Object Oriented Programming

Aditya Kundu

07/Sep/2022

C++ "this" pointer :-

In C++ programming language 'this' is a keyword that refers the current instant. There are three way of usage of 'this' pointer

① It can be used to pass current object as a parameter to another method.

② It can be used to refer current class instant variables.

③ It can be used to declare to indenere.

```
#include <iostream>
Using namespace std;
class employee{
public:
  int id;
  string name;
  float salary;
  employee(int id, string name, float salary)
  { this → id = id;
    this → name = name;
    this → salary = salary;
  }
  void display(){
cout << id << " " << name << " " << salary << endl; }
```

```
#include <string>
int main() {
Employee e1 = Employee(004, "Aditya", 100.000);
Employee e2 = Employee(001, "Alvi", 85,000);
e1.display(); e2.display();
return 0;
}
```

Inheritance :-

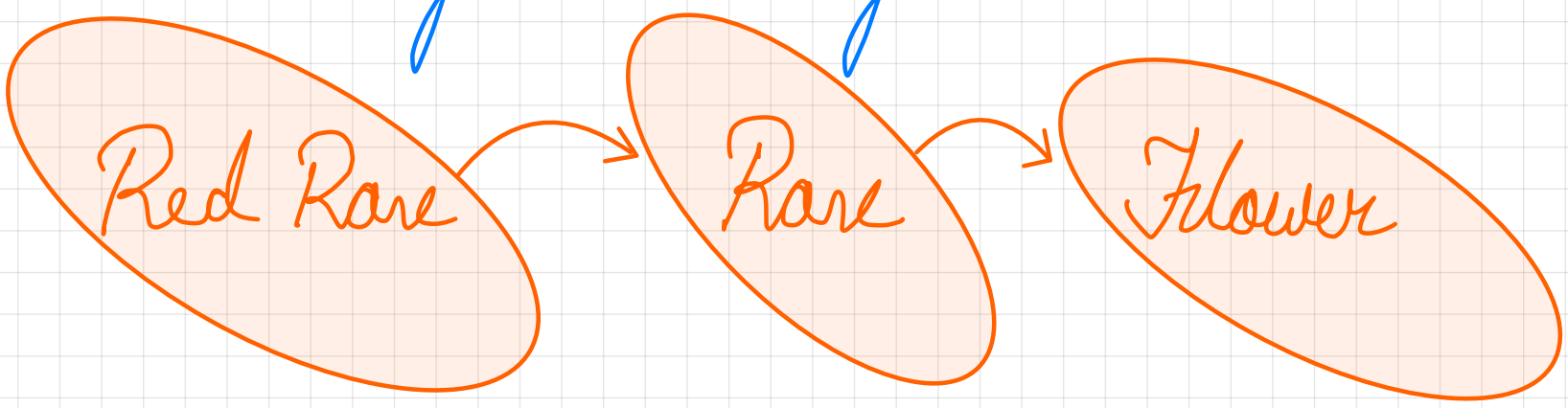
- "IS-A" Relationship

⇒ **Rose IS-A flower**

- Rose has the properties of flowers like fragrance, having petals.
- Rose has some additional properties like having rosy smell.
- Rose is a specialization of flower.
- Flower is a generalization of rose.

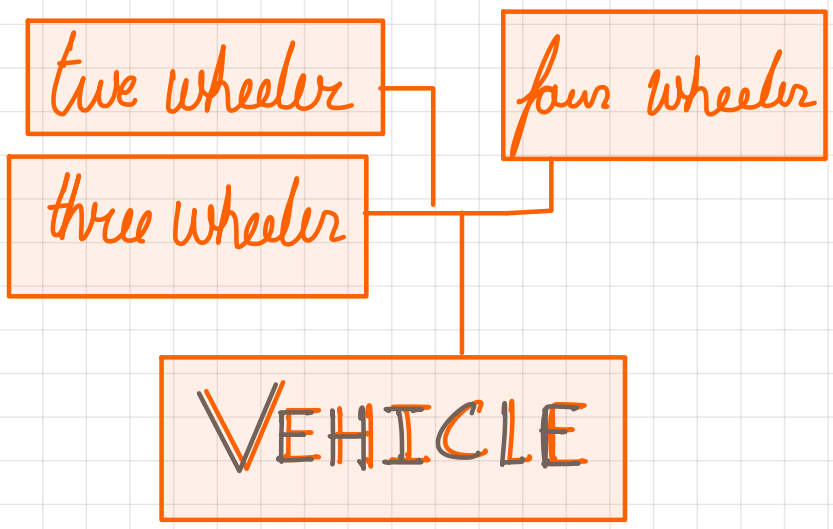
⇒ **"Red Rose IS-A Rose"**

- Red Rose has the properties of rose like rosy fragrance.
- Red rose has some additional property like it is red.
- Red rose is a specialization of rose.
- Rose is a generalization of Red rose.



