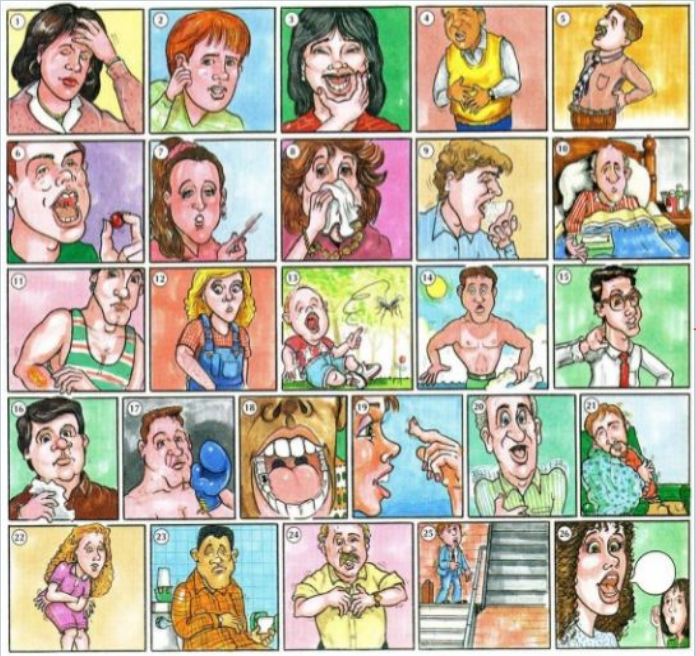
Menu Driven Dispensary

NAME: K LALITH ADITYA
REG NO: 194209
CLASS :BSC COMPUTER SCIENCE(HONS)
2ND YEAR(3RD SEM)

Problems ,Symptoms and Ailments

Please Enter your Problems



Problems ,Symptoms

Remedy with medication

Aum Sri Sairam

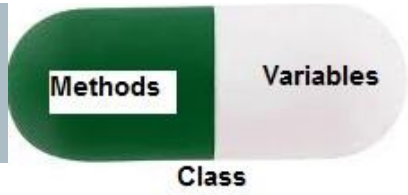
Concepts Template

Concept	Y/N	Context of usage	lines for each concept respectively
Encapsulation	y	for protecting member variables , member functions	836,876,900,591,496,92
Data Abstraction	y	creating objects of classes and using objects to call variables,functions	1064-1102
Static variables and Static functions	y	updating Covid 19 recovery rate	836,872
Objects as Parameters	n		
Inline Functions	y	function when defined ouside preceded by word inline	179
Friend Functions	y	to access protected , private data of a class	861,940
Empty Classes	y	to use its size to perform certain operations	89,96
Constructors(Default)	y	when object of a class is created , cons is invoked	863
Constructor(Parameterized)	y	initializing variables of a class	866,1334,1382,1383,1348,1363
Constructor(Global)	y	when global object of a class is created , cons is invoked	927
Constructor(Function)	n		
Constructor(Copy)	n		
Dynamic memory Allocation	y	pointer of a class by DMA, using pointer to access class function	945,970,983,1009
Memory leaks	n		
Dangling Pointers	n		

Concept	Y/N	Context of usage	lines for each concept respectively
Single Inheritance	y	inheriting from 1 parent	835(child),351(parent)
Multilevel Inheritance	y	inheriting from 1 grand-parent, new features from parent	351(grandparent),835(parent),serious(c)
Multiple Inheritance	y	inheriting from 2 parents	1329(c), 495(p),590(p)
Hierarchical Inheritance	y	2 child classes inheriting from 1 same parent 1..91(p), 495(c),590(c)	2.1392(p),1409,1402,1416= c
Hybrid Inheritance	y	many different types of inheritances are inter-linked	91,899,835,351,495,590
Constructor usage in Inheritance	y	in inheritance , it invokes parent constructor, if child object is created	593,499,94
Diamond Problem	y	use of virtual keyword to avoid compiler error .	1329(c),495,590=p
Function overloading and Overriding	n		
Composition	y	Linkage of objects of diiferent classes, 1 object out of scope, other object destructor is called	886,903
Aggregation	y	Linkage of objects , 1 object out of scope, other object destructor is called only at delete ptr	902,918,1012
Operator Overloading			
Unary	y	increment count for physically disabled	1328-1351
Binary	y	total count for eye problem and ear problem	1355-1388
Special Operators	n		

