C++初学者必看的50个建议

- 18.学习编程最好的方法之一就是阅读源代码；
- 19.在任何时刻都不要认为自己手中的书已经足够了；
- 20.请阅读《The Standard C++ Bible》(中文版：标准C++宝典)，掌握C++标准；
- 21.看得懂的书，请仔细看；看不懂的书，请硬着头皮看；
- 22.别指望看第一遍书就能记住和掌握什么——请看第二遍、第三遍；

第4章 函数与预处理

Function & Preprocessor directives

Main Content

- Function's definition, declaration and call
- Parameters transfer between functions
- Function's overloading
- Function template
- Functions with default format parameters
- Function's Nesting call
- Function's Recursion call

函数（*function*）概述

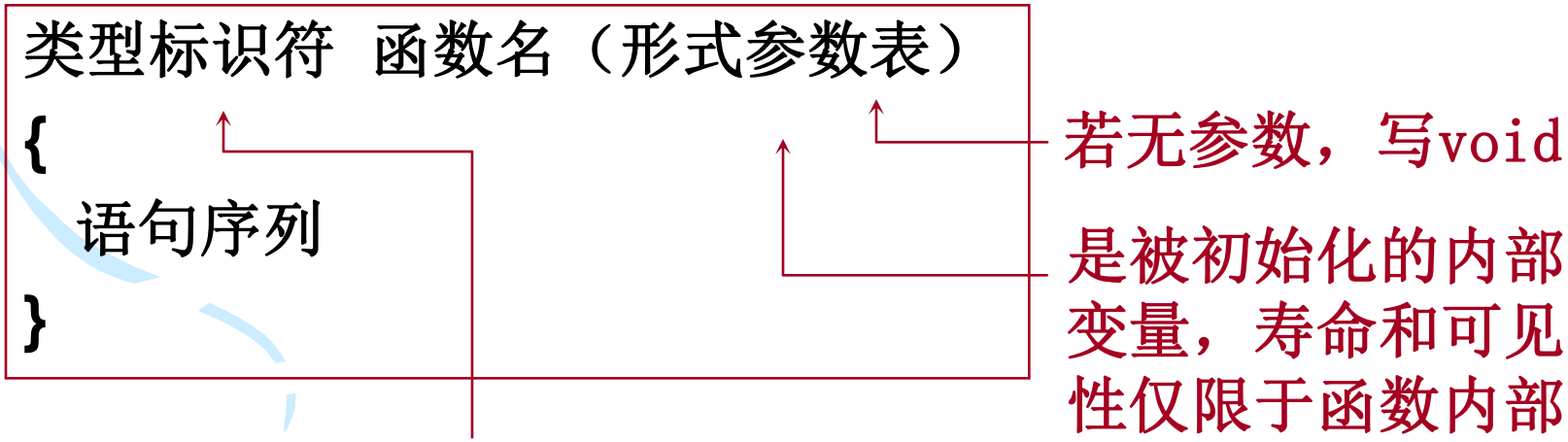
C++的函数（function）是完成既定任务的功能（过程）体，它涵盖了数学函数和一般过程．所以基于过程编程本质上就是基于函数编程。

为了便于规划、组织、编程和调试，一般的做法是把一个大的程序划分为若干个程序模块(即程序文件)，每一个模块实现一部分功能。

在程序进行编译时，以程序模块为编译单位，即分别对每一个编译单位进行编译。如果发现错误，可以在本程序模块范围内查错并改正。

1 Function's Definition

- **Function is the basic abstract module in OOP, and it also is abstract for some tasks.**
- **Syntax Form of function definition:**



- Format parameter list

  <$type_1$> $name_1$, <$type_2$> $name_2$, ..., <$type_n$> $name_n$
- Return value of function
  - Given by return sentence. For example：
    - `return 0;`
    - `return (1);`
  - If the function has no return value, which means **void** type，the return sentence can be omitted.

2 Function Declaration

- **Declaration principle：**

  If a function is defined before its being used, the declaration may be omitted;

  If a function is used before its definition, it must be declared in advanced.

- **Declaration method：**
- **Return type Name （parameters list ）；**

  For example：

  int sum(int n, float m);

//函数申明可以不包含参数的名字，而只要包含函数的类型

3 Function's type

- **Getting parameters and returning value.** For example：

```
int bigger(int a, int b)
{
   return(a>b)?a:b;
 }
```

- **Getting parameters but not return value.** For example：

```
void delay(long a)
{
  for(int i=1;i<=a;i++);
}
```

- C++要求在定义函数时必须指定函数的类型。

3 Function's type

- **Getting no parameters but returning value.** For example:

```
int geti()
{
    int x;
    cout<<"Please input a interger:\n";
    cin>>x;
    return x;
 }
```

- **Getting no parameters and returning no value.** For example：

```
void message()
{
    cout<<"This is a message.\n"
}
```

4 Function's Call

- **Declare a function prototype at the beginning of the program before it being called:**

  类型标识符 被调用函数名 （含类型说明的形参表）;

- **Call Form**

  函数名（实参列表）

  max(a,b)

- **Nesting call**

  Nesting definition is not allowed in any function, but nesting call is permitted.

- **Recursion call**

  Function can call itself directly or indirectly.

例4．1 在主函数中调用其他函数

```
#include <iostream>
using namespace std;
void printstar(void)                    //定义printstar函数
{
  cout<<"****************************** "<<endl;  //输出30个 “*”
}

void print_message(void)                //定义print_message函数
{
  cout<<"Welcome  to  C++!"<<endl;     //输出一行文字
}

int main(void)
{
  printstar( );                  //调用printstar 函数
  print_message( );                   //调用print_message函数
  printstar( );                  //调用printstar 函数
  return 0;
}
```

4.3 函数参数和函数的值

4.3.1 形式参数和实际参数

在定义函数时函数名后面括号中的变量名称为形式参数（**formal parameter**，简称形参）；

在主调函数中调用一个函数时，函数名后面括号中的参数称为实际参数（**actual parameter**，简称实参）。

例4.2 调用函数时的数据传递。

```
#include <iostream>
using namespace std;
int main( )
{
   int a,b,c;
   int max(int x,int y) ;
   cout<<"please enter two integer numbers：";
   cin>>a>>b;
   c=max(a,b);          //调用max函数，给定实参为a,b。函数值赋给c
   cout<<"max="<<c<<endl;
   return 0;
}

int max(int x,int y)          //定义有参函数max
{int z;
 z=x>y?x：y;
 return(z);
}
```

