1.INHERITANCE

TYPES:

* SINLGE LEVEL INHERITANCE
* MULTI LEVEL INHERITANCE
* MULTIPLE INHERITANCE
* HIRERCHICAL INHERTANCE
* HYBRID INHERITANCE

MULTIPLE INHERITANCE

Multiple inheritance allows us to combine the features of Several existing classes as a starting point for defining new classes.

SYNTAX:

class D: visibility B-1, visibility B-2 …. {

..... (Body of D)

};

Where visibility can be either private or public. If in case we define same name for a function with different action in base class as well as derived class, then definition in derived class overrides the function in base class and if we make a fun call() by derived class object it automatically calls the function in derived class.

To call the function in the base class in such cases scope resolution operator can be used.

Ex:

P p;

p.dispaly(); //derived class function

p.A::display(); //function of base class A

P.B::display(); //function of base class B

This may can also occur in case of single inheritance.

HYBRID INHERITANCE

It is the combination of multiple and multilevel inheritance.

VIRTUAL BASE CLASSSES:

When a child is inherited from parent1 and parent 2 which are derived classes of grandfather, the child can have duplicate copies members of grandfather. to avoid such ambiguity concepts of virtual classes are introduced.

The duplication of inherited members due to these multiple paths can be avoided by making the common base class (ancestor class) as virtual bo8'! class while declaring the direct or intermediate base classes.

Syntax: class b1 : virtual public A

ABSTRACT CLASSES:

An abstract class is one that is not used to create objects. An abstract class is designed only to act as a base class (to be inherited by other classes). It is a design concept in program development and provides a base upon which other classes may be built.

Constructors in derived class:

Syntax:

constructor (arglist) : intialization-section

{

assi9nment-section

}

Example:

D(int al, int a2, float bl, float b2, int dl):

A(al, a2},/\* call to constructor A \*/

B(bl, b2) /\* call to constructor B \*/

{

d =d1;

} ; // executes its body

}

Note:

The constructors for virtual base classes are invoked before any non-virtual base classes. If there are multiple virtual base classes, they are invoked in the order in which they are declared. Any non-virtual bases are then constructed before the derived class constructor is executed.

2.Polymorphism

It merely means that same name different action

Types:

1.Compile time polymorphism

The overloaded member functions are 'selected' for invoking by matching arguments, both type and number. This information is known to the compiler at the compile time and, therefore, compiler is able to Select the appropriate function for a particular call at the compile time itself. This is called early binding or static binding or static linking. Also known as compile time polymorphism, early binding simply means that an object is bound to its function call at compile time.

Ex:

Operator overloading and function overloading

2.Run time polymorphism

when it is known what class objects are under consideration, the appropriate version of the function is invoked. Since the function is linked with a particular class much later aft.er the compilation. this process is called late binding. It is also known as dynamic binding. because the selection of the appropriate function is done dynamically at run time. It makes use of pointers.

Pointers:

Declaration syntax:

Datatype \* pointer variable;

* Void pointers are called generic pointers
* Pointer which is not initialized are called null pointer.
* If a pointer of any datatype is assigned with “0”is called null address.
* A dereferencing operator “\*” can be used to access and modify the content stored in the pointer. caution: before accessing its value must be assigned.
* The point.er to function is known as callback function. We can use these function pointers to refer to a function
* Using function pointers, we can allow a C++ program to select a function dynamically at run time
* We can also pass a function as an argument to another function. Here, the function is passed as a pointer.
* The function pointers cannot be dereferenced. C++ also allows us to compare two function pointers.
* C++ provides two types of function pointers; function pointers that point to static member functions and function Pointers that point to non-static member functions. these two function pointers are incompatible with each other. The function pointers that point to the non-static member function requires hidden argument.

Declaration syntax:

Datatype \*(function name)();

Remember that declaring a pointer only creates a pointer. It does not create actual function. For this, we must define the task, which is to be performed by the function. The function must have the same return type and arguments. page 278.

Pointers to object:

An object can also have a pointer by initialising the pointer with its address . that pointer can access the member data using arrow operator.

Ex;

Item \*ptr;

Item x;//object

Ptr=&x;

Ptr->get\_data; //or

(\*ptr).get\_data;//also valid since \*ptr=x

We can also create the objects using pointers and new operator as follows:

Item \*ptr=new item;

This statement allocates enough memory for the data members in the object structure and assigns the address of the memory space to ptr.

This pointer:

C++ uses a unique keyword called this to represent an object that invokes a member function. this is a pointer that points to the object for which this function was called. For example, the function call A.max() will set the pointer this to the address of the object A. The starting address is the same as the address of the first variable in the class stn1cture.

This unique pointer is automatically passed to a member function when it is called. The Pointer this acts aa an implicit argument to all the member functions.

Pointers to derived classes:

We can Use pointer not only to the base objed4 but also to the objects of derived classes.

Virtual functions:

when we use the same function name in both the base and derived classes, the function in base class is declared as virtual using the keyword virtual preceding its normal declaration. When a function is made virtual, C++ determines which function to use at run time based on the type of object pointed to by the base pointer. rather than the type of the pointer. Thus. By making the base pointer to point. to different objects. we can execute different version of the virtual function.

Note: run time polymorphism is achieved only when a virtual function is achieved through a pointer to the base class.

Pure virtual functions:

* A pure virtual function is a function declared in a base class that bas no definition relative to the base class.
* In such cases, the compiler requires each derived class to either define the function or redeclare it as a pure virtual function.
* Remember that a class containing pure virtual functions cannot be used to declare any objects or its own.
* As stated earlier, such classes are called abstract base classes.
* The main objective of an abstract base class is to provide some traits to the derived classes and to create a base Pointer required for achieving run time polymorphism