

INSTITUTE :- TOPS TECHNOLOGIES (BARODA)

NAME :- SRUJAL SOLANKI M

COURSE :- JAVA FULL STACK DEVELOPER

MODULE 3 - OOPS PROGRAMMING

\* LAB EXERCISE \*

1. First C++ Program: Hello World

Write a simple C++ program to display "Hello, World!".

-> #include<iostream>

using namespace std;

int main(){

cout<< "hello, world!";

}

2. Basic Input/Output :

Write a C++ program that accepts user input for their name and age and then displays a personalized greeting.

-> #include<iostream>

#include<string>

using namespace std;

int main(){

string name;

int age;

cout << "please enter your name :";

cin >> name;

cout << "please enter your age :";

cin >> age;

cout << "your name is " << name << endl;

cout << "your age is " << age << endl;

return 0;

}

3. POP vs. OOP Comparison Program

Write two small programs: one using Procedural Programming (POP) to calculate the area of a rectangle, and another using Object-Oriented Programming (OOP) with a

class and object for the same task.

-> using POP

#include<iostream>

using namespace std;

int area\_of\_rectangle(int length,int width);

int main(){

int length,width;

cout <<"enter the length of area : ";

cin >> length;

cout <<"enter the width of area : ";

cin >> width;

int area = area\_of\_rectangle(length,width);

cout <<"the area of rectangle is " << area;

return 0;

}

int area\_of\_rectangle(int length,int width){

return length \* width;

}

-> using OOP

#include<iostream>

using namespace std;

class rectangle{

public:

int length;

int width;

public:

rectangle(int l,int w){

length = l;

width = w;

}

int area(){

return length \* width;

}

};

int main(){

int length,width;

cout <<"enter the length of area : ";

cin >> length;

cout <<"enter the width of area : ";

cin >> width;

rectangle rect(length,width);

cout << "the area of rectangle is "<< rect.area()<< endl;

return 0;

}

4. Setting Up Development Environment

Write a program that asks for two numbers and displays their sum.

-> #include<iostream>

using namespace std;

int main(){

int firstnum, secondnum;

cout << "enter the firstnum : ";

cin >> firstnum;

cout << "enter the secondnum : ";

cin >> secondnum;

int sum = firstnum + secondnum;

cout << "the sum of num is "<< sum << endl;

return 0;

}

\* THEORY EXERCISE \*

1. What are the key differences between Procedural Programming and Object-Oriented Programming (OOP)?

-> 1. Approach :

-> procedural - focuses on writing functions or procedures to perform tasks.

-> OOP - focuses on creating objects that combine both data and methods.

2. Data and Behavior :

-> procedural - Data and functions are separate, and functions operate on the data

-> OOP - Data and behavior are bundled together inside objects.

3. Modularity :

-> Procedural - Relies on dividing the program into smaller functions.

-> OOP - uses classes and objects to structure code, making it easier to manage.

4.Reusability :

-> procedural - Functions can be reused but often require modifications for different use cases.

-> OOP - Reuses code through inheritance, making it more efficient.

2.List and explain the main advantages of OOP over POP.

->1.Encapsulation: Protects data by bundling it with methods, ensuring security.

2.Reusability: Supports inheritance and polymorphism for efficient code reuse.

3.Modularity: Organizes code into classes and objects for better structure and readability.

4.Maintainability: Easier to update and manage individual objects without affecting others.

5.Scalability: Simplifies scaling with modular design and new object additions.

6.Real-World Mapping: Models real-world entities more intuitively with objects.

7.Abstraction: Hides unnecessary details, showing only relevant functionality.

8.Flexibility: Allows dynamic behavior through polymorphism and extensibility.

3. Explain the steps involved in setting up a C++ development environment.

-> STEP 1. To install C compiler we have to install MinGW(minimalist GNU for windows):

1.download the installer from MinGW official site.

2.Run the installer and select gcc from the "basic setup"section.

