第 3 章函数导学

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
}
```

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

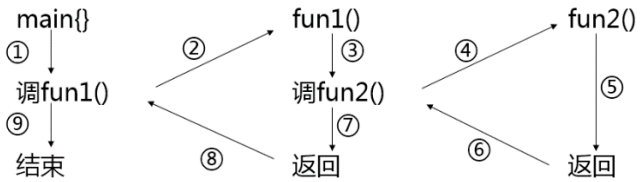
调用函数需要先声明函数原型

- 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

例 3-2 程序

```
#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 2-2 输入一个年份，判断是否闰年

计算如下公式，并输出结果：

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

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
```

```
#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;

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;
}

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;
}
```

函数调用例 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()。

```
#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;
```

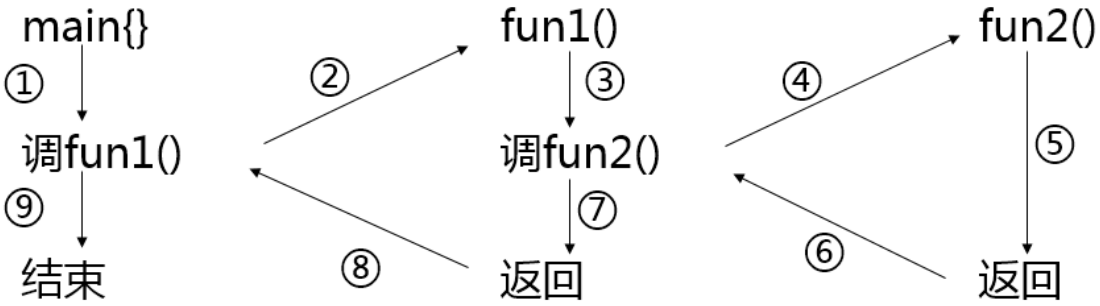
```
        cout << "player rolled " << die1 << " + " << die2 << " = " << sum << endl;
        return sum;
}
```

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。

```
#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;
}
```

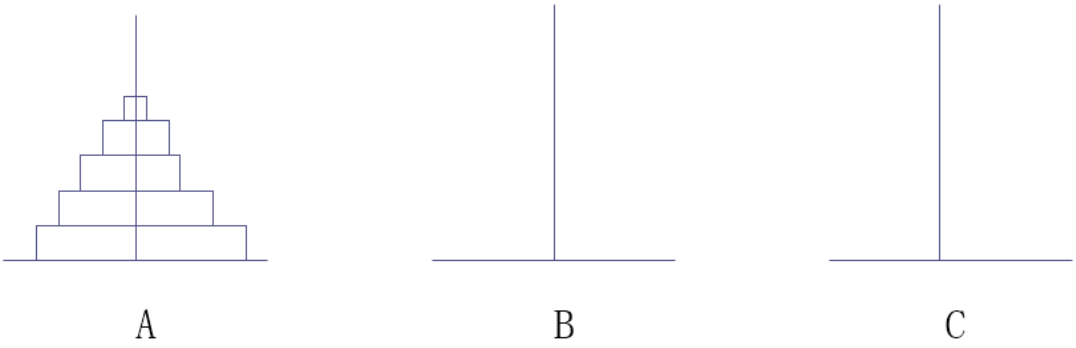
运行结果：

```
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）。

```
#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
```