1 C++入门

> C++ 读作“C加加”，是“C Plus Plus”的简称。顾名思义，C++ 是在C语言的基础上增加新特性，玩出了新花样，所以叫“C Plus Plus”，就像 iPhone 12 Plus 和 iPhone 11、Win10 和 Win7 的关系。

从语法上看，C语言是 C++ 的一部分，C语言代码几乎不用修改就能够以 C++ 的方式编译，这给很多初学者带来了不小的困惑，学习 C++ 之前到底要不要先学习C语言呢？

不过可以明确地说：学了C语言就相当于学了 C++ 的一半，从C语言转向 C++ 时，不需要再从头开始，接着C语言往下学就可以，所以码牛强烈建议先学C语言再学 C++。

1.1 C++和C语言的血缘关系

> 现在看来，C++ 和C语言虽然是两门独立的语言，但是它们却有着扯也扯不清的关系。

早期并没有“C++”这个名字，而是叫做“带类的C”。“带类的C”是作为C语言的一个扩展和补充出现的，它增加了很多新的语法，目的是提高开发效率，如果你有 Java Web 开发经验，那么你可以将它们的关系与 Servlet 和 JSP 的关系类比。

这个时期的 C++ 非常粗糙，仅支持简单的面向对象编程，也没有自己的编译器，而是通过一个预处理程序（名字叫 cfront），先将 C++ 代码“翻译”为C语言代码，再通过C语言编译器合成最终的程序。

随着 C++ 的流行，它的语法也越来越强大，已经能够很完善的支持面向过程编程、面向对象编程（OOP）和泛型编程，几乎成了一门独立的语言，拥有了自己的编译方式。

我们很难说 C++ 拥有独立的编译器，例如 Windows 下的微软编译器（cl.exe）、Linux 下的 GCC 编译器、Mac 下的 Clang 编译器（已经是 Xcode 默认编译器，雄心勃勃，立志超越 GCC），它们都同时支持C语言和 C++，统称为 C/C++ 编译器。对于C语言代码，它们按照C语言的方式来编译；对于 C++ 代码，就按照 C++ 的方式编译。

从表面上看，C、C++ 代码使用同一个编译器来编译，所以上面我们说“后期的 C++ 拥有了自己的编译方式”，而没有说“C++ 拥有了独立的编译器”。

1.2 C++类和对象到底是什么意思？

C++ 是一门面向对象的编程语言，理解 C++，首先要理解**类（Class）**和**对象（Object）**这两个概念。

C++ 中的类（Class）可以看做C语言中结构体（Struct）的升级版。结构体是一种构造类型，

可以包含若干成员变量，每个成员变量的类型可以不同；可以通过结构体来定义结构体变量，每个变量拥有相同的性质。例如：

```
#include <stdio.h>
//定义结构体 Student
struct Student{
    //结构体包含的成员变量
    char *name;
    int age;
    float score;
};
//显示结构体的成员变量
void display(struct Student stu){
    printf("%s的年龄是 %d，成绩是 %f\n", stu.name, stu.age, stu.score);
}
```

```
int main(){
    struct Student stu1;
    //为结构体的成员变量赋值
    stu1.name = "小明";
    stu1.age = 15;
    stu1.score = 92.5;
    //调用函数
    display(stu1);
    return 0;
}
```

运行结果：
小明的年龄是 **15**，成绩是 **92.500000**

1.3 C++ 中的类也是一种构造类型

同时也对C++进行了一些扩展，类的成员不但可以是变量，还可以是函数；

通过类定义出来的变量也有特定的称呼，叫做“对象”。例如：

```
#include <stdio.h>
//通过class关键字类定义类
class Student{
public:
    //类包含的变量
    char *name;
    int age;
    float score;
    //类包含的函数
    void say(){
        printf("%s的年龄是 %d，成绩是 %f\n", name, age, score);
    }
};
int main(){
    //通过类来定义变量，即创建对象
    class Student stu1;  //也可以省略关键字class
    //为类的成员变量赋值
    stu1.name = "小明";
    stu1.age = 15;
    stu1.score = 92.5f;
    //调用类的成员函数
    stu1.say();
    return 0;
}
```

