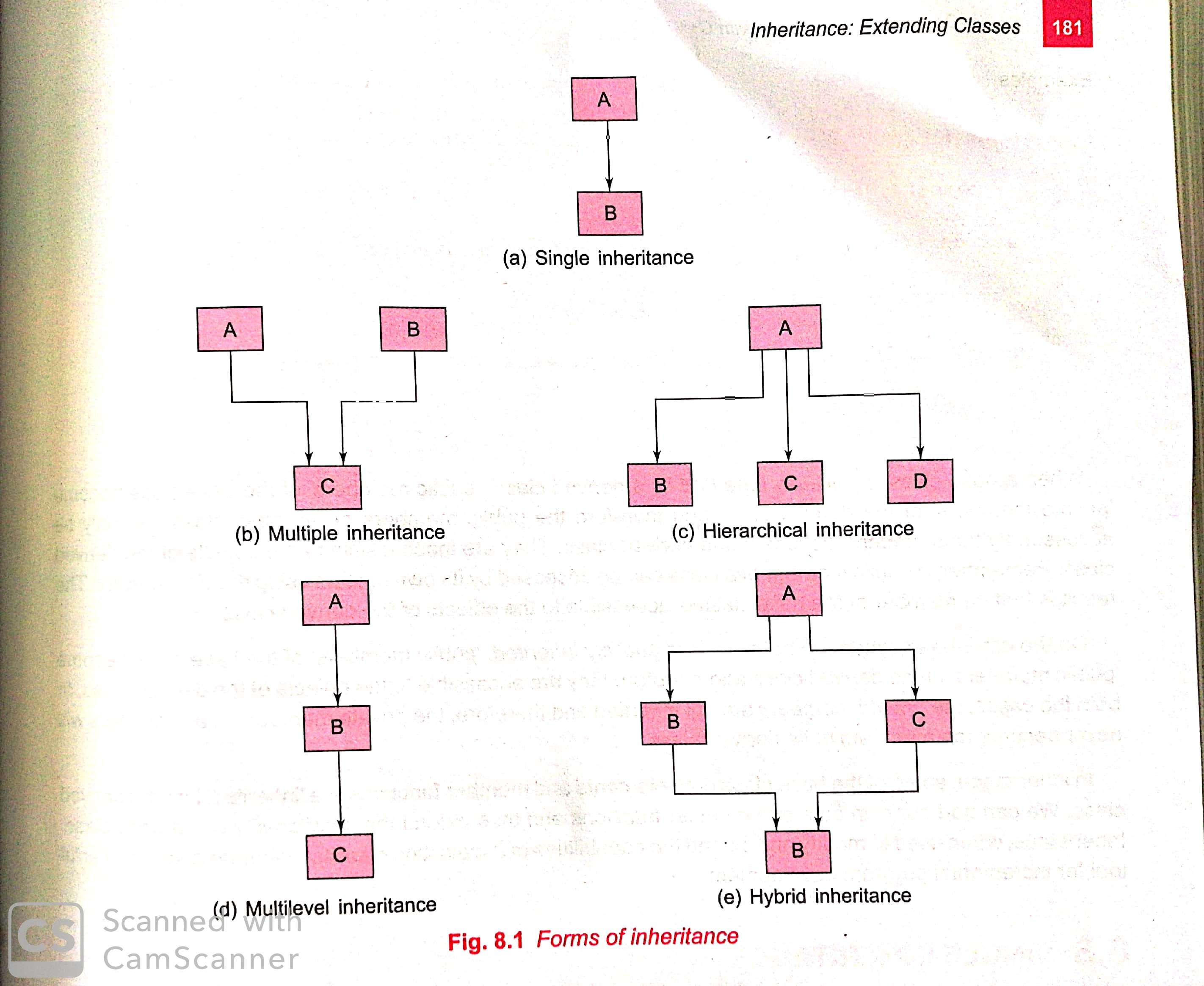
PROGRAMMING IN C++

UNIT -3

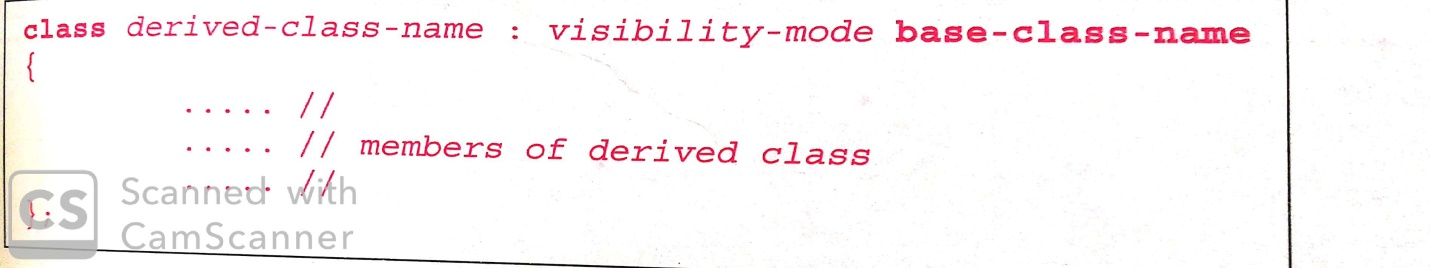
**INHERITANCE : EXTENDING CLASS**

* C++ strongly supports the concept of reusability.
* The C++ classes can be reused in several ways.
* The mechanism of deriving a new class from an old one is called inheritance.
* The old class is referred to as the base class and the new one is called the derived class or subclass.

****

**DEFINING DERIVED CLASSES**

* A derived class can be defined by specifying its relationship with base class in addition to its own details.
* The general form of defining a derived class is:



* The colon indicates that the derived class name is derived from the base class name.
* The visibility-mode is optional and if present, may be either private or public.
* The default visibility-mode is private.
* Visibility mode specifies whether the features of the base class are privately derived or publicly derived.

## C++ Single Inheritance

* **Single inheritance** is defined as the inheritance in which a derived class is inherited from the only one base class.

C++ Inheritance

Where 'A' is the base class, and 'B' is the derived class.

