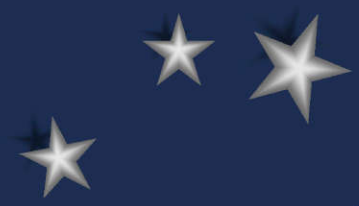
C++语言程序设计

第三章 函数

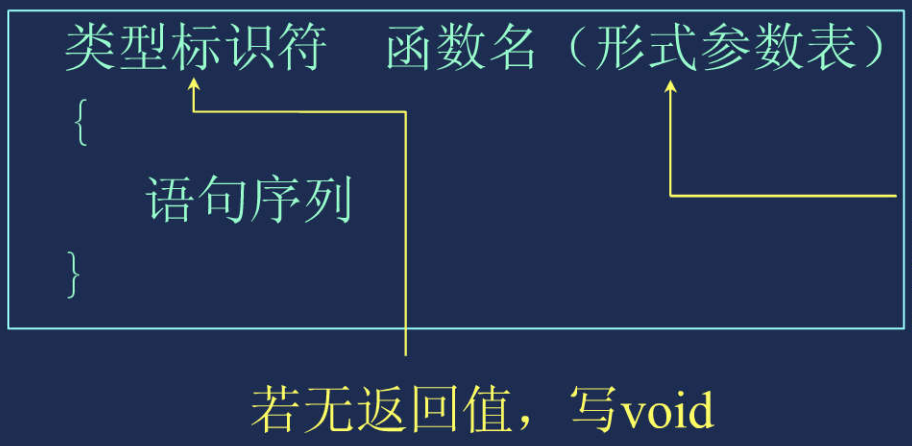
本章主要内容

- 函数的定义和使用
- 内联函数
- 带默认形参值的函数
- 函数重载
- C++系统函数



函数的定义

- 函数是面向对象程序设计中，对功能的抽象
- 函数定义的语法形式



函数的定义

- 形式参数表（形参类型定义必须写在参数表中）

  $\langle type_1 \rangle$ $name_1$, $\langle type_2 \rangle$ $name_2$, ..., $\langle type_n \rangle$ $name_n$

- 函数的返回值
  - 由 return 语句给出，例如：

```
return  0
```

  - 无返回值的函数（void类型），不必写return语句。

函数的调用

- 调用前先声明函数原型（除非函数定义出现在函数调用之前）：
  - 在调用函数中，或程序文件中所有函数之外，按如下形式说明：

    类型标识符 被调用函数名（含类型说明的形参表）；
- 调用形式

  函数名（实参列表）
- 嵌套调用
  - 嵌套调用，但不允许嵌套定义。
  - 递归调用：函数直接或间接调用自身。

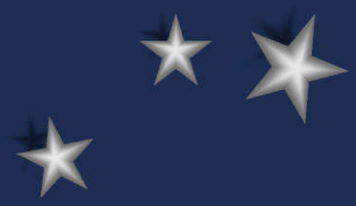
例3-1编写一个求x的n次方的函数

```
#include <iostream>
using namespace std;
double power(double x, int n);
void main()
{ cout<<"5 to the power 2 is "
            <<power(5,2)<<endl;
 }
double power(double x,int n)
{ double val=1.0;
  while(n--) val=val*x;
  return(val);
 }
```

例3-2 数制转换

题目：

输入一个8位二进制数，将其转换为十进制数输出。



```
#include <iostream>
using namespace std;
double power (double x, int n);
void 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;
}
double power (double x, int n)
{
   double val = 1.0;
   while(n--)  val*=x;
   return(val);
}
```

运行结果：

```
Enter an 8 bit binary number
01101001
Decimal value is  105
```

例3-3编写程序求π的值

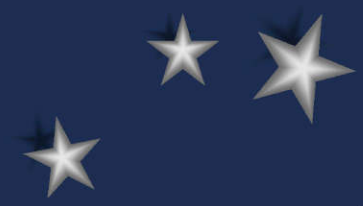
$$\pi = 16\arctan\left(\frac{1}{5}\right) - 4\arctan\left(\frac{1}{239}\right)$$

其中arctan用如下形式的级数计算：

$$\arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots$$

直到级数某项绝对值不大于10^{-15}为止；

π和x均为double型。



```
#include<iostream>
using namespace std;
void main()
{
  double a,b;
  double arctan(double x);//函数原型声明
  a=16.0*arctan(1/5.0);
  b=4.0*arctan(1/239.0);
  //注意：因为整数相除结果取整，
  //如果参数写1/5，1/239，结果就都是0
  cout<<"PI="<<a-b<<endl;
 }
```

```
double arctan(double x)
{ int i;
  double r,e,f,s;
  r=0;e=x;i=1;s=1;
  while(e/i>1e-15)
  {
    f=e/i;
    r+=f;
    e=e*x*x;
    i+=2;
    s=-s;
   }
  return  r;
 }
```

