C++语言程序设计

第三章 函数

中国科大　　黄章进

本章主要内容

- 函数的定义和调用
- 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
- 深度探索

函数的声明与使用

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

```
类型标识符   函数名（形式参数表）
{
    语句序列
}
```

是被初始化的内部变量，寿命和可见性仅限于函数内部

若无返回值，写void

函数的定义

- 形式参数表

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

- 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：
  - 若函数定义在调用点之前，则无需另外声明；
  - 若函数定义在调用点之后，则需要在调用函数前按如下形式声明函数原型：

    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
  - 函数可以嵌套调用，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;

//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3-1编写一个求x的n次方的函数

运行结果：

```
5 to the power 2 is 25
```

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。

例如：

$1101_2=1(2^3)+1(2^2)+0(2^1)+1(2^0)=13_{10}$

所以，如果输入1101，则应输出13

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

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

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;
}
```

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;

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;
}
```

```
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;
}
```

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。

```cpp
#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

计算如下公式，并输出结果：

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

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^{-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>  //对C++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;
}
```

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。

- 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

- srand

  函数原型：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;

//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;
}
```

```
enum GameStatus { WIN, LOSE, PLAYING };

int main() {
  int sum, myPoint;
  GameStatus status;

  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
```

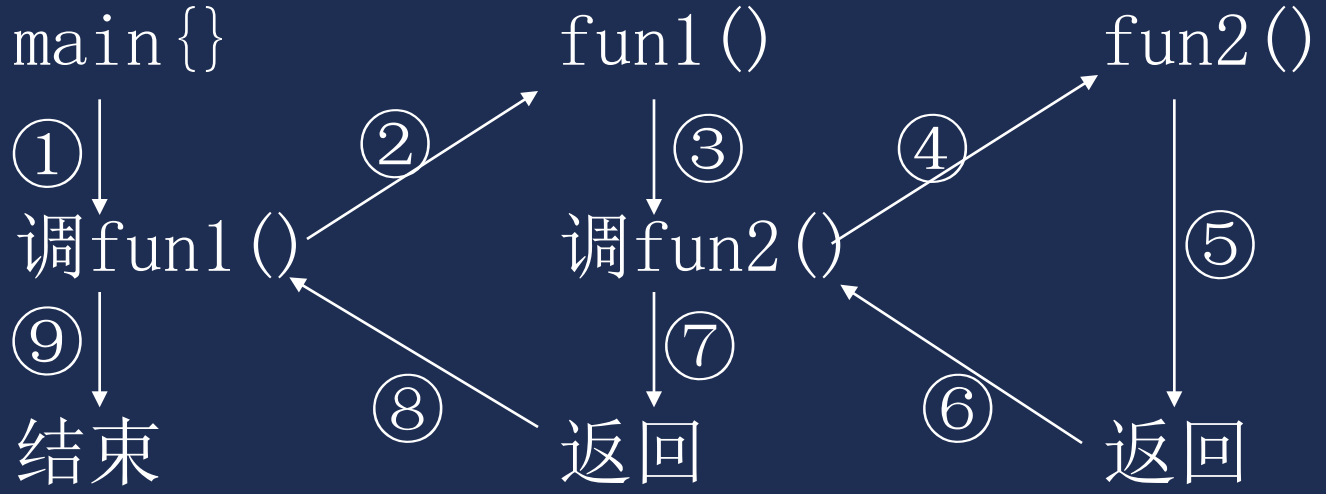
```
switch (sum) {
case 7:    //如果和数为7或11则为胜,状态为WIN
case 11:
  status = WIN;
  break;
case 2:    //和数为2、3或12则为负,状态为LOSE
case 3:
case 12:
  status = LOSE;
  break;
default:    //其它情况,游戏尚无结果,状态为PLAYING,记下点数,为下一轮做准备
  status = PLAYING;
  myPoint = sum;
  cout << "point is " << myPoint << endl;
  break;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

```
    while (status == PLAYING) { //只要状态仍为PLAYING,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
        status = WIN;
      else if (sum == 7)     //出现和数为7则为负
        status = LOSE;
    }

    //当状态不为PLAYING时上面的循环结束,以下程序段输出游戏结果
    if (status == WIN)
      cout << "player wins" << endl;
    else
      cout << "player loses" << endl;

    return 0;
}
```

