INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

EEC-101
Programming with C++

Module-4:
Object Oriented Programming

About Subject

- Object Oriented Programming Concepts
  - Data hiding,
  - Abstract data types,
  - Classes and Access control;
  - Class Implementation-
    - constructors, default constructor, copy constructor,
    - destructor
  - Operator overloading
  - Friend functions
  - Introduction to Templates

Procedural Programming

- Focus is on processes or actions that a program takes
- Programs are typically a collection of functions
- Data is declared separately
- Data is passed as arguments into functions
- Fairly easy to learn

- Functions need to know the structure of the data.
  - if the structure of the data changes many functions must be changed
- As programs get larger they become more:
  - difficult to understand
  - difficult to maintain
  - difficult to extend
  - difficult to debug
  - difficult to reuse code
  - fragile and easier to break

Object-Oriented Programming

- As the name suggests, Object-Oriented Programming uses objects in programming.
- The object-oriented programming centers around modeling real-world problems in terms of objects which contrasts with older approaches that emphasize a function-oriented view.
- The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Object-Oriented Programming

- Classes and Objects
    - focus is on classes that model real-world domain entities
    - allows developers to think at a higher level of abstraction
    - used successfully in very large programs

- Modular
- Easily understandable
- Reusable
- Adaptable to changes

Limitations

- Not a panacea
  - OO Programming won’t make bad code better
  - not suitable for all types of problems
  - not everything decomposes to a class
- Learning curve
  - usually a steeper leaning curve, especially for C++
  - many OO languages, many variations of OO concepts
- Design
  - usually more up-front design is necessary to create good models and hierarchies
- Programs can be:
  - larger in size
  - slower
  - more complex

Object-Oriented Programming

- A program in an object-oriented language is a collection of interacting objects.

- Interacting objects means that they can send and receive messages and process the messages
- There are no functions outside objects i.e., no global functions.

- C++ is not a purely object-oriented language.
- It can have global functions as well as objects.
- A C++ program, like in C, must have the function main, and execution starts at main.

Important object-oriented features

- Objects
- Classes
- Data Abstraction
- Data Hiding
- Data Encapsulation
- Inheritance
- Polymorphism
- Overloading
- Reusability

Class

- The building block of C++ that leads to Object-Oriented programming is a Class.
- It is a user-defined data type, which holds its own data members and member functions

  - blueprint from which objects are created ✗
  - a user-defined data-type ✓
  - has attributes (data) ✓
  - has methods (functions)
  - can hide data and methods
  - provides a public interface

$$\begin{tikzcd}
\text{Account} \arrow[d] \arrow[dr] \arrow[drr] \arrow[rrr] \arrow[drrr] & & & \text{withdraw()} \\
\text{Name} & \text{No} & \text{Money} & \text{deposit()}
\end{tikzcd}$$

- **Example classes**
  - Account
  - Employee
  - Image

Objects

- An object represents a particular instance (i.e. a single occurrence) of a class.
  - created from a class
  - represents a specific instance of a class
  - can create many, many objects
  - each has its own identity
  - each can use the defined class methods

Account x

- **Example Account objects**
  - Frank's account is an instance of an Account
  - Jim's account is an instance of an Account
  - Each has its own balance, can make deposits, withdrawals, etc.

When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Class and Object

- Classes are data types based on which objects are created.
  - Characteristics of an object are represented in a class as **Properties or Attributes**.
  - The actions that can be performed by objects become functions of the class and are referred to as **Methods**.

For example:

- Consider a Class of *Vehicles* under which
  - Car, bike, truck, etc., represent individual Objects.

- Each *Vehicle* Object will have its own No of wheels, brand, Colour, and Price, which form **Properties or Attributes** of the *Vehicle* class
- The associated actions, like Start, Move, Stop, change gear, etc., form the functions or **Methods** of *Vehicles* Class.

Example

```
class Vehicle{
    int numberOfWheels;
    String brandName, color;
    double price;
    void start( );
    void changeGear( );
}
```

Vehicle car;

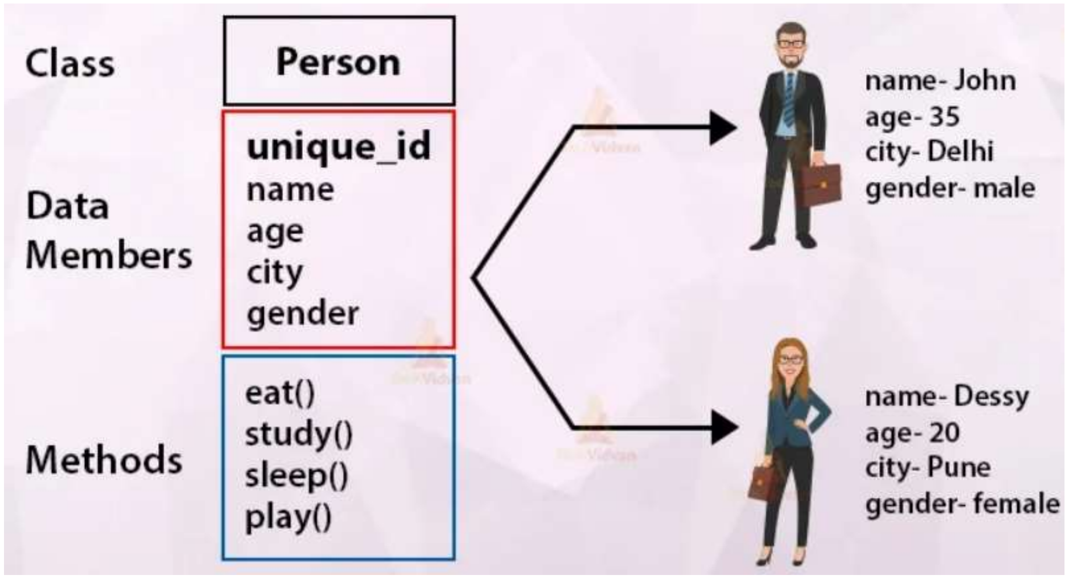
Vehicle bike;

Vehicle truck;

More Examples

```
int high_score;
int low_score;

Account frank_account;
Account jim_account;
```

high_score is an instance of int data type



Class and Object

- No memory is allocated when a class is created.
- Memory is allocated only when an object is created, i.e., when an instance of a class is created.

data type   variable;

int   x;

user defined data type   object_name;

Data Abstraction

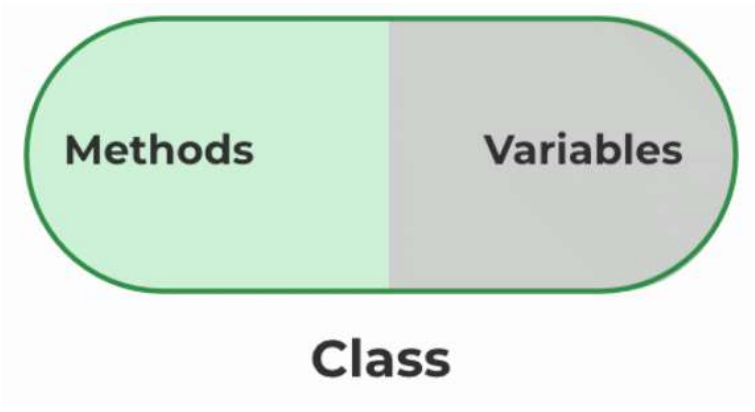
- Data abstraction refers to providing only essential information about the data to the outside world, ignoring unnecessary details or implementation.

