:

#include<stdio.h> int main()

{

int A[100],n,max=0,min=0,max\_p=0,min\_p=0,i,j; printf("\nEnter the size of the array:"); scanf("%d",&n);

printf("\nenter the elements of the array:"); for(i=0;i<n;i++)

{

scanf("%d",&A[i]); if(max<=A[i])

{

max = A[i]; max\_p = i;

}

if(min>=A[i])

{ min = A[i]; min\_p = i;

}

}

printf("\n\nOriginal array:\n"); for(i =0;i<n;i++)

{ printf("%d ,",A[i]);

}

j = A[min\_p];

A[min\_p] = A[max\_p];

A[max\_p] = j;

printf("\n\nUpdated array:\n"); for(i =0;i<n;i++)

{ printf("%d ,",A[i]);

} return 0;

}

Output:

Enter the size of the array:5

enter the elements of the array:15

8

19

12

5

Original array: 15 ,8 ,19 ,12 ,5 Updated array:

19 ,8 ,15 ,12 ,5

Q3. Write a program to delete a no from an array which is already sorted in ascending order.

Program:

#include<stdio.h> int main()

{

int A[100],i,j,n,x,t;

printf("\nEnter the size of the array:"); scanf("%d",&n);

printf("\nEnter the elements of the array:"); for(i =0 ;i<n;i++)

{

scanf("%d",&A[i]);

}

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

{

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

{ if(A[i]<A[j])

{ t = A[i];

A[i] = A[j];

A[j] = t;

}

}

}

printf("\nOriginal array:\n"); for(i=0;i<n;i++)

{ printf("%d\t",A[i]);

}

printf("\nEnter the element to be deleted:"); scanf("%d",&x); for(i =0;i<n;i++)

{ if(A[i]==x)

{ t = i; break;

}

} for(i =t;i<n-1;i++)

{

A[i]=A[i+1];

}

printf("\narray after deletion:\n");

for(i=0;i<n-1;i++)

{ printf("%d\t",A[i]);

} return 0;

}

Output:

Enter the size of the array:5

Enter the elements of the array:25

12

4

8

10

Original array:

4 8 10 12 25

Enter the element to be deleted:10 array after deletion:

4 8 12 25

Q4. Write a program to read and display a 3\*3 matrix.

Program:

#include<stdio.h> int main()

{

int A[3][3],i,j;

printf("\nEnter the elements of the matrix:"); for(i = 0;i<3;i++)

{

for( j=0;j<3;j++)

{

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

}

}

printf("\n\n3X3 Matrix:\n"); for(i = 0;i<3;i++)

{ for( j=0;j<3;j++)

{ printf("%d\t",A[i][j]);

} printf("\n");

}

return 0;

}

Output:

Enter the elements of the matrix:2 3 4

6 7 8

1. 2 9

3X3 Matrix:

1. 3 4

6 7 8

1 2 9

Q5. Write a program to find transpose of a given matrix.

Program:

#include<stdio.h> int main()

{

int A[10][10],T[10][10],i,j,r,c;

printf("\nEnter the size of the matrix(rows and columns):"); scanf("%d%d",&r,&c);

printf("\nEnter the elements of the matrix:"); for(i = 0;i<r;i++)

{

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

{

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

T[j][i]= A[i][j];

}

}

printf("\n\nOriginal Matrix:\n");

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

{ for( j=0;j<c;j++)

{

printf("%d\t",A[i][j]);

} printf("\n");

}

printf("\n\nTranspose Matrix:\n"); for(i = 0;i<c;i++)

{ for( j=0;j<r;j++)

{

printf("%d\t",T[i][j]);

} printf("\n"); } return 0;

}

Output:

Enter the size of the matrix(rows and columns):3

3

Enter the elements of the matrix:4 5 0

2 3 6

9 4 1

Original Matrix:

4 5 0

2 3 6

9 4 1

Transpose Matrix:

1. 2 9
2. 3 4

0 6 1

Q6. Write a program to perform transpose of a sparse matrix.

Program:

#include<stdio.h> #include<stdlib.h> int main()

{

int a[3][3],spar[16][13],i,j,c=0,d; printf("enter the values for a matrix : \n"); for(i=0;i<3;i++){ for(j=0;j<3;j++){

printf("\nEnter element : "); scanf("%d",&a[i][j]);

}

}

for(i=0;i<3;i++){ for(j=0;j<3;j++){ if(a[i][j]!=0){ spar[c][0]=a[i][j]; spar[c][1]=i; spar[c][2]=j; c++;

}

}

}

printf("\n sparse matrix is \n "); for(i=0;i<c;i++){

printf("\n%d\t%d\t%d",spar[i][0],spar[i][1],spar[i][2]);

}

for(i=0;i<c;i++){ d=spar[i][1]; spar[i][1]=spar[i][2]; spar[i][2]=d;

}

printf("\n TRanspose of sparse matrix is \n "); for(i=0;i<c;i++){

printf("\n%d\t%d\t%d",spar[i][0],spar[i][1],spar[i][2]);

} return 0;

} Output:

enter the values for a matrix :

Enter element : 0

Enter element : 1

Enter element : 2

Enter element : 8

Enter element : 0

Enter element : 2

Enter element : 1

Enter element : 0 Enter element : 2 sparse matrix is

1. 0 1
2. 0 2

8 1 0

2 1 2

1. 2 0
2. 2 2

TRanspose of sparse matrix is

1. 1 0
2. 2 0

8 0 1

2 2 1

1. 0 2
2. 2 2

Q7. Write a program to perform addition of two sparse matrices.

Program:

#include<stdio.h> int main()

{

int A[10][10],B[10][10],SP\_1[100][3],SP\_2[100][3],SP\_3[100][3],i,j,c1=0,c2=0; int r,c,c3,m1,n1,m2,n2,a=0,b=0;

printf("\nEnter the size of matrix(row and columns): "); scanf("%d%d",&r,&c);

printf("\nEnter the elements of matrix:"); for(i=0;i<r;i++)

{

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

{

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

if(A[i][j]!=0)

{

SP\_1[c1][0] = A[i][j];

SP\_1[c1][1] = i;

SP\_1[c1++][2] = j;

}

}

}

printf("\nEnter elements of second matrix:"); for(i=0;i<r;i++)

{ for(j=0;j<c;j++)

{

scanf("%d",&B[i][j]); if(B[i][j]!=0)

{

SP\_2[c2][0] = B[i][j];

SP\_2[c2][1] = i;

SP\_2[c2++][2] = j;

}

}

}

printf("\nFirst Matrix :\n"); for(i=0;i<r;i++)

{ for(j=0;j<c;j++)

{

printf("%d\t",A[i][j]);

} printf("\n");

}

printf("\nSecond Matrix :\n"); for(i=0;i<r;i++)

{ for(j=0;j<c;j++)

{ printf("%d\t",B[i][j]);

} printf("\n");

}

printf("\nSparse Matrix 1:\n"); for(i=0;i<c1;i++)

{ for(j=0;j<3;j++)

{ printf("%d\t",SP\_1[i][j]);

} printf("\n");

}

printf("\nSparse Matrix 2:\n"); for(i=0;i<c2;i++)

{ for(j=0;j<3;j++)

{ printf("%d\t",SP\_2[i][j]); } printf("\n");

}

c3 = c1+c2;

for(i=0;i<c3;i++)

{ if(a<c1)