Concept	Y/N	Context of usage lines for each concept respectively
Abstract Classes		
Virtual Functions	y	use of same function in derived class with parent class pointer, in parent as virtual function 1148,1224,599,504,99
Pure Virtual Function	y	print values of data,in various formats deca,octa 1390,1286
Exception	y	if user input is not desired input , Exception handling is used 1.)23,66 2.)1020
Default	n	
Special	n	
Template	n	
Function Template	n	
Class Template	n	
Namespaces-User defined	n	
File Streams	n	
File Operation	n	
IN and OUT	n	
READ and WRITE	n	
EOF file	n	

Encapsulation



- In normal terms **Encapsulation** is defined as wrapping up of data and information under a single unit Encapsulation also lead to data abstraction or hiding. As using encapsulation hides the data.

- static float recovery rate; in class Covid_19 is in protected type .
- In Serious class , float temperature is in private.
- In class A , float a is kept in private. i.e when inheriting from base class , derived class can only access protected data, not private.

Data Abstraction

- Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

- Usage:-In function Problems_Ailment(); creating Display_diseases object ; objects of classes and accessing member functions of that class. Ex.
- object.Display_main(); //it displays all types of diseases.
- In Above, object is calling function name Display_main of class Display_diseases.

Static variables

- **When a variable is declared as static, space for it gets allocated for the lifetime of the program.**

- Ex. static int count; //declaration of static variable
- Usage:- static float recovery_rate; is declared in class Covid_19 in protected.
- If a static variable is declared it should be initialised .So
- float Covid_19::recovery_rate=65.3; Its used to update Indias Covid recovery rate if called.

Inline functions

- Inline function is a function that is expanded in line when it is called. Inline function may increase efficiency if it is small.

- inline void Display_diseases::Display_main(); is defined outside. Function is Display_main() of class Display_diseases. Its defined outside keyword is inline. Speed of
- inline void A:a(){Defn;}>void A:a(){Defn;}

Friend Functions

- A friend function can be given special grant to access private and protected members.

- In class Covid -19 , friend void Problems_Ailment(); is declared to access float recovery_rate; i.e not in public.
- And in function Covid -19 x ; //creating object
- x.recovery_rate; will access data..

Empty Classes

- class empty{}; is declared , but its size is 1 when we do sizeof(empty) ,

- class Display_diseases ; is a class, int Disp_dis_var; is private . Display_diseases is a superior class and is parent classes for many class. So to initialize value of Disp_dis_var. Size of class empty{}. Is used
- i.e Disp_dis_var= sizeof(empty) -1;//=0(First Class).

Constructors(Default)

- A **default constructor** is a **constructor** that either has no parameters, or if it has parameters, all the parameters have **default** values.
- Its used to initialize data.
- Covid_19(){ //default constructor
- cout<<"Covid_19 !!"<<endl<<endl;
- }
- Is used when object of class Covid_19 is created default constructor is Invoked.

Constructors(Parameterized)

- Covid_19(float value){ //parameterised constructor
- recovery_rate=value;
- cout<<"Covid_19!! Indias recovery rate: "<<value<<endl<<endl;
- }
- when object of class Covid_19 is created along with parameter.
- Usage:- Covid_19 object(94.3); //passing parameter initializing reocvery_rate.

Constructors(Global)

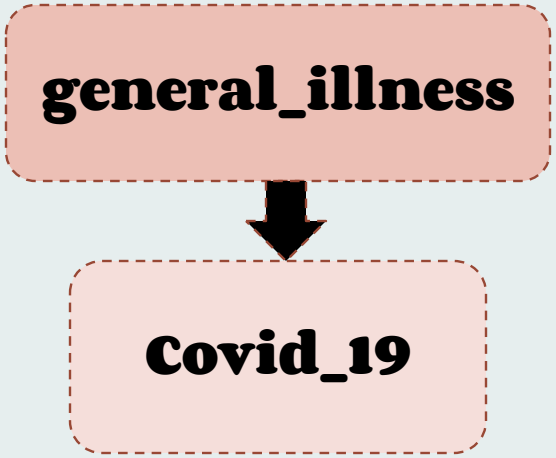
- **Global objects:** For a global object, constructor is called before main() is called. i.e if I create a **Global object of a class, and a local object. Global object constructor is invoked first and of top priority.**
- class Serious{
- //Definition of class(includes constructor)
- } cons_global(98.6);
- So. float temperature is initialised to 98.6.

