A picture containing background pattern

Description automatically generated

**SCHOOL OF ENGINEERING AND INFORMATION TECHNOLOGY**

**DATA STRUCTURES LAB MANUAL**

**For**

**III Semester B. Tech CSE**

**Subject Name: DATA STRUCTURES LAB**

**Subject Code: CSE 2161 Regulation: 2018**

**Submitted by: Fatima Khan**

**Reg. no.: 210101060**

**DEPARTMENT OF**

**COMPUTER SCIENCE AND ENGINEERING**

A picture containing background pattern

Description automatically generated

**SCHOOL OF ENGINEERING AND INFORMATION TECHNOLOGY**

**Certificate**

# Certified to be the bonafide record of the work done by Mr./Ms. ……………………**Fatima Khan**……………………

*Registration No……………****210101060****………………………. Branch ……………………****B. Tech CSE****……………………… in…………………………………………………………………... laboratory, Bachelor of Technology program, during the year 2022-2023.*

Instructor Name………………Professor Sushma Hasan

…………………................

Submitted for End semester Practical Examination held

on………

Lab Programs:

**Topics covered:**

1. Arrays
2. Stacks
3. Queues
4. Linked Lists
5. Binary tree
6. Binary search tree
7. Sorting algorithms

**Programs:**

1. Find the sum of all the elements of an 1D double array of size n using a function Add. The values in the array are read from keyboard.

#include <iostream>

using namespace std;

int n, i, sum=0;

int Add(double\* arr)

{

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

{

sum=sum+arr[i];

}

return sum;

}

int main()

{

cout<<"Enter size of array \n";

cin>>n;

if(n==0)

{

cout<<"Please enter value greater than 0";

}

else

{

double\* arr=(double\*)malloc(sizeof(double)\*n);

cout<<"Enter elements\n";

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

{

cin>>arr[i];

}

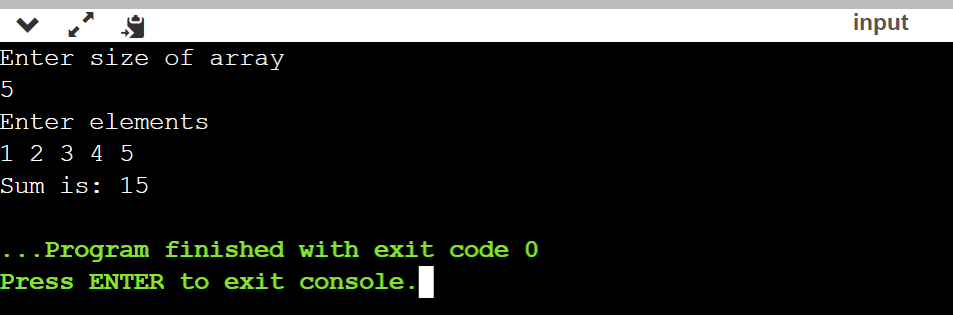
Add(arr);

cout<<"Sum is: "<<sum;

}

return 0;

}



1. Implement an iterative Lsearch(....) function to search for an element in an 1D array of type integer using linear search technique.

#include <iostream>

using namespace std;

int i, n=0, element, j=0;

int Lsearch(int\* arr)

{

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

{

if(arr[i]==element)

cout<<"Element found at: "<<(i+1)<<" position\n";

else

j++;

}

if(j==i)

cout<<"Element not found";

return 0;

}

int main()

{

do

{

cout<<"Enter size of array \n";

cin>>n;

if(n==0)

cout<<"Please enter value greater than 0";

}while(n==0);

int\* arr=(int\*)malloc(sizeof(int)\*n);

cout<<"Enter elements\n";

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

{

cin>>arr[i];

}

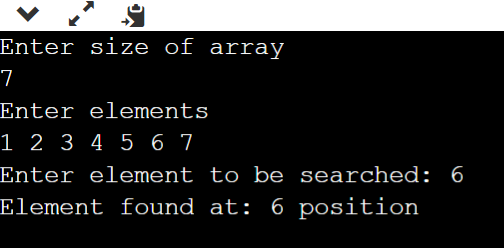
cout<<"Enter element to be searched: ";

cin>>element;

Lsearch(arr);

return 0;

}

1. Implement a C program to read, display and to find the product of two matrices using functions with suitable parameters. Check for the compatibility of the input matrices before multiplication.

#include <iostream>

using namespace std;

void enter(int m1[][10], int m2[][10], int r1, int c1, int r2, int c2);

void multiply(int m1[][10], int m2[][10], int mulres[][10], int r1, int c1, int r2, int c2);

void display(int m1[][10],int m2[][10],int mult[][10], int r1,int r2,int c1, int c2);

int main()

{

int m1[10][10], m2[10][10], mult[10][10], r1, c1, r2, c2, i, j, k;

cout<<"Enter rows and column for first matrix: ";

cin>>r1;

cin>>c1;

cout<<"Enter rows and column for second matrix: ";

cin>>r2;

cin>>c2;

// If colum of first matrix in not equal to row of second matrix, asking user to enter the size of matrix again.

while (c1 != r2)

{

cout<<"Error! column of first matrix not equal to row of second.\n";

cout<<"Enter rows and column for first matrix: ";

cin>>r1;

cin>>c1;

cout<<"Enter rows and column for second matrix: ";

cin>>r2;

cin>>c2;

}

// Function to take matrices data

enter(m1, m2, r1, c1, r2, c2);

// Function to multiply two matrices.

multiply(m1, m2, mult, r1, c1, r2, c2);

// Function to display resultant matrix after multiplication.

display(m1,m2,mult, r1,r2,c1, c2);

return 0;

}

void enter(int m1[][10], int m2[][10], int r1, int c1, int r2, int c2)

{

int i, j;

cout<<"\nEnter elements of matrix 1:\n";

for(i = 0; i < r1; ++i)

{

for(j = 0; j < c1; ++j)

{

cout<<"Enter elements a : ", i + 1,"\t", j + 1;

cin>>m1[i][j];

}

}

cout<<"\nEnter elements of matrix 2:\n";

for(i = 0; i < r2; ++i)

{

for(j = 0; j < c2; ++j)

{

cout<<"Enter elements b : ", i + 1,"\t", j + 1;

cin>>m2[i][j];

}

}

}

void multiply(int m1[][10], int m2[][10], int mult[][10], int r1, int c1, int r2, int c2)

{

int i, j, k;

// Initializing elements of matrix mult to 0.

for(i = 0; i < r1; ++i)

{

for(j = 0; j < c2; ++j)

{

mult[i][j] = 0;

}

}

// Multiplying matrix m1 and m2 and storing in array mult.