运行结果：

PI=3.14159

例3-4

- 寻找并输出11~999之间的数m，它满足m、m^2和m^3均为回文数。
  - 回文：各位数字左右对称的整数。
    例如：11满足上述条件
    $11^2=121$，$11^3=1331$。
- 分析：
  - 10取余的方法，从最低位开始，依次取出该数的各位数字。按反序重新构成新的数，比较与原数是否相等，若相等，则原数为回文。

```
#include <iostream>
using namespace std;
void main()
{
  bool symm(long n);
  long m;
  for(m=11; m<1000; m++)
  if (symm(m)&&symm(m*m)&&symm(m*m*m))
    cout<<"m="<<m<<"  m*m="<<m*m
        <<"  m*m*m="<<m*m*m<<endl;
}
```

```
bool symm(long n)
{
  long i, m;
  i=n ; m=0 ;
  while(i)
  {
   m=m*10+i%10;
   i=i/10;
  }
  return( m==n );
}
```

运行结果：

```
m=11    m*m=121      m*m*m=1331
m=101   m*m=10201    m*m*m=1030301
m=111   m*m=12321    m*m*m=1367631
```

例3-5

计算如下公式，并输出结果：

$$k=\begin{cases}\sqrt{\sin^2(r)+\sin^2(s)} & \text{当}r^2 \le s^2\\ \dfrac{1}{2}\sin(r\times s) & \text{当}r^2 > s^2\end{cases}$$

其中r、s的值由键盘输入。sin x的近似值按如下公式计算，计算精度为10^{-6}：

$$\sin x=\frac{x}{1!}-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots=\sum_{n=1}^{\infty}(-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!}$$

```
#include <iostream>
#include<cmath>
using namespace std;
void main()
{
  double k,r,s;
  double tsin(double x);
  cout<<"r=";
  cin>>r;
  cout<<"s=";
  cin>>s;
  if (r*r<=s*s)
    k=sqrt(tsin(r)*tsin(r)+tsin(s)*tsin(s));
  else
    k=tsin(r*s)/2;
  cout<<k<<endl;
}
```

```
double tsin(double x)
{
  double p=0.000001,g=0,t=x;
  int n=1;
  do {
     g=g+t;
     n++;
     t=-t*x*x/(2*n-1)/(2*n-2);
  }while(fabs(t)>=p);
  return g;
}
```

运行结果：

```
r=5
s=8
1.37781
```

例3-6投骰子的随机游戏

每个骰子有六面，点数分别为1、2、3、4、5、6。游戏者在程序开始时输入一个无符号整数，作为产生随机数的种子。

每轮投两次骰子，第一轮如果和数为7或11则为胜，游戏结束；和数为2、3或12则为负，游戏结束；和数为其它值则将此值作为自己的点数，继续第二轮、第三轮...直到某轮的和数等于点数则取胜，若在此前出现和数为7则为负。

由rolldice函数负责模拟投骰子、计算和数并输出和数。

```
#include <iostream>
#include <cstdlib>
using namespace std;
int rolldice(void);
void main()
{
  int gamestatus,sum,mypoint;
  unsigned seed;
  cout<<"Please enter an unsigned integer:";
  cin>>seed;        //输入随机数种子
  srand(seed);      //将种子传递给rand()
  sum=rolldice(); //第一轮投骰子、计算和数
```

```
  switch(sum)
  {
    case 7:  //如果和数为7或11则为胜,状态为1
    case 11: gamestatus=1; break;
    case 2:  //和数为2、3或12则为负,状态为2
    case 3:
    case 12: gamestatus=2; break;
    default:
//其他情况,游戏尚无结果,状态为0,记下点数,为下一轮做准备
            gamestatus=0;
            mypoint=sum;
            cout<<"point is "<<mypoint<<endl;
            break;
   }
```

```
  while(gamestatus==0) //只要状态仍为 0,就继续进行下一轮
  {
    sum=rolldice();
    if(sum==mypoint) //某轮的和数等于点数则取胜,状态置为1
      gamestatus=1
    else if (sum==7) //出现和数为7则为负,状态置为2
      gamestatus=2;
  }
//当状态不为0时上面的循环结束,以下程序段输出游戏结果
  if( gamestatus==1 )
    cout<<"player wins\n";
  else
    cout<<"player loses\n";
}
```

