计算机程序设计语言（VC++）

第 3 章

函　　数

张晓如，华伟

《C++程序设计基础教程》
人民邮电出版社，2018.05

本章内容

3.1 函数的概念和定义

3.1.1 函数的概念

- 函数是组成源程序的基本模块。
- 函数通过封装在一起的语句实现特定的功能。
- 函数可分为**库函数**和**用户自定义函数**两类。
  - 库函数由系统提供，用户只需要在程序**编译预处理**部分，包含相应的**头文件**便可直接使用；
  - 用户自定义函数通常要**先定义**，然后才能使用。

【例 3-1】 设计一个程序，求角度的正弦值。

- 库函数 sin 用来求正弦值，其参数为弧度；
- 自定义函数 f 将角度转化为弧度；
- 主函数中调用 f 函数和 sin 函数。

3.1 函数的概念和定义

【源程序代码】

```
#include<iostream>
#include<cmath>
using namespace std;
double f(double a) {
    double t;
    t=a*3.1415926/180;
    return t;
}
```

```
int main(){
    double a,r,s;
    cout<<" 请输入一个角度： ";
    cin>>a;  r=f(a);
    cout<<" 弧度： "<<r<<endl;
    s=sin(r);
    cout<<" 正弦
       值： "<<s<<endl;
    return 0;
}
```

- sin 函数是数学库函数：
  - 在标准 C++ 中应包含头文件 cmath ；
  - 在 VC6.0 中应包含头文件 math.h 。
- 任何一个程序有且只能有一个主函数 main 。

3.1 函数的概念和定义

3.1.2 函数定义的基本形式

```
函数类型  函数名（形参列表）          //  函数头部
{
    语句序列                          } 函数体
}
```

- 函数名必须符合自定义标识符的规则。
- 函数体是用花括号括起来的语句系列，实现函数功能。
- 函数定义时的参数称为形参，在形参列表处给出。
  - 每个形参都必须包含参数类型和参数名称；
  - 函数有多个参数时，参数之间用逗号分隔；
  - 如果函数没有参数，则可以在形参列表处用 void 表示；或者缺省，但圆括号不能省略。

3.1 函数的概念和定义

3.1.3 函数类型与返回值

1. 函数类型

- 函数运行结果（返回值）的数据类型。
- 无值型函数的函数类型为 void ，该类函数称为无值型函数，其运行结果不是一个具体的数据。
- 除 void 类型以外的其他类型的函数统称为有值型函数，该类函数的运行结果为某类型的值。如函数类型为：
  - float 时，函数运行结果为一实数；
  - int 时，函数运行结果为一整数；
  - char* 时，函数运行结果为一字符型指针（地址）；
  - 函数的缺省类型为 int 类型，即定义函数时，若未指明函数类型，则该函数的类型为 int 。

3.1 函数的概念和定义

3.1.3 函数类型与返回值

2. return 语句

- 结束函数的运行；若函数为主函数，则结束程序运行。
- 返回有值型函数的运行结果。
- return 语句的格式必须与函数类型一致。

（1）无值型函数

```
return ;                    // 函数运行结束，不返回值
```

（2）有值型函数

- return 表达式； // 函数运行结束，并返回表达式的值

或

- return （表达式）； // 表达式的值即函数运行结果

3.1 函数的概念和定义

3.1.3 函数类型与返回值

【例 3-2 】设计程序求整数的阶乘，要求输出阶乘的数学表达式和阶乘的值。

- 定义函数 fac 求整数 n 的阶乘，并返回其值； fac 函数的函数类型为 int ，具有一个整型参数（传递整数）。
- 定义函数 print 输出阶乘的数学表达式，即输出：
  n ！ =1*2*3*…*（ n-1 ）*n=
  print 函数无返回值，其函数类型为 void 。
- 在主函数中调用 print 函数输出阶乘表达式；调用 fac 函数求阶乘，并将其运行结果（函数的返回值）输出。

3.1 函数的概念和定义

【源程序代码】

```
#include<iostream>
using namespace std;
int fac(int n){
    int t=1;
    for(int i=2;i<=n;i++)
        t*=i;
    return t;
}
```

```
int main(){
    int num;
    cout<<" 请输入一个整数： ";
    cin>>num;
    print(num);                  // A
    cout<<fac(num)<< endl;  // B
    return 0;
}
```

```
void print(int n){
    cout<<n<<"!=";
    if(n>1)
        for(int i=1;i<=n;i++)
            if(i==n) cout<<i<<'=';
            else cout<<i<<'*';
}
```

程序运行结果

请输入一个正整数： 4

4!=1*2*3*4=24

注意：

- A 行为无值型函数的调用。
- B 行为有值型函数的调用。

3.2 函数的调用

3.2.1 函数调用的基本形式

函数名(实参列表)

- 调用函数时，函数名前没有类型。
- 函数调用时的参数称为实际参数，简称**实参**。
  - 实参只有名称，**没有类型**；
  - 多个参数时，各实参之间用逗号隔开；
  - 实参可以是变量、常量、表达式或函数调用表达式；
  - 每个实参通常应有**确定的值**；
  - 实参与形参的个数、类型和顺序通常应该保证一致；
- 调用无参函数时，无需提供参数，但“( )”不能少。
- 注意有值型函数与无值型函数的调用方式的区别。

3.2 函数的调用

3.2.1 函数调用的基本形式

【例 3-3】 设计程序求 3 个整数的最大公约数。

- **穷举法**：从所给整数中的最小者开始，依次向下遍历，直到找到最大公约数为止。
- **辗转相除法（欧几里德算法）**：不断地用两数相除得到的余数去除除数，直到余数为 0 时 ，除数即为最大公约数。