- two wheeler IS-A vehicle
- three wheeler IS-A vehicle
- four wheeler IS-A vehicle.

⇒



14/Sep/2022

```
class Vehicle; // Base class = Vehicle
class twowheeler : Public vehicle; // derived class = two wheeler
class fourwheeler : Public vehicle; // derived class = fourwheeler.
```

Multilevel inheritance :-

"Red Rose Is-A Rose Is-A Flower"


```
class flower;
class rose : Public flower;
class red rose : public rose;
```

Data Members :-

- Derived class inherits all data members of base class.
- Derived class may add data members of its own.

Member functions :-

- Derived class inherits all the member functions of the base class.
- Derived class may overwrite a member function of base class by redefining it with same signature.
- Derived class may overload a member function by redefining it with same name but different signature

Access Specification :-

- Derived class can't access the private members of base class.
- Derived class can't access the protected members of the base class.

Construction & Destruction :-

→ A constructor of the derived class must first call a constructor of the base class to construct the base class instance of the derived class.

→ The destructor of the derived class must call the destructor of the base class to destruct the base class instant of the derived class.

```
class B {
int data1B;
public:
int data2B;
};
class D : public B {
// inheritance B.data2B; —
// Inheritance B.data2B —
int infoD; // address D::infoD —
public:
 1- - -
 : —— .
};
```

Object layout

Object b

data 1B
data 2B

Object d

data 1B
data 2B
info D

This derived class D may add new member function.

Static Member function :-

Derived class doesn't inherit the static member functions of the base class.

Friend Function :-

Derived class doesn't inherit the friend function of base class.

- **Inheritance :-**

```
class B {
  public :
  void F(int i);
   void G( int i);
};
 class D : public B {
  public :
  // inherits B::f(int i) & inherits B::g (int i)
};
  B b; D d;
b. F(1); // calls B::f(int i)
b. G(2); // calls B::g(int i)
d. F(3); // calls B::F(int)
d. G(4); // calls B::g(int)

class B {
public:
void F(int); void g(int); };
                                  // Base class
```

```
class D:: Public B {
Public:
// inherits the B:: F(int) function
Void F(int); // overrides B:: F(int)
Void F(string); // overloads B::F(int)
// inherits B:: g(int) function
Void h(int i);
};
```

```
B b; D d;
b.f(1); // calls B:: f(int) function
b.g(2); // calls B:: g(int) function
d.f(3); // calls D:: f(int) function
d.g(4); // calls B:: g(int) function
d.f("red"); // calls D:: f(strings) function
d.h(5); // calls D:: h(int) function.
```

21/Sep/2022

Access Specification :-

① Derived class can't access the private members of base class.

② Derived class can access the protected member of base class.

- **Protected Access Specification :-**

i) A new protected access specification is introduced for base class.

ii) Derived class can access protected members of base class.

iii) No other class or global function can access the protected members of base class.

iv) A protected member in base class is like public in derived class.

v) A protected member in base class is like private in other classes or global functions.

```
class B {
private :                    // inaccessible to child class and others.
 int data : _ ;
public :
 void print () {
cout << " B object : " ;
cout << data : _ << endl ; }
};
class D : public B { int info _
 public :
void print () { cout << " D object : " ;
cout << data << " , " ;  // inaccessible
cout << info _ << endl ; }
};
   B b (0) ;
   D d (1,2);
b. data _ = 3;                // inaccessible to all.
b. print ();
d. print ();
```

Protected

```
class B {
protected :
int data__ ;
public :
void print () { cout << "Bobject : " ;
                cout << data__ << endl ; }
};
 class D : public B {
int info__ ;
public :
void print () { cout << "Dobject : " ;
                cout << data__ << ", " ;
                cout << info__ << endl ; }
};
        B b(0);
        D d(1,2);
        b.data__ = 5 ;
        b.print();
        d.print();
```

• Polymorphism :-

Q. What is polymorphism?

Ans: ♡ Polymorphism is the ability of the code or function to take more than one form.