- rand

  函数原型：int rand(void);

  所需头文件：<cstdlib>

  功能和返回值：求出并返回一个伪随机数,任意大小的int型数据。

- srand

  函数原型：void srand(unsigned int seed);

  参数：seed产生随机数的种子，通常使用time(NULL)。

  所需头文件：<ctime>

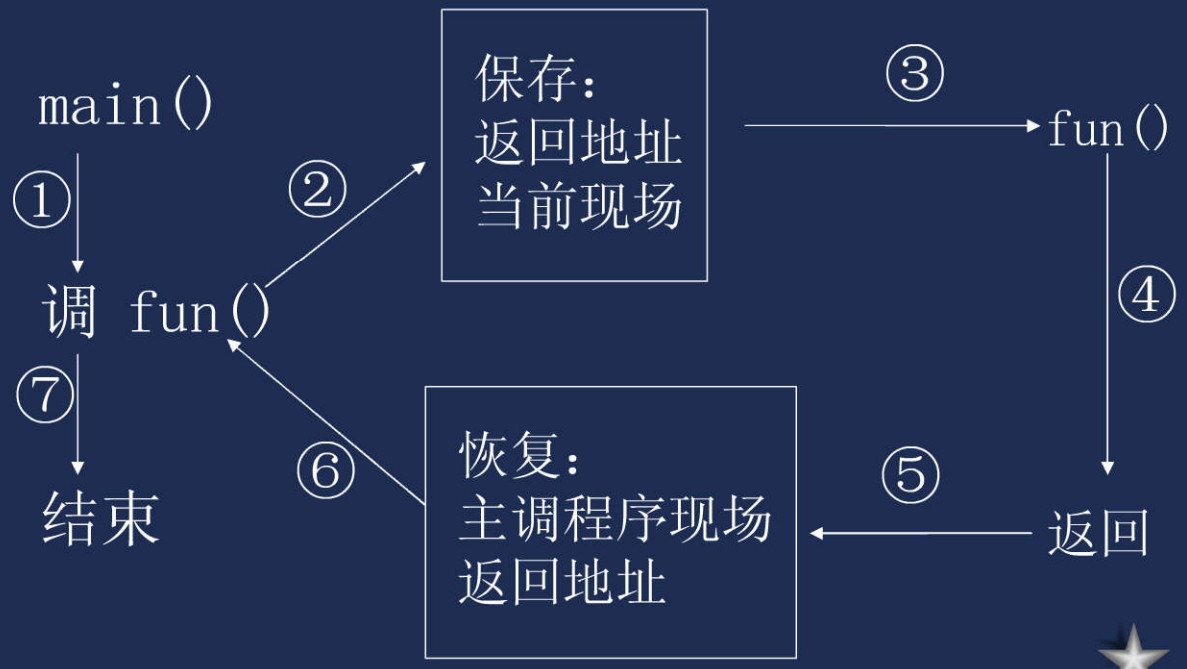
  功能：为使rand()产生一序列伪随机整数而设置起始点。使用1作为seed参数，可以重新初化rand()。

```
int rolldice(void)
{ //投骰子、计算和数、输出和数
  int die1,die2,worksum;
  die1=1+rand()%6;
  die2=1+rand()%6;
  worksum=die1+die2;
  cout<<"player rolled
  "<<die1<<'+'<<die2<<'='<<worksum<<endl;
  return worksum;
}
```

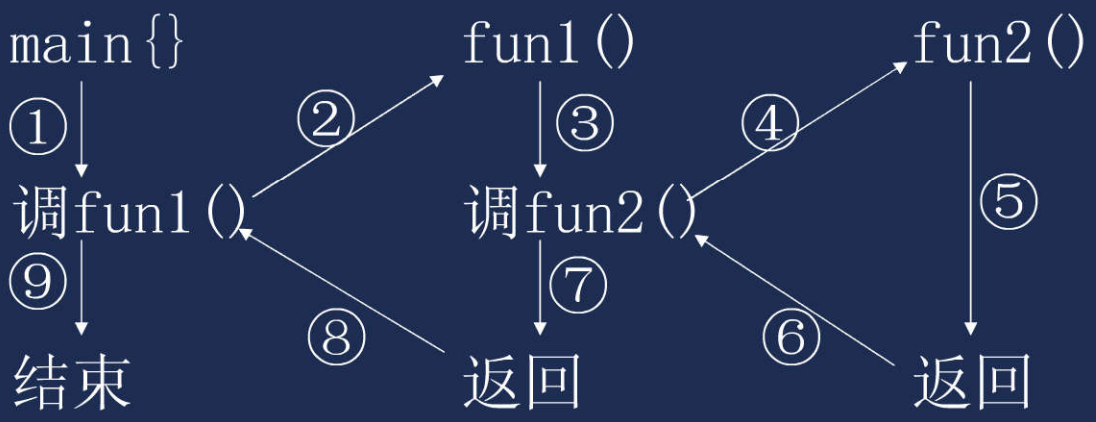
运行结果2:

```
Please enter an unsigned integer:23
player rolled 6+3=9
point is 9
player rolled 5+4=9
player wins
```