运行结果与上例相同。

> class 和 public 都是 C++ 中的关键字，初学者请先忽略 public（后续会深入讲解），把注意力集中在 class 上。

c和c++的区别

> 1. C语言中的 struct 只能包含变量，
> 2. 而 C++ 中的 class 除了可以包含变量，还可以包含函数。display() 是用来处理成员变量的函数，在C语言中，我们将它放在了 struct Student 外面，它和成员变量是分离的；

> 3. 而在 C++ 中，我们将它放在了 class Student 内部，使它和成员变量聚集在一起，看起来更像一个整体。

结构体和类都可以看做一种由用户自己定义的复杂数据类型，

在C语言中可以通过结构体名来定义变量，在 C++ 中可以通过类名来定义变量。

不同的是：

> 通过结构体定义出来的变量还是叫变量，
>
> 而通过类定义出来的变量有了新的名称，叫做对象（Object）。

在第二段代码中，我们先通过 class 关键字定义了一个类 Student，然后又通过 Student 类创建了一个对象 stu1。变量和函数都是类的成员，创建对象后就可以通过点号`.`来使用它们。

可以将类比喻成图纸，对象比喻成零件，图纸说明了零件的参数（成员变量）及其承担的任务（成员函数）；

一张图纸可以生产出多个具有相同性质的零件，不同图纸可以生产不同类型的零件。

重点:

类只是一张图纸，起到说明的作用，不占用内存空间；

对象才是具体的零件，要有地方来存放，才会占用内存空间。

在 C++ 中，通过类名就可以创建对象，即将图纸生产成零件，这个过程叫做类的实例化，因此也称对象是类的一个实例（Instance）。

有些资料也将类的成员变量称为属性（Property），将类的成员函数称为方法（Method）。

1.4 面向对象编程（Object Oriented Programming，OOP）

类是一个通用的概念，C++、Java、C#、PHP 等很多编程语言中都支持类，都可以通过类创建对象。可以将类看做是结构体的升级版，C语言的晚辈们看到了C语言的不足，尝试加以改善，继承了结构体的思想，并进行了升级，让程序员在开发或扩展大中型项目时更加容易。

因为 C++、Java、C#、PHP 等语言都支持类和对象，所以使用这些语言编写程序也被称为面向对象编程，这些语言也被称为面向对象的编程语言。C语言因为不支持类和对象的概念，被称为面向过程的编程语言。

在C语言中，我们会把重复使用或具有某项功能的代码封装成一个函数，将拥有相关功能的多个函数放在一个源文件，再提供一个对应的头文件，这就是一个模块。使用模块时，引入对应的头文件就可以。

而在 C++ 中，多了一层封装，就是类（Class）。类由一组相关联的函数、变量组成，你可以将一个类或多个类放在一个源文件，使用时引入对应的类就可以。下面是C和C++项目组织方式的对比：

