Class and objects

==============

A Class is a user-defined data type that has data members and member functions.

Data members are the data variables and member functions are the functions used to manipulate these variables together.

These data members and member functions define the properties and behaviour of the objects in a Class.

Eg:

Student Class-->

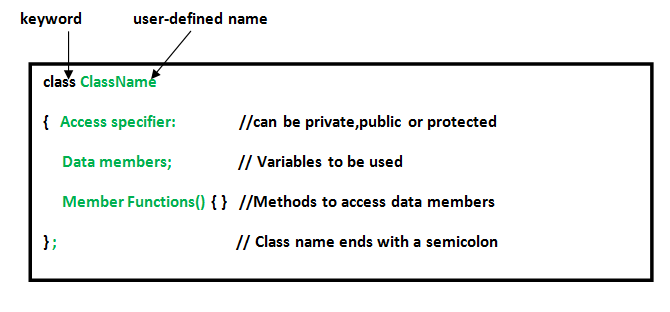
data member-->student\_name,age,mobile,marks..etc

f()--->display(),changes\_name(),marks\_update()..etc

object-->class variable

object-->Object is an instance of a class. All data members and member functions of the class can be accessed with the help of objects.

Syntax



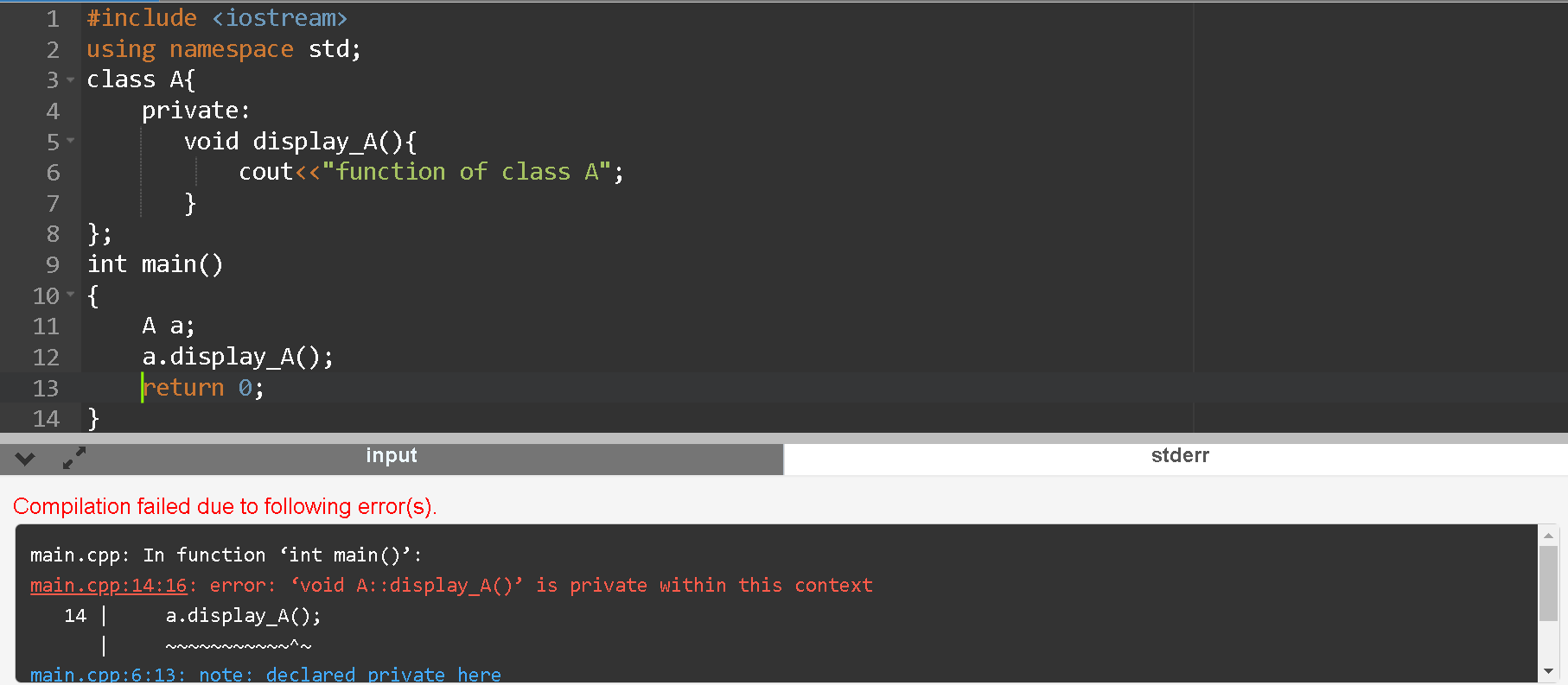
Access specifier

cess specifiers:

* public - members are accessible from outside the class
* private - members cannot be accessed (or viewed) from outside the class
* protected - members cannot be accessed from outside the class, however, they can be accessed in inherited classes.
* 