{

m1 = SP\_1[a][1]; n1 = SP\_1[a][2];

} else

{ m1 = c1+1; n1 = c1+1; } if(b<c2)

{

m2 = SP\_2[b][1]; n2 = SP\_2[b][2];

} else

{ m2 = c2+1; n2 = c2+1; }

if(m1 == m2 && n1==n2)

{

SP\_3[i][0] = SP\_1[a][0] + SP\_2[b][0];

SP\_3[i][1] = m1; SP\_3[i][2] = n1; c3--; a++; b++;

}

else if(m1 <m2)

{

SP\_3[i][0] = SP\_1[a][0];

SP\_3[i][1] = m1;

SP\_3[i][2] = n1;

a++;

}

else if(m2 < m1)

{

SP\_3[i][0] = SP\_2[b][0];

SP\_3[i][1] = m2;

SP\_3[i][2] = n2;

b++; }

else if(m1 == m2 && n1<n2)

{

SP\_3[i][0] = SP\_1[a][0];

SP\_3[i][1] = m1;

SP\_3[i][2] = n1;

a++;

}

else if(m1 == m2 && n2<n1)

{

SP\_3[i][0] = SP\_2[b][0];

SP\_3[i][1] = m2;

SP\_3[i][2] = n2;

b++;

}

}

printf("\n\nAddition of sparse matrix:\n"); for(i=0;i<c3;i++)

{ for(j=0;j<3;j++)

{

printf("%d\t",SP\_3[i][j]);

}

printf("\n");

}

return 0;

}

Output:

Enter the size of matrix(row and columns): 3

3

# Enter the elements of matrix:2 3 4

1. 6 2
2. 3 6

Enter elements of second matrix:8 5 2

1. 2 6

7 3 5

First Matrix :

2 3 4

1. 6 2
2. 3 6

Second Matrix :

1. 5 2
2. 2 6

7 3 5

Sparse Matrix 1:

1. 0 0
2. 0 1
3. 0 2

7 1 0

6 1 1

# 2 1 2

8 2 0

3 2 1

6 2 2

Sparse Matrix 2:

8 0 0

5 0 1

2 0 2

9 1 0

2 1 1

# 6 1 2

7 2 0

3 2 1

5 2 2

Addition of sparse matrix:

10 0 0

8 0 1

6 0 2

16 1 0

8 1 1

8 1 2

15 2 0

6 2 1

11 2 2

Q8. Write a program to perform addition of a two polynomial expressions using arrays.

Program:

#include<stdio.h> int main()

{

int P1[50],P2[50],P3[50],deg3,deg1,deg2,i,j=0,k=0; printf("\nEnter the highest degree of the first polynomial:"); scanf("%d",&deg1);

printf("Enter the first polynomial:"); for(i=0;i<deg1;i++)

{

printf("\nEnter the co-efficient of the term %d:",i+1); scanf("%d",&P1[i]);

}

printf("\nEnter the highest degree of the second polynomial:"); scanf("%d",&deg2);

printf("Enter the second polynomial:"); for(i=0;i<deg2;i++)

{

printf("\nEnter the co-efficient of the term %d:",i+1); scanf("%d",&P2[i]);

}

deg3 = (deg1>deg2)?deg1:deg2; for(i = 0;i<deg3;i++)

{

P3[i]=0; if(j<deg1 && j==i)

P3[i] += P1[j++];

if(k<deg2 && k==i)

P3[i] += P2[k++];

}

printf("\nFirst polynomial:\n"); for(i=deg1-1;i>=0;i--)

{ if(P1[i]!=0)

printf("%dX^%d + ",P1[i],i+1);

}

printf("\nSecond polynomial:\n"); for(i=deg2-1;i>=0;i--)

{ if(P2[i]!=0) printf("%dX^%d + ",P2[i],i+1);

}

printf("\nAddition of polynomials:\n"); for(i=deg3-1;i>=0;i--)

{ if(P3[i]!=0)

printf("%dX^%d + ",P3[i],i+1);

} return 0;

}

Output:

Enter the highest degree of the first polynomial:3 Enter the first polynomial:

Enter the co-efficient of the term 1:4

Enter the co-efficient of the term 2:6

Enter the co-efficient of the term 3:3 Enter the highest degree of the second polynomial:3 Enter the second polynomial:

Enter the co-efficient of the term 1:5

Enter the co-efficient of the term 2:2

Enter the co-efficient of the term 3:6 First polynomial: 3X^3 + 6X^2 + 4X^1 Second polynomial:

6X^3 + 2X^2 + 5X^1

Addition of polynomials:

9X^3 + 8X^2 + 9X^1

Assignment No : 2

Q1. Write a menu driven program to perform following operations on singly linked list: Create, Insert, Delete, and Display.

Program:

#include <stdio.h>

#include <stdlib.h> // Linked List Node struct node { int info;

struct node\* link;

};

struct node\* start = NULL;

// Function to create list with n nodes initially void createList()

{

if (start == NULL) {

int n;

printf("\nEnter the number of nodes: ");

scanf("%d", &n); if (n != 0) { int data;

struct node\* newnode;

struct node\* temp;

newnode = malloc(sizeof(struct node));

start = newnode; temp = start; printf("\nEnter number to" " be inserted : "); scanf("%d", &data);

start->info = data;

for (int i = 2; i <= n; i++) { newnode = malloc(sizeof(struct node));

temp->link = newnode; printf("\nEnter

number to" " be inserted : "); scanf("%d", &data); newnode->info = data; temp = temp->link;

}

}

printf("\nThe list is created\n"); } else

printf("\nThe list is already created\n"); }

// Function to traverse the linked list void traverse() { struct node\* temp;

// List is empty if (start == NULL) printf("\nList is empty\n");

// Else print the LL else { temp = start; while (temp != NULL) { printf("Data = %d\n", temp->info);

temp = temp->link;

}

}

}

// Function to insert at the front // of the linked list void insertAtFront() { int data; struct node\* temp; temp = malloc(sizeof(struct node)); printf("\nEnter number to" " be inserted : "); scanf("%d", &data); temp>info = data;

// Pointer of temp will be

// assigned to start temp>link = start; start = temp;

}

// Function to insert at the end of // the linked list void insertAtEnd() { int data; struct node \*temp, \*head; temp = malloc(sizeof(struct node));

// Enter the number printf("\nEnter number to"

" be inserted : "); scanf("%d", &data);

// Changes links temp->link = 0; temp>info = data; head = start; while (head>link != NULL) {

head = head->link;

}

head->link = temp;

}

// Function to insert at any specified // position in the linked list void insertAtPosition()

{ struct node \*temp, \*newnode;

int pos, data, i = 1; newnode = malloc(sizeof(struct node));

// Enter the position and data printf("\nEnter position and data :");

scanf("%d %d", &pos, &data);

// Change Links temp = start; newnode>info = data; newnode>link = 0; while (i < pos - 1) { temp = temp->link; i++;

}

newnode->link = temp->link; temp>link = newnode;

}

// Function to delete from the front // of the linked list void deleteFirst() { struct node\* temp; if (start == NULL) printf("\nList is empty\n"); else { temp = start; start = start>link; free(temp);

}

}

// Function to delete from the end // of the linked list void deleteEnd() { struct node \*temp, \*prevnode;

if (start == NULL)

printf("\nList is Empty\n"); else {

temp = start; while (temp->link != 0) { prevnode = temp;

temp = temp->link;