- Consider a ***real-life example of a man driving a car***. The man only knows that pressing the accelerator will increase the speed of the car or applying brakes will stop the car, but he does not know how on pressing the accelerator the speed is increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc., in the car. This is what abstraction is.

- Data Abstraction increases the power of programming language by creating user-defined data types.
- Data Abstraction also represents the needed information in the program without presenting the details.

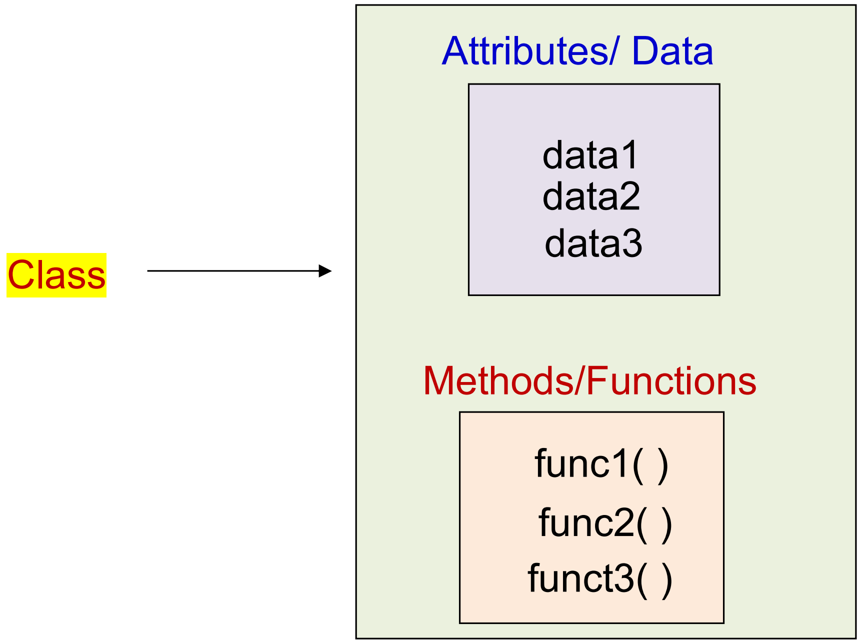
Data Encapsulation

- The property of C++, which allows the association of data and function into a single unit, is called encapsulation.



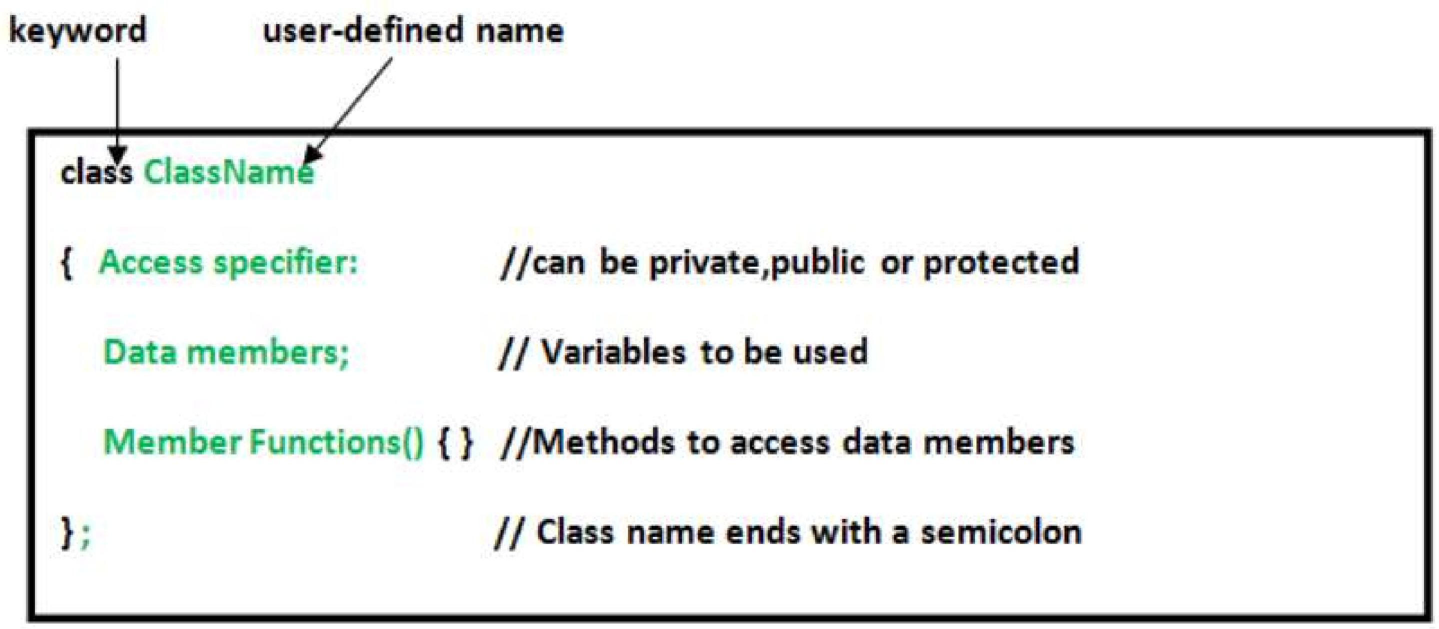
- When using Data Encapsulation, data is not accessed directly; it is only accessible through the functions present inside the class.
- Data Encapsulation enables the important concept of data hiding possible.
- Placing data and functions together in a single unit is the central theme of object-oriented programming.

Class Grouping Data and Functions



Declaring a Class

- When you create the definition of a class you are defining the attributes and behavior of a new type.
  - Attributes are data members.
  - Behavior is defined by methods/ functions.



```
class ClassName

{  Access specifier:          //can be private,public or protected

   Data members;              // Variables to be used

   Member Functions() { }     //Methods to access data members

};                            // Class name ends with a semicolon
```

Declaring Object

- When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.
- **Syntax to Create an Object**
- We can create an object of the given class in the same way we declare the variables of any other inbuilt data type.

```
ClassName ObjectName;
```

- Defining a class does not result in the creation of an object.

