

计算机导论与程序设计 [CS006001-60]

段江涛

机电工程学院



2019 年 10 月

lecture-13 主要内容

用函数实现模块化程序设计

- 1 使用函数进行模块化程序设计
 - 函数三要素: 函数原型声明, 函数定义, 函数调用
 - 函数定义: `int add(int a,int b){ }`
- 2 实参和形参间的数据传递 (值传递和地址传递)
- 3 程序举例

为什么使用函数 (1)

函数调用

```
#include<stdio.h>
#include<math.h>
int main() // 主函数
{
    int a;
    // 库函数调用
    scanf("%d",&a);
    a=(int) fabs(a);
    printf("%d\n",a);
    return 0;
}
```

函数定义

```
int scanf(char format[], args, ...)
{ ...;
    return 整型值;
}

int printf(char format[], args, ...)
{ ...;
    return 整型值;
}

double fabs(double a)
{ // 模拟代码, 没有考虑精度
    if(a<0) return -a;
    else return a;
}
```

为什么使用函数 (2)

函数调用

```
#include<stdio.h>
#include<string.h>
int main() // 主函数
{
    int a;
    char s1[81],char s2[81]="1234";
    // 库函数调用
    strcpy(s1,s2);
    a=strcmp(s1,s2);
    printf("%d",a);
    printf("%d\n",strlen(s1));
    return 0;
}
```

函数定义

```
char[] strcpy(char s1[],char s2[])
{ ...; return s1;
}

int strcmp(char s1[],char s2[])
{ ...; return 整型值; // 1,-1,0
}

int strlen(char s[])
{
    int len=0;
    while(s[len]!='\0') len++;
    return len;
}
```

功能分解是简化程序设计的有效手段

功能分解, 函数调用

```
#include<stdio.h>
#include<math.h>
int add(int a,int b); // 函数原型声明
double output(double a);
int main() // 主函数
{
    int a=10,b=20,c;
    //传递参数a,b的值, 计算结果赋值给变量c
    c=add(a,b); // 函数调用
    output(10.5*c); // 函数调用
    return 0;
}
```

分解功能实现, 函数定义

```
// 通过参数a,b的值, 进行相关计算, 返回整型数据给调用者
int add(int a,int b)
{ return a+b; // 返回整型值 }
// 通过参数a的值, 进行相关计算, 不需要返回数据
void output(double a)
{
    printf("%ld\n", sqrt(a)); // 函数中可调用别的函数
}
```

使用函数进行程序设计的优点

- 1 使用函数可使程序清晰、精炼、简单、灵活。
- 2 函数就是功能。每一个函数用来实现一个特定的功能。函数名应反映其代表的功能。
- 3 在设计较大程序时,往往把它分为若干个程序模块,每一个模块包括一个或多个函数,每个函数实现一个特定的功能。
- 4 一个 C 程序可由一个主函数和若干个其他函数构成。由主函数调用其他函数,其他函数也可以互相调用。

函数三要素: 函数原型声明, 函数定义, 函数调用

函数原型声明, 函数调用

```
#include<stdio.h> // 库函数原型声明
// 函数原型声明, 使编译器认识这个函数, 如果函数定义在调用之前, 可省略声明
int add(int a,int b);
double output(double a);
int main() // 主函数
{
    int a=10,b=20,c;
    // 函数调用, 执行函数功能
    c=add(a,b);
    output(10.5*c);
    return 0;
}
```

函数定义, 定义函数功能

```
// 通过参数a,b的值, 进行相关计算, 返回整型数据给调用者
int add(int a,int b)
{
    return a+b; // 返回整型值
}

// 通过参数a的值, 进行相关计算, 不需要返回数据
void output(double a)
{
    printf("%ld\n", sqrt(a)); // 函数中可调用别的函数
}
```

函数定义: `int add(int a,int b){ }`

函数定义: `int add(int a,int b){ }`

函数定义: 返回类型 函数名 (参数类型 参数, ...) { ... ; }

```
int add(int a, int b)
{
    ...;
    return 整型值; // 必须含return语句, 函数执行结束, 并将返回值返回给调用者
}

double fun(void) // 无参函数, 等效 double fun()
{
    ...;
    return 双精度值; // 必须含return语句, 函数执行结束, 并将返回值返回给调用者
}

void output(double a) // 无返回值函数
{
    ...;
    return; // 可选return语句(注意没有表达式), 仅表示函数执行结束
}
```