函数调用的执行过程



嵌套调用



例3-6 输入两个整数，求平方和。

```
#include <iostream>
using namespace std;
void main()
{
    int a,b;
    int fun1(int x,int y);
    cin>>a>>b;
    cout<<"a、b的平方和："
        <<fun1(a,b)<<endl;
}
```

```
int fun1(int x,int y)
{
    int fun2(int m);
    return (fun2(x)+fun2(y));
}

int fun2(int m)
{
    return (m*m);
}
```

运行结果:

```
3 4
a、b的平方和: 25
```

递归调用

- 函数直接或间接地调用自身，称为递归调用。
- 递归过程的两个阶段：
  - 递推：

4!=4×3!→3!=3×2!→2!=2×1!→1!=1×0!→0!=1

未知 ——————————→ 已知

  - 回归：

4!=4×3!=24←3!=3×2!=6←2!=2×1!=2←1!=1×0!=1←0!=1

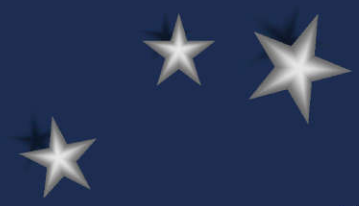
未知 ←—————————— 已知

例3-8 求n!

分析：计算$n!$的公式如下：

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

这是一个递归形式的公式，应该用递归函数实现。



源程序：

```
#include <iostream>
using namespace std;
long fac(int n)
{ long f;
  if (n<0)
    cout<<"n<0,data error!"<<endl;
  else if (n==0) f=1;
  else f=fac(n-1)*n;
  return(f);
}
```

```
void main()
{
  long fac(int n);
  int n;
  long y;
  cout<<"Enter a positive integer:";
  cin>>n;
  y=fac(n);
  cout<<n<<"!="<<y<<endl;
}
```

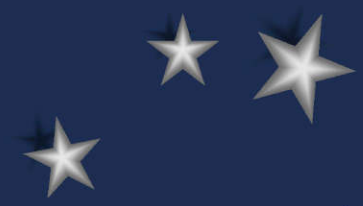
运行结果：

```
Enter a positive integer:8
8!=40320
```

例3-9

- 用递归法计算从n个人中选择k个人组成一个委员会的不同组合数。
- 分析：

由n个人里选k个人的组合数
=由n-1个人里选k个人的组合数
+由n-1个人里选k-1个人的组合数
当n==k或k==0时，组合数为1

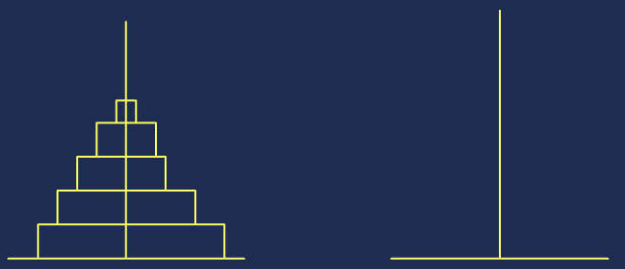


```
#include<iostream>
using namespace std;
void main()
{  int n,k;
   int comm(int n, int k);
   cin>>n>>k;
   cout<<comm(n,k)<<endl;
}
int comm(int n,int k)
{  if (k>n) return 0;
   else if( n==k||k==0 )
     return 1;
   else
     return  comm(n-1,k)+comm(n-1,k-1);
}
```

运行结果：
18 5
8568

例3-10汉诺塔问题

有三根针A、B、C。A针上有N个盘子，大的在下，小的在上，要求把这N个盘子从A针移到C针，在移动过程中可以借助B针，每次只允许移动一个盘，且在移动过程中在三根针上都保持大盘在下，小盘在上。