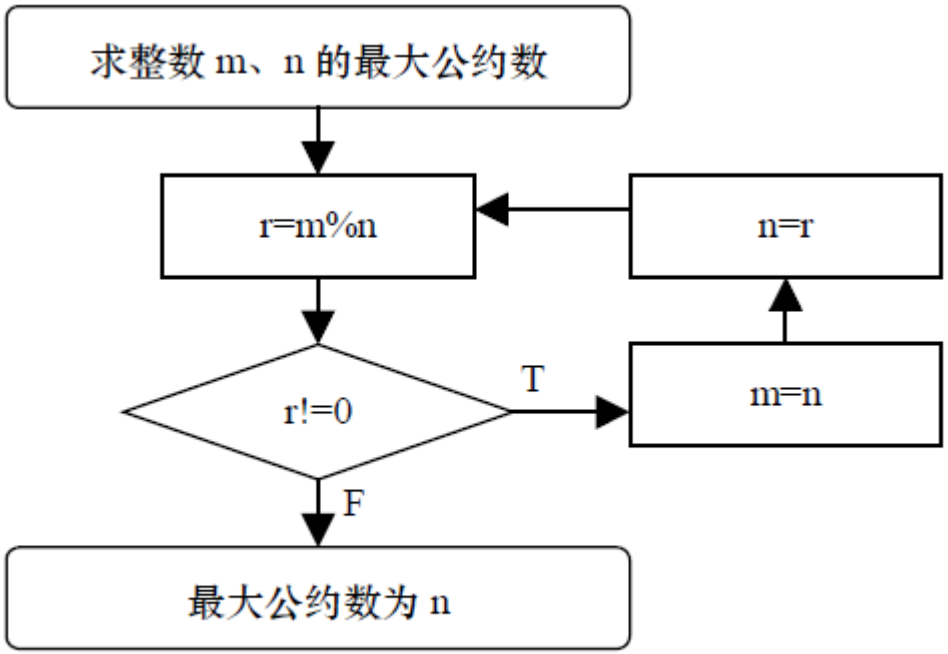


图 3-1 欧几里德算法

3.2 函数的调用

3.2.1 函数调用的基本形式

【例 3-3】 设计程序求 3 个整数的最大公约数。

- 欧几里德算法只能直接求两个整数的最大公约数。
- 定义 gcd 函数求两个整数 m 和 n 的最大公约数，即 gcd 函数有两个整型形参 m 和 n。
- 求 3 个整数 a、b、c 的最大公约数时：
  - 首先调用 gcd 函数求 a 和 b 的最大公约数 g；
  - 再调用 gcd 函数，求出 g 和 c 的最大公约数；
  - 则可求出 3 个整数的最大公约数。

3.2 函数的调用

【源程序代码】

```
int gcd(int m,int n){
    int r;
    while(r=m%n){  // A
       m=n;
       n=r;
    }
    return n;
}
```

```
int main(){
    int a,b,c,g;
    cout<<" 输入 3 个正整
       数： ";
    cin>>a>>b>>c;
    g=gcd(a,b);          // B
    g=gcd(g,c);          // C
    cout<<g<<endl;       // D
    return 0;
}
```

- A 行将 m 和 n 相除得到的余数 r 作为循环条件。
- B 行和 C 行可用一条调用语句实现，即“ g=gcd(gcd(a,b),c);” ；此时，程序首先执行内层的函数调用 gcd(a,b) 。
- B 行、 C 行和 D 行可用一条语句 “ cout<<gcd(gcd(a,b),c)<<endl ;” 实现

3.2 函数的调用

3.2.2 函数的嵌套调用

- 函数不允许**嵌套定义**，即不能在函数体内再定义函数。
- 函数可以**嵌套调用**，即在调用函数的过程中再调用函数。

【例 3-4】设计程序求代数式 $1^k+2^k+\cdots\cdots+n^k$ 的值，其中 k 为整数。

- 该程序求的是 n 项的和，而每项的值为 i^k。
- 设计函数 sum 求 n 项的和，设计函数 powers 求 i^k。
- 主函数在调用 sum 函数的过程中再调用 powers 函数，这属于函数的嵌套调用。