for(i = 0; i < r1; ++i)

{

for(j = 0; j < c2; ++j)

{

for(k=0; k<c1; ++k)

{

mult[i][j] += m1[i][k] \* m2[k][j];

}

}

}

}

void display(int m1[][10],int m2[][10],int mult[][10], int r1,int r2, int c1,int c2)

{

int i, j;

cout<<"\nMatrix 1 :\n";

for(i = 0; i < r1; ++i)

{

for(j = 0; j < c1; ++j)

{

cout<<"\t"<<m1[i][j]<<" ";

if(j == c1 - 1)

cout<<"\n\n";

}

}

cout<<"\nMatrix 2 :\n";

for(i = 0; i < r2; ++i)

{

for(j = 0; j < c2; ++j)

{

cout<<"\t"<<m2[i][j]<<" ";

if(j == c2 - 1)

cout<<"\n\n";

}

}

cout<<"\nOutput Matrix:\n";

for(i = 0; i < r1; ++i)

{

for(j = 0; j < c2; ++j)

{

cout<<"\t"<<mult[i][j]<<" ";

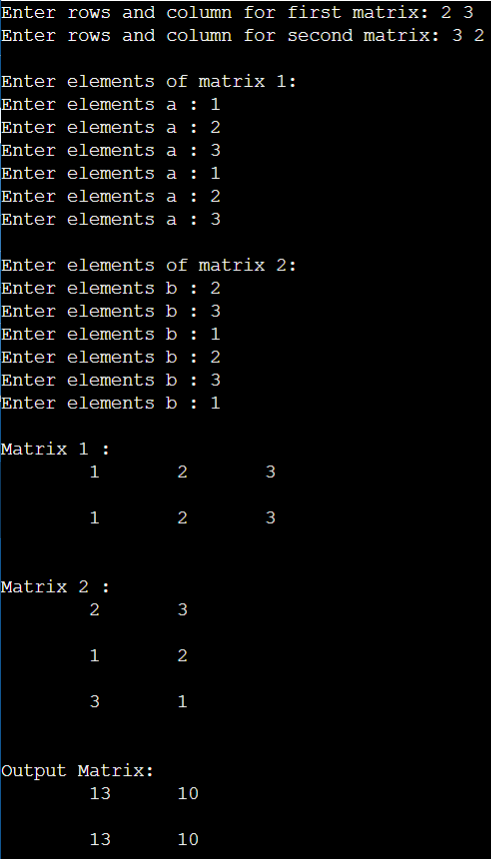
if(j == c2 - 1)

cout<<"\n\n";

}

}

}



1. Find the 2nd largest in a list of numbers using a function (do not sort the list).

#include <iostream>

using namespace std;

int main(){

int n, num[50], lrgst, secnd;

cout<<"Enter number of elements: ";

cin>>n;

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

cout<<"Enter Array Element"<<(i+1)<<": ";

cin>>num[i];

}

if(num[0]<num[1]){

lrgst = num[1];

secnd = num[0];

}

else{

lrgst = num[0];

secnd = num[1];

}

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

if (num[i] > lrgst) {

secnd = lrgst;

lrgst = num[i];

}

else if (num[i] > secnd && num[i] != lrgst) {

secnd = num[i];

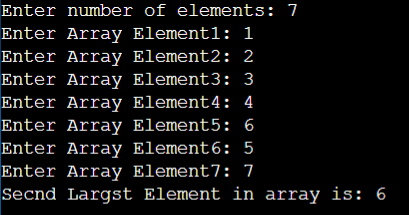
}

}

cout<<"Secnd Largst Element in array is: "<<secnd;

return 0;

}



1. Addition of polynomials with two terms: To perform different operations on polynomial with two terms x, y using 2-D array representations. Operations like addition and multiplication have to be implemented [ref: J.P Trembly]. If the 2D array representation is sparse then optimize the memory usage by using suitable alternative representation.

#include <iostream>

using namespace std;

int h, i, j, p, q;

int main()

{

cout<<"Enter highest degree: ";

cin>>h;h++;

int a[h][h], b[h][h];

cout<<"Enter for first poly : \n";

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

{

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

{

cout<<"Coefficient for x^"<<i<<"y^"<<j<<": ";

cin>>a[i][j];

}

}

cout<<"Enter for second poly : \n";

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

{

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

{

cout<<"Coefficient for x^"<<i<<"y^"<<j<<": ";

cin>>b[i][j];

}

}

cout<<"First Polynomial \n";

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

{

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

{

if(a[i][j]==0){}

else{

cout<<" + ";

cout<<a[i][j]<<"x^"<<i<<"y^"<<j;

//if((i!=(h-1))&&(j!=(h-1)))

}

}

}

cout<<"\n\n Second polynomial \n";

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

{

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

{

if(b[i][j] == 0){}

else{

cout<<" + ";

cout<<b[i][j]<<"x"<<i<<"y"<<j;

//if((i!=h)&&(j!=h))

}}

}

//add

cout<<"\n\nAddition \n";

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

{

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

{

int sum = a[i][j]+b[i][j];

if (sum == 0){}

else{

cout<<" + ";

cout<<sum<<"x^"<<i<<"y^"<<j;

//if((i!=h)&&(j!=h))

}

}

}

//subtract

cout<<"\n\nSubtraction \n";

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

{

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

{

int diff = (a[i][j]-b[i][j]);

if (diff == 0){}

else{

if(diff >=0)

cout<<" + ";

else

cout<<" - ";

cout<<abs(diff)<<"x^"<<i<<"y^"<<j;

//if(i\*j!=(h-1)\*(h-1))

}}

}

//multiplication]

cout<<"\n\n Multiplication \n";

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

{

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

{

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

{

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

{

int mult = a[i][j]\*b[p][q];

if(mult == 0){}

else{

cout<<" + ";

cout<<mult<<"x^"<<(i+p)<<"y^"<<(j+q);