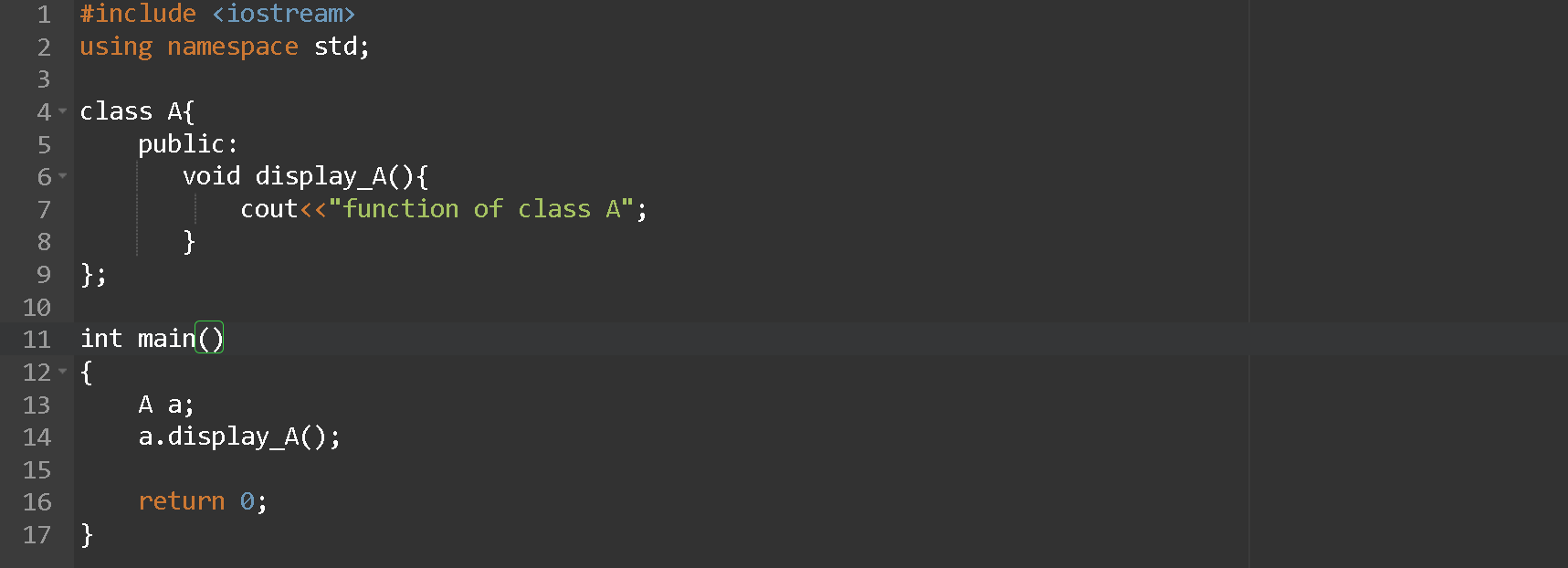
Example:1

Access of private member is restricted



Example2

Access of public member is allowed in main()



Output: function of class A

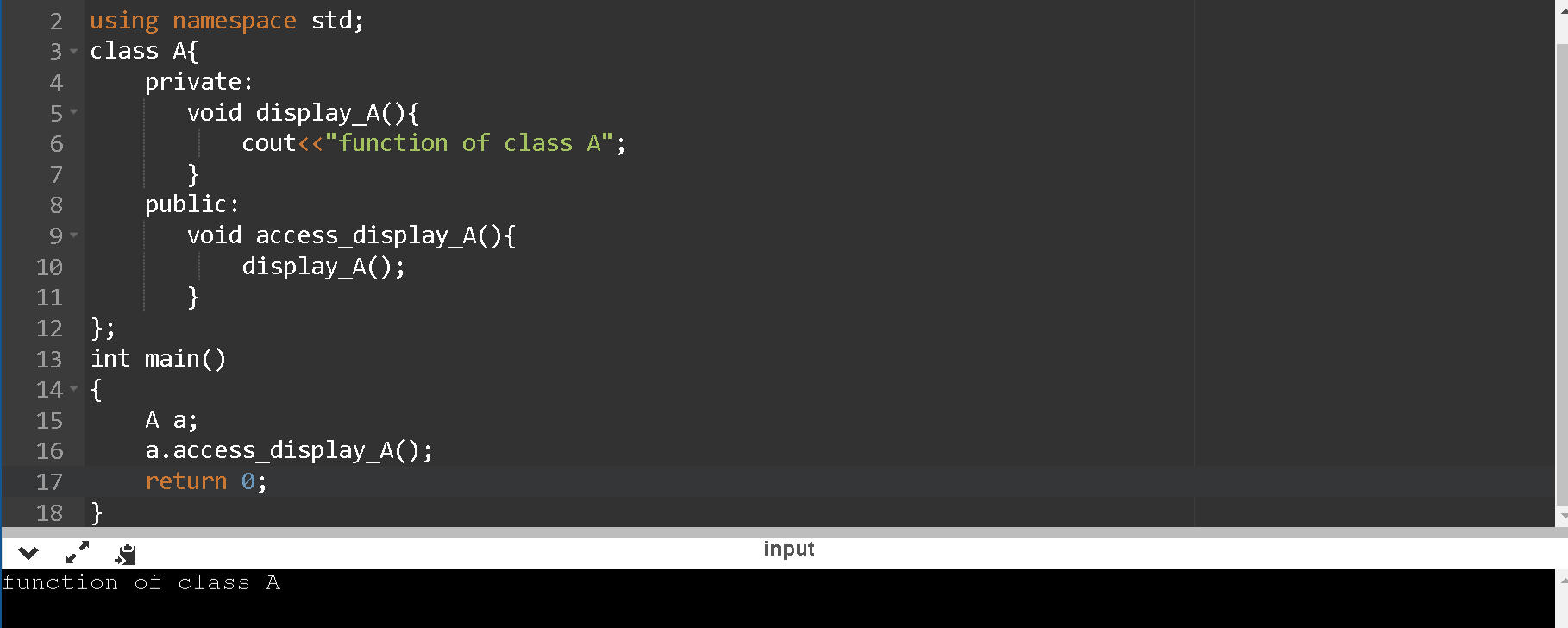
How to access Private member of class

Private: The class members declared as private can be accessed only by the functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. Only the member functions or the friend functions are allowed to access the private data members of a class.

So, to take access of private member, we must define some public member function which can take access of private member(indirectly).

