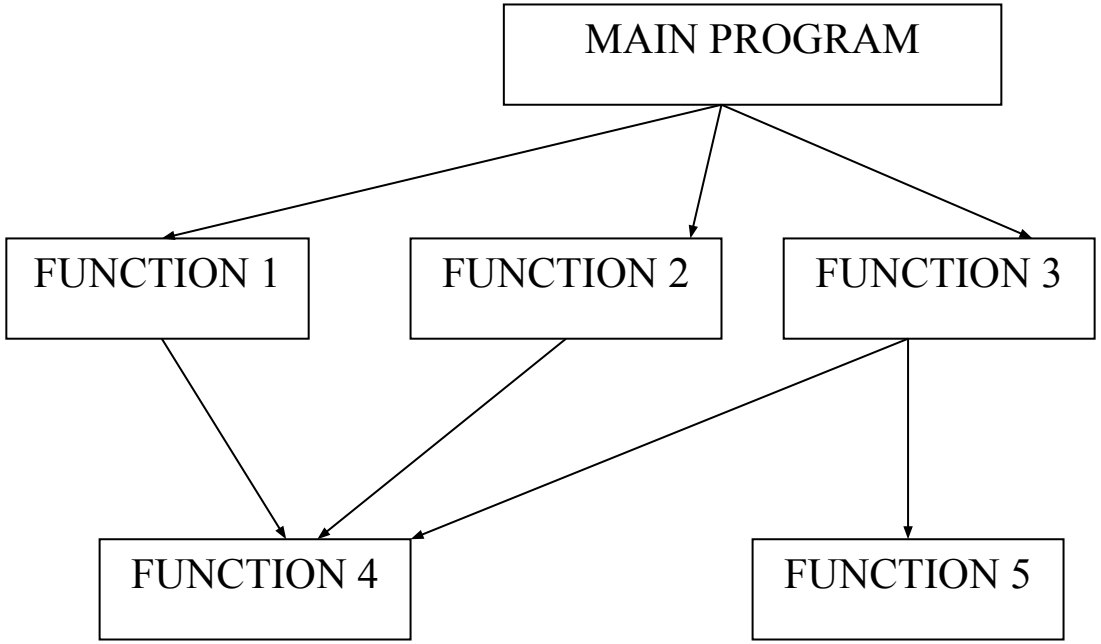
PROCEDURE-ORIENTED (STRUCTURED) PROGRAMMING

VS

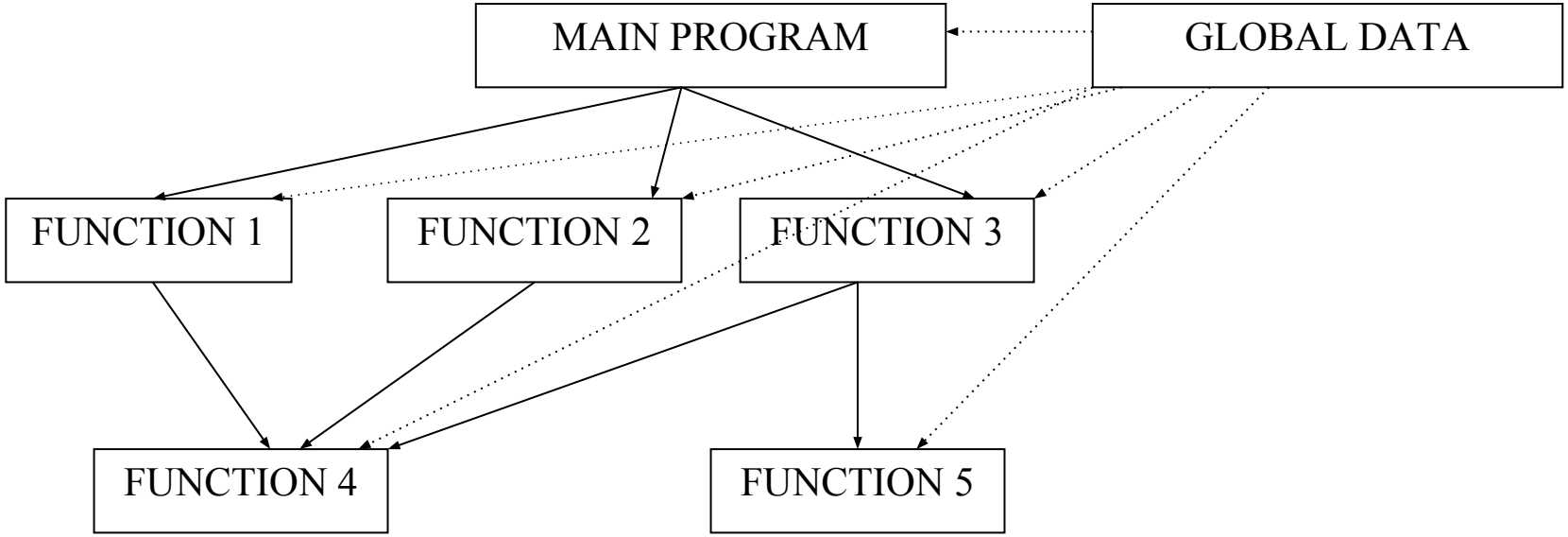
OBJECT-ORIENTED PROGRAMMING

Procedure-Oriented programming

- A problem is viewed as a sequence of tasks.
- A number of functions are written to accomplish these tasks.
- COBOL, FORTRAN, C etc.



Procedure-Oriented programming



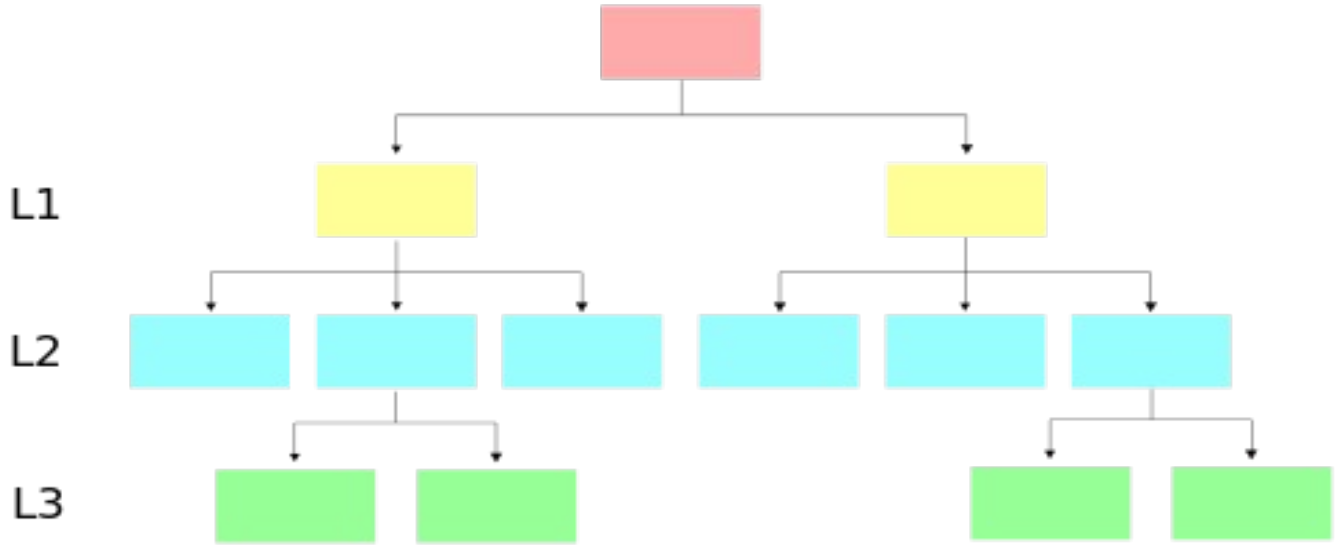
- Functions operate on data.
- Primary focus is on functions.
- Drawbacks:
  - No data security for global data.
  - Does not model real world problems very well.

Characteristics of Procedure-Oriented programming

- Emphasis is on doing things (algorithms).
- Large programs are divided into functions.
- Most of the functions share global data.
- Data move openly around the system from function to function.
- Employ top-down approach in program design.

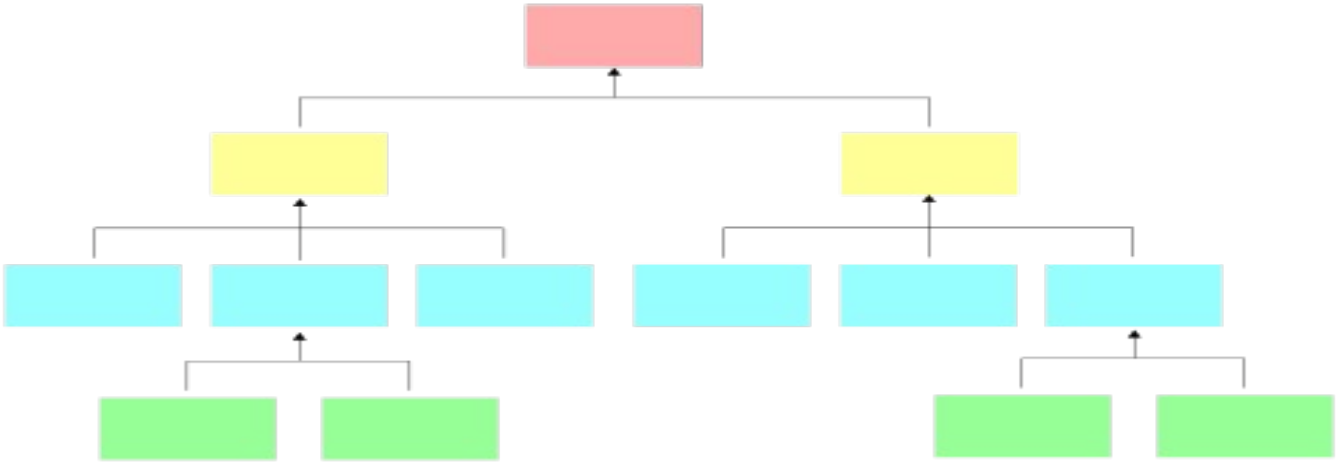
Top-down approach

- A single module will be split into several smaller modules.
- General to specific.



Bottom-up approach

- Lot of small modules will be grouped to form a single large module.
- Specific to general.



Real world problems

Sheldon

Attributes:

- Name
- Age
- Height
- …..

Functions:

- Walking
- Driving
- ……

Real world problems

Sheldon

Attributes:
- Name
- Age
- Height
- …..

Functions:
- Walking
- Driving
- ……

Sedan

Attributes:
- Name
- Fuel
- speed
- …..

Functions:
- setFule
- setSpeed
- beDriven
- ……

Real world problems

Sheldon

Attributes:
- Name
- Age
- Height
- …..

Functions:
- Walking
- Driving
- ……

Sedan

Attributes:
Name
Fuel
speed
…..

Functions:
setFule
setSpeed
- beDriven
- ……

Sheldon is driving his **Sedan**

Real world problems

Sheldon is driving his **Sedan** from A to B

Real world problems

Class: person

Real world problems

Class: person

Attributes:

- Name
- Age
- Height
- …..

Functions:

- Walking
- Driving
- ……

Class: Blueprint of object

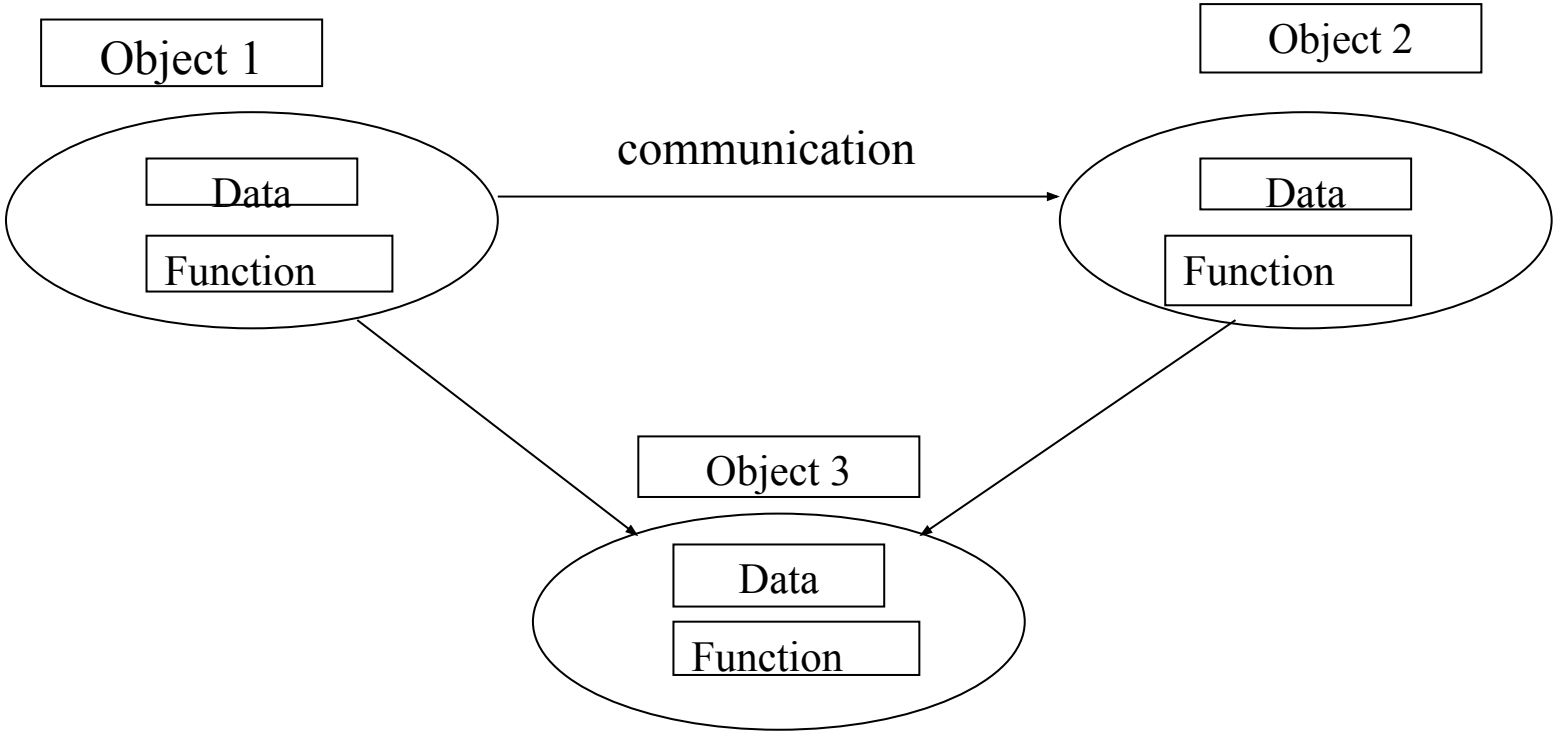
Real world problems

Object: Sheldon

Object: Amy

Class: person

Object-Oriented programming



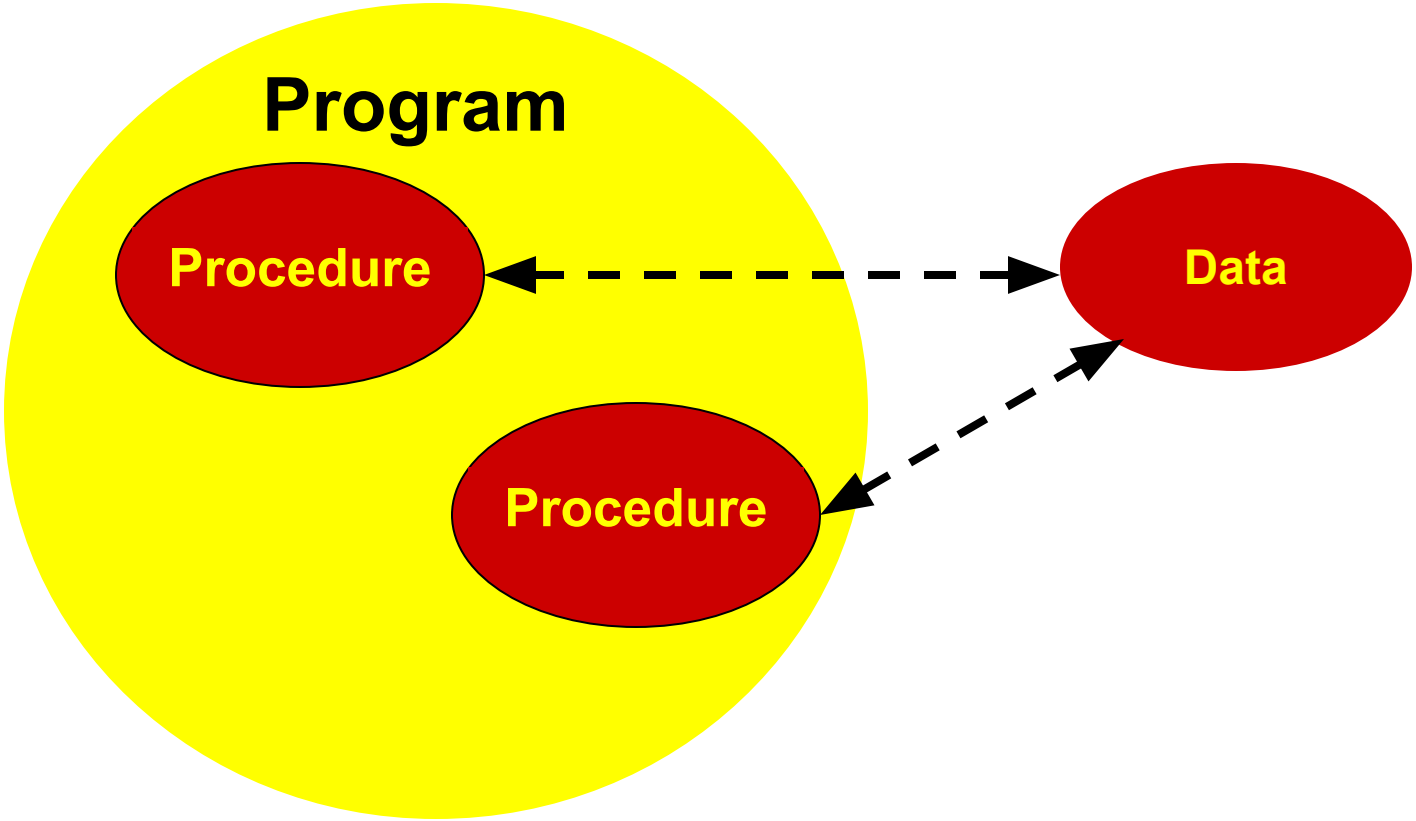
- Allows decomposition of a problem into a number of entities called ***object***.
- Builds data and functions around these objects.
- Data of an object can be accessed only by the function associated with the object and protects it from accidental modification from outside functions.
- Function of one object can communication with the function of other object.
- C++, Java etc.