有关形参与实参的说明

(1) 在定义函数时指定的形参，在未出现函数调用时，它们并不占内存中的存储单元，因此称它们是形式参数或虚拟参数。

只有在发生函数调用时，函数max中的形参才被分配内存单元，以便接收从实参传来的数据。

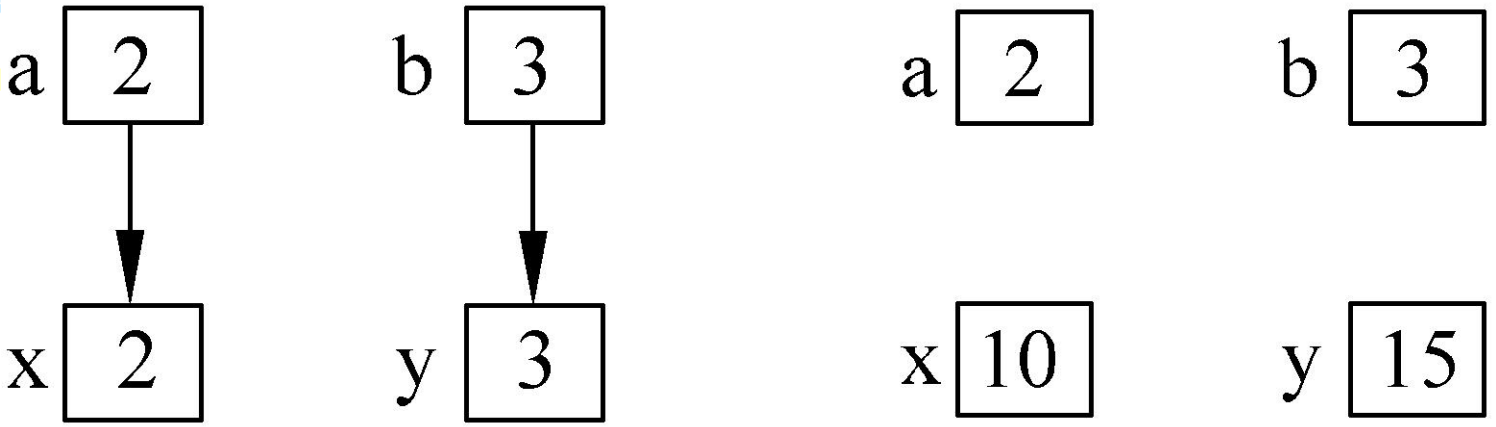
在调用结束后，形参所占的内存单元被释放。

(2) 实参可以是常量、变量或表达式，如max(3, a+b)；但要求a和b有确定的值。

(3) 定义函数时，必须在函数首部指定形参类型。

(4) 实参与形参的类型应相同或赋值兼容。

(5) 实参变量对形参变量的数据传递是“值传递”，即单向传递，只由实参传给形参。 形参的值如果改变，不会改变主调函数中实参的值。



4.4 函数的调用

4.4.1 函数调用的一般形式

函数名（［实参表列］）

调用无参函数，则“实参表列”可以没有，但括号不能省略。多个实参用逗号隔开。

实参与形参个数应相等，类型应匹配。

实参与形参按顺序对应，一对一地传递数据。

```
i=3;
fun(i,++i);//实参按从右至左的顺序求值。
```

4.4.2 函数调用的方式

1．函数语句

不要求函数带回一个值，只是要求函数完成一定的操作。

2. 函数表达式

函数出现在一个表达式中，这时要求函数带回一个确定的值以参加表达式的运算。如**c=2*max(a,b);**

3. 函数参数

函数调用作为一个函数的实参。如

m=max(max(a,b)，max(c,d));

例4.3 对被调用的函数作声明。

```
#include <iostream>
using namespace std;
int main( )
{
  float add(float x,float y);          //对add函数作声明
  float a,b,c;
  cout<<"please enter a,b：";
  cin>>a>>b;
  c=add(a,b);
  cout<<"sum="<<c<<endl;
  return 0;
}
float add(float x,float y)             //定义add函数
{
  float z;
  z=x+y;
  return (z);
}
```

定义和声明的区别

定义是指对函数功能的确立，包括指定函数名、函数类型、形参及其类型、函数体等，它是一个完整的、独立的函数单位。

声明是把函数名、函数类型、形参及其类型通知编译系统。

在函数声明中可以只写形参的类型，如

float add(float a，float b);

称为函数原型(function prototype)。

编译系统并不检查参数名。

它的作用主要是：

根据函数原型在程序编译阶段对调用函数的合法性进行全面检查。

说明：

(1) 一般都把main函数写在最前面，这样对整个程序的结构和作用一目了然，然后再具体了解各函数的细节。

此外，用函数原型来声明函数，还能减少编写程序时可能出现的错误。由于函数声明的位置与函数调用语句的位置比较近，因此在写程序时便于就近参照函数原型来书写函数调用，不易出错。

(2) 函数声明的位置可以在调用函数所在的函数中，也可以在函数之外。

编程风格之函数

- 函数应该短小而迷人，它应只覆盖一到两个屏幕(80*24一屏),并且只作一件事情,而且将它做好。
- 假如你要写一个很复杂的函数。而且你已经估计到假如一般人读这个函数,他可能都不知道这个函数在说些什么，这个时候,使用具有描述性名字的有帮助的函数。
- 另外一个需要考虑的是局部变量的数量，它们不应该超过5-10个,否则你有可能会出错。重新考虑这个函数，将他们分割成更小的函数。人的大脑通常可以很容易的记住7件不同的事情，超过这个数量会引起混乱。你知道你很聪明,但是你可能仍想去明白2周以前的做的事情。

例 数制转换

- 说明：
- 要求输入一个8位二进制数，将其转换为十进制数输出。
- 例如：
- $1101_2 = 1(23) + 1(22) + 0(21) + 1(20)$
  $= 13_{10}$
- 所以，如果输入1101，则应输出13

```
#include <iostream>
using namespace std;
double power (double x, int n);
int main()
{
        int  i;
        int  value = 0;
        char ch;
        cout << "Enter an 8 bit binary number  ";
        for (i = 7; i >= 0; i--)
        {
                cin >> ch;
                if (ch == '1')
                        value += int(power(2,i));
        }
        cout <<"Decimal value is  "<<value<<endl;
        return 0;
}
double power (double x, int n)
{
        double val = 1.0;
          while (n--)
                val *= x;
        return(val);
}
```

例3.5 求一元二次方程式$ax^2+bx+c=0$的根。a,b,c的值在运行时由键盘输入，它们的值满足$b^2-4ac\geq 0$。

根据求x1,x2的算法。它可以编写出以下C++程序：

```
#include <iostream>
#include <cmath>
using namespace std;
int main( )
{
    float  a,b,c,x1,x2;
    cin>>a>>b>>c;
    x1=(-b+sqrt(b*b-4*a*c))/(2*a);
    x2=(-b-sqrt(b*b-4*a*c))/(2*a);
    cout<<"x1="<<x1<<endl;
    cout<<"x2="<<x2<<endl;
    return 0;
}
```

- #include <iostream>
- #include <cmath>
- using namespace std;
- double f1(double a,double b,double c);
- double f2(double a,double b,double c);
- bool test(double a,double b,double c);

```
int main( )
{
    float  a,b,c,x1,x2;
    bool positive;
    cin>>a>>b>>c;
    if (positive=test(a,b,c)==0) cout<<"data error!\n";
    else
    {
        x1=f1(a,b,c);
        x2=f2(a,b,c);
        cout<<"x1="<<x1<<endl;
        cout<<"x2="<<x2<<endl;
    }
    return 0;
}
```

```
double f1(double a,double b,double c)
{     double z;
      z=(-b+sqrt(b*b-4*a*c))/(2*a);
      return z;
}
double f2(double a,double b,double c)
{     double z;
      z=(-b-sqrt(b*b-4*a*c))/(2*a);
      return z;
}
bool test(double a,double b,double c)
{
      if(b*b-4*a*c>0) return 1;
      else return 0;
}
```