函数定义: `int add(int a,int b){ }`

函数调用时数据类型的隐式转换

```
int a=2,b=3,c;  
// pow函数原型: double pow(double x,double y);  
// 编译器自动把a,b"隐式"转换为double  
// 计算a^b的结果是double类型, 赋值语句隐式转换为int, 但是会引起警告信息  
c=pow(a,b);  
c=(int)pow(a,b); // 将函数的返回值强制转换为int, 不会有警告信息
```

实参和形参间的数据传递 (值传递)

```
#include<stdio.h> // 库函数原型声明           // 定义函数求形参x,y中的较大者并返回给调用者
// 函数原型声明, 使编译器认识这个函数
int max(int x,int y);
int main() // 主函数
{
    int a=10,b=20,c;
    scanf("%d%d",&a,&b);
    // 把此处的a,b值拷贝给函数形式参数x,y }
    c=max(a,b); // 实际参数
    printf("较大者=%d\n",c); //与下一句
    等效
    printf("较大者=%d\n",max(a,b));
    return 0;
}
```

在调用函数过程中发生的实参与形参间的数据传递称为“虚实结合”。

实参和形参间的数据传递 (地址传递)

例: 将数组 **a** 中 **n** 个整数按相反顺序存放。

```
#include<stdio.h>

void inv(int x[],int n); // 要求x是地址传递

#define N 100

int main()
{
    int i, n, a[N]={1,2,3,4,5,6,7,8,9,10};
    scanf("%d",&n);
    for(i=0; i<n; i++) scanf("%d",&a[i]);
    // 把实参a数组的地址拷贝给形参x, n的值拷贝给形
    式参数n
    inv(a,n); // 实参a数组的内容被改变
    for(i=0; i<n; i++) printf("%d\u",a[i]);
    return 0;
}
```

```
// 倒置数组x的内容, n是x的长度
// 要求x是地址传递

void inv(int x[],int n)
{
    int i,temp,m=(n-1)/2;
    //以中间元素为界, 前后元素交换
    for(i=0; i<=m; i++)
    {
        temp=x[i];
        x[i]=x[n-1-i];
        x[n-1-i]=temp;
    }
    return; //可选, 函数执行完毕
}
```

值传递与地址传递的不同点

- 值传递, 对形参值的改变不会引起实参值的改变。
- 地址传递虽然不能改变实参的地址, 但是对地址指向内容的改变会引起实参指向内容的改变。
- 数组名表示数组元素在内存中的首地址, 数组元素是连续存放的。
- 用数组名作为参数传递, 就是地址传递。在函数内部对数组元素的改变, 就是改变实参数组元素。

```
void fun(double a[],int n) // a是地址传递, n是值传递
{
    a[1]=20.5; // 改变地址a指向的内容, 就是改变实参数组的元素值
    n=30; // 此处的改变不会影响实参的值
}

int main()
{
    double a[2]={0.8,0.3};
    int n=2;
    fun(a,n); // 实参a的地址拷贝给形参a, 实参n的值拷贝给形参n
    printf("%d,%lf,%lf\n",n,a[0],a[1]); // 2,0.8,20.5
}
```

数组名作为函数参数的注意事项

```

#define M 100 // 估计的数组的第一维长度 // 函数定义时省略第一维大小, 第二维不能省略
#define N 100 // 估计的数组的第二维长度 void fun(int a[][N], int m, int n)
// 函数定义时省略第一维大小 {
// a[]表示a接受地址传递, 以区别于值传递 int i;
void fun(int a[], int n) for(i=0; i<m; i++)
{ {
    int i; for(j=0; j<n; j++)
    for(i=0; i<n; i++) printf("%d\t", a[i][j]);
    printf("%d\t", a[i]); printf("\n");
    printf("\n"); }
} }
// 函数调用 // 函数调用
int x[M]={1, 2, 3, 4, 5}; int x[M][N]={ {1,2}, {3,4}, {5,6},
fun(x, 5); //调用时仅用数组名传递x的地址 {7,8}, {9,10} };
fun(x, 5, 2); //调用时仅用数组名传递x的地址