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
```

嵌套调用



例3-6 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;
}
```

运行结果:

```
Please enter two integers(a and b): 3 4
The sum of square of a and b: 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;
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

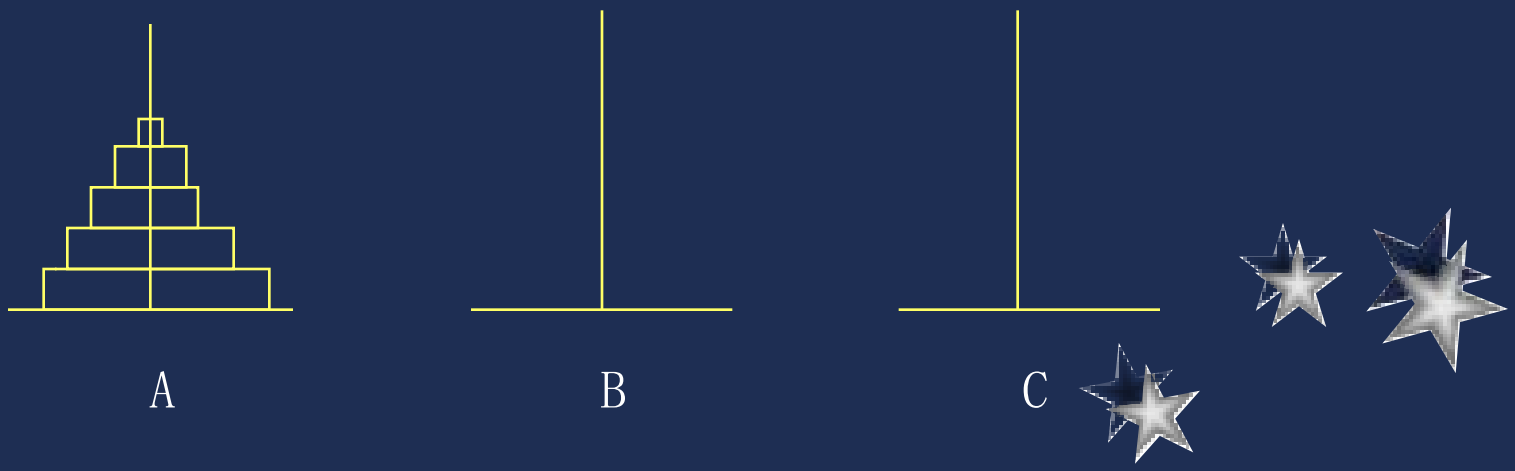
```
#include <iostream>
using namespace std;

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;

//把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;
}

//把n个盘子从src针移动到dest针，以medium针作为中介
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
```

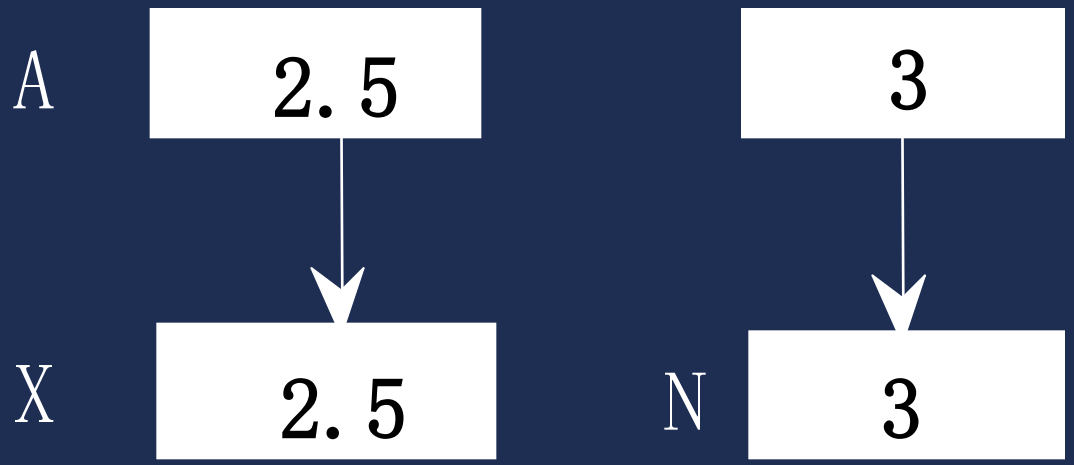
函数的参数传递机制
——传递参数值

- 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) {
   int t = a;
   a = b;
   b = t;
}
```