} free(temp); prevnode->link = 0;

}

}

// Function to delete from any specified // position from the linked list void deletePosition()

{

struct node \*temp, \*position; int i = 1, pos;

// If LL is empty if (start == NULL) printf("\nList is empty\n");

// Otherwise else {

printf("\nEnter index : ");

// Position to be deleted scanf("%d", &pos); position = malloc(sizeof(struct node));

temp = start;

// Traverse till position while (i < pos - 1) { temp = temp->link;

i++;

}

// Change Links position = temp->link;

temp->link = position->link;

// Free memory free(position);

}

}

// Driver Code int main() {

int choice;

while (1) {

printf("\n\t1

Display the

List\n"); printf("\t2 For insertion at" " starting\n"); printf("\t3 For insertion at" " end\n"); printf("\t4 For insertion at " "any

position\n"); printf("\t5 For deletion of " "first element\n"); printf("\t6 For deletion of "

"last element\n"); printf("\t7 For deletion of "

"element at any

position\n"); printf("\t12 To exit\n"); printf("\nEnte r Choice :\n");

scanf("%d", &choice);

switch (choice) { case 1:

traverse(); break; case 2: insertAtFront(); break; case 3:

insertAtEnd(); break; case 4: insertAtPosition(); break; case 5: deleteFirst(); break; case 6: deleteEnd(); break; case 7: deletePosition(); break; case 12: exit(1); break; default: printf("Incorrect Choice\n");

} } return 0; }

Q2.Write a menu driven program to perform following operations on singly linked list : Create, Remove, Search, Count, and Display

Program :

#include<stdio.h>

#include<stdlib.h>

struct node

{

int info;

struct node \*link;

};

struct node \*create\_list(struct node \*start); void

display(struct node \*start); void count(struct node \*start); void search(struct node \*start,int data); struct node \*addatbeg(struct node \*start,int data); struct node \*addatend(struct node \*start,int data); struct node \*reverse(struct node \*start);

int main()

{ struct node

\*start=NULL;

int choice,data,item,pos;

while(1)

{ printf("1.Create List\n");

printf("2.Display\n");

printf("3.Count\n"); printf("4.Search\n"); printf("5.Reverse\n");

printf("6.Quit\n\n");

printf("Enter your choice : "); scanf("%d",&choice);

switch(choice)

{ case 1:

start=create\_list(start);

break;

case 2: display(start); break; case 3: count(start); break; case 4: printf("Enter the element to be searched : ");

scanf("%d",&data); search(start,data);

break;

case 5: start=reverse(start);

break; case 6:

exit(1); default: printf("Wrong choice\n");

}/\*End of switch \*/

}/\*End of while \*/

return 0;

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

struct node \*create\_list(struct node \*start)

{ int i,n,data;

printf("Enter the number of nodes : ");

scanf("%d",&n); start=NULL; if(n==0) return start;

printf("Enter the element to be inserted : ");

scanf("%d",&data); start=addatbeg(start,data);

for(i=2;i<=n;i++)

{ printf("Enter the element to be inserted : "); scanf("%d",&data);

start=addatend(start,data);

}

return start;

}/\*End of create\_list()\*/

void display(struct node \*start)

{ struct node \*p; if(start==NULL)

{

printf("List is empty\n");

return; } p=start;

printf("List is

:\n");

while(p!=NULL

)

{ printf("%d ",p->info); p=p->link;

}

printf("\n\n");

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

void count(struct node \*start)

{ struct node \*p; int cnt=0; p=start;

while(p!=NULL)

{ p=p->link;

cnt++;

}

printf("Number of elements are %d\n",cnt);

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

void search(struct node \*start,int item)

{

struct node \*p=start; int pos=1;

while(p!=NULL)

{

if(p->info==item)

{

printf("Item %d found at position %d\n",item,pos);

return;

} p=p->link; pos++;

}

printf("Item %d not found in list\n",item);

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

struct node \*addatbeg(struct node \*start,int data)

{

struct node \*tmp;

tmp=(struct node \*)malloc(sizeof(struct node)); tmp->info=data; tmp->link=start; start=tmp; return start;

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

struct node \*addatend(struct node \*start,int data)

{

struct node \*p,\*tmp; tmp=(struct node \*)malloc(sizeof(struct node)); tmp-

>info=data;

p=start;

while(p->link!=NULL) p=p->link; p-

>link=tmp; tmp->link=NULL;

return start;

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

struct node \*reverse(struct node \*start)

{ struct node \*prev, \*ptr,

\*next; prev=NULL;

ptr=start;

while(ptr!=NULL)

{ next=ptr->link; ptr->link=prev; prev=ptr;

ptr=next; } start=prev; return start;

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

Q3.Write a menu driven program to perform operations on doubly linked list: Create, Insert, Delete, and Display

Program:

#include <stdio.h>

#include <stdlib.h>

// Linked List Node struct node { int info;

struct node \*prev, \*next;

};

struct node\* start = NULL;

// Function to traverse the linked list void traverse()

{

// List is empty if (start == NULL) { printf("\nList is empty\n"); return;

}

// Else print the Data struct node\* temp;

temp = start;

while (temp != NULL) { printf("Data = %d\n", temp>info); temp = temp->next;

}

}

// Function to insert at the front

// of the linked list void insertAtFront()

{ int data; struct node\* temp;

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

printf("\nEnter number to be inserted: "); scanf("%d", &data); temp->info = data; temp>prev = NULL;

// Pointer of temp will be

// assigned to start temp>next = start; start =

temp;

}

// Function to insert at the end of // the linked list void insertAtEnd() {

int data; struct node \*temp, \*trav;

temp = (struct node\*)malloc(sizeof(struct node)); temp->prev = NULL; temp->next = NULL; printf("\nEnter number to be inserted: "); scanf("%d", &data);

temp->info = data; temp->next = NULL;

trav = start;

// If start is NULL

if (start == NULL) {

start = temp;

}

// Changes Links else { while (trav->next != NULL) trav = trav->next; temp>prev = trav; trav->next =

temp;

}

}

// Function to insert at any specified // position in the linked list void insertAtPosition()

{

int data, pos, i = 1; struct node \*temp, \*newnode; newnode = malloc(sizeof(struct node)); newnode-

>next = NULL;

newnode->prev = NULL;

// Enter the position and data printf("\nEnter position : ");

scanf("%d", &pos);

// If start==NULL, if (start == NULL) {

start = newnode; newnode>prev = NULL; newnode->next

= NULL;

}

// If position==1, else if (pos == 1) {

// this is author method its correct but we can simply call insertAtfront() function for this special case

/\* newnode->next = start; newnode->next->prev = newnode; newnode->prev = NULL; start = newnode; \*/

// now this is improved by Jay Ghughriwala on geeksforgeeks insertAtFront();

}

// Change links

else { printf("\nEnter number to be inserted: ");

scanf("%d", &data); newnode->info = data; temp = start; while (i < pos - 1)

{ temp = temp->next; i++;

}

newnode->next = temp->next;

newnode->prev = temp; temp-

>next = newnode; temp->next->prev = newnode;

}

}

// Function to delete from the front

// of the linked list void deleteFirst() { struct node\* temp; if (start ==

NULL)

printf("\nList is empty\n");

else { temp = start; start = start->next; if (start != NULL) start->prev =

NULL;

free(temp);

}

}

// Function to delete from the end

// of the linked list void deleteEnd()