- Declaring a variable of a class type creates an object. You can have many variables of the same type (class).
- *This is called Instantiation.*

Example

```
class Account
{
   // attributes
   std::string name;
   double balance;

   // methods
   bool withdraw(double amount);
   bool deposit(double amount);
};
```

```
Account frank_account;
Account jim_account;

Account *mary_account = new Account~~()~~;
delete mary_account;
```

Example

```
class player {
// attributes
string name;
int age;
int run;
int wicket;

// methods
bool is_retired();
Bool is_for_auction();
}
```

```
player Dhoni;

player Virat;


player* rohit =new player();
delete rohit;
```

Accessing Class Members

- Syntax for accessing data members of a class.

  objectName .dataMember

- Syntax for accessing a member function of a class.

ObjectName . FunctionName (Actual Arguments)

$\uparrow$

member access specifier

Accessing Class Members

- We can access
  - class attributes
  - class methods

- Some class members will not be accessible (more on that later)

- We need an object to access instance variables

If we have an object (dot operator)

- **Using the dot operator**

```
Account frank_account;

frank_account.balance;
frank_account.deposit(1000.00);
```

Accessing Class Members

If we have a pointer to an object (member of pointer operator)

- Dereference the pointer then use the dot operator.

```
Account  *frank_account = new Account();

(*frank_account).balance;
(*frank_account).deposit(1000.00);
```

- Or use the member of pointer operator (arrow operator)

```
Account *frank_account = new Account();

frank_account->balance;
frank_account->deposit(1000.00);
```

Access Modifier

- All is very similar to the declaration on data structures, except that we can also now include functions and members, but also a new thing called *access specifier*.

- An access specifier is one of the following three keywords: private, public, or protected.
- These specifiers modify the access rights that the members following them acquire:
  - **private members of a class are accessible only from within other members of the same class or from their *friends*.**
  - protected members are accessible from members of their same class and from their friends, but also from members of their derived classes.
  - **public members are accessible from anywhere where the object is visible.**

Access Modifier

`public`, `private`, and `protected`

- `public`
  - accessible everywhere

- `private`
  - accessible only by members or friends of the class

- `protected`
  - used with inheritance – we'll talk about it in the next section

Access Modifier

- The members are usually grouped into two sections, private and public, which defines the visibility of members.
- A class which is totally private serves no useful purpose.

- private members
  - are accessible to their own class members

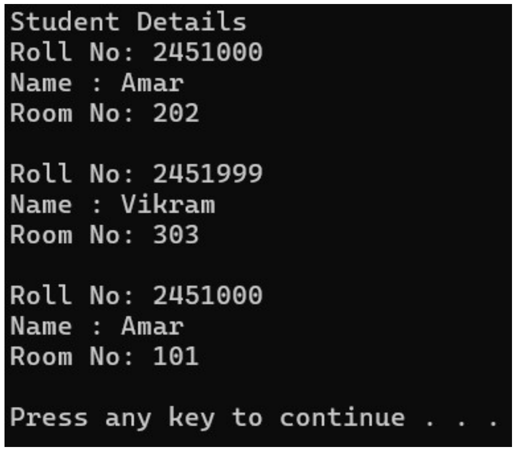
- public members
  - are not only accessible to their own members but also from outside the class.

Declaration of a class

```
using namespace std;
class student // declaring a class called student
{
private: ✓
int roll_no; // roll number
char name[50]; // name of a student

public:
int room_no;

// class methods or functions
void setdata( int roll_no_in, char *name_in, int room_in)
{                   2451000          xxx          101
roll_no= roll_no_in;
strcpy (name, name_in);//copies the string pointed by source to the destination.
room_no=room_in;
}
// display data members on the screen
void outdata ()
{
cout<<"Roll No: "<<roll_no<< endl;
cout<<"Name : "<< name<<endl;
cout<<"Room No: "<<room_no<< endl<<endl;
}
};
```

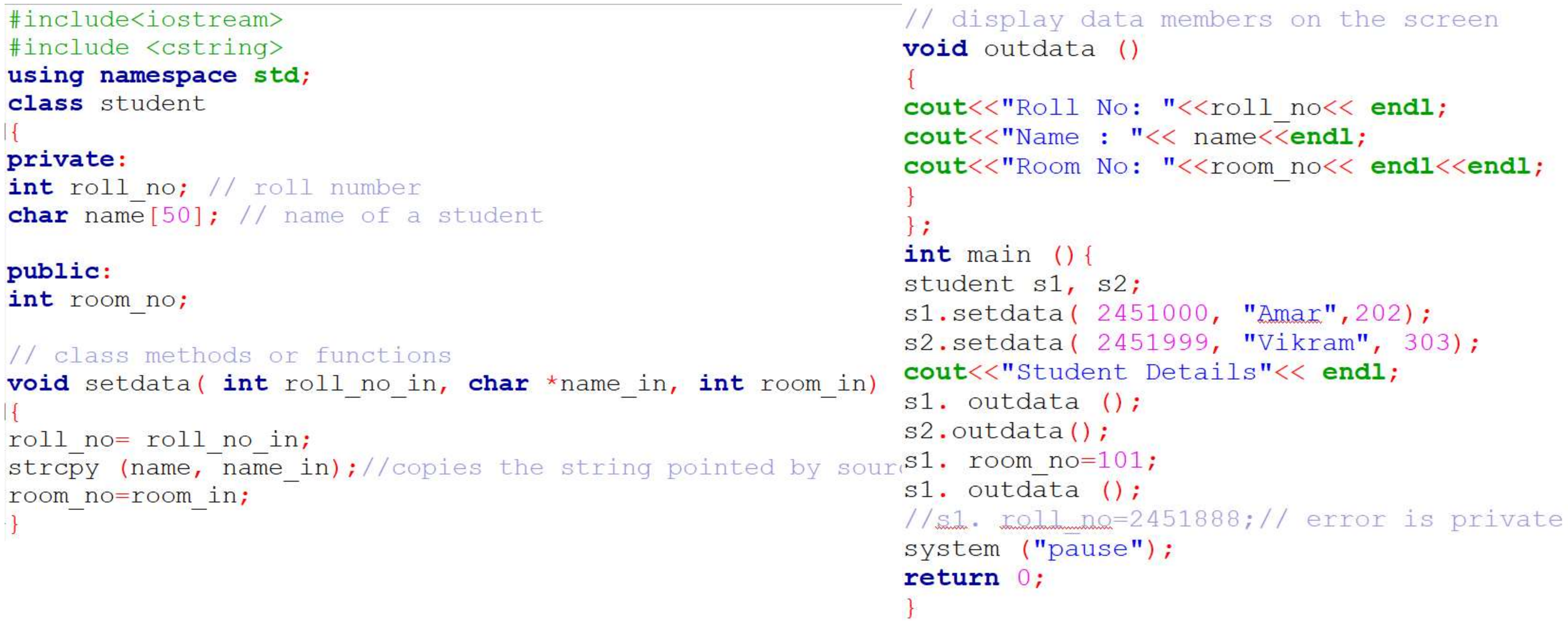
Declaration of Objects

```
int main (){
student s1, s2; // objects of class student
s1.setdata( 2451000, "Amar",202);
s2.setdata( 2451999, "Vikram", 303);
cout<<"Student Details"<< endl;
s1. outdata ();
s2.outdata();
s1. room_no=101;
s1. outdata ();
//s1. roll_no=2451888;// error is private
system ("pause");
return 0;
}
```



```
Student Details
Roll No: 2451000
Name : Amar
Room No: 202

Roll No: 2451999
Name : Vikram
Room No: 303

Roll No: 2451000
Name : Amar
Room No: 101

Press any key to continue . . .
```

