Practical 1:

A.Basic program

#include <iostream> using namespace std; int main() {

cout << "Karthik reddy!" << endl; return 0;

}

B.add two numbers

#include <iostream> using namespace std; int main() {

int a, b, c, sum;

cout << "Enter three numbers: "; cin >> a >> b >> c;

sum = a + b + c;

cout << "The sum is: " << sum << endl; return 0;

}

C.area of rectangle

#include <iostream> using namespace std; int main() {

float length, width, area;

cout << "Enter length of the rectangle: "; cin >> length;

cout << "Enter width of the rectangle: "; cin >> width;

area = length \* width;

cout << "Area of the rectangle is: " << area << endl; return 0;

}

Practical 2:

A.Data types

#include <iostream>

#include <string> // Needed for string type using namespace std;

int main() {

int age = 21;

float height = 6.2f; char grade = 'B'; bool isPassed = true;

string name = "Karthik reddy";

cout << "Name: " << name << endl; cout << "Age (int): " << age << endl;

cout << "Height (float): " << height << endl; cout << "Grade (char): " << grade << endl; cout << "Passed (bool): " << isPassed << endl; return 0;

}

B.basic calculation

#include <iostream>

using namespace std; int main() {

int a, b;

cout << "Enter two numbers: "; cin >> a >> b;

cout << "Addition: " << a + b << endl; cout << "Subtraction: " << a - b << endl;

cout << "Multiplication: " << a \* b << endl; cout << "Division: " << a / b << endl;

cout << "Modulus: " << a % b << endl; return 0;

}

C.logical operators

#include <iostream>

using namespace std; int main() {

int a, b;

cout << "Enter two numbers: "; cin >> a >> b;

(a > 0 && b > 0) && cout << "Both numbers are positive.\n"; (a < 0 || b < 0) && cout << "At least one number is negative.\n";

!(a == b) && cout << "The numbers are not equal.\n"; (a == b) && cout << "The numbers are equal.\n";

return 0;

}

Practical 3: condition statements

A.even or odd

#include <iostream>

using namespace std; int main() {

int number;

cout << "Enter a number: "; cin >> number;

if (number % 2 == 0)

cout << number << " is Even." << endl; else

cout << number << " is Odd." << endl; return 0;

}

B.choice statement

#include <iostream>

using namespace std; int main() {

int choice;

cout << "Menu:\n"; cout << "1. Add\n"; cout << "2. Subtract\n"; cout << "3. Multiply\n"; cout << "4. Divide\n";

cout << "Enter your choice: "; cin >> choice;

int a, b;

cout << "Enter two numbers: "; cin >> a >> b;

switch (choice) {

case 1:

cout << "Sum = " << a + b << endl; break;

case 2:

cout << "Difference = " << a - b << endl; break;

case 3:

cout << "Product = " << a \* b << endl; break;

case 4:

if (b != 0)

cout << "Quotient = " << a / b << endl; else

cout << "Cannot divide by zero!" << endl; break;

default:

cout << "Invalid choice." << endl;

}

return 0;

}

C.

1. triangle pattern

#include <iostream> using namespace std; int main() {

int rows;

cout << "Enter number of rows: "; cin >> rows;

for (int i = 1; i <= rows; i++) { for (int j = 1; j <= i; j++) {

cout << "\* ";

}

cout << endl;

}

return 0;

}

2.sum of N numbers

#include <iostream> using namespace std; int main() {

int N, i = 1, sum = 0;

cout << "Enter a number N: "; cin >> N;

while (i <= N) { sum += i; i++;

}

cout << "Sum of first " << N << " numbers is: " << sum << endl;

return 0;

}

Practical 4:

A.Array

#include <iostream>

using namespace std;

int main() {

int arr[5] = {29,14,26,16,07};

cout << "Element at index 2: " << arr[2] << endl;

arr[3] = 60;

cout << "Modified element at index 3: " << arr[3] << endl;

int sum = 0;

for (int i = 0; i < 5; i++) {

sum += arr[i];

}

cout << "Sum of all elements: " << sum << endl;

return 0;

}

B.