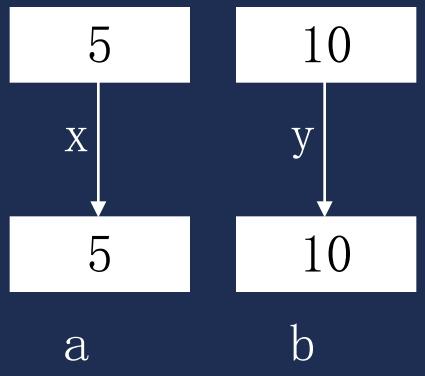
```
int main() {
  int x = 5, y = 10;
  cout << "x = " << x << "  y = " << y << endl;
  swap(x, y);
  cout << "x = " << x << "  y = " << y << endl;
  return 0;
}
```

运行结果:

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

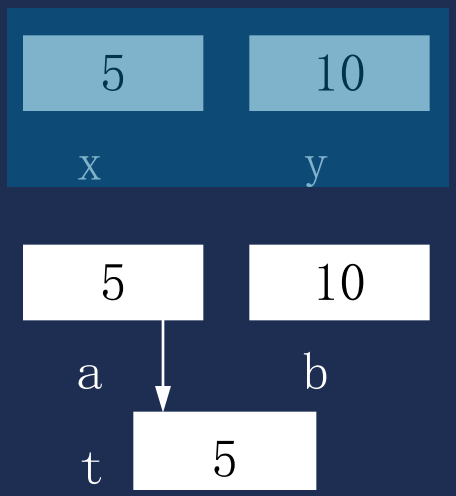
执行主函数中的函数调用

swap(x,y);

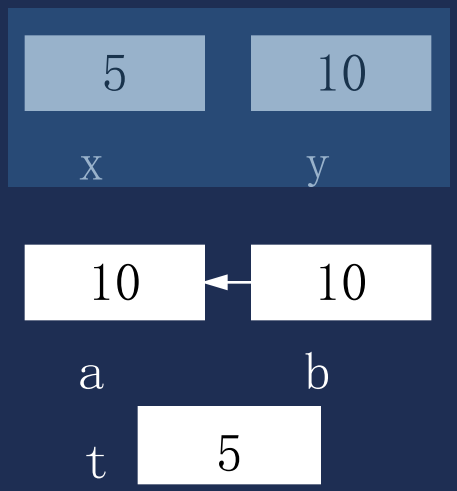


在swap子函数中

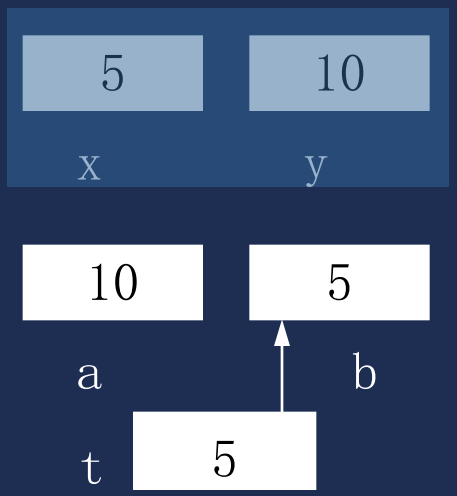
t=a;