3.2 函数的调用

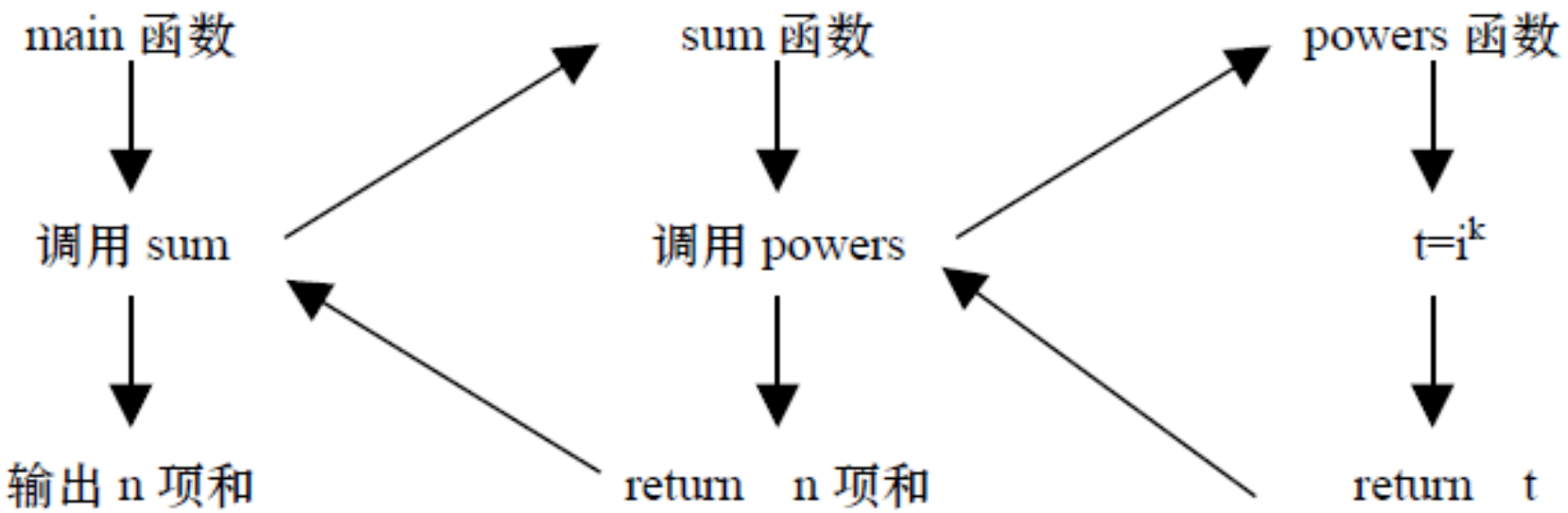


图 3-2 函数调用过程

- 函数 powers 的函数类型为 int ，有两个整型参数 i 和 k ；
- 函数 sum 的函数类型为 int ，有两个整型参数 n 和 k 。

3.2 函数的调用

【源程序代码】

```
#include<iostream>
using namespace std;
int powers(int i,int k) {
    int t=1,j;
    for(j=1;j<=k;j++) t*=i;
    return t;
}                    // 累乘求 i^k
```

```
int sum(int n,int k) {
    int s=0;
    for(int i=1;i<=n;i++)
        s+=powers(i,k);
    return s;
}            // 累加求 n 项的和
```

```
int main(){
    int n,k;  cout<<"Input n and k: ";   cin>>n>>k;
    cout<<"Sum of "<<k<<" powers from 1 to "<<n<<" = ";
    cout<<sum(n,k)<<endl;
    return 0;
}
```

3.2 函数的调用

【游戏环节】

```
#include<iostream>
using namespace std;
double powers(double i,int k){
    double t=1.0;  int j;
    for(j=1;j<=k;j++) t*=i;
    return t;
} //  累乘求 i k
```

```
int main(){
      double p; int q;
       cin>>p>>q; //1.05,365; 0.95,365
       cout<<p<<"^"<<q<<"="<<powers(p,q)<<endl;
       system("pause");
       return 0;
    }
```

3.2 函数的调用

3.2.3 函数的递归调用

- 在函数的调用过程中直接或间接地调用自身函数。
  - 直接递归：在函数体内直接调用自身函数。
  - 间接递归：在函数体内通过其他函数间接调用自身函数。

【例 3-5】编程用递归法求 n!。

- n! 的递归定义：$n!=\begin{cases}1 & n=0\text{或}n=1\\ n\times(n-1)! & n>1\end{cases}$

- 设计函数 long f (int n) 求 n!。
  - 函数体中通过调用语句 f(n-1) 求 (n-1)!，得 n* f(n-1)；

3.2 函数的调用

【源程序代码】

```
#include<iostream>
using namespace std;
long f(int n) {                          // 求 n! 的递归函数
    if(n==1||n==0)return 1;                    // 递归结束条件
    else return n*f(n-1);                      // 递归公式
}
```

```
int main(){
    int n;
    cout<<" 请输入一个正整数： "; cin>>n;
    cout<<n<<"!="<<f(n)<<endl;
    return 0;
}
```

3.2 函数的调用

3.2.3 函数的递归调用

- 递归算法是将原问题转换为用同样方法解决的规模更小的问题，其关键有两点：
  - 递归公式：原问题与新问题之间的关系；
  - 结束条件：转换到何时结束。
- 例 3-5 中，原问题是求 n!，新问题是求 (n-1)！，其关系是 n!=n×(n-1)!，即 f(n)=n*f(n-1)。
- 例 3-5 中，有两个递归结束条件，即 n 为 1 或 n 为 0，其阶乘为确定值 1，此时无需再递归调用。
- 递归算法通常可用下列范式实现：

  if(递归结束条件) 确定操作；
  else 递归公式；

3.3 函数的参数传递

函数的参数用于在主调函数与被调函数之间传递数据，根据所传递的数据形式，可以将 C++ 语言中的参数传递方式分为**值传递**、**地址传递**和**引用传递** 3 种类型。

3.3.1 函数的值传递

- **形参**为基本类型变量、结构体类型变量和类类型变量等。
- **实参**为相应的变量、常量或表达式等。
- 值传递是一种**单向传递**，即只能把实参传递给形参，对形参的操作不能改变实参的值。