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

int arr[5] = {16,14,26,29,05};

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

// Classic for loop instead of range-based loop

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

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

return 0;

}

C. reverse a string

#include <iostream>

#include<algorithm>

using namespace std;

int main() {

string s = "KARTHIK REDDY";

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

cout << s;

return 0;

}

Practical 5:

A.class code on constructor

#include <iostream>

using namespace std;

class Student {

private:

string name;

int rollNo;

float marks;

public:

// Default constructor

Student() {

name = "karthik";

rollNo = 4;

marks = 84.9;

cout << "above average student !" << endl;

}

// Function to display data

void display() {

cout << "Name: " << name << endl;

cout << "Roll No: " << rollNo << endl;

cout << "Marks: " << marks << endl;

}

};

int main() {

// Creating an object (default constructor is called automatically)

Student s1;

// Display initialized values

s1.display();

return 0;

}

B. Parametered construction

include <iostream>

using namespace std;

class Square {

private:

int side;

public:

// Parameterized constructor

Square(int s) {

side = s;

}

void area() {

cout << "Area of square = " << side \* side << endl;

}

};

int main() {

Square sq(6); // Passing side length as argument

sq.area();

return 0;

}

C. Copy constructor

#include <iostream>

using namespace std;

// Create a demo class

class A {

public:

int x;

};

int main() {

// Creating an a1 object

A a1;

a1.x = 15;

cout << "a1's x = " << a1.x << endl;

// Creating another object using a1 (Copy Constructor Calling)

A a2(a1);

cout << "a2's x = " << a2.x << endl;

return 0;

}

D. Destructor to show the object

#include <iostream>

using namespace std;

class MyClass {

private:

// Pointer to dynamically allocated memory

int\* data;

public:

// Constructor

MyClass(int value) {

data = new int; // allocate memory

\*data = value; // assign value

cout << "Value stored = " << \*data << endl;

}

// User-defined destructor: Free the dynamically allocated memory

~MyClass() {

delete data; // deallocate memory

cout << "Destruction: memory released" << endl;

}

};

int main() {

MyClass obj1(10); // create object

return 0;

}

Practical 6:

A.function overloading for area of circle and rectangle

#include <iostream>

using namespace std;

class Shape {

public:

// Member function: calculates area of a circle

void area(double radius) {

cout << "Area of circle = " << 3.14 \* radius \* radius << endl;

}

// Member function: calculates area of a rectangle

void area(double length, double width) {

cout << "Area of rectangle = " << length \* width << endl;

}

};

int main() {

Shape s; // object creation

s.area(6); // calls area() for circle

s.area(2,9 ); // calls area() for rectangle

return 0;

}

B.overload add of two complex numbers

#include <iostream>

using namespace std;

class Complex {

private:

float real, imag; // data members

public:

// Parameterized constructor

Complex(float r = 0, float i = 0) {

real = r;

imag = i;

}

// Operator overloading for +

Complex operator + (const Complex &obj) {

Complex temp;

temp.real = real + obj.real;

temp.imag = imag + obj.imag;

return temp;

}

// Display function

void display() {

cout << real << " + " << imag << "i" << endl;

}

};

int main() {

Complex c1(2.6, 2.9), c2(1.4, 1.6);

Complex c3 = c1 + c2; // calls overloaded + operator

cout << "First complex number: ";

c1.display();

cout << "Second complex number: ";

c2.display();

cout << "Sum: ";

c3.display();

return 0;

}

C.overload to compare two numbers

#include <iostream>

using namespace std;

class Student {

int rollNo;

public:

Student(int r) {

rollNo = r;

}

// Overload == operator

bool operator==(const Student &obj) {

return rollNo == obj.rollNo;

}

};

int main() {

Student s1(29), s2(29), s3(26);

if (s1 == s2)

cout << "s1 and s2 are same\n";

else

cout << "s1 and s2 are different\n";

if (s1 == s3)

cout << "s1 and s3 are same\n";

else

cout << "s1 and s3 are different\n";

return 0;

}

D.virtual function with override

#include <iostream>

using namespace std;

// Base class

class Shape {

public:

// Virtual function

virtual void draw() {

cout << "Creating a shape" << endl;

}

};

// Derived class

class Circle : public Shape {

public:

// Override virtual function

void draw() override {

cout << "Drawing a square" << endl;

