单选题 1分

设置

关于C++主函数特性，下列叙述正确的是（ ）

A 主函数在同一个C++文件中可以有两个

B 主函数类型必须是void类型

C 主函数一定有返回值

D 每个C++程序都必须有一个main()函数

提交

调试以下程序，并改正错误。

```
Using namespace std
#include<iostream>;
Using std::endl;
int main()
float num1,num2,num3;
cin<<num1<<num2<<num3;
cout>>“The multiplication is”>>setw(30)>> num1*num2*num3>>endl;
}
```

1. 关键字Using中包含了大写字母，应改为using；
2. using namespace std不是预处理指令，要以分号结尾；
3. #include<iostream>;是预处理指令，不能以分号结尾，而且要作为程序的开头行；
4. 使用了using namespace std，就不必单独使用std::endl;；
5. int main()后缺少左括号{；
6. cin通常与提取操作符>>连用； cout通常与提取操作符<<连用；
7. “The multiplication is”为中文引号，须改为英文引号；
8. 使用setw(30)必须包含头文件iomanip

C++

桂林電子科技大学
GUILIN UNIVERSITY OF ELECTRONIC TECHNOLOGY

人工智能学院
SCHOOL OF ARTIFICIAL INTELLIGENCE

第三讲 函数

赵彬

zhaobin@guet.edu.cn

3.1 函数的概念作用

函数（function）代表一个功能。在一个程序文件中，只能有一个主函数（main），主函数作为程序的入口，在程序运行时调用其他函数模块，其他函数模块之间可以相互调用。在实际开发过程中，主函数等同于总调度中心，调动每个函数进行实现所需的功能要求。

软件开发者和开发商通常将一些通用的功能写成函数，将函数放置函数库中供人使用，如iostream库，包含了多个常见的输入/输出函数。

3.1 函数的概念和作用

随着需求和性能的提升，个人难以支撑大型程序的开发工作量，更无法将所有的内容放到一个主函数中。函数由此应运而生，将一个大型程序划分成若干个程序模块，每个程序模块完成一部分功能的实现。

- 将不同的程序模块分配给不同的人进行开发，这样有效地提高了编程和调试的效率
- 程序模块为编译单位，在程序进行编译时，即分别对每一个程序模块进行编译
- 当发现编译错误时，开发者能准确找到错误程序模块，在模块范围内进行纠错处理

3.2 函数的分类

从用户使用的角度，函数分为：系统函数和自定义函数

① 系统函数也称为库函数，由编译系统提供，用户无需定义，可以直接调用。

② 自定义函数，即用户自己定义的函数，用于解决用户特定的需求

3.2 函数的分类

从函数的形式角度，函数分为：无参函数和带参函数

① 无参函数，调用时无需给出参数

② 带参函数，主调函数调用其他函数时需要把数据传递给被调函数

3.2 函数的分类

- **无参函数的格式**

```
类型标识符 函数名称（[void]）
{
        声明部分（语句）
}
```

- **带参函数的格式**

```
类型标识符 函数名称（形式参数列表）
{
        声明部分（语句）
}
```

3.2 函数的分类

➢ 无参函数的格式

【例 1】主函数调用其他函数模块

```
#include<iostream>
using namespace std;
void transfer(void)                                    //定义transfer函数
{
      cout << "This is the transfer function" << endl;          //输出一行文字
}
int main()
{
    transfer();                                        //调用transfer函数
      cout << "This is the main function" << endl;             //输出一行文字
      transfer();                                      //调用transfer函数
      return 0;
}
```

3.2 函数的分类

➢ 带参函数的格式

例如：

```
int min(int a, int b)          //函数首部，函数返回值为整型，有两个整型形参
{                               //函数体的声明部分
        int c = 0;
        c = a < b ? a : b;              //条件运算符，将a和b中的小者的值赋给整型变量c
        return c;                   //将c的值作为函数值返回给调用点处
}
```

这是一个求a和b二者中的小者的函数。在调用该函数时，主调函数将实际参数的值传递到被调函数的形参a和b中。花括号内为函数体，函数体的语句功能是求出a和b中最小的值并赋值给c, return c的作用是将c的值返回到主调函数中，c也被称为函数返回值。