}

}

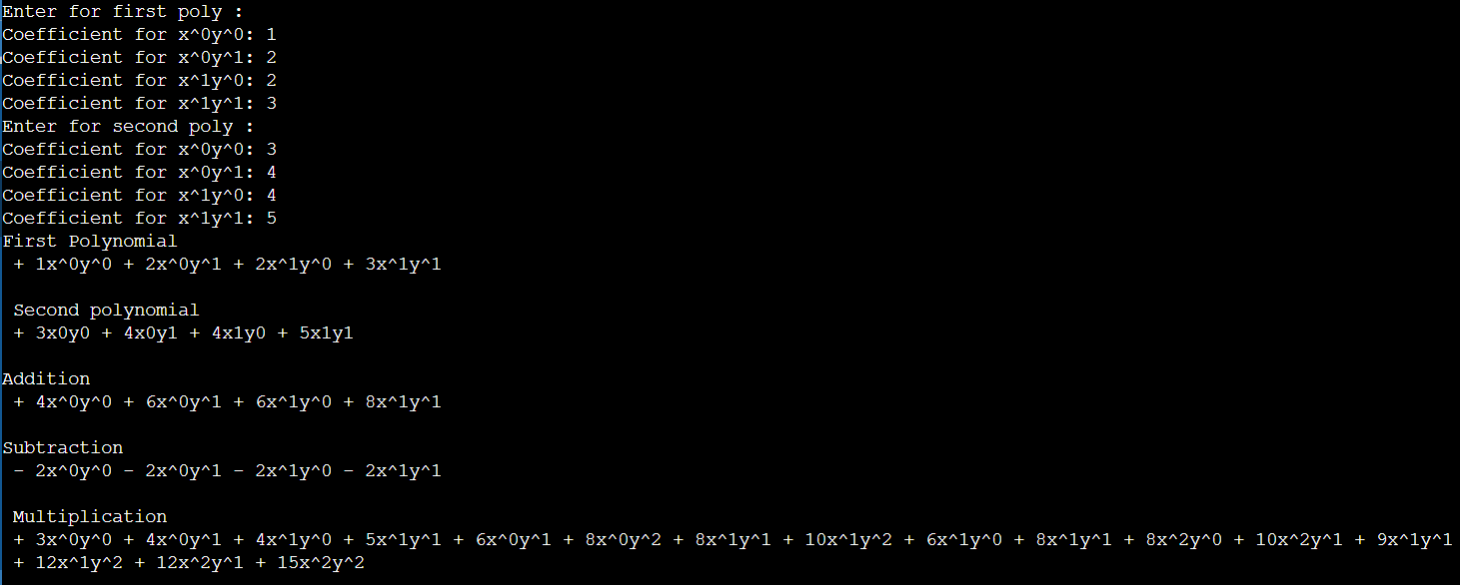
}

}

}

return 0;

}



1. Write a function Reverse to reverse the elements of an integer array using pointer. Access the elements of the array using dereference operator. Read the values from the keyboard in main function. Display the result in the main function.

#include <iostream>

using namespace std;

int rev(int \*ptr, int l)

{

int temp;

for(int i=0;i<l/2;i++)

{

temp=\*(ptr+l-i-1);

\*(ptr+l-i-1)=\*(ptr+i);

\*(ptr+i)=temp;

}

return 0;

}

int main()

{

int l, i;

cout<<"Enter array length: ";

cin>>l;

int a[l];

cout<<"Enter array: ";

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

{

cin>>a[i];

}

rev(a, l);

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

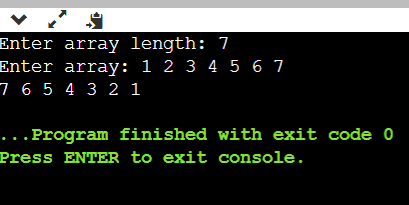
{

cout<<a[i]<<" ";

}

return 0;

}



1. Write a function Smallest to find the smallest element in an array using pointer. Create a dynamically allocated array and read the values from keyboard in main. Display the result in the main function.

#include <iostream>

using namespace std;

int smallest(int \*ptr, int l)

{

int temp=\*ptr;

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

{

if(\*(ptr+i)<temp)

temp=\*(ptr+i);

}

return temp;

}

int main()

{

int l, i;

cout<<"Enter array length: ";

cin>>l;

int a[l];

cout<<"Enter array: ";

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

{

cin>>a[i];

}

cout<<"Display array: ";

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

{

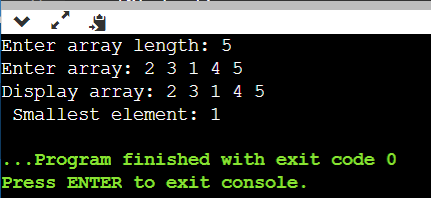
cout<<a[i]<<" ";

}

cout<<"\n Smallest element: "<<smallest(a, l);

return 0;

}



1. Write a C++ program to add, subtract, multiply two matrices.

#include <iostream>

using namespace std;

void enter(int m1[][10], int m2[][10], int r1, int c1, int r2, int c2);

void multiply(int m1[][10], int m2[][10], int mulres[][10], int r1, int c1, int r2, int c2);

void display(int m1[][10],int m2[][10],int mult[][10], int r1,int r2,int c1, int c2);

int main()

{

int m1[10][10], m2[10][10], mult[10][10], r1, c1, r2, c2, i, j, k;

c1=r1=r2=0;c2=0;

do{

if((((c1==r1)&&(c2==r1))&&(r2==r1))==0)

cout<<"Enter again. All dimensions must be equal to be compatible with bot Addition and Multiplication\n";

cout<<"Enter rows and column for first matrix: ";

cin>>r1;

cin>>c1;

cout<<"Enter rows and column for second matrix: ";

cin>>r2;

cin>>c2;

}while((((c1==r1)&&(c2==r1))&&(r2==r1))==0);

cout<<"\nEnter elements of matrix 1:\n";

for(i = 0; i < r1; ++i)

{

for(j = 0; j < c1; ++j)

{

cout<<"Enter elements a : ", i + 1,"\t", j + 1;

cin>>m1[i][j];

}

}

cout<<"\nEnter elements of matrix 2:\n";

for(i = 0; i < r2; ++i)

{

for(j = 0; j < c2; ++j)

{

cout<<"Enter elements b : ", i + 1,"\t", j + 1;

cin>>m2[i][j];

}

}

// Initializing elements of matrix mult to 0.

for(i = 0; i < r1; ++i)

{

for(j = 0; j < c2; ++j)

{

mult[i][j]=0;

}

}

// Multiplying matrix m1 and m2 and storing in array mult.

for(i = 0; i < r1; ++i)

{

for(j = 0; j < c2; ++j)

{

for(k=0; k<c1; ++k)

{

mult[i][j] += m1[i][k] \* m2[k][j];

}

}

}

cout<<"\nMatrix 1 :\n";

for(i = 0; i < r1; ++i)

{

for(j = 0; j < c1; ++j)

{

cout<<"\t"<<m1[i][j]<<" ";

if(j == c1 - 1)

cout<<"\n\n";

}

}

cout<<"\nMatrix 2 :\n";

for(i = 0; i < r2; ++i)

{

for(j = 0; j < c2; ++j)

{

cout<<"\t"<<m2[i][j]<<" ";

if(j == c2 - 1)

cout<<"\n\n";