}

};

int main() {

Shape\* shapePtr; // Base class pointer

Circle c;

shapePtr = &c; // Pointing to derived object

shapePtr->draw(); // Calls Circle's draw() due to virtual function

return 0;

}

Practical 7:

A.integer pointer and modify pointer

#include <iostream>

using namespace std;

int main() {

int num = 14; // Declare an integer variable

int \*ptr = &num; // Declare an integer pointer and store the address of num

cout << "Original value of num: " << num << endl;

\*ptr = 29; // Modify the value of num using the pointer

cout << "Modified value of num: " << num << endl;

return 0;

}

B.use pointer to access a array

#include <iostream>

using namespace std;

int main() {

int arr[5] = {2, 14, 26, 29, 30}; // Declare an array

int \*ptr = arr; // Pointer pointing to the first element of the array

cout << "Array elements using pointer: " << endl;

for (int i = 0; i < 5; i++) {

cout << \*(ptr + i) << " "; // Accessing elements using pointer

}

return 0;

}

C. Dynamically allocate memory by new

#include <iostream>

using namespace std;

int main() {

int size;

cout << "Enter the size of the array: ";

cin >> size;

// Dynamically allocate memory for an integer array

int \*arr = new int[size];

// Input values

cout << "Enter " << size << " elements: ";

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

cin >> arr[i]; // arr[i] is same as \*(arr + i)

}

// Display values

cout << "You entered: ";

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

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

}

// Free allocated memory

delete[] arr;

return 0;

}

Practical 8:

A.Single inheritance

#include <iostream>

using namespace std;

// Base class

class Parent {

public:

void displayParent() {

cout << "This is the Base class." << endl;

}

};

// Derived class (inherits from Parent)

class Child : public Parent {

public:

void displayChild() {

cout << "This is the Derived class." << endl;

}

};

int main() {

// Create object of Child class

Child obj;

// Access method of Parent class through Child object

obj.displayParent();

// Access method of Child class

obj.displayChild();

return 0;

}

B. Multiple inheritance

#include <iostream>

using namespace std;

// First base class

class Student {

public:

void displayStudent() {

cout << "This is Student class." << endl;

}

};

// Second base class

class Gaming {

public:

void displayGaming() {

cout << "This is Gaming class." << endl;

}

};

// Derived class inheriting from two base classes

class Result : public Student, public Gaming {

public:

void displayResult() {

cout << "This is Result class, combining Student and Gaming." << endl;

}

};

int main() {

// Create object of derived class

Result obj;

// Access methods from both base classes

obj.displayStudent();

obj.displayGaming();

// Access method from derived class

obj.displayResult();

return 0;

}

C.multiple- level inheritance

#include <iostream>

using namespace std;

// Base class

class Grandfather {

public:

void displayGrandfather() {

cout << "This is the Grandfather class: John." << endl;

}

};

// Derived from Grandfather

class Parent : public Grandfather {

public:

void displayParent() {

cout << "This is the Parent class: Michael." << endl;

}

};

// Derived from Parent

class Child : public Parent {

public:

void displayChild() {

cout << "This is the Child class: David." << endl;

}

};

int main() {

// Create object of Child class

Child obj;

// Access methods from all classes

obj.displayGrandfather(); // From Grandfather

obj.displayParent(); // From Parent

obj.displayChild(); // From Child

return 0;

}

D. Hierarchical inheritance

#include <iostream>

using namespace std;

// Base class

class Parent {

public:

void displayParent() {

cout << "This is the Parent class: Venkat reddy." << endl;

}

};

// First derived class

class Child1 : public Parent {

public:

void displayChild1() {

cout << "This is Child1 class: karthik." << endl;

}

};

// Second derived class

class Child2 : public Parent {

public:

void displayChild2() {

cout << "This is Child2 class: rohit reddyl." << endl;

}

};

int main() {

// Object of Child1

Child1 obj1;

cout << "Accessing through Child1 object:" << endl;

obj1.displayParent();

obj1.displayChild1();

cout << endl;

// Object of Child2

Child2 obj2;

cout << "Accessing through Child2 object:" << endl;

obj2.displayParent();

obj2.displayChild2();

return 0;