Characteristics of Object-Oriented Programming

- Emphasis is on data rather than procedure.
- Programs are divided into *objects*.
- Data is hidden and can not be accessed by external functions.
- Object may communicate with each other through functions.
- Follows bottom-up approach.

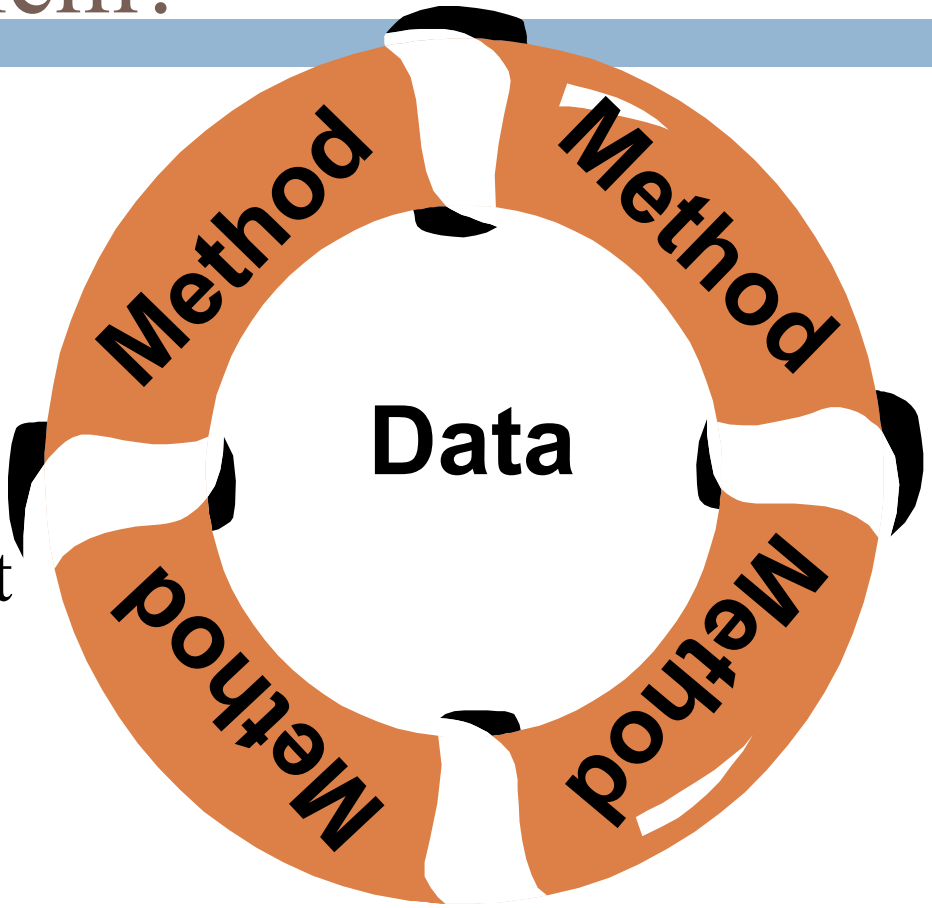
How Procedure-oriented Programming Looks at a Problem

- Procedures/functions
- Procedures act on data



How Object-Oriented Programming Looks at a Problem?

- Objects
  - Data
  - Methods (functions)
- Methods surround the data and protect data from other objects



OBJECT

Basic concepts of Object-Oriented Programming

- Objects.
- Classes.
- Encapsulation and data abstraction.
- Inheritance.
- Polymorphism.
- Dynamic binding.
- Message passing.

A simple program

- Input: name and age of a person.
- Output: display name and age of that person.

A simple program

```
#include<iostream.h>
class person
{
        char name[30];
        int age;

        public:
          void getdata(void);
          void putdata(void);

};
```

keyword

Attributes
By default “private”

Functions/methods

A simple program

```
#include<iostream.h>
class person
{
        char name[30];
        int age;

        public:
          void getdata(void);
          void putdata(void);

};

void person :: getdata(void)
{
        cout << "Enter name:";
        cin >> name;
        cout << "Enter age:";
        cin >> age;
}
```

Header file

Scope resolution operator

Insertion or *put to* operator/ bit-wise left-shift

Extraction or *get from* operator

A simple program

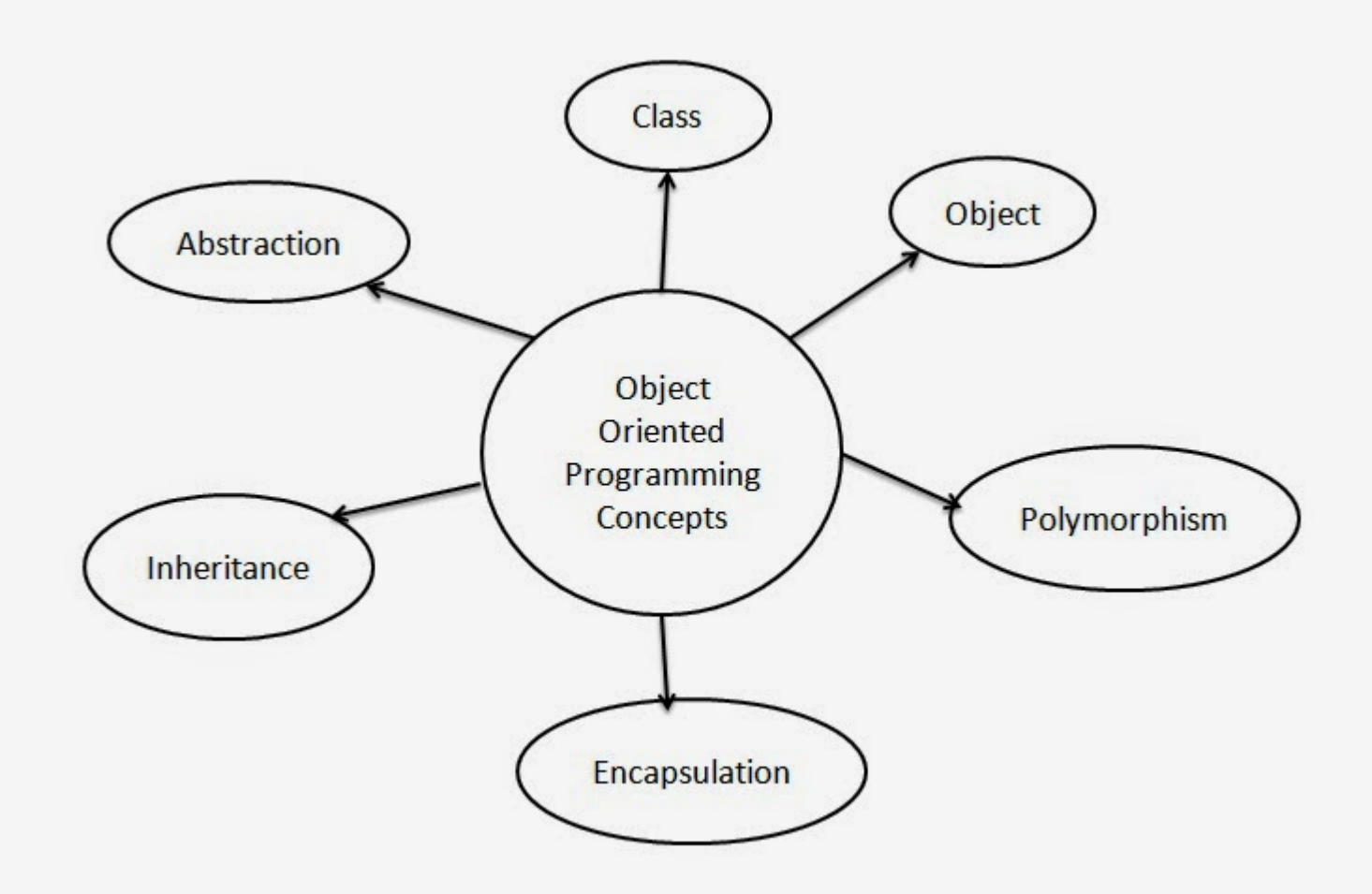
```
#include<iostream.h>
class person
{
        char name[30];
        int age;

        public:
          void getdata(void);
          void putdata(void);

};

void person :: getdata(void)
{
        cout << "Enter name:";
        cin >> name;
        cout << "Enter age:";
        cin >> age;
}

void person :: putdata(void)
{
        cout << "\nName: " << name;
        cout << "\nage: " << age;

}
```

A simple program

```
#include<iostream.h>
class person
{
        char name[30];
        int age;

        public:
          void getdata(void);
          void putdata(void);

};

void person :: getdata(void)
{
        cout << "Enter name:";
        cin >> name;
        cout << "Enter age:";
        cin >> age;
}

void person :: putdata(void)
{
        cout << "\nName: " << name;
        cout << "\nage: " << age;

}

void main()
{
        person p;

        p.getdata();
        p.putdata();


}
```

dot operator

Basic concepts of Object-Oriented Programming

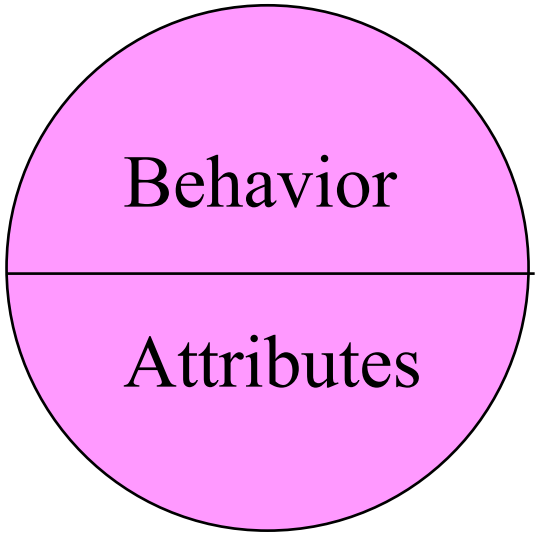


Objects

- Can be **concrete** and **tangible** entities.
  - Sheldon, amy, book, car, laptop etc..
- Can be **abstract** entities and do not have to be tangible.
  - Database, email, webpage, song etc..
- An object can contain other objects
  - House = kitchen + bedrooms + …
  - Laptop = keyboard + display + processor + …

Objects' three properties

- **Unique identity** – name, serial number, relationship with another object ..
- **Set of attributes** – location, speed, size, name, phone number …
- **Behaviors (action)**– walking, driving, take picture, send email …

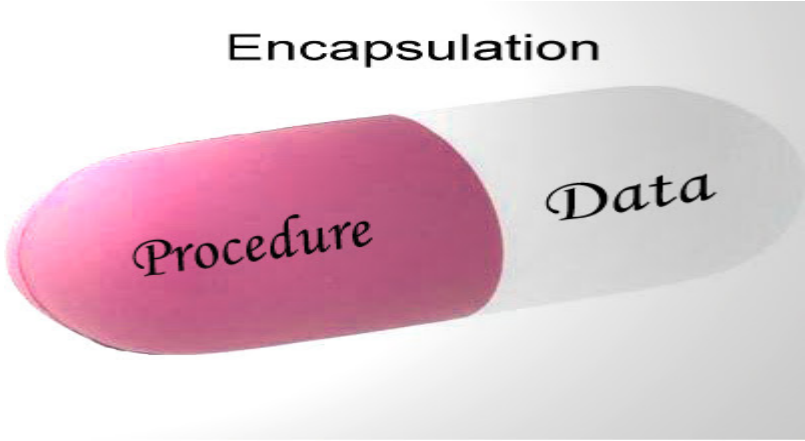


classes

- Describe the **commonalities** of similar objects.
  - Person : sheldon, leonard, amy, panny ..
  - Car : sedan, ford, toyota ...
  - Classroom : CSE 101, CSE 102 …
  - Building: CSE, EEE, CE, ME …
- **Blueprint** of object.
- Describe both the attributes and the behaviors
  - Person: name, age .. + sleep, walk ..
- **User-defined** data types and behave like the built-in types.
  - Fruit mango;

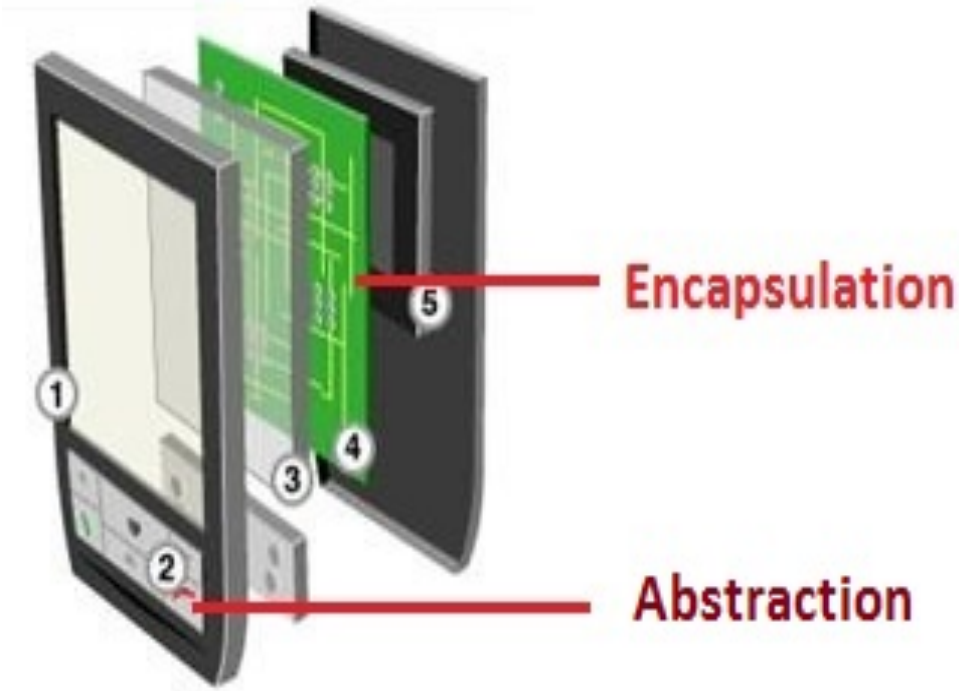
Encapsulation

- The wrapping up of data and functions into a single unit is known as **encapsulation**.
- Data is not accessible to the outside world.
- Only those functions which are wrapped in the class can access it.
- This insulation of the data from direct access by the program is known as '**data hiding**' or '**information hiding**'.