EXAMPLE PROGRAM

#include <iostream.h>

**class** Account

{

**public**:

**float** salary = 60000;

 };

**class** Programmer: **public** Account

 {

**public**:

**float** bonus = 5000;

   };

**int** main()

{

     Programmer p1;

     cout<<"Salary: "<<p1.salary<<endl;

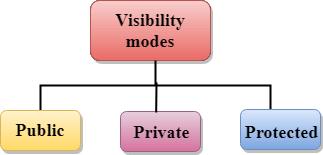
     cout<<"Bonus: "<<p1.bonus<<endl;

**return** 0;

}

## How to make a Private Member Inheritable

* The private member is not inheritable.
* If we modify the visibility mode by making it public, but this takes away the advantage of data hiding.
* C++ introduces a third visibility modifier, i.e., **protected**.
* The member who is declared as protected will be accessible to all the member functions within the class as well as the class immediately derived from it.
* **Visibility modes can be classified into three categories:**



* **Public**: When the member is declared as public, it is accessible to all the functions of the program.
* **Private**: When the member is declared as private, it is accessible within the class only.
* **Protected**: When the member is declared as protected, it is accessible within its own class as well as the class immediately derived from it.

## Multilevel Inheritance

C++ Inheritance

* When one class inherits another class which is further inherited by another class, it is known as multi level inheritance in C++.
* Inheritance is transitive so the last derived class acquires all the members of all its base classes.

EXAMPLE PROGRAM

#include <iostream.h>

class base

{

public:

int x;

void getdata()

{

cout << "Enter value of x= "; cin >> x;

}

};

class derive1 : public base

{

public:

int y;

void readdata()

{

cout << "\nEnter value of y= "; cin >> y;

}

};

class derive2 : public derive1

{

private:

int z;

public:

void indata()

{

cout << "\nEnter value of z= ";

cin >> z;

}

void product()

{

cout << "\nProduct= " << x \* y \* z;

}

};

int main()

{

derive2 a; //object of derived class

a.getdata();

a.readdata();

a.indata();

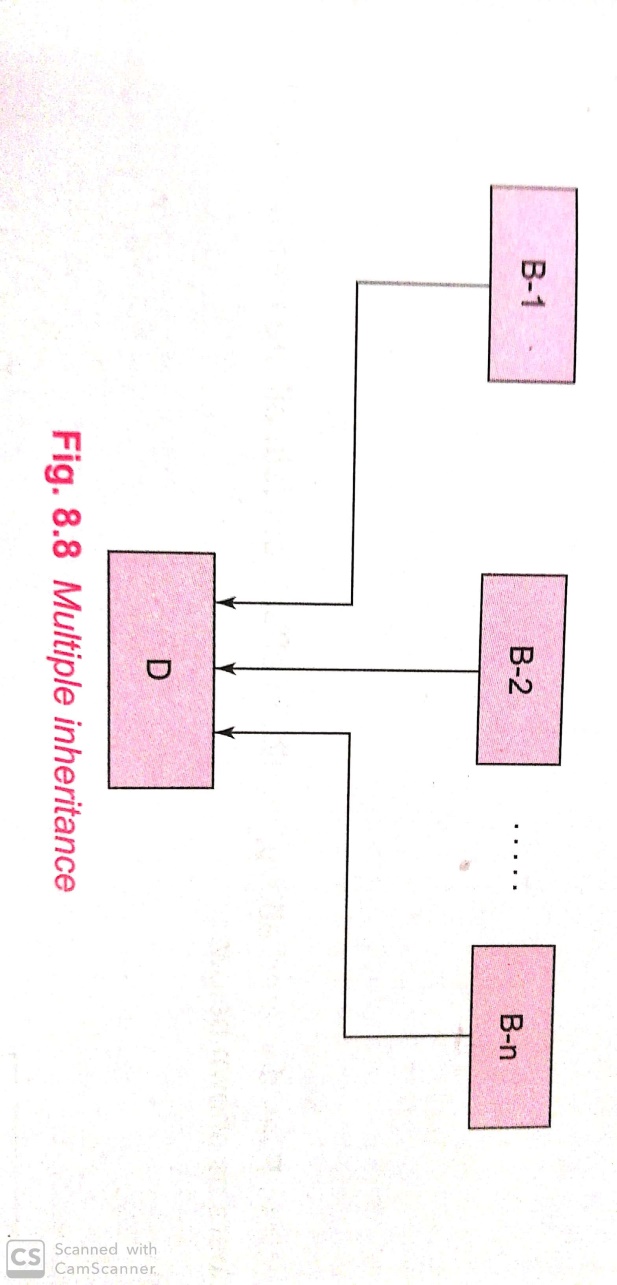
a.product();

return 0;

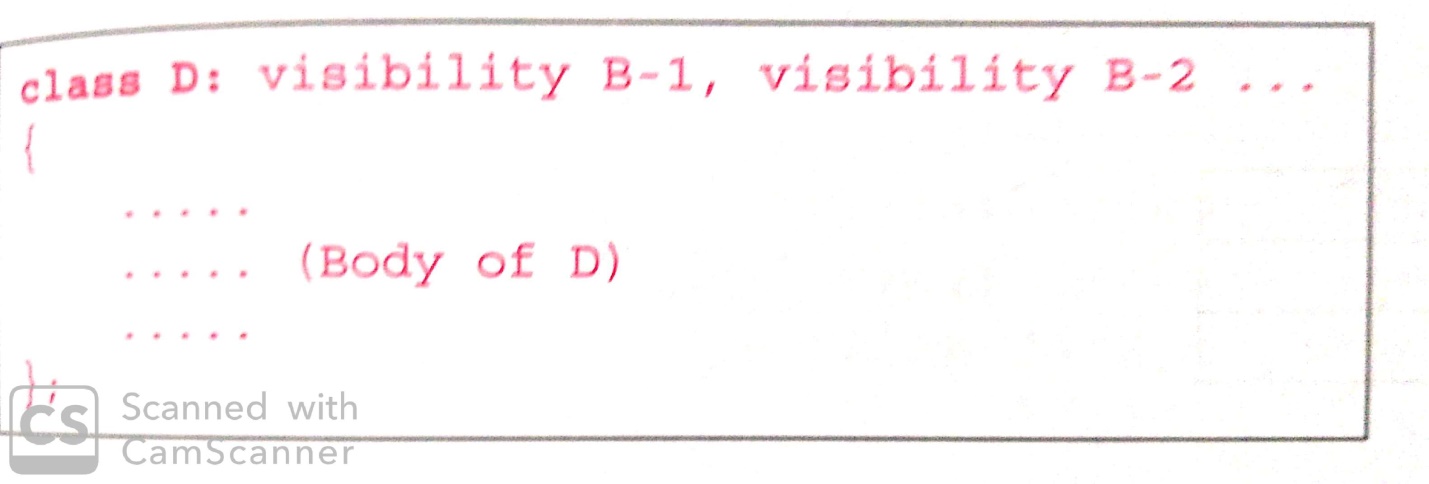
}

MULTIPLE INHERITANCE

* A class can inherit the attributes of two or more classes.
* This is known as multiple inheritance.
* Multiple inheritance allows us to combine the features of several existing classes as a starting point for defining new classes.