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) {...}
```

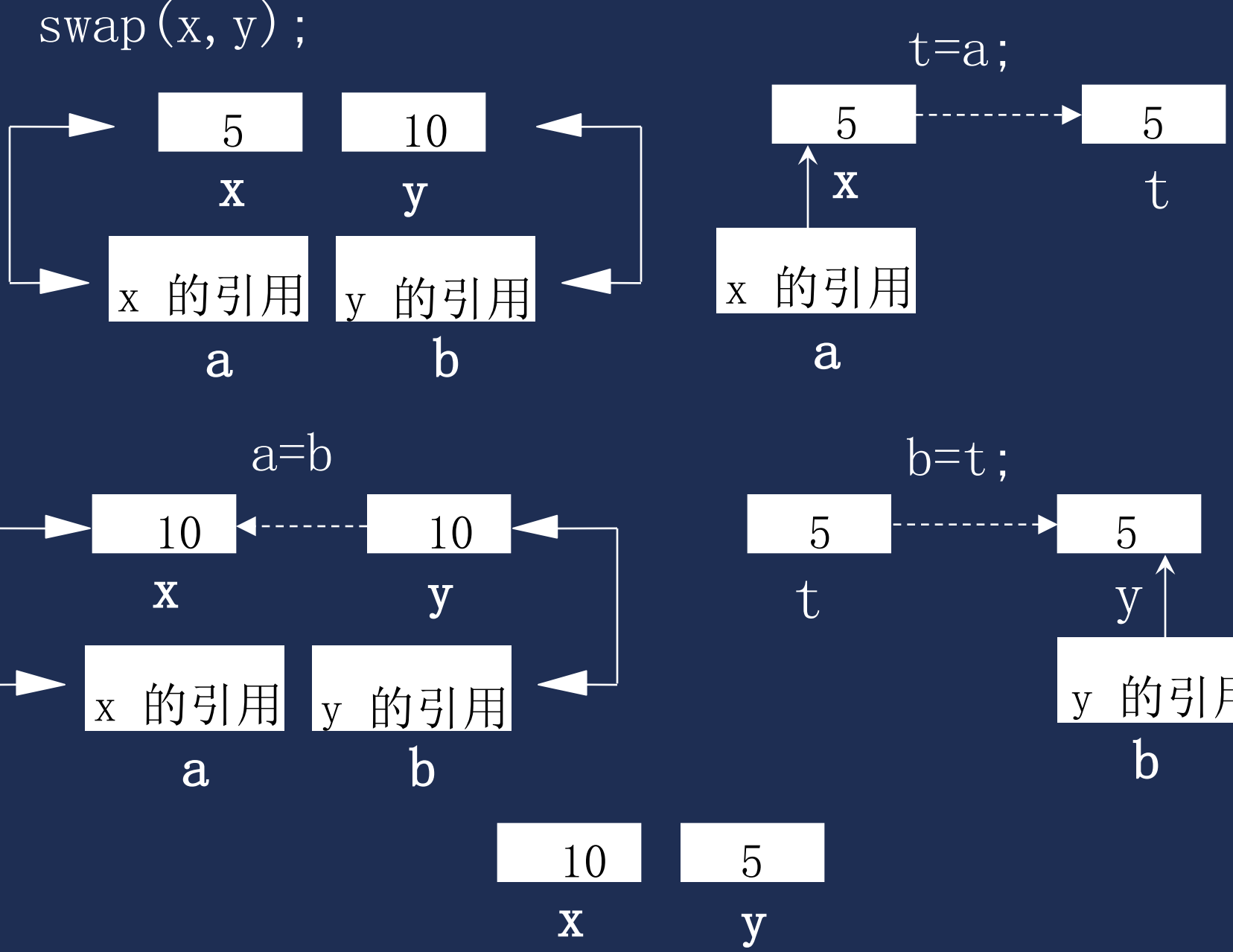
例3-12 输入两个整数交换后输出

```
#include<iostream>
using namespace std;
void swap(int& a, int& b) {
   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

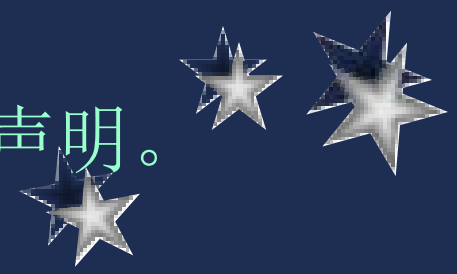
x = 10 y = 5


swap(x, y);
5
10
x
y
x 的引用
y 的引用
a
b
t=a;
5
5
x
t
x 的引用
a
a=b
10
10
x
y
x 的引用
y 的引用
a
b
b=t;
5
5
t
y
y 的引用
b
10
5
x
y

内联函数声明与使用

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;