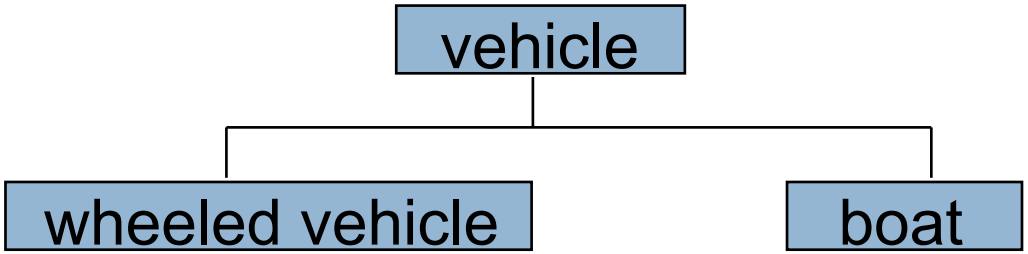
Abstraction

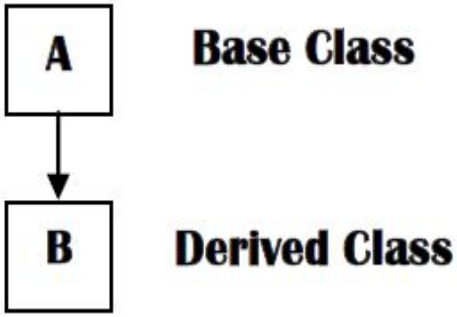
- Representing **essential features** without including the background details or explanations.



Inheritance

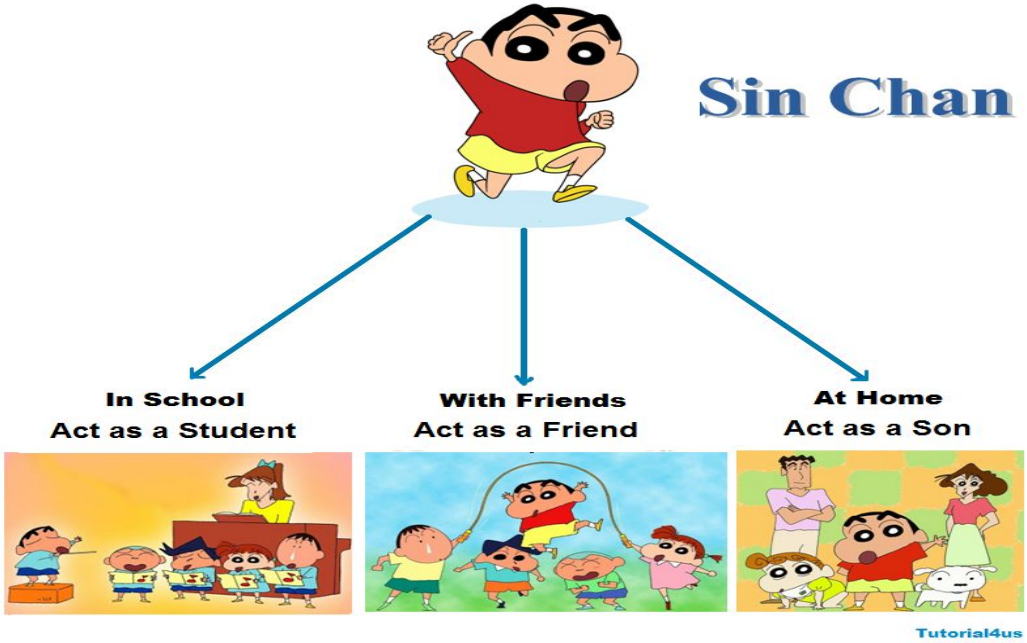
- A process by which objects of one class acquire the properties of objects of another class.
- Provides the idea of **reusability**.
- Inherited class is called **parent/base/super** class.
- Class that inherits parent class is known as **child/derived/sub** class.



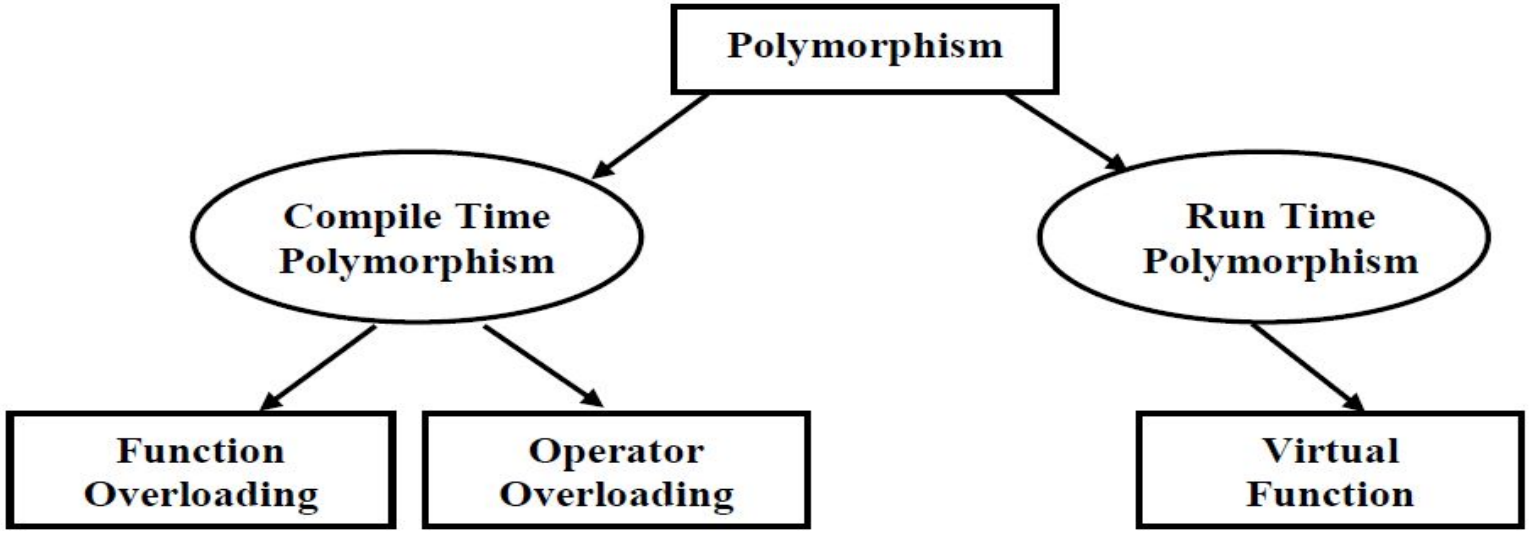


Polymorphism

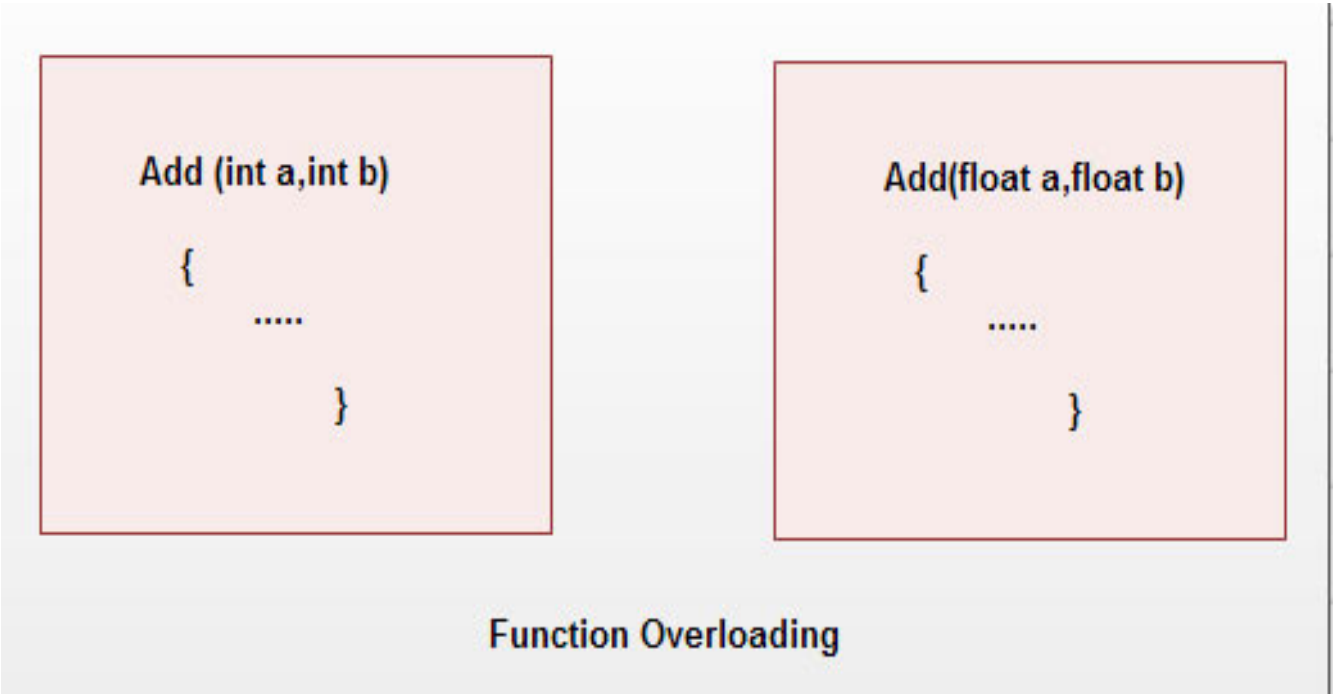
- Poly - **many** and morph – **form**(behavior).
- An operation may exhibit different behavior in different instances.



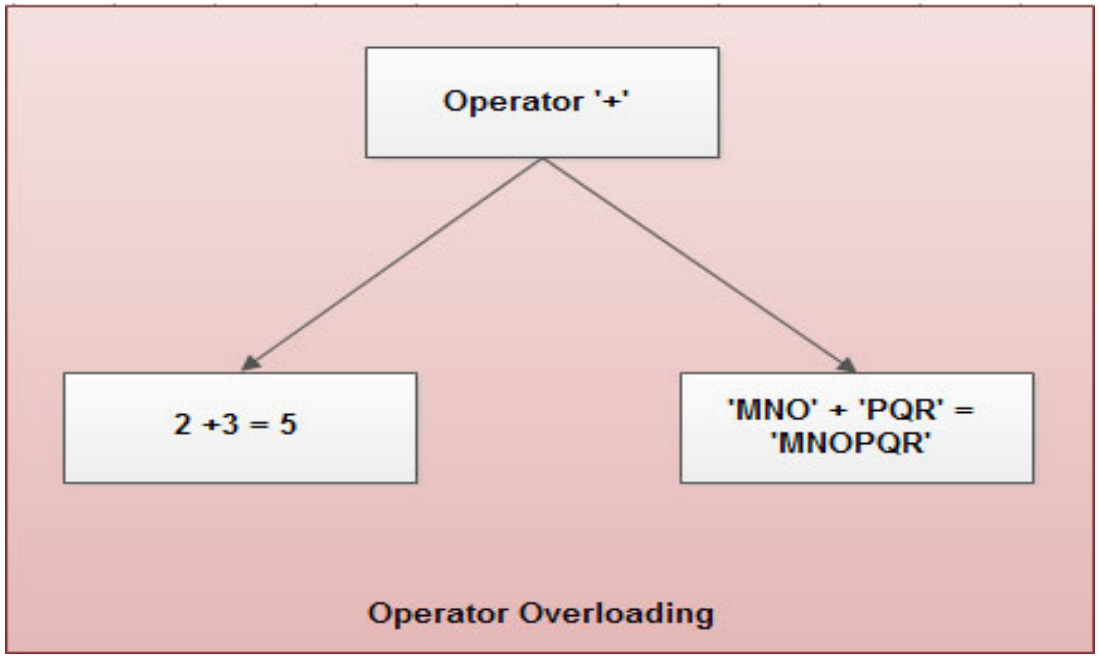
Polymorphism



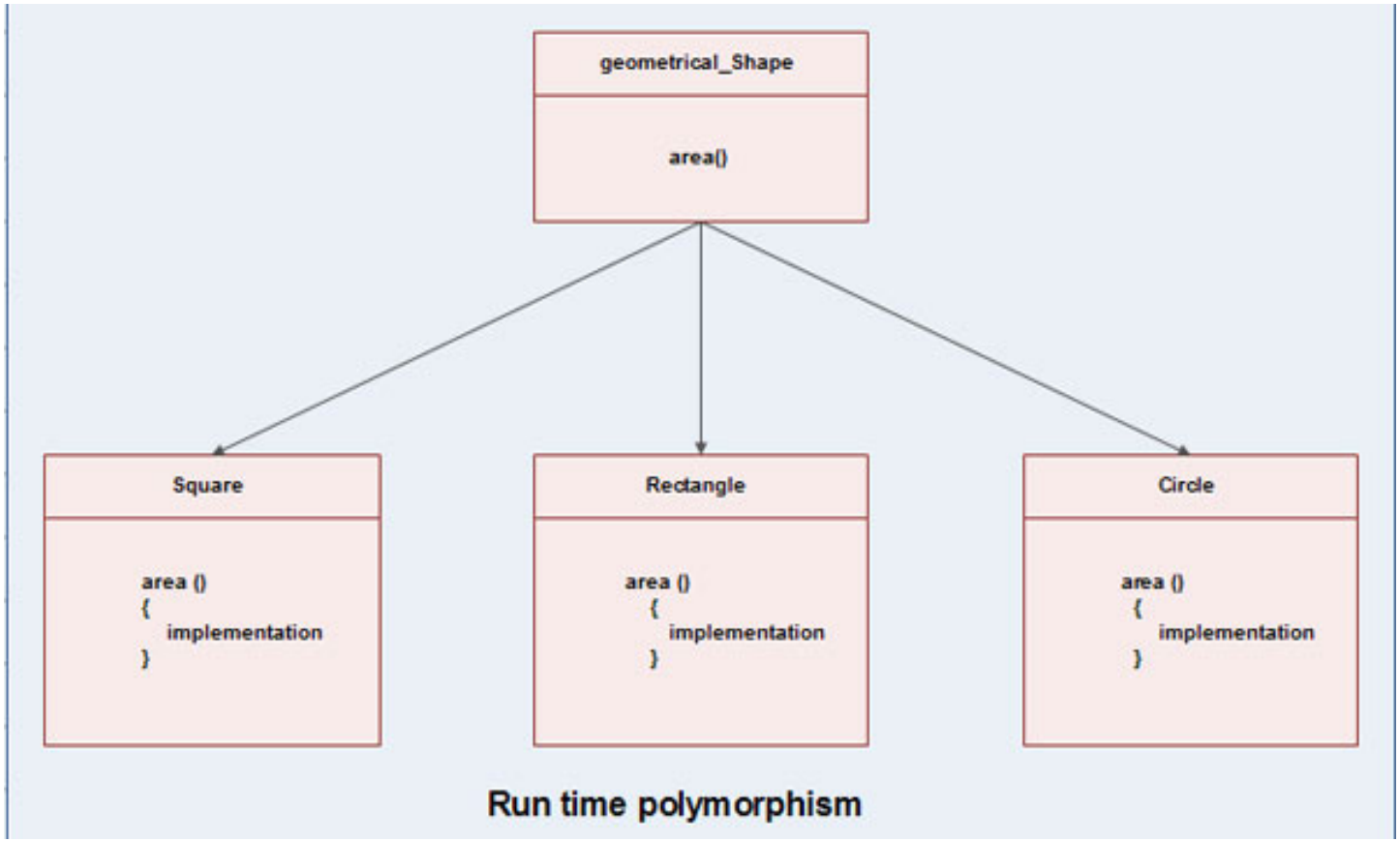
Function overloading



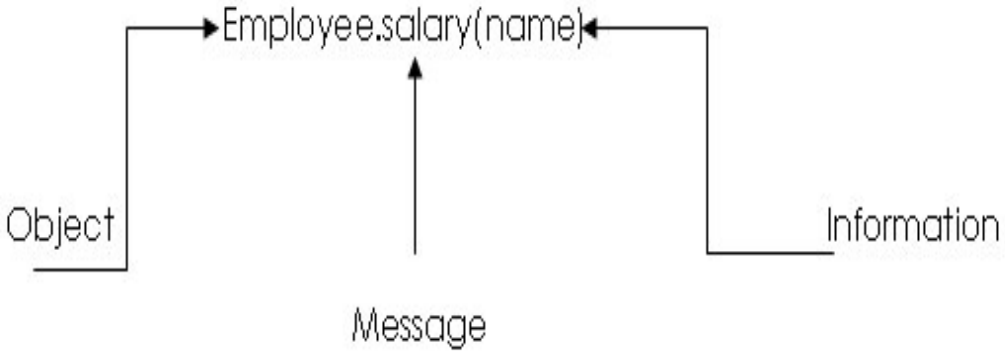
Operator overloading

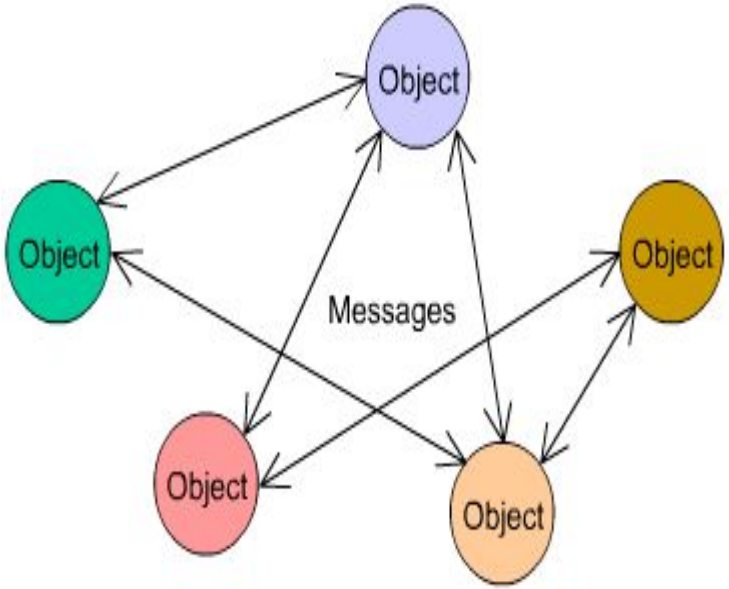


Virtual function



Message passing





Interaction of objects via message passing

Structure in C

```
struct Student{
        char name[20];
        int roll;
        float total

};

void main(){

        struct Student A;   // C declaration
        strcpy(A.name, "Sheldon");
        A.roll = 999;
        A.total = 595.5

}
```

By default public

Limitations of C structure

```
struct complex{
        float x;
        float y;
};
        struct complex c1, c2, c3;
```

- struct data type can not be treated like built-in data type.
  - c3 = c1 + c2; is illegal.
- Do not permit **data hiding** because all the structure members are public members.