* The syntax of a derived class with multiple base classes is as follows.



Where,

* Visibility may be either public or private.
* The base classes are separated by commas.

Example Program

#include<iostream.h>

class M

{

protected:

int m;

public:

void get\_m(int);

};

class N

{

protected:

int n;

public:

void get\_n(int);

};

class P: public M, public N

{

public:

void display();

}

void M::get\_m(int x)

{

m=x;

}

void N::get\_m(int y)

{

n=y;

}

void P::display()

{

cout<<”m= “<<m<<”\n”;

cout<<”n= “<<n<<”\n”;

cout<<”m\*n = “ << m\*n <<”\n”;

}

void main()

{

P p;

p.get\_m(10);

p.get\_n(20);

p.display();

}

Ambiguity resolution in inheritance

* When a function with the same appears in more than one base class.

**Example**

class M

{

public:

void display()

{

cout<<”class m\n”;

}

};

class N

{

public:

void display()

{

cout<<”class n \n”;

}

};

class P: public M,public N

{

public:

void display()

{

M::display();

N::display();

}

};

int main()

{

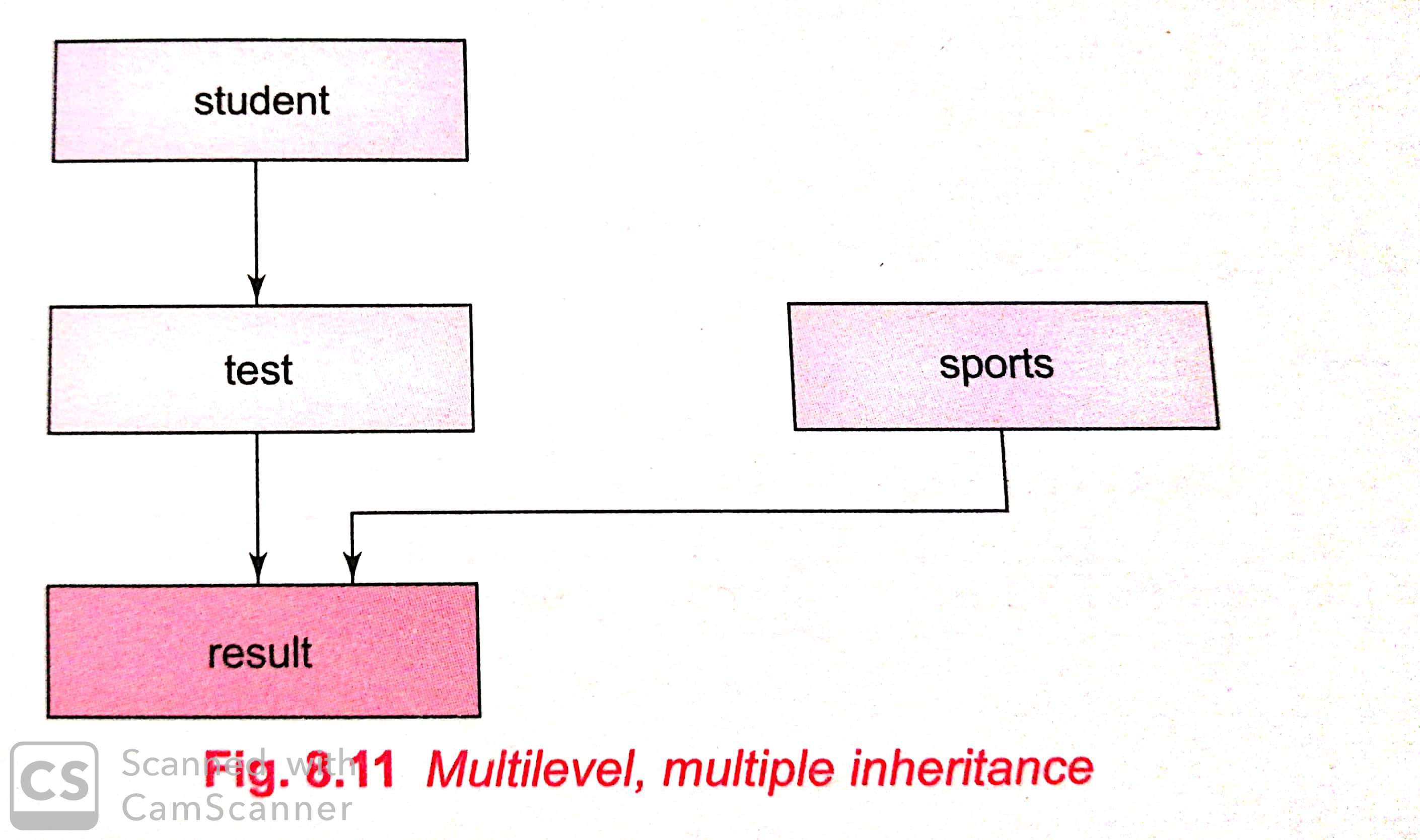
P p;

p.display();

