第六讲 函数

18 信息安全 吕九洋

目录

第零节 函数初探

为什么要使用函数？

我们来看一个例子：

输入三个数，判断他们数位之和是否相等

```c
#include <stdio.h>

int digitsum(int n)
{
    int ans = 0;
    while(n) {
        ans += n % 10;
        n /= 10;
    }
    return ans;
}

int main()
{
    int a, b, c;
    scanf("%d%d%d", &a, &b, &c);
    if(digitsum(a) == digitsum(b) && digitsum(b) ==digitsum(c))
        printf("Equal\n");
    else
        printf("Not equal\n");
    return 0;
}
```

```c
#include <stdio.h>

int main()
{
    int a, b, c;
    scanf("%d%d%d", &a, &b, &c);
    int suma = 0;
    while(a) {
        suma += a % 10;
        a /= 10;
    }
    int sumb = 0;
    while(b) {
        sumb += b % 10;
        b /= 10;
    }
    int sumc = 0;
    while(c) {
        sumc += c % 10;
        c /= 10;
    }
    if(suma == sumb && sumb == sumc)
        printf("Equal\n");
    else
        printf("Not equal\n");
    return 0;
}
```

可以看出右侧代码存在大量重复的片段，而且main函数里面代码过多丧失了可读性。

为什么要使用函数？

- “函数”可以将实现了某一**功能**，并需要反复使用的代码包装起来形成一个**功能模块**（即写成一个“函数”），那么当程序中需要使用该项功能时，只需写一条语句，调用实现该功能的“函数”即可。
- 不同的程序员可以分别写不同的函数，拼起来形成一个大程序。

第一节 函数的定义和使用

1.1.1 函数的定义

```
返回值类型 函数名(参数1类型 参数1名称, 参数2类型 参数2名称……)
{
    语句组(函数体);
    返回值;
}
```

- **函数不允许嵌套定义**，在一个函数内定义另一个函数是非法的，但是允许嵌套使用。

```
int sum(int a, int b)
{
    int ans = 0;
    ans = a + b;
    return ans;
}
```

1.1.2 函数的调用

`调用函数： 函数名（参数1，参数2，……）`

- 对函数的调用，也是一个表达式。函数调用表达式的值，由函数内部的`return`语句决定。
- `printf`也是函数，它的声明在`<stdio.h>`头文件里。

Q: 为什么大侠们使用绝招时都要先喊一声“降龙十八掌”之类的？
A: 因为**函数调用之前要先声明**。

```
#include <stdio.h>
int sum(int a, int b); //声明
int main()
{
        int a = 1, b = 2;
        printf("%d\n", sum(a, b));
}
int sum(int a, int b)
{
        return a + b;
}
```

1.1.3 函数的返回值

return语句语法如下：

```
return 返回值;
```

- return语句的功能是**结束函数的执行，并将“返回值”作为结果返回**。“返回值”是常量、变量或复杂的表达式均可。
- 如果函数返回值类型为`void`，return语句就直接写`return ;`

```
int a[] = {1, 1, 2, 3, 5, 8, 13};
void show(int n)
{
    for(int i = 0; i < n; i++)
        printf("%d ", a[i]);
    return ;
}
```

1.1.3 函数的返回值

return 语句作为函数的出口，可以在函数中多次出现。多个return语句的返回值可以不同。

```
int Max(int x,int y) //求两个整型变量中的较大值
{
    if(x > y)
        return x;
    return y; /*除void外函数一定要有返回，返回意味着函数结束，
                这里我们不写else，写else有的IDE会报错*/
}
```

1.1.4 ~~mian~~ main函数

C/C++程序从main函数开始执行，执行到`return 0;`则结束。

main函数也是一个函数。

```
//C++基本语法
#include <iostream>
using namespace std;

int main()
{
        cout << "Hello, World!" << endl;
        return 0;
}
```

