第 3 章函数

本章主要内容

函数的定义与调用

- 函数定义的语法形式
- 函数的调用
  - 调用前先声明函数
  - 调用形式
  - 嵌套调用
  - 递归调用
- 函数的参数传递
  - 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
  - 值传递是传递参数值，即单向传递
  - 引用传递可以实现双向传递
  - 常引用作参数可以保障实参数据的安全

内联函数

- 声明时使用关键字 inline
- 编译时在调用出用函数体进行替换，节省了参数传递、控制转移等开销

constexpr 函数

- constexpr 修饰的函数在其所有参数都是 constexpr 时，一定返回 constexpr
- 函数体中必须有且仅有一条 return 语句

带默认参数值的函数

- 可以预先设置默认的参数值，调用时如给出实参，则采用实参值，否则采用预先设置的

默认参数值

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

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
- 使用系统函数时要包含相应的头文件

函数定义

函数

函数：定义好的、可重用的功能模块

定义函数：将一个模块的算法用 C++描述出来

函数名：功能模块的名字

函数的参数：计算所需要的数据和条件

函数的返回值：需要返回的计算结果

函数定义的语法形式

函数名

```
类型标识符  函数名（形式参数表）
{
  语句序列
}
```

形式参数表

```
类型标识符  函数名（形式参数表）
{
  语句序列
}
```

〈type1〉 name1, 〈type2〉 name2, ..., 〈typen〉 namen
是被初始化的内部变量，寿命和可见性仅限于函数内部

语句序列

```
类型标识符  函数名（形式参数表）
{
  语句序列
}
```

程序要执行的操作序列
最后一句是 return 语句

类型标识符

```
类型标识符  函数名（形式参数表）
{
  语句序列
}
```

- 表示返回值类型，由 return 语句给出返回值
- 若无返回值，写void，不必写return语句。

函数的调用

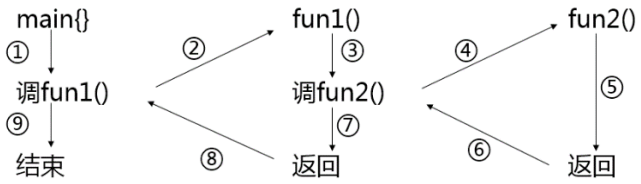
调用函数需要先声明函数原型

- 若函数定义在调用点之前，可以不另外声明；
- 若函数定义在调用点之后，必须要在调用函数前声明函数原型：
- 函数原型：类型标识符 被调用函数名（含类型说明的形参表）

函数调用形式

- 函数名（实参列表）

嵌套调用



嵌套调用：在一个函数的函数体中，调用另一函数。

例 3-1 编写一个求 x 的 n 次方的函数

```
#include <iostream>
using namespace std;

//计算 x 的 n 次方
double power(double x, int n) {
        double val = 1.0;
        while (n--) val *= x;
        return val;
}

int main() {
        cout << "5 to the power 2 is "
    << power(5, 2) << endl;
        return 0;
}
```

例 3-2

例 3-2 数制转换

- 输入一个 8 位二进制数，将其转换为十进制数输出。
- 例如：从键盘输入 1101
- $1101_2=1\times2^3+1\times2^2+0\times2^1+1\times2^0=13_{10}$
- 所以，程序应输出 13

源代码：

```
#include <iostream>
using namespace std;

double power (double x, int n); //计算 x 的 n 次方

int main() {
        int  value = 0;
        cout << "Enter an 8 bit binary number  ";
        for (int i = 7; i >= 0; i--) {
          char ch;
          cin >> ch;
          if (ch == '1')
            value += static_cast<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 3-3

编写程序求π的值

- π的计算公式如下：

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

- 其中 arctan 用如下形式的级数计算：

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots$$

- 直到级数某项绝对值不大于 10^{-15} 为止；π和 x 均为 double 型。

arctan 函数

```
#include <iostream>
using namespace std;

double arctan(double x) {
        double sqr = x * x;
        double e = x;
        double r = 0;
        int i = 1;
        while (e / i > 1e-15) {
                double f = e / i;
                r = (i % 4 == 1) ? r + f : r - f;
                e = e * sqr;
                i += 2;
        }
        return r;
}
```

主程序