}

}

cout<<"Added Matrix \n";

for(i = 0; i < r2; ++i)

{

for(j = 0; j < c2; ++j)

{

cout<<"\t"<<(m1[i][j]+m2[i][j])<<" ";

}

cout<<"\n";

}

cout<<"Added Matrix \n";

for(i = 0; i < r2; ++i)

{

for(j = 0; j < c2; ++j)

{

if((m1[i][j]-m2[i][j])>=0)

cout<<" + ";

else

cout<<" - ";

cout<<abs(m1[i][j]-m2[i][j])<<"\t";

}

cout<<"\n";

}

cout<<"\nMultuplied Matrix:\n";

for(i = 0; i < r1; ++i)

{

for(j = 0; j < c2; ++j)

{

cout<<"\t"<<mult[i][j]<<" ";

if(j == c2 - 1)

cout<<"\n\n";

}

}

return 0;

}

Text

Description automatically generated

1. Implement stack using:

Array with Global variable

#include<iostream>

using namespace std;

int stack[100],choice,n,top,x,i;

void push(void);

void pop(void);

void display(void);

int main()

{

//clrscr();

top=-1;

cout<<"\n"<< "Enter the size of STACK[MAX=100]";

cin>>n;

cout<<"\n\t"<<" STACK OPERATIONS USING ARRAY";

cout<<"\n\t--------------------------------";

cout<<"\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT";

do

{

cout<<"\n Enter the Choice:";

cin>>choice;

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

cout<<"\n\t EXIT POINT ";

break;

}

default:

{

cout<<"\n\t Please Enter a Valid Choice(1/2/3/4)";

}

}

}

while(choice!=4);

return 0;

}

void push()

{

if(top>=n-1)

{

cout<<"\n\tSTACK is over flow";

}

else

{

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

cin>>x;

top++;

stack[top]=x;

}

}

void pop()

{

if(top<=-1)

{

cout<<"\n\t Stack is under flow";

}

else

{

cout<<"\n\t The popped elements is "<<stack[top];

top--;

}

}

void display()

{

if(top>=0)

{

cout<<"\n The elements in STACK \n";

for(i=top; i>=0; i--)

cout<<"\n "<<stack[i];

cout<<"\n Press Next Choice";

}

else

{

cout<<"\n The STACK is empty";

}}

Text

Description automatically generated

Local variable

#include <iostream>

#include <conio.h>

using namespace std;

#define MAX 10

void display(int stack[], int TOP)

{

int i=0;

if(TOP==-1)

{

cout<<"Stack is Empty .\n";

return;

}

cout<<stack[TOP];

for(i=TOP-1;i >=0;i--)

{

cout<<"\n"<<stack[i];

}

cout<<"\n\n";

}

/\* function : PUSH(),

to push an item into stack.

\*/

void PUSH(int stack[],int item, int \*TOP)

{

if(\*TOP==MAX-1)

{

cout<<"\nSTACK is FULL CAN't ADD ITEM\n";

return;

}

(\*TOP)=(\*TOP)+1;

stack[\*TOP]= item;

}

void POP(int stack[], int \*TOP)

{

int deletedItem;

if((\*TOP)==-1)

{

cout<<"STACK is EMPTY.\n";

return;

}

deletedItem=stack[\*TOP];

(\*TOP)--;

cout<<"deleted successfully\n"<<deletedItem;

return;

}

int main()

{

int STACK[MAX],TOP;

int ITEM=0;

int choice=0;

TOP=-1;

while(1)

{

cout<<"\n Enter Choice (1: display, 2: insert (PUSH), 3:remove(POP)), 4: Exit..:";

cin>>choice;

switch(choice)

{

case 1:

display(STACK, TOP);

break;

case 2:

cout<<"Enter Item to be insert :";

cin>>ITEM;

PUSH(STACK,ITEM, &TOP);

break;

case 3:

POP(STACK, &TOP);

break;

case 4:

exit(0);

default:

cout<<"\nInvalid choice.";

break;

}

getch();

}// end of while(1)

}

Text

Description automatically generated

Using linked list

#include <iostream>

using namespace std;

// Structure to create a node with data and the next pointer

struct Node {

int data;

struct Node \*next;

};

Node\* top = NULL;

// Push() operation on a stack

void push(int value) {

struct Node \*newNode= (struct Node \*)malloc(sizeof(struct Node));

newNode->data = value; // assign value to the node

if (top == NULL) {

newNode->next = NULL;

} else

newNode->next = top; // Make the node as top

top = newNode; // top always points to the newly created node

cout<<value<<" is Inserted\n\n";

}

void pop() {

if (top == NULL) {

cout<<"\nStack Underflow\n";

} else {

struct Node \*temp = top;

int temp\_data = top->data;

top = top->next;

free(temp);

cout<<"Popped element is : \n"<<temp\_data<<endl;

}

}

void display() {

// Display the elements of the stack

if (top == NULL) {

cout<<"\nStack Underflow\n";

} else {

cout<<"The stack is \n";

struct Node \*temp = top;

while (temp != NULL) {

cout<<" "<< temp->data;

temp = temp->next;

}

}cout<<endl;

}

int main() {

int choice, value;

cout<<"\nImplementation of Stack using Linked List\n";

while (1) {

cout<<"1. Push\n2. Pop\n3. Display\n4. Exit\n";

cout<<"\nEnter your choice : ";

cin>>choice;

switch (choice) {

case 1:

cout<<"\nEnter the value to insert: ";

cin>>value;

push(value);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

exit(0);

break;

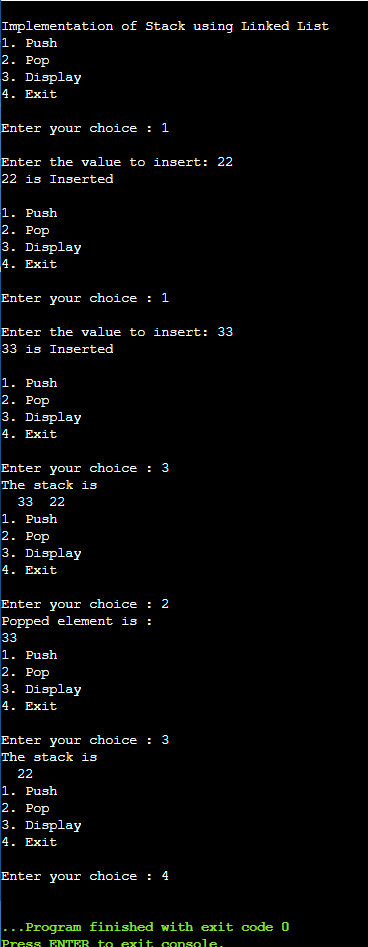
default:

cout<<"\nWrong Choice\n";