Example: Access Modifier



```cpp
#include<iostream>
#include <cstring>
using namespace std;
class student
{
private:
int roll_no; // roll number
char name[50]; // name of a student

public:
int room_no;

// class methods or functions
void setdata( int roll_no_in, char *name_in, int room_in)
{
roll_no= roll_no_in;
strcpy (name, name_in);//copies the string pointed by sour
room_no=room_in;
}
```

```cpp
// display data members on the screen
void outdata ()
{
cout<<"Roll No: "<<roll_no<< endl;
cout<<"Name : "<< name<<endl;
cout<<"Room No: "<<room_no<< endl<<endl;
}
};
int main (){
student s1, s2;
s1.setdata( 2451000, "Amar",202);
s2.setdata( 2451999, "Vikram", 303);
cout<<"Student Details"<< endl;
s1. outdata ();
s2.outdata();
s1. room_no=101;
s1. outdata ();
//s1. roll_no=2451888;// error is private
system ("pause");
return 0;
}
```

Class Instantiation

- An example of class instantiation for creating objects is shown below, the keyword class is optional

```
class student s1, s2;
        or
student s1;
student s2;
student s[100];
```

- student s1, s2, s3 ,s4; // creates multiple objects of the class.
- The definition of an object is similar to that of a variable of any primitive data type.

Class Instantiation

- Objects can also be created by placing their names immediately after closing the braces like in the creation of the structure variable.

- Thus, the definition

```
 class  student
{
……….
……….
} s1 , s2 , s3 , s4;// creates objects s1 , s2 , s3  and s4 of the class student
```

Accessing Class Members

- Syntax for accessing data member of a class.

  objectName .dataMember

- Syntax for accessing
- member function of a class.

ObjectName . FunctionName (Actual Arguments)

$\uparrow$

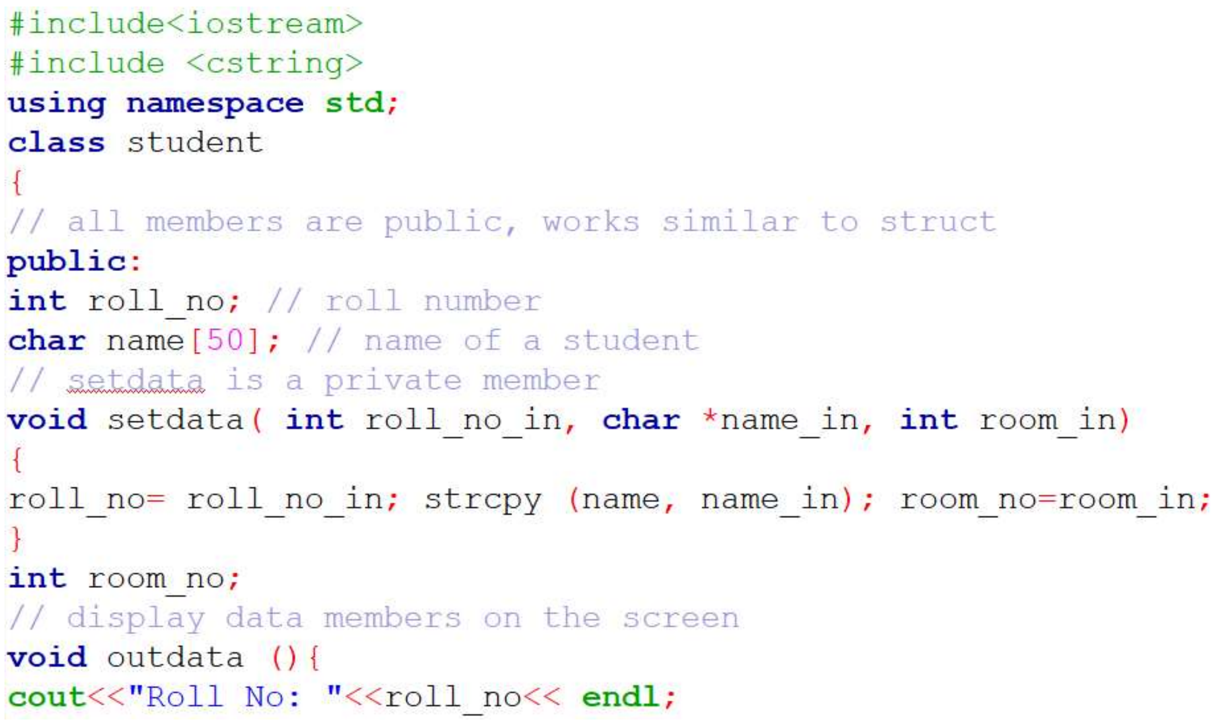
member access
specifier

Example: private member

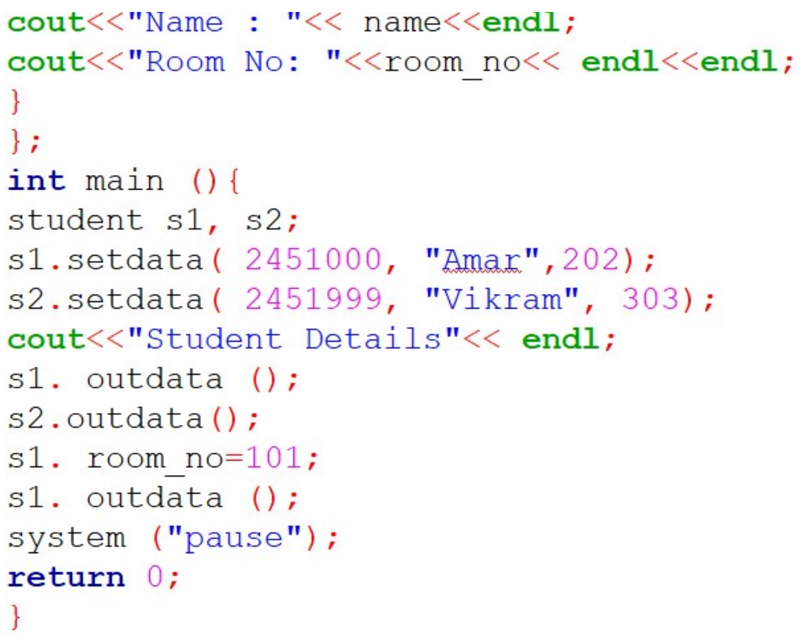
```
#include<iostream>
#include <cstring>
using namespace std;
class student
{
private:
int roll_no; // roll number
char name[50]; // name of a student
// setdata is a private member
void setdata( int roll_no_in, char *name_in, int room_in)
{
roll_no= roll_no_in; strcpy (name, name_in); room_no=room_in;
}
public:
int room_no;

// display data members on the screen
void outdata ()
{
cout<<"Roll No: "<<roll_no<< endl;
cout<<"Name : "<< name<<endl;
cout<<"Room No: "<<room_no<< endl<<endl;
}
};
int main (){
student s1, s2;
s1.setdata( 2451000, "Amar",202); // error
// cannot access private member
s2.setdata( 2451999, "Vikram", 303);// error
// cannot access private member
cout<<"Student Details"<< endl;
s1. outdata ();
s2.outdata();
s1. room_no=101;
s1. outdata ();
//s1. roll_no=2451888;// error is private
system ("pause");
return 0;
}
```