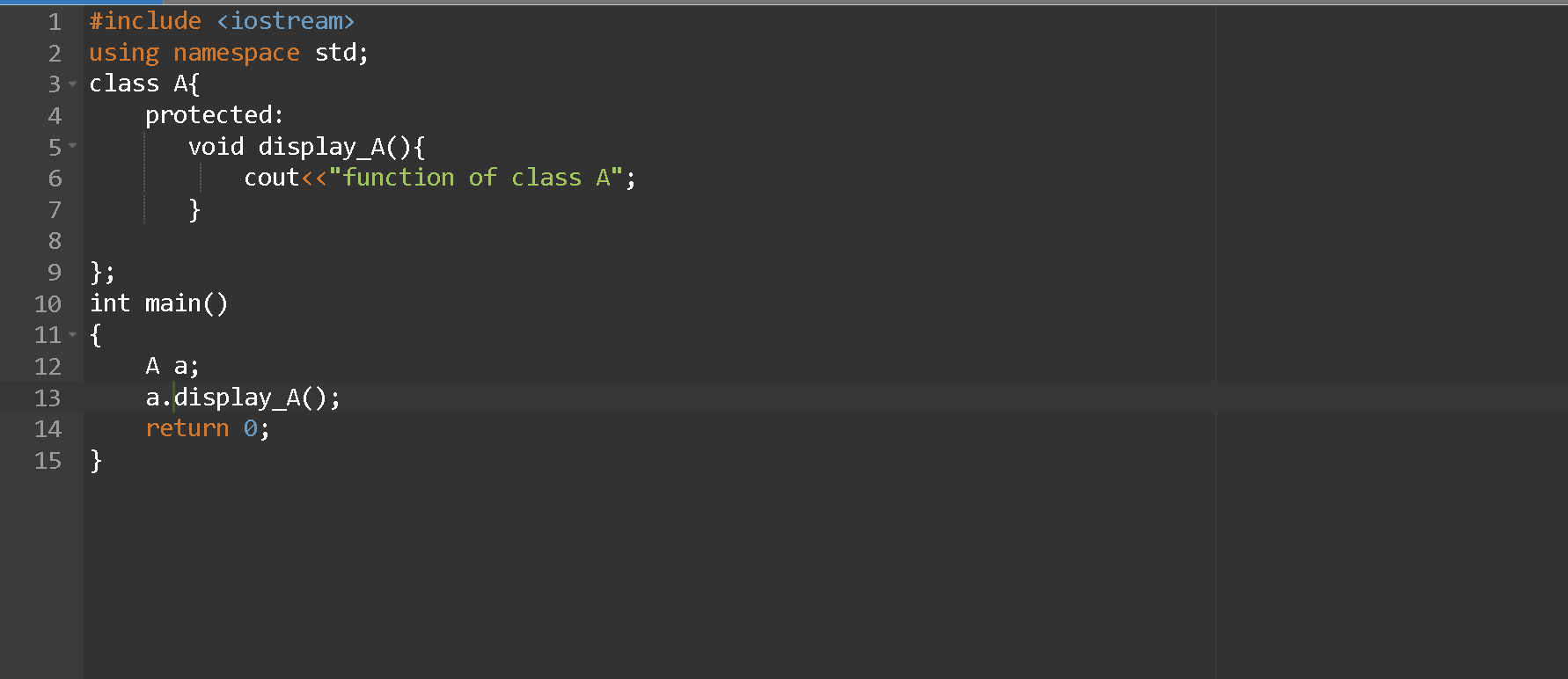
Example 3

public member function can take access of private member(indirectly).



Protected works as a private when there are no child class

Example 4



Output: error

Constructors:

Constructors are member functions of a class which are used to initialize the data members of the class objects. These functions are automatically called when an object of its class is created.

There is no need to call these functions.

Characteristics of constructors:

1)The name of a constructor is same as that of class in which it is declared.

2)Constructors do not have any return type, not even void.

3)Constructors are always defined in the public section of the class.

4)They cannot be inherited, though a derived class can call the base class constructor.

5)A constructor may not be static.

6)Like other C++ functions, constructors can also have default arguments.

7)Member functions may be called

8)Constructors are not called directly

9)Constructors show polymorphism in a class

Types of Constructors:

There are three types of constructors.

1. Default Constructors

2. Parameterized Constructors

3. Copy Constructors

A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.

Sample program

#include <iostream>

using namespace std;

class Employee

{

public:

Employee()

{

cout<<"Default Constructor Invoked"<<endl;

}

};

int main(void)

{

Employee e1; //creating an object of Employee

Employee e2;

return 0;

}

A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

Sample program

#include <iostream>

using namespace std;

class Employee {

public:

int id;//data member (also instance variable)

string name;//data member(also instance variable)

float salary;

Employee(int i, string n, float s)

{

id = i;

name = n;

salary = s;

}

void display()

{

cout<<id<<" "<<name<<" "<<salary<<endl;

}

};

int main() {

Employee e1 =Employee(101, "ajay", 44444); //creating an object of Employee

Employee e2=Employee(102, "ankur", 33333);

e1.display();

e2.display();

return 0;

}

A copy constructor is a member function that initializes an object using another object of the same class.

#include<iostream>

#include<string.h>

using namespace std;

class student

{

int rno;

char name[50];

double fee;

public:

student(int,char[],double);

student(student &t) //copy constructor

{

rno=t.rno;

strcpy(name,t.name);

fee=t.fee;

}

void display();

};

student::student(int no,char n[],double f)

{

rno=no;

strcpy(name,n);

fee=f;

}

void student::display()

{

cout<<endl<<rno<<"\t"<<name<<"\t"<<fee;

}

int main()

{

student s(101,"ajay",45665);

s.display();

student m (s); //copy constructor called

m.display();

return 0;

}

Inheritance

In C++, inheritance is a process in which one object acquires all the properties and behaviours of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviours which are defined in other class.

To Achieve DRY Principal.