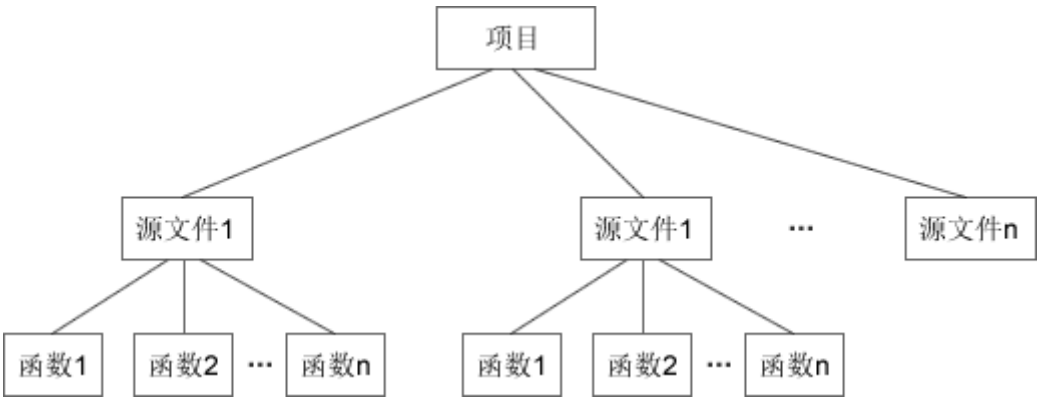


图1：C语言中项目的组织方式

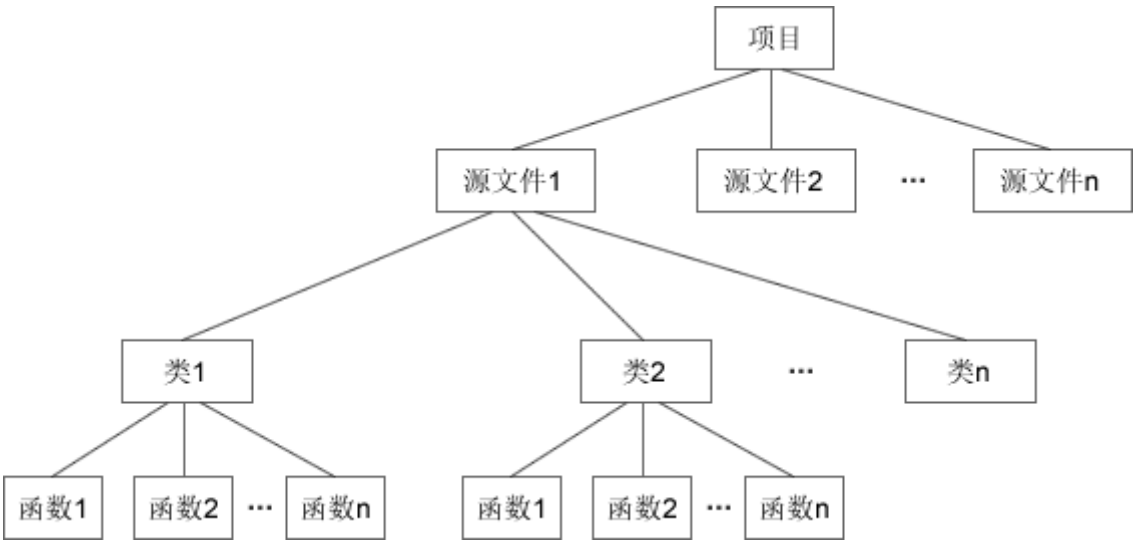


图2：C++中项目的组织方式

不要小看类（Class）这一层封装，它有很多特性，极大地方便了中大型程序的开发，它让 C++ 成为面向对象的语言。

面向对象编程在代码执行效率上绝对没有任何优势，它的主要目的是方便程序员组织和管理代码，快速梳理编程思路，带来编程思想上的革新。

面向对象编程是针对开发中大规模的程序而提出来的，目的是提高软件开发的效率。不要把面向对象和面向过程对立起来，面向对象和面向过程不是矛盾的，而是各有用途、互为补充的。如果你希望开发一个贪吃蛇游戏，类和对象或许是多余的，几个函数就可以搞定；但如果开发一款大型游戏，那你绝对离不开面向对象。

2 命名空间

> 一个中大型软件往往由多名程序员共同开发，会使用大量的变量和函数，不可避免地会出现变量或函数的命名冲突。当所有人的代码都测试通过，没有问题时，将它们结合到一起就有可能会出现命名冲突。

例如　小李和小韩都参与了一个文件管理系统的开发，它们都定义了一个全局变量 fp，用来指明当前打开的文件，将他们的代码整合在一起编译时，很明显编译器会提示 fp 重复定义（Redefinition）错误。