分析：

将n 个盘子从A针移到C针可以分解为下面三个步骤：

①将A 上n-1个盘子移到 B针上（借助C针）；

②把A针上剩下的一个盘子移到C针上；

③将n-1个盘子从B针移到C针上（借助A针）；

事实上，上面三个步骤包含两种操作：

①将多个盘子从一个针移到另一个针上，这是一个递归的过程。 hanoi函数实现。

②将1个盘子从一个针上移到另一针上。
用move函数实现。

```
#include <iostream>
using namespace std;
void move(char getone,char putone)
{ cout<<getone<<"-->"<<putone<<endl; }
void hanoi(int n,char one,char two,char three)
{ void move(char getone,char putone);
  if (n==1) move (one,three);
  else
  {  hanoi(n-1,one,three,two);
     move(one,three);
     hanoi(n-1,two,one,three);
  }
}
```

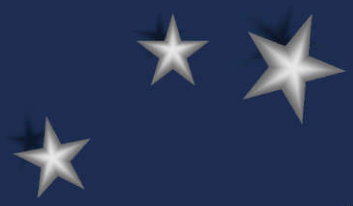
```
void main()
{
  void hanoi(int n,char one,char two,char three);
  int m;
  cout<<"Enter the number of diskes:";
  cin>>m;
  cout<<"the steps to moving "<<m
      <<" diskes:"<<endl;
  hanoi(m,'A','B','C');
}
```

运行结果：

```
Enter the number of diskes:3
the steps to moving 3 diskes:
A-->C
A-->B
C-->B
A-->C
B-->A
B-->C
A-->C
```

函数的参数传递机制
——传递参数值

- 在函数被调用时才分配形参的存储单元。
- 实参可以是常量、变量或表达式。
- 实参类型必须与形参相符。
- 传递时是传递参数值，即单向传递。



函数的参数传递机制
——参数值传递举例

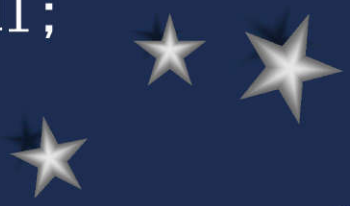
主调函数：　　D = power(A, 3)

被调函数：

```
double power(double X, int N)
```

例3-11 输入两个整数交换后输出

```
#include<iostream>
using namespace std;
void Swap(int a, int b);
void main()
{
  int x(5), y(10);
  cout<<"x="<<x<<"     y="<<y<<endl;
  Swap(x,y);
  cout<<"x="<<x<<"     y="<<y<<endl;
}
```



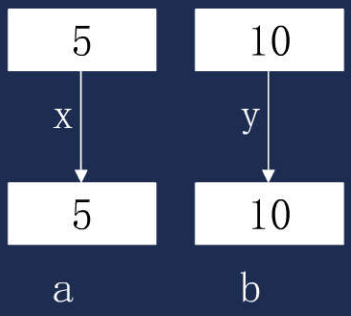
```
void Swap(int a, int b)
{
  int t;
  t=a;
  a=b;
  b=t;
}
```

运行结果:

x=5 y=10

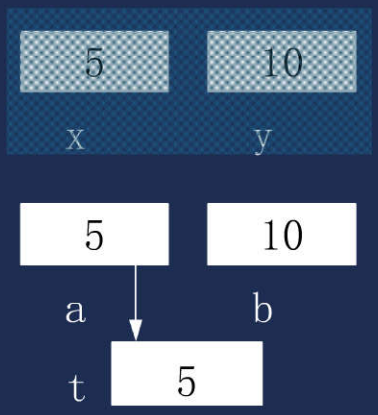
x=5 y=10

执行主函数中的函数调用
Swap(x, y);

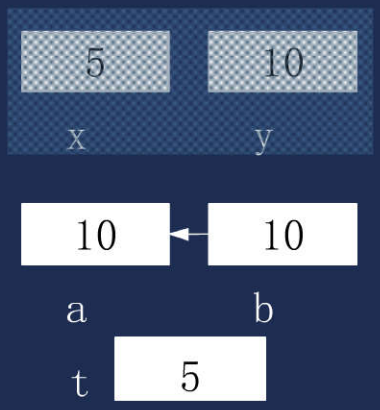


在Swap子函数中

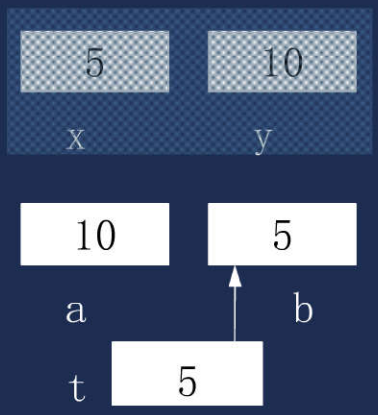
t=a;



a=b;



b=t;



