# 引用

int a = 9;

int & b = a;

引用指的是其他变量的别名。

cout<< b <<endl;

b++;

cout<<a<<endl;

引用一旦决定了是那个变量的别名，就不会再代表其他变量的别名了。

注意：引用在定义的时候必须要初始化。

引用有什么好处？

通常都是和函数的形参联合在一起使用的。

用引用作为函数的形参。

1 可以通过引用形参修改外部实参的值。

写一个函数，交换两个变量的值。

从而在一定程度上取代了原来需要指针才能完成的工作，提供了代码的简洁性和安全性。

2 和用普通变量作为函数参数相比，用引用作为函数参数可以减少因为函数调用所带来的系统开销。当传递的参数所占用的内存空间较大时，一般都用引用的方式来传递。

3 引用可以作为函数的返回值类型吗？

4 引用的缺点：它太灵活了，对函数的封装性有一定的破坏。

# 类和对象

# 相关的概念

抽象：通过分析问题，把要解决的问题转换成可以用C或者C++代码实现的数据结构或程序框架。

封装：C语言中的结构体就体现了封装的概念。把相关的数据集成在了一起，形成了一个整体。数据上的封装

面向对象中的封装，不但可以封装数据，还可以封装行为。在类里面函数，用函数模拟类所具有的行为。

class Student{

public:

int id;

int age;

int score;

void eat(){

cout<<"去吃饭"<<endl;

}

};

继承：

目的是实现代码的重用。

多态：

# 类和对象

类 class

可以定义一种新的类型，实现了对数据和行为的封装，可以用于模拟一些真实的事物，比如学生类。

对象

指通过类类型定义个变量，称为对象，也称为对象实例，对象个体。

学生（类） 一个具体的学生 张三（学生对象）

1类定义的语法

class Student{

public:

int id;

int age;

int score;

void eat(){

cout<<"去吃饭"<<endl;

}

};

int main(){

//struct Student s;

Student s;

s.id = 7;

s.age = 20;

cout<<s.id<<" "<<s.age<<endl;

s.eat();

}

2 对类中成员的访问

通常都是通过具体的对象来进行的。

3 关于成员函数的细节

成员函数调用中的目的对象。指对成员函数发起调用的那个对象。

成员函数可以带有默认形参值，也可以重载。

成员函数也可以是内联函数。

成员函数的定义方式:

1 定义在类的里面（简单）

2 定义在类的外面，注意仍然需要在类里面进行声明

Student:: 类作用域限定

好处：可以更快的看到类的总体结构。

对类成员的访问控制

在类的外部，是否可以对类成员进行访问？

成员的访问属性

public 修饰 可以在类的外部被访问

private修饰 不可以在类的外部被访问

protected 修饰 不可以在类的外部被访问

默认的访问属性 是 private

类里面成员的分类

数据成员

函数成员

在成员函数中访问类的其他成员是没有限制的，因为他们是一个整体了。

#include <iostream>

using namespace std;

void f(int &a){

a++;

}

void swap(int &a,int &b){

int t = a;

a = b;

b = t;

}

void swap(int \*pa, int \*pb){

int t = \*pa;

\*pa = \*pb;

\*pb = t;

}

/\*

int main(){

int a = 9, b = 10;

//swap(a,b);

swap(&a,&b);

cout<<a<<" "<<b<<endl;

}

\*/

/\*

int main(){

int a = 9;

f(a);

cout<<a<<endl;

}

\*/

/\*

struct Student{

int id;

int age;

int score;

};

\*/

class Student{

public:

int id;

int age;

int score;

void eat();

// void eat(){

// cout<<"去吃饭"<<endl;

// }

private:

int money;

};

void Student::eat(){

if (money>100)

cout<<id<<" 去馆子"<<endl;

else

cout<<id<<" 去食堂"<<endl;

}

int main(){

//struct Student s;

Student s;

s.id = 7;

s.age = 20;

cout<<s.id<<" "<<s.age<<endl;

//s.money = 120;

s.eat();

//Student.eat();

//s.money = 100;

}