Structure in C++

```
struct Student A;   // C declaration
Student A; //C++ declaration
```

- The only **difference** between a structure and a class in C++ :
  - By default, the members of a class are private.
  - By default, the members of a class are public.

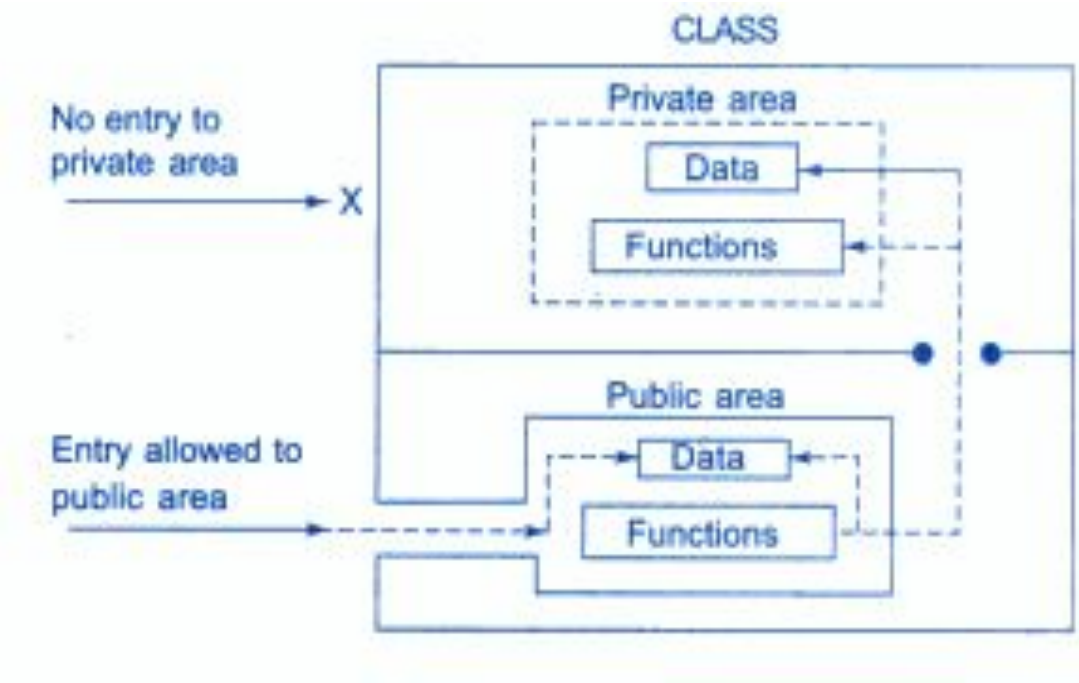
Specifying a class

```
class class_name{
        private:
          variable declarations;
          function declarations;
        public:
          variable declarations;
          function declarations;
};
```

- Class members:
  - Functions – member functions.
  - Variables – data members.
- Visibility labels:
  - Private – can be accessed only from within the class.
  - Public – can be accessed from outside of the class also.
  - Protected – used in inheritance.

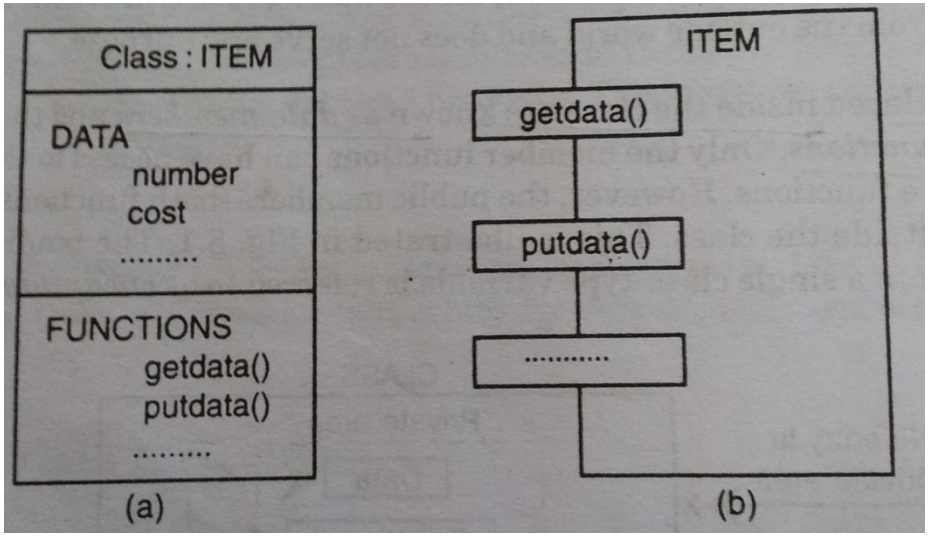
Specifying a class

- Only the member functions can have access to the private data members and private functions.



Simple class example and its representation

```
class item
{
        int number; //variables declaration
        float cost; //private by default

      public:
        void getdata(int a, float b);
        void putdata(void);

};
```



Representation of a class

Creating objects

Once a class has been declared, we can create variables of that type by using the class name.

```
item X ;    // memory for X is created
```

- It creates a variable of of type item. In c++ the class variables are known as objects. Therefore X is called an object of type item.
- Objects can also be created when a class is defined by placing their names immediately after the closing brace as we do in the case of structures. For e.g.-

```
class item
        { --------
            ------------ } x,y,z;
```

But usually we declare the objects close to the place where they are used & not at the time of class definition.

Accessing class members

```
class item
{
        int number; //variables declaration
        float cost; //private by default

      public:
        void getdata(int a, float b);
        void putdata(void);

};
```

item x;

```
object-name.function-name (actual arguments);
```

```
x.getdata(100,75.5);
```
→ legal

```
getdata(100,75.5);
```
→ illegal

```
x.number=100;
```
→ illegal

Defining member functions

- Outside the class definition.
- Inside the class definition.

Outside the class definition

```
return-type class-name :: function-name (argument declaration)
{
        Function body

}
```

Outside the class definition

Membership label

```
return-type class-name :: function-name (argument declaration)
{
        Function body

}
```

```
void item :: getdata(int a, float b)
{
        number=a;
        cost=b;

}

void item :: putdata(void)
{
        cout<< "Number :" << number << "\n";
        cout<< "Cost :" << cost << "\n";

}
```

Characteristics of member function

- Several different classes can use the same function name. The '**membership label**' will resolve their scope.
- Member function can access the private data of the class. A non-member function (except **friend function**) can not do so.
- A member function can call another member function directly without using dot operator.

Inside the class definition

```
class item
{
        int number; //variables declaration
        float cost; //private by default

      public:
        void getdata(int a, float b);
        void putdata(void)
        {
                cout<< "Number :" << number << "\n";
                cout<< "Cost :" << cost << "\n";

        }

};

void item :: getdata(int a, float b)
{
        number=a;
        cost=b;

}
```

- When a function is defined inside of a class, it is treated as an inline function.

Inline function

- Function is used to save **memory space**.
  - Do not need to write the same code again and again.
- When function is called, it takes a lot of **extra time** for-
  - Jumping to the function.
  - Saving registers.
  - Pushing function arguments into the stack.
  - Returning to the calling function.
- When a function is **small**, a substantial percentage of execution time may be spent in such overheads.
- One solution is using **inline function**.

Inline function

- Compiler replaces the **function call** with the corresponding **function code**.
- The **speed benefits** of inline functions **diminish** as the function **grows** in size.
- The functions are made inline when they are **small enough** (one or two lines).
- No inline function in java.

```
inline function-header
{
        function body

}
```

```
inline double cube(double a)
{

        return a*a*a;
}
```

Making an outside function inline

```
class item
{
        .....
        .....
       public:
                void getdata(int a, float b);

};
inline void item :: getdata(int a, float b)
{
        number = a;
        cost = b;

}
```

C++ program with class

```
#include <iostream.h>
class item
{
        int number;  //variables declaration
        float cost;  //private by default

      public:
        void getdata(int a, float b);
        void putdata(void)
        {
                cout<< "Number :" << number << "\n";
                cout<< "Cost :" << cost << "\n";
        }

};


void item :: getdata(int a, float b)
{
        number=a;
        cost=b;
}

void main()
{

        item x;                    //create object x
        cout << "\nobject x " << "\n";
        x.getdata(100, 299.95);
        x.putdata();

        item y;                    //create another object
        y.getdata(200, 175.50);
        y.putdata();

}
```

C++ program with class

```
#include <iostream.h>
class item
{
        int number; //variables declaration
        float cost; //private by default

      public:
        void getdata(int a, float b);
        void putdata(void)
        {
                cout<< "Number :" << number << "\n";
                cout<< "Cost :" << cost << "\n";
        }

};


void item :: getdata(int a, float b)
{
        number=a;
        cost=b;
}

void main()
{

        item x;                    //create object x
        cout << "\nobject x " << "\n";
        x.getdata(100, 299.95);
        x.putdata();

        item y;                    //create another object
        y.getdata(200, 175.50);
        y.putdata();

}
```

C++ program with class

```
#include <iostream.h>
class item
{
        int number; //variables declaration
        float cost; //private by default

      public:
        void getdata(int a, float b);
        void putdata(void)
        {
                cout<< "Number :" << number << "\n";
                cout<< "Cost :" << cost << "\n";
        }

};

void item :: getdata(int a, float b)
{
        number=a;
        cost=b;
}

void main()
{

        item x;                    //create object x
        cout << "\nobject x " << "\n";
        x.getdata(100, 299.95);
        x.putdata();

        item y;                    //create another object
        y.getdata(200, 175.50);
        y.putdata();

}
```

C++ program with class

```
#include <iostream.h>
class item
{
        int number; //variables declaration
        float cost; //private by default

      public:
        void getdata(int a, float b);
        void putdata(void)
        {
                cout<< "Number :" << number << "\n";
                cout<< "Cost :" << cost << "\n";
        }

};


void item :: getdata(int a, float b)
{
        number=a;
        cost=b;
}
void main()
{

        item x;                    //create object x
        cout << "\nobject x " << "\n";
        x.getdata(100, 299.95);
        x.putdata();

        item y;                    //create another object
        y.getdata(200, 175.50);
        y.putdata();

}
```

Java program with class

```
public class FirstJava {
    int number;
    double cost;
    public void getData(int a, double b)
    {
        number = a;
        cost = b;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" + "cost:" + cost);;
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        x.getData(100,125.58);
        x.putData();
    }
}
```

Java program with class

```
public class FirstJava {
    int number;
    double cost;
    public void getData(int a, double b)
    {
        number = a;
        cost = b;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" + "cost:" + cost);;
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        x.getData(100,125.58);
        x.putData();
    }
}
```

Java program with class

```
public class FirstJava {
    int number;
    double cost;
    public void getData(int a, double b)
    {
        number = a;
        cost = b;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" + "cost:" + cost);;
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        x.getData(100,125.58);
        x.putData();
    }
}
```

Java program with class

```
public class FirstJava {
    int number;
    double cost;
    public void getData(int a, double b)
    {
        number = a;
        cost = b;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" + "cost:" + cost);;
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        x.getData(100,125.58);
        x.putData();
    }
}
```

Java program with class

```
public class FirstJava {
    int number;
    double cost;
    public void getData(int a, double b)
    {
        number = a;
        cost = b;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" + "cost:" + cost);;
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        x.getData(100,125.58);
        x.putData();
    }
}
```

Nesting of member functions

```
#include <iostream>
using namespace std;
class Set{

   int m, n;
  public:
   void input(void);
   void display(void);

};
void Set :: input(void)
{
   cout << "Input values of m and n" << endl;
   cin >> m >> n;
}
void Set ::display()
{
   if (m >= n)
      cout << "Largest value = " << m;
   else
      cout << "Largest value = " << n;
}
int main()
{
   Set A;
   A.input();
   A.display();
   return 0;
}
```

Nesting of member functions

```
#include <iostream>
using namespace std;
class Set{

    int m, n;
  public:
    void input(void);
    void display(void);

};
void Set :: input(void)
{
    cout << "Input values of m and n" << endl;
    cin >> m >> n;
}
void Set ::display()
{
    if (m >= n)
        cout << "Largest value = " << m;
    else
        cout << "Largest value = " << n;
}
int main()
{
    Set A;
    A.input();
    A.display();
    return 0;
}
```

Nesting of member functions

```
#include <iostream>
using namespace std;
class Set{

   int m, n;
  public:
   void input(void);
   void display(void);

};
void Set :: input(void)
{
   cout << "Input values of m and n" << endl;
   cin >> m >> n;
}
void Set ::display()
{
   if (m >= n)
      cout << "Largest value = " << m;
   else
      cout << "Largest value = " << n;
}
int main()
{
   Set A;
   A.input();
   A.display();
   return 0;
}
```

```
Input values of m and n
12
34
```

Nesting of member functions

```
#include <iostream>
using namespace std;
class Set{

   int m, n;
  public:
   void input(void);
   void display(void);

};
void Set :: input(void)
{
   cout << "Input values of m and n" << endl;
   cin >> m >> n;
}
void Set ::display()
{
   if (m >= n)
      cout << "Largest value = " << m;
   else
      cout << "Largest value = " << n;
}
int main()
{
   Set A;
   A.input();
   A.display();
   return 0;
}
```

Nesting of member functions

```
#include <iostream>
using namespace std;
class Set{

   int m, n;
  public:
   void input(void);
   void display(void);

};
void Set :: input(void)
{
   cout << "Input values of m and n" << endl;
   cin >> m >> n;
}
void Set ::display()
{
   if (m >= n)
      cout << "Largest value = " << m;
   else
      cout << "Largest value = " << n;
}
int main()
{
   Set A;
   A.input();
   A.display();
   return 0;
}
```

```
Largest value = 34
```

Nesting of member functions

```
#include <iostream>
using namespace std;
class Set{
    int m, n;
  public:
    void input(void);
    void display(void);
    void largest(void);
};
void Set::largest()
{
    if (m >= n)
        cout << "Largest value = " << m;
    else
        cout << "Largest value = " << n;
}
void Set :: input(void)
{
    cout << "Input values of m and n" << endl;
    cin >> m >> n;
}
void Set ::display()
{
    largest();
}
```

```
int main()
{
    Set A;
    A.input();
    A.display();
    return 0;
}
```

Nesting of member functions

```
#include <iostream>
using namespace std;
class Set{
    int m, n;
  public:
    void input(void);
    void display(void);
    void largest(void);
};
void Set::largest()
{
    if (m >= n)
        cout << "Largest value = " << m;
    else
        cout << "Largest value = " << n;
}
void Set :: input(void)
{
    cout << "Input values of m and n" << endl;
    cin >> m >> n;
}
void Set ::display()
{
    largest();
}
```

```
int main()
{
    Set A;
    A.input();
    A.display();
    return 0;
}
```

Nesting of member functions

```
#include <iostream>
using namespace std;
class Set{
   int m, n;
  public:
   void input(void);
   void display(void);
   int largest(void);
};
int Set::largest()
{
   if (m >= n)
      return m;
   else
      return n;
}
void Set :: input(void)
{
   cout << "Input values of m and n" << endl;
   cin >> m >> n;
}
void Set ::display()
{
   cout << "Largest value = " << largest();
}
```

```
int main()
{
   Set A;
   A.input();
   A.display();
   return 0;
}
```

Private member functions

```
class sample
{
        int m;
        void read(void);
       public:
        void update(void);
        void write(void);
};

if s1 is an object of sample, then
        s1.read(); is illegal
```

Private member functions

- Delete an account in a customer file.
- Provide increment to an employee.

```
class sample
{
        int m;
        void read(void);
       public:
        void update(void);
        void write(void);
};

if s1 is an object of sample, then
        s1.read(); is illegal

void sample :: update(void)
{

        read(); //simple call; no object is used
}
```

Arrays within a class

```
const int size = 10;

class array
{
        int a[size];
       public:
        void setval(void);
        void display(void);
};
```

Memory allocation for objects

- **Member functions** are created and placed in the memory space **only once** when they are **defined** as a part of a class specification, **since all the objects** belonging to that class use the **same member functions**.
- Only space for **member variables** is **allocated separately** for each object when the object is declared.