3.3 函数的调用和声明

➢ 函数调用的形式

① 函数调用的一般形式

函数名称（[实参列表]）

如果调用无参函数，那么“实参列表”可以省略，但括号不能省略。倘若要实参列表有多个实参，那么需要用逗号隔开。实参和形参是相对而言的，主调函数的参数列表称之为实参，被调函数的参数列表称之为形参。值得注意的是，倘若实参列表存在多个实参，不同编译系统有着不同的求值顺序。假设变量x的值为5，有以下函数调用：

```
fun(x, ++x);
```

如果按照自左至右的求参顺序，则函数调用相当于fun(5,6) ,若按照自右至左顺序进行求参，则等同于fun(6,6)。常见的编译系统是按自右至左进行求参。

3.3 函数的调用和声明

② 函数调用的方式

从函数在语句中的作用角度看，可分为三种函数调用方式:

（1）函数语句

将函数调用作为单独的一个语句，只要求函数完成一系列的操作，不要求返回函数值。如例1所示:

```
transfer( );
```

（2）函数表达式

将函数的返回值参与表示式的运算中，此时要求被调函数需要返回一个确定的值。如:

```
z = 3 * min(x, y);
```

（3）函数参数

被调函数作为一个函数的实参。如:

```
k = 3 * min(z, min(x, y));
```

其中(x, y)为函数调用，其返回值作为外层min函数调用的一个实参。

3.3 函数的调用和声明

➢ 函数的递归

C++允许在调用一个函数的过程中又间接或直接地调用该函数自身，称之为递归调用。例如：

```
int fun(int a)
{
        int b = 0, c = 0;
        c = fun(b);             //调用函数fun的过程中，再次调用fun函数本身
        return (2 * c);
}
```

3.3 函数的调用和声明

➢ 函数的递归

- ✓ 该函数是直接调用本身，如图1 直接递归调用。
- ✓ 如果是间接调用本函数，如调用fun1函数过程中调用fun2函数，而且在调用fun2函数时再次调用fun1函数，过程如图2 所示。

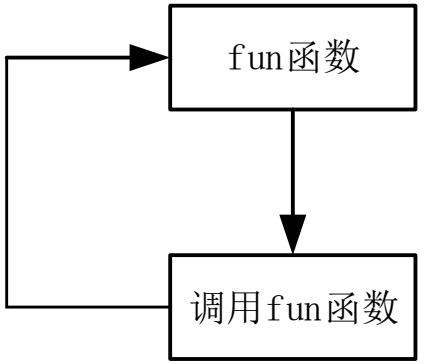

图1 直接调用图

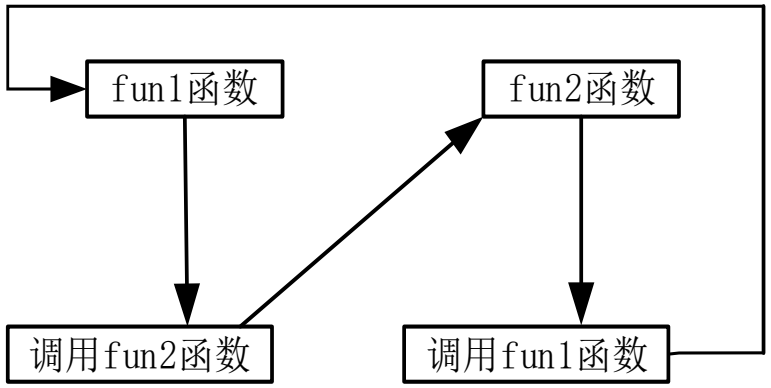

图2 间接调用图

3.3 函数的调用和声明

- 函数的递归

【例 2-1】用递归函数求1到n的累加和

```
#include<iostream>
using namespace std;
int fun(int a)
{
    int b = 0;
    if (a > 0)                          //边界条件 a大于0才继续执行回推
            b = a + fun(a - 1);         //回推操作
    else
            return 0;
    return b;
}
int main()
{
    cout << fun(3) << endl;
    return 0;
}
```