Dynamic Memory Allocation

- Dynamically allocated memory is allocated on **Heap** and non-static and local variables get memory allocated on **Stack**.

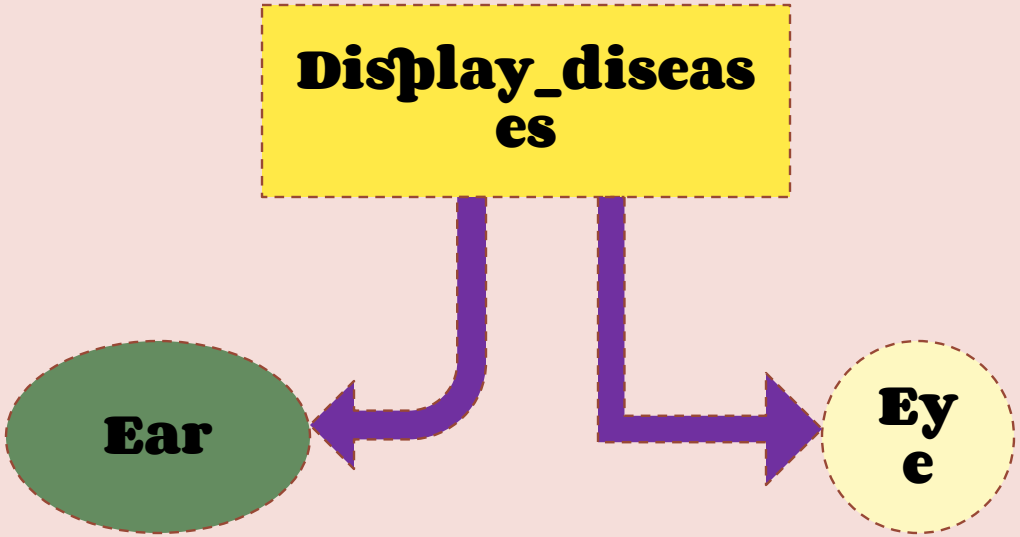
- Display_diseases* Fptr =new Display_diseases();
- Fptr is pointer , memory is allocated.
- Fptr->Ear();
- Fptr->Eye();,Dynamically accesses Member functions of class Display_diseases.

Single Inheritance



- class general_illness{//Definition1};
- class Covid_19:private general_illness{Definition2};