C++初学者必看的50个建议

- 23.请看《Effective C++》和《More Effective C++》以及《Exceptional C++》；
- 25.和别人一起讨论有意义的C++知识点，而不是争吵XX行不行或者YY与ZZ哪个好；
- 26.请看《程序设计实践》，并严格的按照其要求去做；
- 27.不要因为C和C++中有一些语法和关键字看上去相同，就认为它们的意义和作用完全一样；
- 29.请不要认为学过XX语言再改学C++会有什么问题——你只不过又在学一门全新的语言而已；

4.6 函数的重载

Function overloading

在编程时，有时要实现的是同一类的功能，只是有些细节不同。

此时允许用同一函数名定义多个函数，这些函数的参数个数和参数类型不同。这就是函数的重载(function overloading)。

例4.5 求3个数中最大的数

```
#include <iostream>
using namespace std;
int main( )
{
    int max(int a,int b,int c);                    //函数声明
    double max(double a,double b,double c);        //函数声明
    long max(long a,long b,long c);                //函数声明
    int i1,i2,i3,i;
    cin>>i1>>i2>>i3;        //输入3个整数
    i=max(i1, i2, i3);                  //求整数中的最大者
    cout<<"i_max="<<i<<endl;
```

```
double d1,d2,d3,d;
cin>>d1>>d2>>d3;          //输入3个双精度数
d=max(d1, d2, d3);       //求双精度数中最大者
cout<<"d_max="<<d<<endl;

long g1,g2,g3,g;
cin>>g1>>g2>>g3;          //输入3个长整数
g=max(g1, g2, g3);       //求长整数中的最大者
cout<<"g_max="<<g<<endl;
return 0;
}
```

```
int max(int a,int b,int c)          //定义整数函数
{
   if(b>a) a=b;
   if(c>a) a=c;
   return a;
}
double max(double a,double b,double c) //定义双精度数函数
{
   if(b>a) a=b;
   if(c>a) a=c;
   return a;
}
long max(long a,long b,long c)       //定义长整数函数
{
   if(b>a) a=b;
   if(c>a) a=c;
   return a;
}
```

例4.6 求两个或3个整数中的最大数

```
#include <iostream>
using namespace std;
int main( )
{
  int max(int a, int b, int c);    //函数声明
  int max(int a, int b);           //函数声明
  int a=8,b=-12,c=27;
  cout<<"max(a,b,c)="<<max(a,b,c)<<endl;    //3个整数
  cout<<"max(a,b)="<<max(a,b)<<endl;        //两个整数
  return 0;
}
```

```
int max(int a,int b,int c)
{
    if(b>a) a=b;
    if(c>a) a=c;
    return a;
}
int max(int a,int b)
{
    if(a>b) return a;
    else return b;
}
```

参数的个数和类型可以都不同。但不能只有函数的类型不同而参数的个数和类型相同。

- **Matching order of overloading parameters** According to the optimal matching of type or number between format and practical parameters, complier will determine which function should be called. Calling order is as follows：
  - Strict matching;
  - Matching through inner transfer
  - Matching through user self-define；

Example1：void print(double);

void print(int);

While calling:

print(1); //调用void print(int),严格匹配；

print('a'); //调用void print(int),内部转换匹配；

print(1.0); // 调用void print(double),严格匹配；

print(3.14f); // 调用void print(double),内部匹配；

Example2： **void print(long);**

void print(double);

While calling：

int a;

print(a); //error, Creating duality，the complier can't determine which function will be called.

//VC++6.0: ambiguous call to overloaded function

Resolvent ： Matching through user self-define.

for example:

print((double) a); or **print((long) a);**

or **void print(int);**

- 注：不要将不同功能的函数声明为重载函数，以免出现调用结果的误解、混淆。
- 这样不好：
- int add(int x,int y)
- { return x+y; }

- float add(float x,float y)
- { return x-y; }

- 参数的个数和类型可以都不同。但不能只有函数的类型不同而参数的个数和类型相同。

4.7 函数模板

function template

函数模板(function template) 是建立一个通用函数，其函数类型和形参类型不具体指定，用一个虚拟的类型来代表。

定义函数模板的一般形式为

template < typename T> 或 template <class T>

通用函数定义

它只适用于函数的参数个数相同而类型不同，且函数体相同的情况。

例4.7 函数模板

```
#include <iostream>
using namespace std;
template <typename T>   //模板声明，其中T为类型参数
T max(T a,T b,T c)                    //定义一个通用函数，用T作虚拟的类型名
{
    if(b>a) a=b;
    if(c>a) a=c;
    return a;
}
int main( )
{
    int i1=185,i2=-76,i3=567,i;
    double d1=56.87,d2=90.23,d3=-3214.78,d;
    long g1=67854,g2=-912456,g3=673456,g;
    i=max(i1,i2,i3);          //调用模板函数，此时T被int取代
    d=max(d1,d2,d3);          //调用模板函数，此时T被double取代
    g=max(g1,g2,g3);          //调用模板函数，此时T被long取代
    cout<<"i_max="<<i<<endl;
    cout<<"f_max="<<d<<endl;
    cout<<"g_max="<<g<<endl;
    return 0;
}
//d=max(d1,d2,d3);          //调用模板函数，此时T被double取代
//err:no matching function for call to `max(int&, double&, long int&)'
```

4.8 有默认参数的函数

Functions with default format parameters

一般情况下，实参个数应与形参相同。

函数在声明时可以预先给出默认的形参值，调用时如给出实参，则采用实参值，否则采用预先给出的默认形参值。

float area(float r=6.5);

指定r的默认值为6.5，如果在调用此函数时，确认r的值为6.5，则可以不必给出实参的值。

```
area( );      //相当于area(6.5);
area(7.5);    //形参得到的值为7.5
```

如果有多个形参，可以对每个形参或部分形参指定默认值。

实参与形参的结合是从左至右顺序进行的。因此指定默认值的参数必须放在形参表列中的最右端，否则出错。

```
void f1(float a，int b=0，int c，char d='a');
void f2(float a，int c，int b=0, char d='a');

fun(i,++i);//实参按从右至左的顺序求值。
```

Effect region of default format parameters' value

- 在相同的作用域内，缺省形参值的说明应保持唯一，但如果在不同的作用域内，允许说明不同的缺省形参。

- **For example：**

```
int add(int x=1,int y=2);
int main()
{  int add(int x=3,int y=4);
    add（ ）；//使用局部缺省形参值（实现3+4）
}
void fun(void)
{  ...
    add（ ）；//使用全局缺省形参值（实现1+2）
}
```