{ struct node\* temp; if (start == NULL) printf("\nList is empty\n"); temp = start; while (temp>next != NULL) temp = temp->next; if (start->next == NULL) start = NULL;

else { temp->prev->next = NULL;

free(temp);

}

}

// Function to delete from any specified // position from the linked list void deletePosition()

{

int pos, i = 1;

struct node \*temp, \*position;

temp = start;

// If DLL is empty if (start == NULL)

printf("\nList is empty\n");

// Otherwise else {

// Position to be deleted printf("\nEnter position : ");

scanf("%d", &pos);

// If the position is the first node if (pos == 1) {

deleteFirst(); // im,proved by Jay Ghughriwala on GeeksforGeeks

if (start != NULL) {

start->prev = NULL;

} free(position);

return;

}

// Traverse till position while (i < pos - 1) {

temp = temp->next;

i++;

}

// Change Links position = temp->next; if (position->next != NULL) position->next->prev = temp; temp->next = position>next;

// Free memory free(position);

}

}

// Driver Code int main() { int choice;

while (1) {

printf("\n\t1 Display the List\n"); printf("\t2 For insertion at" " starting\n"); printf("\t3 For insertion at" " end\n"); printf("\t4 For insertion at " "any position\n"); printf("\t5 For deletion of " "first element\n"); printf("\t6 For deletion of " "last element\n"); printf("\t7 For deletion of " "element at any position\n"); printf("\t8 To exit\n"); printf("\nEnter Choice

:\n"); scanf("%d", &choice);

switch (choice) { case 1: traverse(); break; case 2: insertAtFront(); break; case 3: insertAtEnd(); break; case 4: insertAtPosition(); break; case 5:

deleteFirst(); break; case 6: deleteEnd(); break; case 7:

deletePosition(); break; case 8:

exit(1); break; default: printf("Incorrect Choice. Try Again \n"); continue;

}

} return0; }

Q4.Implement circular linked list and perform operations on it: Create, Insert, Delete, and Display.

Program :

#include<stdio.h> #include<stdlib.h> struct

node {

int data;

struct node \*next;

};

struct node \*head;

void beginsert ();

void lastinsert (); void randominsert(); void begin\_delete(); void last\_delete(); void random\_delete();

void display(); void search(); void main ()

{ int choice

=0; while(choice != 7)

{

printf("\n\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\n");

printf("\nChoose one option from the following list ...\n");

printf("\n1.Insert in begining\n2.Insert at last\n3.Delete from Beginning\n4.Delete from last\n5.Search for an

element\n6.Show\n7.Exit\n");

printf("\nEnter your choice?\n");

scanf("\n%d",&choice); switch(choice)

{ case 1: beginsert(); break; case 2:

lastinsert(); break; case 3:

begin\_delete();

break; case 4: last\_delete(); break; case

5: search();

break; case 6:

display(); break; case

7: exit(0); break; default: printf("Please enter valid choice..");

}

} } void beginsert()

{

struct node \*ptr,\*temp;

int item; ptr = (struct node \*)malloc(sizeof(struct node));

if(ptr == NULL)

{

printf("\nOVERFLOW");

} else

{

printf("\nEnter the node data?"); scanf("%d",&item); ptr -> data

= item; if(head == NULL)

{ head = ptr;

ptr -> next = head;

} else

{ temp = head; while(temp->next != head) temp = temp->next; ptr>next = head; temp -> next = ptr; head = ptr;

}

printf("\nnode inserted\n");

}

}

void lastinsert()

{

struct node \*ptr,\*temp;

int item; ptr = (struct node \*)malloc(sizeof(struct node));

if(ptr == NULL)

{

printf("\nOVERFLOW\n");

} else

{ printf("\nEnter Data?"); scanf("%d",&item); ptr->data = item; if(head == NULL)

{ head = ptr;

ptr -> next = head;

} else

{ temp = head; while(temp -> next != head)

{

temp = temp -> next; } temp ->

next = ptr; ptr -> next = head;

}

printf("\nnode inserted\n");

}

}

void begin\_delete()

{ struct node

\*ptr; if(head == NULL)

{

printf("\nUNDERFLOW");

}

else if(head->next == head)

{ head = NULL; free(head); printf("\nnode deleted\n"); } else { ptr = head; while(ptr -> next != head) ptr = ptr -> next; ptr->next = head>next; free(head); head = ptr->next; printf("\nnode deleted\n");

} } void last\_delete() { struct node \*ptr, \*preptr;

if(head==NULL)

{

printf("\nUNDERFLOW");

}

else if (head ->next == head)

{ head = NULL; free(head);

printf("\nnode deleted\n");

} else { ptr = head;

while(ptr ->next != head)

{ preptr=ptr; ptr

= ptr->next;

}

preptr->next = ptr -> next; free(ptr); printf("\nnode deleted\n");

}

}

void search() { struct node \*ptr; int item,i=0,flag=1; ptr = head; if(ptr

== NULL)

{

printf("\nEmpty List\n");

} else {

printf("\nEnter item which you want to search?\n"); scanf("%d",&item); if(head ->data == item)

{

printf("item found at location %d",i+1); flag=0; } else

{

while (ptr->next != head)

{

if(ptr->data == item)

{

printf("item found at location %d ",i+1); flag=0;

break; } else { flag=1; } i++; ptr = ptr -> next;

} } if(flag != 0)

{

printf("Item not found\n");

}

}

}

void display() { struct node \*ptr; ptr=head;

if(head == NULL)

{

printf("\nnothing to print");

} else {

printf("\n printing values ... \n");

while(ptr -> next != head)

{

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

ptr = ptr -> next;

}

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

}

}

Q5. Represent polynomial as linked list and write driven program to preform addition evaluation

Program :

#include<stdio.h>

#include<stdlib.h>

typedef struct link {

int coeff;

int pow;

struct link \* next;

} my\_poly;

void my\_create\_poly(my\_poly \*\*); void

my\_show\_poly(my\_poly \*);

void my\_add\_poly(my\_poly \*\*, my\_poly \*, my\_poly \*);

int main(void) {

int ch; do {

my\_poly \* poly1, \* poly2, \* poly3;

printf("\nCreate 1st expression\n"); my\_create\_poly(&poly1);

printf("\nStored the 1st expression");

my\_show\_poly(poly1);

printf("\nCreate 2nd expression\n"); my\_create\_poly(&poly2);

printf("\nStored the 2nd expression");

my\_show\_poly(poly2);

my\_add\_poly(&poly3, poly1, poly2);

my\_show\_poly(poly3);

printf("\nAdd two more expressions? (Y = 1/N = 0): ");

scanf("%d", &ch);

} while (ch); return 0;

}

void my\_create\_poly(my\_poly \*\* node) {

int flag; //A flag to control the menu int

coeff, pow;

my\_poly \* tmp\_node; //To hold the temporary last address

tmp\_node = (my\_poly \*) malloc(sizeof(my\_poly)); //create the first node \*node = tmp\_node; //Store the head address to the reference variable do {

//Get the user data printf("\nEnter Coeff:"); scanf("%d", &coeff); tmp\_node>coeff = coeff; printf("\nEnter Pow:"); scanf("%d", &pow); tmp\_node->pow = pow;

//Done storing user data

//Now increase the Linked on user condition

tmp\_node->next = NULL;

