* **Module-3 Introduction to OOPS Programming**:

Q.1 Introduction to C++:

* **What are the key differences between procedural programming and object-oriented programming?**
* Answer:

|  |  |  |
| --- | --- | --- |
| Features | Procedural programming | Object – oriented programming |
| Approach | Follows a top-down approach | Follows a bottom-up approach |
| Focus | Focuses on function or procedures | Focuses on object and classes |
| Data handling | Data is global and shared between functions | Data is encapsulated within objects |
| Security | Less secure | More secure |
| Reusability | Limited code reuse | Promotes inheritance |

* **List and explain the main advantages of OOP over POP**
* Answer:

1. Modularity:

* Code is divided into classes and object , making it easier to manage and organize

1. Reusability:

* Classes can be reused in different programs using inheritance

1. Encapsulation:

* Data is hidden inside objects, protecting it from unauthorized access.

1. Abstraction:

* Complex details are hidden, only essential features are exposed to user

1. Inheritance :

* One class can inherit features from another, reducing code duplication

1. Polymorphism:

* Same function or method name can behave differently based on the object type

1. Better for large projects:

* Easier to manage and scale big projects due to modular structure.

1. Real-world modelling:

* Objects represent real-world enities, making problem-solving more natural

1. Easier maintenance:

* Changes in one part of the program usually don’t affect other parts.

1. Flexibility and extensibility:

* Easy to add new features without modifying much existing code.
* **Explain the steps involved in setting up a C++ development environment**.
* Answer:

Step 1: install a C++ compiler:

1. Install and add the compiler path to system environment variables

Step 2: choose and install a code editor or IDE:

1. Dev-C++:

* Simple and easy to use

Step 3: Configure the compiler in your IDE

1. In dev C++, compiler are usually pre-configured

* **What are the main input/output operations in C++? Provide examples**.
* Answer:

1. Output operation – cout:

* Used to display text or variables on the screen.
* Example:
* Cout << “hello world”;

1. Input operation – cin:

* Used to take input from the user.
* Example:
* Cin >> variable;

1. Multiple inputs:

* You can take multiple inputs using one cin.
* Example:
* Int a, b;

Cin >> a >> b;

Q.2 variables, Data types, and operator:

* **What are the different data types available in C++? Explain the example.**
* Answer:

1. Derived data type:
2. Primitive data type:
3. User define data type:
4. Primitive data type:

|  |  |  |
| --- | --- | --- |
| Data type | Description | Example |
| Int | Stores integer | Int a = 25; |
| Float | Store decimal number | Float a = 2.25; |
| Char | Stores character | Char c = ‘a’; |
| Double | Stores decimal numbers | Double price = 99.98; |

1. Derived data type:

|  |  |  |
| --- | --- | --- |
| Data type | Description | Example |
| Array | Collection of elements of the same data type | Int arr[5]={10,20,30,40,50}; |
| Pointer | Stores address of another variable | Int \*ptr = &age; |
| Function | Block of code that performs a task | Int sum(int a , int b)  {  a+b;  } |

1. User – define data type:

|  |  |  |
| --- | --- | --- |
| Data type | Description | Example |
| Struct | Group of related variables | Struct student {int age, char name[30]} |
| Union | Like struct, but shares memory | Union data { int I, float f}; |
| Class | Blueprint for creating objects | Class student{public :int id} |

* **Explain the difference between implicit and explicit type conversion in C++.**
* Answer :

1. Implicit type conversion :

* Automatically done by the compiler
* Happens when you mix data types in an expression
* Converts smaller data type to larger to avoid data loss

1. Explicit type conversion

* Manually done by the programmer
* You forcefully convert one data type to another
* Syntax:
* (type)variable;
* **What are the different types of operator in c++? Provide example of each.**
* Answer:

1. Arithmetic operator:

|  |  |  |
| --- | --- | --- |
| Operator | function | example |
| + | Addition | a + b |
| - | subtraction | a – b |
| \* | multiplication | a \* b |
| / | division | a / b |
| % | module | a % b |
| ++ | increment | a++ |
| -- | decrement | a-- |

* Example:

#include<iostream>

Using namespace std;

main ()

{

Int num1, num2,ans;

Float ans1;

cout<<"\n enter num1";

cin>>num1;

cout<<"\n enter num2";

cin>>num2;

ans = num1+num2;

cout<<"\n addition is"<<ans;

ans = num1 - num2;

cout<<"\n substraction is"<<ans;

ans = num1 \* num2;

cout<<"\n multiplication is"<<ans;

ans1 = (float)num1/num2;

cout<<"\n division is"<<ans1;

num1++;

cout<<"\n increment is"<<num1;

num2--;

cout<<"\n decrement is"<<num2;

}

1. Relational operator:

|  |  |  |
| --- | --- | --- |
| Operator | function | example |
| == | Equal to | a = b |
| > | Greater than | a >b |
| < | Less than | a < b |
| >= | Greater than equal to | a >= b |
| <= | Less than equal to | a <= b |
| != | Not equal to | a != b |

* Example:

#include<iostream>

Using namespace std;

Main ()

{

Int a=5, b=10,c;

c=a>b;

cout<<"\n grater is"<<c;

c=a<b;

cout<<"\n less than is"<<c;

c=a>=b;

cout<<"\n grater than equal to is"<<c;

c = a<=b;

cout<<"\n less than equal to is"<<c;

c = a != b;

cout<<"\n not equal to is"<<c;

c = a==b;

cout<<"\n equal to is"<<c;

}

1. Assignment operator:

|  |  |  |
| --- | --- | --- |
| Operator | function | example |
| += | Addition equal to | a += |
| -= | Subtraction equal to | a -= |
| \*= | Multiplication equal to | a \*= |
| /= | Division equal to | a /= |
| = | Equal to | a = b |
| %= | Module equal to | a %= b |
| != | Not equal to | a != b |

* Example:

#include<iostream>

Using namespace std;

main (){

int num=5;

num += 5;

cout<<"\n adition equal to"<<num;

num -= 5;

cout<<"\n substraction equal to"<<num;

num \*= 10;

cout<<"\n multiplication equal to"<<num;

num /= 10;

cout<<"\n division equal to"<<num;

num %= 3;

cout<<"\n module equal to"<<num;

}

1. Logical operator:

|  |  |  |
| --- | --- | --- |
| operator | function | example |
| && | And | a>b && b>a |
| || | Or | a<b || b<a |
| ! | Not | a ! b |

* Example:

#include<iostream>

Using namespace std;

main ()

{

int a,b;

cout<<"enter a";

cin>>a;

cout<<"enter b";

cin>>b;

if(a>b && b<a)

{

cout<<"a is grater";

}

else

{

cout<<"b is grater";

}

if(a>b || b<a)

{

cout<<"\n a is grater";

}

else

{

cout<<"\n b is grater";

}

if(a != b)

{

cout<<"\n a is not equal to b";

}

}

1. Bitwise operator:

|  |  |
| --- | --- |
| operator | function |
| & | And |
| ^ | exclusive |
| ~ | Complement |
| | | Or |
| << | Left shift |
| >> | Right shift |

1. Conditional operator:

|  |  |
| --- | --- |
| operator | function |
| sizeof | Return the size of variable |
| & | Return the address of variable |
| \* | Pointer to a variable |
| ?: | Conditional expression |

* **Explain the purpose and use of constants and literals in c++.**
* Answer:

1. Constant in c++

* A constant is a values that cannot be changed once it is defined during the execution of a program

🡪 Purpose of constant:

* To protect variables from being modified
* To improve code readability and maintainability
* To avoid using “magic numbers” directly in the code
* To ensure the value remains fixed throughout the program

1. Literals in c++

* A literal is a fixed value used directly in the code without any identifier

🡪 purpose of literals

* To present fixed value in code
* Used in assignments, expressions, conditions, etc

Q.3 Control flow statements

* **What are the conditional statements in c++? Explain the if-else and switch statement**
* Answer

Conditional statements in c++ are used to make decision in a program based on certain conditions. They allow the program to execute different block of code depending on whether a condition is true or false.