DRY🡪Do not Repeat Yourself

Syntax

class derived\_class\_name : visibility-mode base\_class\_name

{

// body of the derived class.

}

Types of Inheritance

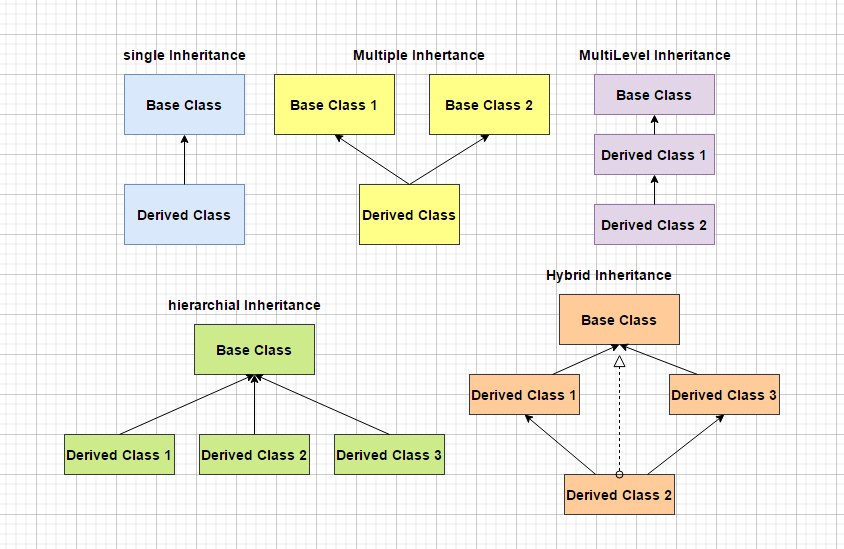
Single inheritance

Multiple inheritance

Hierarchical inheritance

Multilevel inheritance

Hybrid inheritance



Visibility Mode

Public

The public and protected members of the base class remain public, and protected members of the derived class when we inherit a child class from the parent class in public visibility mode.

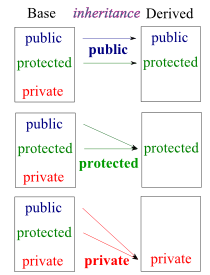
Protected

The public and protected members of the base class becomes protected members of the derived class when we inherit a child class from the parent class in protected visibility mode.

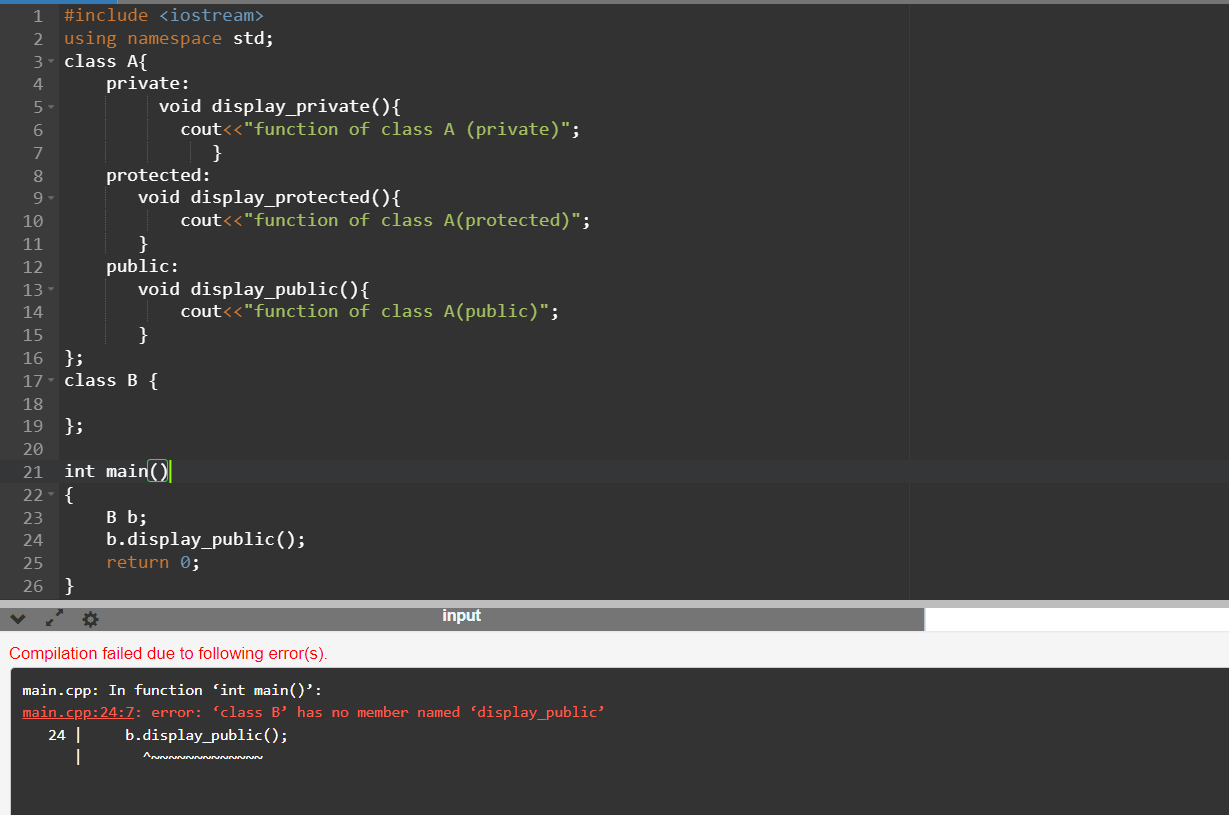
Private

The public and protected members of the base class becomes private members of the derived class when we inherit a child class from the parent class in private visibility mode.