Example

- if all members are public, then works similar to struct



```cpp
#include<iostream>
#include <cstring>
using namespace std;
class student
{
// all members are public, works similar to struct
public:
int roll_no; // roll number
char name[50]; // name of a student
// setdata is a private member
void setdata( int roll_no_in, char *name_in, int room_in)
{
roll_no= roll_no_in; strcpy (name, name_in); room_no=room_in;
}
int room_no;
// display data members on the screen
void outdata (){
cout<<"Roll No: "<<roll_no<< endl;
```



```cpp
cout<<"Name : "<< name<<endl;
cout<<"Room No: "<<room_no<< endl<<endl;
}
};
int main (){
student s1, s2;
s1.setdata( 2451000, "Amar",202);
s2.setdata( 2451999, "Vikram", 303);
cout<<"Student Details"<< endl;
s1. outdata ();
s2.outdata();
s1. room_no=101;
s1. outdata ();
system ("pause");
return 0;
}
```

Default access

- By default, all members of a class declared with the class keyword have private access for all its members.
- Therefore, any member that is declared before one other class specifier automatically has private access.

- For example:

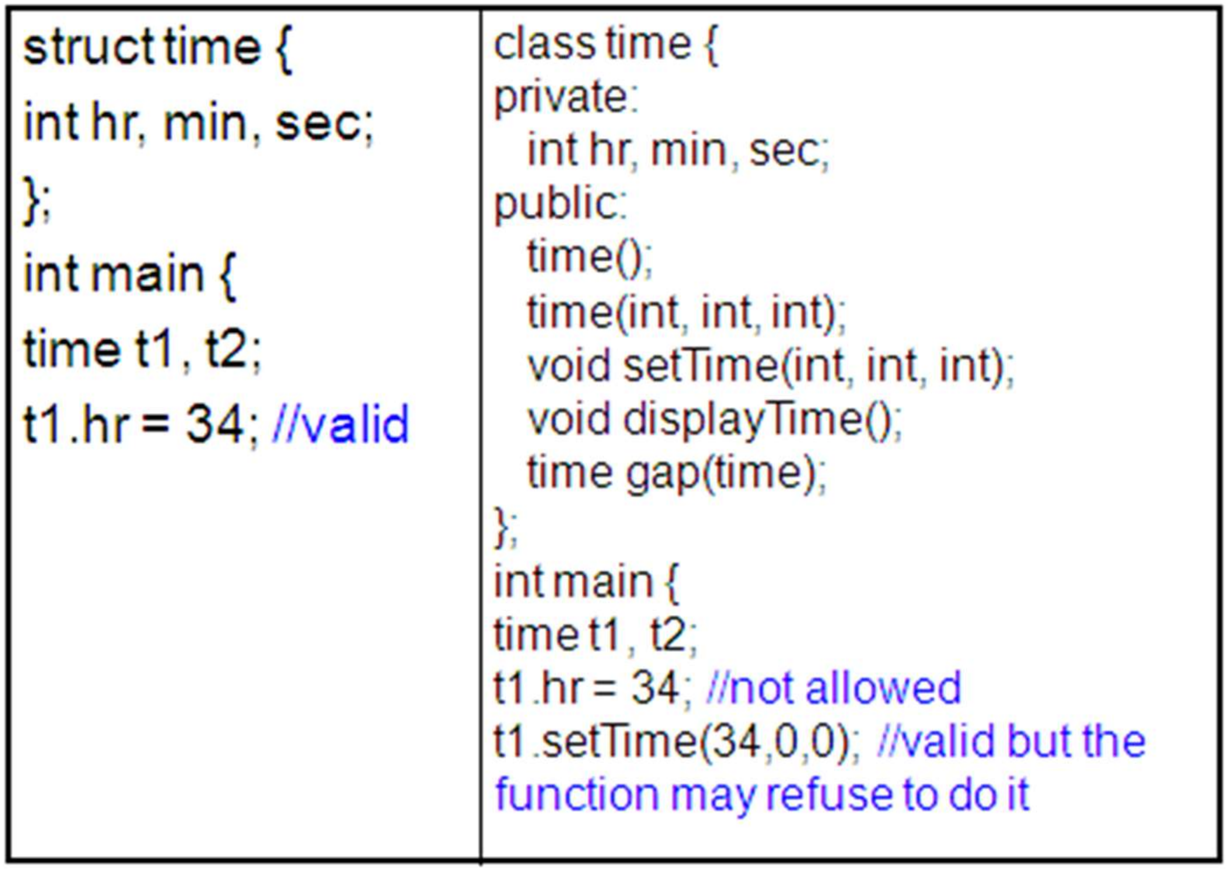
```
class Employee {
 int age, emp_code; //
 public:
        void setvalues (int,int);
        void disp ();
} e1;
```

Example: All members private

```
#include <cstring>
using namespace std;
class student
{
// all members are private by default
int roll_no; // roll number
char name[50]; // name of a student
// setdata is a private member
void setdata( int roll_no_in, char *name_in, int room_in)
{
roll_no= roll_no_in; strcpy (name, name_in); room_no=room_in;
}
int room_no;
// display data members on the screen
void outdata (){
cout<<"Roll No: "<<roll_no<< endl;
cout<<"Name : "<< name<<endl;
cout<<"Room No: "<<room_no<< endl<<endl;
}
};
```

```
int main (){
student s1, s2;
//s1.setdata( 2451000, "Amar",202);
//s2.setdata( 2451999, "Vikram", 303);
cout<<"Student Details"<< endl;
//s1. outdata ();
//s2.outdata();
//s1. room_no=101;
//s1. outdata ();
// All the commented out statements cause error
// as they are private members and cannon be accessed
system ("pause");
return 0;
}
```

Struct vs. Class



```
struct time {
int hr, min, sec;
};
int main {
time t1, t2;
t1.hr = 34; //valid
```

```
class time {
private:
  int hr, min, sec;
public:
  time();
  time(int, int, int);
  void setTime(int, int, int);
  void displayTime();
  time gap(time);
};
int main {
time t1, t2;
t1.hr = 34; //not allowed
t1.setTime(34,0,0); //valid but the
function may refuse to do it
```

Summary

- *Class* is an expanded concept of a data structure:
  - **instead of holding only data, it can hold both data and functions.**
- An *object* is an instantiation of a class.
- In terms of variables, a class would be the type, and an object would be the variable.