* Types of conditional statements in c++:

|  |  |
| --- | --- |
| Statement type | Description |
| If statement | Executes a block of code if the condition is true |
| If--else | Executes one block if the condition is true, another if else |
| Else if leader | Tests multiple conditions in sequence |
| Nested if | If statement inside another if statement |
| Switch statement | Selects one of many block of code to be executed |

1. If –else statement:

* In a if else statement , it executes one block if the condition is true, if condition is false it executed else statement
* Example :

#include<iostream>

Using namespace std;

Main ()

{

Int num1;

Cout<<”enter the number”:

Cin>>num1;

If(num1 % 2 == 0)

{

Cout<<”number is positive”;

}

Else

{

Cout<<”number is negative”;

}

}

1. Switch statement:

* A switch statement in c++ is used to select one block of code from many options, based on the value of a variable or expression.
* It is cleaner alternative to multiple if..else if conditions when checking a single variable against many value
* Example:

#include <iostream>

using namespace std;

main() {

int day = 3;

switch(day) {

case 1:

cout << "Monday";

break;

case 2:

cout << "Tuesday";

break;

case 3:

cout << "Wednesday";

break;

case 4:

cout << "Thursday";

break;

case 5:

cout << "Friday";

break;

default:

cout << "Weekend";

}

}

* What is the difference between for, while, and do-while loops in c++?
* Answer:

1. For loop:

* When the number of iterations is known
* Syntax:

For (initialization; condition; increment)

{

Block of code;

}

* Example:

for (int i = 1; i <= 5; i++)

{

cout << i << " ";

}

1. While loop:

* When the condition is checked before execution and loop may not run at all condition is false initially
* Syntax:

While(condition)

{

Block of code;

}

* Example:

int i = 1;

while (i <= 5)

{

cout << i << " ";

i++;

}

1. Do-while loop:

* When the loop must run at least once, even if the condition is false.
* Syntax:

do {

// code block

} while (condition);

* Example:

int i = 1;

do

{

cout << i << " ";

i++;

} while (i <= 5);

* **How are break and continue statements used in loops? Provide example**
* Answer :

1. Break statement

* Used to immediately exit a loop, regardless of the loop condition
* Example :

#include <iostream>

using namespace std;

main() {

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

{

if (i == 5)

break;

cout << i << " ";

}

}

1. Continue statement:

* Used to skip the current iteration and jump to the next loop cycle.
* Example:

#include <iostream>

using namespace std;

main()

{

for (int i = 1; i <= 5; i++)

{

if (i == 3)

continue;

cout << i << " ";

}

}

* **Explain nested control structures with an example.**
* Answer:
* Nested control structures occur when you place one control structure inside another. This allows you to write more complex decision making or iteration logic.
* Example:

#include <iostream>

using namespace std;

main() {

int age = 20;

char gender = 'M';

if (age >= 18)

{

if (gender == 'M') {

cout << "Eligible Male";

}

else {

cout << "Eligible Female";

}

}

else {

cout << "Not Eligible";

}

}

Q.4 functions and scope:

* **What is a function in c++? Explain the concept of function declaration, definition and calling**
* Answer:

Functions:

Functions is a set of statement than take the input, do some specific computation and produce the output

1. Function declaration:

* Tells the compiler about the function name, return type, and parameters

1. Function definition:

* Describe how the function performs its task.

1. Function call:

* Can be made from main() function

Example:

#include<stdio.h>

Void display (); // function declaration

Void display ()

{

Printf (“hello world”); // function definition

}

Main ()

{

display (); // function call

}

* **What is the scope of variable in c++? Differentiate between local and global scope.**
* Answer:
* Scope:
* Scope refers to the region or part of a program where a variable can be accessed or used. It determines the lifetime and visibility of the variable.
* Types of variable scope in c++:
* Local scope
* Global scope

1. Local variable:

* Declared inside a function or block
* Exists only during function execution
* Cannot be accessed from outside that function

1. Global variable:

* Declared outside all functions
* Accessible by all functions in the life
* Example:

#include<iostream>

using namespace std;

// global variable

int c;

void add(int a, int b)