Static data members

- Characteristics:
  - **Initialized to zero** when the **first object** of its class is created. No other initialization is permitted.
  - **Only one copy** of that member is created for the entire class and is **shared by all the objects** of that class.
  - It is **visible** only within the class, but its **lifetime** is the entire program.

C++ program for static data member

```
#include <iostream>

using namespace std;

class Item{

    static int countNum;
    int number;
   public:
       void getdata(int a)
       {
           number = a;
           countNum++;
       }
       void getcount(void)
       {
           cout << "count:" << countNum << "\n";
       }
};
```

C++ program for static data member

```
#include <iostream>

using namespace std;

class Item{

    static int countNum;
    int number;
   public:
       void getdata(int a)
       {
           number = a;
           countNum++;
       }
       void getcount(void)
       {
           cout << "count:" << countNum << "\n";
       }
};
```

C++ program for static data member

```
#include <iostream>

using namespace std;

class Item{

    static int countNum;
    int number;
   public:
       void getdata(int a)
       {
           number = a;
           countNum++;
       }
       void getcount(void)
       {
           cout << "count:" << countNum << "\n";
       }
};
```

C++ program for static data member

```cpp
#include <iostream>

using namespace std;

class Item{

    static int countNum;
    int number;
   public:
        void getdata(int a)
        {
            number = a;
            countNum++;
        }
        void getcount(void)
        {
            cout << "count:" << countNum << "\n";
        }
};
```

```cpp
int Item :: countNum;

int main()
{
    Item a, b, c;
    a.getcount();
    b.getcount();
    c.getcount();

    a.getdata(100);
    b.getdata(100);
    c.getdata(100);

    cout << "After reading data" << "\n";

    a.getcount();
    b.getcount();
    c.getcount();

    return 0;
}
```

C++ program for static data member

```
#include <iostream>

using namespace std;

class Item{

    static int countNum;
    int number;
   public:
       void getdata(int a)
       {
           number = a;
           countNum++;
       }
       void getcount(void)
       {
           cout << "count:" << countNum << "\n";
       }
};
```

```
int Item :: countNum;

int main()
{
    Item a, b, c;
    a.getcount();
    b.getcount();
    c.getcount();

    a.getdata(100);
    b.getdata(100);
    c.getdata(100);

    cout << "After reading data" << "\n";

    a.getcount();
    b.getcount();
    c.getcount();

    return 0;
}
```

C++ program for static data member

```
#include <iostream>

using namespace std;

class Item{

    static int countNum;
    int number;
   public:
       void getdata(int a)
       {
           number = a;
           countNum++;
       }
       void getcount(void)
       {
           cout << "count:" << countNum << "\n";
       }
};
```

```
int Item :: countNum;

int main()
{
    Item a, b, c;
    a.getcount();
    b.getcount();
    c.getcount();

    a.getdata(100);
    b.getdata(100);
    c.getdata(100);

    cout << "After reading data" << "\n";

    a.getcount();
    b.getcount();
    c.getcount();

    return 0;
}
```

```
count:0
count:0
count:0
```

C++ program for static data member

```cpp
#include <iostream>

using namespace std;

class Item{

    static int countNum;
    int number;
   public:
      void getdata(int a)
      {
          number = a;
          countNum++;
      }
      void getcount(void)
      {
          cout << "count:" << countNum << "\n";
      }
};
```

```cpp
int Item :: countNum;

int main()
{
    Item a, b, c;
    a.getcount();
    b.getcount();
    c.getcount();

    a.getdata(100);
    b.getdata(100);
    c.getdata(100);

    cout << "After reading data" << "\n";

    a.getcount();
    b.getcount();
    c.getcount();

    return 0;
}
```

```
count:0
count:0
count:0
```

C++ program for static data member

```
#include <iostream>

using namespace std;

class Item{

    static int countNum;
    int number;
   public:
       void getdata(int a)
       {
           number = a;
           countNum++;
       }
       void getcount(void)
       {
           cout << "count:" << countNum << "\n";
       }
};
```

```
int Item :: countNum;

int main()
{
    Item a, b, c;
    a.getcount();
    b.getcount();
    c.getcount();

    a.getdata(100);
    b.getdata(100);
    c.getdata(100);

    cout << "After reading data" << "\n";

    a.getcount();
    b.getcount();
    c.getcount();

    return 0;
}
```

```
count:0
count:0
count:0
```

C++ program for static data member

```
#include <iostream>

using namespace std;

class Item{

    static int countNum;
    int number;
   public:
       void getdata(int a)
       {
           number = a;
           countNum++;
       }
       void getcount(void)
       {
           cout << "count:" << countNum << "\n";
       }
};
```

```
int Item :: countNum;

int main()
{
    Item a, b, c;
    a.getcount();
    b.getcount();
    c.getcount();

    a.getdata(100);
    b.getdata(100);
    c.getdata(100);

    cout << "After reading data" << "\n";

    a.getcount();
    b.getcount();
    c.getcount();

    return 0;
}
```

```
count:0
count:0
count:0
```

```
After reading data
count:3
count:3
count:3
```

Static data member

- The **type and scope** of each static member variable must be defined **outside** the class definition.
- This is necessary because the static data members are **store separately** rather than as a part of an object.
- Since they are associated with the class itself rather than with any class object, they are also known as ***class variables***.
- int Item :: countNum = 10 ;

Static data member in Java

```
public class FirstJava {
    static int number;
    double cost;
    public void getData()
    {
        cost = ++number ;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" + "cost:" + cost);
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        x.getData();
        x.putData();
    }
}
```

Static data member in Java

```
public class FirstJava {
    static int number;
    double cost;
    public void getData()
    {
        cost = ++number ;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" + "cost:" + cost);
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        x.getData();
        x.putData();
    }
}
```

Static data member in Java

```
public class FirstJava {
    static int number;
    double cost;
    public void getData()
    {
        cost = ++number ;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" + "cost:" + cost);
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        x.getData();
        x.putData();
    }
}
```

Static data member in Java

```
public class FirstJava {
    static int number;
    double cost;
    public void getData()
    {
        cost = ++number ;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" + "cost:" + cost);
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        x.getData();
        x.putData();
    }
}
```

Static data member in Java

```
public class FirstJava {
    static int number;
    double cost;
    public void getData()
    {
        cost = ++number ;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" + "cost:" + cost);
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        x.getData();
        x.putData();
    }
}
```

Static data member in Java

```
public class FirstJava {
    static int number;
    double cost;
    public void getData()
    {
        cost = ++number ;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" + "cost:" + cost);
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        x.getData();
        x.putData();
    }
}
```

Static data member in Java

```
public class FirstJava {
    static int number;
    double cost;
    public void getData()
    {
        cost = ++number ;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" + "cost:" + cost);
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        x.getData();
        x.putData();
    }
}
```

Static data member in Java

```
public class FirstJava {
    static int number;
    double cost;
    public void getData()
    {
        cost = ++number ;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" + "cost:" + cost);
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        x.getData();
        x.putData();
    }
}
```

```
number:1
cost:1.0
```

Static data member in Java

```
public class FirstJava {
    static int number;
    double cost;
    public void getData()
    {
        cost = number++ ;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" + "cost:" + cost);
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        x.getData();
        x.putData();
    }
}
```

Static data member in Java

```
public class FirstJava {
    static int number;
    double cost;
    public void getData()
    {
        cost = number++ ;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" + "cost:" + cost);
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        x.getData();
        x.putData();
    }
}
```

```
number:1
cost:0.0
```

Static member function

```
#include <iostream>

using namespace std;

class Test{

    int code;
    static int count;

    public:
      void setCode(void)
      {
          code = ++count;
      }
      void showCode(void)
      {
          cout << "Object number: " << code << endl;
      }
      static void showCount(void) //static member function
      {
          cout << "count:" <<count <<endl;
      }
};
```

Static member function

```
#include <iostream>

using namespace std;

class Test{

    int code;
    static int count;

    public:
      void setCode(void)
      {
          code = ++count;
      }
      void showCode(void)
      {
          cout << "Object number: " << code << endl;
      }
      static void showCount(void) //static member function
      {
          cout << "count:" <<count <<endl;
      }
};
```

Static member function

```
#include <iostream>

using namespace std;

class Test{

    int code;
    static int count;

    public:
      void setCode(void)
      {
          code = ++count;
      }
      void showCode(void)
      {
          cout << "Object number: " << code << endl;
      }
      static void showCount(void) //static member function
      {
          cout << "count:" <<count <<endl;
      }
};
```

Static member function

```
#include <iostream>

using namespace std;

class Test{

    int code;
    static int count;

    public:
      void setCode(void)
      {
          code = ++count;
      }
      void showCode(void)
      {
          cout << "Object number: " << code << endl;
      }
      static void showCount(void) //static member function
      {
          cout << "count:" <<count <<endl;
      }
};
```

Static member function

```
#include <iostream>

using namespace std;

class Test{

    int code;
    static int count;

    public:
      void setCode(void)
      {
          code = ++count;
      }
      void showCode(void)
      {
          cout << "Object number: " << code << endl;
      }
      static void showCount(void)  //static member fu
      {
          cout << "count:" <<count <<endl;
      }
};
```

```
int Test :: count;

int main()
{

    Test t1, t2;

    t1.setCode();
    t2.setCode();

    Test :: showCount(); //accessing static function

    Test t3;
    t3.setCode();

    Test :: showCount();

    t1.showCode();
    t2.showCode();
    t3.showCode();

    return 0;
}
```

Static member function

```
#include <iostream>

using namespace std;

class Test{

    int code;
    static int count;

    public:
      void setCode(void)
      {
          code = ++count;
      }
      void showCode(void)
      {
          cout << "Object number: " << code << endl;
      }
      static void showCount(void)  //static member fu
      {
          cout << "count:" <<count <<endl;
      }
};
```

```
int Test :: count;

int main()
{

    Test t1, t2;

    t1.setCode();
    t2.setCode();

    Test :: showCount(); //accessing static function

    Test t3;
    t3.setCode();

    Test :: showCount();

    t1.showCode();
    t2.showCode();
    t3.showCode();

    return 0;
}
```

Static member function

```
#include <iostream>

using namespace std;

class Test{

    int code;
    static int count;

    public:
      void setCode(void)
      {
          code = ++count;
      }
      void showCode(void)
      {
          cout << "Object number: " << code << endl;
      }
      static void showCount(void)  //static member fu
      {
          cout << "count:" <<count <<endl;
      }
};
```

```
int Test :: count;

int main()
{

    Test t1, t2;

    t1.setCode();
    t2.setCode();

    Test :: showCount(); //accessing static function

    Test t3;
    t3.setCode();

    Test :: showCount();

    t1.showCode();
    t2.showCode();
    t3.showCode();

    return 0;
}
```

Static member function

```
#include <iostream>

using namespace std;

class Test{

    int code;
    static int count;

    public:
      void setCode(void)
      {
          code = ++count;
      }
      void showCode(void)
      {
          cout << "Object number: " << code << endl;
      }
      static void showCount(void) //static member fu
      {
          cout << "count:" <<count <<endl;
      }
};
```

```
int Test :: count;

int main()
{

    Test t1, t2;

    t1.setCode();
    t2.setCode();

    Test :: showCount(); //accessing static function

    Test t3;
    t3.setCode();

    Test :: showCount();

    t1.showCode();
    t2.showCode();
    t3.showCode();

    return 0;
}
```

```
count:2
```

Static member function

```
#include <iostream>

using namespace std;

class Test{

    int code;
    static int count;

    public:
      void setCode(void)
      {
          code = ++count;
      }
      void showCode(void)
      {
          cout << "Object number: " << code << endl;
      }
      static void showCount(void)  //static member fu
      {
          cout << "count:" <<count <<endl;
      }
};
```

```
int Test :: count;

int main()
{

    Test t1, t2;

    t1.setCode();
    t2.setCode();

    Test :: showCount(); //accessing static function

    Test t3;
    t3.setCode();

    Test :: showCount();

    t1.showCode();
    t2.showCode();
    t3.showCode();

    return 0;
}
```

```
count:2
```

Static member function

```
#include <iostream>

using namespace std;

class Test{

    int code;
    static int count;

    public:
      void setCode(void)
      {
          code = ++count;
      }
      void showCode(void)
      {
          cout << "Object number: " << code << endl;
      }
      static void showCount(void)  //static member fu
      {
          cout << "count:" <<count <<endl;
      }
};
```

```
int Test :: count;

int main()
{

    Test t1, t2;

    t1.setCode();
    t2.setCode();

    Test :: showCount(); //accessing static function

    Test t3;
    t3.setCode();

    Test :: showCount();

    t1.showCode();
    t2.showCode();
    t3.showCode();

    return 0;
}
```

count:2

count:3

Static member function

```
#include <iostream>

using namespace std;

class Test{

    int code;
    static int count;

    public:
      void setCode(void)
      {
          code = ++count;
      }
      void showCode(void)
      {
          cout << "Object number: " << code << endl;
      }
      static void showCount(void)  //static member fu
      {
          cout << "count:" <<count <<endl;
      }
};
```

```
int Test :: count;

int main()
{

    Test t1, t2;

    t1.setCode();
    t2.setCode();

    Test :: showCount(); //accessing static function

    Test t3;
    t3.setCode();

    Test :: showCount();

    t1.showCode();
    t2.showCode();
    t3.showCode();

    return 0;
}
```

count:2

count:3

Static member function

```
#include <iostream>

using namespace std;

class Test{

    int code;
    static int count;

    public:
      void setCode(void)
      {
          code = ++count;
      }
      void showCode(void)
      {
          cout << "Object number: " << code << endl;
      }
      static void showCount(void)  //static member fu
      {
          cout << "count:" <<count <<endl;
      }
};
```

```
int Test :: count;

int main()
{

    Test t1, t2;

    t1.setCode();
    t2.setCode();

    Test :: showCount(); //accessing static function

    Test t3;
    t3.setCode();

    Test :: showCount();

    t1.showCode();
    t2.showCode();
    t3.showCode();

    return 0;
}
```

```
count:2
```

```
count:3
```

```
Object number: 1
Object number: 2
Object number: 3
```

Static member function in Java

```
public class FirstJava {
    static int number;
    public static void getData()
    {
        number++ ;
    }
    public void putData()
    {
        System.out.println("number:"+ number + "\n" );
    }
    public static void main(String[] args) {
        FirstJava x = new FirstJava();
        getData();
        x.putData();
    }
}
```

Arrays of Objects

```
#include <iostream>

using namespace std;

class Employee{

    char name[30];
    float age;
   public:
    void getData(void);
    void putData(void);
};

void Employee :: getData(void)
{
    cout << "Enter name:" ;
    cin >> name;
    cout << "Enter age:" ;
    cin >> age;
}
void Employee :: putData(void)
{
    cout << "Name: " << name << "\n";
    cout << "Age: " << age << "\n";
}
```

```
const int size = 3;
int main()
{
    Employee manager[size];
    for(int i=0; i < size ; i++)
    {
        cout << "\nDetails of manager:" << i+1 << "\n";
        manager[i].getData();
    }
    cout << "\n";
    for(int i=0; i < size ; i++)
    {
        cout << "\nmanager:" << i+1 << "\n";
        manager[i].putData();
    }
    return 0;
}
```

Arrays of Objects

```
#include <iostream>

using namespace std;

class Employee{

    char name[30];
    float age;
   public:
    void getData(void);
    void putData(void);
};

void Employee :: getData(void)
{
    cout << "Enter name:" ;
    cin >> name;
    cout << "Enter age:" ;
    cin >> age;
}
void Employee :: putData(void)
{
    cout << "Name: " << name << "\n";
    cout << "Age: " << age << "\n";
}
```

```
const int size = 3;
int main()
{
    Employee manager[size];
    for(int i=0; i < size ; i++)
    {
        cout << "\nDetails of manager:" << i+1 << "\n";
        manager[i].getData();
    }
    cout << "\n";
    for(int i=0; i < size ; i++)
    {
        cout << "\nmanager:" << i+1 << "\n";
        manager[i].putData();
    }
    return 0;
}
```

Arrays of Objects

```
#include <iostream>

using namespace std;

class Employee{

    char name[30];
    float age;
   public:
    void getData(void);
    void putData(void);
};

void Employee :: getData(void)
{
    cout << "Enter name:" ;
    cin >> name;
    cout << "Enter age:" ;
    cin >> age;
}
void Employee :: putData(void)
{
    cout << "Name: " << name << "\n";
    cout << "Age: " << age << "\n";
}
```

```
const int size = 3;
int main()
{
    Employee manager[size];
    for(int i=0; i < size ; i++)
    {
        cout << "\nDetails of manager:" << i+1 << "\n";
        manager[i].getData();
    }
    cout << "\n";
    for(int i=0; i < size ; i++)
    {
        cout << "\nmanager:" << i+1 << "\n";
        manager[i].putData();
    }
    return 0;
}
```

Objects as function arguments

- An object may be used as a function argument.
  - A copy of the entire object is passed to the function (**pass-by-value**).
  - Only the address of the object is transferred to the function (**pass-by-reference**).