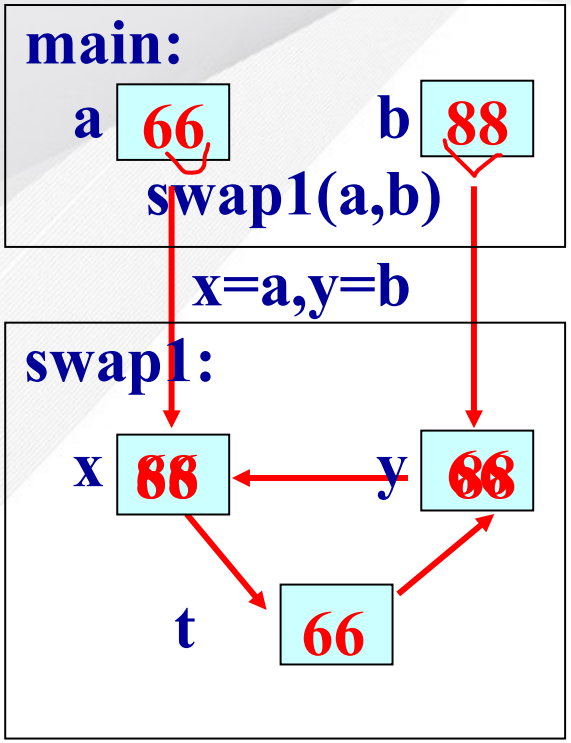
【例 3-7】 分析下列程序的输出结果。

3.3 函数的参数传递

【源程序代码】

```
void swap1(int x, int y) {
    int t;
    t=x; x=y; y=t;
    cout<<x<<','<<y<<'\n';
}
```

```
int main(void){
    int a=66,b=88;
    cout<<a<<','<<b<<'\n';
    swap1(a,b);
    cout<<a<<','<<b<<'\n';
    return 0;
}
```



程序运行结果

```
66, 88
88, 66
66, 88
```

3.3 函数的参数传递

3.3.2 函数的引用传递

1. 引用变量概述

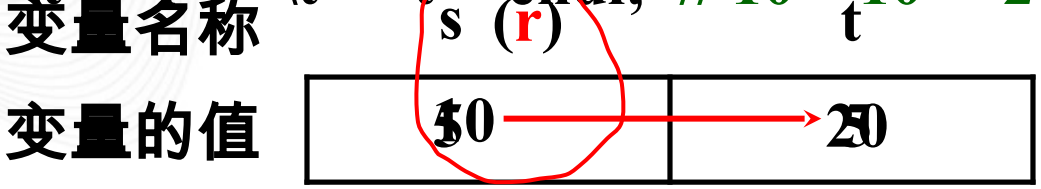
引用是给已定义的变量重新命名，其定义的一般格式如下：

数据类型 & 引用名 = 变量名；

- “&”号是引用标志，表示所定义的变量是个引用变量；
- 不能与取地址运算符混淆。

如：

```
int s=5 , &r=s,t=s ; // 将 r 定义为 s 的别名， t 是一个新变量
r=10,t=20;
cout<<s<<'\t'<<r<<'\t'<<t<<endl; // 10 10 20
```



3.3 函数的参数传递

3.3.2 函数的引用传递

2. 引用传递方式

- 形参为某种类型的引用。
- 实参为相应的变量。
- 引用传递：
  - 形参是对实参的重新命名；
  - 形参和实参是同一个内存空间的两个名称；
  - 对形参的操作就是对实参的操作。

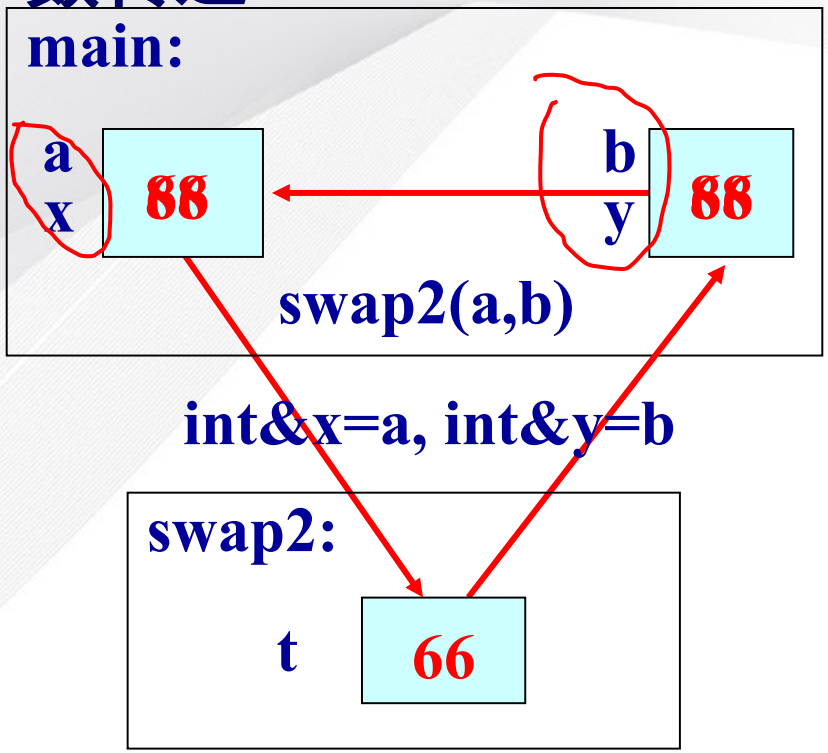
【例 3-9】 分析下列程序的输出结果。

3.3 函数的参数传递

【源程序代码】

```
void swap2(int &x, int &y) {
    int t;
    t=x; x=y; y=t;
    cout<<x<<','<<y<<'\n';
}
```

```
int main(void){
    int a=66,b=88;
    cout<<a<<','<<b<<'\n';
    swap2(a,b);
    cout<<a<<','<<b<<'\n';
    return 0;
}
```