}

}

}



1. Dynamic array implementation of Stacks.

#include<iostream>

using namespace std;

int n, choice,top,x,i;

int\* stack;

void push()

{

if(top>=n-1)

{

cout<<"Stack overflow. Size is doubled."<<"\n";

n=n\*2;

int\* newStack = new int[n];

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

newStack[i] = stack[i];

}

delete[] stack;

stack = newStack;

}

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

cin>>x;

top++;

stack[top]=x;

}

void pop()

{

if(top<=-1)

{

cout<<"\n\t Stack is under flow";

}

else

{

cout<<"\n\t The popped elements is "<<stack[top];

top--;

}

}

void display()

{

if(top>=0)

{

cout<<"\n The elements in STACK \n";

for(i=top; i>=0; i--)

cout<<"\n "<<stack[i];

cout<<"\n Press Next Choice";

}

else

{

cout<<"\n The STACK is empty";

}

}

int main()

{

n=1;

stack=new int[n];

top=-1;

cout<<"\n"<< "Enter the size of STACK";

cin>>n;choice=1;

do

{

cout<<"\n\t"<<" STACK OPERATIONS USING ARRAY";

cout<<"\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT";

cout<<"\n Enter the Choice:";

cin>>choice;

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

cout<<"\n\t EXIT POINT ";

break;

}

default:

{

cout<<"\n\t Please Enter a Valid Choice(1/2/3/4)";

}

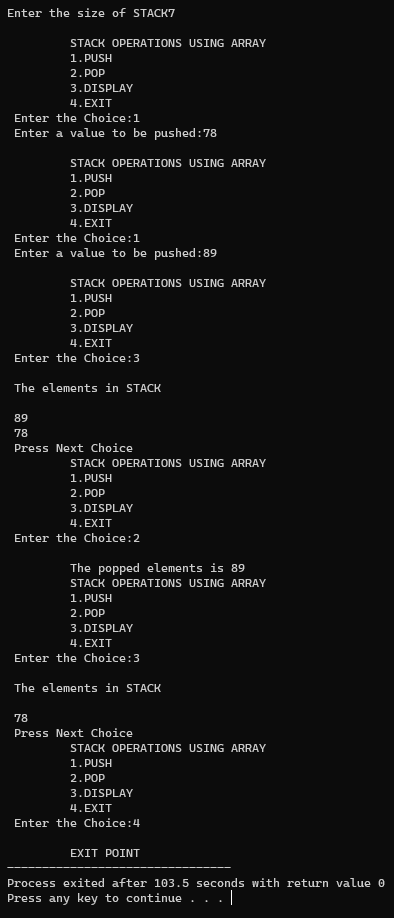
}

}

while(choice!=4);

return 0;

}



1. Implement Queue using:

Array with Global variable,

#include <iostream>

using namespace std;

int queue[3], n = 3, front = - 1, rear = - 1;

void Insert() {

int val;

if (rear == n - 1)

cout<<"Queue Overflow"<<endl;

else {

if (front == - 1)

{front = 0;}

cout<<"Insert the element in queue : "<<endl;

cin>>val;

rear++;

queue[rear] = val;

}

}

void Delete() {

if (front == - 1 || front > rear) {

cout<<"Queue Underflow ";

return ;

} else {

cout<<"Element deleted from queue is : "<< queue[front] <<endl;

front++;;

}

}

void Display() {

if (front == - 1)

cout<<"Queue is empty"<<endl;

else {

cout<<"Queue elements are : ";

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

cout<<queue[i]<<" ";

cout<<endl;

}

}

int main() {

int ch;

cout<<"1) Insert element to queue"<<endl;

cout<<"2) Delete element from queue"<<endl;

cout<<"3) Display all the elements of queue"<<endl;

cout<<"4) Exit"<<endl;

do {

cout<<"Enter your choice : "<<endl;

cin>>ch;

switch (ch) {

case 1: Insert();

break;

case 2: Delete();

break;

case 3: Display();

break;

case 4: cout<<"Exit"<<endl;

break;

default: cout<<"Invalid choice"<<endl;

}

} while(ch!=4);

return 0;

}

Text

Description automatically generated

Local variable

#include <iostream>

#define MAX 10

using namespace std;

void display(int [], int ,int);

/\* push (insert) item into stack\*/

void insrt(int [],int, int \*,int \*);

/\* pop (remove) item from stack\*/

void delte(int [], int\*, int \*);

int main()

{

int QUEUE[MAX],REAR,FRNT ;

/\* display stack element\*/

int ITEM=0;

int choice=0;

REAR=-1;

FRNT = -1;

while(choice != 4)

{

cout<< "QUEUE IMPLEMENTATION \n1: display \n2: insert (PUSH) \n3: remove(POP) \n4: Exit \nEnter choice : ";

cin>> choice;

switch(choice)

{

case 1:

display(QUEUE, FRNT, REAR);

break;

case 2:

cout<< "Enter item to be insert :";

cin>> ITEM;

insrt(QUEUE,ITEM, &FRNT, &REAR);

break;

case 3:

delte(QUEUE, &FRNT, &REAR);

break;

case 4:

exit(0);

default:

cout<< "\nInvalid choice.";

break;

}

} // end of while(1)

}

/\* function : display(),

to display stack elements.

\*/

void display(int queue[], int frnt, int rear)

{

int i=0;

if(rear == -1)

{

cout<< "Queue is Empty .\n";

return;

}

for(i=rear;i >=frnt;i--)

{

cout<< "\n" << queue[i];

}

cout<< "\n\n";

}

/\* function : PUSH(),

to push an item into stack.

\*/

void insrt(int queue[],int item, int \*FRNT, int \*REAR)

{

if(\*REAR>=MAX-1)

{

cout<< "\nQUEUE is FULL CAN'T ADD ITEM\n";

return;

}

(\*REAR)++;

if (\*FRNT == -1) {(\*FRNT)=0;}

queue[\*REAR]=item;

}

/\* function : POP(),

to pop an item from stack.