Objects as function arguments

```
#include <iostream>

using namespace std;

class Time{
    int hours;
    int minutes;
   public:
    void getTime(int h, int m)
    {
        hours = h;
        minutes = m;
    }
    void putTime()
    {
        cout << hours << "hours and ";
        cout << minutes << "minutes " << "\n";
    }
    void sum(Time, Time);
};

void Time :: sum(Time t1, Time t2)
{
    minutes = t1.minutes + t2.minutes;
    hours = minutes/60;
    minutes = minutes%60;
    hours = hours + t1.hours + t2.hours;
}
```

Objects as function arguments

```
#include <iostream>

using namespace std;

class Time{
    int hours;
    int minutes;
   public:
    void getTime(int h, int m)
    {
        hours = h;
        minutes = m;
    }
    void putTime()
    {
        cout << hours << "hours and ";
        cout << minutes << "minutes " << "\n";
    }
    void sum(Time, Time);
};

void Time :: sum(Time t1, Time t2)
{
    minutes = t1.minutes + t2.minutes;
    hours = minutes/60;
    minutes = minutes%60;
    hours = hours + t1.hours + t2.hours;
}
```

Objects as function arguments

```
#include <iostream>

using namespace std;

class Time{
    int hours;
    int minutes;
   public:
    void getTime(int h, int m)
    {
        hours = h;
        minutes = m;
    }
    void putTime()
    {
        cout << hours << "hours and ";
        cout << minutes << "minutes " << "\n";
    }
    void sum(Time, Time);
};

void Time :: sum(Time t1, Time t2)
{
    minutes = t1.minutes + t2.minutes;
    hours = minutes/60;
    minutes = minutes%60;
    hours = hours + t1.hours + t2.hours;
}
```

Objects as function arguments

```
#include <iostream>

using namespace std;

class Time{
    int hours;
    int minutes;
   public:
    void getTime(int h, int m)
    {
        hours = h;
        minutes = m;
    }
    void putTime()
    {
        cout << hours << "hours and ";
        cout << minutes << "minutes " << "\n";
    }
    void sum(Time, Time);
};

void Time :: sum(Time t1, Time t2)
{
    minutes = t1.minutes + t2.minutes;
    hours = minutes/60;
    minutes = minutes%60;
    hours = hours + t1.hours + t2.hours;
}
```

Objects as function arguments

```
#include <iostream>

using namespace std;

class Time{
    int hours;
    int minutes;
   public:
    void getTime(int h, int m)
    {
        hours = h;
        minutes = m;
    }
    void putTime()
    {
        cout << hours << "hours and ";
        cout << minutes << "minutes " << "\n";
    }
    void sum(Time, Time);
};

void Time :: sum(Time t1, Time t2)
{
    minutes = t1.minutes + t2.minutes;
    hours = minutes/60;
    minutes = minutes%60;
    hours = hours + t1.hours + t2.hours;
}
```

```
int main()
{
    Time T1, T2, T3;

    T1.getTime(2, 45);
    T2.getTime(3, 30);

    T3.sum(T1, T2);

    cout << "T1 = ";
    T1.putTime();
    cout << "T2 = ";
    T2.putTime();
    cout << "T3 = ";
    T3.putTime();

    return 0;
}
```

Objects as function arguments

```
#include <iostream>

using namespace std;

class Time{
    int hours;
    int minutes;
   public:
    void getTime(int h, int m)
    {
        hours = h;
        minutes = m;
    }
    void putTime()
    {
        cout << hours << "hours and ";
        cout << minutes << "minutes " << "\n";
    }
    void sum(Time, Time);
};

void Time :: sum(Time t1, Time t2)
{
    minutes = t1.minutes + t2.minutes;
    hours = minutes/60;
    minutes = minutes%60;
    hours = hours + t1.hours + t2.hours;
}
```

```
int main()
{
    Time T1, T2, T3;

    T1.getTime(2, 45);
    T2.getTime(3, 30);

    T3.sum(T1, T2);

    cout << "T1 = ";
    T1.putTime();
    cout << "T2 = ";
    T2.putTime();
    cout << "T3 = ";
    T3.putTime();

    return 0;
}
```

Objects as function arguments

```
#include <iostream>

using namespace std;

class Time{
    int hours;
    int minutes;
   public:
    void getTime(int h, int m)
    {
        hours = h;
        minutes = m;
    }
    void putTime()
    {
        cout << hours << "hours and ";
        cout << minutes << "minutes " << "\n";
    }
    void sum(Time, Time);
};

void Time :: sum(Time t1, Time t2)
{
    minutes = t1.minutes + t2.minutes;
    hours = minutes/60;
    minutes = minutes%60;
    hours = hours + t1.hours + t2.hours;
}
```

```
int main()
{
    Time T1, T2, T3;

    T1.getTime(2, 45);
    T2.getTime(3, 30);

    T3.sum(T1, T2);

    cout << "T1 = ";
    T1.putTime();
    cout << "T2 = ";
    T2.putTime();
    cout << "T3 = ";
    T3.putTime();

    return 0;
}
```

Objects as function arguments

```
#include <iostream>

using namespace std;

class Time{
    int hours;
    int minutes;
   public:
    void getTime(int h, int m)
    {
        hours = h;
        minutes = m;
    }
    void putTime()
    {
        cout << hours << "hours and ";
        cout << minutes << "minutes " << "\n";
    }
    void sum(Time, Time);
};

void Time :: sum(Time t1, Time t2)
{
    minutes = t1.minutes + t2.minutes;
    hours = minutes/60;
    minutes = minutes%60;
    hours = hours + t1.hours + t2.hours;
}
```

```
int main()
{
    Time T1, T2, T3;

    T1.getTime(2, 45);
    T2.getTime(3, 30);

    T3.sum(T1, T2);

    cout << "T1 = ";
    T1.putTime();
    cout << "T2 = ";
    T2.putTime();
    cout << "T3 = ";
    T3.putTime();

    return 0;
}
```

Objects as function arguments

```
#include <iostream>

using namespace std;

class Time{
    int hours;
    int minutes;
   public:
    void getTime(int h, int m)
    {
        hours = h;
        minutes = m;
    }
    void putTime()
    {
        cout << hours << "hours and ";
        cout << minutes << "minutes " << "\n";
    }
    void sum(Time, Time);
};

void Time :: sum(Time t1, Time t2)
{
    minutes = t1.minutes + t2.minutes;
    hours = minutes/60;
    minutes = minutes%60;
    hours = hours + t1.hours + t2.hours;
}
```

```
int main()
{
    Time T1, T2, T3;

    T1.getTime(2, 45);
    T2.getTime(3, 30);

    T3.sum(T1, T2);

    cout << "T1 = ";
    T1.putTime();
    cout << "T2 = ";
    T2.putTime();
    cout << "T3 = ";
    T3.putTime();

    return 0;
}
```

```
T1 = 2hours and 45minutes
T2 = 3hours and 30minutes
T3 = 6hours and 15minutes
```

Friendly functions

- A **non-member function** cannot have an access to the private data of a class.
- However, there could be a situation where two classes share a particular function.
  - Manager and scientist class use ***income_tax*** function.
  - ***Income_tax*** function can be made friendly with both the classes.
- The function declaration should be preceded by the keyword ***friend***.
- Functions declared with the keyword ***friend*** are known as friend functions.
- **Friend function** has the full access rights to the private members of the class.
- A function can be declared as ***friend*** in any number of classes.

Friendly functions

```
class abc
{
    ....
    ....
public:
    ....
    ....
    friend void xyz(void);  // declaration
};
```

Friendly functions

```
#include <iostream>

using namespace std;

class Sample{
   int a;
   int b;
  public:
   void setValue(){a=25; b=40;}
   friend float mean(Sample S);
};
float mean(Sample S)
{
   return float(S.a + S.b)/2.0;
}
int main()
{
   Sample X;
   X.setValue();
   cout << "Mean value =" << mean(X) << "\n";

    return 0;
}
```

Friendly functions

```
#include <iostream>

using namespace std;

class Sample{
   int a;
   int b;
  public:
   void setValue(){a=25; b=40;}
   friend float mean(Sample S);
};
float mean(Sample S)
{
   return float(S.a + S.b)/2.0;
}
int main()
{
   Sample X;
   X.setValue();
   cout << "Mean value =" << mean(X) << "\n";

    return 0;
}
```

Friendly functions

```
#include <iostream>

using namespace std;

class Sample{
   int a;
   int b;
  public:
   void setValue(){a=25; b=40;}
   friend float mean(Sample S);
};
float mean(Sample S)
{
   return float(S.a + S.b)/2.0;
}
int main()
{
   Sample X;
   X.setValue();
   cout << "Mean value =" << mean(X) << "\n";

    return 0;
}
```

Friendly functions

```
#include <iostream>

using namespace std;

class Sample{
   int a;
   int b;
  public:
   void setValue(){a=25; b=40;}
   friend float mean(Sample S);
};
float mean(Sample S)
{
   return float(S.a + S.b)/2.0;
}
int main()
{
   Sample X;
   X.setValue();
   cout << "Mean value =" << mean(X) << "\n";

    return 0;
}
```

Friendly functions

```
#include <iostream>

using namespace std;

class Sample{
    int a;
    int b;
  public:
    void setValue(){a=25; b=40;}
    friend float mean(Sample S);
};
float mean(Sample S)
{
    return float(S.a + S.b)/2.0;
}
int main()
{
    Sample X;
    X.setValue();
    cout << "Mean value =" << mean(X) << "\n";

     return 0;
}
```

```
Mean value =32.5
```

A function friendly to two classes

```
#include <iostream>
using namespace std;

class ABC;
class XYZ{
    int x;
   public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

class ABC{
    int x;
   public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

void max(XYZ m, ABC n)
{
    if(m.x >= n.x)
        cout << "Max:" << m.x;
    else
        cout << "Max:"<< n.x;
}
```

Forward declaration

A function friendly to two classes

```
#include <iostream>
using namespace std;

class ABC;
class XYZ{
    int x;
   public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

class ABC{
    int x;
   public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

void max(XYZ m, ABC n)
{
    if(m.x >= n.x)
        cout << "Max:" << m.x;
    else
        cout << "Max:"<< n.x;
}
```

A function friendly to two classes

```
#include <iostream>
using namespace std;

class ABC;
class XYZ{
    int x;
   public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

class ABC{
    int x;
   public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

void max(XYZ m, ABC n)
{
    if(m.x >= n.x)
        cout << "Max:" << m.x;
    else
        cout << "Max:"<< n.x;
}
```

A function friendly to two classes

```
#include <iostream>
using namespace std;

class ABC;
class XYZ{
    int x;
  public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

class ABC{
    int x;
  public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

void max(XYZ m, ABC n)
{
    if(m.x >= n.x)
        cout << "Max:" << m.x;
    else
        cout << "Max:"<< n.x;
}
```

A function friendly to two classes

```
#include <iostream>
using namespace std;

class ABC;
class XYZ{
    int x;
  public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

class ABC{
    int x;
  public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

void max(XYZ m, ABC n)
{
    if(m.x >= n.x)
        cout << "Max:" << m.x;
    else
        cout << "Max:"<< n.x;
}
```

```
int main()
{
    ABC abc;
    abc.setValue(10);
    XYZ xyz;
    xyz.setValue(20);
    max(xyz, abc);

    return 0;
}
```

A function friendly to two classes

```
#include <iostream>
using namespace std;

class ABC;
class XYZ{
    int x;
  public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

class ABC{
    int x;
  public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

void max(XYZ m, ABC n)
{
    if(m.x >= n.x)
        cout << "Max:" << m.x;
    else
        cout << "Max:"<< n.x;
}
```

```
int main()
{
    ABC abc;
    abc.setValue(10);
    XYZ xyz;
    xyz.setValue(20);
    max(xyz, abc);

     return 0;
}
```

A function friendly to two classes

```
#include <iostream>
using namespace std;

class ABC;
class XYZ{
    int x;
  public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

class ABC{
    int x;
  public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

void max(XYZ m, ABC n)
{
    if(m.x >= n.x)
        cout << "Max:" << m.x;
    else
        cout << "Max:"<< n.x;
}
```

```
int main()
{
    ABC abc;
    abc.setValue(10);
    XYZ xyz;
    xyz.setValue(20);
    max(xyz, abc);

     return 0;
}
```

A function friendly to two classes

```
#include <iostream>
using namespace std;

class ABC;
class XYZ{
    int x;
  public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

class ABC{
    int x;
  public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

void max(XYZ m, ABC n)
{
    if(m.x >= n.x)
        cout << "Max:" << m.x;
    else
        cout << "Max:"<< n.x;
}
```

```
int main()
{
    ABC abc;
    abc.setValue(10);
    XYZ xyz;
    xyz.setValue(20);
    max(xyz, abc);

    return 0;
}
```

A function friendly to two classes

```
#include <iostream>
using namespace std;

class ABC;
class XYZ{
    int x;
  public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

class ABC{
    int x;
  public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

void max(XYZ m, ABC n)
{
    if(m.x >= n.x)
        cout << "Max:" << m.x;
    else
        cout << "Max:"<< n.x;
}
```

```
int main()
{
    ABC abc;
    abc.setValue(10);
    XYZ xyz;
    xyz.setValue(20);
    max(xyz, abc);

    return 0;
}
```

A function friendly to two classes

```
#include <iostream>
using namespace std;

class ABC;
class XYZ{
    int x;
  public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

class ABC{
    int x;
  public:
    void setValue(int i){x = i;}
    friend void max(XYZ, ABC);
};

void max(XYZ m, ABC n)
{
    if(m.x >= n.x)
        cout << "Max:" << m.x;
    else
        cout << "Max:"<< n.x;
}
```

```
int main()
{
    ABC abc;
    abc.setValue(10);
    XYZ xyz;
    xyz.setValue(20);
    max(xyz, abc);

    return 0;
}
```

```
Max:20
```

Characteristics of a friend function

- It is **not in the scope of the class** to which it has been declared as friend.
- Since it is not in the scope of the class, it **cannot be called using the object** of that class.
- It can be **invoked like a normal function** without the help of any object.
- Unlike member functions, it cannot access the member names directly and has **to use an object name and dot membership operator** with each member name (e.g. A.x).
- It can be **declared** either in the **public or the private** part of a class without affecting its meaning.
- Usually, it has the **objects as arguments**.

Friend function in Java

- **Java** does not have the friend keyword.