程序运行结果

```
66, 88
88, 66
88, 66
```

3.3 函数的参数传递

3.3.3 函数的地址传递

1. 指针变量概述

- 指针变量是存储地址（指针）的变量，其定义的一般格式如下：

  数据类型 * 变量名；

  例如：

```
char c='A',*p1;   // 定义普通变量 c 和指针变量 p1
p1=&c;            // p1 中存放 c 的地址，即 p1 指向变量 c
int  x,*p2=&x;    // 定义变量 x 并用 &x 初始化指针变量 p2
```

  - 例题中“ *”号是指针标志，表示所定义的变量是指针变量；定义的指针变量是 p1 和 p2 ，不是 *p1 和 *p2 ；或者说变量 p1 的类型是 char* ，变量 p2 的类型是 int* 。
  - 例题中的“ &”是取地址运算符， &c 表示取变量 c 的地址。

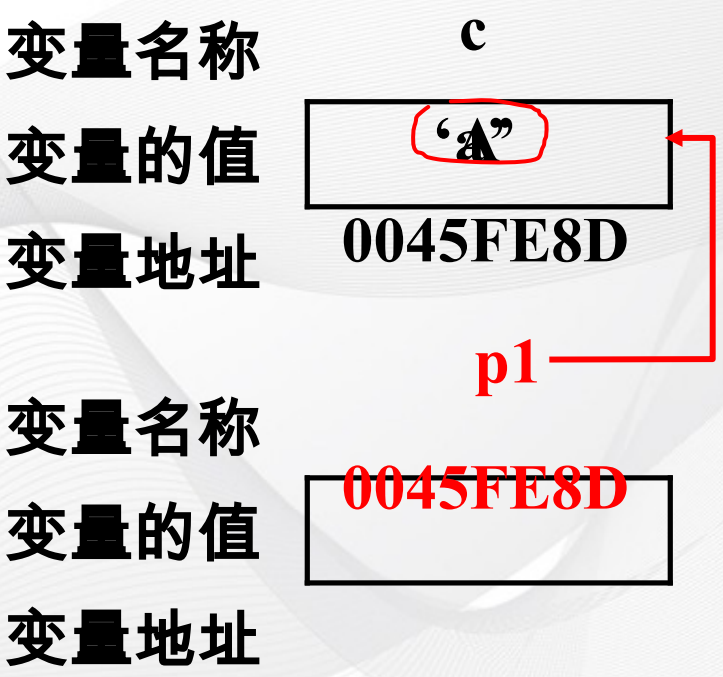
- 各种指针都占 4 个字节，但指针必须与其所指对象类型相同

3.3 函数的参数传递

3.3.3 函数的引用传递

- 引用传递时，形参前的运算符 & 是引用运算符，不是取地址运算符，表示形参是一个引用变量；其对应的实参必须是相同类型的变量，不能是地址。

3.3 函数的参数传递




```
char c='A', *p1; // * 表示 p1 是指针
p1=&c;           //& 取地址运算符
*p1='a'; // * 取指针所指值运算符
cout<<p1<<endl;      // 0045FE8D
cout<<*p1<<endl;     // a
```

第 1 行定义变量时，* 表示所定义的变量是指针变量；第 3 行对于已定义的指针变量，* 表示取指针变量所指的值。

- 指针的值与指针所指的值
  - 指针的值是指针变量中保存的地址，如 p1 的值为 c 的地址；
  - 指针所指的值是指针所指内存空间的值，如 *p1 为 c 的值，原来为'A'，重新赋值后为'a'。

3.3 函数的参数传递

3.3.3 函数的地址传递

2. 地址传递方式

- 形参为某种类型的指针。
- 实参为相应的地址：
  - 变量的地址；
  - 保存了某个地址的指针变量。
- 被调用函数中：
  - 既可以操作指针；
  - 也可以操作指针所指的内存空间。
- 地址传递可以改变实参的值。

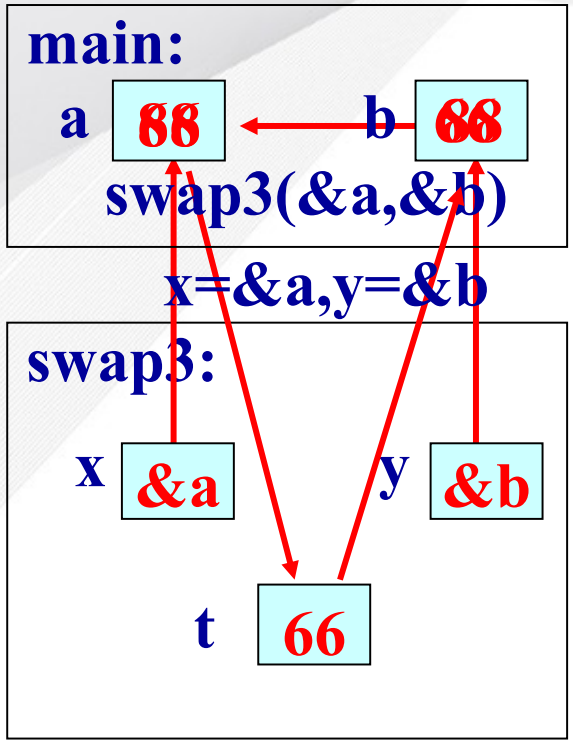
【例 3-8】 分析下列程序的输出结果。

3.3 函数的参数传递

【源程序代码】（交换 *x 和 *y ）

```
void swap3(int *x, int *y) {
    int t;
    t=*x; *x=*y; *y=t;
    cout<<*x<<','<<*y<<'\n';
}
```

```
int main(void){
    int a=66,b=88;
    cout<<a<<','<<b<<'\n';
    swap3(&a,&b);
    cout<<a<<','<<b<<'\n';
    return 0;
}
```