```
int main() {
        double a = 16.0 * arctan(1/5.0);
        double 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;
        return 0;
}
```

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：

用除以 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;
//判断 n 是否为回文数
bool symm(unsigned n) {
  unsigned i = n;
        unsigned m = 0;
        while (i > 0) {
          m = m * 10 + i % 10;
          i /= 10;
  }
  return m == n;
}
int main() {
        for(unsigned m = 11; m < 1000; m++)
          if (symm(m) && symm(m * m) && symm(m * m * m)) {
```

```
            cout << "m = " << m;
            cout << "  m * m = " << m * m;
            cout << "  m * m * m = "
                 << m * m * m << endl;
        }
    return 0;
}
```

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

例 3-5 计算分段函数，并输出结果

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

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^{-10}：

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

```
#include <iostream>
#include <cmath>  //对标准库中数学函数的说明
using namespace std;

const double TINY_VALUE = 1e-10;
```

计算精度为10^{-10}

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

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

```
int main() {
  double k, r, s;
  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;
  return 0;
}
```

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

运行结果：

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

例 3-6

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。

rand 函数

- 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

srand 函数

- void srand(unsigned int seed);
- 参数：seed 产生随机数的种子
- 所需头文件：<cstdlib>
- 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()。

源代码：

```
#include <iostream>
#include <cstdlib>
using namespace std;

enum GameStatus { WIN, LOSE, PLAYING };
 int main() {
   int sum, myPoint;
   GameStatus status;
   unsigned seed;
   int rollDice();
   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 WIN
```

```
        case 11:
            status = WIN;
            break;
        case 2:   //和数为 2、3 或 12 则为负,状态为 LOSE
        case 3:
        case 12:
            status = LOSE;
            break;
        default:  //其它情况，尚无结果，状态为 PLAYING，记下点数
          status = PLAYING;
          myPoint = sum;
          cout << "point is " << myPoint << endl;
          break;
        }
   while (status == PLAYING) { //只要状态为 PLAYING，继续
      sum = rollDice();
      if (sum == myPoint)    //某轮的和数等于点数则取胜
          status = WIN;
      else if (sum == 7)    //出现和数为 7 则为负
          status = LOSE;
  }

  //当状态不为 PLAYING 时循环结束，输出游戏结果
  if (status == WIN)
    cout << "player wins" << endl;
  else
    cout << "player loses" << endl;
  return 0;
}

//投骰子、计算和数、输出和数
int rollDice() {
        int die1 = 1 + rand() % 6;
```

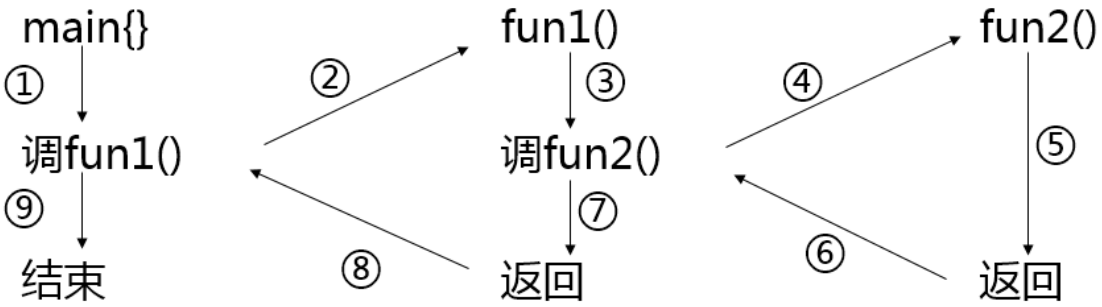
```
        int die2 = 1 + rand() % 6;
        int sum = die1 + die2;
        cout << "player rolled " << die1 << " + " << die2 << " = " << sum << endl;
        return sum;
}
```

运行结果：

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

函数的嵌套调用

- 嵌套调用



例 3-7 输入两个整数，求平方和

```
#include <iostream>
using namespace std;
int fun2(int m) {
        return m * m;
}
int fun1(int x,int y) {
        return fun2(x) + fun2(y);
```

```
}
int main() {
        int a, b;
        cout<<"Please enter two integers (a and b): ";
        cin >> a >> b;
        cout << "The sum of square of a and  b: "
         << fun1(a, b) << endl;
        return 0;
}
```

函数的递归调用

定义

- 函数直接或间接地调用自身，称为递归调用。

例：计算 n!