1.2 局部变量与全局变量

- 在函数外部定义的变量称为全局变量，在函数内部定义的变量称为局部变量。

```
#include <stdio.h>
int a[105]; //数组a-全局
int n; //n-全局
int muln(int x) //x-局部
{
    int t; //t-局部
    t = x * n;
    return t;
}
int main()
{
    scanf("%d", &n);
    for(int i = 0; i < n; i++) //i-局部
        scanf("%d", &a[i]);
}
```

1.2.1 全局变量

- 全局变量的作用域是从变量定义的位置开始到文件结束，可以在文件中位于全局变量定义后面的任何函数中使用。
- 过多地使用全局变量，会增加调试难度。因为多个函数都能改变全局变量的值。
- 全局变量在定义时默认初值为0。

1.2.2 局部变量

- 局部变量的作用域是在定义该变量的函数内部，局部变量只在定义它的函数内有效。
- 在不同的函数中局部变量名可以相同。
- 一个局部变量和一个全局变量是可以重名的，在相同的作用域内局部变量有效。但易出错！尽量避免使用。
- 在代码块中定义的变量的存在时间和作用域将被限制在该代码块中。如`for(int i; i<=n; i++) sum += i;`中的`i`是在该`for`循环语句中定义的，存在时间和作用域只能被限制在该for循环语句中。
- 局部变量值是随机的，要**初始化初值**。
- 局部变量受栈空间大小限制，大数组需要注意。通俗地说，main函数里数组不能开很大（十万级别）。

1.2 示例

- 写一个判断素数的函数，返回0或1代表是否为素数

```
#include <stdio.h>
bool IsPrime(int n)
{
    if(n <= 1)
        return false;
    for(int i = 2; i < n; ++i) //看看有没有n的因子
        if(n%i == 0) return false;
    return true;
}
int main()
{
    int n;
    scanf("%d", &n);
    printf("%d\n", IsPrime(n));
    return 0;
}
```

1.3 函数参数的传递

- 在C++语言中，函数调用方式分传值调用和传址调用。

1.3.1 传值调用

传值调用即将实参的**数据值**传递给形参

```
#include<iostream>
using namespace std;
void swap(int a, int b)
{
     int t = a; a = b; b = t;
}
int main()
{
      int c = 1, d = 2;
      swap(c, d);
      cout << c << ' ' << d << endl;
      return 0;
} //程序输出为: 1 2
```

在此例中，虽然在swap函数中交换了a,b两数的值，但是在main中却没有交换。因为swap函数只是交换c,d两变量的值，实参值没有改变，并没有达到交换的目的。

1.3.2 传址调用

传址调用是将实参变量的地址值传递给形参，写法为在形参前加`&`。严格来说gcc(C语言编译器)不支持引用传参。

```
#include<iostream>
using namespace std;
void swap(int &a, int &b)   //定义swap()函数，形参是传址调用
{
     int tmp = a; a = b; b = tmp;
}
int main()
{
      int c = 1, d = 2;
      swap(c, d);                //交换变量
      cout << c << ' ' << d <<endl;
      return 0;
}  //程序输出为: 2 1
```

在此例中，因为swap函数的参数为传址调用，&a是指实参变量的地址值传递给形参，所以，在函数swap中修改a,b的值相当于在主函数main中修改c,d的值。

什么时候传值？什么时候传址？

- 一般情况下都是传值调用
- 当变量经过函数值改变时传址

```
void changeToThree(int &x)
{
    x = 3;
}
```

- 想要多个返回值时传址
- 引用就是起别名

1.3.3 函数参数的传递

原则： 用什么传什么

```
返回值类型 函数名(参数1类型 参数1名称, 参数2类型 参数2名称……)
{
    函数体;
    返回值;
}
```

1.3.4 整型/实型变量作函数参数

```
double dis(double x1, double y1, double x2, double y2)
{
    return sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
}
int fun(double x)
{
    return x; //double -> int 类型转换
}
int Max(int a, int b)
{
    return a > b ? a : b;
}
```