Swapping private data of classes

```
#include <iostream>
using namespace std;
class Class_2;
class Class_1{
    int value1;
  public:
    void inData(int a)
    {
        value1 = a;
    }
    void display(void)
    {
        cout << value1 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
class Class_2{
    int value2;
  public:
    void inData(int a)
    {
        value2 = a;
    }
    void display(void)
    {
        cout << value2 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
```

```
void exchange(Class_1 &x, Class_2 &y)
{
    int temp = x.value1;
    x.value1 = y.value2;
    y.value2 = temp;
}

int main()
{

    Class_1 C1;
    Class_2 C2;
    C1.inData(100);
    C2.inData(200);

    cout << "values before exchange" << endl;
    C1.display();
    C2.display();

    exchange(C1,C2);

    cout << "values after exchange" << endl;
    C1.display();
    C2.display();

    return 0;
}
```

Swapping private data of classes

```
#include <iostream>
using namespace std;
class Class_2;
class Class_1{
    int value1;
  public:
    void inData(int a)
    {
        value1 = a;
    }
    void display(void)
    {
        cout << value1 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
class Class_2{
    int value2;
  public:
    void inData(int a)
    {
        value2 = a;
    }
    void display(void)
    {
        cout << value2 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
```

```
void exchange(Class_1 &x, Class_2 &y)
{
    int temp = x.value1;
    x.value1 = y.value2;
    y.value2 = temp;
}

int main()
{

    Class_1 C1;
    Class_2 C2;
    C1.inData(100);
    C2.inData(200);

    cout << "values before exchange" << endl;
    C1.display();
    C2.display();

    exchange(C1,C2);

    cout << "values after exchange" << endl;
    C1.display();
    C2.display();

    return 0;
}
```

Swapping private data of classes

```
#include <iostream>
using namespace std;
class Class_2;
class Class_1{
    int value1;
  public:
    void inData(int a)
    {
        value1 = a;
    }
    void display(void)
    {
        cout << value1 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
class Class_2{
    int value2;
  public:
    void inData(int a)
    {
        value2 = a;
    }
    void display(void)
    {
        cout << value2 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
```

```
void exchange(Class_1 &x, Class_2 &y)
{
    int temp = x.value1;
    x.value1 = y.value2;
    y.value2 = temp;
}

int main()
{

    Class_1 C1;
    Class_2 C2;
    C1.inData(100);
    C2.inData(200);

    cout << "values before exchange" << endl;
    C1.display();
    C2.display();

    exchange(C1,C2);

    cout << "values after exchange" << endl;
    C1.display();
    C2.display();

    return 0;
}
```

Swapping private data of classes

```
#include <iostream>
using namespace std;
class Class_2;
class Class_1{
    int value1;
  public:
    void inData(int a)
    {
        value1 = a;
    }
    void display(void)
    {
        cout << value1 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
class Class_2{
    int value2;
  public:
    void inData(int a)
    {
        value2 = a;
    }
    void display(void)
    {
        cout << value2 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
```

```
void exchange(Class_1 &x, Class_2 &y)
{
    int temp = x.value1;
    x.value1 = y.value2;
    y.value2 = temp;
}

int main()
{

    Class_1 C1;
    Class_2 C2;
    C1.inData(100);
    C2.inData(200);

    cout << "values before exchange" << endl;
    C1.display();
    C2.display();

    exchange(C1,C2);

    cout << "values after exchange" << endl;
    C1.display();
    C2.display();

    return 0;
}
```

Swapping private data of classes

```
#include <iostream>
using namespace std;
class Class_2;
class Class_1{
    int value1;
  public:
    void inData(int a)
    {
        value1 = a;
    }
    void display(void)
    {
        cout << value1 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
class Class_2{
    int value2;
  public:
    void inData(int a)
    {
        value2 = a;
    }
    void display(void)
    {
        cout << value2 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
```

```
void exchange(Class_1 &x, Class_2 &y)
{
    int temp = x.value1;
    x.value1 = y.value2;
    y.value2 = temp;
}

int main()
{

    Class_1 C1;
    Class_2 C2;
    C1.inData(100);
    C2.inData(200);

    cout << "values before exchange" << endl;
    C1.display();
    C2.display();

    exchange(C1,C2);

    cout << "values after exchange" << endl;
    C1.display();
    C2.display();

    return 0;
}
```

Swapping private data of classes

```
#include <iostream>
using namespace std;
class Class_2;
class Class_1{
    int value1;
  public:
    void inData(int a)
    {
        value1 = a;
    }
    void display(void)
    {
        cout << value1 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
class Class_2{
    int value2;
  public:
    void inData(int a)
    {
        value2 = a;
    }
    void display(void)
    {
        cout << value2 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
```

```
void exchange(Class_1 &x, Class_2 &y)
{
    int temp = x.value1;
    x.value1 = y.value2;
    y.value2 = temp;
}

int main()
{

    Class_1 C1;
    Class_2 C2;
    C1.inData(100);
    C2.inData(200);

    cout << "values before exchange" << endl;
    C1.display();
    C2.display();

    exchange(C1,C2);

    cout << "values after exchange" << endl;
    C1.display();
    C2.display();

    return 0;
}
```

Swapping private data of classes

```
#include <iostream>
using namespace std;
class Class_2;
class Class_1{
    int value1;
  public:
    void inData(int a)
    {
        value1 = a;
    }
    void display(void)
    {
        cout << value1 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
class Class_2{
    int value2;
  public:
    void inData(int a)
    {
        value2 = a;
    }
    void display(void)
    {
        cout << value2 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
```

```
void exchange(Class_1 &x, Class_2 &y)
{
    int temp = x.value1;
    x.value1 = y.value2;
    y.value2 = temp;
}

int main()
{

    Class_1 C1;
    Class_2 C2;
    C1.inData(100);
    C2.inData(200);

    cout << "values before exchange" << endl;
    C1.display();
    C2.display();

    exchange(C1,C2);

    cout << "values after exchange" << endl;
    C1.display();
    C2.display();

    return 0;
}
```

Swapping private data of classes

```
#include <iostream>
using namespace std;
class Class_2;
class Class_1{
    int value1;
  public:
    void inData(int a)
    {
        value1 = a;
    }
    void display(void)
    {
        cout << value1 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
class Class_2{
    int value2;
  public:
    void inData(int a)
    {
        value2 = a;
    }
    void display(void)
    {
        cout << value2 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
```

```
void exchange(Class_1 &x, Class_2 &y)
{
    int temp = x.value1;
    x.value1 = y.value2;
    y.value2 = temp;
}

int main()
{

    Class_1 C1;
    Class_2 C2;
    C1.inData(100);
    C2.inData(200);

    cout << "values before exchange" << endl;
    C1.display();
    C2.display();

    exchange(C1,C2);

    cout << "values after exchange" << endl;
    C1.display();
    C2.display();

    return 0;
}
```

Swapping private data of classes

```
#include <iostream>
using namespace std;
class Class_2;
class Class_1{
    int value1;
  public:
    void inData(int a)
    {
        value1 = a;
    }
    void display(void)
    {
        cout << value1 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
class Class_2{
    int value2;
  public:
    void inData(int a)
    {
        value2 = a;
    }
    void display(void)
    {
        cout << value2 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
```

```
void exchange(Class_1 &x, Class_2 &y)
{
    int temp = x.value1;
    x.value1 = y.value2;
    y.value2 = temp;
}

int main()
{

    Class_1 C1;
    Class_2 C2;
    C1.inData(100);
    C2.inData(200);

    cout << "values before exchange" << endl;
    C1.display();
    C2.display();

    exchange(C1,C2);

    cout << "values after exchange" << endl;
    C1.display();
    C2.display();

    return 0;
}
```

Swapping private data of classes

```
#include <iostream>
using namespace std;
class Class_2;
class Class_1{
    int value1;
  public:
    void inData(int a)
    {
        value1 = a;
    }
    void display(void)
    {
        cout << value1 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
class Class_2{
    int value2;
  public:
    void inData(int a)
    {
        value2 = a;
    }
    void display(void)
    {
        cout << value2 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
```

```
void exchange(Class_1 &x, Class_2 &y)
{
    int temp = x.value1;
    x.value1 = y.value2;
    y.value2 = temp;
}

int main()
{

    Class_1 C1;
    Class_2 C2;
    C1.inData(100);
    C2.inData(200);

    cout << "values before exchange" << endl;
    C1.display();
    C2.display();

    exchange(C1,C2);

    cout << "values after exchange" << endl;
    C1.display();
    C2.display();

    return 0;
}
```

```
values before exchange
100
200
```

Swapping private data of classes

```
#include <iostream>
using namespace std;
class Class_2;
class Class_1{
    int value1;
  public:
    void inData(int a)
    {
        value1 = a;
    }
    void display(void)
    {
        cout << value1 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
class Class_2{
    int value2;
  public:
    void inData(int a)
    {
        value2 = a;
    }
    void display(void)
    {
        cout << value2 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
```

```
void exchange(Class_1 &x, Class_2 &y)
{
    int temp = x.value1;
    x.value1 = y.value2;
    y.value2 = temp;
}

int main()
{

    Class_1 C1;
    Class_2 C2;
    C1.inData(100);
    C2.inData(200);

    cout << "values before exchange" << endl;
    C1.display();
    C2.display();

    exchange(C1,C2);

    cout << "values after exchange" << endl;
    C1.display();
    C2.display();

    return 0;
}
```

Swapping private data of classes

```
#include <iostream>
using namespace std;
class Class_2;
class Class_1{
    int value1;
  public:
    void inData(int a)
    {
        value1 = a;
    }
    void display(void)
    {
        cout << value1 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
class Class_2{
    int value2;
  public:
    void inData(int a)
    {
        value2 = a;
    }
    void display(void)
    {
        cout << value2 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
```

```
void exchange(Class_1 &x, Class_2 &y)
{
    int temp = x.value1;
    x.value1 = y.value2;
    y.value2 = temp;
}

int main()
{

    Class_1 C1;
    Class_2 C2;
    C1.inData(100);
    C2.inData(200);

    cout << "values before exchange" << endl;
    C1.display();
    C2.display();

    exchange(C1,C2);

    cout << "values after exchange" << endl;
    C1.display();
    C2.display();

    return 0;
}
```

Swapping private data of classes

```
#include <iostream>
using namespace std;
class Class_2;
class Class_1{
    int value1;
  public:
    void inData(int a)
    {
        value1 = a;
    }
    void display(void)
    {
        cout << value1 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
class Class_2{
    int value2;
  public:
    void inData(int a)
    {
        value2 = a;
    }
    void display(void)
    {
        cout << value2 << "\n";
    }
    friend void exchange(Class_1 &, Class_2 &);
};
```

```
void exchange(Class_1 &x, Class_2 &y)
{
    int temp = x.value1;
    x.value1 = y.value2;
    y.value2 = temp;
}

int main()
{

    Class_1 C1;
    Class_2 C2;
    C1.inData(100);
    C2.inData(200);

    cout << "values before exchange" << endl;
    C1.display();
    C2.display();

    exchange(C1,C2);

    cout << "values after exchange" << endl;
    C1.display();
    C2.display();

    return 0;
}
```

```
values after exchange
200
100
```

Returning objects

```
#include <iostream>
using namespace std;

class Complex{   //x+iy form
   float x;      //real part
   float y;      //imaginary part

  public:
   void input(float real, float imag)
   {
      x = real;
      y = imag;
   }
   friend Complex sum(Complex c1, Complex c2);
   void show(Complex);
};

Complex sum(Complex c1, Complex c2)
{
   Complex c3;
   c3.x = c1.x + c2.x;
   c3.y = c1.y + c2.y;
   return c3;
}

void Complex :: show(Complex c)
{
   cout << c.x << "+j" << c.y << endl;
}
```

```
int main()
{
   Complex A, B, C;

   A.input(3.1, 5.65);
   B.input(2.75, 1.2);

   C = sum(A, B); // C = A + B

   cout << "A = "; A.show(A);
   cout << "B = "; B.show(B);
   cout << "C = "; C.show(C);
   return 0;
}
```

Returning objects

```
#include <iostream>
using namespace std;

class Complex{   //x+iy form
   float x;      //real part
   float y;      //imaginary part

  public:
   void input(float real, float imag)
   {
      x = real;
      y = imag;
   }
   friend Complex sum(Complex c1, Complex c2);
   void show(Complex);
};

Complex sum(Complex c1, Complex c2)
{
   Complex c3;
   c3.x = c1.x + c2.x;
   c3.y = c1.y + c2.y;
   return c3;
}

void Complex :: show(Complex c)
{
   cout << c.x << "+j" << c.y << endl;
}
```

```
int main()
{
   Complex A, B, C;

   A.input(3.1, 5.65);
   B.input(2.75, 1.2);

   C = sum(A, B); // C = A + B

   cout << "A = "; A.show(A);
   cout << "B = "; B.show(B);
   cout << "C = "; C.show(C);
   return 0;
}
```

Returning objects

```
#include <iostream>
using namespace std;

class Complex{   //x+iy form
   float x;      //real part
   float y;      //imaginary part

  public:
   void input(float real, float imag)
   {
      x = real;
      y = imag;
   }
   friend Complex sum(Complex c1, Complex c2);
   void show(Complex);
};

Complex sum(Complex c1, Complex c2)
{
   Complex c3;
   c3.x = c1.x + c2.x;
   c3.y = c1.y + c2.y;
   return c3;
}

void Complex :: show(Complex c)
{
   cout << c.x << "+j" << c.y << endl;
}
```

```
int main()
{
   Complex A, B, C;

   A.input(3.1, 5.65);
   B.input(2.75, 1.2);

   C = sum(A, B); // C = A + B

   cout << "A = "; A.show(A);
   cout << "B = "; B.show(B);
   cout << "C = "; C.show(C);
   return 0;
}
```

Returning objects

```
#include <iostream>
using namespace std;

class Complex{   //x+iy form
   float x;      //real part
   float y;      //imaginary part

  public:
   void input(float real, float imag)
   {
      x = real;
      y = imag;
   }
   friend Complex sum(Complex c1, Complex c2);
   void show(Complex);
};

Complex sum(Complex c1, Complex c2)
{
   Complex c3;
   c3.x = c1.x + c2.x;
   c3.y = c1.y + c2.y;
   return c3;
}

void Complex :: show(Complex c)
{
   cout << c.x << "+j" << c.y << endl;
}
```

```
int main()
{
   Complex A, B, C;

   A.input(3.1, 5.65);
   B.input(2.75, 1.2);

   C = sum(A, B); // C = A + B

   cout << "A = "; A.show(A);
   cout << "B = "; B.show(B);
   cout << "C = "; C.show(C);
   return 0;
}
```

```
A = 3.1+j5.65
B = 2.75+j1.2
C = 5.85+j6.85
```

Friendly functions

```
class X
{
        .....
        .....
        int fun1();          // member function of X
        .....
};

class Y
{
        .....
        .....
        friend int X :: fun1();          // fun1() of X
                                         // is friend of Y
        .....
};
```

Friendly functions

```
class Z
{
        .....
        friend class X;

};
```

Friend class

```
// friend class
#include <iostream>
using namespace std;

class Square;

class Rectangle {
    int width, height;
  public:
    int area ()
      {return (width * height);}
    void convert (Square a);
};

class Square {
  friend class Rectangle;
  private:
    int side;
  public:
    Square (int a){side = a;}
};

void Rectangle::convert (Square a) {
  width = a.side;
  height = a.side;
}
```

Friend class

```
// friend class
#include <iostream>
using namespace std;

class Square;

class Rectangle {
    int width, height;
  public:
    int area ()
      {return (width * height);}
    void convert (Square a);
};

class Square {
  friend class Rectangle;
  private:
    int side;
  public:
    Square (int a){side = a;}
};

void Rectangle::convert (Square a) {
  width = a.side;
  height = a.side;
}
```

Friend class

```
// friend class
#include <iostream>
using namespace std;

class Square;

class Rectangle {
    int width, height;
  public:
    int area ()
      {return (width * height);}
    void convert (Square a);
};

class Square {
  friend class Rectangle;
  private:
    int side;
  public:
    Square (int a){side = a;}
};

void Rectangle::convert (Square a) {
  width = a.side;
  height = a.side;
}
```

Friend class

```
// friend class
#include <iostream>
using namespace std;

class Square;

class Rectangle {
    int width, height;
  public:
    int area ()
      {return (width * height);}
    void convert (Square a);
};

class Square {
  friend class Rectangle;
  private:
    int side;
  public:
    Square (int a){side = a;}
};

void Rectangle::convert (Square a) {
  width = a.side;
  height = a.side;
}
```

Friend class

```
// friend class
#include <iostream>
using namespace std;

class Square;

class Rectangle {
    int width, height;
  public:
    int area ()
      {return (width * height);}
    void convert (Square a);
};

class Square {
  friend class Rectangle;
  private:
    int side;
  public:
    Square (int a){side = a;}
};

void Rectangle::convert (Square a) {
  width = a.side;
  height = a.side;
}
```

```
int main () {
  Rectangle rect;
  Square sqr (4);
  rect.convert(sqr);
  cout << rect.area();
  return 0;
}
```

Friend class

```
// friend class
#include <iostream>
using namespace std;

class Square;

class Rectangle {
    int width, height;
  public:
    int area ()
      {return (width * height);}
    void convert (Square a);
};

class Square {
  friend class Rectangle;
  private:
    int side;
  public:
    Square (int a){side = a;}
};

void Rectangle::convert (Square a) {
  width = a.side;
  height = a.side;
}
```

```
int main () {
  Rectangle rect;
  Square sqr (4);
  rect.convert(sqr);
  cout << rect.area();
  return 0;
}
```

Friend class

```
// friend class
#include <iostream>
using namespace std;

class Square;

class Rectangle {
    int width, height;
  public:
    int area ()
      {return (width * height);}
    void convert (Square a);
};

class Square {
  friend class Rectangle;
  private:
    int side;
  public:
    Square (int a){side = a;}
};

void Rectangle::convert (Square a) {
  width = a.side;
  height = a.side;
}
```

```
int main () {
  Rectangle rect;
  Square sqr (4);
  rect.convert(sqr);
  cout << rect.area();
  return 0;
}
```

Friend class

```
// friend class
#include <iostream>
using namespace std;

class Square;

class Rectangle {
    int width, height;
  public:
    int area ()
      {return (width * height);}
    void convert (Square a);
};

class Square {
  friend class Rectangle;
  private:
    int side;
  public:
    Square (int a){side = a;}
};

void Rectangle::convert (Square a) {
  width = a.side;
  height = a.side;
}
```

```
int main () {
  Rectangle rect;
  Square sqr (4);
  rect.convert(sqr);
  cout << rect.area();
  return 0;
}
```

Friend class

```
// friend class
#include <iostream>
using namespace std;

class Square;

class Rectangle {
    int width, height;
  public:
    int area ()
      {return (width * height);}
    void convert (Square a);
};

class Square {
  friend class Rectangle;
  private:
    int side;
  public:
    Square (int a){side = a;}
};

void Rectangle::convert (Square a) {
  width = a.side;
  height = a.side;
}
```

```
int main () {
  Rectangle rect;
  Square sqr (4);
  rect.convert(sqr);
  cout << rect.area();
  return 0;
}
```

```
16
```

Friend class

```
1   // friend class
2   #include <iostream>
3   using namespace std;
4
5   class Square;
6
7   class Rectangle {
8       int width, height;
9     public:
10      int area ()
11        {return (width * height);}
12      void convert (Square a);
13  };
14
15  class Square {
16    //friend class Rectangle;
17    private:
18      int side;
19    public:
20      Square (int a){side = a;}
21  };
22
23  void Rectangle::convert (Square a) {
24    width = a.side;
25    height = a.side;
26  }
```

Friend class

```
// friend class
#include <iostream>
using namespace std;

class Square;

class Rectangle {
    int width, height;
  public:
    int area ()
      {return (width * height);}
    void convert (Square a);
};

class Square {
  //friend class Rectangle;
  private:
    int side;
  public:
    Square (int a){side = a;}
    int side_square(void){return side;}
};

void Rectangle::convert (Square a) {
  width = a.side_square();
  height = a.side_square();
}
```

Constructor

- A **special member function** whose task is to initialize the objects of its class.
- **Special** – name is same as class name.
- It is **invoked whenever an object** of its associated class is **created**.
- It is called **constructor** because it constructs the values of data members of the class.