//Ask user for continuation printf("\nContinue adding more terms to the polynomial list?(Y = 1/N = 0): ");

scanf("%d", &flag);

//printf("\nFLAG: %c\n", flag); //Grow the linked list on condition

if(flag) {

tmp\_node->next = (my\_poly \*) malloc(sizeof(my\_poly)); //Grow the list

tmp\_node = tmp\_node->next; tmp\_node->next = NULL;

}

} while (flag);

}

void my\_show\_poly(my\_poly \* node) {

printf("\nThe polynomial expression is:\n"); while(node != NULL) {

printf("%dx^%d", node->coeff, node->pow);

node = node->next;

if(node != NULL) printf(" + ");

}

}

void my\_add\_poly(my\_poly \*\* result, my\_poly \* poly1, my\_poly \* poly2) {

my\_poly \* tmp\_node; //Temporary storage for the linked list tmp\_node = (my\_poly \*) malloc(sizeof(my\_poly)); tmp\_node->next = NULL;

\*result = tmp\_node; //Copy the head address to the result linked list

//Loop while both of the linked lists have value while(poly1 && poly2) { if (poly1->pow > poly2->pow) { tmp\_node->pow = poly1->pow; tmp\_node->coeff = poly1->coeff;

poly1 = poly1->next;

}

else if (poly1->pow < poly2->pow) {

tmp\_node->pow = poly2->pow; tmp\_node->coeff = poly2->coeff;

poly2 = poly2->next;

}

else {

tmp\_node->pow = poly1->pow;

tmp\_node->coeff = poly1->coeff + poly2->coeff;

poly1 = poly1->next;

poly2 = poly2->next;

}

//Grow the linked list on condition if(poly1 && poly2) {

tmp\_node->next = (my\_poly \*) malloc(sizeof(my\_poly));

tmp\_node = tmp\_node->next; tmp\_node->next = NULL;

}

}

//Loop while either of the linked lists has value while(poly1 || poly2) {

//We have to create the list at beginning

//As the last while loop will not create any unnecessary node tmp\_node->next = (my\_poly \*) malloc(sizeof(my\_poly)); tmp\_node = tmp\_node->next; tmp\_node->next = NULL;

if(poly1) {

tmp\_node->pow = poly1->pow;

tmp\_node->coeff = poly1->coeff;

poly1 = poly1->next;

} if(poly2) {

tmp\_node->pow = poly2->pow;

tmp\_node->coeff = poly2->coeff;

poly2 = poly2->next;

}

}

printf("\nAddition Complete");

}

Assignment No : 3

Q1.Implement Stack using Array ADT.

## PROGRAM

#include <iostream> using namespace std; int stack[100], n=100, top=-1; void push(int val) { if(top>=n-1) cout<<"Stack Overflow"<<endl;

else { top++; stack[top]=val;

}

}

void pop() { if(top<=-1) cout<<"Stack Underflow"<<endl;

else {

cout<<"The popped element is "<< stack[top] <<endl;

top--;

}

}

void display() { if(top>=0) { cout<<"Stack elements are:"; for(int i=top; i>=0; i--) cout<<stack[i]<<" "; cout<<endl;

} else

cout<<"Stack is empty";

}

int main() { int ch, val; cout<<"1) Push in stack"<<endl; cout<<"2) Pop from stack"<<endl; cout<<"3) Display stack"<<endl; cout<<"4) Exit"<<endl;

do { cout<<"Enter choice: "<<endl; cin>>ch; switch(ch) { case 1: {

cout<<"Enter value to be pushed:"<<endl;

cin>>val; push(val); break;

} case 2: { pop(); break;

} case 3: { display(); break;

} case 4: { cout<<"Exit"<<endl; break;

}

default: { cout<<"Invalid Choice"<<endl;

}

}

}while(ch!=4); return 0;

}

## **OUTPUT**

Q2.Implement Stack using Linked List. Program

#include <iostream> using namespace std; struct Node { int data; struct Node \*next;

};

struct Node\* top = NULL; void push(int val) { struct Node\* newnode = (struct Node\*) malloc(sizeof(struct Node)); newnode->data = val; newnode->next = top; top = newnode;

}

void pop() { if(top==NULL) cout<<"Stack Underflow"<<endl;

else {

cout<<"The popped element is "<< top->data <<endl; top = top->next;

}

}

void display() { struct Node\* ptr; if(top==NULL) cout<<"stack is empty";

else { ptr = top; cout<<"Stack elements are: "; while (ptr != NULL) { cout<< ptr->data <<" "; ptr = ptr->next;

}

}

cout<<endl;

}

int main() { int ch, val; cout<<"1) Push in stack"<<endl; cout<<"2) Pop from stack"<<endl; cout<<"3) Display stack"<<endl; cout<<"4) Exit"<<endl;

do { cout<<"Enter choice: "<<endl; cin>>ch; switch(ch) { case 1: {

cout<<"Enter value to be pushed:"<<endl;

cin>>val; push(val); break;

} case 2: { pop(); break; } case 3: { display(); break;

} case 4: { cout<<"Exit"<<endl; break;

}

default: { cout<<"Invalid Choice"<<endl;

}

}

}while(ch!=4); return 0;

}

Q3.Implement stack as an ADT. Use this ADT to perform expression conversion. (Infix – Postfix)

## PROGRAM

#include<iostream> #include<stack> using namespace std; bool IsOperator(char); bool IsOperand(char); bool eqlOrhigher(char, char); string convert(string);

int main()

{

string infix\_expression, postfix\_expression;

int ch; do

{

cout << " Enter an infix expression: "; cin >> infix\_expression; postfix\_expression = convert(infix\_expression); cout << "\n Your Infix expression is: " << infix\_expression; cout << "\n Postfix expression is: " << postfix\_expression; cout << "\n \t Do you want to enter infix expression (1/ 0)?";

cin >> ch;

}

while(ch == 1); return 0;

}

bool IsOperator(char c)

{

if(c == '+' || c == '-' || c == '\*' || c == '/' || c == '^' ) return true; return false;

}

bool IsOperand(char c)

{

if( c >= 'A' && c <= 'Z') return true; if (c >= 'a' && c <= 'z') return true; if(c >= '0' && c <= '9') return true; return false;

}

int precedence(char op)

{ if(op == '+' || op == '-') return 1; if (op == '\*' || op == '/') return 2; if(op == '^') return 3; return 0;

}

bool eqlOrhigher (char op1, char op2)

{

int p1 = precedence(op1); int p2 = precedence(op2); if (p1 == p2)

{

if (op1 == '^' ) return false; return true;

}

return (p1>p2 ? true : false);

}

string convert(string infix)

{

stack <char> S; string postfix =""; char ch;

S.push( '(' ); infix += ')';

for(int i = 0; i<infix.length(); i++)

{ ch = infix[i];

if(ch == ' ') continue; else if(ch == '(') S.push(ch); else if(IsOperand(ch)) postfix += ch; else if(IsOperator(ch))

{

while(!S.empty() && eqlOrhigher(S.top(), ch))

{

postfix += S.top();

S.pop();

}

S.push(ch);

}

else if(ch == ')')

{

while(!S.empty() && S.top() != '(')

{

postfix += S.top();

S.pop();

}

S.pop();

}

}

return postfix;

}

Q4.Write a program for Expression Evaluation using Stack.