}

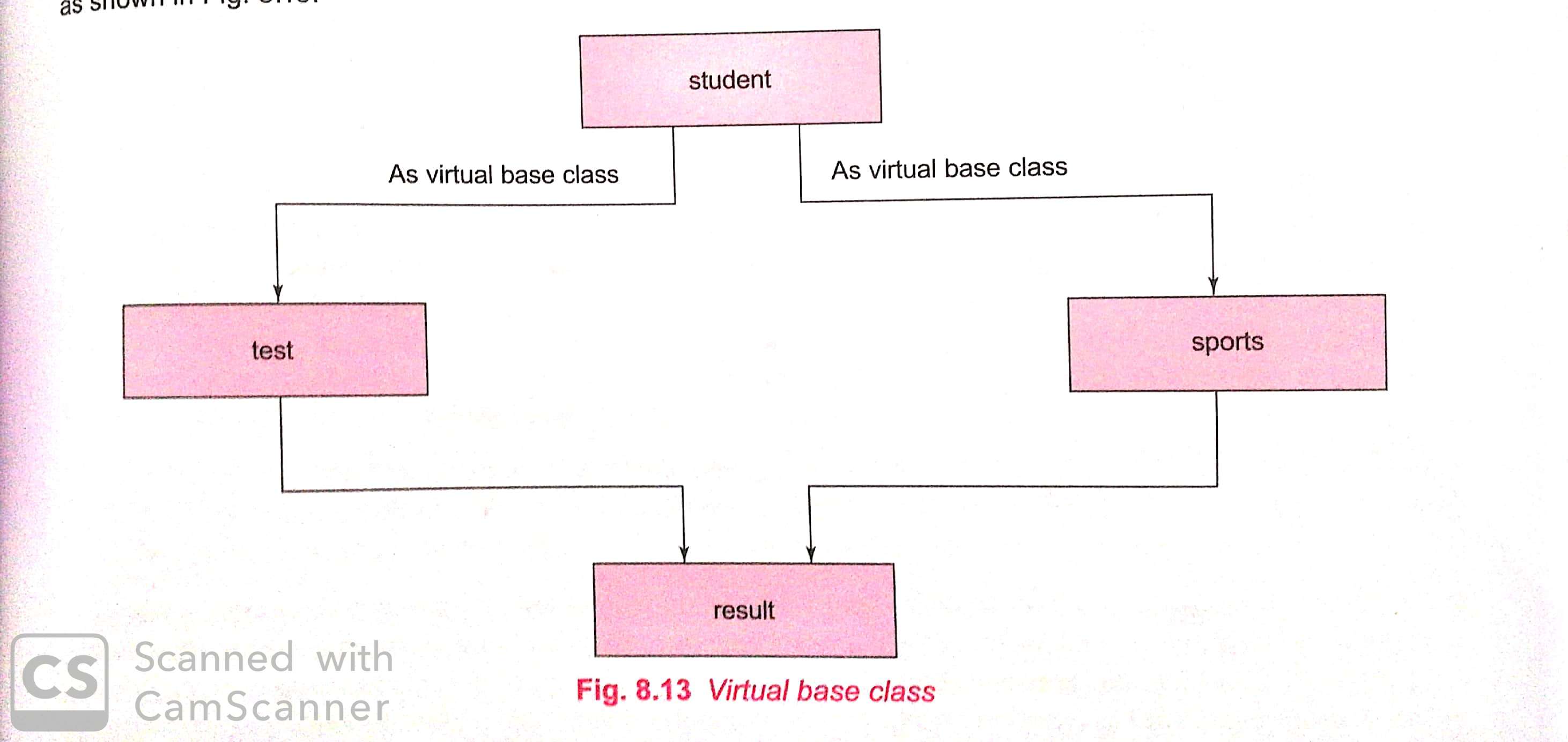
**HYBRID INHERITANCE**

* It is a concept when we have to use two types of inheritance in the same program.



VIRTUAL BASE CLASSES

* The duplication of inherited members due to these multiple paths can be avoided by making the common base class (ancestor class) as virtual base class while declaring the direct or intermediate base classes
* When a class is made a virtual base class, C++ takes necessary care to see that only one copy of that class is inherited, regardless of how many inheritance paths exist between the virtual base class and a derived class.

****

**Example Program:-**

**#include<iostream.h>**

**class student**

**{**

**protected:**

**int roll\_number;**

**public:**

**void get\_number(int a)**

**{**

**roll\_number =a;**

**}**

**void put\_number()**

**{**

**cout<<”roll no :” <<roll\_number << “\n”;**

**}**

**};**

**class test: virtual public student**

**{**

**protected:**

**float part1,part2;**

**public:**

**void get\_marks(float x,float y)**

**{**

**part1=x;**

**part2=y;**

**}**

**void put\_marks()**

**{**

**cout<<”marks obtained : “<<”\n” <<”part1 =”<<part1<<”\n”<<”part2 =”<<part2<<”\n”;**

**}**

**};**

**class sports : public virtual student**

**{**

**protected:**

**float score;**

**public:**

**void get\_score(float s)**

**{**

**score=s;**

**}**

**void put\_score()**

**{**

**cout<<”sports wt:”<< score << “\n\n”;**

**}**

**};**

**class result :public test, public sports**

**{**

**float total;**

**public:**

**void display();**

**};**

**void result::display()**

**{**

**total=part1+part2+score;**

**put\_number();**

**put\_marks();**

**put\_score();**

**cout<<”total score : “ <<total<<”\n”;**

**}**

**int main()**

**{**

**result student\_1;**

**student\_1.get\_number(678);**

**student\_1.get\_marks(30.5,25.5);**

**student\_1.get\_score(7.0);**

**student\_1.display();**

**return 0;**

**}**

**ABSTRACT CLASS**

* **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 design concept in program development and provides a base upon which other classes may be built.**
* **The general form of using abstract class is shown below.**

## Pure Virtual definitions

* Pure Virtual functions can be given a small definition in the Abstract class, which you want all the derived classes to have. Still you cannot create object of Abstract class.
* Also, the Pure Virtual function must be defined outside the class definition. If you will define it inside the class definition, complier will give an error. Inline pure virtual definition is Illegal.

// Abstract base class

class Base

{

public:

virtual void show() = 0; //Pure Virtual Function

};

void Base :: show() //Pure Virtual definition

{

cout << "Pure Virtual definition\n";

}

class Derived:public Base

{

public:

void show()

{

cout << "Implementation of Virtual Function in Derived class\n";

}

};

int main()

{

Derived d;

d.show();

}

**CHAPTER – 9**

**POINTERS**

* A pointer is a derived data type that refers to another data variables by storing the variables memory address rather than data.
* A pointer variable defines where to get the value of a specific data variable instead of defining actual data.

**DECLARING AND INITIALIZING POINTERS**

* **The declaration of a pointer variable takes the following form:**

**Data-type \*pointer-variable**

* **Here pointer variable is the name of the pointer, and the data-type refers to one of the valid C++ data types such as int ……**
* **The data-type is followed by an asterisk(\*) symbol, which distinguishes a pointer variable from other variables to the compiler.**
* **However it is necessary to understand that a pointer is able to point to only one data type at the specific time.**