♡ The behaviour depends on the types of data used in the operation. Eg. function overloading, operator overloading, virtual function

♡ There are two types of polymorphism one is compile time and other is run time.

• Compile time :-

i) It means that an object is bonded to its function call at compile time that is linking to function call to function definition is done at compile time.

ii) The function which is being called is known at the compile time

iii) This can be implemented using a pointer.

iv) This is also known as early binding or static binding. Eg. (overloading) function overloading, operator overloading.

• Runtime Polymorphism :-

i) It means that an object is bonded to its function call at runtime that is linking to function call to function definition is done at runtime.

ii) The function which is being get called is unknown until appropriate function selection is done.

iii) This can be implemented using a pointer.

iv) This is known as late binding or dynamic binding. (Overriding)

Eg. Virtual function.

Q. Difference between dynamic and static? Remember

- Compile time overloading can be implemented using (overloading)

ie, ① Function Overloading ⑪ Operator Overloading

Function Overloading :-

- This means that we can use a same function name to create function that performs variety of diff tasks.
- We can design multiple functions with one function name but different argument list.
- The function would perform different operations depending on the argument list in the function call.

Eg.
```
int add (int a, int b);
void add (int a, int b);
void add (int a, int b, int c);
void add (float a, float b, float c);
```
} Function with same name performs diff. operation depending on argument and return type.

Operator Overloading :-

- This means that giving a special meaning to an operator is known as operator overloading.

- It provides a flexible option for the creation of new definition for most of the C++ operators. We can overload all the C++ operators except the following.

i) Class members access operator
ii) Scope resolution operator
iii) Size operator
iv) Conditional operator.

- The semantics of an operator can be extended, we can't change its syntax.
- When an operator is overloaded, its original meaning is not lost.

Eg. The operator '+' when overloaded to add two vectors can still be used to add two integers.

04/Nov/2022

```
#include <iostream>
    using namespace std;
class B {
public:
        void f()
{ cout<<"B::f()"<<endl;
 }
   virtual void g(){
   cout<<"B::g()"<<endl; }
};
    class D:: public B {
public: void f(){
    cout<<"D::f()"<<endl;
```

```
    virtual void g()
  { cout << " D:: g() " << endl;
  }
};
    int main()
{   B b;
    D d;
  B* pb = &b;
  B* pd = &d;
  b.f();
  b.g();
  pb -> f();
  pb -> g();
  pd -> f();
  pd -> g();
  return 0;
}
```

09/Nov/2022

Q. What is template ?

Ans: Templates are specification of a collection of functions or classes which are parameterised by types.

Eg: i) Function search, min or max,

ⓐ The basic algorithm in these functions are the same independent of types.

(b) Yet. we need to write different versions of there functions for strong type checking in C++.

(ii) Classes, List, Queue.

(a) The data members and the methods are almost the same for list of numbers, list of objects.

(b) Yet, we need to define different classes.

Function Template :-

- A function template describe how a function should be built
- Supplies the definition of the function using some arbitary type
- Can be considered the definition for a set of overloaded version of a function.
- Is identified by the keyword "template"
- Note. that every template parameter is a built in type of class type parameter. Remember

Class template :-

(i) A class template describes how a class should be built.

(ii) Supplies the class description & the definition of the member function using some arbitary type name, is a parameterised type, parameterised member function.

(iii) Can be considered a definition for a unbounded set of class types.

(iv) Used for also container classes.

(v) Every template parameter is a built in type of class type parameter.

Q. Template can be considered as a kind of macro?

Ans: When an object of a specific type is defined for actual use, the template definition for the class is substituted with the reqd. data type. Since a template is defined with a parameter, that would be replaced by a specified data type at the time of actual use of the class or function, the templates are sometimes called parameterized classes or function.

- Overloaded function & function templates :-

A template function may be overloaded either by template function or ordinary function of its name. In such cases the overloading resolution is accomplished as follows —

i) Call an ordinary function that has an exact matching.

ii) Call an template function that could be created with an exact match.

iii) Try normal overloading resolution to ordinary function and call the one that matches.

An error is generated if no match is found. Note that, no automatic conversion are applied to arguments on the template function.