- 公式 1：n!=n×(n-1)×(n-2)×(n-3)×...×2×1.
- 公式 2：

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

例如，计算 4!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。

```
#include <iostream>
using namespace std;
unsigned fac(int n){
   unsigned f;
   if (n == 0)
      f = 1;
  else
       f = fac(n - 1) * n;
  return f;
}
int main() {
   unsigned n;
   cout << "Enter a positive integer:";
   cin >> n;
   unsigned y = fac(n);
   cout << n << "! = " << y << endl;
   return 0;
}
```

运行结果：

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

例 3-9

用递归法计算从 n 个人中选选 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。

源代码：

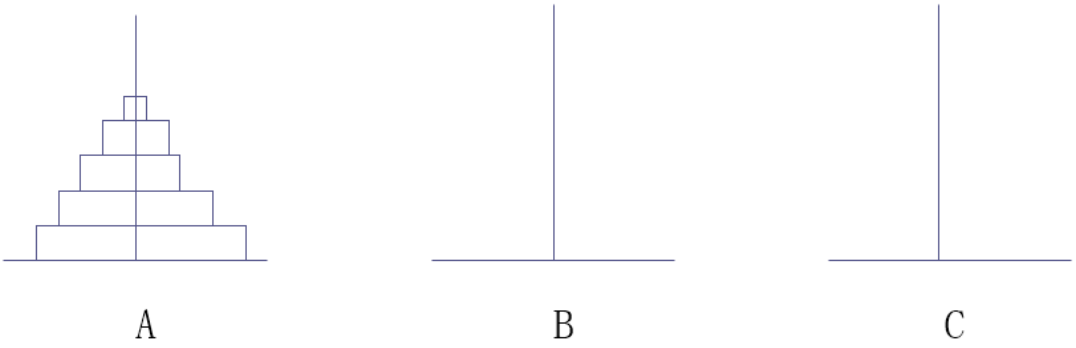
```
#include <iostream>
using namespace std;

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);
}

int main() {
        int n, k;
        cout << "Please enter two integers n and k: ";
        cin >> n >> k;
        cout << "C(n, k) = " << comm(n, k) << endl;
        return 0;
}
```

运行结果：

```
18 5
8568
```

例 3-10

- 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。



- 将 n 个盘子从 A 针移到 C 针可以分解为三个步骤：
  - 将 A 上 n-1 个盘子移到 B 针上（借助 C 针）；
  - 把 A 针上剩下的一个盘子移到 C 针上；
  - 将 n-1 个盘子从 B 针移到 C 针上（借助 A 针）。

源代码：

```
#include <iostream>
using namespace std;
//将 src 针的最上面一个盘子移动到 dest 针上
void move(char src, char dest) {
        cout << src << " --> " << dest << endl;
}
//将 n 个盘子从 src 针移动到 dest 针，以 medium 针作为中转
void hanoi(int n, char src, char medium, char dest)
{
  if (n == 1)
        move(src, dest);
  else {
        hanoi(n - 1, src, dest, medium);
        move(src, dest);
        hanoi(n - 1, medium, src, dest);
  }
}
int main() {
```

```
    int m;
    cout << "Enter the number of diskes: ";
    cin >> m;
    cout << "the steps to moving " << m << " diskes:" << endl;
    hanoi(m,'A','B','C');
    return 0;
}
```

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

- 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
- 值传递是传递参数值，即单向传递
- 引用传递可以实现双向传递
- 常引用作参数可以保障实参数据的安全

引用类型

引用的概念

- 引用(&)是标识符的别名；
- 定义一个引用时，必须同时对它进行初始化，使它指向一个已存在的对象。
- 例如：

```
int i, j;
int &ri = i; //定义 int 引用 ri，并初始化为变量 i 的引用
j = 10;
ri = j;  //相当于 i = j;
```