例4.8 求2个或3个正整数中的最大数

```
#include <iostream>
using namespace std;
int main( )
{
   int max(int a, int b, int c=0);//函数声明,形参c有默认值
   int a,b,c;
   cin>>a>>b>>c;
   cout<<"max(a,b,c)="<<max(a,b,c)<<endl;          //输出3个数中的
最大者
   cout<<"max(a,b)="<<max(a,b)<<endl;     //输出2个数中的最大者
   return 0;
}

int max(int a,int b,int c)       //函数定义
{
   if(b>a) a=b;
   if(c>a) a=c;
   return a;
}
```

注意：

(1) 如果函数的定义在函数调用之后，则必须在函数声明中给出默认值。否则应在函数定义中给出默认值。

(2) 一个函数不能既作为重载函数，又作为有默认参数的函数。

因为当调用函数时如果少写一个参数，系统无法判定是利用重载函数还是利用默认参数的函数。

- 题目：要求输出国际象棋棋盘。
  1.程序分析：用i控制行，j来控制列，根据i+j的和的变化来控制输出黑方格，还是白方格。
- 白方格用ASCII码的219码。

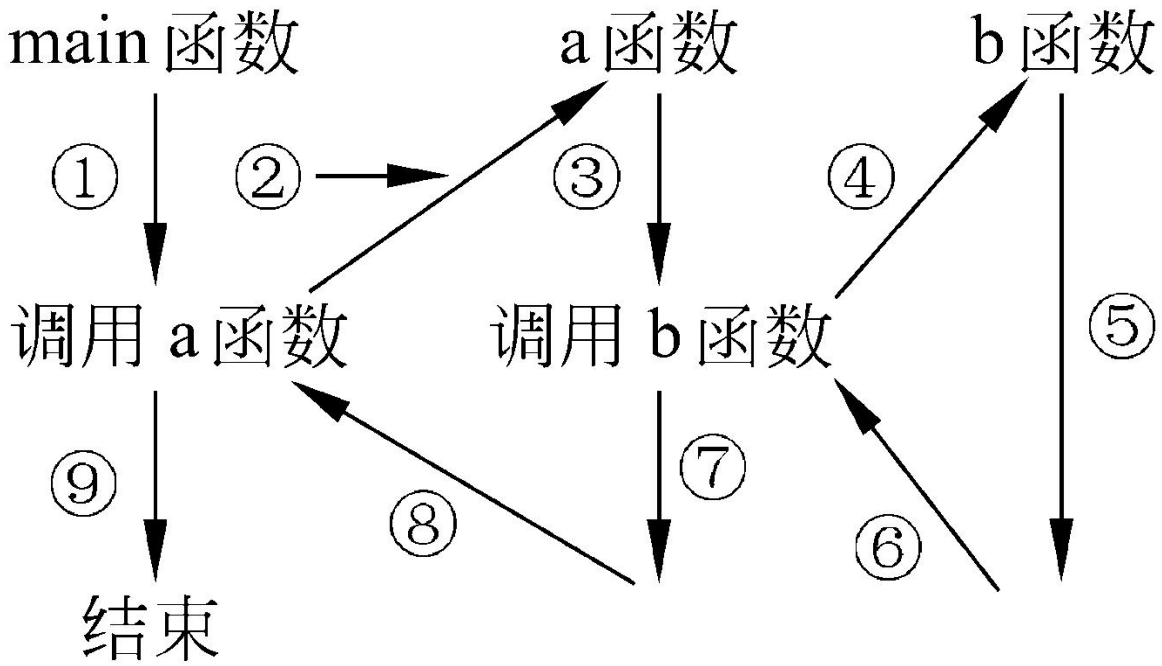
```
#include <iostream>
using namespace std;
int main()
{
int i,j;
for(i=0;i<8;i++)
{
        for(j=0;j<8;j++)
        if((i+j)%2==0)
                cout<<'\xdb'<<'\xdb';
        else
                cout<<"  ";
        cout<<"\n";
        }
return 0;
}
```

4.9 函数的嵌套调用

Function's Nesting call

C++不允许对函数作嵌套定义

C++可以嵌套调用函数



Example 输入两个整数，求平方和。

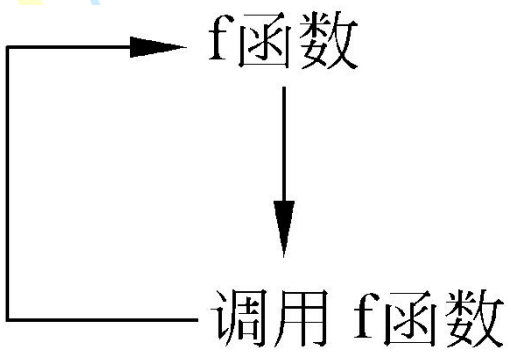
```
#include <iostream>
using namespace std;
int main( )
{
        int a,b;
        int fun1(int x,int y);//声明fun1函数
        cin>>a>>b;
        cout<<"a,b的平方和："<<fun1(a,b)<<endl;
}
int fun1(int x,int y)
{
        int fun2(int m);//声明fun2函数
        return (fun2(x)+fun2(y));
}
int fun2(int m)
{
        return (m*m);
}
```

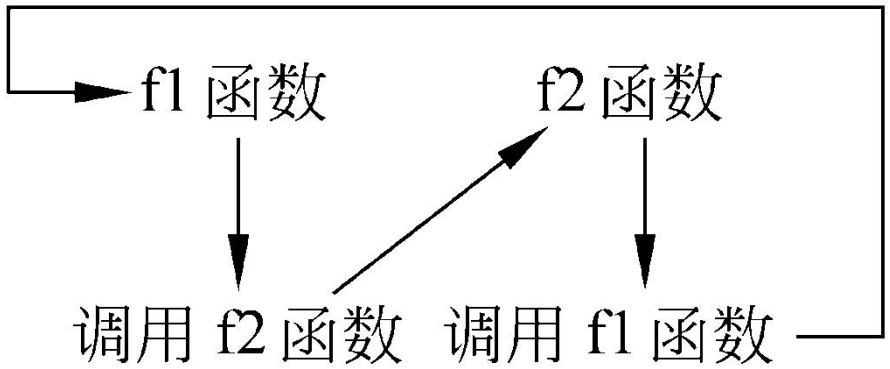
4.10 函数的递归调用

Function's Recursion call

在调用一个函数的过程中又直接或间接地调用该函数本身，称为函数的递归(recursive)调用。包含递归调用的函数称为递归函数。

一般用条件语句控制递归调用的次数或终止递归调用。





例4.10 有5个人坐在一起，问第5个人多少岁？他说比第4个人大两岁。问第4个人岁数，他说比第3个人大两岁。问第3个人，又说比第2个人大两岁。问第2个人，说比第1个人大两岁。最后问第1个人，他说是10岁。请问第5个人多大？