3.Add the bin directory(e.g, C: MinGW\bin) to the system PATH.

4.open a terminal and verify the installation.

-> STEP 2. how to download devlopment ENvironment like VS code.

-> To install VS code : we have to go to Visual Studio official site and download VS code.

-> Install C/C++ extension :

-> TO install C extension open VScode and go to extension and search C/C++ by microsoft and install it.

-> Step 3. Test your setup :

-> open your VScode.

-> create a new file (e.g., hello.cpp) and print "hello world".

4. What are the main input/output operations in C++? Provide examples.

-> C++ primarily uses the <iostream> library for input and output operations.

-> (cin)Used to read data from the user via the standard input (keyboard).

Syntax: cin >> variable;

-> (cout)Used to display data to the user via the standard output (console).

Syntax: cout << value;

-> Formatted Input/Output (setw, setprecision)

Requires <iomanip> library for formatting.

--> EXAMPLE :#include <iostream>

#include <iomanip>

using namespace std;

int main() {

double num = 123.45678;

cout << "Default: " << num << endl;

cout << "Formatted: " << fixed << setprecision(2) << num << endl;

return 0;

}

\* THEORY EXERCISE \*

1. What are the different data types available in C++? Explain with examples.

> DATA TYPES :

> Data types specify the types of data a variable can hold.

> common data types :

>> primitive datatypes :

> int : for integers.

> float : for floating point numbers.Float is used for decimal numbers.

> char : for single characters.it only accept single characters.

> double : for double precision floating point numbers.

> void : for functions that do not return a value.

-> EXAMPLE : int age = 25;

float temperature = 43.5;

char grade = 'A'; single cot is compulsary in char datatype.

double pi = 3.141;

>> derived datatypes :

> arrays : Represents a collection of elements of the same type.

> pointers : Stores memory addresses of other variables.

> functions : Represents blocks of reusable code.

2. Explain the difference between implicit and explicit type conversion in C++.

>> Implicit type conversion is performed automatically by the c++ compiler when it encounters a situation where a datatypes needs to be converted to another type for compatibility.

> EXAMPLE : #include <iostream>

using namespace std;

int main() {

int a = 5;

double b = 4.5;

double result = a + b;

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

output : 9.5

return 0;

}

>> Explicit type conversion requires the programmer to manually specify the desired type to which a variable should be converted.

3. What are the different types of operators in C++? Provide examples of each.

1 > Arithmetic operator :

> The operators perform basic mathematical operation like.

> '+' - addition,

> '-' - subtraction,

> '\*' - multiplication,

> '/' - divison,

> '%' - modulus (Remainder when a is divided by b).

2 > Relational operator :

> The relational operators compare two values and return 1 (true) and 0 (false).

> '==' - equal to (a == b(true if a equals to b))

> '!=' - not equal to (a != b(true if a is not equal to b))

> '<' - less than (a < b(true if a less than b))

> '>' - greater than (a > b(true if a is greater than b))

> '<=' - less than or equal to (a <= b(true if a is less than or equal to b))

> '>=' - greater than or equal to (a >= b(true if a is greater than or equal to b))

3 > Logical operator :

> These operators are used to combine multiple conditions.

> '&&' - Logical AND example: a = 1, b = 1 (a > 3) && (b > 2) (true if both conditions are true)

> '!' - Logical NOT example: a = 5, b = 10 (a != b) (true if a is not equal to b)

> '||' - Logical OR example: a = 5, b = 10 (a > 4) || (b > 11) (true if atleast one condition is true)

4 > Assignment operator :

> = Assign a = b (Assigns b to a)

> += Add and assign a += b (Equivalent to a = a + b)

> -= Subtract and assign a -= b (Equivalent to a = a - b)

> \*= Multiply and assign a \*= b (Equivalent to a = a \* b)

> /= Divide and assign a /= b (Equivalent to a = a / b)

> %= Modulus and assign a %= b (Equivalent to a = a % b)

5 > Increment/Decrement operator :

> these operator increases or decreases value by 1.