## PROGRAM:-

#include <iostream>

#include <string.h>

using namespace std;

struct Stack

{

int top; unsigned capacity; int\* array;

};

struct Stack\* createStack( unsigned capacity )

{

struct Stack\* stack = (struct Stack\*) malloc(sizeof(struct Stack));

if (!stack) return NULL;

stack->top = -1; stack->capacity = capacity; stack->array = (int\*) malloc(stack->capacity \* sizeof(int));

if (!stack->array) return NULL;

return stack;

}

int isEmpty(struct Stack\* stack)

{

return stack->top == -1 ;

}

char peek(struct Stack\* stack)

{

return stack->array[stack->top];

}

char pop(struct Stack\* stack)

{

if (!isEmpty(stack)) return stack->array[stack->top--] ; return '$';

}

void push(struct Stack\* stack, char op)

{

stack->array[++stack->top] = op;

}

int evaluatePostfix(char\* exp)

{

struct Stack\* stack = createStack(strlen(exp));

int i;

if (!stack) return -1; for (i = 0; exp[i]; ++i)

{

if (isdigit(exp[i])) push(stack, exp[i] - '0');

else

{

int val1 = pop(stack); int val2 = pop(stack); switch (exp[i])

{

case '+': push(stack, val2 + val1); break; case '-': push(stack, val2 - val1); break; case '\*': push(stack, val2 \* val1); break; case '/': push(stack, val2/val1); break;

}

}

}

return pop(stack);

}

int main()

{

char exp[] = "231\*+1234-"; cout<<"postfix evaluation: "<< evaluatePostfix(exp);

return 0;

}

### OUTPUT

Q5.Implement stack as an ADT. Use this ADT to perform expression conversion (Infix – Prefix)

## PROGRAM:-

#include <bits/stdc++.h> using namespace std;

bool isOperator(char c)

{

return (!isalpha(c) && !isdigit(c));

}

int getPriority(char C)

{ if (C == '-' || C == '+') return 1; else if (C == '\*' || C == '/') return 2; else if (C == '^') return 3; return 0;

}

string infixToPostfix(string infix)

{

infix = '(' + infix + ')'; int l = infix.size(); stack<char> char\_stack; string output;

for (int i = 0; i < l; i++)

{

if (isalpha(infix[i]) || isdigit(infix[i]))

output += infix[i]; else if (infix[i] == '(')

char\_stack.push('(');

else if (infix[i] == ')')

{

while (char\_stack.top() != '(')

{

output += char\_stack.top(); char\_stack.pop();

}

char\_stack.pop();

}

else

{

if (isOperator(char\_stack.top()))

{

while (getPriority(infix[i]) <= getPriority(char\_stack.top()))

{

output += char\_stack.top(); char\_stack.pop();

}

char\_stack.push(infix[i]);

}

}

}

return output;

}

string infixToPrefix(string infix)

{

int l = infix.size();

reverse(infix.begin(), infix.end());

for (int i = 0; i < l; i++) {

if (infix[i] == '(') { infix[i] = ')'; i++;

} else if (infix[i] == ')') { infix[i] = '('; i++;

}

}

string prefix = infixToPostfix(infix); reverse(prefix.begin(), prefix.end());

return prefix;

}

int main()

{

string s = ("(a-b/c)\*(a/k-l)"); cout << infixToPrefix(s) << std::endl; return 0;

}

Q6.Implement circular queue using arrays

## PROGRAM:-

#include <stdio.h> int queue[100]; int front=-1; int rear=-1;

void enqueue(int a)

{

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

{

front=0; rear=0; queue[rear]=a;

}

else if((rear+1)%100==front)

{

printf("Queue is overflow..");

}

else

{

rear=(rear+1)%100; queue[rear]=a;

}

}

int dequeue()

{

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

{

printf("\nQueue is underflow..");

}

else if(front==rear)

{

printf("\ndeleted"); front=-1; rear=-1;

}

else

{

printf("\ndeleted"); front=(front+1)%100;

}

}

void display()

{

int i=front; if(front==-1 && rear==-1)

{

printf("\n Queue is empty..");

} else

{

printf("\nelement in a Queue are\n :"); while(i<=rear)

{

printf("%d,", queue[i]); i=(i+1)%100;

}

}

}

int main()

{

int choice=1,x;

while(choice<4)

{

printf("\n Press 1: Insert an element"); printf("\nPress 2: Delete an element"); printf("\nPress 3: Display the element"); printf("\nEnter your choice"); scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Enter the element which is to be inserted"); scanf("%d", &x); enqueue(x); break; case 2: dequeue(); break; case 3: display();

}}

return 0;

}

Q7.Implement job scheduling algorithm using queue

## PROGRAM:-

#include<iostream> #include<bits/stdc++.h> using namespace std; struct Process

{ int pid; int bt; int priority;

};

bool comparison(Process a, Process b)

{

return (a.priority > b.priority);

}

void findWaitingTime(Process proc[], int n,int wt[])

{

wt[0] = 0; for (int i = 1; i < n ; i++ ) wt[i] = proc[i-1].bt + wt[i-1] ;

}

void findTurnAroundTime( Process proc[], int n,int wt[], int tat[])

{ for (int i = 0; i < n ; i++) tat[i] = proc[i].bt + wt[i];

}

void findavgTime(Process proc[], int n)

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0; findWaitingTime(proc, n, wt); findTurnAroundTime(proc, n, wt, tat); cout<< " Waiting time " << " Turn around time\n";

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

{

total\_wt = total\_wt + wt[i]; total\_tat = total\_tat + tat[i]; cout << " " << proc[i].pid << "\t\t"<< proc[i].bt << "\t " << wt[i]<< "\t\t " << tat[i] <<endl;

}

cout << "\nAverage waiting time = "<< (float)total\_wt / (float)n; cout << "\nAverage turn around time = "<< (float)total\_tat / (float)n;

}

void priorityScheduling(Process proc[], int n)

{

sort(proc, proc + n, comparison); cout<< "Order in which processes gets executed \n";

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

cout << proc[i].pid <<" " ; findavgTime(proc, n);

}

int main()

{

Process proc[] = {{1, 10, 2}, {2, 5, 0}, {3, 8, 1}}; int n = sizeof proc / sizeof proc[0]; priorityScheduling(proc, n); return 0;

}

Assignment 4

Q1) Create binary tree and perform recursive traversals.

Program:

#include<stdio.h>

#include<conio.h> #include<stdlib.h> struct Node

{

int data;

struct Node \*left,\*right;

};

struct node \*create()

{

struct Node \*temp; int data,choice;

temp=(struct Node \*)malloc(sizeof(struct Node)); printf("\nPress 0 to exit"); printf("\nPress 1 for new node"); printf("\nEnter your choice:"); scanf("%d",&choice);

if(choice==0)

{

return 0;

}

else

{

printf("\nEnter data:"); scanf("%d",&data);

temp->data=data;

printf("\nEnter left child of %d",data); temp->left=create();

printf("\nEnter right child of %d",data);

temp->right=create(); return temp;

}

}

void inorder(struct Node \*ptr)

{

if(ptr)

{

inorder(ptr->left); printf("%d\t",ptr->data); inorder(ptr->right);

}

}

void preorder(struct Node \*ptr)

{

if(ptr)

{

printf("%d\t",ptr->data); preorder(ptr->left);

preorder(ptr->right);

}

}

void postorder(struct Node \*ptr)

{

if(ptr)

{

postorder(ptr->left); postorder(ptr->right); printf("%d\t",ptr->data);

}

}

void main()

{

clrscr();

struct Node \*root;

struct Node \*ptr; root=create();

ptr=root;

printf("\nInorder Traversal is:\t");

inorder(ptr);

printf("\nPreorder Traversal is:\t"); preorder(ptr);

printf("\nPostorder Traversal is:\t");

postorder(ptr); getch();

}

Q2) Create a binary tree. Find height of the tree and print leaf nodes. Find mirror image, print original and mirror image using level-wise printing.