- 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

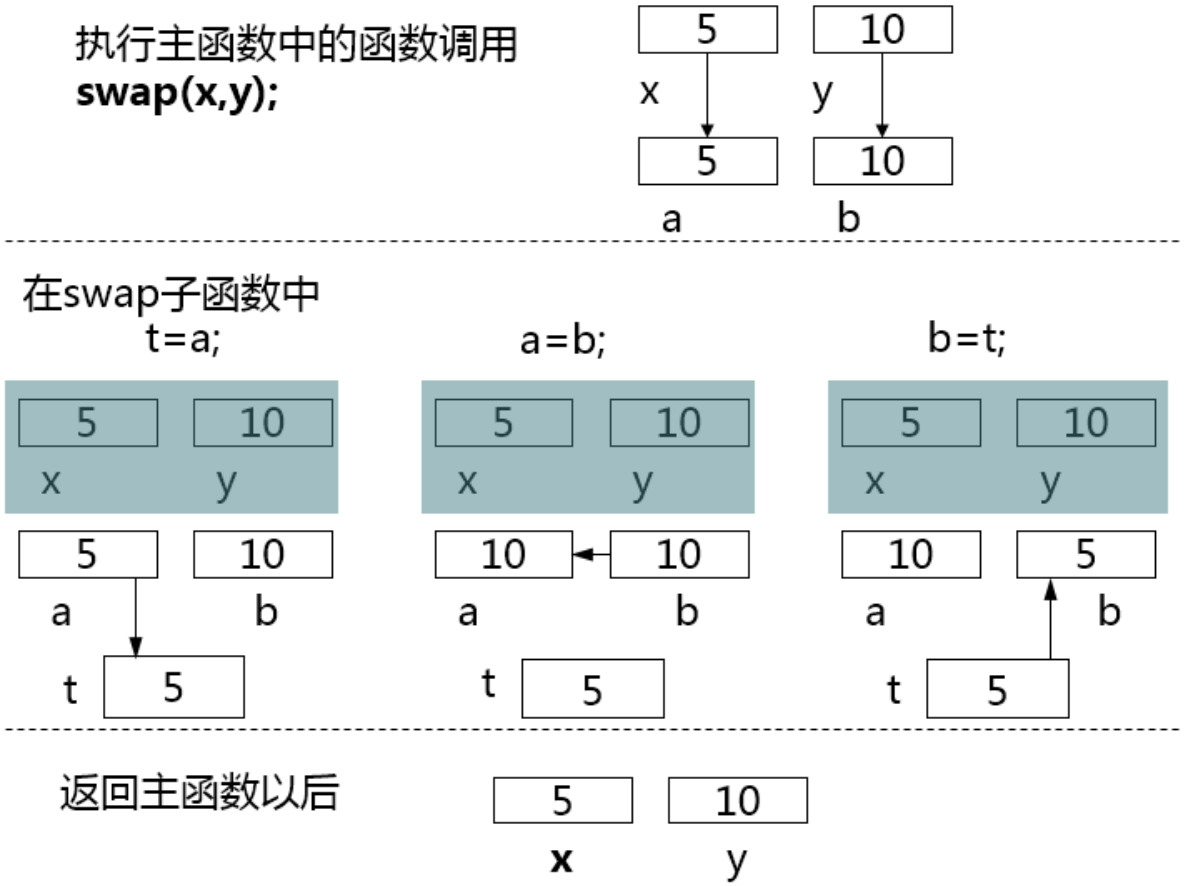
例 3-11 输入两个整数并交换（值传递）

```
#include<iostream>
using namespace std;
void swap(int a, int b) {
    int t = a;
    a = b;
    b = t;
}
int main() {
    int x = 5, y = 10;
    cout<<"x = "<<x<<"  y = "<<y<<endl;
    swap(x, y);
    cout<<"x = "<<x<<"  y = "<<y<<endl;
    return 0;
}
```