> '++' increment by 1

> '--' decrement by 1

6 > Bitwise operator :

> bitwise operator perform operations on binary represent of number.

> '&' Bitwise AND

> '|' Bitwise OR

> '~' Bitwise NOT

> '<<' Bitwise LEFT SHIFT

> '>>' Bitwise RIGHT SHIFT

7 > conditional(ternary)operator :

> ?: Example : a = 10,b = 20; (a > b) ? a : b;

4. Explain the purpose and use of constants and literals in C++.

> A constant in C++ refers to a fixed value that cannot be altered during the execution of the program. These are used to represent values that should remain unchanged to ensure program integrity and readability.

Types of Constants in C++ :

>Literal Constants:

->Values directly written in the code (e.g., 10, 3.14, 'A').

>const Keyword:

->Used to define named constants.

>Enumerations (enum):

->Used to define a set of named integral constants.

>Preprocessor Constants (#define):

-> Defined using macros before compilation.

2. Variables, Data Types, and Operators

\* LAB EXERCISE \*

--> Write a C++ program that demonstrates the use of variables and constants. Create variables of different data types and perform operations on them.

-> #include <iostream>

using namespace std;

int main() {

int num1 = 10;

float num2 = 5.5;

char grade = 'A';

bool isPassing = true;

string name = "Alice";

const double PI = 3.14159;

int sum = num1 + 15;

float product = num2 \* 2.0;

bool isExcellent = (grade == 'A');

double area = PI \* 5 \* 5;

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

cout << "Sum of " << num1 << " and 15: " << sum << endl;

cout << "Product of " << num2 << " and 2: " << product << endl;

cout << "Grade: " << grade << ", Passing: " << isPassing << endl;

cout << "Is Excellent: " << (isExcellent ? "Yes" : "No") << endl;

cout << "Area of circle with radius 5: " << area << endl;

return 0;

}

--> Write a C++ program that performs both implicit and explicit type conversions and prints the results.

-> #include<iostream>

using namespace std;

int main(){

int integer = 5;

double implicitresult = (double)integer;

double pi = 3.14;

int explicitresult = (int)pi;

cout << "implicit conversion (int to double) : "<<implicitresult<< endl;

cout << "explicit conversion (double to int) : "<<explicitresult<< endl;

return 0;

}

-->Operator Demonstration

-> Write a C++ program that demonstrates arithmetic, relational, logical, and bitwise operators. Perform operations using each type of operator and display the results.

> ARITHMETIC OPERATOR : // (+,-,\*,/,%)

#include<iostream>

using namespace std;

int main(){

int firstnum,secondnum;

cout <<"enter the firstnum : ";

cin >> firstnum;

cout<<"enter the secondnum : ";

cin >> secondnum;

cout <<"the sum of first and second num is : " <<firstnum + secondnum << endl;

cout <<"the substraction of first and second num is : " <<firstnum - secondnum << endl;

cout <<"the multiply of first and second num is : " <<firstnum \* secondnum << endl;

cout <<"the divsion of first and second num is : " <<firstnum / secondnum << endl;

cout <<"the modulus of first and second num is : " <<firstnum % secondnum << endl;

return 0;

}

> Relational OPERATER : (==, >, <, !=, <=, >=)

#include<iostream>

using namespace std;

int main(){

int a,b;

cout << "enter the value of a :";

cin >> a;

cout << "enter the value of b :";

cin >> b;

cout << "a == b is :" <<( a == b) <<"\n";

cout << "a < b is :" <<( a < b) << "\n";

cout << "a > b is :" <<( a > b) << "\n";

cout << "a != b is :" <<( a != b) << "\n";

cout << "a >= b is :" <<( a >= b) << "\n";

cout << "a <= b is :" <<( a <= b) << "\n";

return 0;

}

> LOGICAL OPERATOR : (&& , ||, !)