\*/

void delte(int queue[], int \*FRNT, int \*REAR)

{

int deletedItem;

if(\*REAR < \*FRNT)

{

cout<< "QUEUE is EMPTY.\n";

return;

}

deletedItem=queue[\*FRNT];

\*FRNT=\*FRNT+1;

cout<< "deleted successfully\n" << deletedItem;

return;

}

Text

Description automatically generated

Using linked list

#include <iostream>

#include <cstdlib>

using namespace std;

struct node {

int data;

struct node \*next;

};

struct node\* front = NULL;

struct node\* rear = NULL;

struct node\* temp;

void Insert(int val) {

if (rear == NULL) {

rear = new node;

rear->next = NULL;

rear->data = val;

front = rear;

} else {

temp=new node;

rear->next = temp;

temp->data = val;

temp->next = NULL;

rear = temp;

}

}

void Delete() {

temp = front;

if (front == NULL) {

cout<<"Queue is empty!!"<<endl; } else if (temp->next != NULL) {

temp = temp->next;

cout<<"Element deleted from queue is : "<<front->data<<endl;

free(front);

front = temp;

} else {

cout<<"Element deleted from queue is : "<<front->data<<endl;

free(front);

front = NULL;

rear = NULL;

}

}

void Display() {

temp = front;

if ((front == NULL) && (rear == NULL)) {

cout<<"Queue is empty"<<endl;

return;

}

while (temp != NULL) {

cout<<temp->data<<" "; temp = temp->next;

}

cout<<endl;

}

int main() {

int choice, value;

while (1) {

cout<<"1. Push\n2. Pop\n3. Display\n4. Exit\n";

cout<<"\nEnter your choice : ";

cin>>choice;

switch (choice) {

case 1:

cout<<"\nEnter the value to insert: ";

cin>>value;

Insert(value);

break;

case 2:

Delete();

break;

case 3:

Display();

break;

case 4:

exit(0);

break;

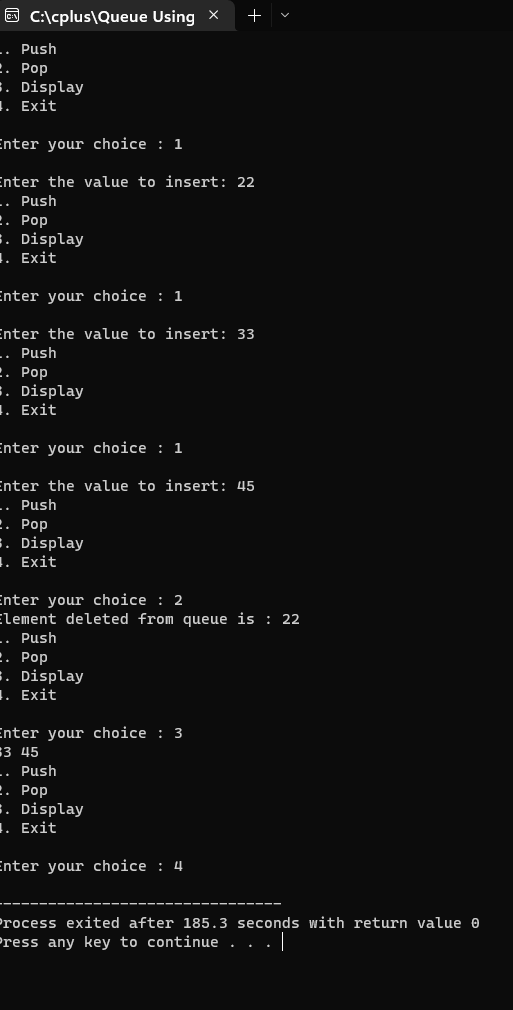
default:

cout<<"\nWrong Choice\n";

}

}

}



1. Implement two stacks using same array

Program: #include <iostream>

const int MAX\_SIZE = 100;

class Stack { private:

int data[MAX\_SIZE]; // Array to store the stack elements int top1, top2; // Indexes of the top elements of the two stacks

public:

// Constructor to initialize the stack

Stack() { top1 = -1; top2 = MAX\_SIZE;

}

// Push element x to the first stack void push1(int x) {

// Check if there is space in the stack if (top1 < top2 - 1) { top1++; data[top1] = x;

} else { std::cout << "Error: stack overflow" << std::endl;

}

}

// Push element x to the second stack void push2(int x) {

// Check if there is space in the stack if (top1 < top2 - 1) { top2--; data[top2] = x;

} else {

std::cout << "Error: stack overflow" << std::endl;

}

}

// Pop element from the first stack int pop1() {

// Check if the stack is empty if (top1 >= 0) { int x = data[top1]; top1--; return x;

} else { std::cout << "Error: stack underflow" << std::endl; return -1;

}

}

// Pop element from the second stack int pop2() {

// Check if the stack is empty if (top2 < MAX\_SIZE) { int x = data[top2]; top2++; return x;

} else { std::cout << "Error: stack underflow" << std::endl; return -1;

}

}

};

int main() {

Stack s;

s.push1(1);

s.push1(2);

s.push1(3);

s.push2(4);

s.push2(5);

s.push2(6);

std::cout << "Popped element from stack 1: " << s.pop1() << std::endl; std::cout << "Popped element from stack 2: " << s.pop2() << std::endl;

return 0;

}

Result :

Graphical user interface, text

Description automatically generated

1. Insertion with all cases considered, deletion and finding in a linked list

// Linked list operations in C++

#include <stdlib.h>

#include <iostream>

using namespace std;

// Create a node

struct Node {

int DATA;

struct Node\* ptr2next;

};

void insertAtBeginning(struct Node\*\* head\_ref, int new\_DATA) {

// Allocate memory to a node

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

// insert the DATA

newNode->DATA = new\_DATA;

newNode->ptr2next = (\*head\_ref);

// Move head to new node

(\*head\_ref) = newNode;

}

// Insert a node after a node

void insertAfter(struct Node\* prev\_node, int new\_DATA) {

if (prev\_node == NULL) {

cout << "the given previous node cannot be NULL";

return;

}

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->DATA = new\_DATA;

newNode->ptr2next = prev\_node->ptr2next;

prev\_node->ptr2next = newNode;

}

// Insert at the end

void insertAtEnd(struct Node\*\* head\_ref, int new\_DATA) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

struct Node\* last = \*head\_ref; / used in step 5\*/

newNode->DATA = new\_DATA;

newNode->ptr2next = NULL;

if (\*head\_ref == NULL) {

\*head\_ref = newNode;

return;

}

while (last->ptr2next != NULL) last = last->ptr2next;

last->ptr2next = newNode;

return;

}

// Delete a node

void deleteNode(struct Node\*\* head\_ref, int key) {

struct Node \*temp = \*head\_ref, \*prev;

if (temp != NULL && temp->DATA == key) {

\*head\_ref = temp->ptr2next;

free(temp);

return;

}

// Find the key to be deleted

while (temp != NULL && temp->DATA != key) {

prev = temp;

temp = temp->ptr2next;

}

// If the key is not present

if (temp == NULL) return;

// Remove the node

prev->ptr2next = temp->ptr2next;

free(temp);