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;
}
```

缺省形参值的作用

- 函数在声明时可以预先给出缺省的形参值，调用时如给出实参，则采用实参值，否则采用预先给出的缺省形参值。
- 例如：

```
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
}
```

缺省形参值的说明次序

- 有缺省参数的形参必须在形参列表的最后，也就是说缺省形参值的右面不能有无缺省值的参数。因为调用时实参与形参的结合是从左向右的顺序。
- 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
```

带缺省形参值的函数

缺省形参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，则缺省形参值必须在函数原型声明中给出；而如果只有函数的定义，或函数定义在前，则缺省形参值需在函数定义中给出。
- 例：

```
int add(int x = 5,int y = 6);
//原型声明在前
int main() {
  add();
}
int add(int x,int y) {
//此处不能再指定缺省值
  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();
}
```

带默认形参值的函数

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唯一，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);
int 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(void)
{   ...
    add();  //使用全局默认形参值（实现1+2）
}
```

函数重载

函数重载

- 函数的重载：两个以上的函数，具有相同的函数名，但是形参的个数或者类型不同。
- C中实现整数和浮点数的加法：

```
int iadd(int x, int y);
float fadd(float x, float y);
```

重载函数的声明

函数重载

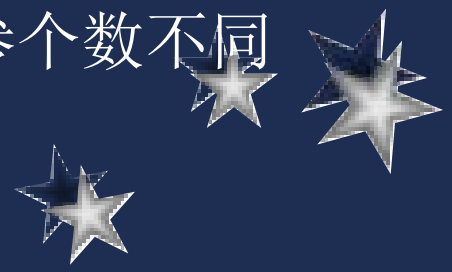
- C++允许功能相近的函数在相同的作用域内以相同函数名声明，从而形成重载。方便使用，便于记忆。
- 例：

```
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。这样不好:

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

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

注意事项

函数重载

- 当使用具有缺省形参值的函数重载形式时，要防止二义性

```
void fun(int length, int width=2, int height=33);
void fun(int length);

fun(1);
```

- 编译器无法确定执行哪个重构函数，报语法错误

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
```

使用C++系统函数

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（sprt）、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：cmath 或 math.h

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。

```
#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;    //转化为弧度值
   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

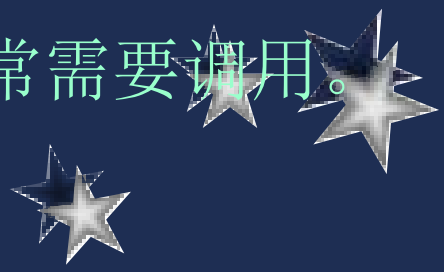
cos(30)=0.866025

tan(30)=0.57735

使用C++系统函数

标准函数与非标准函数

- 标准C++函数
  - C++标准中规定的函数；
  - 各种编译环境普遍支持，因此用标准函数的程序移植性好；
  - 很多标准C++函数继承自标准C，头文件以c开头：cmath，cstdlib，cstdio，ctime……
- 非标准C++函数
  - 与特定操作系统或编译环境相关；
  - 在处理和操作系统相关事务时常常需要调用。



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
- http://www.cplusplus.com/
  - 库函数 http://www.cplusplus.com/reference/
- http://www.cppreference.com/