#include<iostream>

using namespace std;

int main(){

int a,b;

cout << "enter the value of a :";

cin >> a;

cout << "enter the value of b :";

cin >> b;

cout << "a && b is :" <<( (a < b) && (a > b)) <<"\n";

cout << "a || b is :" <<((a < b) || (a > b)) << "\n";

cout << "a ! b is :" <<((a < b) !=(a > b) ) << "\n";

return 0;

}

> BITWISE OPERATOR : (&, |, ~)

#include<iostream>

using namespace std;

int main(){

int a,b;

cout << "enter the value of a :";

cin >> a;

cout << "enter the value of b :";

cin >> b;

int result1 = a & b;

int result2 = a | b;

cout << "a & b is :" << result1 << "\n";

cout << "a | b is :" << result2 << "\n";

return 0;

}

\* CONTROL FLOW-STATEMENTS \*

\* THEORY EXERCISE \*

1.What are conditional statements in C++? Explain the if-else and switch statements.

> Conditional statements in C++ are used to make decisions in a program based on certain conditions. They allow the program to execute specific blocks of code when a condition evaluates to true, and optionally execute alternative blocks of code when the condition is false.

-> If-else statement : If- else statement is used to execute code blocks based on a Boolean condition.

-> Switch statement :the switch statement is used to execute one block of code out of multiple options based on the value of an expression. It is typically used when there are many specific values to test.

2.What is the difference between for, while, and do-while loops in C++?

> For loop : for loop is typically used when the number of iterations is known beforehand

> while loop : The while loop is used when the condition needs to be checked before entering the loop, and the number of iterations is not predetermined.

> do while loop : The do-while loop guarantees that the loop body will execute at least once, as the condition is checked after executing the loop body.

3.How are break and continue statements used in loops? Provide examples.

> Break statement : The break statement is used to exit a loop immediately, regardless of the loop's condition. Once a break statement is executed, the control jumps to the first statement after the loop.

-> Example :#include <iostream>

using namespace std;

int main() {

for (int i = 1; i <= 10; i++) {

if (i == 5) {

break; // Exit the loop when i equals 5

}

cout << "i = " << i << endl;

}

cout << "Loop terminated." << endl;

return 0;

}

> Continue statement : The continue statement skips the remaining code in the current iteration of the loop and proceeds to the next iteration. It does not terminate the loop but simply skips the rest of the loop body for that iteration.

-> Example :#include <iostream>

using namespace std;

int main() {

for (int i = 1; i <= 10; i++) {

if (i % 2 == 0) {

continue; // Skip the rest of the loop body for even numbers

}

cout << "i = " << i << endl;

}

return 0;

}

4.Explain nested control structures with an example.

> nested control structures occur when one control structure is placed inside another.These are used to handle more complex logic by combining different control mechanisms.

-> Example : #include<iostream>

using namespace std;

int main(){

int age, salary;

cout << "enter your age : ";

cin >> age;

if (age > 20){

cout << "enter your salary : ";

cin >> salary;

if(salary >= 8000){

cout << "your are eligible for credit card";

}

else {

cout << "your are not eligible for credit card";

}

}

return 0;

}

\* LAB EXERCISE \*

> Write a C++ program that takes a student’s marks as input and calculates the grade based on if-else conditions.

-> #include<iostream>

using namespace std;

int main(){

int marks;

cout << "enter the students marks : ";

cin >> marks;

if (marks > 90 && marks < 100){

cout << "grade is A+";

}else if(marks > 70 && marks < 90){

cout << "grade is B";

}else if(marks > 50 && marks < 70){

cout << "grade is C";

} else {

cout << "grade is D";

}

return 0;

}

> Write a C++ program that asks the user to guess a number between 1 and 100. The program should provide hints if the guess is too high or too low. Use loops to allow the user multiple attempts.