}

// Search a node

bool searchNode(struct Node\*\* head\_ref, int key) {

struct Node\* current = \*head\_ref;

while (current != NULL) {

if (current->DATA == key) return true;

current = current->ptr2next;

}

return false;

}

// Sort the linked list

void sortLinkedList(struct Node\*\* head\_ref) {

struct Node \*current = \*head\_ref, \*index = NULL;

int temp;

if (head\_ref == NULL) {

return;

} else {

while (current != NULL) {

// index points to the node ptr2next to current

index = current->ptr2next;

while (index != NULL) {

if (current->DATA > index->DATA) {

temp = current->DATA;

current->DATA = index->DATA;

index->DATA = temp;

}

index = index->ptr2next;

}

current = current->ptr2next;

}

}

}

// Print the linked list

void printList(struct Node\* node) {

while (node != NULL) {

cout << node->DATA << " ";

node = node->ptr2next;

}

}

// Driver program

int main() {

struct Node\* head = NULL;

insertAtEnd(&head, 7);

insertAtBeginning(&head, 8);

insertAtBeginning(&head, 9);

insertAtEnd(&head, 8);

insertAfter(head->next, 1);

cout << "Linked list: ";

printList(head);

cout << "\nAfter deleting an element: ";

deleteNode(&head, 9);

printList(head);

int item\_to\_find = 8;

if (searchNode(&head, item\_to\_find)) {

cout << endl << item\_to\_find << " is found";

} else {

cout << endl << item\_to\_find << " is not found";

}

sortLinkedList(&head);

cout << "\nSorted List: ";

printList(head);

}

Graphical user interface, text

Description automatically generated

1. Binary tree insertions, deletions, traversals

#include <iostream>

struct Node {

int data;

Node\* left;

Node\* right;

Node(int data) : data(data), left(nullptr), right(nullptr) {}

};

class BinaryTree {

public:

// Constructor

BinaryTree() : root\_(nullptr) {}

// Destructor

~BinaryTree() {

delete\_subtree(root\_);

}

// Insert a new node into the binary tree

void insert(int data) {

root\_ = insert(root\_, data);

}

// Delete a node from the binary tree

void remove(int data) {

root\_ = remove(root\_, data);

}

// Perform an in-order traversal of the binary tree

void in\_order() const {

in\_order(root\_);

std::cout << std::endl;

}

// Perform a pre-order traversal of the binary tree

void pre\_order() const {

pre\_order(root\_);

std::cout << std::endl;

}

// Perform a post-order traversal of the binary tree

void post\_order() const {

post\_order(root\_);

std::cout << std::endl;

}

private:

// Recursively delete the subtree rooted at the given node

void delete\_subtree(Node\* node) {

if (!node) {

return;

}

delete\_subtree(node->left);

delete\_subtree(node->right);

delete node;

}

// Recursively insert a new node into the subtree rooted at the given node

Node\* insert(Node\* node, int data) {

if (!node) {

return new Node(data);

}

if (data < node->data) {

node->left = insert(node->left, data);

} else {

node->right = insert(node->right, data);

}

return node;

}

// Recursively remove a node from the subtree rooted at the given node

Node\* remove(Node\* node, int data) {

if (!node) {

return nullptr;

}

if (data < node->data) {

node->left = remove(node->left, data);

} else if (data > node->data) {

node->right = remove(node->right, data);

} else {

if (!node->left) {

Node\* right = node->right;

delete node;

return right;

}

if (!node->right) {

Node\* left = node->left;

delete node;

return left;

}

Node\* min\_node = find\_min(node->right);

node->data = min\_node->data;

node->right = remove(node->right, min\_node->data);

}

return node;

}

// Find the node with the minimum value in the

Node\* find\_min(Node\* node) {

while (node->left) {

node = node->left;

}

return node;

}

// Recursively perform an in-order traversal of the subtree rooted at the given node

void in\_order(Node\* node) const {

if (!node) {

return;

}

in\_order(node->left);

std::cout << node->data << " ";

in\_order(node->right);

}

// Recursively perform a pre-order traversal of the subtree rooted at the given node

void pre\_order(Node\* node) const {

if (!node) {

return;

}

std::cout << node->data << " ";

pre\_order(node->left);

pre\_order(node->right);

}

// Recursively perform a post-order traversal of the subtree rooted at the given node

void post\_order(Node\* node) const {

if (!node) {

return;

}

post\_order(node->left);

post\_order(node->right);

std::cout << node->data << " ";

}

Node\* root\_;

};

int main() {

BinaryTree tree;

// Insert some elements into the tree

tree.insert(9);

tree.insert(8);

tree.insert(7);

tree.insert(6);

tree.insert(5);

tree.insert(4);

tree.insert(3);

// Delete an element from the tree

tree.remove(4);

// Perform in-order, pre-order, and post-order traversals

std::cout << "In-order traversal: ";

tree.in\_order();

std::cout << "Pre-order traversal: ";

tree.pre\_order();

std:: cout << "Post-order traversal: ";

tree.post\_order();

return 0;

}

Text

Description automatically generated

1. Binary search tree insertion, deletion, search and traversals

// Binary Search Tree operations C++

// Binary Search Tree operations C++

#include <iostream>

using namespace std;

struct node {

int data;

struct node \*leftchild, \*rightchild;

};

// Create a node

struct node \*new\_Node(int item) {

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

temp->data = item;

temp->leftchild = temp->rightchild = NULL;

return temp;

}

// Inorder Traversal

void inorder(struct node \*root) {

if (root != NULL) {

// Traverse leftchild

inorder(root->leftchild);

// Traverse root

cout << root->data << " -> ";

// Traverse rightchild

inorder(root->rightchild);

}

}

// Insert a node

struct node \*insert(struct node \*node, int data) {

// Return a new node if the tree is empty

if (node == NULL) return new\_Node(data);

// Traverse to the rightchild place and insert the node

if (data < node->data)

node->leftchild = insert(node->leftchild, data);

else

node->rightchild = insert(node->rightchild, data);

return node;

}

// Find the inorder successor

struct node \*minValueNode(struct node \*node) {

struct node \*current = node;

// Find the leftchildmost leaf

while (current && current->leftchild != NULL)

current = current->leftchild;

return current;

}

// Deleting a node

struct node \*deleteNode(struct node \*root, int data) {

// Return if the tree is empty

if (root == NULL) return root;

// Find the node to be deleted

if (data < root->data)

root->leftchild = deleteNode(root->leftchild, data);

else if (data > root->data)

root->rightchild = deleteNode(root->rightchild, data);

else {

// If the node is with only one child or no child

if (root->leftchild == NULL) {

struct node \*temp = root->rightchild;

free(root);

return temp;