Constructor

```
#include <iostream>
using namespace std;

class integer
{
   int m,n ;
  public:
   integer(void); // constructor declared
   ......
   ......
};

integer :: integer(void)   // constructor defined
{
   m = 0; n = 0;
}
```

```
integer int1;
```

Creates int1 and initializes its data members m and n to zero

Constructor

- If a normal member function is defined for zero initialization, we would need to invoke this function for each of the objects separately.
- This is inconvenient for a large number of objects.

Default constructor

- A constructor that accepts no parameters is called the **default constructor**.
- Default constructor for **class A** is ***A::A()*.**
- If no constructor is defined, then the compiler supplies a default constructor.
- ***A a;*** -- invokes the default constructor of the compiler to create the object ***a***.

Special characteristics

- Should be **declared** in the **public section**.
- **Automatically invoked** when the objects are created.
- No **return types**, not even **void** and therefore cannot return values.
- Cannot be **inherited**, though a derived class can call the base class constructor.
- They can have **default arguments**.
- Constructors cannot be **virtual**.
- Make implicit call to the operators **new** and **delete** when allocation is required.

Parameterized constructors

```
1   #include <iostream>
2   using namespace std;
3
4   class integer
5   {
6       int m,n ;
7     public:
8       integer(int x, int y); // constructor declared
9
10  };
11
12  integer :: integer(int x, int y)  // constructor defined
13  {
14      m = x; n = y;
15  }
16
17  int main()
18  {
19  integer int1;
20
21  return 0;
22  }
```

integer int1;

Parameterized constructors

```
#include <iostream>
using namespace std;

class integer
{
   int m,n ;
  public:
   integer(int x, int y); // constructor declared

};

integer :: integer(int x, int y)  // constructor defined
{
   m = x; n = y;
}

int main()
{
integer int1 = integer(50, 100);

integer int2(0, 10);

return 0;
}
```

Explicit call

Implicit call – shorthand method

Parameterized constructors

```
#include <iostream>
using namespace std;

class integer
{
   int m,n ;
  public:
   integer(int x, int y); // constructor declared
   void display(void)
   {
      cout << " m = " << m << "\n";
      cout << " n = " << n << "\n";
   }

};

integer :: integer(int x, int y)  // constructor defined
{
   m = x; n = y;
}
```

```
int main()
{
integer int1 = integer(50, 100);

integer int2(25, 75);

cout << "\nOBJECT1" << "\n";
int1.display();

cout << "\nOBJECT2" << "\n";
int2.display();

return 0;
}
```

Parameterized constructors

```
#include <iostream>
using namespace std;

class integer
{
   int m,n ;
  public:
   integer(int x, int y); // constructor declared
   void display(void)
   {
      cout << " m = " << m << "\n";
      cout << " n = " << n << "\n";
   }

};

integer :: integer(int x, int y)  // constructor defined
{
   m = x; n = y;
}
```

```
int main()
{
integer int1 = integer(50, 100);

integer int2(25, 75);

cout << "\nOBJECT1" << "\n";
int1.display();

cout << "\nOBJECT2" << "\n";
int2.display();

return 0;
}
```

```
OBJECT1
 m = 50
 n = 100

OBJECT2
 m = 25
 n = 75
```

Parameterized constructors

```
#include <iostream>
using namespace std;

class integer
{
   int m,n ;
  public:
   integer(int x, int y)   // constructor defined
   {
      m = x; n = y;
   }
   void display(void)
   {
      cout << " m = " << m << "\n";
      cout << " n = " << n << "\n";
   }

};
```

Inline function

Parameterized constructors

```
#include <iostream>
using namespace std;

class A
{
    ....
    ....

  public:
    A(A);
};
```

Parameter can be of any type except that of the class to which it belongs

Parameterized constructors

```
#include <iostream>
using namespace std;

class A
{
    ....
    ....

  public:
    A(A&);
};
```

Can accept a reference to its own class as a parameter – copy constructor

Multiple constructor in a class

```
#include <iostream>
using namespace std;

class Integer
{
    int m,n ;
  public:
    Integer(){m=0; n=0;}

    Integer(int x, int y)
    {
        m = x;
        n = y;
    }

    Integer(Integer &i)
    {
        m = i.m;
        n = i.n;
    }

    void display(void)
    {
        cout << " m = " << m << ",";
        cout << " n = " << n << "\n";
    }

};
```

```
int main()
{
Integer I1;

Integer I2(20, 40);

Integer I3(I2);

I1.display();
I2.display();
I3.display();
return 0;
}
```

Multiple constructor in a class

```
#include <iostream>
using namespace std;

class Integer
{
    int m,n ;
  public:
    Integer(){m=0; n=0;}

    Integer(int x, int y)
    {
        m = x;
        n = y;
    }

    Integer(Integer &i)
    {
        m = i.m;
        n = i.n;
    }

    void display(void)
    {
        cout << " m = " << m << ",";
        cout << " n = " << n << "\n";
    }

};
```

```
int main()
{
Integer I1;

Integer I2(20, 40);

Integer I3(I2);

I1.display();
I2.display();
I3.display();
return 0;
}
```

Multiple constructor in a class

```
#include <iostream>
using namespace std;

class Integer
{
    int m,n ;
   public:
    Integer(){m=0; n=0;}

    Integer(int x, int y)
    {
        m = x;
        n = y;
    }

    Integer(Integer &i)
    {
        m = i.m;
        n = i.n;
    }

    void display(void)
    {
        cout << " m = " << m << ",";
        cout << " n = " << n << "\n";
    }

};
```

copy constructor

```
int main()
{
Integer I1;

Integer I2(20, 40);

Integer I3(I2);

I1.display();
I2.display();
I3.display();
return 0;
}
```

Multiple constructor in a class

```
#include <iostream>
using namespace std;

class Integer
{
   int m,n ;
  public:
  //Integer(){m=0; n=0;}

   Integer(int x, int y)
   {
      m = x;
      n = y;
   }

   Integer(Integer &i)
   {
      m = i.m;
      n = i.n;
   }

   void display(void)
   {
      cout << " m = " << m << ",";
      cout << " n = " << n << "\n";
   }

};
```

```
int main()
{
Integer I1;

Integer I2(20, 40);

Integer I3(I2);

I1.display();
I2.display();
I3.display();
return 0;
}
```

Multiple constructor in a class

```
#include <iostream>
using namespace std;

class Integer
{
   int m,n ;
  public:
  //Integer(){m=0; n=0;}

   Integer(int x, int y)
   {
      m = x;
      n = y;
   }

   Integer(Integer &i)
   {
      m = i.m;
      n = i.n;
   }

   void display(void)
   {
      cout << " m = " << m << ",";
      cout << " n = " << n << "\n";
   }

};
```

```
int main()
{
Integer I1;

Integer I2(20, 40);

Integer I3(I2);

I1.display();
I2.display();
I3.display();
return 0;
}
```

Multiple constructor in a class

```
#include <iostream>
using namespace std;

class Complex
{
   float x, y ;
  public:
  Complex(){}
  Complex(float a){x = y = a;}
  Complex(float real, float imag)
  {x = real; y = imag;}
   friend Complex sum(Complex, Complex);
   friend void show(Complex);
};

Complex sum(Complex c1, Complex c2)
{
   Complex c3;
   c3.x = c1.x + c2.x;
   c3.y = c1.y + c2.y;
   return c3;
}

void show(Complex c)
{
   cout << c.x << " + j" << c.y << endl;
}
```

Multiple constructor in a class

```
#include <iostream>
using namespace std;

class Complex
{
   float x, y ;
  public:
  Complex(){}
  Complex(float a){x = y = a;}
  Complex(float real, float imag)
  {x = real; y = imag;}
   friend Complex sum(Complex, Complex);
   friend void show(Complex);
};

Complex sum(Complex c1, Complex c2)
{
   Complex c3;
   c3.x = c1.x + c2.x;
   c3.y = c1.y + c2.y;
   return c3;
}

void show(Complex c)
{
   cout << c.x << " + j" << c.y << endl;
}
```

Multiple constructor in a class

```
#include <iostream>
using namespace std;

class Complex
{
   float x, y ;
  public:
  Complex(){}
  Complex(float a){x = y = a;}
  Complex(float real, float imag)
  {x = real; y = imag;}
   friend Complex sum(Complex, Complex);
   friend void show(Complex);
};

Complex sum(Complex c1, Complex c2)
{
   Complex c3;
   c3.x = c1.x + c2.x;
   c3.y = c1.y + c2.y;
   return c3;
}

void show(Complex c)
{
   cout << c.x << " + j" << c.y << endl;
}
```

Multiple constructor in a class

```
#include <iostream>
using namespace std;

class Complex
{
   float x, y ;
  public:
  Complex(){}
  Complex(float a){x = y = a;}
  Complex(float real, float imag)
  {x = real; y = imag;}
   friend Complex sum(Complex, Complex);
   friend void show(Complex);
};

Complex sum(Complex c1, Complex c2)
{
   Complex c3;
   c3.x = c1.x + c2.x;
   c3.y = c1.y + c2.y;
   return c3;
}

void show(Complex c)
{
   cout << c.x << " + j" << c.y << endl;
}
```

Multiple constructor in a class

```
#include <iostream>
using namespace std;

class Complex
{
   float x, y ;
  public:
  Complex(){}
  Complex(float a){x = y = a;}
  Complex(float real, float imag)
  {x = real; y = imag;}
   friend Complex sum(Complex, Complex);
   friend void show(Complex);
};

Complex sum(Complex c1, Complex c2)
{
   Complex c3;
   c3.x = c1.x + c2.x;
   c3.y = c1.y + c2.y;
   return c3;
}

void show(Complex c)
{
   cout << c.x << " + j" << c.y << endl;
}
```

```
int main()
{
Complex A(2.7, 3.5);
Complex B(1.6);
Complex C;

C = sum(A, B);

cout << "A = "; show(A);
cout << "B = "; show(B);
cout << "C = "; show(C);

//Another way to give initial value

Complex P, Q, R;
P = Complex(2.5, 3.9);
Q = Complex(1.6, 2.5);
R = sum(P, Q);
cout << "\n";
cout << "P = "; show(P);
cout << "Q = "; show(Q);
cout << "R = "; show(R);


return 0;
}
```

Multiple constructor in a class

```
#include <iostream>
using namespace std;

class Complex
{
   float x, y ;
  public:
  Complex(){}
  Complex(float a){x = y = a;}
  Complex(float real, float imag)
  {x = real; y = imag;}
   friend Complex sum(Complex, Complex);
   friend void show(Complex);
};

Complex sum(Complex c1, Complex c2)
{
   Complex c3;
   c3.x = c1.x + c2.x;
   c3.y = c1.y + c2.y;
   return c3;
}

void show(Complex c)
{
   cout << c.x << " + j" << c.y << endl;
}
```

```
int main()
{
Complex A(2.7, 3.5);
Complex B(1.6);
Complex C;

C = sum(A, B);

cout << "A = "; show(A);
cout << "B = "; show(B);
cout << "C = "; show(C);

//Another way to give initial value

Complex P, Q, R;
P = Complex(2.5, 3.9);
Q = Complex(1.6, 2.5);
R = sum(P, Q);
cout << "\n";
cout << "P = "; show(P);
cout << "Q = "; show(Q);
cout << "R = "; show(R);


return 0;
}
```

Multiple constructor in a class

```
#include <iostream>
using namespace std;

class Complex
{
   float x, y ;
  public:
  Complex(){}
  Complex(float a){x = y = a;}
  Complex(float real, float imag)
  {x = real; y = imag;}
   friend Complex sum(Complex, Complex);
   friend void show(Complex);
};

Complex sum(Complex c1, Complex c2)
{
   Complex c3;
   c3.x = c1.x + c2.x;
   c3.y = c1.y + c2.y;
   return c3;
}

void show(Complex c)
{
   cout << c.x << " + j" << c.y << endl;
}
```

```
int main()
{
Complex A(2.7, 3.5);
Complex B(1.6);
Complex C;

C = sum(A, B);

cout << "A = "; show(A);
cout << "B = "; show(B);
cout << "C = "; show(C);

//Another way to give initial value

Complex P, Q, R;
P = Complex(2.5, 3.9);
Q = Complex(1.6, 2.5);
R = sum(P, Q);
cout << "\n";
cout << "P = "; show(P);
cout << "Q = "; show(Q);
cout << "R = "; show(R);


return 0;
}
```

Multiple constructor in a class

```
#include <iostream>
using namespace std;

class Complex
{
   float x, y ;
  public:
  Complex(){}
  Complex(float a){x = y = a;}
  Complex(float real, float imag)
  {x = real; y = imag;}
   friend Complex sum(Complex, Complex);
   friend void show(Complex);
};

Complex sum(Complex c1, Complex c2)
{
   Complex c3;
   c3.x = c1.x + c2.x;
   c3.y = c1.y + c2.y;
   return c3;
}

void show(Complex c)
{
   cout << c.x << " + j" << c.y << endl;
}
```

```
int main()
{
Complex A(2.7, 3.5);
Complex B(1.6);
Complex C;

C = sum(A, B);

cout << "A = "; show(A);
cout << "B = "; show(B);
cout << "C = "; show(C);

//Another way to give initial value

Complex P, Q, R;
P = Complex(2.5, 3.9);
Q = Complex(1.6, 2.5);
R = sum(P, Q);
cout << "\n";
cout << "P = "; show(P);
cout << "Q = "; show(Q);
cout << "R = "; show(R);


return 0;
}
```

Multiple constructor in a class

```
#include <iostream>
using namespace std;

class Complex
{
   float x, y ;
  public:
  Complex(){}
  Complex(float a){x = y = a;}
  Complex(float real, float imag)
  {x = real; y = imag;}
   friend Complex sum(Complex, Complex);
   friend void show(Complex);
};

Complex sum(Complex c1, Complex c2)
{
   Complex c3;
   c3.x = c1.x + c2.x;
   c3.y = c1.y + c2.y;
   return c3;
}

void show(Complex c)
{
   cout << c.x << " + j" << c.y << endl;
}
```

```
int main()
{
Complex A(2.7, 3.5);
Complex B(1.6);
Complex C;

C = sum(A, B);

cout << "A = "; show(A);
cout << "B = "; show(B);
cout << "C = "; show(C);

//Another way to give initial value

Complex P, Q, R;
P = Complex(2.5, 3.9);
Q = Complex(1.6, 2.5);
R = sum(P, Q);
cout << "\n";
cout << "P = "; show(P);
cout << "Q = "; show(Q);
cout << "R = "; show(R);


return 0;
}
```

Multiple constructor in a class

```
#include <iostream>
using namespace std;

class Complex
{
   float x, y ;
  public:
  Complex(){}
  Complex(float a){x = y = a;}
  Complex(float real, float imag)
  {x = real; y = imag;}
   friend Complex sum(Complex, Complex);
   friend void show(Complex);
};

Complex sum(Complex c1, Complex c2)
{
   Complex c3;
   c3.x = c1.x + c2.x;
   c3.y = c1.y + c2.y;
   return c3;
}

void show(Complex c)
{
   cout << c.x << " + j" << c.y << endl;
}
```

```
int main()
{
Complex A(2.7, 3.5);
Complex B(1.6);
Complex C;

C = sum(A, B);

cout << "A = "; show(A);
cout << "B = "; show(B);
cout << "C = "; show(C);

//Another way to give initial value

Complex P, Q, R;
P = Complex(2.5, 3.9);
Q = Complex(1.6, 2.5);
R = sum(P, Q);
cout << "\n";
cout << "P = "; show(P);
cout << "Q = "; show(Q);
cout << "R = "; show(R);


return 0;
}
```

```
A = 2.7 + j3.5
B = 1.6 + j1.6
C = 4.3 + j5.1
```

References

- “Object oriented programming with C++” – E balagurusamy, second edition.
- http://www.cplusplus.com/doc/tutorial/inheritance/
- Web.