3.3 函数的调用和声明

➢ 函数的递归

【例 2-2】用非递归函数求1到n的累加和

递归需要占内存去保存调用函数的信息，递归的深度越深所消耗的内存越大。为了考虑效率，常常使用非递归方法求解

```
#include<iostream>
using namespace std;
int fun(int n)
{
    int sum = 0;
    for (int i = 1; i <= n; i++) {
            sum += i;
    }
    return sum;
}
int main()
{
    cout << fun(3) << endl;
    return 0;
}
```

3.4 函数的嵌套

在C++语言中不允许函数的嵌套定义，换而言之，一个函数内不能存在另一个函数的定义，如下所示。

```
void fun1()                    //函数fun1首部
{
    ...                        //函数fun1的函数体
    void fun2()                //定义fun2函数，这是非法的
    {
        ...                    //函数fun2的函数体
    }
}
```

3.4 函数的嵌套

同一个程序中，不同函数的定义都是相互独立和平行的，如下所示。

```
int fun1(){...}
double fun2(){...}
float fun3(){...}
long fun4(){...}
```

虽然C++的规定中，函数不能嵌套定义，但是能嵌套调用，即在调用一个函数的过程中，又调用另外一个函数。

3.4 函数的嵌套

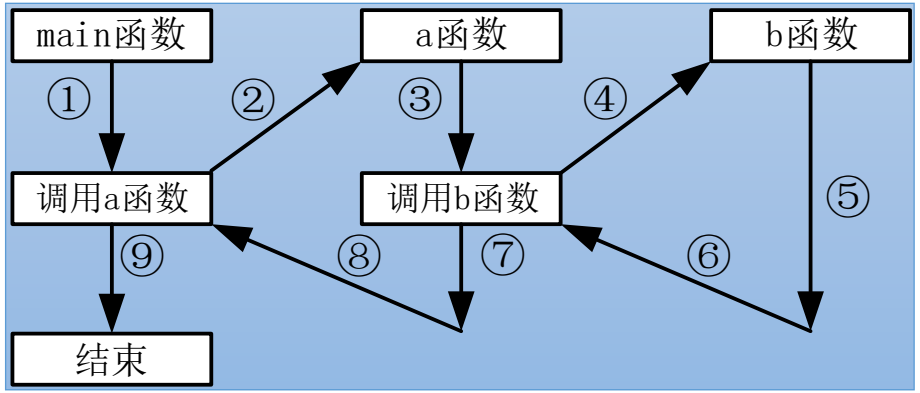

图3 函数嵌套调用图

图3表示两层嵌套调用(考虑main函数共三层函数)执行过程:

（1）执行语句首先从入口main函数进行;
（2）遇到调用函数a的语句，程序跳转到函数a的定义处;
（3）执行函数a的函数体语句;
（4）遇到调用函数b的语句，程序跳转到函数b的定义处;
（5）执行函数b的函数体语句，如果函数b没有继续嵌套调用函数，则执行完函数b的全部操作;
（6）返回调用函数b处，即返回函数a处;
（7）执行完函数a的剩余部分;
（8）返回主函数并执行主函数的剩余部分;
（9）程序结束。

在嵌套调用函数时，值得注意的是：如果调用函数的定义在主调函数之后，那么需要作出函数声明，反之则无需声明。

3.4 函数的嵌套

【例 3】任意输入三边长，求解是否能够组成直角三角形

```
#include<iostream>
using namespace std;
bool isTriangle(int a, int b, int c);          //函数声明
bool rightTriangle(int a, int b, int c);       //函数声明
int main()
{
    int a = 0, b = 0, c = 0;
    cout << "请输入边长a,b,c：" << endl;
    cin >> a >> b >> c;
    if (rightTriangle(a, b, c) == true)   //调用rightTriangle判断能否构成直角三角形
    {
        cout << "能构成直角三角形！" << endl;
    }
    else cout << "不能构成直角三角形！" << endl;
    return 0;
}
```

3.4 函数的嵌套

【例 3】任意输入三边长，求解是否能够组成直角三角形（续）

```
bool isTriangle(int a, int b, int c)       //定义判断能否构成三角形的函数
{
      if ((a + b) > c && (a + c) > b && (b + c) > a)   //任意两边之和大于第三边
                  return true;
      else  return false;
}
bool rightTriangle(int a, int b, int c)   //定义判断能否构成直角三角形的函数
{
      if (isTriangle(a, b, c)) //   调用isTriangle函数
      {                                 //勾股定理判断直角三角形
                if ((a * a) == (b * b + c * c) || (b * b)== (a * a + c * c) || (c * c) == (a * a + b * b))
                      return true;
      }
      return false;
}
```

3.5 函数的声明

在C++语言中，函数的定义是相互独立的，即一个函数不能在另一个函数内部定义。但函数调用是可以相互嵌套的，一个函数中可以调用其他函数。那么一个函数内调用另外一个函数需要具备什么条件呢?