Classes are generally declared using the keyword **class,** with the following format:

```
class class_name {
        access_specifier_1:
member1;
        access_specifier_2:
member2; ...
} object_names;
```

•class_name is a valid identifier for the class
•object_names is an optional list of names for objects of this class

The body of the declaration can contain members, that can be either data or function declarations, and optionally access specifiers.

Defining Member Functions

- Member functions can be defined in two places
  - Inside the class definition
  - Outside the class definition

- Normally only small functions are defined inside the class definition.

Defining Member Functions inside Class definition

```
#include<iostream>
#include <cstring>
using namespace std;
class student
{
private:
int roll_no; // roll number
char name[50]; // name of a student
// setdata is a private member
void setdata( int roll_no_in, char *name_in, int room_in)
{
roll_no= roll_no_in; strcpy (name, name_in); room_no=room_in;
}
public:
int room_no;

// display data members on the screen
void outdata ()
{
cout<<"Roll No: "<<roll_no<< endl;
cout<<"Name : "<< name<<endl;
cout<<"Room No: "<<room_no<< endl<<endl;
}
};
int main (){
student s1, s2;
s1.setdata( 2451000, "Amar",202); // error
// cannot access private member
s2.setdata( 2451999, "Vikram", 303);// error
// cannot access private member
cout<<"Student Details"<< endl;
s1. outdata ();
s2.outdata();
s1. room_no=101;
s1. outdata ();
//s1. roll_no=2451888;// error is private
system ("pause");
return 0;
}
```

Defining Member Function Outside the Class Definition

- The general form is

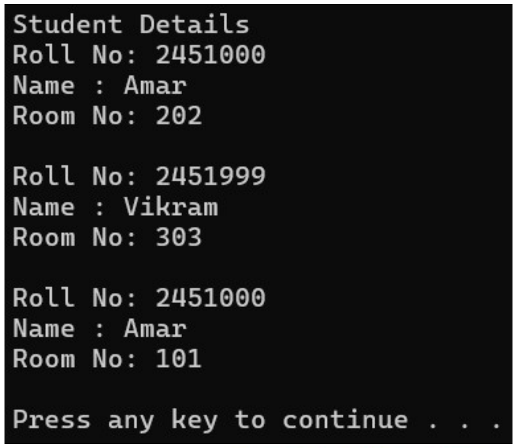
```
return_type class-name:: function_name (argument declaration)
    {
            Function body
    }
```

- However, a function prototype is to be defined inside the class

Defining Member function outside Class definition

```cpp
#include<iostream>
#include <cstring>
using namespace std;
class student
{

int roll_no; // roll number
char name[50]; // name of a student

public:
int room_no;
// display data members on the screen
// only function prototype is placed inside
// the class definition
void setdata( int , char *, int );
void outdata ();
};
int main (){
student s1, s2;
s1.setdata( 2451000, "Amar",202);
s2.setdata( 2451999, "Vikram", 303);
cout<<"Student Details"<< endl;
s1. outdata ();
s2.outdata();
system ("pause");
return 0;
}
```

```cpp
// Function definitions
//class name is to be included
void student::setdata( int roll_no_in, char *name_in, int room_in)
{
roll_no= roll_no_in; strcpy (name, name_in); room_no=room_in;
}

void student::outdata (){
cout<<"Roll No: "<<roll_no<< endl;
cout<<"Name : "<< name<<endl;
cout<<"Room No: "<<room_no<< endl<<endl;
}
```



```
Student Details
Roll No: 2451000
Name : Amar
Room No: 202

Roll No: 2451999
Name : Vikram
Room No: 303

Roll No: 2451000
Name : Amar
Room No: 101

Press any key to continue . . .
```

Types of Member Functions

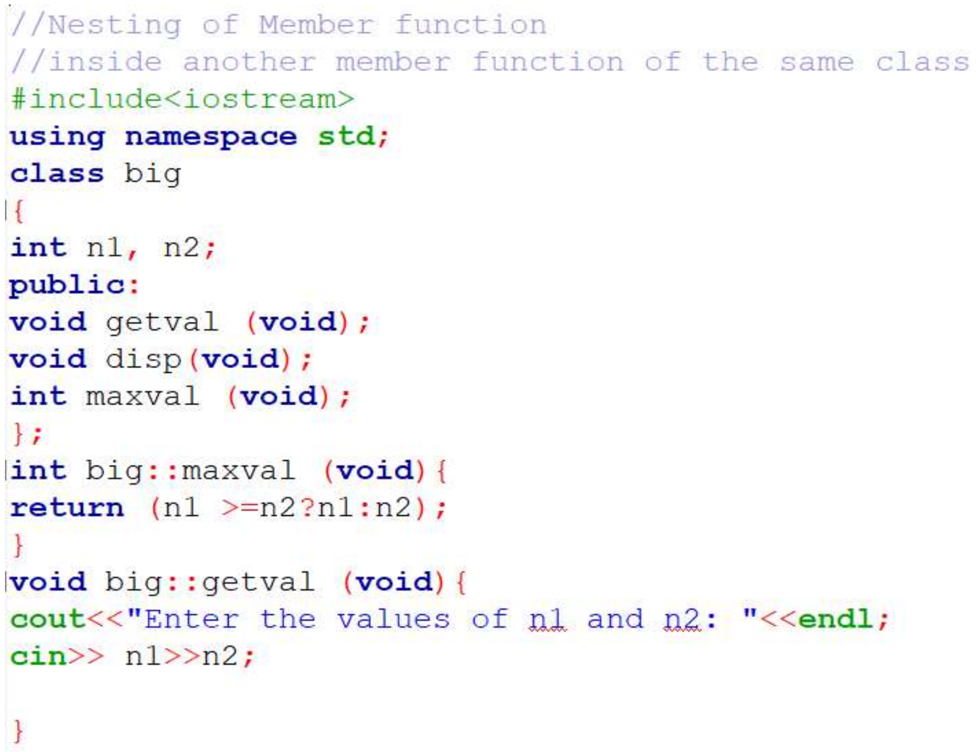
- There are different types of member functions that can be declared inside the class are as follows:
    1. Nested Member Function
    2. Constant Member Function
    3. Inline function
    4. Friend function