为了解决合作开发时的命名冲突问题，C++ 引入了**命名空间（Namespace）**的概念。请看下面的例子：

小李与小韩各自定义了以自己姓氏为名的命名空间，此时再将他们的 fp 变量放在一起编译就不会有任何问题。

> 命名空间有时也被称为名字空间、名称空间。

namespace 是C++中的关键字，用来定义一个命名空间，语法格式为：

```
namespace name{
 //variables, functions, classes
}
```

`name` 是命名空间的名字，它里面可以包含变量、函数、类、typedef、#define 等，最后由 `{ }` 包围。

使用变量、函数时要指明它们所在的命名空间。以上面的 fp 变量为例，可以这样来使用：

```
Li::fp = fopen("one.txt", "r");  //使用小李定义的变量 fpHan::fp = fopen("two.txt", "rb+");  //使用小韩定义的变量 fp
```

`::` 是一个新符号，称为域解析操作符，在C++中用来指明要使用的命名空间。

除了直接使用域解析操作符，还可以采用 using 关键字声明，例如：

```
using Li::fp;fp = fopen("one.txt", "r");  //使用小李定义的变量 fpHan :: fp = fopen("two.txt", "rb+");  //使用小韩定义的变量 fp
```

在代码的开头用 `using` 声明了 Li::fp，它的意思是，using 声明以后的程序中如果出现了未指明命名空间的 fp，就使用 Li::fp；但是若要使用小韩定义的 fp，仍然需要 Han::fp。

using 声明不仅可以针对命名空间中的一个变量，也可以用于声明整个命名空间，例如：

```
using namespace Li;fp = fopen("one.txt", "r");  //使用小李定义的变量 fpHan::fp = fopen("two.txt", "rb+");  //使用小韩定义的变量 fp
```

如果命名空间 Li 中还定义了其他的变量，那么同样具有 fp 变量的效果。在 using 声明后，如果有未具体指定命名空间的变量产生了命名冲突，那么默认采用命名空间 Li 中的变量。

命名空间内部不仅可以声明或定义变量，对于其它能在命名空间以外声明或定义的名称，同样也都能在命名空间内部进行声明或定义，例如类、函数、typedef、#define 等都可以出现在命名空间中。

站在编译和链接的角度，代码中出现的变量名、函数名、类名等都是一种符号（Symbol）。有的符号可以指代一个内存位置，例如变量名、函数名；有的符号仅仅是一个新的名称，例如 typedef 定义的类型别名。

下面来看一个命名空间完整示例代码：

```
#include <stdio.h>//将类定义在命名空间中namespace Diy{    class Student{    public:        char *name;        int age;        float score;      public:        void say(){            printf("%s的年龄是 %d，成绩是 %f\n", name, age, score);        }   };}int main(){    Diy::Student stu1;    stu1.name = "小明";    stu1.age = 15;    stu1.score = 92.5f;    stu1.say();    return 0;}
```

运行结果：
小明的年龄是 15，成绩是 92.500000

3 cin和cout简介

定义:

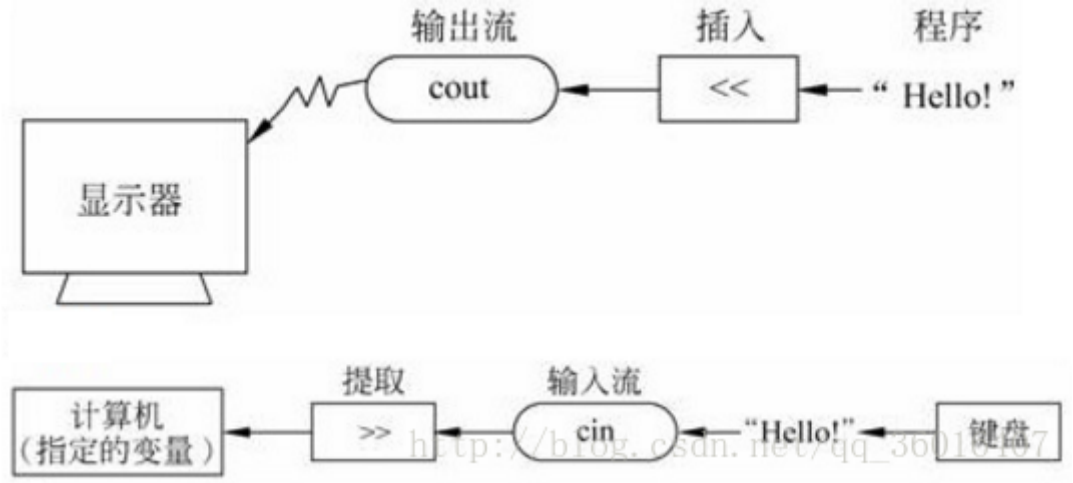
1. cin是C++编程语言中的标准输入流对象，即**istream类的对象**。cin主要用于从标准输入读取数据，这里的标准输入，指的是终端或键盘。
2. cout是流的对象，即**ostream类的对象**
3. cerr是标准错误输出流的对象，也是ostream 类的对象。这里的标准输出指的是终端键盘，标准错误输出指的是终端的屏幕。

   在理解cin功能时，不得不提标准输入缓冲区。当我们从键盘输入字符串的时候需要敲一下回车键才能够将这个字符串送入到缓冲区中，

那么敲入的这个回车键(\r)会被转换为一个换行符\n，这个换行符\n也会被存储在cin的缓冲区中并且被当成一个字符来计算!

> 比如我们在键盘上敲下了123456这个字符串，然后敲一下回车键（\r）将这个字符串送入了缓冲区中，那么此时缓冲区中的字节个数是7 ，而不是6。

cin读取数据也是从缓冲区中获取数据，缓冲区为空时，cin的成员函数会阻塞等待数据的到来，一旦缓冲区中有数据，就触发cin的成员函数去读取数据。



有关流对象cin、cout和流运算符的定义等信息是存放在C++的输入输出流库中的，因此如果在程序中使用cin、cout和流运算符，就必须使用预处理命令把头文件iostream包含到本文件中，并使用命名空间std：

1.2.cin和cout的基本操作

cout表达式

> cout语句的一般格式为：
>
> cout<<表达式1<<表达式2<<表达式3...<<表达式n;

cin表达式

> cin语句的一般格式为：
>
> cin>>变量1>>变量2>>变量3>>.....>>变量n;

1.2.1count 操作

一个cout语句可以分成若干行

```
    cout<<"this is"  //注意没有分号
        <<"a C++"
        <<"program."
        <<endl;
```

或者这样写 也是等效的

```
    cout<<"this is ";   //有分号
    cout<<"a C++";
    cout<<"program.";
    cout<<endl;
```

输出结果差不多。

不能用一个插入运算符“<<”插入多个输出项：

1.2.2 在用cout输出时，用户不必通知计算机按何种类型输出

> 系统会自动判别输出数据的类型，使输出的数据按相应的类型输出。如已定义a为int型，b为float型，c为char型，则：cout<<a<<' '<<b<<' '<<c<<endl;

```
cin：与cout类似，cin语句可以分成若干行

1)cin>>a>>b>>c>>d;
```

```
    cin>>a  //这样的写法比较清晰
       >>b
       >>c
       >>d;
```

```
    cin>>a;
    cin>>b;
    cin>>c;
```