返回主函数以后

函数的参数传递
——用引用做形参

- 引用(&)是标识符的别名,例如:

```
int i,j;
int &ri=i;
    //建立一个int型的引用ri,并将其
    //初始化为变量i的一个别名
j=10;
ri=j;//相当于 i=j;
```

- 声明一个引用时，必须同时对它进行初始化，使它指向一个已存在的对象。
- 一旦一个引用被初始化后，就不能改为指向其它对象。
- 引用可以作为形参

```
void swap(int& a, int& b) {...}
```

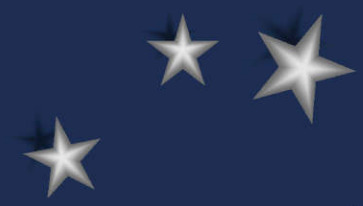
例3-12 输入两个整数交换后输出

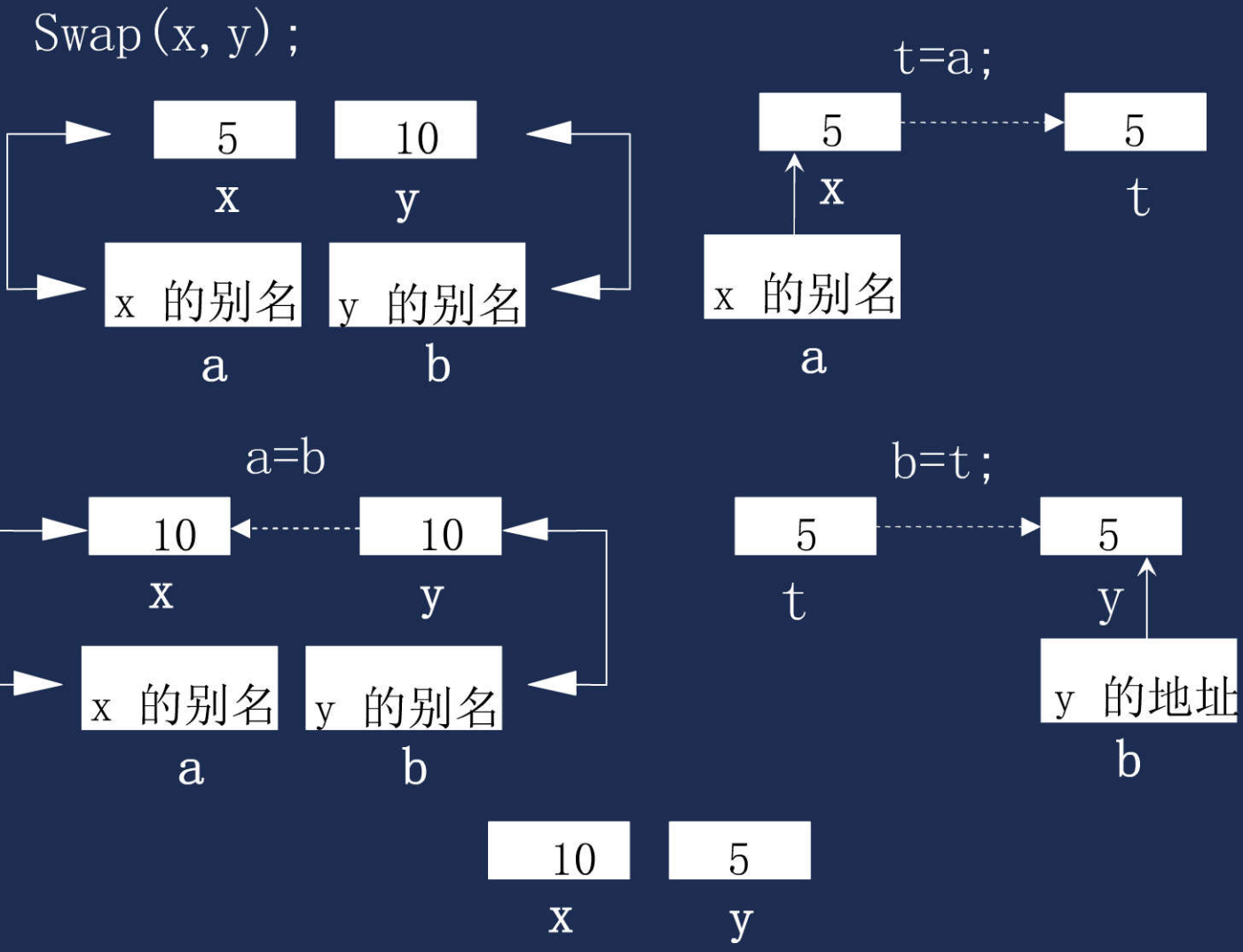
```
#include<iostream>
using namespace std;
void Swap(int& a, int& b);
void main()
{  int x(5), y(10);
   cout<<"x="<<x<<"     y="<<y<<endl;
   Swap(x,y);
   cout<<"x="<<x<<"     y="<<y<<endl;
}
void Swap(int& a, int& b)
{  int t;
   t=a;
   a=b;
   b=t;
}
```