11/Nov/2022

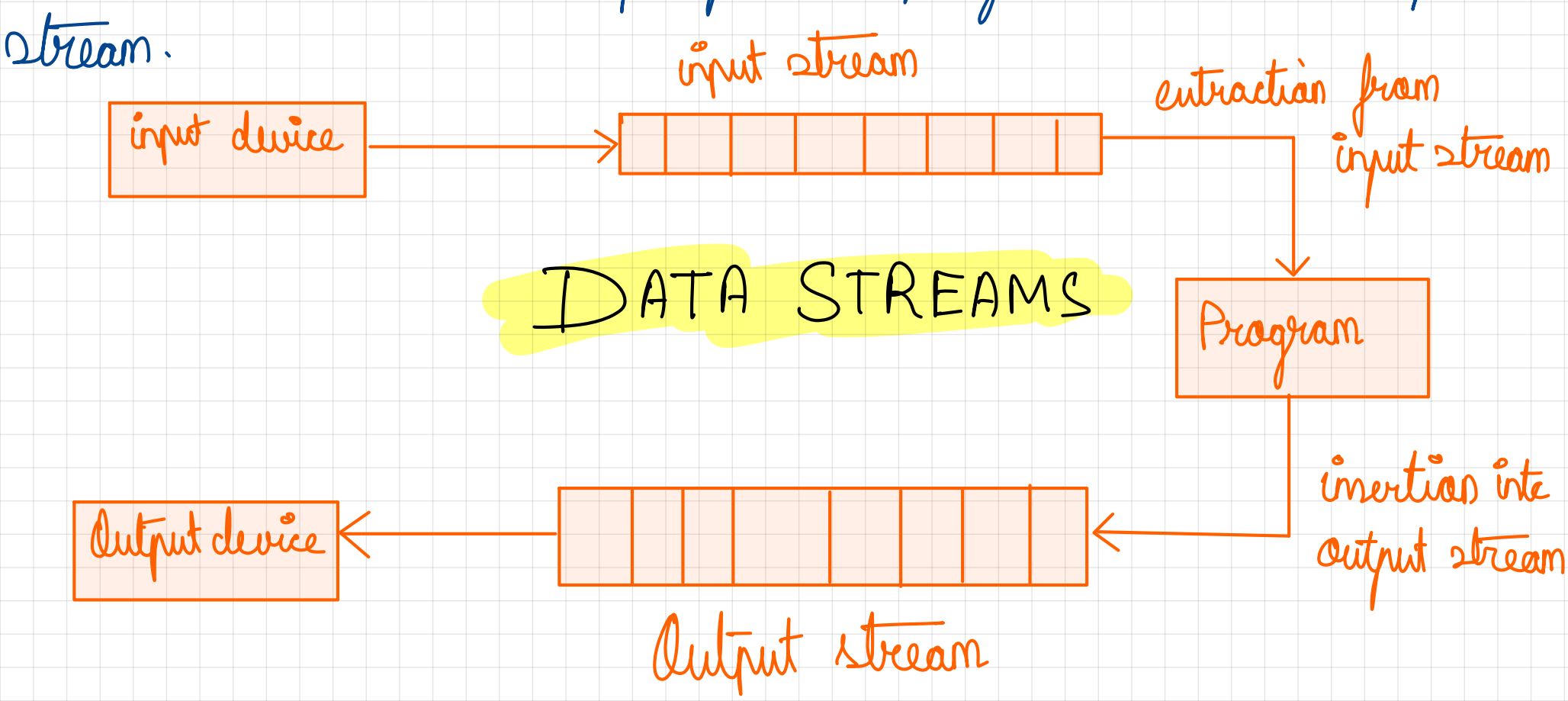
• Input / Output Operations :-

Every program take some data as input and generate procesed data as output. following the input output cycle.

C++ uses the concept of stream and stream classes to implement its I/O operation with the console and the disk file.

The I/O system in C++ is designed to work with a variety of devices including terminals, disks and tape drives. Although each device is very different, the I/O system supply an interface to the programmer, that is independent of actual device being accesed, this interface is known as stream.

A stream is a sequence of bytes. It acts in a either as a source from which the input data can be obtained. or as a destination to which the output data can be sent. The source stream that provides data to the program is called the input stream and the destination stream that recieves output from the program is called the output stream.



Cin represents the input stream connected to the standard input device (Keyboard) and the cout represent the output stream connected to the standard output device (screen).

- **C++ stream classes :-**

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

get function : The classes Istream & Ostream define two member function get() and put() respectively to handle the single character input output operation. There are 2 types of get function.

i) get(char) ii) get(void)