Note: we can’t inherit the private member in any case



Example 1



Explanation: B is a normal independent class ,

display\_public() is a member f() of Class A ,can be called by A itself or

Its child.

The above problem can be solved by.

class B : public A{

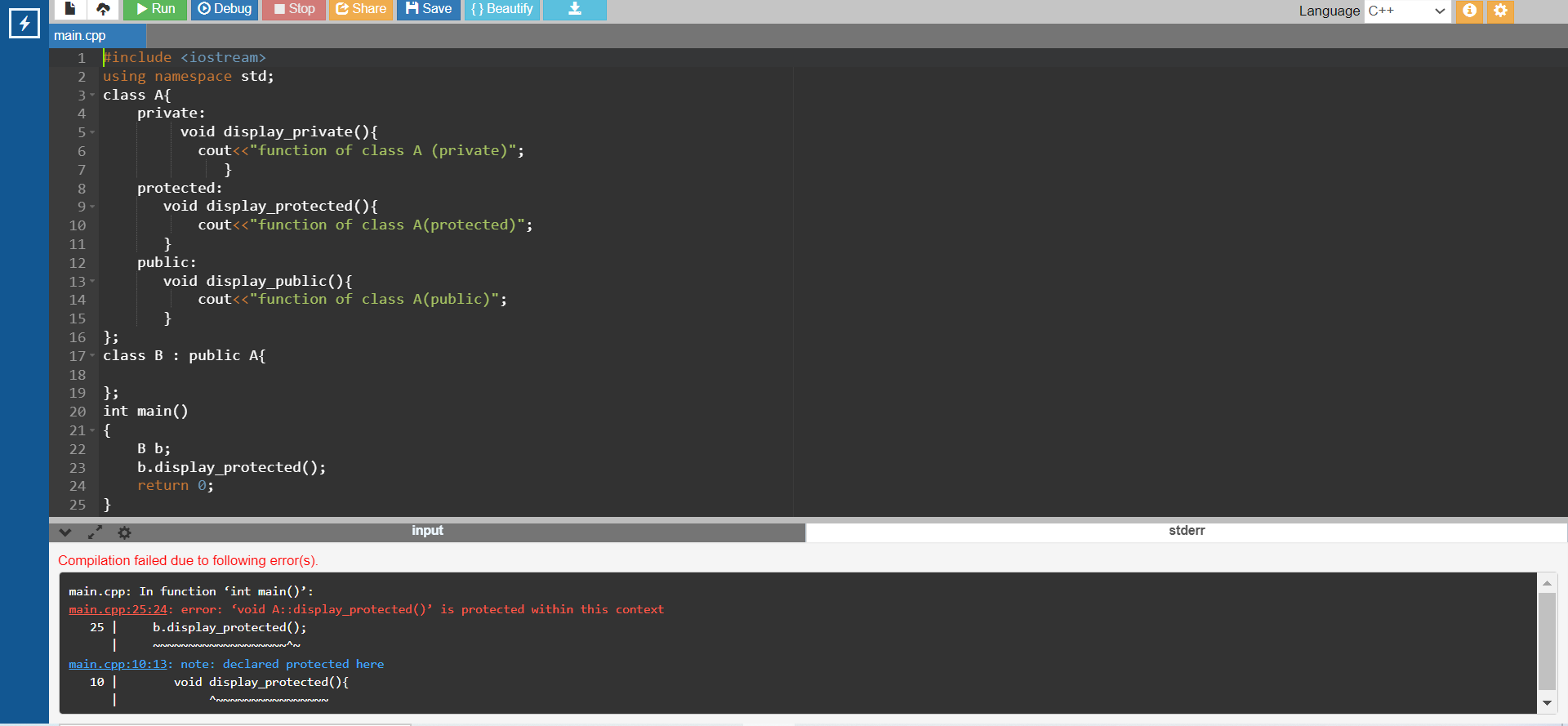
};



Example 2

AS we know we can take access of the protected member of parent

Class in child class Lets try it.



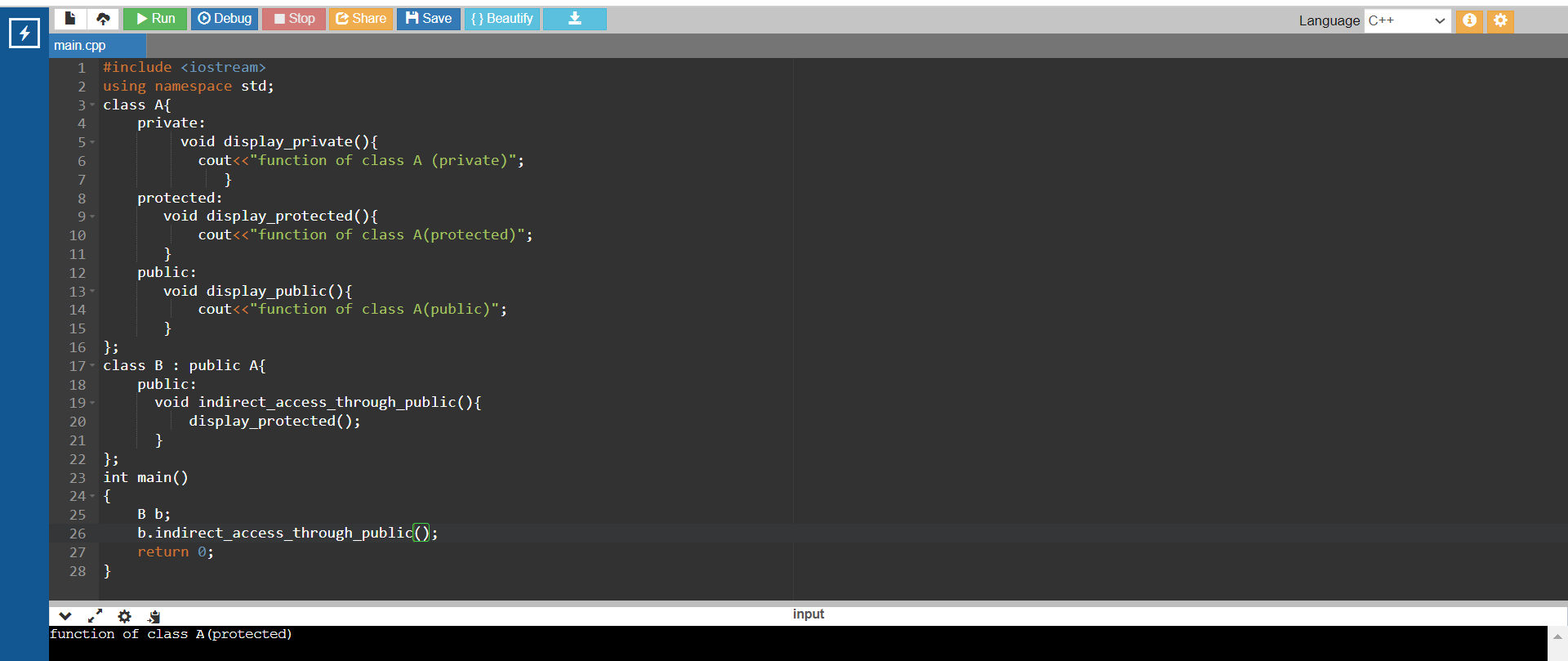
Explanation: This error occurred because we take access of Protected