1.3.5 一维数组作函数参数

写法如下：

`函数类型 函数名(数组类型名 数组名[])`

不用写出数组的元素个数。

例如：

```
void PrintArray(int a[])
{
    ...
}
int b[] = {1, 2, 3};
int main()
{
    PrintArray(b); //调用时只用写数组名
}
```

- 数组作为函数参数时是传址引用的，即形参数组改变了，实参数组也会改变。

1.3.6 二维数组作为函数的参数

- 二维数组作为形参时，必须写明数组有多少列，不用写明有多少行。

```
void PrintArray(int a[][5])
{
    printf("%d", a[4][3]);
}
```

1.4 模块化编程思想

适当使用函数

```
8   void read()
9   {...
17
18  int check()
19  {...
26
27  void print()
28  {...
41
42  int main()
43  {
44      read();
45      if(check() == 1) cout << "Repeat Error!" << endl;
46      else print();
47      return 0;
48  }
```

```
8   void read()
9   {
10      cin >> n;
11      for(int i = 0; i < n; i++)
12          cin >> t, a[t]++, mp[t]++, v.push_back(t);
13      cin >> n;
14      for(int i = 0; i < n; i++)
15          cin >> t, b[t]++, mp[t]+=2, v.push_back(t);
16  }
17
18  int check()
19  {
20      for(it = a.begin(); it != a.end(); it++)
21          if((*it).second >= 2) return 1;
22      for(it = b.begin(); it != b.end(); it++)
23          if((*it).second >= 2) return 1;
24      return 0;
25  }
26
27  void print()
28  {
29      cout << "CU:";
30      for(int i = 0; i < v.size(); i++)
31          if(mp[v[i]] && s.count(v[i])==0) cout << v[i] << " ", s.insert(v[i]);
32      cout << endl;
33      cout << "CD:";
34      for(int i = 0; i < v.size(); i++)
35          if(mp[v[i]] == 1)cout << v[i] << " ", mp[v[i]]+=10;
36      cout << endl;
37      cout << "CI:";
38      for(int i = 0; i < v.size(); i++)
39          if(mp[v[i]] == 3)cout << v[i] << " ", mp[v[i]]+=100;
```

第二节 常用标准库函数

C/C++参考文档

- 一旦使用头文件后不可以定义库函数，比如，使用`<stdio.h>`后不能再自己定义`printf`函数。原则上，平常定义函数时也应尽量避免与库函数重名。

2.1 输入输出函数

函数名：`scanf`
所需头文件：`<stdio.h>`
功能：输入数据
示例：`scanf("%d", &a);`

函数名：`printf`
所需头文件：`<stdio.h>`
功能：输出数据
示例：`printf("ans is %d", a);`

2.2 数学函数 `<math.h>`

函数名：`abs`
参数：需要求绝对值的整数n。
函数原型：`int abs(int n);`
所需头文件：`<stdlib.h>` 或 `<math.h>`
功能和返回值：返回n的绝对值
示例：`int n = -1; int t = abs(n);`

函数名：`fabs`
参数：需要求绝对值的浮点数x。
函数原型：`double fabs(double x);`
所需头文件：`<math.h>`
功能和返回值：计算并返回浮点参数x的绝对值。
示例：`double x = -1.0; double t = fabs(x);`

函数名：`sqrt`
函数原型：`double sqrt(double x);`
所需头文件：`<math.h>`
功能和返回值：计算并返回x的平方根。
示例：`double t = sqrt(3.0);`

函数名：`pow`
函数原型：`double pow(double x, double y);`
所需头文件：`<math.h>`
功能和返回值：计算并返回**double型**x的y次幂。

函数名：`ceil`
函数原型：`double ceil(double x);`
所需头文件：`<math.h>`
功能：对浮点数x向上取整，并**以double型浮点数形式**存储结果。
返回值：返回一个double型的大于或等于x的最小整数
示例：`double t = ceil(x);`