运行结果:

x=5 y=10

x=10 y=5




Swap(x, y);
5
10
x
y
x 的别名
y 的别名
a
b
t=a;
5
5
x
t
x 的别名
a
a=b
10
10
x
y
x 的别名
y 的别名
a
b
b=t;
5
5
t
y
y 的地址
b
10
5
x
y

内联函数定义与使用

内联函数

对于一些功能简单、规模较小又使用频繁的函数，可以设计为内联函数。内联函数不是在调用时发生控制转移，而是在编译时将函数体嵌入在每一个调用处，这样就节省了参数传递、控制转移等开销。语法形式如下：

inline 类型说明符 函数名（形参表）
{函数体语句；}

内联函数定义与使用

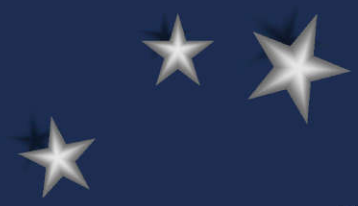
内联函数

- 定义时使用关键字 inline。
- 注意：
  - 内联函数体内不能有循环语句和switch语句。
  - 内联函数的定义(或声明)必须出现在内联函数第一次被调用之前。
  - 对内联函数不能进行异常接口声明。

例3-14 内联函数应用举例

内联函数

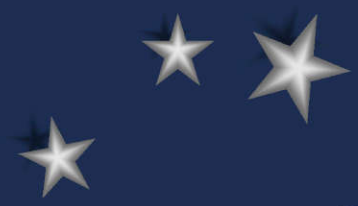
```
#include<iostream>
using namespace std;
inline double CalArea(double radius)
{ return 3.14*radius*radius;
}
void main()
{
  double r(3.0);
  double area;
  area=CalArea(r);
  cout<<area<<endl;
 }
```



默认形参值的作用

- 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。
- 例如：

```
int add(int x=5,int y=6)
{ return x+y; }
void main()
{ add(10,20);//10+20
  add(10);    //10+6
  add();      //5+6
}
```



默认形参值的说明次序

- 默认形参值必须从右向左顺序声明，并且在默认形参值的右面不能有非默认形参值的参数。因为调用时实参取代形参是从左向右的顺序。
- 例：

```
int add(int x,int y=5,int z=6);//正确
int add(int x=1,int y=5,int z);//错误
int add(int x=1,int y,int z=6);//错误
```

默认形参值与函数的调用位置

- 调用出现在函数体实现之前时，默认形参值必须在函数原形中给出；而当调用出现在函数体实现之后时，默认形参值可以在函数实现时（或函数原形中）给出。

例：

```
#include<iostream>
using namespace std;
int add(int x=5,int y=6);
void main()
{ cout<<add()<<endl;
   //调用在实现前
}
int add(int x,int y)
{ return x+y; }
```

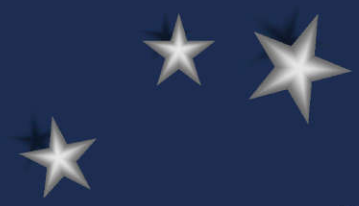
```
#include<iostream>
using namespace std;
int add(int x=5,int y=6)
{ return  x+y; }
void main()
{ cout<<add()<<endl;
   //调用在实现后
 }
```

默认形参值的作用域

- 在相同的作用域内，默认形参值的说明应保持惟一，但如果在不同的作用域内，允许说明不同的默认形参。
- 例：

```
int add(int x=1,int y=2);
void main()
{ int add(int x=3,int y=4);
  add();//使用局部默认形参值（实现3+4）
}
void fun()
{   ...
  add();//使用全局默认形参值（实现1+2）
}
```

P81 例3-15 带默认形参值的函数举例。



重载函数的声明

函数重载

- C++允许功能相近的函数在相同的作用域内以相同函数名声明，从而形成重载。方便使用，便于记忆。
- 例：

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

} 形参类型不同

```
int add(int x,int y);
int add(int x,int y,int z);
```

} 形参个数不同

注意事项

函数重载

- 重载函数的形参必须不同:个数不同或类型不同。
- 编译程序将根据实参和形参的类型及个数的最佳匹配来选择调用哪一个函数。

```
int add(int x,int y);
int add(int a,int b);
```