这几种写法都是一样的效果

1.2.3cin输入自动截取长度

在用cin输入时,系统也会根据变量的类型从输入流中提取相应长度的字节。

如有：

```
char c1,c2;int a;

float b;

cin>>c1>>c2>>a>>b;
```

不能用cin语句把空格字符和回车换行符作为字符输入给字符变量，他们将被跳过。

最后总结，

> C++的iostream库和C中的stdio库中分别的cout/cin和printf/scanf相比有哪些优势呢?
>
> 首先是类型处理更加安全，更加智能
>
> 我们无须应对int、float中的%d、%f，而且扩展性极强，对于新定义的类，
>
> printf想要输入输出一个自定义的类的成员是天方夜谭的，
>
> 而iostream中使用的位运算符都是可重载的，并且可以将清空缓冲区的自由交给了用户（在printf中的输出是没有缓冲区的），
>
> 而且流风格的写法也更加自然和简洁

4 引用的概念

4.1 引用定义

引用：就是某一变量（目标）的一个别名，对引用的操作与对变量直接操作完全一样。

引用的声明方法：类型标识符 &引用名=目标变量名；

如下：定义引用ra，它是变量a的引用，即别名。

```
int a;

int &ra=a;
```

> （1）&在此不是求地址运算符，而是起标识作用。
>
> （2）声明引用时，必须同时对其进行初始化。
>
> （3）类型标识符是指目标变量的类型。
>
> （4）引用声明完毕后，相当于目标变量有两个名称即该目标原名称和引用名，且不能再把该引用名作为其他变量名的别名。
>
> （5）声明一个引用，不是新定义了一个变量，它只表示该引用名是目标变量名的一个别名，它本身不是一种数据类型，因此引用本身不占存储单元，系统也不给引用分配存储单元。故：对引用求地址，就是对目标变量求地址。&ra与&a相等。
>
> （6）不能建立数组的引用。因为数组是一个由若干个元素所组成的集合，所以无法建立一个数组的别名。

```
#include<iostream.h>
void main(){
    int a=5;
    int &b=a;
    b=6;
    cout<<"a="<<a<<",b="<<b<<endl;//a=6,b=6
    int c=7;
    b=c;
    cout<<"a="<<a<<",b="<<b<<endl;//a=7,b=7
}
```

```
#include<iostream.h>
void main(){
    int a[]={1,2,3,4};
    int &b=a;
    //编译错误：cannot convert from 'int [4]' to 'int &'
}
```

4.2 引用作为参数

引用的一个重要作用就是作为函数的参数。以前的C语言中函数参数传递是值传递，如果有大块数据作为参数传递的时候，采用的方案往往是指针，因为这样可以避免将整块数据全部压栈，可以提高程序的效率。但是现在（C++中）又增加了一种同样有效率的选择（在某些特殊情况下又是必须的选择），就是引用。

```
#include<iostream.h>
////此处函数的形参p1, p2都是引用
void swap(int &p1,int &p2){
    int p=p1;
    p1=p2;
    p2=p;
}
```

为在程序中调用该函数，则相应的主调函数的调用点处，直接以变量作为实参进行调用即可，而不需要实参变量有任何的特殊要求。如：对应上面定义的swap函数，相应的主调函数可写为：

```
void main(){
    int a,b;
   cin>>a>>b;//输入a,b两个变量的值
   swap(a,b);//直接以a和b作为实参调用swap函数
   cout<<"a="<<a<<",b="<<b<<endl;
}
```

上述程序运行时，如果输入数据10 20并回车后，则输出结果为a=20,b=10。

由上例可以看出：

（1）传递引用给函数与传递指针的效果是一样的。这时，被调函数的形参就成为原来主调函数中的实参变量或对象的一个别名来使用，所以在被调函数中对形参变量的操作就是对其相应的目标对象（在主调函数中）的操作。

（2）使用引用传递函数的参数，在内存中并没有产生实参的副本，它是直接对实参操作；而使用一般变量传递函数的参数，当发生函数调用时，需要给形参分配存储单元，形参变量是实参变量的副本；