-> #include<iostream>

#include<time.h>

#include<cstdlib>

using namespace std;

int main(){

srand(time(NULL));

int random,guess;

int no\_of\_guesses = 0;

random = rand() % 100 + 1;

do

{ cout << "enter the number between 1 to 100 : ";

cin >> guess;

no\_of\_guesses++;

if(guess > random){

cout << "enter smaller number\n";

}else if (guess < random){

cout << "enter bigger number\n";

}else{

cout << "you guessed correct number in tries :" << no\_of\_guesses;

}

} while (guess != random);

return 0;

}

> Write a C++ program to display the multiplication table of a given number using a for loop.

-> #include<iostream>

using namespace std;

int main(){

int num;

cout << "enter the number : ";

cin >> num;

for(int i = 1; i <= 10; i++){

cout << num << " X " << i << " = " <<num \* i << endl;

}

return 0;

}

> Write a program that prints a right-angled triangle using stars (\*) with a nested loop.

-> #include<iostream>

using namespace std;

int main(){

int rows;

cout << " enter the rows : ";

cin >> rows;

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

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

cout << "\* ";

}

cout << "\n";

}

return 0;

}

FUNCTIONS AND SCOPE

\* THEORY EXERCISE \*

1. What is a function in C++? Explain the concept of function declaration, definition, and

calling.

-> In C++, a function is a block of code designed to perform a specific task. It provides

modularity, reusability, and better organization for code. A function can take input in the form of parameters and return output as a result.

->Function Declaration (Prototype)

A function declaration tells the compiler about the function's name, return type, and parameters without providing the actual implementation.

->Function Definition

The function definition provides the actual implementation of the function. It includes the function header (same as the declaration) and a block of code enclosed in {}.

->Function Call

To execute a function, you call it from another function, such as main(), by using its name and passing arguments that match the parameter list.

2. What is the scope of variables in C++? Differentiate between local and global scope.

->In C++, the scope of a variable defines the region of the program where the variable can

be accessed or modified. There are primarily two types of variable scopes: local and global.

->Local Scope

A variable with local scope is declared within a block, function, or method, and its accessibility is limited to that block.

-> Global Scope

A variable with global scope is declared outside of all functions and blocks, making it accessible throughout the program (from the point of declaration onward).

3. Explain recursion in C++ with an example.

->Recursion is a programming concept where a function calls itself directly or indirectly to solve a problem. It is typically used for problems that can be divided into smaller subproblems of the same type. A recursive function must have a base case to terminate the recursion and prevent infinite loops.

> example :#include<iostream>

using namespace std;

int factorial(int n);

int main(){

int number;

cout << "enter a positive number :";

cin >> number;

if (number < 0){

cout << "factorial is not defined for negative numbers.";

}else {

cout << "factorial of " << number << " is:" << factorial(number) << endl;

}

return 0;

}

int factorial(int n){

if (n == 0){

return 1;

}else {

return n \* factorial(n - 1);

}

}

4. What are function prototypes in C++? Why are they used?

-> A function prototype in C++ is a declaration of a function that specifies its name, return type, and parameters without defining the body of the function. It serves as a promise to the compiler about the existence of a function, allowing the function to be used before it is fully defined.

\* LAB EXERCISE \*

1.Write a C++ program that defines functions for basic arithmetic operations (add,subtract,

multiply, divide). The main function should call these based on user input.

->#include<iostream>

using namespace std;

int choose(int choice,int a,int b);

int main(){

int a,b,choice;

cout << "choose one operator :";

cout << "\n1.addition\n2.subtraction\n3.multiply\n4.division : ";

cin >> choice;

cout << "enter the number of a : ";

cin >> a;

cout << "enter the number of b : ";

cin >> b;

int result = choose (choice,a,b);

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

return 0;

}

int choose( int choice, int a, int b){

switch(choice){

case 1: return a + b;

case 2: return a - b;

case 3: return a \* b;

case 4:

if (b != 0)

return a / b;

else {

cout << "error! division by zero";