① 首先被调函数必须是已经存在的函数，即编译系统的库函数或用户自定义函数

② 如果调用的是库函数，那么需要在本文件的开头使用预处理命令#include将库函数内容包含到当前文件中。例如，前面使用到过#include<iostream>,其中iostream是一个头文件。在iostream文件中包括了输入流（istream）和输出流（ostream）的一些函数声明和宏定义信息。如果不使用iostream文件，就无法使用输入输出流的对象。

③ 如果调用的是用户自定义函数，如果被调函数处于主调函数之后，则必须要在主调函数中或之前对被调函作出声明。

3.5 函数的声明

例：新生入学报到，正常情况下本应学生带齐资料到学校报到处进行入学办理，但有特殊情况的学生需要暂缓注册。该学生只需要将个人的基本信息（姓名、性别、准考号、录取编码等）提交至学校，请求给予注册并暂缓入学。学校就可将该学生列入学生名单，分配班级和学号，因此可以从学生名单中查到该学生对应的基本信息。待他本人到校补办手续，才最后正式确认入学。这就是报到的提前“声明”，如果无此声明，该学生将不被学校列入学生名单。

3.5 函数的声明

【例 4】对被调函数作提前声明（主函数中）

```
#include<iostream>
using namespace std;
int main()
{
        int add(int x, int y);
        int a, b, c;
        cout << "请输入a,b的值：" << endl;
        cin >> a >> b;
        c = add(a, b);
        cout << "和为" << c << endl;
        return 0;
}
int add(int x, int y)
{
        return x + y;
}
```

3.5 函数的声明

【例 4】对被调函数作提前声明（主函数前）

```
#include<iostream>
using namespace std;
double fun1(double, double); //本行和以下两行函数声明在所有函数之前且在函数外部
float fun2(float, float);                                    //因而作用域是整个文件
int fun3(int, int);
int main()
{                              //在main函数中不必对它所调用的函数作声明
    cout << "调用fun1(3.14,,3.14)的结果为：" << fun1(3.14, 3.14) << endl;
    cout << "调用fun2(2.5,2.5)的结果为：" << fun2(2.5, 2.5) << endl;
    cout << "调用fun3(3,3)的结果为：" << fun3(3, 3) << endl;
    return 0;
}
```

3.5 函数的声明

【例 4】对被调函数作提前声明（主函数前）（续）

```
double fun1(double d1,double d2)          //定义fun1函数
{     return d1 + d2;
}
float fun2(float f1, float f2)            //定义fun2函数
{     return f1 + f2;
}
int fun3(int i1, int i2)                  //定义fun3函数
{
      return i1 + i2;
}
```

如果一个函数需要被多个函数调用，用提前外部声明的方法可以减少重复的声明

3.5 函数的声明

- 函数的定义：函数的定义是一个完整的函数单元，其包括函数类型、函数名称、形参及形参类型、函数体等。在整个程序中，函数的定义只能出现一次，并且函数首部与花括号间是无分号的。

- 函数的声明：函数的声明是对编译系统的一个说明，其包括函数类型、函数名称、形参类型。与函数的定义不同，它不包含函数体，形参名称可以省略，并且多次声明。函数的声明必须以分号结束。

3.6函数的传递

大多数情况下，函数调用往往是带参调用。主调函数与被调函数之间往往存在数据传递的关系。函数定义首部的括号中的变量名称是形式参数（formal parameter,简称形参）。主调函数调用该函数时，函数名称后的括号里的参数称之为实际参数（actual parameter,简称实参），其中实参也可以是一个表达式。