Nesting of Member Functions

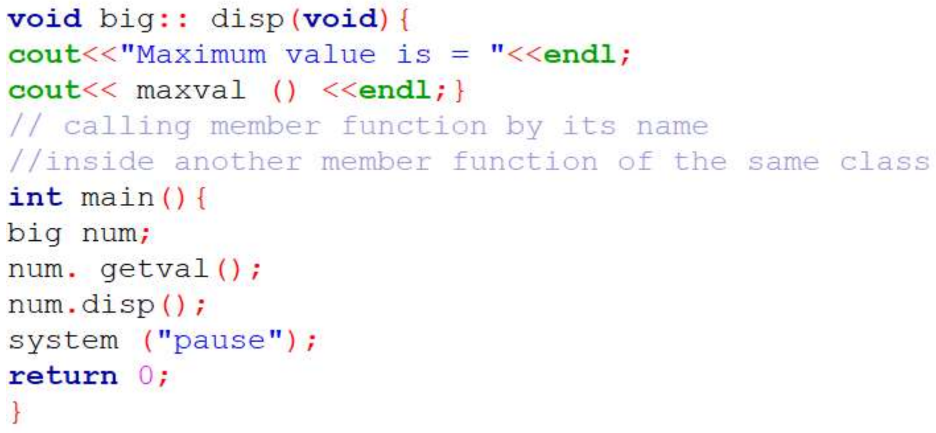
- The public member functions of a class can be invoked only by using an object of that class using a (.) operator.
- However ,a member function of the same class can call another member function of the same class using its name. **We do not need an object to call them.**

" When a member function is called by another member function it is called nesting of member function."

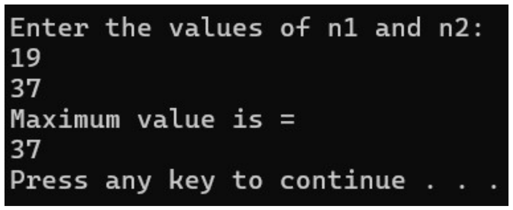
Example:Nesting of Member functions



```
//Nesting of Member function
//inside another member function of the same class
#include<iostream>
using namespace std;
class big
{
int n1, n2;
public:
void getval  (void);
void disp(void);
int maxval  (void);
};
int big::maxval  (void){
return  (n1 >=n2?n1:n2);
}
void big::getval  (void){
cout<<"Enter the values of n1 and n2: "<<endl;
cin>> n1>>n2;

}
```



```
void big:: disp(void){
cout<<"Maximum value is = "<<endl;
cout<< maxval  ()  <<endl;}
// calling member function by its name
//inside another member function of the same class
int main(){
big num;
num. getval();
num.disp();
system ("pause");
return 0;
}
```



```
Enter the values of n1 and n2:
19
37
Maximum value is =
37
Press any key to continue . . .
```

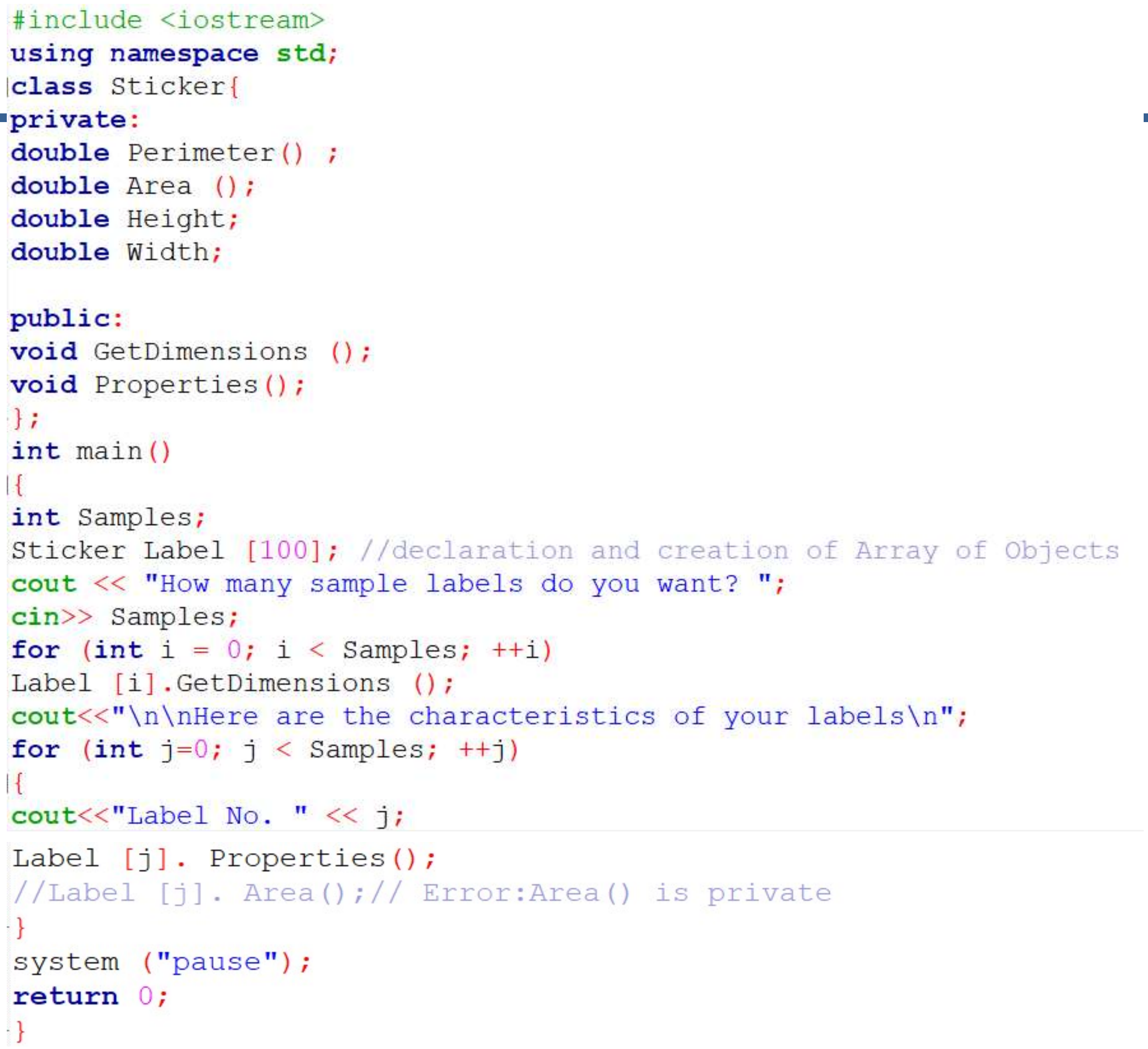
Private Member Function

- A private member function can only be called by another function that is a member of its class
- Even an object can not invoke a private function usi.ng the dot operator.