Hierarchical Inheritance

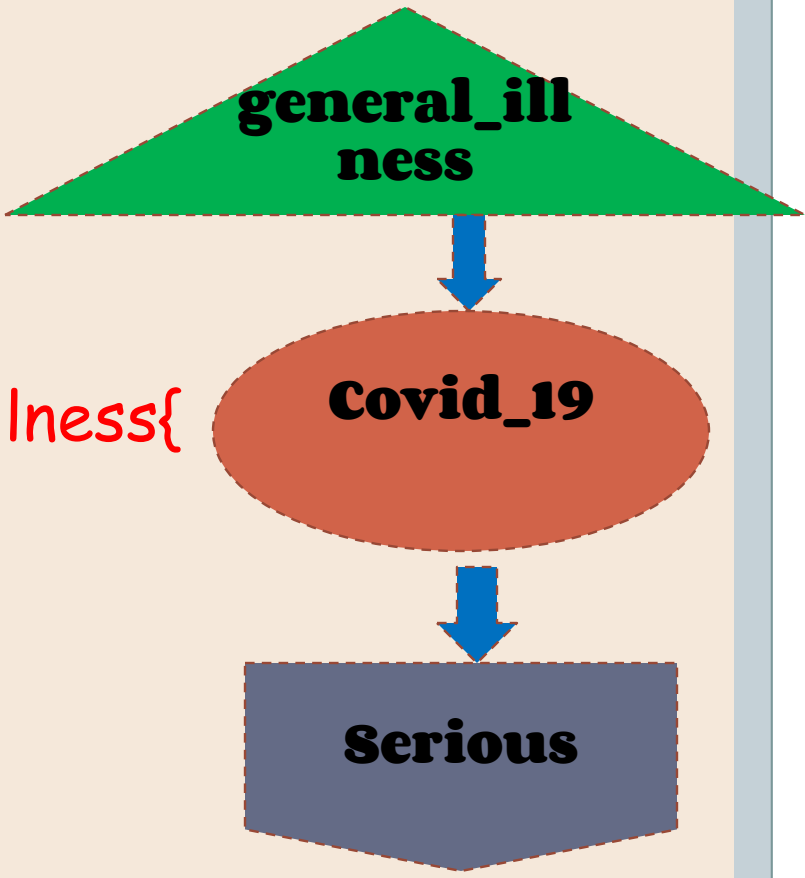


- Class Display_diseases{defn1};
- Class Ear:public Display_diseases{defn2};
- Class Eye:public Display_diseases{defn3};

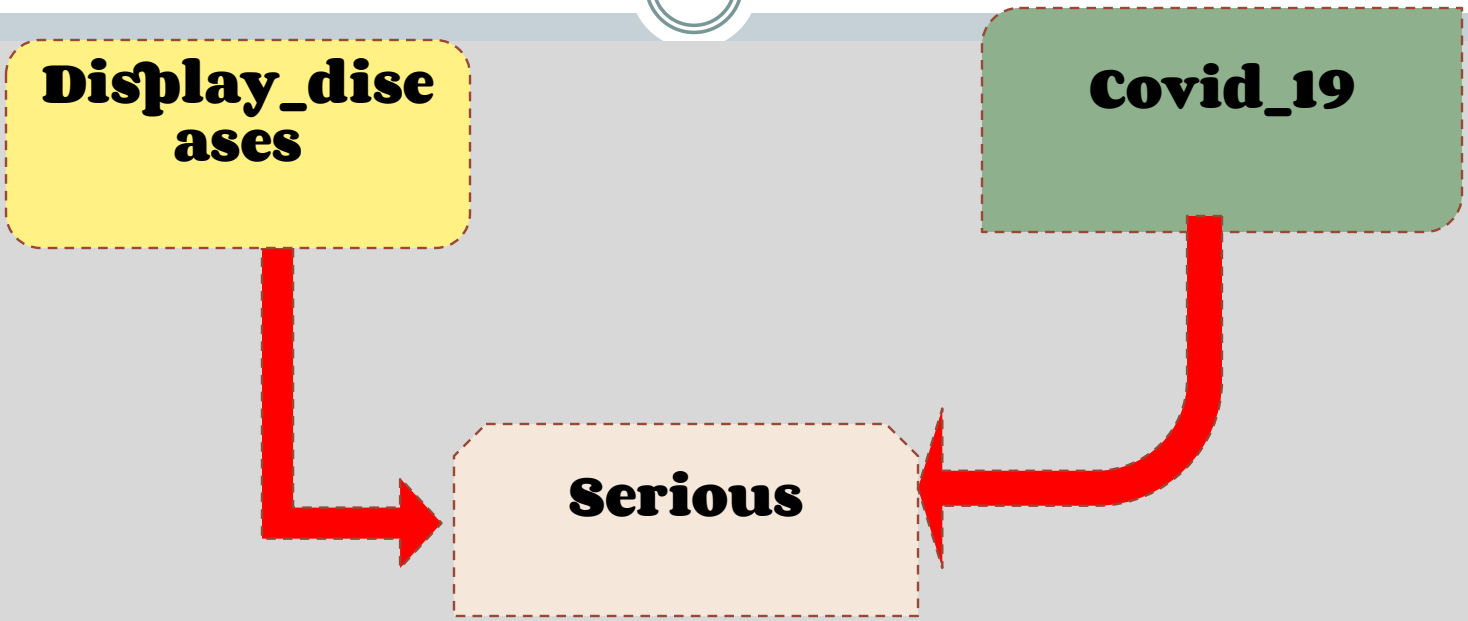
Multilevel Inheritance

- Class general_illness{
- Defn1
- };

- Class Covid_19:public general_illness{
- Defn 2
- };
- Class Serious:public Covid_19{
- Defn 3
- };