如果传递的是对象，还将调用拷贝构造函数。因此，当参数传递的数据较大时，用引用比用一般变量传递参数的效率和所占空间都好。

（3）使用指针作为函数的参数虽然也能达到与使用引用的效果，但是，在被调函数中同样要给形参**分配存储单元**，且需要**重复使用"*指针变量名**"的形式进行运算，这很容易产生错误且程序的阅读性较差；另一方面，在主调函数的调用点处，必须用变量的地址作为实参。而引用更容易使用，更清晰。

如果既要利用引用提高程序的效率，又要保护传递给函数的数据不在函数中被改变，就应使用常引用。

4.3 常引用

> **定义：** 常引用声明方式：const 类型标识符 &引用名 = 目标变量名；

用这种方式声明的引用，不能通过引用对目标变量的值进行修改,从而使引用的目标成为const，达到了引用的安全性。

```
#include<iostream.h>
void main(){
    int a=1;
    int &b=a;
    b=2;
    cout<<"a="<<a<<endl;//2
    int c=1;
    const int &d=c;
//    d=2;//编译错误 error C2166: l-value specifies const object
    c=2;//正确
}
```

这不光是让代码更健壮，也有其它方面的需求。

【例4】：假设有如下函数声明：

string foo();

void bar(string &s);

那么下面的表达式将是非法的：

bar(foo());

bar("hello world");

原因在于foo( )和"hello world"串都会产生一个临时对象，**而在C++中，临时对象都是const类型的。**因此上面的表达式就是试图将一个const类型的对象转换为非const类型，这是非法的。

引用型参数应该在能被定义为const的情况下，尽量定义为const 。

3、引用作为返回值

要以引用返回函数值，则函数定义时要按以下格式：

类型标识符 &函数名 （形参列表及类型说明）

{ 函数体 }

说明：

（1）以引用返回函数值，定义函数时需要在函数名前加&

（2）用引用返回一个函数值的最大好处是，在内存中不产生被返回值的副本。

【例5】以下程序中定义了一个普通的函数fn1（它用返回值的方法返回函数值），另外一个函数fn2，它以引用的方法返回函数值。

```
#include<iostream.h>
float temp;//定义全局变量temp
float fn1(float r);//声明函数fn1
float &fn2(float r);//声明函数fn2 r
float fn1(float r){//定义函数fn1，它以返回值的方法返回函数值
    temp=(float)(r*r*3.14);
    return temp;
}
float &fn2(float r){//定义函数fn2，它以引用方式返回函数值
    temp=(float)(r*r*3.14);
    return temp;
}
void main(){
    float a=fn1(10.0);//第1种情况，系统生成要返回值的副本（即临时变量）
//    float &b=fn1(10.0); //第2种情况，可能会出错（不同 C++系统有不同规定）
/*
    编译错误：cannot convert from 'float' to 'float &'
    A reference that is not to 'const' cannot be bound to a non-lvalue
*/
    //不能从被调函数中返回一个临时变量或局部变量的引用
    float c=fn2(10.0);//第3种情况，系统不生成返回值的副本
    //可以从被调函数中返回一个全局变量的引用
    float &d=fn2(10.0); //第4种情况，系统不生成返回值的副本
    cout<<"a="<<a<<",c="<<c<<",d="<<d<<endl;
    //a=314,c=314,d=314
}
```

引用作为返回值，必须遵守以下规则：

（1）**不能返回局部变量的引用。**这条可以参照Effective C++[1]的Item 31。**主要原因是局部变量会在函数返回后被销毁，因此被返回的引用就成为了"无所指"的引用，程序会进入未知状态。**如【例5】中的第2种情况出现编译错误。

（2）**不能返回函数内部new分配的内存的引用。**这条可以参照Effective C++[1]的Item 31。虽然不存在局部变量的被动销毁问题，可对于这种情况（返回函数内部new分配内存的引用），又面临其它尴尬局面。例如，被函数返回的引用只是作为一个临时变量出现，而没有被赋予一个实际的变量，那么这个引用所指向的空间（由new分配）就无法释放，造成memory leak。