编译器不以形参名来区分

```
int add(int x,int y);
void add(int x,int y);
```

编译器不以返回值来区分

- 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。

```
int add(int x,int y)
{  return x+y;  }
```

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

例3-16重载函数应用举例

编写三个名为add的重载函数，分别实现两整数相加、两实数相加和两个复数相加的功能。

```
#include<iostream>
using namespace std;
struct complex
{
  double real;
  double imaginary;
};
```

```
void main()
{
  int m, n;
  double x, y;
  complex c1, c2, c3;
  int add(int m, int n);
  double add(double x, double y);
  complex add(complex c1, complex c2);
  cout<<"Enter two integer: ";
  cin>>m>>n;
  cout<<"integer
  <<m<<'+'<<n<<"="<<add(m,n)<<endl;
```

```
    cout<<"Enter two real number: ";
    cin>>x>>y;
    cout<<"real number "<<x<<'+'<<y<<"= "<<add(x,y)<<endl;
    cout<<"Enter the first complex number: ";
    cin>>c1.real>>c1.imaginary;
    cout<<"Enter the second complex number: ";
    cin>>c2.real>>c2.imaginary;
    c3=add(c1,c2);
    cout<<"complex number ("<<c1.real<<','
        <<c1.imaginary <<")+("<<c2.real<<','
        <<c2.imaginary<<")=("<<c3.real<<','
        <<c3.imaginary<<")\n";
}
```

```
int add(int m, int n)
{  return m+n; }

double add(double x, double y)
{  return x+y; }

complex add(complex c1, complex c2)
{
  complex c;
  c.real=c1.real+c2.real;
  c.imaginary=c1.imaginary+c2.imaginary;
  return c;
}
```

运行结果：

```
Enter two integer: 3 5
integer 3+5=8
Enter two real number: 2.3 5.8
real number 2.3+5.8= 8.1
Enter the first complex number: 12.3 45.6
Enter the second complex number: 56.7 67.8
complex number (12.3,45.6)+(56.7,67.8)= (69,113.4)
```

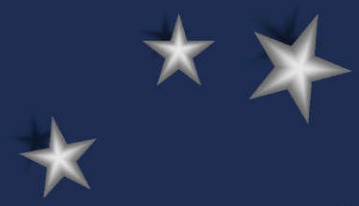
C++系统函数

- C++的系统库中提供了几百个函数可供程序员使用。

  例如：求平方根函数（sqrt）、求绝对值函数（abs）等。

- 使用系统函数时要包含相应的头文件。

  例如：math.h 或 cmath



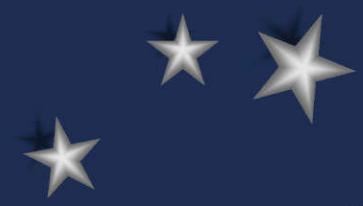
例3-17系统函数应用举例

- 题目：

  从键盘输入一个角度值，求出该角度的正弦值、余弦值和正切值。

- 分析：

  系统函数中提供了求正弦值、余弦值和正切值的函数：sin()、cos()、tan()，函数的说明在头文件math.h和cmath中。



```
#include<iostream>
#include<math>
using namespace std;
const double pi(3.14159265);
void main()
{   double a,b;
    cin>>a;
    b=a*pi/180;
    cout<<"sin("<<a<<")="<<sin(b)<<endl;
    cout<<"cos("<<a<<")="<<cos(b)<<endl;
    cout<<"tan("<<a<<")="<<tan(b)<<endl;
}
```

运行结果：

30
sin(30)=0.5
cos(30)=0.866025
tan(30)=0.57735

查找系统函数的使用说明

使用C++系统函数

- 查编译系统的库函数手册
- 查联机帮助——VC++6.0联机帮助的使用方法：

help/Contents
->(“活动子集”栏)Visual C++ Documentation
->Visual C++ Documentation
->Using Visual C++
-> Visual C++ Programmer’s Guide
-> Run-Time Library Reference
->Run Time Routines by Category
-> Run Time Routines by Category

小结与复习建议

- 主要内容
  - 函数的声明和调用、函数间的参数传递、内联函数、带默认形参值的函数、函数重载、C++系统函数
- 达到的目标
  - 学会将一段功能相对独立的程序写成一个函数，为下一章学习类和对象打好必要的基础。
- 实验任务
  - 实验三
- 作业
  - 2-13 2-14 2-17 2-28 2-36 2-37