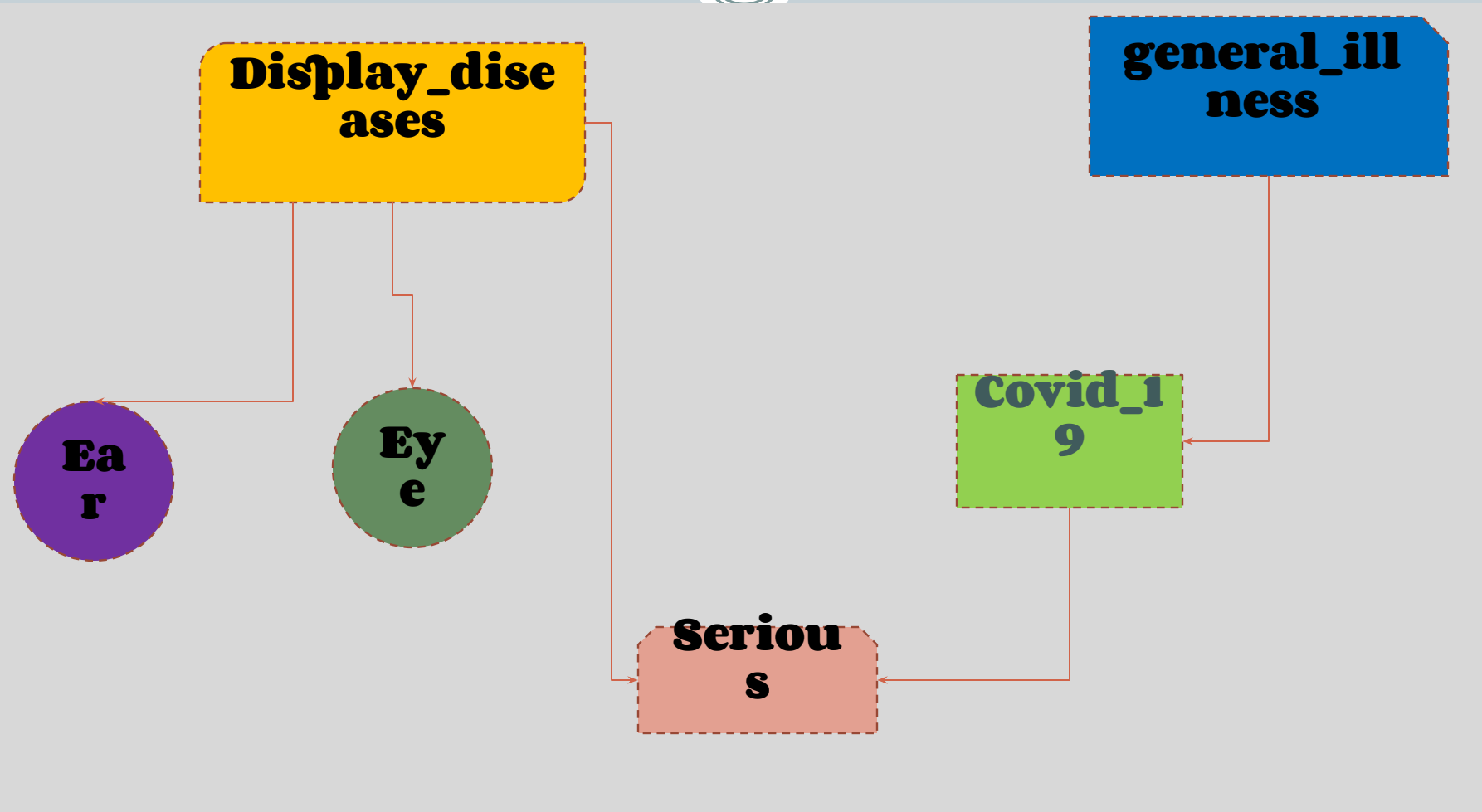


Multiple Inheritance

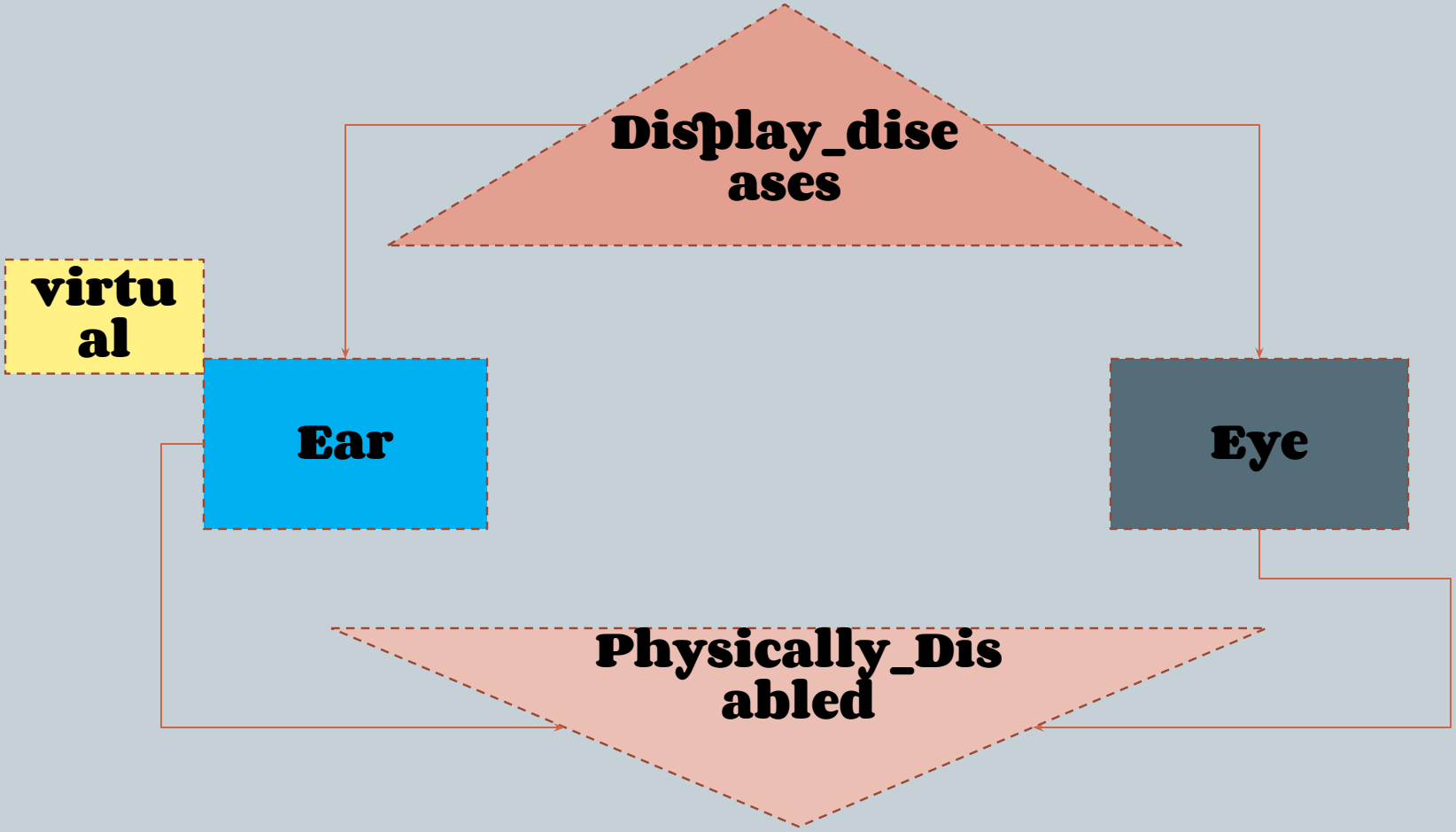


- Class Display_diseases{defn};
- Class Covid_19{defn1};
- Class Serious:public Display_diseases, public Covid_19{defn3};

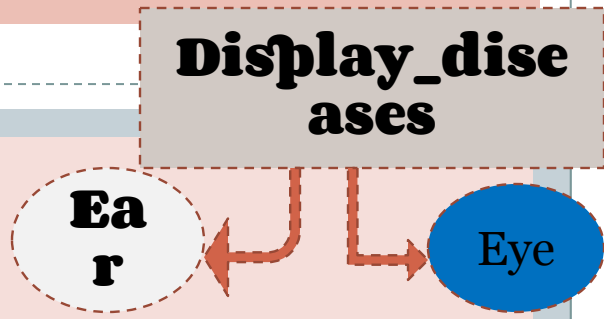
Hybrid Inheritance



Diamond Problem



Constructor Usage In Inheritance



- In case of hierarchical Inheritance ::--
- When object of Ear or Eye is declared
- along with constructor of Ear or Eye, constructor of its parent class is invoked i.e constructor of Display_diseases.
- In Display_diseases constructor will be
- Display_diseases(){
- cout<<"Cons:This is Display_diseases class , parent class of Eye , Ear"<<endl<<endl;
- Disp_dis_var=sizeof(empty)-1; //i.e 1-1 =0
- }