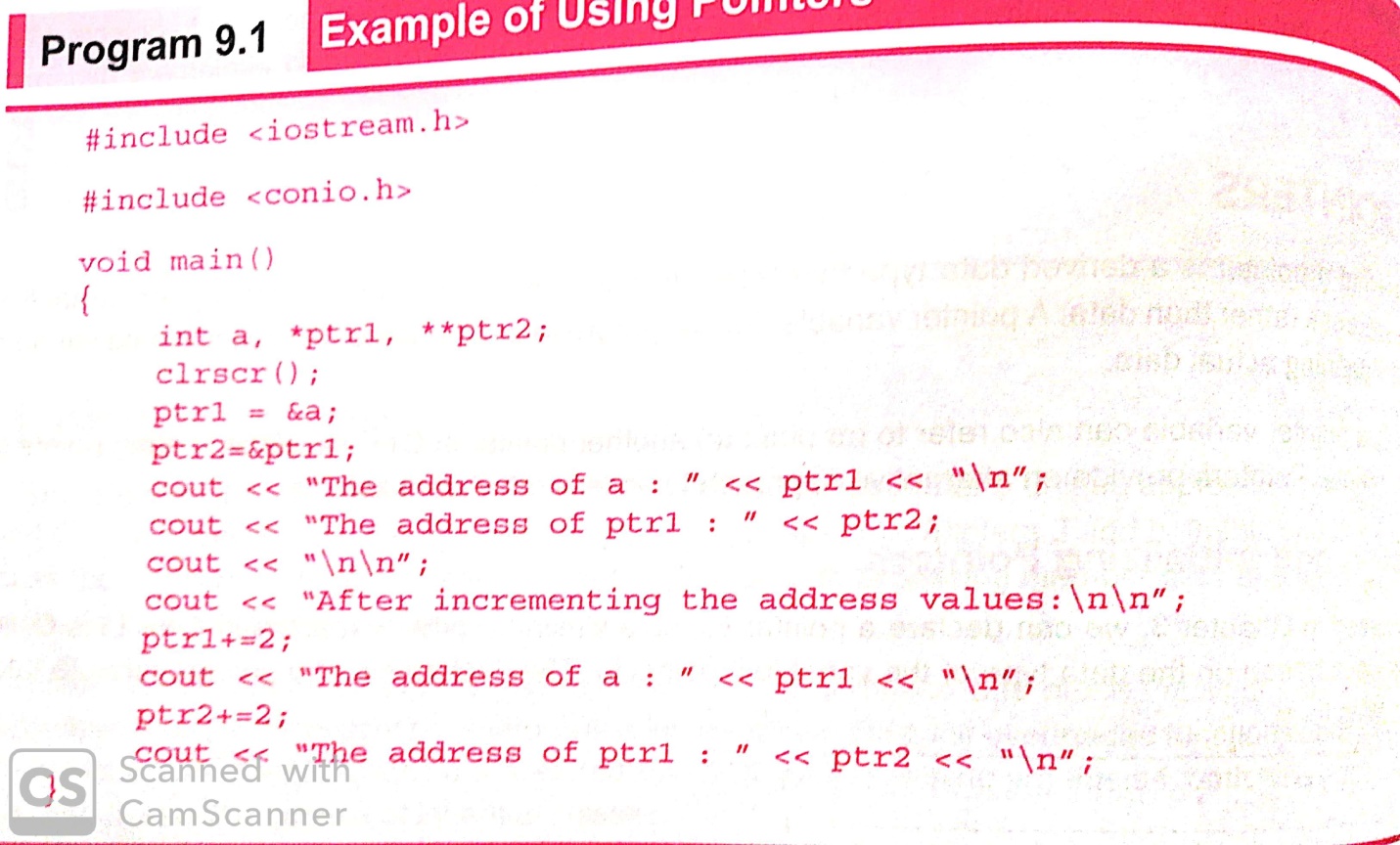
**int \*ptr;**

* **Here ptr is a pointer variable and points to an integer data type. The pointer variable, ptr should contain the memory location of any integer variable.**
* **We can initialize a pointer variable as follows:**

**int \*ptr,a; // declaration**

**ptr = &a; // initialization**

* **The pointer variable ptr contains the address of the variable a.**
* **We use the ‘address of’ operator or reference operator to retrieve the address of the variable.**
* **The second statement assigns the address of the variable to the pointer ptr.**