{

// int a,b is local variable

c=a+b;

cout<<"\n addition is="<<c;

}

main()

{

add(5,6);

}

* **Explain recursion in c++ with an example.**
* Answer :
* Recursion:
* Recursion is a programming technique where a function calls itself in order to solve a problem.
* Example:
* #include<iostream>
* using namespace std;
* int fact(int n)
* {
* int fac;
* if(n==1)
* {
* return 1;
* }
* else
* {
* fac = n\*fact(n-1);
* return fac;
* }
* }
* main()
* {
* cout<<"factorial is"<<fact(6);
* }
* **What are function prototypes in c++? Why are they used?**
* Answer:
* A function prototype is a declaration of a function that tells the compiler
* The function’s name
* Its return type
* The number and types of parameters
* Why use function prototypes?
* Compiler awareness:

= informs the compiler about a function before it is used.

* Enables top-down compilation:

= allows calling a function before it’s defined.

* Type checking:

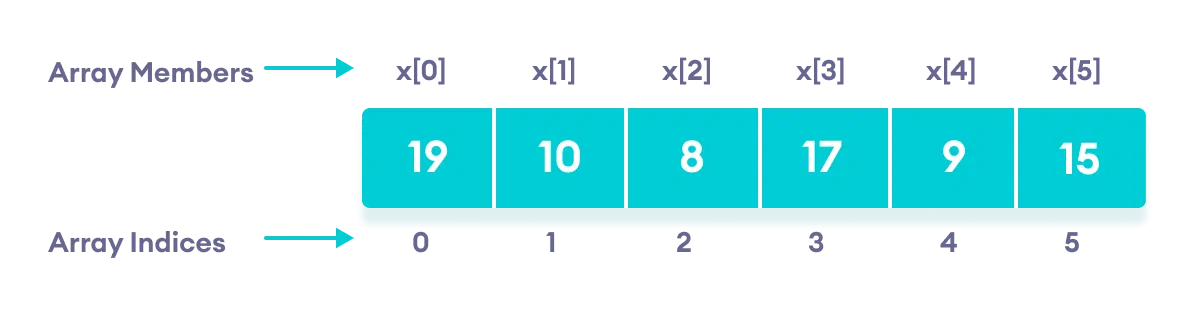
= ensures correct arguments and return types.

* Reduces errors

= catches mismatches in argument number early

Q.5 Arrays and strings:

* **What are the arrays in c++? Explain the difference between single dimensional and multi dimensional arrays.**
* Answer:
* Arrays are collection of elements of the same data type. An array is a data structure that stores a fixed size data type



* Difference between single dimensional and multi dimensional arrays.

|  |  |  |
| --- | --- | --- |
| Features | One dimensional array | Multi-dimensional array |
| Definition | An array with a single row or column | An array with more than one row and column |
| Structure | Linear | Matrix-like |
| Representation | Int arr[5] | Int arr[5][5] |
| Memory | Uses less memory | Uses more memory |

* **Explain string handling in c++ with examples.**
* Answer:
* What is a string?
* A string is a sequence of characters used to store and manipulate text.
* In c++, strings can be handled in two main ways:

🡪 Using C-style string

🡪 Using the string class from the standard template library

* Example:

#include<iostream>

#include<cstring>

using namespace std;

main()

{

char str[50],count=1;

int len,i;

cout<<"enter the string";

cin>>str;

len = strlen(str);

for(i=0; i<len/2; i++)

{

if(str[i] != str[len-1-i])

{

count= 0;

break;

}

}

if(count==1)

{

cout<<"\n string is palindrome";