#include<stdio.h>

#include<conio.h> #include<stdlib.h> struct Node

{

int data;

struct node \*left,\*right;

};

struct Node \*Create()

{

struct Node \*temp; int data,choice;

temp=(struct Node \*)malloc(sizeof(struct Node)); printf("\nPress 0 to exit"); printf("\nPress 1 for new node"); printf("\nEnter your choice:"); scanf("%d",&choice);

if(choice==0)

{

return 0;

}

else

{

printf("\nEnter data:"); scanf("%d",&data);

temp->data=data;

printf("\nEnter left child of %d:",data);

temp->left=Create();

printf("\nEnter right child of %d:",data);

temp->right=Create(); return temp;

}

}

int Height(struct Node \*r)

{

if(r==NULL) return -1; else {

int lheight=Height(r->left); int rheight=Height(r->right); if(lheight>rheight)

return(lheight+1);

else

return(rheight+1);

}

}

void Leafnode(struct Node \*r)

{

if(r!=NULL)

{

if(r->right==NULL && r->left==NULL)

printf("%d\t",r->data); Leafnode(r->right);

Leafnode(r->left);

}

} void Level(struct Node \*r,int level)

{

if(r==NULL) return; else if(level==0)

printf("%d->",r->data);

else

{

Level(r->left,level-1);

Level(r->right,level-1);

}

}

void LevelOrder(struct Node \*r)

{

int h,i; if(!r) return; h=Height(r)+1; for(i=0;i<h;i++)

{

printf("\nLevel %d\n:",i);

Level(r,i);

}

printf("\nComplete level order traversal");

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

{

Level(r,i);

}

printf("\n");

}

void Inorder(struct Node \*r)

{

if(r)

{

Inorder(r->left); printf("%d\t",r->data);

Inorder(r->right);

}

}

void Mirror(struct Node \*r)

{

if(!r) return ;

Mirror(r->right); printf("\t%d",r->data);

Mirror(r->left);

}

void MirrorLevel(struct Node \*r,int level)

{

if(r==NULL) return; else if(level==0)

printf("%d->",r->data);

else

{

MirrorLevel(r->right,level-1);

MirrorLevel(r->left,level-1);

}

}

void MirrorLevelOrder(struct Node \*r)

{

int h,i; if(!r) return; h=Height(r)+1; for(i=0;i<h;i++)

{

printf("Level %d:",i); MirrorLevel(r,i); printf("\n");

}

printf("\nComplete level order traversal");

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

{

MirrorLevel(r,i);

}

printf("\n");

}

void main()

{

clrscr(); struct Node \*root; struct Node \*r; int level;

root=Create();

r=root;

printf("\nHeight is:"); printf("%d",Height(r)); printf("\nLeaf nodes are:"); Leafnode(r);

printf("\nLevel order of original tree:");

Level(r,level); LevelOrder(r);

printf("\nInorder Traversal is:\t"); Inorder(r);

printf("\nMirror image is:"); Mirror(r);

printf("\nLevel order of mirror image of tree:");

MirrorLevel(r,level); MirrorLevelOrder(r);

getch();

}

Assignment 5

Q1) Represent graph using adjacency list/adjacency matrix and perform Depth First Search.

Program:

#include<stdio.h> #include<conio.h> int a[20][20],visited[20],n; void dfs(int v)

{

int i; visited[v]=1; for (i=1;i<=n;i++)

if(a[v][i] && !visited[i])

{

printf("\n %d->%d",v,i);

dfs(i);

}

}

void main()

{

int i,j,count=0;

clrscr();

printf("\n Enter number of vertices:"); scanf("%d",&n);

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

{

visited[i]=0; for (j=1;j<=n;j++)

a[i][j]=0;

}

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

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

scanf("%d",&a[i][j]); dfs(1);

printf("\n");

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

{

if(visited[i])

count++;

}

if(count==n)

printf("\n Graph is connected"); else

printf("\n Graph is not connected");

getch();

}

Output:

Enter number of vertices:2 Enter the adjacency matrix:

1 1 1

1. 1 1

1->2

Graph is connected

Enter number of vertices:3 Enter the adjacency matrix:

1. 0 0

0 1 0

0 0 1

Graph is not connected

Q2) Represent graph using adjacency list/adjacency matrix and perform Breadth First Search.

Program:

#include<stdio.h> #include<conio.h>

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1; void bfs(int v)

{

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

if(a[v][i] && !visited[i])

q[++r]=i;

if(f<=r)

{

visited[q[f]]=1;

bfs(q[f++]);

}

}

void main()

{

int v;

clrscr();

printf("\n Enter the number of vertices:"); scanf("%d",&n);

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

{

q[i]=0;

visited[i]=0;

}

printf("\n Enter graph data in matrix form:\n");

for (i=1;i<=n;i++) for (j=1;j<=n;j++) scanf("%d",&a[i][j]);

printf("\n Enter the starting vertex:");

scanf("%d",&v);

bfs(v);

printf("\n The node which are reachable are:\n");

for (i=1;i<=n;i++) if(visited[i])

printf("%d\t",i);

else

printf("\n Bfs is not possible");

getch();

}

Output:

Enter the number of vertices:3 Enter graph data in matrix form:

# 1 1 0

# 0 1 1

1 1 1

Enter the starting vertex:1

The node which are reachable are:

1 2 3

Q3) Implement minimum cost spanning tree algorithm.

Program:

#include<stdio.h>

#include<stdlib.h> #include<conio.h>

int comparator(const void \*p1,const void \*p2)

{

const int (\*x)[3]=p1; const int (\*y)[3]=p2;

return (\*x)[2]-(\*y)[2];

}

void makeSet(int parent[],int rank[],int n)

{

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

{

parent[i]=i;

rank[i]=0;

}

}

int findParent(int parent[],int component)

{

if(parent[component]==component)

return component;

return parent[component]=findParent(parent,parent[component]);

}

void unionSet(int u,int v,int parent[],int rank[],int n)

{

u=findParent(parent,u); v=findParent(parent,v);

if(rank[u]<rank[v])

{

parent[u]=v;

}

else if(rank[u]<rank[v])

{

parent[v]=u;

}

else

{

parent[v]=u;

rank[u]++;

}

}

void kruskalAlgo(int n,int edge[n][3])

{

qsort(edge,n,sizeof(edge[0]),comparator);

int parent[n]; int rank[n]; makeSet(parent,rank,n);

int minCost=0;

printf("Following are the edges in the constructed MST\n"); for(int i=0;i<n;i++)

{

int v1=findParent(parent,edge[i][0]); int v2=findParent(parent,edge[i][1]);

int wt=edge[i][2];

if(v1!=v2)

{

unionSet(v1,v2,parent,rank,n);

minCost+=wt;

printf("%d -- %d == %d\n",edge[i][0],edge[i][1],wt);

}

}

printf("Minimum Cost Spanning Tree: %d\n",minCost);