3.6 函数的传递

【例 5】函数调用时的数据传递

```
#include<iostream>
using namespace std;
int min(int x, int y)                    //定义带参函数min, x和y为形参
{
        return x < y ? x : y;
}
int main()
{
        int a = 0, b = 0, c = 0;
        cout << "请输入两个整数：" << endl;
        cin >> a >> b;
        c = min(a, b);                   //调用min函数，给定实参为a,b，函数返回值赋给c
        cout << "min = " << c;
        return 0;}
```

3.6函数的返回值

- 主调函数通常调用函数时，希望被调函数能返回一个确认的值，这个值称之为函数的返回值
- 被调函数的一个确定返回值能通过return语句返回至主调函数中。如果主调函数需要被调函数的返回值，那么return语句必须存在。如果不需要返回值，那么被调函数的返回类型应当是void类型并且无需return语句
- 函数常常存在一个以上的return语句，往往执行到哪个return语句，就哪条语句起作用，并结束被调函数的流程
- return语句的后面的括号可以省略，例如“return x;”与“return (x)”效果一致。return语句也可以返回一个表达式，表达式应当是一个确定的值。例如“return a+b;”

3.7 带默认值的函数

通常情况下，被调函数的形参的数据是从主调函数的实参中获取的，因此实参的个数与形参的个数应相等。但在特殊情况下，需要多次调用某一函数并使用同一实参，C ++提供了便捷的方法，那就是给定形参一个默认值，从而形参不需要从实参中获取数据，如下所示。

```
int area( int r = 3);
```

指定r的默认值为3，如果调用该函数时，确定r的值取3，则无需给出实参，如

```
area( );                    //相当于area(3);
```

如果需要取其他值，那么可以通过实参给定，如

```
area( 6 ); //形参得到的数值为6，而不是3
```

3.7 带默认值的函数

通过设定默认值，使得编程更加灵活和便捷。如果有多个形参，可以选择每一个形参带有默认值，也可以部分形参设定默认值。如求一个圆锥体体积的函数，形参r表示圆锥体的底面积半径，h为圆锥体的高。函数原型如下：

```
double volume(double r ,double h = 11.5 );          //只对形参h指定默认值11.5
```

函数调用可以采用以下两种形式：

```
volume(32.5 );                                      //相当于volume(32.5 ,11.5);

volume(42.3 ,12.6 );                                //相当于volume(42.3 ,12.6);
```

3.7 带默认值的函数

形参与实参的结合是由左到右的顺序进行的，因此第一个实参必须与第一个形参相结合，第二个实参与第二个形参相结合……。因此指定默认值的参数当放在函数的参数列表的最右端，否则将出现错误。如下：

```
int fun1(double a, int b = 0, int c ,char d =’y’);          //错误

int fun2(double a, int c, int b = 0 ,char d =’y’);          //正确
```

如果调用上面fun2函数，那么可以采用以下形式：

```
fun2(2.5, 3, 2,’a’);                    //形参的值全部由实参给出
fun2(2.5, 3, 2);                        //最后一个形参的值取默认值’y’
fun2(2.5, 3);                           //最后两个形参的值取默认值
```

从上述可看出，调用带有默认值的函数，形参和实参的个数不同，实参未给出时，将使用函数定义时的默认值。

3.7 带默认值的函数

【例 6】求两个或三个数的最小值，用带默认值的函数实现

- 如果函数的定义位于调用函数之前，那么在函数定义的首部应给出默认值。
- 如果函数的定义位于调用函数之后，那么在函数调用之前应该有被调函数的声明，声明必须给出函数的默认值，定义处可以不给默认值

```
#include<iostream>
using namespace std;
int min(int a, int b, int c = 2147483647)          //函数定义
{
    if (b < a) a = b;
    if (c < a)a = c;
    return a;
}
int main()
{
    int a = 0, b = 0, c = 0;
    cin >> a >> b >> c;
    cout << "min(a,b,c) = " << min(a, b, c) << endl;  //输出三个数中的最小值
    cout << "min(a,b) = " << min(a, b) << endl;       //输出两个数中的最小值
    return 0;
}
```

3.8 函数的重载

在程序中，一般是根据不同的数据类型调用不同名称的函数。如果程序中出现这一类的情况较多，那么对于开发者而言，重复编写相同功能但类型不同的函数，开发效率低。此时，C++提供了函数重载的特征，能用同一个函数名称实现定义多个函数。