}

else

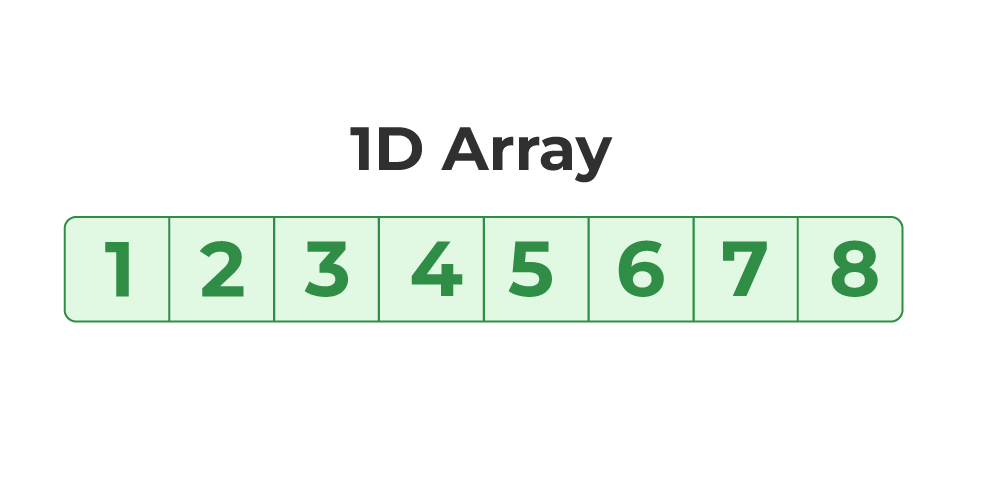
{

cout<<"\n string is not palindrome";

}

}

* **How are arrays initialized in c++? Provide examples of both 1D and 2D arrays.**
* Answer:
* Arrays:
* Arrays are collection of elements of the same data type. An array is a data structure that stores a fixed size data type
* 1D array [ one – dimensional array ]



* Example:

#include<iostream>

using namespace std;

main()

{

int arr[5];

int i,sum=0,ave;

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

{

cout<<"\n enter arr"<<i;

cin>>arr[i];

}

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

{

sum = sum + arr[i];

}

cout<<"\n sum is="<<sum;

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

{

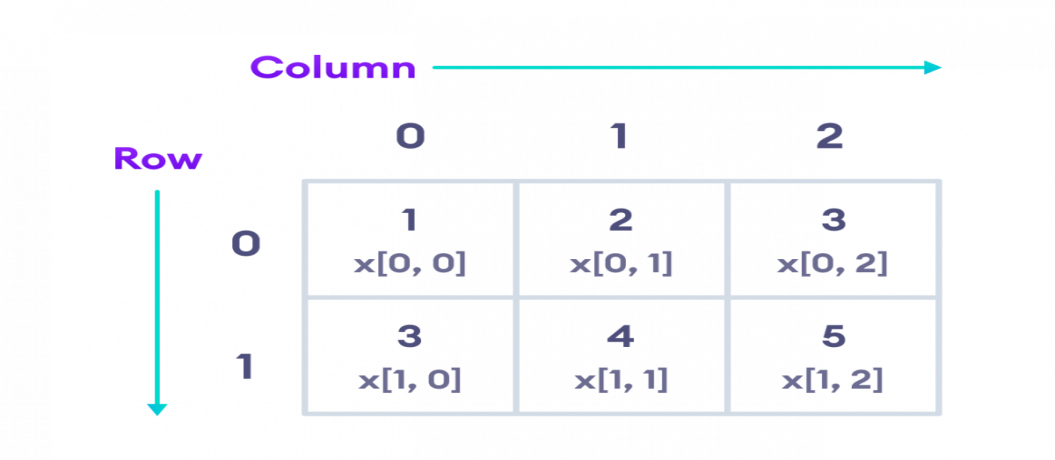
ave = sum/5;

}

cout<<"\n average is="<<ave;

}

* 2D array [ 2 dimensional array]:



* Example:

#include<iostream>

using namespace std;

main(){

int arr1[3][3], arr2[3][3],arr3[3][3];

int i,j;

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

{

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

{

cout<<"\n enter arr"<<i<<j;

cin>>arr1[i][j];

}

}

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

{

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

{

cout<<"\t arr1"<<arr1[i][j];

}

cout<<"\n";

}

cout<<"=====================================================================";

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

{

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

{

cout<<"\n enter arr"<<i<<j;

cin>>arr2[i][j];

}

}

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

{

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

{

cout<<"\t arr2"<<arr2[i][j];

}

cout<<"\n";