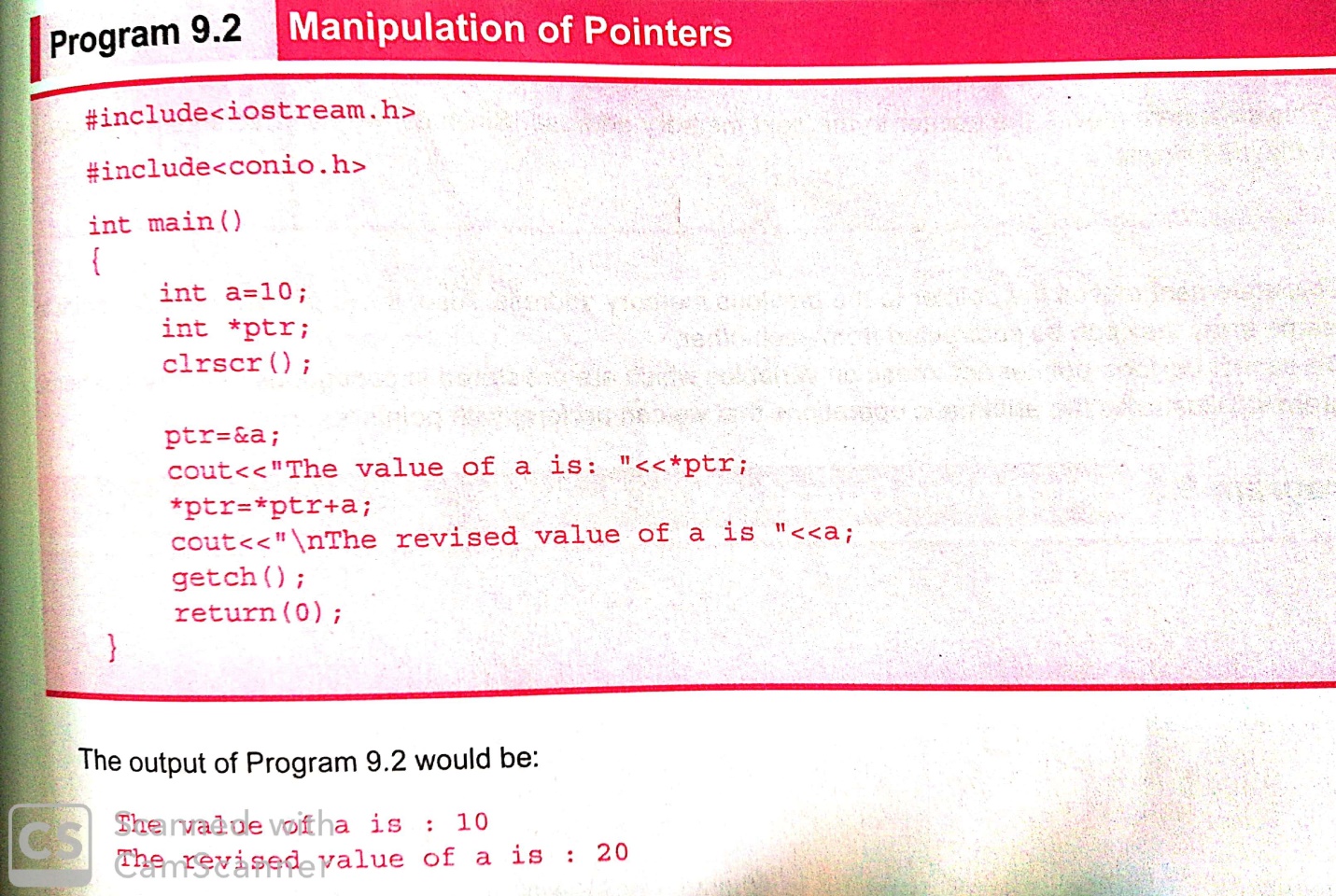
****

**Manipulation of pointers**

* **We can manipulate a pointer with the indirection operator which is also known as dereference operator with this operator, we can indirectly access the data variable content.**
* **It takes the following general form:**

**\*pointer\_variable**

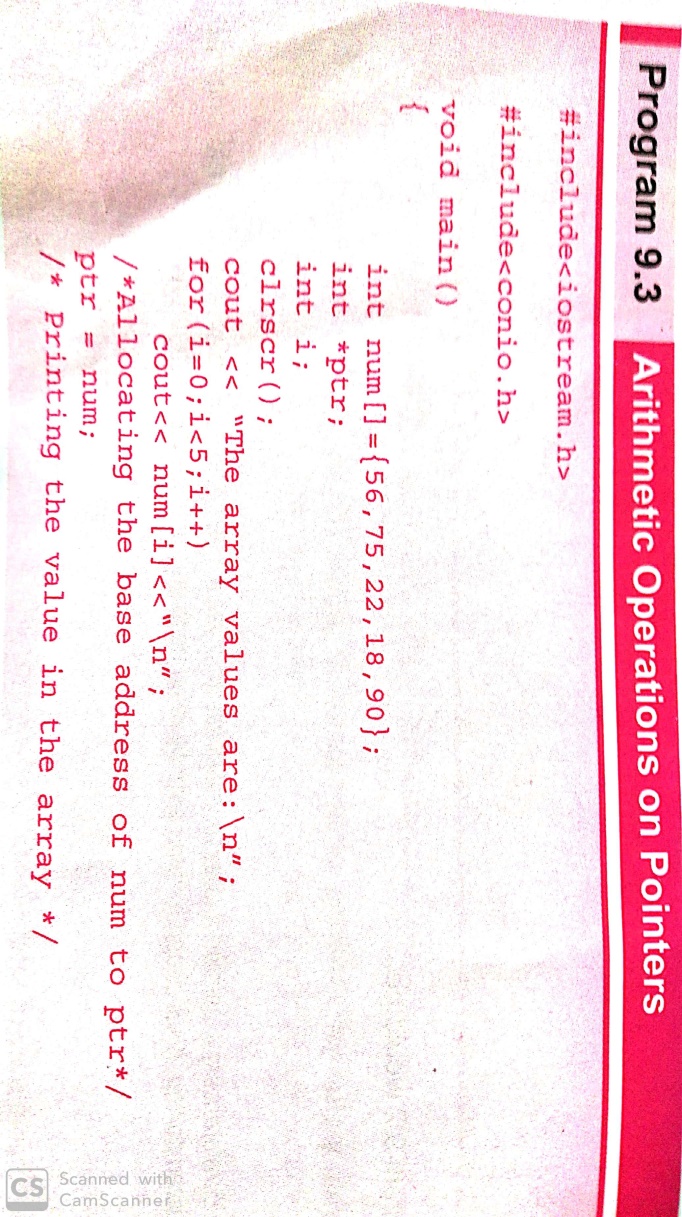
* **The use of dereference operator to access and alter the contents of the variable being pointed by the pointer variable.**