}

int main()

{

clrscr();

int edge[5][3]={{0,1,10},

{0,2,6},

{0,3,5},

{1,3,15}, {2,3,4}}; kruskalAlgo(5,edge);

getch();

}

Output:

Following are the edges in the constructed MST

2 -- 3 == 4

0 -- 3 == 5

0 -- 1 == 10

Minimum Cost Spanning Tree: 19

Q4) Implement shortest path algorithm

Program:

#include <limits.h>

#include <stdbool.h>

#include <stdio.h> #include<conio.h> #define V 9

int minDistance(int dist[], bool sptSet[])

{

int min = INT\_MAX, min\_index; for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v] <= min) min = dist[v], min\_index = v;

return min\_index;

}

void printSolution(int dist[])

{

printf("Vertex \t\t Distance from Source\n"); for (int i = 0; i < V; i++)

printf("%d \t\t\t\t %d\n", i, dist[i]);

}

void dijkstra(int graph[V][V], int src)

{

int dist[V]; bool sptSet[V]; for (int i = 0; i < V; i++) dist[i] = INT\_MAX, sptSet[i] = false; dist[src] = 0;

for (int count = 0; count < V - 1; count++)

{

int u = minDistance(dist, sptSet);

sptSet[u] = true; for (int v = 0; v < V; v++)

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX && dist[u] + graph[u][v] < dist[v]) dist[v] = dist[u] + graph[u][v];

}

printSolution(dist);

}

int main()

{

clrscr();

int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },

{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },

{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },

{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },

{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },

{ 8, 11, 0, 0, 0, 0, 1, 0, 7 }, { 0, 0, 2, 0, 0, 0, 6, 7, 0 } }; dijkstra(graph, 0);

getch();

# return 0;

}

Assignment 6

Q.1. Write a program to implement Merge sort method.

Program:

#include <stdio.h>

void merge(int a[], int beg, int mid, int end)

{

int i, j, k;

int n1 = mid - beg + 1;

int n2 = end - mid;

int LeftArray[n1], RightArray[n2];

for (int i = 0; i < n1; i++)

LeftArray[i] = a[beg + i];

for (int j = 0; j < n2; j++)

RightArray[j] = a[mid + 1 + j];

i = 0;

j = 0;

k = beg;

while (i < n1 && j < n2)

{

if(LeftArray[i] <= RightArray[j])

{

a[k] = LeftArray[i];

i++;

}

else

{

a[k] = RightArray[j];

j++;

}

k++;

}

while (i<n1)

{

a[k] = LeftArray[i];

i++;

k++;

}

while (j<n2)

{

a[k] = RightArray[j];

j++;

k++;

}

}

void mergeSort(int a[], int beg, int end)

{

if (beg < end)

{

int mid = (beg + end) / 2;

mergeSort(a, beg, mid);

mergeSort(a, mid + 1, end);

merge(a, beg, mid, end);

}

}

void printArray(int a[], int n)

{

int i;

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

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

printf("\n");

}

int main()

{

int a[] = { 12, 31, 25, 8, 32, 17, 40, 42 };

int n = sizeof(a) / sizeof(a[0]);

printf("Before sorting array elements are - \n");

printArray(a, n);

mergeSort(a, 0, n - 1);

printf("After sorting array elements are - \n");

printArray(a, n);

return 0;

}

Output:

Before sorting array elements are -

12 31 25 8 32 17 40 42

After sorting array elements are -

8 12 17 25 31 32 40 42

Q.2. Write a program to implement Bubble sort method.

Program:

#include<stdio.h>

void print(int a[], int n) //function to print array elements

{

int i;

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

{

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

}

}

void bubble(int a[], int n) // function to implement bubble sort

{

int i, j, temp;

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

{

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

{

if(a[j] < a[i])

{

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

}

}

void main ()

{

int i, j,temp;

int a[5] = { 10, 35, 32, 13, 26};

int n = sizeof(a)/sizeof(a[0]);

printf("Before sorting array elements are - \n");

print(a, n);

bubble(a, n);

printf("\nAfter sorting array elements are - \n");

print(a, n);

}

Output:

Before sorting array elements are -

10 35 32 13 26

After sorting array elements are -

10 13 26 32 35

Q.3. Write a program to implement Fibonacci Search.

Program:

#include <stdio.h>

int min(int x, int y) { return (x <= y) ? x : y; }

int fibMonaccianSearch(int arr[], int x, int n)

{

int fibMMm2 = 0;

int fibMMm1 = 1;

int fibM = fibMMm2 + fibMMm1;

while (fibM < n) {

fibMMm2 = fibMMm1;

fibMMm1 = fibM;

fibM = fibMMm2 + fibMMm1;

}

int offset = -1;

while (fibM > 1) {

int i = min(offset + fibMMm2, n - 1);

if (arr[i] < x) {

fibM = fibMMm1;

fibMMm1 = fibMMm2;

fibMMm2 = fibM - fibMMm1;

offset = i;

}

else if (arr[i] > x) {

fibM = fibMMm2;

fibMMm1 = fibMMm1 - fibMMm2;

fibMMm2 = fibM - fibMMm1;

}

else

return i;

}

if (fibMMm1 && arr[offset + 1] == x)

return offset + 1;

return -1;

}

int main(void)

{

int arr[]

= { 10, 22, 35, 40, 45, 50, 80, 82, 85, 90, 100,235};

int n = sizeof(arr) / sizeof(arr[0]);

int x = 235;

int ind = fibMonaccianSearch(arr, x, n);

if(ind>=0)

printf("Found at index: %d",ind);

else

printf("%d isn't present in the array",x);

return 0;

}

Output:

Found at index: 11

Q.4. Write a program to implement Merge sort method.

Program:

#include <stdio.h>

#include <stdlib.h>

void merge(int arr[], int l,

int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[],

int l, int r)

{

if (l < r)

{

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void printArray(int A[], int size)

{

int i;

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

printf("%d ", A[i]);

printf("\n");

}

int main()

{

int arr[] = {12, 11, 13, 5, 6, 7};

int arr\_size = sizeof(arr) / sizeof(arr[0]);

printf("Given array is \n");

printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

printf("\nSorted array is \n");

printArray(arr, arr\_size);

return 0;

}

Output:

Given array is

12 11 13 5 6 7

Sorted array is

5 6 7 11 12 13

Q.5. Write a program to implement Heap sort method

Program:

#include <stdio.h>

void swap(int\* a, int\* b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void heapify(int arr[], int N, int i)

{

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < N && arr[left] > arr[largest])

largest = left;

if (right < N && arr[right] > arr[largest])

largest = right;

if (largest != i) {

swap(&arr[i], &arr[largest]);

heapify(arr, N, largest);

}

}

void heapSort(int arr[], int N)

{

for (int i = N / 2 - 1; i >= 0; i--)

heapify(arr, N, i);

for (int i = N - 1; i >= 0; i--) {

swap(&arr[0], &arr[i]);

heapify(arr, i, 0);

}

}

void printArray(int arr[], int N)

{

for (int i = 0; i < N; i++)

printf("%d ", arr[i]);

printf("\n");

}

int main()

{

int arr[] = { 12, 11, 13, 5, 6, 7 };

int N = sizeof(arr) / sizeof(arr[0]);

heapSort(arr, N);

printf("Sorted array is\n");

printArray(arr, N);

}

Output:

Sorted array is

5 6 7 11 12 13