}

cout<<"====================================================================================";

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

{

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

{

arr3[i][j] = arr1[i][j]+arr2[i][j];

}

}

cout<<"===================================================================================";

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

{

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

{

cout<<"\t sum is="<<arr3[i][j];

}

cout<<"\n";

}

}

* **Explain string operations and functions in c++.**
* Answer:
* String operations:

1. String class operations

* #include <string>
* using namespace std;

1. c-style string functions [ from <cstring> ]

* #include <cstring>
* String functions:

1. Strlen():

* Strlen function is used to find the length of string

1. Strcpy ():

* Strcpy function is used to copy one string to another string

1. Strcmp () :

* Strcmp is used to compare two string

1. Strcat ():

* Strcat function is used to concat two strings. It is used to join two string

1. strchr():

* strchr is used to find the first occurrence of a character in a string

Q.6 introduction to object-oriented programming:

* **Explain the key concepts of object-oriented programming.**
* Answer:
* Object-oriented programming is a programming paradigm based on the concept of “objects”, which can contain data and functions. C++ is one of the most popular OOP languages.

1. Class:

* Class is combination of data member and member functions.

1. Object:

* Object is a instance of a class

1. Encapsulation:

* Wrapping data and functions into a single unit and hiding internal details.

1. Abstraction:

* Hiding unnecessary details and showing only essential features to the user.

1. Inheritance:

* Inheritance is a [is-a] relationship , it create a new class from base class

1. Polymorphism:

* The ability to use the same function or operator in different ways.
* **What are classes and objects in c++? Provide an example**.
* Answer:
* Classes:
* Class is a combination of data member and member function. A class in c++ is a user-defined data type that serves as a blueprint for creating objects.
* Object:
* An object is an instance of a class. When a class is defined, no memory is allocated until an object of that class is created

🡪 Example:

#include<iostream>

using namespace std;

class calc{

public:

int a,b,c;

void getvalue(){

cout<<"enter a and b";

cin>>a>>b;

}

void add()

{

cout<<"\n addition is"<<a+b;

}

void sub()

{

cout<<"\n substraction is"<<a-b;

}

void mult()

{

cout<<"\n multiplication is"<<a\*b;

}

void div()

{

cout<<"\n division is"<<a/b;

}

};

main()

{

calc c;

c.getvalue();

c.add();

c.sub();

c.mult();

c.div();

}

* **What is inheritance in c++? Explain with an example.**
* Answer:
* Inheritance:
* Inheritance is a [ is-a ] relationship , in a inheritance a new class is created from a base class . a new class is known as a derived class, child class and sub class

And a base class is known as super class and a parent class.

* There are five types of inheritance:

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance
4. Hierarchical inheritance
5. Hybrid inheritance
6. Single inheritance:

* The a new class derived from only one base class is known as a single inheritance
* [Animal] ← Base Class

|

| (Dog inherits from Animal)

↓

[Dog] ← Derived Class

* example:

#include<iostream>

using namespace std;

class category{

public:

int catid;

char catname[20];

void getcategory()

{

cout<<"enter catid";

cin>>catid;

cout<<"enter catname";

cin>>catname;

}

};

class product : public category{

public:

int pid;

char pname[50];

int price;

void getproduct(){

cout<<"\n enter pid";

cin>>pid;

cout<<"\n enter pname";

cin>>pname;

cout<<"\n enter price";

cin>>price;

}

void showproduct(){

cout<<"\n catid="<<catid<<"\t catname="<<catname;

cout<<"\n pid="<<pid<<"\t pname="<<pname<<"\t price="<<price;

}

};

main()

{

product p1;

p1.getcategory();

p1.getproduct();

p1.showproduct();

}

1. multiple inheritance:

* in a multiple inheritance a new class is derived from two or more base class
* [A] [B]
* \ /
* [C]
* Example:

#include<iostream>

using namespace std;

class category{

public:

int catid;

char catname[20];

void getcategory()

{

cout<<"enter catid catname";

cin>>catid>>catname;

}

};

class brand{

public:

int bid;

char bname[50];

void getbrand()