程序运行结果

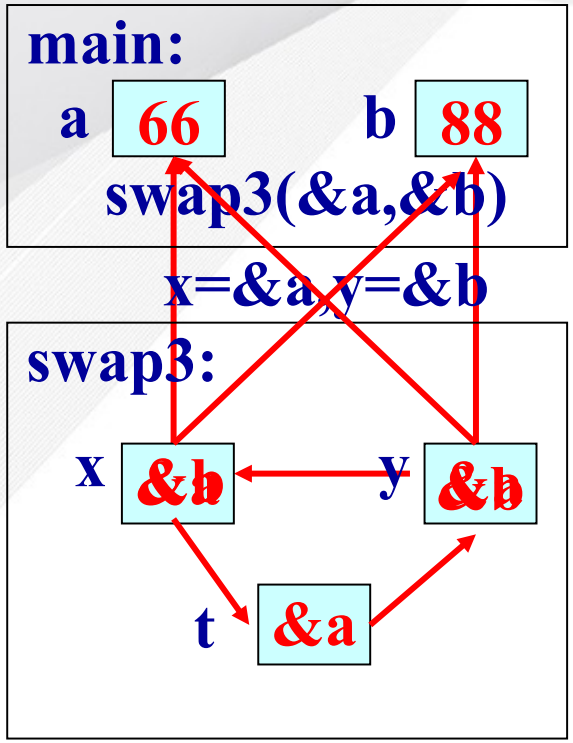
```
66, 88
88, 66
88, 66
```

3.3 函数的参数传递

【修改 swap2 函数】（交换 x 和 y ）

```
void swap3(int *x, int *y) {
    int *t;
    t=x; x=y; y=t;
    cout<<*x<<','<<*y<<'\n';
}
```

```
int main(void){
    int a=66,b=88;
    cout<<a<<','<<b<<'\n';
    swap3(&a,&b);
    cout<<a<<','<<b<<'\n';
    return 0;
}
```



程序运行结果

```
66, 88
88, 66
66, 88
```

3.3 函数的参数传递

3.3.3 函数的引用传递

- **指针传递**时，形参前的运算符 * 是指针运算符，表示形参是一个指针变量；其对应的实参必须是相同类型的地址。
- **引用传递**时，形参前的运算符 & 是引用运算符，不是取地址运算符，表示形参是一个引用变量；其对应的实参必须是相同类型的变量，不能是地址。
- 三种传递方式：
  - 值传递**一定不会**改变实参的值；
  - 引用传递**一定会**改变实参的值；
  - 地址传递**可能**会改变实参的值。
- 若通过参数带回操作结果，应该选用引用传递或地址传递；且地址传递时，必须通过取值运算操作内存空间。

3.4 函数的其他特性

3.4.1 函数参数的默认值

- 给自定义函数的参数指定一个**默认值**。
  - 在**定义**函数时，列出参数的默认值；
  - 在**说明**函数原型时，列出参数的默认值；
  - 具有默认值的参数都必须位于形参列表的**右侧**。
- 调用具有默认参数的函数时：
  - 若提供了**实参**，则以提供的**实参**调用函数；
  - 若没有提供实参，则以**默认值**作为实参调用函数；
  - 提供的实参个数不能少于没有默认值的参数个数。

【例 3-10 】 分析下列程序的输出结果。

3.4 函数的其他特性

【源程序代码】

```
#include <iostream>
using namespace std;
void f(double x=9.9) {  //A
    cout<<"x="<<x<<endl;
}
```

```
int main(void){
    f(8.8);   // B
    f();      // C, 等同于 f(9.9)
    void f(double=7.7);  // D
    f(6.6);   // E
    f();      // F, 等同于 f(7.7)
    return 0;
}
```

程序运行结果

```
x=8.8     //  B 行输出
x=9.9     //  C 行输出
x=6.6     //  E 行输出
x=7.7     //  F 行输出
```

在不同的作用域内，可以设置不同的默认值；调用时默认值由最近的定义或说明决定。

3.4 函数的其他特性

3.4.2 函数的重载

- 函数重载是指一组名称相同，参数不同的函数。
  参数个数、或参数类型、或参数次序不同。
- 函数类型不同不能作为函数重载的依据。
- 调用重载函数时，根据提供的实参确定调用哪个函数。

【例 3-11 】定义重载函数，分别求解三角形、矩形和圆的面积。

- 函数名称皆为 area 、函数类型都为 double ；
- 函数参数不同：三角形为 double x, double y, double z ；矩形为 double x, double y ；圆为 double x 。

3.4 函数的其他特性

【源程序代码】

```
double area(double x, double y, double z) {          // A
    double s=(x+y+z)/2;
    return sqrt(s*(s-x)*(s-y)*(s-z));
}
double area(double x, double y) {  return x*y;  }   // B
double area(double x) {  return PI*x*x;  }           // C
```

```
int main(){
    cout<<area(3)<<endl;        //  调用 C 行 1 个参数的 area 函数
    cout<<area(3,4)<<endl;      //  调用 B 行 2 个参数的 area 函数
    cout<<area(3,4,5)<<endl;    //  调用 A 行 3 个参数的 area 函数
    return 0;
```