（3）**可以返回类成员的引用，但最好是const。**这条原则可以参照Effective C++[1]的Item 30。主要原因是当对象的属性是与某种业务规则（business rule）相关联的时候，其赋值常常与某些其它属性或者对象的状态有关，因此有必要将赋值操作封装在一个业务规则当中。如果其它对象可以获得该属性的非常量引用（或指针），那么对该属性的单纯赋值就会破坏业务规则的完整性。

（4）引用与一些操作符的重载：流操作符<<和>>，这两个操作符常常希望被连续使用，例如：cout << "hello" << endl; 因此这两个操作符的返回值应该是一个仍然支持这两个操作符的流引用。可选的其它方案包括：返回一个流对象和返回一个流对象指针。但是对于返回一个流对象，程序必须重新（拷贝）构造一个新的流对象，也就是说，连续的两个<<操作符实际上是针对不同对象的！这无法让人接受。对于返回一个流指针则不能连续使用<<操作符。因此，返回一个流对象引用是惟一选择。这个唯一选择很关键，它说明了引用的重要性以及无可替代性，也许这就是C++语言中引入引用这个概念的原因吧。 赋值操作符=。这个操作符象流操作符一样，是可以连续使用的，例如：x = j = 10;或者(x=10)=100;赋值操作符的返回值必须是一个左值，以便可以被继续赋值。因此引用成了这个操作符的惟一返回值选择。

【例6】 测试用返回引用的函数值作为赋值表达式的左值。

```
#include<iostream.h>
int &put(int n);
int vals[10];
int error=-1;
void main(){
    put(0)=10;//以put(0)函数值作为左值，等价于vals[0]=10;
    put(9)=20;//以put(9)函数值作为左值，等价于vals[9]=20;
    cout<<vals[0]<<endl;//10
    cout<<vals[9]<<endl;//20
}
int &put(int n){
    if(n>=0 && n<=9)
        return vals[n];
    else{
        cout<<"subscript error";
        return error;
    }
}
```

（5）在另外的一些操作符中，却千万不能返回引用：+-*/ 四则运算符。它们不能返回引用，Effective C++[1]的Item23详细的讨论了这个问题。主要原因是这四个操作符没有side effect，因此，它们必须构造一个对象作为返回值，可选的方案包括：返回一个对象、返回一个局部变量的引用，返回一个new分配的对象的引用、返回一个静态对象引用。根据前面提到的引用作为返回值的三个规则，第2、3两个方案都被否决了。静态对象的引用又因为((a+b) == (c+d))会永远为true而导致错误。所以可选的只剩下返回一个对象了。

4、引用和多态

引用是除指针外另一个可以产生多态效果的手段。这意味着，一个基类的引用可以指向它的派生类实例。

【例7】：

class A;

class B:public A{ … … }

B b;

A &Ref = b;//用派生类对象初始化基类对象的引用

Ref 只能用来访问派生类对象中从基类继承下来的成员，是基类引用指向派生类。如果A类中定义有虚函数，并且在B类中重写了这个虚函数，就可以通过Ref产生多态效果。

引用总结

（1）在引用的使用中，单纯给某个变量取个别名是毫无意义的，**引用的目的主要用于在函数参数传递中，解决大块数据或对象的传递效率和空间不如意的问题。**

（2）用引用传递函数的参数，能保证参数传递中不产生副本，提高传递的效率，且通过const的使用，保证了引用传递的安全性。

（3）引用与指针的区别是，指针通过某个指针变量指向一个对象后，对它所指向的变量间接操作。程序中使用指针，程序的可读性差；而引用本身就是目标变量的别名，对引用的操作就是对目标变量的操作。

（4）使用引用的时机。流操作符<<和>>、赋值操作符=的返回值、拷贝构造函数的参数、赋值操作符=的参数、其它情况都推荐使用引用。