每一个人的年龄都比其前1个人的年龄大两岁。即

age(5)=age(4)+2

age(4)=age(3)+2

age(3)=age(2)+2

age(2)=age(1)+2

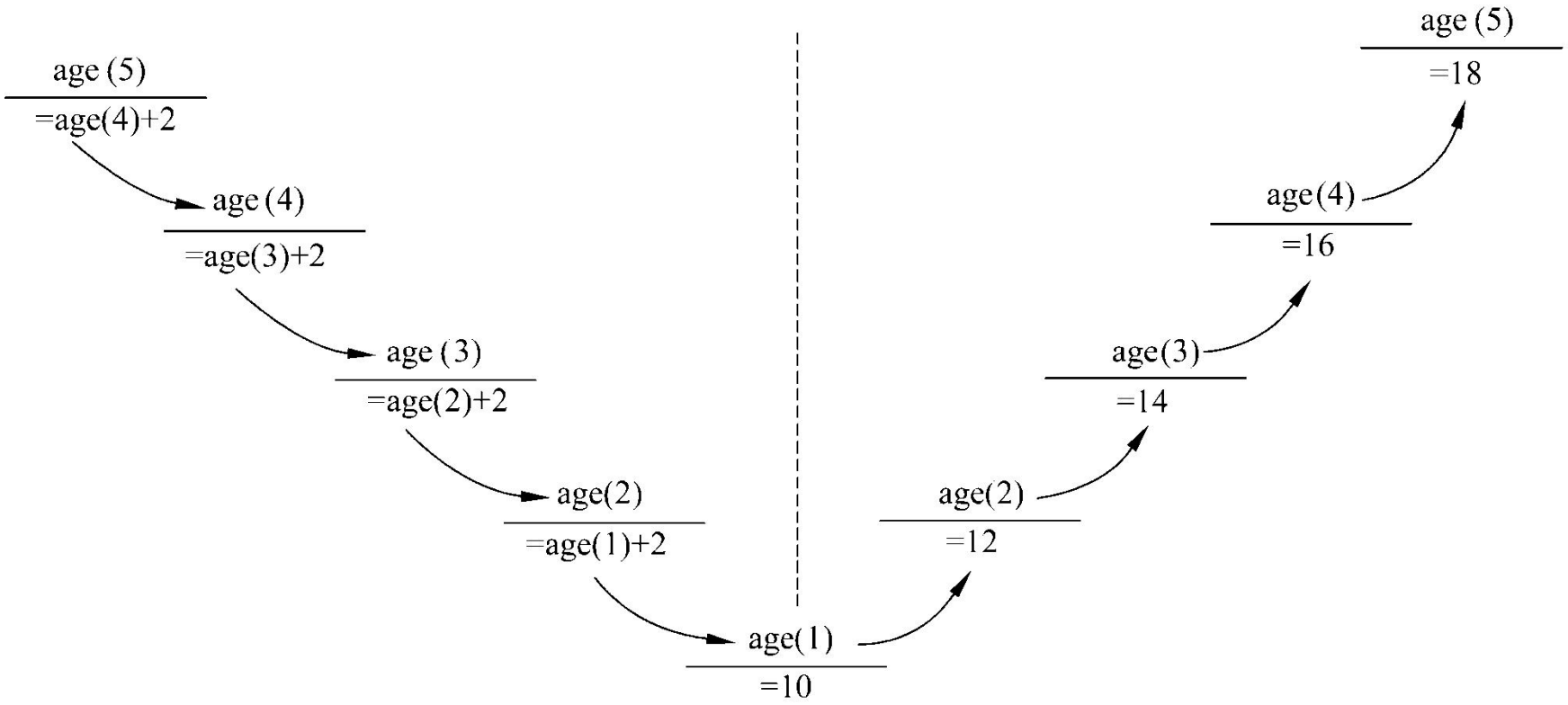
age(1)=10

可以用式子表述如下：

age(n)=10　　(n=1)

age(n)=age(n-1)+2　　(n>1)

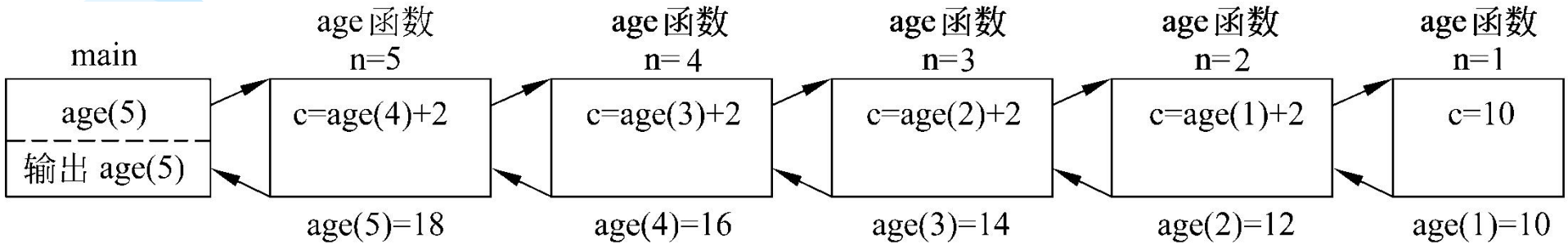
可以看到，当n>1时，求第n个人的年龄的公式是相同的。因此可以用一个函数表示上述关系。图4.11表示求第5个人年龄的过程。



递归过程实现

```
#include <iostream>
using namespace std;
int age(int);                    //函数声明
int main( )                      //主函数
 {
    cout<<age(5)<<endl;
    return 0;
 }
int age(int n)                   //求年龄的递归函数
{
    int c;                       //用c作为存放年龄的变量
    if(n==1) c=10;               //当n=1时，年龄为10
    else c=age(n-1)+2;           //当n＞1时，年龄是前者的年龄加2
    return c;                    //将年龄值带回主函数
}
```

函数调用过程如图4.12所示。



例4.11 用递归方法求n！

求n!可以用下面的递归公式表示：

$$n! = \begin{cases} 1 & (n=0,1) \\ n\cdot(n-1)! & (n>1) \end{cases}$$

```
#include <iostream>
using namespace std;
long fac(int);                          //函数声明
```

```
int main( )
{
    int n;                              //n为需要求阶乘的整数
    long y;                             //y为存放n!的变量
    cout<<"please input an integer ： ";
    cin>>n;
    y=fac(n);                                   //调用fac函数以求n!
    cout<<n<<"!="<<y<<endl;
    return 0;
}
```

```
long fac(int n)                          //递归函数
{
    long f;
    if(n<0)
    {
        cout<<"n<0,data error!"<<endl;
        f=-1;
    }
    else if (n==0||n==1) f=1;            //0!和1!的值为1
    else f=fac(n-1)*n;                   //n>1时，进行递归调用
    return f;
}
```

习题3.22

```
#include <iostream>
using namespace std;
int main()
{ int day=9,x1,x2;
  x2=1;
  while(day>0)
  { x1=(x2+1)*2;      // 第1天的桃子是第2天桃子加1后的2倍
    x2=x1;
    day--;
  }
  cout<<"total="<<x1<<endl;
  return 0;
}
```

```
#include <iostream>
using namespace std;
int total(int);                    //函数声明
int main( )                        //主函数
{ cout<<"total="<<total(10)<<endl;
  return 0;
}
int total(int day)
{ int sum;                         //用sum作为存放桃子总数的变量
  if(day==1) sum=1;                //当day=1时，桃子总数为1
  else sum=(total(day-1)+1)*2;  // 第1天的桃子数是第2天桃子数加1后的2倍
  return sum;
}
```