3.4 函数的其他特性

3.4.3 内联函数

- 将简单函数定义为**内联函数**，可在编译时将函数代码直接插入调用处，以提高程序的运行效率。
- 内联函数定义的一般格式

```
inline 函数类型 函数名（形参列表）
{
        语句序列
}
```

- 内联函数定义时：
  - 内联函数仅限于简单的函数，函数体内不应包含循环语句、 switch 分支语句和复杂嵌套的 if 语句。
  - 用户指定的内联函数，系统不一定处理为内联函数。

3.4 函数的其他特性

3.4.4 exit 函数和 abort 函数

- 库函数，头文件 <cstdlib> ；
- 功能是终止程序的执行。

1. exit 函数的一般格式

   exit（表达式）；

2. abort 函数的一般格式

   abort（ ）；

3.6 变量的作用域与存储类型

3.6.1 变量的作用域

- 变量的**作用域**是指变量的**有效使用范围**，包括**块作用域**、**文件作用域**、**函数原型作用域**、**函数作用域**和**类作用域**。
- 按作用域的不同，可将变量分为**局部变量**（块作用域）和**全局变量**（文件作用域）两类。

1．块作用域

- 块内定义的变量具有块作用域，只能在该语句块内使用。
- 同一个块内不允许定义名称相同的变量。
- 函数体是一个块，函数形参的作用域为函数体。

【例 3-15】 分析下列程序的输出结果。

3.6 变量的作用域与存储类型

【源程序代码】

```
#include <iostream>
using namespace std;
void f(int);
int main( ){
    int x=1;                    // A
    {
        int x=2;                // B
        cout<<x<<'\t';          // C
    }
    f(x);                       // D
    return 0;
}
```

```
void f(int a){
    int b=3;                    // E
    {
        int c=b++;              // F
        cout<<c<<'\t';
    }
    cout<<a<<'\t';
    cout<<b<<endl;              // G
}
```

程序运行结果

```
2  3  1  4
```

3.6 变量的作用域与存储类型

3.6.1 变量的作用域

2．文件作用域

- 函数外定义的变量是**全局变量**，具有**文件作用域**，又称**外部变量**。
- 全局变量通常也应该先定义后使用；若使用在前，定义在后，则使用前要用关键字 **extern** 说明。
- 全局变量具有默认初始值 0。
- 当局部变量与全局变量同名时，按照局部优先原则，默认使用的是局部变量；要使用全局变量可在变量名称前加作用域运算符“ **::**”。

【例 3-16】 分析下列程序的输出结果。

3.6 变量的作用域与存储类型

【源程序代码】

```
#include <iostream>
using namespace std;
int m=3;   // 全局变量
void f(){
    extern int n;
    // A 全局变量说明
        // 不能初始化
    m++;
    n++;
}
int n;        // 全局变量
```

```
int main() {
    int m=6; // 局部变量
    cout<<m<<'\t'<<n<<endl;   // B
    f();
    cout<<::m<<'\t'<<n<<endl; // C
    return 0;
}
```

程序运行结果

```
6  0
4  1
```

3.6 变量的作用域与存储类型

3.6.1 变量的作用域

3. 函数原型作用域

- 函数原型说明语句中形参的作用域仅限于该原型说明语句，属于**函数原型作用域**。
- 函数原型说明语句中可以省略函数形参的名称，只需说明形参的类型。

```
int fun(float x, float y);
```

等价于：

```
int fun(float, float);
```

4. 函数作用域与类作用域

3.6 变量的作用域与存储类型

3.6.2 变量的存储类型

一个完整的变量说明除了包括**数据类型**和**变量名称**外，还应包括变量的**存储区域**，变量的存储区域通过变量的**存储类型**加以说明。变量定义的完整格式为：

存储类型 数据类型 变量名；

- **静态**存储类型：有默认初始值 **0** ，生命期为整个源程序
  - **静态**类型
  - **外部**类型
- **动态**存储类型：无默认初始值，生命期为作用域
  - **自动**类型
  - **寄存器**类型

3.6 变量的作用域与存储类型

3.6.2 变量的存储类型

1．自动类型变量

- 存储类型的关键字为 auto ；
- 局部变量的默认存储类型为自动类型。

```
void f(int a){
    float x;
}
```

等同于：

```
void f(auto int a){
    auto float x;
}
```

3.6 变量的作用域与存储类型

3.6.2 变量的存储类型

2．静态类型变量

- 存储类型的关键字为 static ；
- 可分为局部静态变量和全局静态变量。
  - 局部静态变量：块中用 static 说明的变量，具有块作用域，作用域外存在但不可用，进作用域延续前值。
  - 全局变量总是静态存储，用 static 说明的全局变量仅限于在定义它的源程序中使用，而未使用 static 说明的全局变量可以被其他文件使用。

【例 3-16】 分析下列程序的输出结果。

3.6 变量的作用域与存储类型

【源程序代码】

```
# include <iostream>
using namespace std;
void f(){
    auto int i=0;        // A
    static int j,k=1;    // B
    i++;
    j++;
    k++;
    cout<<i<<'\t'<<j<<'\t'<<k<<'\n';
}
```

```
int main() {
    f();
    f();
    return 0;
}
```

程序运行结果

```
1  1  2
1  2  3
```

3.6 变量的作用域与存储类型

3.6.2 变量的存储类型

3．寄存器类型变量

- 存储类型的关键字为 register ；
- 建议系统在寄存器分配存储空间。

4．外部变量

- 函数外部说明的变量，存储类型的关键字为 extern 。
- 在同一源程序中，使用在前、定义在后的全局变量，使用前要说明为外部变量；
- 在一个源程序文件中，使用其他文件中定义的全局变量时，要说明为外部变量。