函数名：`floor`
函数原型：`double floor(double x);`
所需头文件：`<math.h>`
功能：对浮点数x向下取整，并**以double型浮点数形式**存储结果。
返回值：返回一个double型的小于或等于x的最大整数
示例：`double t = floor(x);`

函数名：`max`
函数原型：`type max (type a, type b);`
参数：type任何数值数据类型；a和b是参与比较的两个数，必须是相同类型。
所需头文件：`<stdlib.h>`
功能和返回值：比较a和b并返回其中较大者。

函数名：`min`
函数原型：`type min (type a, type b);`
参数：type任何数值数据类型。a和b是参与比较的两个数，必须是相同类型。
所需头文件：`<stdlib.h>`
功能和返回值：比较a和b并返回其中较小者。

函数名：`cos，cosh`
函数原型：`double cos(double x);`
参数：弧度值x
所需头文件：`<math.h>`
功能和返回值：计算并返回x的余弦值(cos)或双曲余弦值(cosh)。

函数名：`log`
函数原型：`double log(double x);`
所需头文件：`<math.h>`
功能和返回值：计算并返回x的自然对数。如果x是负数，返回值不确定。如果x为0，返回INF(无穷大)。

2.3 缓冲区操作函数

函数名：`memset`
函数原型：`void *memset(void *dest, int c, size_t count);`
参数：目的指针dest；设置的字符c；字符个数count。
所需头文件：`<string.h>`
功能：设置dest的前count个字节为字符c。
返回值：返回dest的值。
示例：

```
int a[100];
memset(a, 0, sizeof(a));
```

2.4 字符串操作函数

函数名：`strlen`

函数原型：`unsigned int strlen(const char*string);`

参数：以空字符结尾的字符串string。

所需头文件：`<string.h>`

功能和返回值：返回string中的字符个数。

示例：

```
char s[] = "zlsnb";
printf("%d\n", strlen(s));
```

函数名：`strcat`

函数原型：

```
char *strcat(char*strDestination, const char *strSource);
```

参数：目的字符串 源字符串

所需头文件：`<string.h>`

功能：连接两字符串

示例：

```
char str1[]="zls", str2[]="nb";
puts(strcat(str1,str2));
```

函数名：`strcmp`

函数原型：`int strcmp(const char*string1，constchar*string2);`

参数：两个被比较的字符串。

所需头文件：`<string.h>`

功能：按**字典序**比较string1和string2，并返回一个值指出它们之间的关系。

返回值：返回值<0，string1小于string2；返回值＝0，string1等于string2；返回值>0，string1大于string2。

```
1."A"<"B"
2."A"<"AB"
3."10"<"2"
4."A"<"a"
5."compare"<"computer"
```

```
char str1[]="zls", str2[]="Tourist";
int result = strcmp(str1,str2); //1, 代表字典序"zls">"Tourist"
```

第三节 简单递推

3.1 斐波那契数列

1，1，2，3，5，8，13，21，34.....

斐波那契数列怎么求？

$$F_1 = F_2 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

最直接：循环

```
int f[105];
f[1] = 1; f[2] = 1;
for(int i = 3; i <= n; i++)
    f[i] = f[i-1] + f[i-2];
```

那如何不开数组求出斐波那契数列第n项？

3.2 递推算法

递推算法是一种简单的算法，即通过已知条件，利用特定关系得出中间推论，直至得到结果的算法。
简单递推格式

```
函数类型  函数名（参数）
{
    终止条件;
    函数关系;
}
```

$$F_1 = F_2 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

```
int Fibonacci(int n)
{
    if(n == 1 || n == 2) return 1;
    return Fibonacci(n-1) + Fibonacci(n-2);
}
```