从三个数中取最小值：

```
double min1(double a, double b, double c); //求三个双精度中的最小值
long min2(long a, long b, long c);         //求三个长整型中的最小值
int min3(int a, int b, int c);             //求三个双精度中的最小值
```

3.8 函数的重载

【例 7】利用函数重载求三个数中的最小值（分别考虑双精度、长整型以及整型的情况）

```
#include<iostream>
using namespace std;
double min(double a, double b, double c);                    //函数声明
long min(long a, long b, long c);                            //函数声明
int min(int a, int b, int c);                                //函数声明
int main()
{
    double d = 0, d1 = 0, d2 = 0, d3 = 0;
    cin >> d1 >> d2 >> d3;                                   //输入3个双精度数
    d = min(d1, d2, d3);                                     //求3个双精度数中的最小值
    cout << "3个双精度的最小值为：" << d << endl;
    long l = 0, l1 = 0, l2 = 0, l3 = 0;
    cin >> l1 >> l2 >> l3;                                   //输入3个长整型数
    l = min(l1, l2, l3);                                     //求3个长整型数中的最小值
    cout << "3个长整型的最小值为：" << l << endl;
    int i = 0, i1 = 0, i2 = 0, i3 = 0, i4 = 0;
    cin >> i1 >> i2 >> i3;                                   //输入3个整型数
    i = min(i1, i2, i3);                                     //求3个整型数中的最小值
    cout << "3个整型的最小值为：" << i << endl;
    return 0 }
```

3.8 函数的重载

【例 7】利用函数重载求三个数中的最小值（分别考虑双精度、长整型以及整型的情况）（续）

```
double min(double a, double b, double c)      //定义求3个双精度数中的最小值的函数
{
    if (b < a) a = b;
    if (c < a)a = c;
    return a;
}
long min(long a, long b, long c)              //定义求3个长整型中的最小值的函数
{
    if (b < a) a = b;
    if (c < a)a = c;
    return a;
}
int min(int a, int b, int c)                  //定义求3个整型中的最小值的函数
{
    if (b < a) a = b;
    if (c < a)a = c;
    return a;
}
```

3.8 函数的重载

【例 8】实现求两个整数或三个整数中的最小值

```
#include<iostream>
using namespace std;
int min(int a, int b);                          //函数声明
int min(int a, int b, int c);                   //函数声明
int main()
{
    int a = 3, b = -4, c = -5;
    cout << "min(a,b) = " << min(a, b) << endl;          //输出两个整数中的最小值
    cout << "min(a,b,c) = " << min(a, b, c) << endl;     //输出三个整数中的最小值
    return 0;
}
int min(int a, int b)                           //定义求两个整数中的最小值的函数
{
    if (a < b)return a;
    else return b; }
int min(int a, int b, int c)                    //定义求三个整数中的最小值的函数
{
    if (b < a) a = b;
    if (c < a) a = c;
    return a;}
```

3.9 函数的内联

为了提高程序的运行速度，C++提供了内联函数的的特性。内联函数与普通函数之间的区别不在于编写方式，而在于编译系统如何将它们组合到程序中。普通函数的来回跳转将消耗计算机一定的性能，内联函数则无需来回跳转，因而相对于普通函数具有更快的运行速度。但是内联函数也有弊端，虽然节省了运行时间但是增加了内存消耗。如果程序在5个不同的地方调用内联函数，那么相当于该程序复制了5份内联函数的代码副本。

内联函数的声明和定义：

```
inline int min(int a, int b);                    //内联函数min的声明
inline int min(int a, int b) {return a<b?a:b};   //内联函数min的定义
```

如果函数过大或者函数用于递归，编译系统将不会把该函数作为内联函数，甚至一些编译系统不支持内联函数。

3.9 函数的内联

【例 9】用内联函数计算一个整数的平方

```
#include<iostream>
using namespace std;
inline int square(int x)                          //定义内联函数square
{
    return x * x;
}
int main()
{
    int a = 5, b = 6, c = 11;
    cout << "a = " << a << ", a 的平方为：" << square(a) << endl;    //调用内联函数square( a )
    cout << "b = " << b << ", b 的平方为：" << square(b) << endl;    //调用内联函数square( b )
    cout << "a + b 的平方为：" << square(a + b) << endl;             //调用内联函数square(a + b)
    cout << "c = " << c << ", c 的平方为：" << square(c) << endl;    //调用内联函数square( c )
    return 0;
}
```