3.7 程序举例

【例 3-22】 设计程序求 [n1 ， n2] 区间内的所有素数。要求判断一个整数是否为素数和输出分别用一个函数实现，并按每行 5 个素数的方式输出。

- 设计函数 f 判断参数 n 是否为素数：
  - 若参数 n 是素数，则 f 函数返回 1 ，否则返回 0 ；
  - int f(int n) ；
- 设计函数 show 输出 n1 至 n2 区间的素数：

  void show(int n1,int n2) ；
- 主函数中定义并输入区间范围；然后调用 show 函数输出其中的素数；在 show 函数中再调用 f 函数判断所处理的整数是否为素数，若是则按指定格式输出。

3.7 程序举例

【源程序代码】

```
#include <iostream>
#include <cmath>
using namespace std;
int f(int n){
    for(int i=2;i<=sqrt(n);i++)
        if(n%i==0)return 0;
    return 1;      // A
}
/*A 行为循环结束后，若没有返回 0 ，则返回 1 ；前面不能加
  else ，否则第 1 次循环时，不返回 0 ，就返回 1 ，不能实现
  2 与 √n 之间所有整数的遍历。 */
```

3.7 程序举例

【源程序代码】

```
void show(int n1,int n2){
    int count=0;
    while(n1<=n2){
        if(f(n1)){  // 等同于 f(n1)==1 ，若 n1 是素数，则操作：
            cout<<n1<<'\t';
            count++;
            if(count%5==0)
                cout<<endl;
        }
        n1++;
    }
    cout<<endl;
}
```

3.7 程序举例

【源程序代码】

```
int main(){
    int number1,number2;
    cout<<" 请输入区间范围 (n1 n2) : ";
    cin>>number1>>number2;
    show(number1,number2);
    return 0;
}
```

程序运行结果

```
请输入区间范围 (n1 n2) : 100
  200
101 103 107 109 113
127 131 137 139 149
151 157 163 167 173
179 181 191 193 197
199
```

3.7 程序举例

【例 3-23】 利用函数求 e^x 的近似解，要求最后一项小于 10^{-6}，其中，求 e^x 的近似公式如下。

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^{n-1}}{(n-1)!} + \frac{x^n}{n!}$$

- 定义递归函数 int f(int n)，求 n!。
- 定义递归函数 int fun(int x,int n)，求 x^n。
- 定义函数 double sum(int x)，求各项的和，即 e^x 的值。
- 主函数中输入 x 的值调用 sum 函数求 e^x 的值。在 sum 函数中调用 fun 函数求通项的分子，即 x^n；调用 f 函数求通项的分母，即 n！。

3.7 程序举例

【源程序代码】

```
#include <iostream>
using namespace std;
int f(int n){                    // 求 n ！
    if(n==0||n==1) return 1;
    else return n*f(n-1);
}
int fun(int x,int n){      // 求 x^n
    if(n==0) return 1;
    else if(n==1)return x;
        else return fun(x,n/2)*fun(x,n-n/2);
}
```

3.7 程序举例

【源程序代码】

```
double sum(int x){
    double s=0,t;
    int n=0;
    do{
        t=1.0*fun(x,n)/f(n);
        s+=t;
        n++;
    }while(t>1e-6);
    return s;
}
/* 函数 fun 和 f 的返回值皆为整型，为防止整除，分子首先乘
   1.0*/
```

3.7 程序举例

【源程序代码】

```
int main(){
    int x;
    cout<<" 请输入正整数 x ： ";
    cin>>x;
    cout<<'e'<<'^'<<x<<'='<<sum(x)<<'\n';
    return 0;
}
```

程序运行结果
请输入正整数 x ： 2
e^2=7.38867

通项 $x^n/n!$ ，可用下列函数实现：

```
double fun(int x,int n){
    if(n==0) return 1;
    else return fun(x,n-1)* x/n;
}
```

$$\frac{x^n}{n!}=\begin{cases}1 & n=0\\ \dfrac{x^{n-1}}{(n-1)!}\times\dfrac{x}{n} & n>0\end{cases}$$

3.8 习题

1. 设计程序用迭代法求方程 $3x^3-2x^2+5x-7=0$ 在 1 附近的一个根，精确达到 10^{-6}。牛顿迭代公式为 $x=x-f(x)/f'(x)$。要求定义两个函数分别求 $f(x)$ 和 $f'(x)$。

2. 设计程序求 100 以内的孪生素数对，要求用一个函数判断某一正整数是否为素数。所谓孪生素数对，是指差为 2 的一对素数。

3. 设计程序求组合数 C(m,r)。其中 $C(m,r)=m!/(r!(m-r)!)$，m 和 r 为正整数，且 m>r。要求设计两个函数分别求阶乘和组合数。

3.8 习题

4. 设计程序用递归法将十进制整数转换为十六进制整数。用递归法将十进制整数 n 转换为十六进制整数的方法是求出 n 与 16
相除的余数 t（ t=n%16 ），并逆序输出；然后以 n/16 作为参数调用递归函数，直到参数小于 10 为止。为了实现逆序输出，应将输出语句置于递归语句之后；将大于等于 10 的余数 t 转换为相应十六进制数的方法是“ char('A'+t-10)”。

5. 设计程序，分别用宏定义和函数求圆的面积，其中圆的半径可以为表达式。