}

default:

cout << "invalid choice";

return 0;

}

}

2. Write a C++ program that calculates the factorial of a number using recursion.

-> #include<iostream>

using namespace std;

int factorial(int n);

int main(){

int number;

cout << "enter a positive number :";

cin >> number;

if (number < 0){

cout << "factorial is not defined for negative numbers.";

}else {

cout << "factorial of " << number << " is:" << factorial(number) << endl;

}

return 0;

}

int factorial(int n){

if (n == 0){

return 1;

}else {

return n \* factorial(n - 1);

}

}

\* ARRAYS and STRINGS \*

\* THEORY EXERCISE \*

1.What are arrays in C++? Explain the difference between single-dimensional and multi-

dimensional arrays.

-> In C++, array is a collection of elements of the same data type stored in contiguous memory locations. Arrays provides a way to group similar data together and access using indices.

-> there are two types of array.

> single dimensional array

> multi dimensional array

--> single dimensional array : a single dimensional array is essentially a list of elements of the same type stored in a single row.

--> multi dimensional array : a multi dimensional array is an array of arrays. It can have two or more dimensions.

2. Explain string handling in C++ with examples.

-> In C++, strings are sequences of characters used to store and manipulate text. String handling can be done using two primary methods :

> C-style strings: These are character arrays (char[]).

> C++ Standard Template Library (STL) strings: These are implemented using the std::string class, which provides more functionality and ease of use.

3. How are arrays initialized in C++? Provide examples of both 1D and 2D arrays.

-> arrays can be initialized in several ways. The method of initialization depends on the type of array (1D or 2D) and whether you explicitly provide values.

> example of 1d array :

#include<iostream>

using namespace std;

int main(){

int arr[5] = {1,2,3,4,5};

int aarr[5] = {1,2};

int barr[5] = {};

cout << "arr1 :";

for(int i = 0; i < 5; i++) cout << arr[i] << " ";

cout << "\naarr2 : ";

for(int i = 0; i < 5; i++) cout << aarr[i] << " ";

cout << "\nbarr3 : ";

for(int i = 0; i < 5; i++) cout << barr[i] << " " ;

return 0;

}

> example of 2d array :

#include<iostream>

using namespace std;

int main(){

int arr[2][3] = {{1,2,3},{4,5,6}};

int aarr[2][3] = {{1,2},{3}};

int barr[2][3] = {};

cout << "\narr :";

for (int i = 0; i < 2; i++){

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

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

}

}

cout << "\naarr :";

for(int i = 0; i < 2; i++){

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

cout << aarr[i][j] << " ";

}

}

cout << "\nbarr :";

for(int i = 0; i < 2;i++){

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

cout << barr[i][j] << " ";

}

}

return 0;

}

\* LAB EXERCISE \*

1. array sum and average :Write a C++ program that accepts an array of integers, calculates the sum and average, and displays the results.

-> #include<iostream>

using namespace std;

int main(){

int num;

cout << "enter the number of elements : ";

cin >> num;

int arr[num];

cout << "enter " << num << " integers :";

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

cin >> arr[i];

}

int sum = 0;

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

sum += arr[i];

}

int avg = sum / num;

cout <<" sum : " << sum << endl;

cout << " avg : " << avg;

return 0;

}

2. Write a C++ program to perform matrix addition on two 2x2 matrices.

->

#include<iostream>

using namespace std;

int main(){

int matrix1[2][2];

int matrix2[2][2];

int result[2][2];

cout << "enter elements for first 2x2 matrix : " << endl;

for(int i = 0; i < 2; i++){

for(int j = 0; j < 2; j++){

cin >> matrix1[i][j];

}

}

cout << "enter elements for second 2x2 matrix : " << endl;

for(int i = 0; i < 2; i++){

for(int j = 0; j < 2; j++){

cin >> matrix2[i][j];