运行结果：

```
x = 5     y = 10
x = 5     y = 10
```

例 3-11 参数传递示意图

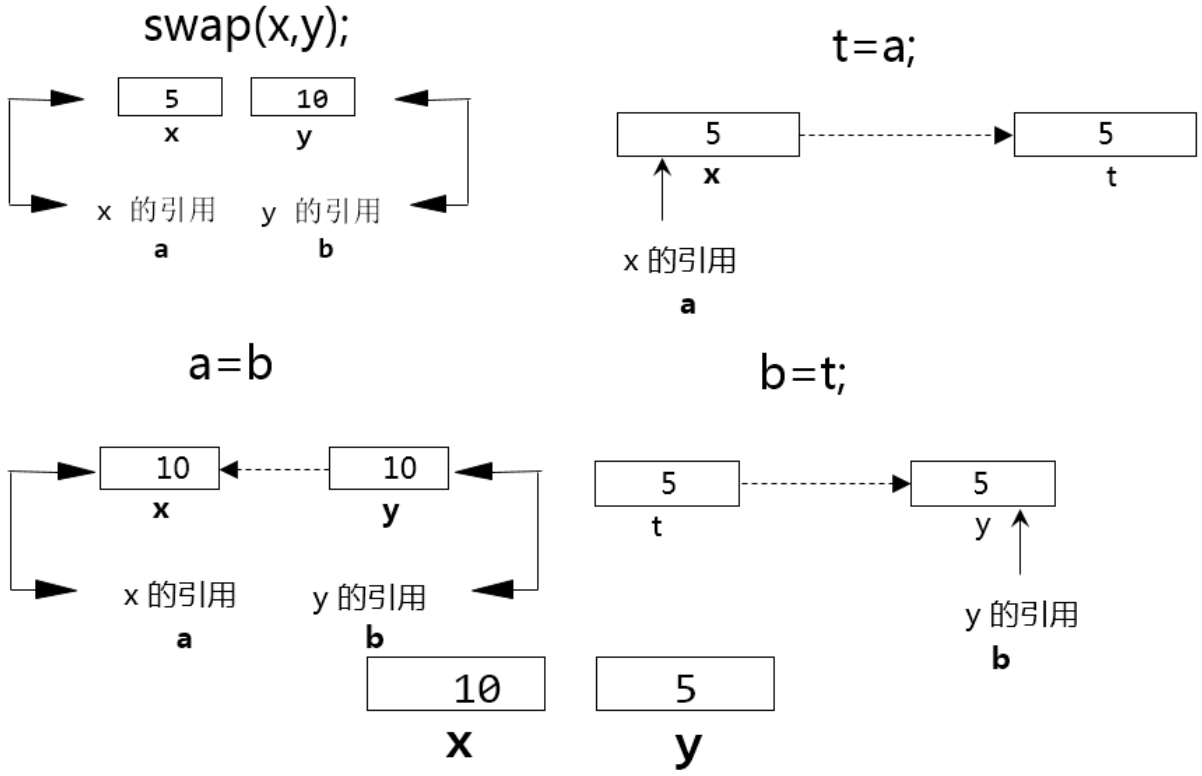


例 3-12 输入两个整数并交换（引用传递）

```
#include<iostream>
using namespace std;
void swap(int& a, int& b) {
        int t = a;
        a = b;
        b = t;
}
int main() {
        int x = 5, y = 10;
        cout<<"x = "<<x<<"  y = "<<y<<endl;
        swap(x, y);
        cout<<"x = "<<x<<"  y = "<<y<< endl;
        return 0;
```

```
}
```

例 3-12 参数传递示意图



含有可变参数的函数

含有可变参数的函数

- C++标准中提供了两种主要的方法
  - 如果所有的实参类型相同，可以传递一个名为 initializer_list 的标准库类型；
  - 如果实参的类型不同，我们可以编写可变参数的模板（第 9 章）。
- initializer_list
  - initializer_list 是一种标准库类型，用于表示某种特定类型的值的数组，该类型定义在同名的头文件中

initializer_list 提供的操作

initializer_list提供的操作	
initializer_list<T> lst;	默认初始化；T类型元素的空列表
initializer_list<T> lst{a, b, c...}	lst的元素数量和初始值一样多；lst的元素是对应初始值的副本；列表中的元素是const
lst2(lst) lst2 = lst	拷贝或者赋值一个initializer_list对象但不拷贝列表中的元素；拷贝后原始列表和副本共享元素
lst.size()	列表中的元素数量
lst.begin()	返回指向lst首元素的指针
lst.end()	返回指向lst尾元素下一位置的指针

initializer_list 的使用方法

- initializer_list 是一个类模板（第 9 章详细介绍模板）
- 使用模板时，我们需要在模板名字后面跟一对尖括号，括号内给出类型参数。例如：
  - initializer_list<string> ls; // initializer_list 的元素类型是 string
  - initializer_list<int> li; // initializer_list 的元素类型是 int
- initializer_list 比较特殊的一点是，其对象中的元素永远是常量值，我们无法改变 initializer_list 对象中元素的值。
- 含有 initializer_list 形参的函数也可以同时拥有其他形参

initializer_list 使用举例

- 在编写代码输出程序产生的错误信息时，最好统一用一个函数实现该功能，使得对所有错误的处理能够整齐划一。然而错误信息的种类不同，调用错误信息输出函数时传递的参数也会各不相同。
- 使用 initializer_list 编写一个错误信息输出函数，使其可以作用于可变数量的形参。