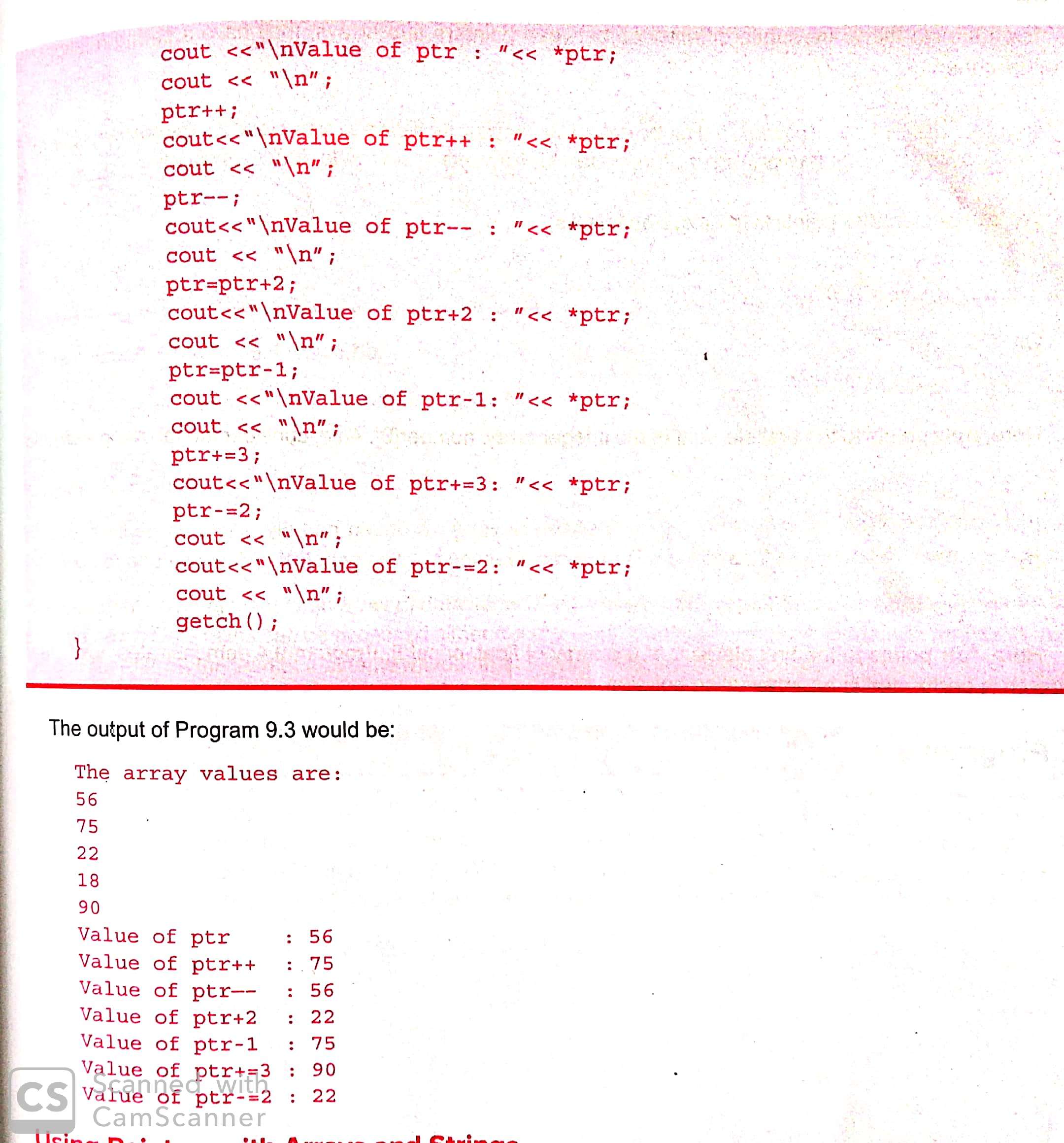
****

**Pointer expressions and pointer arithmetic**

* **C++ allows pointers to perform the following arithmetic operations.**
  + **A pointer can be incremented (++) or decremented (--)**
  + **Any integer can be added to or subtracted from a pointer**
  + **One pointer can be subtracted from another.**
* **We can increment the pointer variable.**

**aptr++ (or) ++aptr**

****

****

**Arrays of pointers**

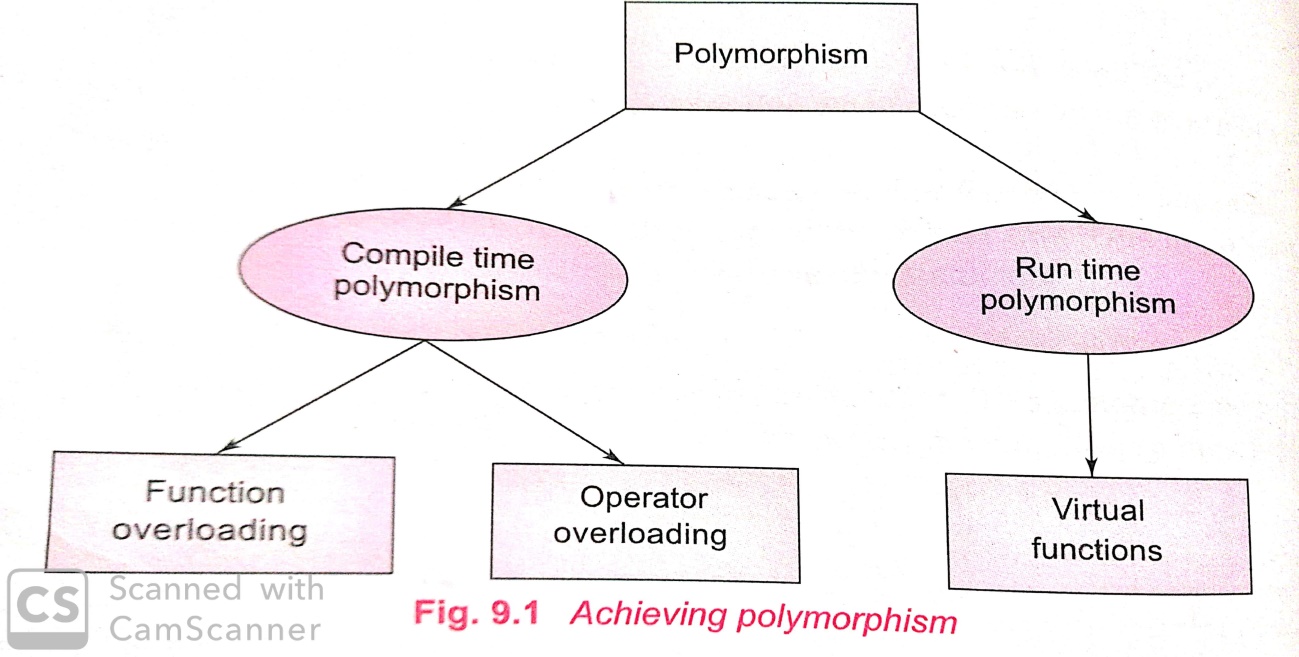
* **An array of pointers point to an array of data items.**
* **Each element of the pointer array points to an item of the data array.**
* **Data items can be accessed either directly or by dereferencing the elements of pointer array.**

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

**POLYMORPHISM**

* **Polymorphism means ‘ one name,multiple forms’.**

****

**Compile time polymorphism**

* **It simply means that an object is bound to its function call at the compile time.**
* **Compile time polymorphism is achieved in two ways:**
  + **Function overloading**
  + **Operator overloading**

**RUN TIME POLYMORPHISM**

* **The use of class resolution operator in resolving the ambiguity works fine for us.**
* **In c++ run time polymorphism is achieved with the help of virtual functions.**

**VIRTUAL FUNCTIOS**

* **Polymorphism refers to the property by which objects belonging to different classes are able to respond to the same message.**
* **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.**

**EXAMPLE PROGRAMS**

**#include<iostream.h>**

**Class base**

**{**

**Public:**

**Void display()**

**{**

**Cout<<”\n Display base”;**

**}**

**Virtual void show()**

**{**

**Cout<<”\n show case”;**

**}**

**};**

**Class derived: public base**

**{**

**Public:**

**Void display()**

**{**

**Cout<<”\n Display derived”;**

**}**

**Void show()**

**{**

**Cout<<”\n show derived”;**

**}**

**};**

**RULES FOR VIRTUAL FUNCTIONS**

* 1. **The virtual functions must be members of some class.**
  2. **They cannot be static members.**
  3. **They are accessed by using object pointers.**
  4. **A virtual function can be a friend of another class.**
  5. **A virtual function in a base class must be defined, even though it may not be used.**