C++初学者必看的50个建议

- 31.学习编程的秘诀是：编程，编程，再编程；
- 33.记住：面向对象技术不只是C++专有的；
- 34.请把书上的程序例子亲手输入到电脑上实践，即使配套光盘中有源代码；
- 35.把在书中看到的有意义的例子扩充；
- 37.经常回顾自己以前写过的程序，并尝试重写，把自己学到的新知识运用进去；

4.11变量的作用域（scope）

在函数（复合语句）内部定义的变量是内部变量，又称为局部变量(local variable)。

局部变量只在函数（复合语句）范围内有效。

在函数（复合语句）之外定义的变量是外部变量，称为全局变量(global variable) 。

全局变量的有效范围为从定义变量的位置开始到本文件结束。

local variable & global variable

```
int p=1,q=5;                //全局变量
float f1(int a)             //定义函数f1
{
    int b,c;
    ┆
}
char c1,c2;                 //全局变量c1、c2
char f2 (int x, int y)      //定义函数f2
{
    int i,j;
    ┆
}
main ( )//主函数
{
    int m,n;
    ┆
}
```

形参变量a的作用范围

局部变量m、n的作用范围

全局变量c1、c2的作用范围

全局变量p、q的作用范围

```
int p=1,q=5;                  //全局变量
float f1(int a)               //定义函数f1
{
   int b,c;
    ┆
}
char c1,c2;                   //全局变量c1、c2
char f2 (int x, int y)        //定义函数f2
{
   int i,j;
    ┆
}
main ( )//主函数
{
   int m,n;
    ┆
}
```

形参变量a的作用范围

局部变量m、n的作用范围

全局变量c1、c2的作用范围

全局变量p、q的作用范围

说明

- 局部变量
- (1) 在函数中定义，主函数main中定义的变量也是局部变量。
- (2)形式参数也是局部变量，其作用范围只在本函数范围内。
- (3)不同函数的同名变量代表不同的对象。
- (4)未初始化时值不确定。

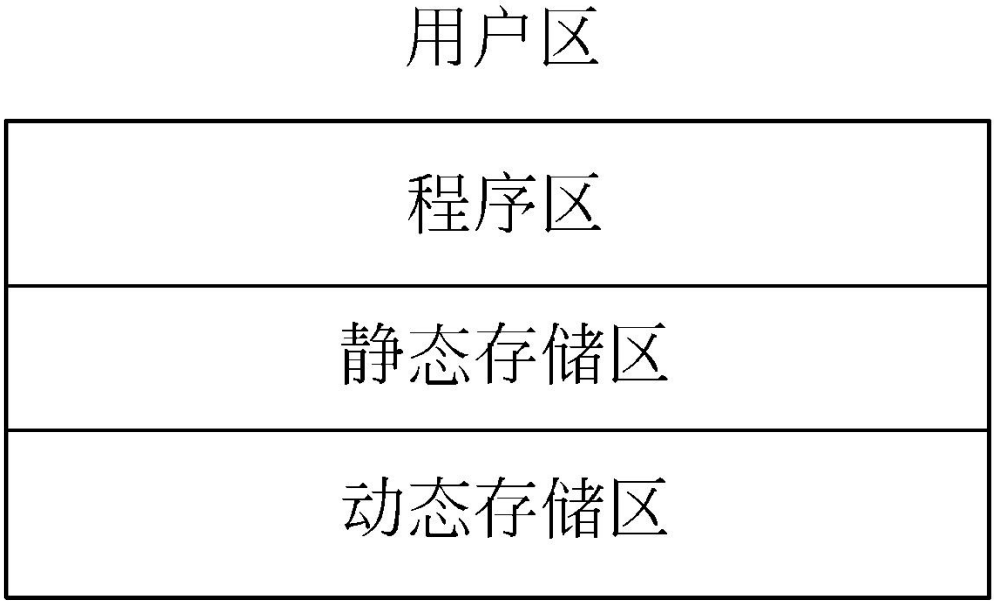
- 全局变量
- (1) 增加了函数间数据联系的渠道。
- (2) 尽量少用全局变量
  - ①函数的独立性降低。
  - ②降低程序的清晰性。
- (3) 在局部变量的作用范围内，同名全局变量被屏蔽。
- (4)默认值为0(或'\0')。

4.12 变量的存储类别(storage class)

存储期(storage duration，也称生存期) 指变量在内存中的存在时间，由变量存储类别决定（自动、静态、*寄存器*、外部)。

静态存储-系统对变量分配固定的存储空间。

动态存储-系统对变量动态地分配存储空间。



在静态存储区中存放全局变量及静态局部变量。在程序执行过程中它们占据固定的存储单元。

在动态存储区中存放以下数据：

①函数形式参数。调用函数时给形参分配存储空间。

②函数中的自动变量。

③函数调用时的现场保护和返回地址等。

对以上这些数据，在函数调用开始时分配动态存储空间，函数结束时释放这些空间。

4.12.2 自动变量

局部变量，默认为自动变量(auto variable)，编译系统对它们是动态地分配存储空间的。

在调用该函数时，系统给形参和函数中定义的变量分配存储空间，数据存储在动态存储区中。在函数调用结束时就自动释放这些空间。

4.12.3 用static声明静态局部变量

静态局部变量(static local variable)

局部变量

静态存储方式

有时希望函数中的局部变量的值在函数调用结束后能够保留，在下一次该函数调用时，该变量的值能够继续使用。

可指定该局部变量为静态局部变量。如

static float a;

例4.12 静态局部变量的值

```
#include <iostream>
using namespace std;
int f(int a)                    //定义f函数，a为形参
 {auto int  b=0;                  //定义b为自动变量
  static int c=3;                 //定义c为静态局部变量
  b=b+1;
  c=c+1;
  return a+b+c;
 }

 int main( )
 {int a=2,i;
  for(i=0;i<3;i++)
  cout<<f(a)<<" ";
  cout<<endl;
  return 0;
 }
```

对静态局部变量的说明：

(1) 静态局部变量在静态存储区内分配存储单元。在程序整个运行期间都不释放。

(2) 为静态局部变量赋初值是在编译时进行的，以后每次调用函数时不再重新赋初值而只是保留上次函数调用结束时的值。而为自动变量赋初值，是在函数调用时进行，每调用一次函数重新给一次初值。

(3) 如果对静态局部变量不赋初值，编译时自动赋初值0或空字符。而对自动变量来说，如果不赋初值，则它的值是一个不确定的值。

(4) 虽然静态局部变量在函数调用结束后仍然存在，但其他函数是不能引用它的。

(5)使用静态局部变量降低了程序的可读性，因此尽量少用静态局部变量。

例4.13 输出 1～5 的阶乘值(即1！,2！,3！,4！,5！)。

```
#include <iostream>
using namespace std;
int fac(int);                    //函数声明
int main( )
 {int i;
  for(i=1;i<=5;i++)
   cout<<i<<"!="<<fac(i)<<endl;
return 0;
 }

int fac(int n)
 {static int f=1;                  //f为静态局部变量，函数结束时f的值不释放
  f=f*n;                          //在f原值基础上乘以n
  return f;
 }
```