内联函数

内联函数

- 声明时使用关键字 inline。
- 编译时在调用处用函数体进行替换，节省了参数传递、控制转移等开销。
- 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

```
#include <iostream>
using namespace std;

const double PI = 3.14159265358979;
inline double calArea(double radius) {
        return PI * radius * radius;
}

int main() {
        double r = 3.0;
        double area = calArea(r);
        cout << area << endl;
        return 0;
}
```

constexpr 函数

constexpr 函数语法规定

- constexpr 修饰的函数在其所有参数都是 constexpr 时，一定返回 constexpr；
- 函数体中必须有且仅有一条 return 语句。

constexpr 函数举例

- constexpr int get_size() { return 20; }
- constexpr int foo = get_size(); //正确：foo 是一个常量表达式

带默认参数值的函数

默认参数值

- 可以预先设置默认的参数值，调用时如给出实参，则采用实参值，否则采用预先设置

的默认参数值。

- 例：

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

默认参数值的说明次序

- 有默认参数的形参必须列在形参列表的最右，即默认参数值的右面不能有无默认值的参数；
- 调用时实参与形参的结合次序是从左向右。
- 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
```

默认参数值与函数的调用位置

- 如果一个函数有原型声明，且原型声明在定义之前，则默认参数值应在函数原型声明中给出；如果只有函数的定义，或函数定义在前，则默认参数值可以函数定义中给出。
- 例：

```
int add(int x = 5,int y = 6);
//原型声明在前
int main() {
  add();
}
int add(int x,int y) {
//此处不能再指定默认值
  return x + y;
}
```

```
int add(int x = 5,int y = 6) {
//只有定义，没有原型声明
  return  x + y;
}
int main() {
  add();
}
```

例 3-15

例 3-15 计算长方体的体积

- 函数 getVolume 计算体积
  - 有三个形参：length（长）、width（宽）、height（高），其中 width 和 height 带有默认值 2 和 3。
- 主函数中以不同形式调用 getVolume 函数。

源代码

```
//3_15.cpp
#include <iostream>
#include <iomanip>
using namespace std;

int getVolume(int length, int width = 2, int height = 3);

int main() {
        const int X = 10, Y = 12, Z = 15;
        cout << "Some box data is " ;
        cout << getVolume(X, Y, Z) << endl;
        cout << "Some box data is " ;
        cout << getVolume(X, Y) << endl;
        cout << "Some box data is " ;
        cout << getVolume(X) << endl;
        return 0;
}

int getVolume(int length, int width, int height) {
        cout << setw(5) << length << setw(5) << width << setw(5)
          << height << '\t';
        return length * width * height;
}
```

函数重载

函数重载的概念

- 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
  - 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。这样不好：

```
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

- 编写两个名为 sumOfSquare 的重载函数，分别求两整数的平方和及两实数的平方和。

```
#include <iostream>
using namespace std;
int sumOfSquare(int a, int b) {
    return a * a + b * b;
}
double sumOfSquare(double a, double b) {
    return a * a + b * b;
}
int main() {
    int m, n;
```

```
    cout << "Enter two integer: ";
    cin >> m >> n;
    cout<<"Their sum of square: "<<sumOfSquare(m, n)<<endl;
    double x, y;
    cout << "Enter two real number: ";
    cin >> x >> y;
    cout<<"Their sum of square: "<<sumOfSquare(x, y)<<endl;
    return 0;
}
```

- 运行结果：

```
Enter two integer: 3 5
Their sum of square: 34
Enter two real number: 2.3 5.8
Their sum of square: 38.93
```

C++系统函数

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（sprt）
  - 求绝对值函数（abs）
- 使用系统函数时要包含相应的头文件，例如：

```
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 cmath 中。
- 源代码

```
#include <iostream>
#include <cmath>
using namespace std;
```

```
const double PI = 3.14159265358979;

int main() {
    double angle;
    cout << "Please enter an angle: ";
    cin >> angle;        //输入角度值
    double radian = angle * PI / 180; //转为弧度
    cout << "sin(" << angle << ") = " << sin(radian) <<endl;
    cout << "cos(" << angle << ") = " << cos(radian) <<endl;
    cout << "tan(" << angle << ") = " << tan(radian) <<endl;
    return 0;
}
```

- 运行结果

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

小结

- 主要内容
  - 函数的声明和调用、函数间的参数传递、内联函数、带默认参数值的函数、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。