递推和循环（迭代）之间的关系

递推和循环可以达到相同的效果

- 递推不占数组空间
- 循环相当于把整个结果存起来，一次操作满足多次询问

关于递归和递推之后还会进行深入学习

课后练习

请尽量用函数完成本次练习

A. zls的小弟

问题描述

zls有一句名言："我小弟的小弟，还是我的小弟"。

zls开始招收小弟了！zls需要1天时间研究如何招收小弟，之后他每天都能招收一个小弟。而每个zls的小弟经过1天研习zls经验之后也可以招收小弟。那么第n天zls一共有多少小弟呢？

输入

一个整数t。接下来t行，每行一个数字n $(1 \leq n \leq 60)$，分别代表t次询问。

输出

第n天zls的小弟数量（包括zls自己）。

样例输入

```
5
1
2
3
5
50
```

样例输出

```
1
1
2
5
12586269025
```

B. zls的一元二次方程

问题描述

zls作为新时代优秀青年，上可写机器学习，下可造机器人解一元二次方程，深受周围很多女生的爱慕。然而ljy只会手解一元二次方程，与zls的巨大差距已经肉眼不可见了。为了能略微缩小与zls之间的巨大差距，ljy想写一个程序解**一般的**一元二次方程。你能帮帮他吗？

输入

三个浮点数a, b, c，代表一元二次方程 $ax^2 + bx + c = 0$

输出

解的数量和解，中间以一个空格隔开

样例输入

```
1 2 1
```

样例输出

```
1 -1.000000
```

样例输入

```
3 50 4
```

样例输出

```
2 -0.080388 -16.586279
```

样例输入

```
1 1 1
```

样例输出

```
0
```

C. zls的高考题

问题描述

zls今天看见了这么多师弟师妹来学习，不由得兴奋地想起当年的自己。想到这，zls情不自禁地拿出2016全国三卷理科数学高考最后一道选择题准备再体验一把当年秒题的快感。原题如下：

定义“规范01数列” a 如下: a 共有 $2m$ 项, 其中 m 项为 0 , m 项为1, 且对任意 $k \leq 2m$, $a_1, a_2,\ldots, a_k$中 0 的个数不少于 1 的个数.若 $m = 4$，则不同的“规范01数列”共有（14）个。

当然，zls很快就做出来这道题，但他还是认为这道题太简单了，他在想 m 取其他值结果是多少呢？

输入

一个正整数m, $(1 \leq m \leq 8)$

输出

问题的解，保证结果在int范围内

样例输入

```
4
```

样例输出

```
14
```

样例输入

```
8
```

样例输出

```
1430
```

D. zls的女神

问题描述

初见长路踩红砖球场鼎沸彩裙飘 / 白云蓝天新鲜空气载着憧憬向往 / 帆布包匆匆忙忙也道书意浓 / 自信笑语脚步轻扬可是正当年的青春。

zls就这样遇见了自己的女神。走在白杨大道上，zls想把自己和女神的距离缩短到一块地砖，一米，一厘米之内。已知zls一步步长x厘米，女神步长y厘米，女神和zls同时同向踏上白杨大道，问在多远zls和女神第一次相遇？（相遇指脚步同时落在同一位置）

输入

两个`int`范围内的正整数，代表zls和女神的步长，由于zls要尽量与女神相遇，所以他会调整自己的速度，故不用考虑时间。

输出

zls第一次遇到女神的位置，单位为厘米，不考虑初始位置。数据保证zls一定可以和女神相遇。

样例输入

```
80 60
```

样例输出

```
240
```

样例输入

```
100000000 40
```

样例输出

```
100000000
```

E. zlstql!

问题描述

众所周知，刷题网站 *Codeforces* 上世界排名前三的账号分别是zls的小号、小小号和小号的小号（zlstql!）。原因就在于zls平常使用的IDE。

ACM队队员都有自己喜欢的IDE，夏教喜欢用 *Codeblocks*，zxls爱用*Clion*，jdl和jdl都钟情于*VS*，ljy习惯用*Dev-C++*，而zls日常竟然是直接在记事本里编程的！(zlstql!)。但是记事本里冒号和分号很难区分，现在给你一份ljy误改过的zls的代码（zls的代码怎么会有bug），你能把程序中的所有冒号换成分号吗？