}

}

for(int i = 0; i < 2; i++){

for(int j = 0; j < 2; j++){

result[i][j] = matrix1[i][j] + matrix2[i][j];

}

}

cout << "the resulting matrix after addition is : " << endl;

for(int i = 0; i < 2; i++){

for(int j = 0; j < 2; j++){

cout << result[i][j];

}

cout << endl;

}

return 0;

}

3. Write a C++ program to check if a given string is a palindrome (reads the same forwards and backwards).

-> #include <iostream>

#include <string>

using namespace std;

int main() {

string str, reversedStr;

cout << "Enter a string: ";

cin >> str;

reversedStr = string(str.rbegin(), str.rend());

if (str == reversedStr) {

cout << "The string \"" << str << "\" is a palindrome." << endl;

} else {

cout << "The string \"" << str << "\" is not a palindrome." << endl;

}

return 0;

}

2. What are classes and objects in C++? Provide an example.

->

\* INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING \*

\* THEORY EXERCISE \*

1. Explain the key concepts of Object-Oriented Programming (OOP).

-> Object-Oriented Programming (OOP) is a programming paradigm centered around the concept of objects, which combine data and behavior.

> class/object - template and instance of the blueprint, respectively.

> Encapsulation - protects data by bundling it with methods and restricting direct access.

> Inheritance - enables code reuse by inheriting features of a parent class.

> polymorphism - Allows different behaviors for the same method depending on the object.

> Abstractions - Hides implementation details and exposes only relevant functionality.

2.What are classes and objects in c++? provide an example.

-> classes and objects are the foundation of object-oriented programming (OOP).

> class: A class is a user-defined data type that serves as a blueprint for creating objects. It encapsulates data (attributes) and functions (methods) that operate on that data.

> object: An object is an instance of a class. It is a specific realization of the class blueprint, containing its own copies of the data members and having the ability to use the methods defined by the class.

> Example :#include<iostream>

using namespace std;

class car{

public :

string brand;

int speed;

car(string carbrand, int carspeed){

brand = carbrand;

speed = carspeed;

}

void display(){

cout << "car brand is :"<< brand <<", car speed is : "<< speed <<" km/h"<<endl;

}

void updatespeed(int newspeed){

speed = newspeed;

cout << "speed is updated"<< speed << "km/h"<< endl;

}

};

int main()

{

car skoda("kushaq",150);

car audi("AUDI A6",190);

car BMW("BMW X7",320);

skoda.display();

audi.display();

BMW.display();

skoda.updatespeed(250);

audi.updatespeed(300);

return 0;

}

3. What is inheritance in C++? Explain with an example.

-> Inheritance is a key concept in object-oriented programming (OOP) that allows a class to acquire the properties and behaviors of another class. The class from which properties are inherited is called the base class (or parent class), and the class that inherits is called the derived class (or child class).

\* Types of inheritance in C++

> Single Inheritance: A derived class inherits from one base class.

> Multiple Inheritance: A derived class inherits from more than one base class.

> Multilevel Inheritance: A class inherits from a derived class, creating a chain.

> Hierarchical Inheritance: Multiple derived classes inherit from a single base class.

> Hybrid Inheritance: A combination of two or more types of inheritance.

> Exmaple :#include<iostream> // inheritance function

using namespace std;

class polygon{

protected : // protected

int width, height;

public :

void set\_value(int a, int b){

width = a;

height = b;

}

};

class rectangle : public polygon{

public:

int area()

{

return width \* height;

}

};

class triangle : public polygon{

public :

int area()

{

return width \* height /2;

}

};

int main()

{

rectangle rect;

triangle trgl;

rect.set\_value (7,5);

trgl.set\_value (9,7);

cout << rect.area() <<'\n';

cout << trgl.area() <<'\n';

return 0;

}

4. What is encapsulation in C++? How is it achieved in classes?

-> Encapsulation is an object-oriented programming concept that refers to the bundling of data (variables) and functions (methods) into a single unit, typically a class. It helps in protecting the data from unauthorized access and ensures that the internal workings of a class are hidden from the outside world. This is achieved through access specifiers like private, protected, and public.

