***OOP-CLASS***

CLASS is the core of OOP, it contains:

OOP is a technical allow us to describe an object in a particular way, detail attributes, and behavior. It has so benefits like encapsulation, inheritance,….. . we use class to use this technique.

* ***Class:***
* When we Declare are the object of a class, the object is like a copy of the class.
* We can use the function in a class by following:
* <return type> <class\_name>::<function(or other)>(){}
* ***Attribute:***

1. Function:

* Like it is named, the attribute is the list of the characteristic of each object.

1. Syntax:

Private:

//attribute;

//We can declare the attribute of the object here

Ex: Class SinhVien{

Private:

Int id, gpa;

String name, Class;

};

1. \*Notes:

* The Private can’t be accessed when we’re out of the scope of the class.

1. Static variable:

* Function:

It is a common variable of the class, so when we change the static variable in any object of the class, the variable also changes following this object.

* Syntax:

Class {

Private:

Static <type> variable;};

//we only assign value for variables when we’re out of the class.

<type> sv::variable=value;

Ex function:

Class test{

Private:

Static int dem;

Public:

Int get\_dem()

{

Return dem;

}

Void tangdem();

};

Int test::dem=0;

Void test::tangdem(){

dem++;

}

Int main(){

Test a;

a.Tangdem();

a.Tangdem();

test b;

cout<<b.get\_dem;

return 0;

}

Output: 2; (static variable is common variable)

* ***Behavior:***

1. Function:

* It can describe the behavior of an object.

1. Syntax:

Class….{

Public:

Function();

};

1. \*Notes:

* To use public out of class we do follow below:
* <return type> <class\_name>::<function(or other)>(){}

1. Each manipulator:

* Constructor:

+ Syntax:

Class sv{

Public:

Sv();

Sv(int ,int, string, string); (overloading)

};

sv::sv(){

cout<<”Hello”;

}

Sv::sv(int a, int b, string s, string s1){

a=b;

s=s1;….

}

When we call a constructor in main or function,the constructor will replay value follow type we declare.

* Destructor:

+ syntax:

Like a constructor but it is more than a tilde(~).

~sv(); (it can’t overloading)

sv::~sv(){};

When we’re at the end of the scope of the object of a class, the destructor will be called, it will destroy the constructor and complement code in the destructor.

* Setter, getter:

We know that we can’t access directly private in class, so we can write a function to get a value of attributes in class.

Vd:

Private:

Int a;

Main: cout<<a; ->wrong, we can’t make it

Instead, we can :

Private:

Int a;

Public:

Int get\_a(int a){

Return a;

} //write a function directly in the class.

Main: cout<<class.get\_a; -> right.

* ***This Pointer:***

1. Function:

* We use it to identify an attribute in class.

1. Syntax:

This->x;

EX:

Class ex{

Private:

Int x;

Public:

Void show(int);

};

Void ex::show(int x)

{

This->x=x;

}

Int main(){

ex a;

ex.show(5);

}

* Pointer “this” point member function in class.
* ***Friend function, friend class:***

1. Function:

* We can use the friend function or friend class to access the attribute, behavior, or method from 1 Class to other class, 1 function to other class and they are not related.

1. Syntax:

Class a{

//friend class

Friend class b;

Private:

Int dem;

Public:

//friend function

Friend <return\_type> name(parameter);

}

Class b{

Public:

<return\_type> name(a&);

};

Ex:

Class sinhvien{

Friend class giaovien;

Private:

Int gpa=2, dem=0;

Public:

Friend void show(int);

};

Class giaovien{

Public:

Void change(sinhvien&);

};

Void giaovien::change(sinhvien &a){

a.gpa=3.2;

int get\_gpa(){

return gpa;

}

}

Void show(int dem){

Cout<<dem<<” “;

}

Int main(){

Sinhvien a;

Giaovien a1;

a1.change(a);

show();

cout<<get\_gpa;

return 0;

}

//3.2 is GPA of class giaovien, 0 is dem of class sinhvien

Output: 3.2 0

* ***Operator overloading:***

1. Function:

* To use the operator among between class.

Ex: class sophuc{};

Main: sophuc a,b,c;

c= a+b; -> invalid because 1 class have more attribute, so we can’t use operator “+”.

* To solution above case, we have to use function overloading.

1. Syntax:

Overloading need to be seft-defined, so we not have a default form.

Each EX:

Operator “+”,>>, <<:

Class sophuc{

Private:

Int a,b;

Public:

Sophuc operator +(sophuc another)

{

Sophuc tong;

Tong.a=a+another.a;

Tong.b=b+another.b;

Return tong;

}

(//Or we use friend function

Friend sophuc operator +(sophuc a, sophuc b)

{

Sophuc Tong;

Tong.a=a.a+b.a;

Tong.b=a.b+b.b;

Return Tong;

} )

//insertion

Friend istream& operator >>(istream& in, sophuc m)

{

In>>m.a>>m.b;

Return in;

};

//extraction

Friend ostream& operator <<(ostream out, sophuc m)

{

out<<m.a<<” “<<m.b;

return out;

}

} ;  
similarly, we can use operator overloading to seft-defined operator your need.