输入

程序的行数n，接下来n行是程序代码

输出

转换完的代码

样例输入

```
14
#include<stdio.h>
int main() {
    for(int i=100: i<1e4: ++i) {
        int ans=i://临时变量
        int num=0://记录A^3+B^3+C^3的值
        while(ans>0) {
            num+=(ans%10)*(ans%10)*(ans%10):
            ans/=10://每次更新ans,用于找下一位
        }
        if(num==i)
            printf("%d\n",i):
    }
    return 0:
}
```

样例输出

```
#include<stdio.h>
int main() {
    for(int i=100; i<1e4; ++i) {
        int ans=i;//临时变量
        int num=0;//记录A^3+B^3+C^3的值
        while(ans>0) {
            num+=(ans%10)*(ans%10)*(ans%10);
            ans/=10;//每次更新ans,用于找下一位
        }
        if(num==i)
            printf("%d\n",i);
    }
    return 0;
}
```

- 请使用C++中的string，不知道如何读入可以看这个

F. zls的24点

问题描述

“24点”是一种益智游戏，玩法是把扑克牌上4个数字通过加减乘除以及括号运算，使最后的计算结果是24。如(3+5-4)×6=24。zls觉得一般的24点太无聊了，他发明了“zls的24点”。盒子里装着 n 张写有数字的扑克牌，数字大小不限，可以进行任意次以下操作：

1.对于盒子中的任意两个数a,b，可以进行

$$(1)\,a+b\ (2)\,a-b\ (3)\,a\times b\ (4)\,a\div b\ (b\neq 0)\ (5)\,a^b\ (6)\,log_a b$$

任何一个操作后并将结果放回盒子，所有运算都遵从数学规律。

2.对于盒子中的任意一个数 a，可以进行

$$(1)\,a!\text{(阶乘)}\ (2)\,\frac{a}{100}\ (3)\,\sqrt{a}\ (4)\,|a|\ (5)\,\lceil a\rceil\text{(向上取整)}\ (6)\,\lfloor a\rfloor\text{(向下取整)}$$

任何一个操作后并将结果放回盒子，所有运算都遵从数学规律。

输入

游戏开始时扑克牌的数目n (1 ≤ n ≤ 5)，

接下来 n 个整数，代表 n 张扑克牌上面的数值。

输出

若进行若干次操作后，盒子中能只剩下24一个数，则输出Yes，

反之输出No。

样例输入

```
4 0 0 0 0
```

样例输出

```
Yes
```

样例输入

```
1 0
```

样例输出

```
No
```

样例说明

(0! + 0! + 0! + 0!)! = 24

只有一个0时无论如何也计算不出24，故输出No

样例输入

```
2 0 2
```

样例输出

```
Yes
```

样例输入

```
4 4 4 4 4
```

样例输出

```
Yes
```

样例输入

```
5 100000 100000 100000 100000 100024
```

样例输出

```
Yes
```

G. zls女装

问题描述

zls的女装需要一块 $m \times n$ ($1 \leq m, n \leq 50$)的布料，然而对于少女心的zls来说，红黄蓝绿四种颜色TA全都要！如果聪明的你设计出来这件衣服，就可以看到zls女装了！

输入

四种颜色块的数量a_i ($1 \leq a_i \leq 100$)。一个颜色连通的区域称为一个颜色块。

输出

m, n，之后是m行n列的zls女装设计图，四种颜色分别用ABCD代替。答案不唯一。

样例输入

```
1 1 3 1
```

样例输出（答案不唯一）

```
3 4
ABCD
BBBD
CBCD
```

样例输入

```
1 1 1 1
```

样例输出（答案不唯一）

```
3 7
AAAAAAA
ABACADA
AAAAAAA
```