Member of B in main() .

(from parent f() comes as a protected ,mentioned earlier)

Solution of this problem is :

Example 3



Polymorphism

Polymorphism word is the combination of "poly," which means many + "morphs," which means forms, which together means many forms. Polymorphism in C++ is when the behavior of the same object or function is different in different contexts.

Types of Polymorphism

Compile-time Polymorphism

Runtime Polymorphism

poly->many

morphism->forms

1)compile time (early binding)

Compile time polymorphism: The overloaded functions are invoked by matching the type and number of arguments. This information is available at the compile time and, therefore, compiler selects the appropriate function at the compile time. It is achieved by function overloading and operator overloading which is also known as static binding or early binding.

linker link the respective f() in compile time itself

2)run time (late binding): (function overiding)

Run time polymorphism: Run time polymorphism is achieved when the object's method is invoked at the run time instead of compile time. It is achieved by method overriding which is also known as dynamic binding or late binding.

linker link the respective f() in run time only.

function overloading:

Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters.

we can overload a f() either by

1> change the data\_type of parameter.

2> change the number of parameters.

simply by changing signature we achieve polymorphism

(same f() name can be used differently)

Note: return type has no affect.

eg:

void add(int ,int);

void add(double ,int);

void add(int ,double);

void add(int ,int,int);

//int add(int,int); // error same as void add(int ,int);

Example 1

# include <iostream>

using namespace std;

void add(int a,int b){

cout<<"addition:"<<a+b<<endl;

}

void add(double a,int b){

cout<<"addition:"<<a+b<<endl;

}

void add(int a,int b,int c){

cout<<"addition:"<<a+b+c<<endl;

}

int main()

{

add(4,5); // according to f() linkage ---->compile time .int int

add(4.5,9);

add(3,4,5);

}

Default parameter

A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the calling function doesn’t provide a value for the argument. In case any value is passed, the default value is overridden.

# include <iostream>

using namespace std;

void marksheet(int a,int b,int c,string Uname="RGPV"){

// Uname =default

cout<<Uname<<" Examination\n";

cout<<"subject1:"<<a<<endl;

cout<<"subject2:"<<b<<endl;

cout<<"subject3:"<<c<<endl;

}

int main()

{

marksheet(34,56,78);

marksheet(34,44,45,"BU Bhopal");

}

Example #

# include <iostream>

using namespace std;

void add(int a,int b,int c=0,int d=0){ // right most cornor

cout<<"addition:"<<a+b+c+d<<endl;

}

//Note: ensure that ,all non-default arguments are placed before any //default arguments in your function definition

int main()

{

add(2,3); // a=2,b=3,c=0,d=0

// add(2,3,5); // a=2,b=3,c=5,d=0

// add(2,3,5,6); // a=2,b=3,c=5,d=6

}

Operator overloading

Operator overloading is a compile-time polymorphism. It is an idea of giving special meaning to an existing operator in C++ without changing its original meaning.

# include <iostream>

using namespace std;

class Student {

public:

int marks;

Student(int a){

marks=a;

}

Student(){

marks=0;

}

Student operator +(Student s){ // s=b

Student temp;

if(marks>s.marks)

temp.marks=marks;

else

temp.marks=s.marks;// s.marks=b.marks(21)+23

return temp;}

};

// highest marks amoung 4 student

int main(){

Student a(23),b(21),c(45),d(2),e;

e=a+b+c+d; // a+b--> a.operator+(b)

cout<<e.marks;

}

Runtime Polymorphism

Function overriding

The child class inherits all the data members and the member functions present in the parent class. If we wish to override any functionality in the child class, we can implement function overriding. Function overriding means creating a newer version of the parent class function in the child class.

A virtual function (also known as virtual methods) is a member function that is declared within a base class and is re-defined (overridden) 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 method.

Virtual functions ensure that the correct function is called for an object,

Problem Question:

WAP to define a class named Cybrom having data member

Protected

enroll,duration,fees,name,course.

Public function

1)set\_course(course) function 🡪detail (make it virtual)

course="Data Science" then -🡪duration=8,fees=60000.

course=" Data Analytics" then -🡪duration=5,fees=40000.

course=" python " then -🡪duration=6,fees=50000.

course=" java " then -🡪duration=10,fees=80000.

course="Data Science" then -🡪duration=8,fees=60000.