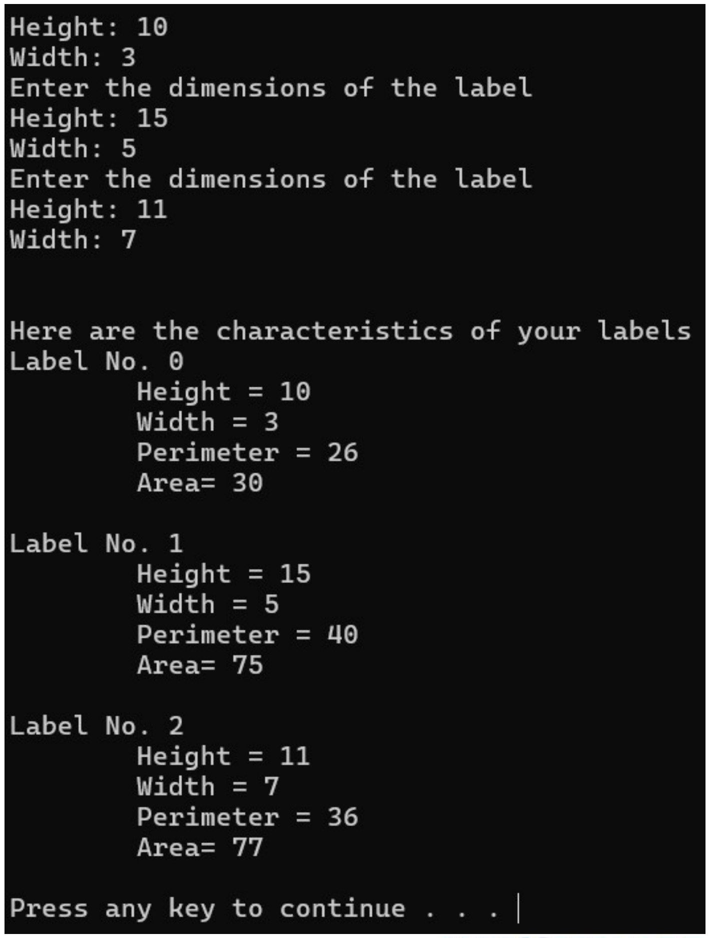
Example

```
#include <iostream>
using namespace std;
class Sticker{
private:
double Perimeter() ;
double Area ();
double Height;
double Width;

public:
void GetDimensions ();
void Properties();
};
int main()
{
int Samples;
Sticker Label [100]; //declaration and creation of Array of Objects
cout << "How many sample labels do you want? ";
cin>> Samples;
for (int i = 0; i < Samples; ++i)
Label [i].GetDimensions ();
cout<<"\n\nHere are the characteristics of your labels\n";
for (int j=0; j < Samples; ++j)
{
cout<<"Label No. " << j;
Label [j]. Properties();
//Label [j]. Area();// Error:Area() is private
}
system ("pause");
return 0;
void Sticker::GetDimensions ()
{
cout<<"Enter the dimensions of the label\n";
cout<<"Height: ";
cin>> Height;
cout << "Width: "; cin>> Width;
}
void Sticker::Properties(){
cout<<"\n\tHeight = "<<Height;
cout<<"\n\tWidth = "<< Width;
cout << "\n\tPerimeter = "<< Perimeter();
cout << "\n\tArea= " << Area();
// private member function can be called by
//another function that is a member of its class
cout<<"\n\n";
}
double Sticker:: Perimeter (){
return 2*(Height+Width);
}
double Sticker::Area(){
return Height*Width;
}
```

Contd…



```cpp
#include <iostream>
using namespace std;
class Sticker{
private:
double Perimeter() ;
double Area ();
double Height;
double Width;

public:
void GetDimensions ();
void Properties();
};
int main()
{
int Samples;
Sticker Label [100]; //declaration and creation of Array of Objects
cout << "How many sample labels do you want? ";
cin>> Samples;
for (int i = 0; i < Samples; ++i)
Label [i].GetDimensions ();
cout<<"\n\nHere are the characteristics of your labels\n";
for (int j=0; j < Samples; ++j)
{
cout<<"Label No. " << j;
Label [j]. Properties();
//Label [j]. Area();// Error:Area() is private
}
system ("pause");
return 0;
}
```



```
Height: 10
Width: 3
Enter the dimensions of the label
Height: 15
Width: 5
Enter the dimensions of the label
Height: 11
Width: 7


Here are the characteristics of your labels
Label No. 0
        Height = 10
        Width = 3
        Perimeter = 26
        Area= 30

Label No. 1
        Height = 15
        Width = 5
        Perimeter = 40
        Area= 75

Label No. 2
        Height = 11
        Width = 7
        Perimeter = 36
        Area= 77

Press any key to continue . . .
```

Constant Member Function

If a member function of a class does not alter any data in the class, then this member function may be declared as a CONSTANT member function using the keyword const.

- For example:
  int max_num (int,int) const;
  void print() const;
- The qualifier const appears both in member function declaration and definition.
- Once a member function is declared as **const**, it cannot alter the data values of the class.
- The compiler will generate an error message if such functions try to alter the data values.

Example:Constant Member Function

```cpp
#include <iostream>
using namespace std;
class Sticker{
private:
double Perimeter () const;
double Area ()const;
double Height;
double Width;

public:
void GetDimensions ();
void Properties ();
};
int main()
{
Sticker Label; //declaration  of Objects
Label.GetDimensions ();
cout<<"\n\nHere are the characteristics of your labels\
Label . Properties();
system ("pause");
return 0;
}
```

```cpp
void Sticker::GetDimensions ()
{
cout<<"Enter the dimensions of the label\n";
cout<<"Height: ";
cin>> Height;
cout << "Width: "; cin>> Width;
}
void Sticker::Properties(){
cout<<"\n\tHeight = "<<Height;
cout<<"\n\tWidth = "<< Width;
cout << "\n\tPerimeter = "<< Perimeter();
cout << "\n\tArea= " << Area();
// private member function can be called by
//another function that is a member of its class
cout<<"\n\n";
}
double Sticker:: Perimeter ()const{
//Height=Height*2;
// error: constant function cannot change the data value
return 2*(Height+Width);
}
double Sticker::Area()const{
//Width= Width+2;
// error: constant function cannot change the data value
return Height*Width;
}
```

Inline Function

The inline functions are a C++ enhancement designed to speed up programs.

The coding part of all the inline functions and a normal function is not different.

It is just the same except for a minor change, or one can say an addition is that an inline function starts with a keyword inline.

Example

```
//in-line function
#include<iostream>
using namespace std;
class big
{
int n1, n2;
public:
void getval (void);
void disp(void);
int maxval (void);
};
int big::maxval (void){
return (n1 >=n2?n1:n2);
}
void big::getval (void){
cout<<"Enter the values of n1 and n2: "<<endl;
cin>> n1>>n2;}

inline void big:: disp(void){
cout<<"Maximum value is = "<<endl;
cout<< maxval () <<endl;}

int main(){
big num;
num. getval();
num.disp();
system ("pause");
return 0;
}
```