**PURE VIRTUAL FUNCTIONS**

* **A “do-nothing” function may be defined as follows.**

**Virtual void display()=0;**

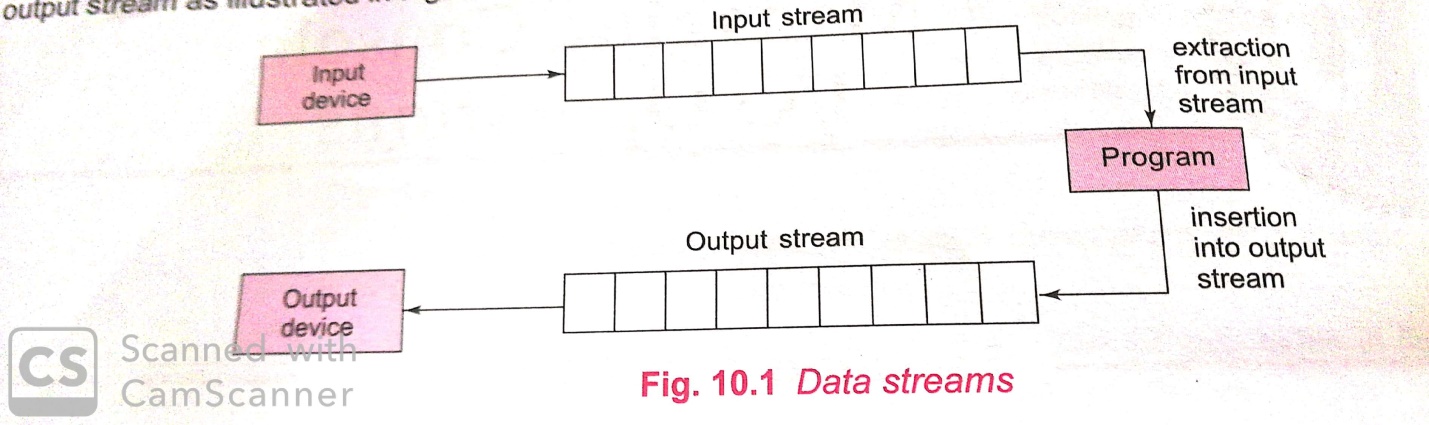
* **Such functions are called pure virtual functions.**
* **A pure virtual function is a function declared in a base class that has no definition relative to the base class.**

**MANAGING CONSOLE I/O OPERATIONS**

* **C++ supports a rich set of I/O functions and operations to do this.**
* **C++ uses the concept of stream and stream classes to implement its I/O operations with the console and disk files.**

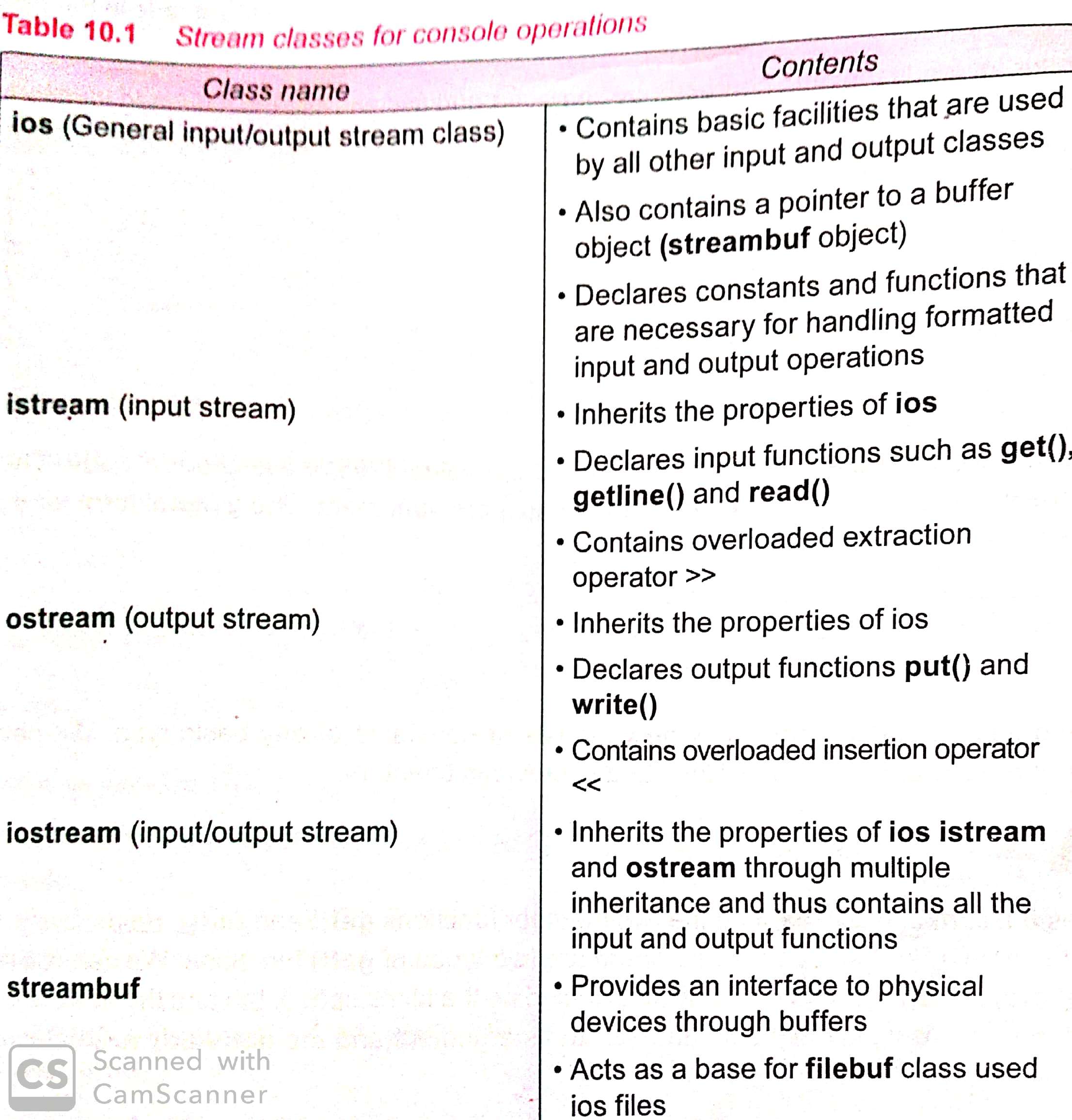
**C++ STREAMS**

* **C++ is designed to work with a wide variety of devices including terminals, disks and tape drives.**
* **The I/O system supplies an interface to the programmer that is independent of the actual device being accessed.**
* **This interface is known as stream. A stream is a sequence of bytes.**
* **It acts either as a source from which the input data can be obtained or as a destination to which the output data can be sent.**
* **The source stream that provides data to the program is called the input stream and the destination stream that receives output from the program is called the output stream.**

****

**C++ STREAM CLASSES**

* **C++ I/O system contains a hierarchy of classes that are used to define various streams to deal with both the console and disk files.**
* **These classes are called stream classes.**

****