- **Similarly in class Ear**
- Ear(){
- cout<<"Cons:This is Ear class , derived class of Display_diseases"<<endl<<endl;
- Ear_var=2;
- }
- int main(){
- Ear object; //invokes child , parent constructor
- }

Composition

- Object composition is used for objects that have a **"has-a"** relationship with each other.
- class B{
- public:
- B(){
- cout<<"Constructing class B object"<<endl<<endl;
- }
- ~B(){
- cout<<"Destructing class B object"<<endl<<endl;
- }
- };

- class Serious{
- B composition;
- public:
- Serious(){ //constructor
- {cout<<"Constructing Serious object"<<endl0};
- ~Serious()
- {cout<<"Destructing class Serious object<<endl;}
- };
- If object of Serious becomes out of scope, B class destructor is called. It means if Serious object is lost, B object is also lost.

Aggregation

- Aggregation is a type of association that is used to represent the "HAS-A" relationship between two objects. But the lifetime of a part class does not depend on the lifetime of the whole class neither whole class can exist without an object of part class.
- class A{
- float a;
- public:
- A(float m){ //parameterized constructor
- this->a=m;}
- ~A(){cout<<"Destructing class A object"<<endl;}
- };

- class Serious{
- Public: A* aptr;
- void Aggregation(){
- aptr = new A(5);}
- };
- int main(){
- Serious S;
- S.Aggregation();
- }
- Even when object of class Serious is destroyed , object of A i.e aptr is not . Destructor For A is not invoked until programmer does:
- delete aptr;

Unary Operator Overloading

- In unary operator function, no arguments should be passed. It works only with one class objects. It is a overloading of an operator operating on a single operand.

```
class Physically_Disabled{
protected:
int val;
public:
Physically_Disabled(int m){
this->val=m;
}
void operator+(){
val++;
cout<<"count after increment is:"<<val<<endl;}
};
```

Ex: Increment Victim count

- **Function for unary operator overloading**
- void func_unary(){
- int number;
- cout<<"Enter Physically_Disabled count!"<<endl;
- cin>>number;
- Physically_Disabled victims(number);
- +victims;
- return;
- }

Binary Operator Overloading

- **In binary operator overloading function, there should be one argument to be passed. It is overloading of an operator operating on two operands.**
- class Eye_Ear_victims{
- public:
- int val;
- Eye_Ear_victims(){ //default constructor
- this->val=0;}
- Eye_Ear_victims(int m){ //parameterised constructor
- this->val=m;}
- Eye_Ear_victims operator+(Eye_Ear_victims &O2){

- Eye_Ear_victims Tot;
- Tot.val=this->val+O2.val;
- return Tot;}
- };

Ex: Adding total for 2 problems

```
void func_binary(){
  int number1,number2;
  cout<<"Enter Eye_victim count!"<<endl;
  cin>>number1;
  cout<<"Enter Ear_victim count!"<<endl;
  cin>>number2;

  Eye_Ear_victims O1(number1);
  Eye_Ear_victims O2(number2);
  Eye_Ear_victims Tot;
  Tot=O1+O2; //binary operator overloading
  cout<<"Total Eye_Ear_victims count is:"<<Tot.val<<endl<<endl;
  return;
}
```

Virtual Functions

- A virtual function is a member function which is declared within a base class and is re-defined(Over riden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

- Display_diseases *ptr1, base;
- ptr1 =&base; //address of object of base class
- ptr1->display_priority(); //function of base class
- ptr1 =&object2; //address of object of derived class
- ptr1->display_priority(); //function of derived class

Pure Virtual Functions

- class number{
- definition
- virtual void display()=0; //pure virtual function only declared
- };

- class dectype: public number{
- public:
- void display() //defined
- {cout<<"Last Year: count(in decimal type) is:"<< val <<endl;}
- };

Exception Handling

```
void try_catch_block(int x){
  try{
      if(x==1||x==5)
          return;
      else
          throw x;
  }
  catch(...){
  cout<<"Exception caught!! Enter above numbers only"<<endl;
      cout<<endl;
      main();
  }
}
```