}

Practical 9:file handling

A.create a file using stream

#include <iostream>

#include <fstream> // Required for file handling

using namespace std;

int main() {

// Create an ofstream object

ofstream outFile;

// Open a file in write mode

outFile.open("example.txt");

if (!outFile) {

cout << "Error opening file!" << endl;

return 1;

}

// Write to the file

outFile << "Hello, this is a sample text file." << endl;

outFile << "Writing data using ofstream in C++." << endl;

// Close the file

outFile.close();

cout << "Data input to the file successfully!" << endl;

return 0;

}

B.create a file using ifstream

#include <iostream>

#include <fstream> // Required for file handling

#include <string> // Required for using string

using namespace std;

int main() {

// Create an ifstream object

ifstream inFile;

// Open the file in read mode

inFile.open("example.txt");

if (!inFile) {

cout << "Error opening file!" << endl;

return 1;

}

string line;

cout << "Output from file:" << endl;

// Read the file line by line

while (getline(inFile, line)) {

cout << line << endl;

}

// Close the file

inFile.close();

return 0;

}

C.create a file for student

#include <iostream>

#include <fstream>

using namespace std;

class Student {

public:

int roll;

char name[50];

float marks;

void input() {

cout << "Enter Roll Number: ";

cin >> roll;

cin.ignore(); // Ignore leftover newline

cout << "Enter Name: ";

cin.getline(name, 50);

cout << "Enter Marks: ";

cin >> marks;

}

void display() {

cout << "Roll: " << roll

<< ", Name: " << name

<< ", Marks: " << marks << endl;

}

};

int main() {

Student s;

fstream file;

// Open file in binary write mode

file.open("student.dat", ios::out | ios::binary);

if (!file) {

cout << "Error opening file!" << endl;

return 1;

}

int n;

cout << "How many student records do you want to enter? ";

cin >> n;

cin.ignore(); // Ignore leftover newline

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

cout << "\nEnter details for student " << i + 1 << ":\n";

s.input();

file.write((char\*)&s, sizeof(s)); // Write object in binary

}

file.close();

cout << "\nStudent records saved to student.dat successfully!" << endl;

return 0;

}

Practical 10: exception handling

A.use try throw catch in program for division

#include <iostream>

using namespace std;

int main() {

int num, den;

cout << "Enter numerator: ";

cin >> num;

cout << "Enter denominator: ";

cin >> den;

try {

if (den == 0) {

throw "Division by zero is not allowed!"; // Throwing an exception

}

double result = (double)num / den;

cout << "Result: " << result << endl;

}

catch (const char \*msg) {

cout << "Exception: " << msg << endl; // Handling the exception

}

cout << "Program continues after exception handling." << endl;

return 0;

}

B.use multiple catch(int,char,string)

#include <iostream>

#include <string>

using namespace std;

int main() {

int choice;

cout << "Enter your choice (1: int, 2: char, 3: string): ";

cin >> choice;

try {

if (choice == 1) {

throw 100; // Throw integer exception

} else if (choice == 2) {

throw 'A'; // Throw character exception

} else if (choice == 3) {

throw string("This is a string exception"); // Throw string exception

} else {

cout << "No exception thrown!" << endl;

}

}

catch (int e) {

cout << "Caught an integer exception: " << e << endl;

}

catch (char e) {

cout << "Caught a character exception: " << e << endl;

}

catch (string e) {

cout << "Caught a string exception: " << e << endl;

}

cout << "Program continues after exception handling." << endl;

return 0;

}

C.Validate input(negative age) and throw

Custom exception

#include <iostream>

#include <string>

using namespace std;

// Custom Exception Class

class InvalidAgeException {

public:

string message;

InvalidAgeException(string msg) {

message = msg;

}

};

int main() {

int age;

cout << "Enter your age: ";

cin >> age;

try {

if (age < 5) {

throw InvalidAgeException("Age must be at least 5 years!");

}

if (age > 120) {

throw InvalidAgeException("Age cannot exceed 120 years!");

}

cout << "Your age is: " << age << endl;

}

catch (InvalidAgeException &e) {

cout << "Exception: " << e.message << endl;

}

cout << "Program continues after exception handling." << endl;

return 0;

}