4.12.5 用extern声明外部变量

全局变量的作用域为从变量的定义处开始直到文件结束。

编译时将全局变量分配在静态存储区。

有时需要用**extern**来声明全局变量，以将全局变量的作用域扩展到多个文件。

1. 在一个文件内声明全局变量

一般都把全局变量的定义放在引用它的所有函数之前，这样可以避免在函数中多加一个**extern**提前引用声明。

例4.14 用extern对外部变量作提前引用声明。

```
#include <iostream>
using namespace std;
int max(int,int);              //函数声明
void main( )
 {extern int a,b;              //对全局变量a,b作提前引用声明
  cout<<max(a,b)<<endl;
 }
int a=15,b=-7;                 //定义全局变量a,b
int max(int x,int y)
 {int z;
  z=x>y?x：y;
  return z;
 }
```

2. 在多文件的程序中声明外部变量

- file1.cpp
- #include <iostream>
- using namespace std;
- extern int a,b;
- int main( )
- {
- cout<<a<<","<<b<<endl;
- return 0;
- }

- file2.cpp
- int a=3,b=4;

4.12.6 用static声明静态外部变量

用static声明外部变量使其只限于被本文件引用，而不能被其他文件引用。例如：

file1.cpp

```
static int a=3;
int main ()
{……}
```

file2.cpp

```
extern int a;
```

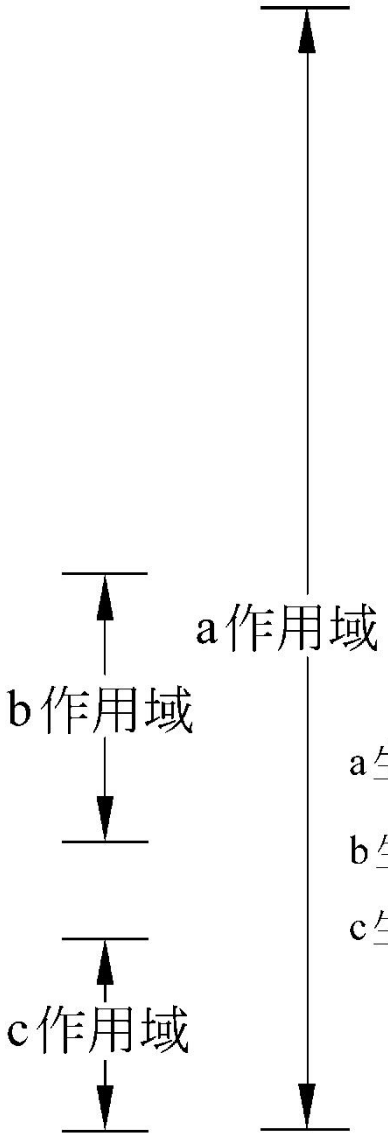
如果已知道其他文件不需要引用本文件的全局变量，可以对本文件中的全局变量都加上static。

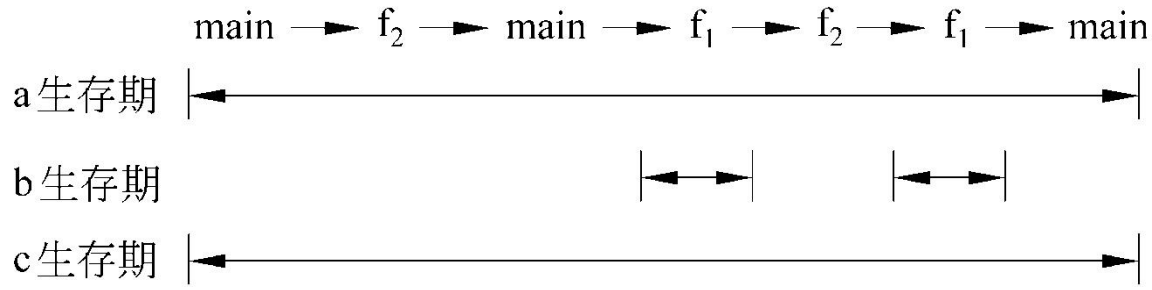
4.13 变量属性小结

- 一个变量除了数据类型以外，还有3种属性：
  - (1) 存储类别 C++允许使用auto，static，register和extern 4种存储类别。
  - (2) 作用域 指程序中可以引用该变量的区域。
  - (3) 存储期 指变量在内存的存储期限。

- auto和 static 只能用于变量的定义语句中。
- extern只能用来声明已定义的外部变量。

```
文件 file1.cpp
int a;
main 函数
{ ⋮
f2( );
 ⋮
f1( );
}
f1 函数
{auto int b;
  ⋮
  f2( );
  ⋮
}
f2 函数
{static int c;
  ⋮
}
```





4.14 关于变量的声明和定义

函数的声明是函数的原型，

函数的定义是函数功能的确立。它是一个文件中的独立模块。

对变量而言，在声明部分出现的变量有两种情况：一种是需要建立存储空间的(如int a;)；另一种是不需要建立存储空间的（如extern int a；）。

前者称为定义性声明(defining declaration)，或简称为定义（definition）。

后者称为引用性声明(referenceing declaration)。

一般为了叙述方便，把建立存储空间的声明称为定义，而把不需要建立存储空间的声明称为声明。

4.15 内部函数和外部函数

函数本质上是全局的。但是,也可以指定函数只能被本文件调用，而不能被其他文件调用。

根据函数能否被其他源文件调用,将函数区分为内部函数和外部函数。

4.15.1 内部函数

内部函数又称静态(static)函数，只能被本文件中其他函数所调用。定义内部函数的一般格式为

static 类型标识符 函数名(形参表)

如

```
static  int  fun(int  a, int  b)
```

如果在不同的文件中有同名的内部函数,互不干扰。

通常把只能由同一文件使用的函数和外部变量放在一个文件中,在它们前面都冠以static使之局部化,其他文件不能引用。

4.15.2 外部函数

(1) 在定义函数时,如果在函数首部的最左端冠以关键字extern,则表示此函数是外部函数，可供其他文件调用。例如

```
extern  int  fun (int  a,  int  b)
```

extern可省略,系统默认为外部函数。

(2) 在需要调用此函数的文件中,用extern声明所用的函数是外部函数。

C++允许在声明函数时省写extern，即函数原型声明。即直接利用函数原型就可扩展函数作用域。

例4.15 输入两个整数，要求输出其中的大者。用外部函数实现。

file1.cpp（文件１）

```
#include <iostream>
using namespace std;
int main( )
 {extern int max(int,int);  //声明在本函数中将要调用在其他文件中定义的max函数
  int a,b;
  cin>>a>>b;
  cout<<max(a,b)<<endl;
  return 0;
 }
```