2)parameterized Constructor

Enroll,name,course

3)display ()

Also define child class named CybromIntregratedCouse

Data member:

additional\_course

parameterized constructor

Enroll,name,course, additional\_course

//Call the superclass constructor in the subclass

2) set\_course(course) function 🡪modified definition of intregrated course.

# include<iostream>

using namespace std;

class Cybrom{

protected:

int enroll,duration,fees;

string name,course;

public:

void set\_course(string course){

if(course=="Data Science")

duration=8,fees=60000;

else if(course=="Data Analytics")

duration=5,fees=40000;

else if(course=="python")

duration=6,fees=50000;

else if(course=="java")

duration=10,fees=80000;

}

Cybrom(int en,string n,string c){

enroll=en;name=n;course=c;

}

void display(){

cout<<"enrollment number:"<<enroll<<endl;

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

cout<<"course:"<<course<<endl;

cout<<"duration:"<<duration<<endl;

cout<<"fees:"<<fees<<endl;

}

};

class CybromIntregratedCouse: public Cybrom{

string additional\_course;

public:

CybromIntregratedCouse(int en,string n,string c,string ac)

: Cybrom(en,n,c) // Call the superclass constructor in the subclass' initialization list.

{

additional\_course=ac;

}

void set\_course(string acourse){

if(additional\_course=="networking")

course=course+" and "+additional\_course, duration=11,fees=95000;

else if(additional\_course=="linux")

course=course+" and "+additional\_course,duration=12,fees=80000;

}

};

int main(){

Cybrom \*base;

Cybrom c(101,"ajay","python");

CybromIntregratedCouse c2(101,"ajay","python","linux");

base=&c2;

base->set\_course("linux");

base->display();

}

Example 2

Create a Vehicle class with max\_speed, mileage ,capacity,total\_price instance attributes(protected).

Public function

parameterized constructor.

fare()--> The default fare charge of any vehicle is seating capacity \* 100.

display()->display vehicle details.

And also,

Create a Bus child class that inherits from the Vehicle class.

Private member : name.

Public function

parameterized constructor.

Give the Bus capacity a default value of 50.

override fare()->

If Vehicle is Bus instance, we need to add an extra 10% on full fare as a maintenance charge.

So total fare for bus instance will become the final amount = total fare + 10% of the total fare.

override display()->display all details.

Soln:

# include <iostream>

using namespace std;

class Vehicle{

protected:

int max\_speed,milage,capacity;

double total\_price;

public:

Vehicle(int mx,int mi,int c){

max\_speed=mx;

milage=mi;

capacity=c;

total\_price=0;

}

virtual void fare(){

total\_price=capacity\*100;

}

virtual void display(){

cout<<"maximum speed:"<<max\_speed<<" milage:"<<milage<<endl;

cout<<"capacity:"<<capacity<<" price:"<<total\_price<<endl;

}

};

class Bus : public Vehicle{

private:

string name;

public:

Bus(string n,int mx,int mi,int c=50):Vehicle (mx,mi,c) {

name=n;

}

void fare(){

total\_price=capacity\*100;

total\_price=total\_price+(.1\*total\_price);

}

void display(){

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

cout<<"maximum speed:"<<max\_speed<<" milage:"<<milage<<endl;

cout<<"capacity:"<<capacity<<" price:"<<total\_price<<endl;

}

};

int main(){

Vehicle \*v;

Bus b("volvo",200,30);

v=&b;

v->fare();

v->display();

// b.fare();

// b.display();

//Vehicle v(120,20,200);

//v.fare();

//v.display();

}

/\*

==================================================

\*/

Abstract class

In programming, an abstract class in C++ has at least one pure virtual function by definition. In other words, a function that has no definition. The abstract class's descendants must define the pure virtual function; otherwise, the subclass would become an abstract class in its own right.

Abstract class:it is merely a design of base class

here we declared some function (pure virtual f())

and also may have some defination.

we can't create a object of base class(abstract)

we have to implement all pure virtual f() in child.

Abstract --->min one pure virtual f()

class BaseAbstractClass{ // design

public:

virtual void f1()=0; //decleartion

virtual void f2(int a)=0;

void displaymessage(){

cout<<"i m f() of abstract class only be accessed by child";

}

};

class Derived: public BaseAbstractClass{

public:

void f1(){

cout<<"implementation of f1 in derived\n";

}

void f2(int a){

cout<<"implementation of f2 in derived\n";

}

};

int main()

{

//BaseAbstractClass obj; // abstract class have no objects

BaseAbstractClass \*base;

Derived d;

base=&d;

base->f1();

base->f2(3);

// d.f1();

// d.f2(3);

return 0;

}

# include<iostream>

using namespace std;

class Employee{

protected: string name ,eid;

int salary;

public:

Employee(string n,string e,int salary){

name=n;

eid=e;

salary=salary;

}

void display(){

cout<<"eid:"<<eid<<endl;

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

cout<<"salary:"<<salary<<endl;

}

};

int main(){

Employee e("ajay","101",200);

e.display();

}

Static function and variable

The static keyword is used with a variable to make the memory of the variable static once a static variable is declared its memory can’t be changed. Static members of a class are not associated with the objects of the class. Just like a static variable once declared is allocated with memory that can’t be changed every object points to the same memory.

Rules of static member

A static member function is independent of any object of the class.

A static member function can be called even if no objects of the class exist.

A static member function can also be accessed using the class name through the scope resolution operator.

A static member function can access static data members and static member functions inside or outside of the class.

Static member functions have a scope inside the class and cannot access the current object pointer.

You can also use a static member function to determine how many objects of the class have been created.

Example 1

WAP to create a class named Bank ,having public static member

name\_bank,rate,customer\_number.

In default constructor increment the value customer\_number.

Find count of customer of bank.