-> Encapsulation is achieved by:

> Declaring the data members as private or protected.

> Providing public member functions (getters and setters) to access and modify the

data.

\* LAB EXERCISE \*

1.Write a C++ program that defines a class Calculator with functions for addition,

subtraction, multiplication, and division. Create objects to use these functions.

-> #include<iostream>

#include<string>

using namespace std;

class calculator{

public:

int add(int a, int b){

return a + b;

}

int subtract(int a, int b){

return a - b;

}

int multiply (int a, int b){

return a \* b;

}

int divide(int a, int b){

return a / b;

}

};

int main(){

calculator calc;

int num1,num2;

cout << "enter two number for addition :";

cin >> num1 >> num2;

cout << "result of addition is : " << calc.add(num1,num2) << endl;

cout << "enter two number for subtraction :";

cin >> num1 >> num2;

cout << "result of addition is : " << calc.subtract(num1,num2) << endl;

cout << "enter two number for multiply :";

cin >> num1 >> num2;

cout << "result of addition is : " << calc.multiply(num1,num2) << endl;

cout << "enter two number for divide :";

cin >> num1 >> num2;

cout << "result of addition is : " << calc.divide(num1,num2) << endl;

return 0;

}

2.Create a class BankAccount with data members like balance and member functions like deposit and withdraw. Implement encapsulation by keeping the data members private.

-> #include<iostream>

#include<string>

using namespace std;

class bankaccount{

private:

string accountholder;

double balance; // data hiding

public:

bankaccount(string name,double initialbalance){

accountholder = name;

balance = initialbalance;

}

void deposit(double amount){

balance = balance + amount;

cout << "deposited amount : "<< amount <<". new balance : " << balance <<"\n";

}

void withdrawl(double amount){

if(amount > balance){

cout <<"insuffcient balance" <<"\n";

}else {

balance = balance - amount;

cout << "withdrawl amount is : " << amount<<". new balance : "<<balance <<"\n";

}

}

void display(){

cout <<"accountholder : "<< accountholder << ", current balance is : " << balance <<"\n";

}

};

int main()

{

bankaccount account("dhanish",10000);

account.display();

account.deposit(50000);

account.withdrawl(15000);

account.display();

return 0;

}

3.Write a program that implements inheritance using a base class Person and derived

classes Student and Teacher. Demonstrate reusability through inheritance.

-> #include <iostream>

#include <string>

using namespace std;

// Base class

class Person {

protected:

string name;

int age;

public:

// Constructor for Person

Person(string personName, int personAge) : name(personName), age(personAge) {}

// Method to display basic details

void displayBasicInfo() {

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

cout << "Age: " << age << endl;

}

};

// Derived class: Student

class Student : public Person {

private:

string studentID;

public:

// Constructor for Student

Student(string studentName, int studentAge, string id)

: Person(studentName, studentAge), studentID(id) {}

// Method to display student-specific details

void displayStudentInfo() {

displayBasicInfo();

cout << "Student ID: " << studentID << endl;

}

};

// Derived class: Teacher

class Teacher : public Person {

private:

string subject;

public:

// Constructor for Teacher

Teacher(string teacherName, int teacherAge, string subjectSpecialization)

: Person(teacherName, teacherAge), subject(subjectSpecialization) {}

// Method to display teacher-specific details

void displayTeacherInfo() {

displayBasicInfo();

cout << "Subject: " << subject << endl;

}

};

int main() {

// Create a Student object

Student student1("Alice", 20, "S12345");

cout << "Student Details:" << endl;

student1.displayStudentInfo();

// Create a Teacher object

Teacher teacher1("Mr. Smith", 40, "Mathematics");

cout << "\nTeacher Details:" << endl;

teacher1.displayTeacherInfo();

return 0;

}