file2.cpp（文件2）

```
int max(int x,int y)
{int z;
 z=x>y?x：y;
 return z;
 }
```

int max(int,int);　　//省略了extern，即函数原型

利用函数原型扩展函数作用域最常见的例子是#include命令的应用。

在指定的头文件中包含调用库函数时所需的信息，再用#include命令包含相应的头文件。

例如，在头文件cmath中包括了所有数学函数的原型和其他有关信息，用户只需用以下#include命令：

#include <cmath>

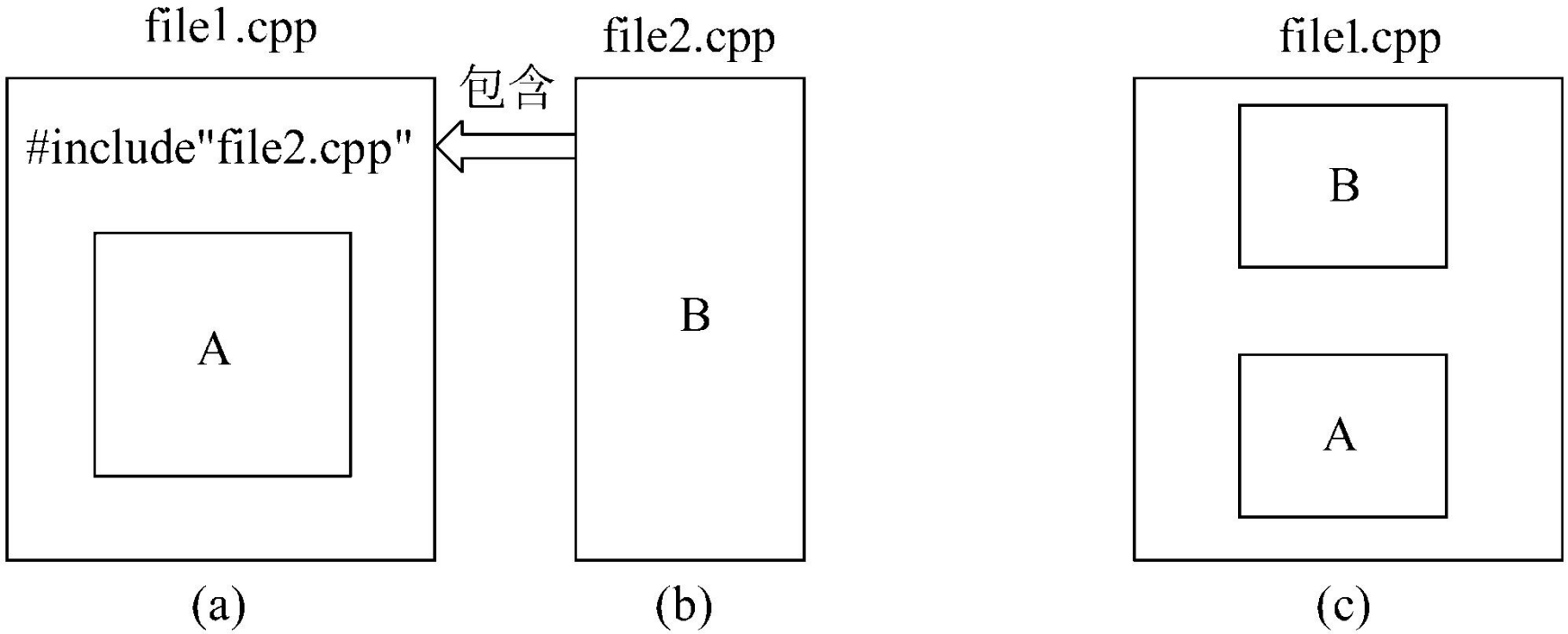
4.16 头文件&文件包含

1.“文件包含”的作用

所谓“文件包含”处理是指一个源文件可以将另外一个源文件的全部内容包含进来，即将另外的文件包含到本文件之中。

C++提供了#include命令用来实现“文件包含”的操作。如在file1.cpp中有以下#include命令：

#include "file2.cpp“


file1.cpp
file2.cpp
包含
#include"file2.cpp"
A
B
(a)
(b)
file1.cpp
B
A
(c)

头文件

许多程序需要使用系统提供的库函数；

C++规定在调用函数前必须对该函数做原型声明；

库函数的数量太多；

库函数的开发者把这些信息写在一个文件中，用户只需将该文件“包含”进来即可。

这种常用在文件头部的被包含的文件称为“标题文件”或“头部文件”。

2. include命令的两种形式

在#include命令中，文件名除了可以用尖括号括起来以外，还可以用双撇号括起来。#include命令的一般形式为

#include <文件名>

或

#include "文件名"

如

#include <iostream>

或

#include "iostream"

二者的区别是： 用尖括号时，系统到系统目录中寻找要包含的文件，如果找不到，编译系统就给出出错信息。

用双撇号时，在双撇号中指出文件路径和文件名。

如果在双撇号中没有给出绝对路径，如

#include "file2.c"

则默认指用户当前目录中的文件。系统先在用户当前目录中寻找要包含的文件，若找不到，再到系统目录中寻找。

如果程序中要包含的是用户自己编写的文件，宜用双撇号形式。

3.头文件的内容

头文件一般包含以下几类内容：

(1) 对类型的声明。

(2) 函数声明。

(3) 内置(inline)函数的定义。

(4) 宏定义。用#define定义的符号常量和用const声明的常变量。

(5) 全局变量定义。

(6) 外部变量声明。如entern int a;

(7) 还可以根据需要包含其他头文件。

不同的头文件包括以上不同的信息，提供给程序设计者使用，这样，程序设计者不需自己重复书写这些信息，只需用一行#include命令就把这些信息包含到本文件了，大大地提高了编程效率。由于有了#include命令，就把不同的文件组合在一起，形成一个文件。因此说，头文件是源文件之间的接口。

4. 关于C++标准库

在C++编译系统中，提供了许多系统函数和宏定义，而对函数的声明则分别存放在不同的头文件中。此外还增加了预定义的模板和类。

新的C++标准库中的头文件一般不再包括后缀.h，例如

```
#include <string>
```

但C++编译系统保留了C的头文件，即提供两种不同的头文件，由程序设计者选用。如

```
#include <iostream.h>          //C形式的头文件
#include <iostream>            //C++形式的头文件
```

建议尽量用符合C++标准的形式，即在包含C++头文件时一般不用后缀。如果用户自己编写头文件，可以用.h为后缀。

作业

- P126：3，4，11，12
- 思考：7，9

```
// 编写一个函数求下列函数值。
// s(x,n)=x/(1!+2!+3!+…+n!)

#include <iostream>
using namespace std;
float fact(int k) // 求K的阶乘函数
{
        int i;
        float f=1.0;
        for(i=1;i<=k;i++)
                f=f*i;
        return f;
}
float s(int n) // 求1!+2!+3!+…+n!的函数
{
        int i;
        float s=0.0;
        for(i=1;i<=n;i++)
                s=s+fact(i);
        return s;
}
int main()
{
        float x,f;
        int n;
        cout<<"Please input x:";
        cin>>x;
        cout<<"Please input n:";
        cin>>n;
        f=x/s(n); // 求本题的表达式值
        cout<<"The result is "<<f;
        return 0;
}
```