形参和局部变量的存储

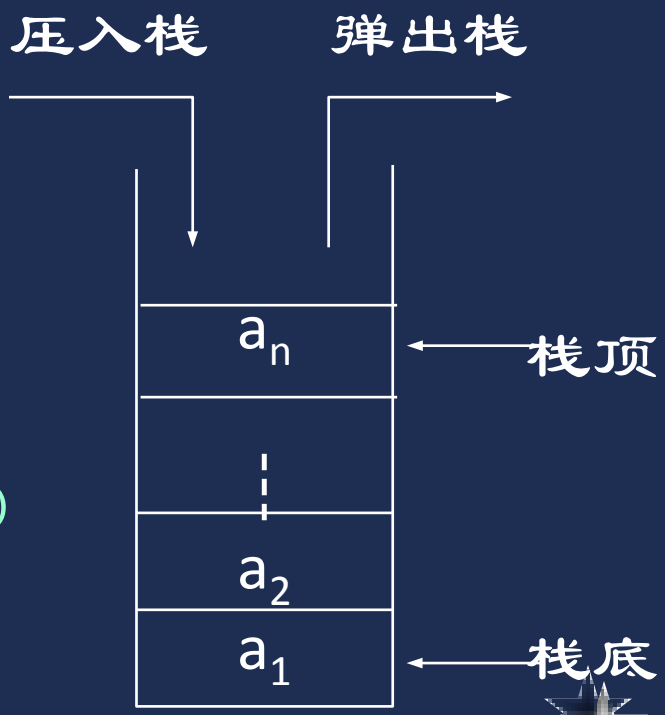
深度探索

- 为什么不能为形参和局部变量分配固定地址？
  - 他们仅在函数调用时生效，函数返回后即失效，分配固定地址造成空间浪费
  - 更重要的是，发生递归调用时，多次调用间的形参和局部变量应彼此独立
- 需要栈式存储

栈

深度探索

- 栈是一种容纳数据的容器
  - 数据只能从栈的一端存入（压入栈）
  - 数据只能从栈的同一端取出（弹出栈）



运行栈

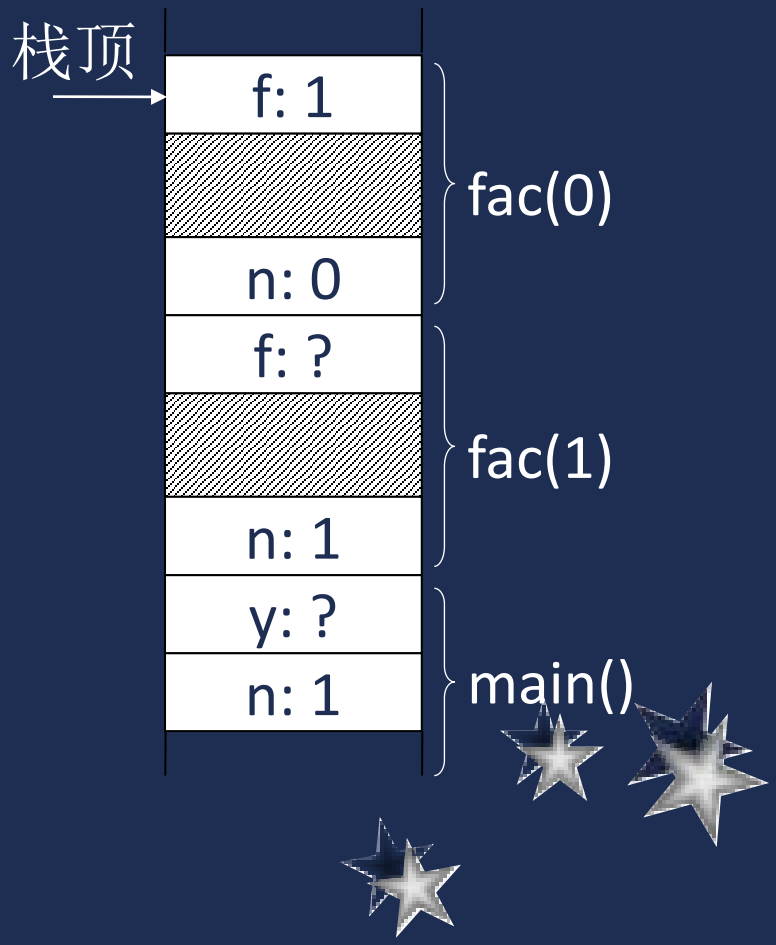
深度探索

- 运行栈是一段区域的内存空间
- 运行栈分为一个一个栈帧
  - 每个栈帧对应一次函数调用
  - 栈帧中包括：
    - 本次函数调用的形参值
    - 控制信息
    - 局部变量值
    - 一些临时数据
  - 函数调用时，会有一个栈帧被压入运行栈
  - 返回时，会有一个栈帧被弹出