{

cout<<"\n enter bid bname";

cin>>bid>>bname;

}

};

class product : public category, public brand

{

public:

int pid;

char pname[50];

void getproduct(){

cout<<"\n enter pid pname";

cin>>pid>>pname;

}

void show()

{

cout<<"\n catid="<<catid<<"\t catname="<<catname;

cout<<"\n bid="<<bid<<"\t bname="<<bname;

cout<<"\n pid="<<pid<<"\t pname="<<pname;

}

};

main()

{

product p1;

p1.getcategory();

p1.getbrand();

p1.getproduct();

p1.show();

}

1. Multilevel inheritance:

* A class is derived from another derived class is known as a multilevel inheritance

[A]

↓

[B]

↓

[C]

* Example:

class A

{

public:

void a()

{

cout << "A\n";

}

};

class B : public A

{

public:

void b()

{

cout << "B\n";

}

};

class C : public B {

public:

void c()

{

cout << "C\n";

}

};

1. Hierarchical inheritance:

* Multiple derived class derive from a single base class is known as a hierarchical inheritance

[Parent]

/ | \

[Child1][Child2][Child3]

* Example:

#include<iostream>

using namespace std;

class A{

public:

int a;

void getA()

{

cout<<"enter the value of a";

cin>>a;

}

};

class B: public A

{

public:

int b;

void getB()

{

cout<<"enter the value of b";

cin>>b;

}

void add()

{

cout<<"\n addition is="<<a+b;

}

};

class C:public A{

public:

int c;

void getC()

{

cout<<"enter the value of c";

cin>>c;

}

void add()

{

cout<<"\n the addition is="<<a+c;

}

};

main()

{

C c1;

c1.getA();

c1.getC();

c1.add();

B b1;

b1.getA();

b1.getB();

b1.add();

}

1. Hybrid inheritance:

* Hybrid inheritance is a combination of multilevel and multiple inheritance

[A]

/ \

[B] [C]

\ /

[D]

- Example:

#include<iostream>

using namespace std;

class A{

public:

int a;

void getA()

{

cout<<"enter the a";

cin>>a;

}

};

class B:virtual public A{

public:

int b;

void getB()

{

cout<<"enter the b";

cin>>b;

}

};

class C:virtual public A{

public:

int c;

void getC(){

cout<<"\n enter the c";

cin>>c;

}

};

class D:public C, public B{

public:

int d;

void getD(){

cout<<"enter the value of d";

cin>>d;

}

void add()

{

cout<<"\n addition is"<<a+b+c+d;

}

};

main()

{

D d1;

d1.getA();

d1.getB();

d1.getC();

d1.getD();

d1.add();

}

* **What is encapsulation in c++? How is it achieved in classes?**
* Answer:
* What is encapsulation?
* Encapsulation is the binding of data and functions that operate on that data into a single unit and restricting access to some components for security and protection
* Think of it like a capsule: it contains both data and method that act on the data but hides the inner working from the outside.
* How it achieved
* Use the class keyword
* Mark data member as private
* Use public functions to get or set values
* Example:

#include<iostream>

using namespace std;

class bank{

public:

char username[50];

int accountno, balance;

public:

bank(){

cout<<"\n enter username";

cin>>username;

cout<<"\n enter accountno";

cin>>accountno;

cout<<"\n enter balance";

cin>>balance;

cout<<"\n username="<<username;

cout<<"\n accountno="<<accountno;

cout<<"\n balance="<<balance;

}

bank(char username[50], int accountno, int balance)

{

this->username[50]=username[50];

this->accountno=accountno;

this->balance=balance;

}

};

main()

{

bank b;

int choice;

float i;

cout<<"\n ===account details===";

cout<<"\n 1. your account is saving";

cout<<"\n 2. your account is current";

cout<<"\n enter you choice";

cin>>choice;

switch(choice)

{

case 1:

//i= 0.5+(b.balance/100);

i = b.balance+0.5;

cout<<"\n saving account balance is"<<i;

break;

case 2:

//i = 0.5-(b.balance/100);

i=b.balance-0.5;

cout<<"\n current account balance is"<<i;

break;

}

}