#include <iostream>

using namespace std;

class Bank{

public :

static string name\_bank;

static double rate;

static int customer\_number;

Bank(){

customer\_number++;

}

static void change\_rate(double modified\_value){

rate=modified\_value;

}

static void customer\_number\_detail(){

cout<<customer\_number<<endl;

}

};

string Bank::name\_bank="BOB";

double Bank::rate=4.7;

int Bank::customer\_number=0;

int main()

{

Bank c1,c2,c3;

Bank::customer\_number\_detail();

cout<<c1.name\_bank<<"---->"<<c1.rate<<endl;

cout<<c2.name\_bank<<endl;

cout<<c3.name\_bank<<endl;

Bank::change\_rate(9.9);

cout<<c1.rate;

}

Scope of Variable

The scope of a variable refers to the extent of the block of code in which the variable can be accessed, modified, and worked with.

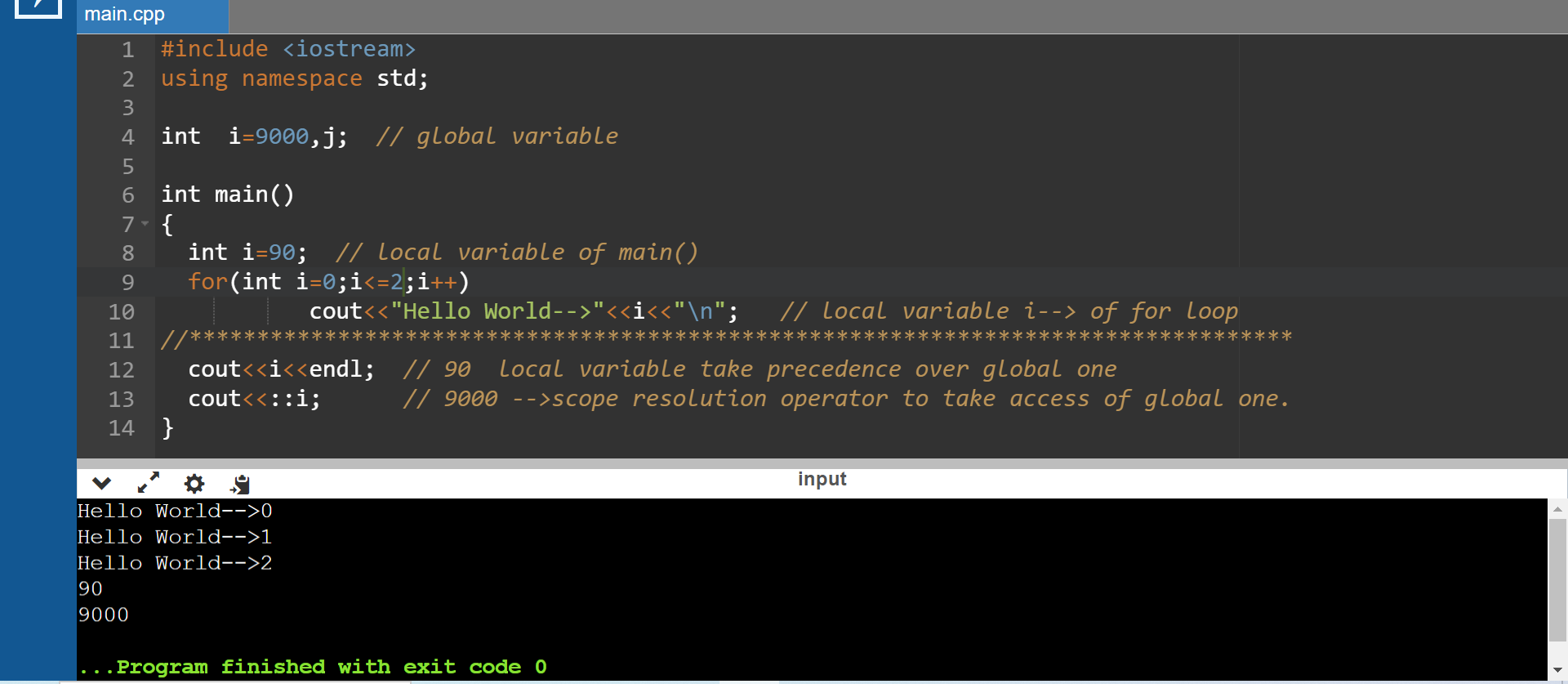
There are mainly two kinds of variable scopes.

Global Variables:

Global variables are declared at the top of the program outside any function and are accessible from anywhere in the entire program.

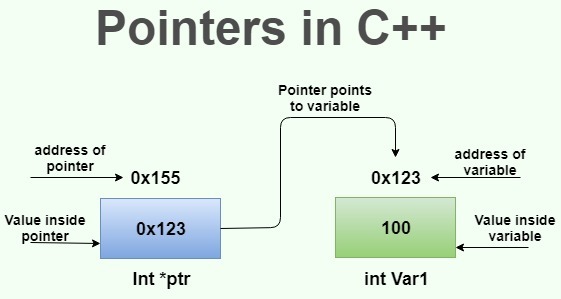
Local Variables:

Local variables are declared inside a function or a block of code and are accessible only within the function or the block of code in which it was declared.



Pointer

The pointer in C++ language is a variable, it is also known as locator or indicator that points to an address of a value.



datatype \*var\_name; // syntax

int Var1=100;

int \*ptr; // ptr can point to an address which holds int data

ptr=&Var1;

Example

#include <iostream>

using namespace std;

int main()

{

int num=30;

int ∗ p;

p=&num;//stores the address of number variable

cout<<"Address of number variable is:"<<&num<<endl;

cout<<"Address of p variable is:"<<p<<endl;

cout<<"Value of p variable is:"<<\*p<<endl;

return 0;

}