运行栈示意图

深度探索

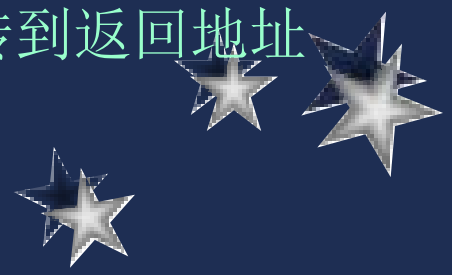
```
unsigned fac(unsigned n) {
  unsigned f;
  if (n == 0)
    f = 1;
  else
    f = fac(n - 1) * n;
  return f;
}
int main() {
  unsigned n;
  cin >> n;
  unsigned y = fac(n);
  ……
}
```



函数调用的执行过程

深度探索

- 栈指针esp：指向运行栈栈顶
- 帧指针ebp：定位形参和局部变量
- 传递参数：调用前把实参压入堆栈
- 函数调用时的几步关键操作
  - call指令：将下一条指令地址（返回地址）压入运行栈，转到函数入口地址
  - 被调函数入口处：将当前ebp压入运行栈，用ebp保存esp，调整esp为局部变量留出空间
  - 被调函数出口处，leave指令：用ebp恢复esp，从运行栈中弹出ebp原值
  - ret指令：将返回地址从运行栈弹出，转到返回地址



运行栈的数据分布

深度探索


```
unsigned fac(unsigned n) {
  unsigned f;
  if (n == 0)
    f = 1;
  else
    f = fac(n - 1) * n;
  return f;
}
```

形参和局部变量定位：

ebp + 8：形参n

ebp - 4：局部变量f

函数调用的执行过程

深度探索

```
0040132F    push   %ebp
00401330    mov    %esp,%ebp
00401332    and    $0xfffffff0,%esp
00401335    sub    $0x20,%esp
00401338    call   0x413380 <__main>
0040133D    movl   $0x7,0x4(%esp)
00401345    movl   $0x5,(%esp)
0040134C    call   0x401318 <add(int, int)>
00401351    mov    %eax,0x1c(%esp)
00401355    mov    $0x0,%eax
0040135A    leave
0040135B    ret
```

```
int add(int a,  int b)
{
    int c = a+b;
    return c;
}

int main()
{
    int x = add(5, 7);

    return 0;
}
```

形参和局部变量定位：

esp, esp+4：实参a,b

esp + 1c：局部变量x

函数调用的执行过程

深度探索

```
00401318        push   %ebp
00401319        mov    %esp,%ebp
0040131B        sub    $0x10,%esp
0040131E        mov    0xc(%ebp),%eax
00401321        mov    0x8(%ebp),%edx
00401324        lea    (%edx,%eax,1),%eax
00401327        mov    %eax,-0x4(%ebp)
0040132A        mov    -0x4(%ebp),%eax
0040132D        leave
0040132E        ret
```

```
int add(int a, int b)
{
    int c = a+b;
    return c;
}

int main()
{
    int x = add(5,7);

    return 0;
}
```

形参和局部变量定位：

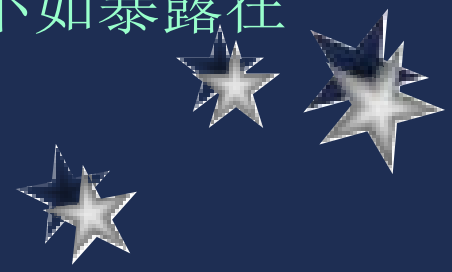
ebp+8, esp+c：形参a,b

ebp - 4：局部变量c

函数声明的意义

深度探索

- 以错误方式调用函数的危险性
  - 函数的原型信息（参数个数和类型、返回值类型）在编译后即不存在；
  - 如果不要求声明函数，以错误的方式（错误的参数数量或类型）调用函数，会产生不可预期的结果，但很多情况下不会给出错误提示。
- 函数原型是主调函数与被调函数间的协议
- 运行结果错误 vs 编译错误
  - 一个错误，与其被淹没在运行中，不如暴露在编译时。



C语言的反例

- C语言允许
  - 只声明函数名和返回类型，而不声明参数类型
  - 不声明函数，直接调用
- 后果：隐蔽错误
  - 如果给出add()的完整声明，则会自动进行类型转换。
- 声明带来了类型安全

```
double add();
int main() {
   double s = add(1, 2);
   ….
   return 0;
}

double add(double a,
   double b) {
   return a + b;
}
```

不完整的函数声明

错误的调用，压入运行栈的是整数！

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。