3.9 函数的内联

内联函数是C++新增的特征。宏也可以看作内联代码的原始实现，但是二者具有一定的区别。

例：定义计算平方的宏

```
#define SQUARE(X)  X*X                    //计算平方的宏
```

宏与内联函数相比，宏没有通过参数传递，仅仅是文本替换实现的

```
a= SQUARE(6.5)                    //等同于a=6.5*6.5
b= SQUARE(6.5+7.5)                //等同于b=6.5+7.5* 6.5+7.5          X
```

修改宏

```
#define SQUARE(X)  (X)*(X)        //计算平方的宏
b= SQUARE(6.5+7.5)                //等同于b=(6.5+7.5)*(6.5+7.5)      √
```

3.10 变量的作用域

程序中的每一个变量都有其有效范围，也被称作变量的作用域。如果在作用域范围外是无法访问该变量的。在函数内部所定义的变量称之为内部变量，它的作用域范围仅仅局限在本函数内部。换而言之，只有在这函数内部才能访问该些变量，函数外部是无法访问的。同样，复合语句内部所定义的变量，其作用范围仅在复合语句内部有效。这些都称之为**局部变量**（local variable），

3.10 变量的作用域

```
int fun1(int a)          //函数fun1，变量a的作用范围是整个fun1函数。
{
        int b, c;        //变量b和c的作用范围是变量定义处到fun1函数结束。
        ...
}
float fun2(int x, int y)    //函数fun2，变量x和y的作用范围是整个fun2函数。
{
        int i, j;        //变量i和j的作用范围是变量定义处到fun2函数结束。.
}
int main()               //主函数
{
        int m, n;        //变量m和n的作用范围是变量定义处到main函数结束。
        ...
        {
                int p, q;       //变量p和q的作用范围是变量定义处到复合语句结束。
                ...
        }
        return 0;
}
```

3.10 变量的作用域

说明:

- main函数定义的变量m和n只在主函数内有效，不会因为在main函数定义从而变量的作用范围为全文件或程序有效。同时main函数也不能使用其他函数的局部变量。
- 不同函数可以定义相同名称的变量，它们是相互独立的对象，互不干扰。例如在fun1函数定义了变量m和n，又在fun2函数定义变量m和n，二者不会相互混淆，因为它们所占用的内存单元并不相同。
- 可以在一个函数的复合语句内定义变量，这些变量的有效范围仅在复合语句内，通常也将复合语句称之为程序块或分程序。
- 函数定义处的形参也属于局部变量。如fun1函数中的形参a的有效范围仅在fun1函数内，其他函数不能使用该变量。

3.10 变量的作用域

说明（续）：

- 在函数声明处出现的变量，其有效范围只在声明处的那行代码的括号内。实际上，编译系统对于函数声明中的变量名称是忽略的，即在函数被调用时，声明处的变量是不分配内存单元的。

```
int min(int a, int b)  //函数声明中出现a,b
…
int min(int x, int y)  //函数定义中的形参是x,y
{
cout<<x<<y<<endl;    //x,y在函数体中有效，合法
cout<<a<<b<<endl;   //a,b在函数体中无效，非法
}
```

3.11 本章小结

- ✓ 程序是由若干个源文件构成，源文件包含若干个函数，每个函数完成程序一部分功能，其中main函数在整个程序中，存在且唯一。
- ✓ 函数的调用方式分为三种：①函数语句，②函数表达式，③函数参数，其中函数自身调用又称之为函数的递归。
- ✓ 不同的函数都是相互独立和平行的，不允许相互嵌套定义，但能嵌套调用。调用某一个函数必须在调用处提前定义或声明。
- ✓ 主调函数与被调函数之间可以通过参数传递实现联系的桥梁。
- ✓ 为了使得编程更加灵活和高效，C++提供了带默认值的函数和函数重载，带默认值的函数设定了部分形参的初始值，使得变量更加灵活。
- ✓ 程序中的每一个变量都有其作用域