} else if (root->rightchild == NULL) {

struct node \*temp = root->leftchild;

free(root);

return temp;

}

// If the node has two children

struct node \*temp = minValueNode(root->rightchild);

// Place the inorder successor in position of the node to be deleted

root->data = temp->data;

// Delete the inorder successor

root->rightchild = deleteNode(root->rightchild, temp->data);

}

return root;

}

int main() {

struct node \*root = NULL;

root = insert(root, 9);

root = insert(root, 4);

root = insert(root, 2);

root = insert(root, 7);

root = insert(root, 8);

root = insert(root, 11);

root = insert(root, 13);

root = insert(root, 5);

cout << "Inorder traversal: ";

inorder(root);

cout << "\nAfter deleting 10\n";

root = deleteNode(root, 11);

cout << "Inorder traversal: ";

inorder(root);

}

Graphical user interface, text

Description automatically generated

1. All sorting algorithms covered in class

#include using namespace std;

// Function to sort an array using Insertion Sort

void insertionSort(int arr[], int n)

{ int i, key, j;

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

{ key = arr[i]; j = i - 1; // Move elements of arr[0..i-1], that are greater than key, to one //position ahead of their current position

while (j >= 0 && arr[j] > key) { arr[j + 1] = arr[j]; j = j - 1; }

arr[j + 1] = key;

} }

// Function to print an array

void printArray(int arr[], int n)

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

cout << arr[i] << " ";

cout << endl; }

int main() { int arr[] = { 12, 11, 13, 5, 6 };

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

insertionSort(arr, n);

printArray(arr, n); return 0; }

Text

Description automatically generated

## Bubble sort

Program : #include <iostream>

// Function to implement bubble sort void bubbleSort(int arr[], int n)

{

// flag to track if the list is already sorted bool sorted = false;

while (!sorted)

{ sorted = true;

// loop through the array and compare adjacent elements

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

{

// if the current element is greater than the next element, swap them

if (arr[i] > arr[i + 1])

{ int temp = arr[i]; arr[i] = arr[i + 1]; arr[i + 1] = temp;

// if a swap occurs, the list is not yet sorted sorted = false;

}

}

}

}

int main()

{ int arr[] = {5, 2, 1, 3, 4}; int n = sizeof(arr) / sizeof(arr[0]);

// call bubble sort function bubbleSort(arr, n);

// print the sorted array for (int i = 0; i < n; i++) std::cout << arr[i] << " ";

return 0;

}

Result:

Text

Description automatically generated

## Merge sort

Program : #include <iostream>

// Function to merge two sorted arrays void merge(int arr[], int left[], int leftSize, int right[], int rightSize)

{ int i = 0, j = 0, k = 0;

// loop until one of the arrays is fully traversed while (i < leftSize && j < rightSize)

{

// compare the elements of the two arrays and add the smaller element to the original array

if (left[i] < right[j]) arr[k++] = left[i++]; else arr[k++] = right[j++];

}

// add any remaining elements from the left array while (i < leftSize) arr[k++] = left[i++];

// add any remaining elements from the right array while (j < rightSize) arr[k++] = right[j++];

}

// Function to implement merge sort void mergeSort(int arr[], int n)

{

// base case: if the array has only one element, it is already sorted if (n < 2) return;

int mid = n / 2;

// create left and right arrays int \*left = new int[mid]; int \*right = new int[n - mid]; // copy the elements of the original array into the left and right arrays for (int i = 0; i < mid; i++) left[i] = arr[i]; for (int i = mid; i < n; i++) right[i - mid] = arr[i];

// sort the left and right arrays mergeSort(left, mid); mergeSort(right, n - mid);

// merge the left and right arrays back into the original array merge(arr, left, mid, right, n - mid);

// deallocate memory

delete[] left; delete[] right;

}

int main()

{ int arr[] = {56, 22, 13, 33, 41}; int n = sizeof(arr) / sizeof(arr[0]);

// call merge sort function mergeSort(arr, n);

// print the sorted array for (int i = 0; i < n; i++) std::cout << arr[i] << " ";

return 0;

}

Result:

Graphical user interface, text

Description automatically generated

## Selection sort

Program : #include <iostream>

// Function to implement selection sort void selectionSort(int arr[], int n)

{

// loop through the array

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

{

// assume the first element is the minimum int minIndex = i;

// find the index of the minimum element in the remaining portion of the array

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

{ if (arr[j] < arr[minIndex]) minIndex = j;

}

// if the minimum element is not the current element, swap them if (minIndex != i)

{ int temp = arr[i]; arr[i] = arr[minIndex]; arr[minIndex] = temp;

}

}

}

int main()

{ int arr[] = {56, 22, 13, 34, 49}; int n = sizeof(arr) / sizeof(arr[0]);

// call selection sort function selectionSort(arr, n);

// print the sorted array for (int i = 0; i < n; i++) std::cout << arr[i] << " ";

return 0;

}

Result:

Text

Description automatically generated

**17) Linear search and binary search**

## Linear search

Program : #include <iostream>

// Function to perform linear search int linearSearch(int arr[], int size, int key) { // Iterate through the array and search for the key

for (int i = 0; i < size; i++) { if (arr[i] == key) { return i; // Return the index if the key is found

}

}

return -1; // Return -1 if the key is not found

}

int main() { int arr[] = {4, 2, 5, 1, 3}; int size = sizeof(arr) / sizeof(arr[0]); // Calculate the size of the array int key = 3; // Element to search for

int index = linearSearch(arr, size, key); // Perform linear search

if (index == -1) { std::cout << "Element not found" << std::endl;

} else { std::cout << "Element found at index " << index << std::endl;

}

return 0;

}

Result :

Text

Description automatically generated

## Binary search

Program : #include <iostream>

using namespace std;

int binarySearch(int arr[], int size, int target) { // Set the left and right indices of the search area int left = 0; int right = size - 1;

// Keep searching until the search area is empty while (left <= right) {

// Find the middle index of the search area int middle = (left + right) / 2;

// Check if the target is at the middle index if (arr[middle] == target) { return middle;

}

// If the target is less than the value at the middle index,

// search the left half of the search area.

// Otherwise, search the right half.

if (arr[middle] > target) { right = middle - 1;

} else { left = middle + 1;

}

}

// If the target is not found, return -1 return -1; }

int main() {

// Create an array of sorted integers to search int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; int size = sizeof(arr) / sizeof(arr[0]);

// Perform a binary search for the target value 6 int target = 6; int result = binarySearch(arr, size, target);

// Print the result of the search if (result == -1) { cout << "Target not found" << endl;

} else { cout << "Target found at index " << result << endl;

}

return 0;

}

Result :

Text

Description automatically generated