```

例: PM2.5

给出一组 PM2.5 数据,按以下分级标准统计各级天气的天数,并计算出 PM2.5 平均值。

PM2.5 分级标准为:

一级优 ($0 \leq PM2.5 \leq 50$)

二级良 ($51 \leq PM2.5 \leq 100$)

三级轻度污染 ($101 \leq PM2.5 \leq 150$)

四级中度污染 ($151 \leq PM2.5 \leq 200$)

五级重度污染 ($201 \leq PM2.5 \leq 300$)

六级严重污染 ($PM2.5 > 300$)

输入说明,输入分为两行,

第一行是一个整数 n 表示天数 ($1 < n \leq 100$)

第二行为 n 个非负整数 P_i ($0 \leq P_i \leq 1000$) 表示每天的 PM2.5 值,整数之间用空格分隔。

输出说明,输出两行数据,

第一行为 PM2.5 平均值,结果保留 2 位小数;

第二行依次输出一级优,二级良,三级轻度污染,四级中度污染,五级重度污染,六级严重污染的天数。

例: PM2.5(非模块化设计)

```
#include <stdio.h>

int main()
{
    int i = 0, n, pm25, day[6] = {0, 0, 0, 0, 0, 0}, sum = 0;
    scanf("%d", &n);
    while(i < n)
    {
        scanf("%d", &pm25);
        sum += pm25;
        if(pm25 >= 0 && pm25 <= 50 ) day[0]++;
        else if(pm25 >= 51 && pm25 <= 100 ) day[1]++;
        else if(pm25 >= 101 && pm25 <= 150 ) day[2]++;
        else if(pm25 >= 151 && pm25 <= 200 ) day[3]++;
        else if(pm25 >= 201 && pm25 <= 300 ) day[4]++;
        else day[5]++;
        i++;
    }
    printf("%.2f\n", (float)sum/n);
    for(i = 0; i < 6; i++)
        if(i == 5) printf("%d\n", day[i]);
        else printf("%d_", day[i]);
    return 0;
}
```

例: PM2.5(模块化设计, 主程序)

```
#include <stdio.h>

float haze(int pm25[], int day[], int n, int m); // 函数声明

#define N 1000 // 估计的最大值

int main()
{
    int i = 0, n, pm25[N], day[6];
    scanf("%d", &n);
    while(i < n)
    {
        scanf("%d", &pm25[i]);
        i++;
    }
    printf("%.2f\n", haze(pm25, day, n, 6)); // 函数调用
    for(i = 0; i < 6; i++)
        if(i == 5) printf("%d\n", day[i]);
        else printf("%d_ ", day[i]);
    return 0;
}
```


例: PM2.5(模块化设计, 雾霾统计信息)

```
// pm25[]: PM2.5值, day[]: 不同标准对应的统计天数;  
// n是pm25数组的长度, m是数组day的长度, 返回PM2.5平均值  
float haze(int pm25[], int day[], int n, int m)  
{  
    int i=0, sum=0;  
    // 初始化day数组  
    for(i=0; i<m; i++) day[i]=0;  
    while(i < n)  
    {  
        sum += pm25[i];  
        if(pm25[i] >= 0 && pm25[i] <= 50 ) day[0]++;  
        else if(pm25[i] >= 51 && pm25[i] <= 100 ) day[1]++;  
        else if(pm25[i] >= 101 && pm25[i] <= 150 ) day[2]++;  
        else if(pm25[i] >= 151 && pm25[i] <= 200 ) day[3]++;  
        else if(pm25[i] >= 201 && pm25[i] <= 300 ) day[4]++;  
        else day[5]++;  
        i++;  
    }  
    return (float) sum/n;  
}
```

用数组做参数, 可获得函数计算结果的多值传递, return 仅返回一个值。

例: 数字排序 (主程序)

给定 n 个整数, 请计算每个整数各位数字和, 按各位数字和从大到小的顺序输出。

```
#include <stdio.h>

#define N 1000 // 估计的数组大小

int bitsSum(int a); // 计算整数a的各位之和

void sort(int a[], int n); // 从大到小排序

void output(int a[], int n); // 输出

int main()
{
    int i, n; // n是实际数组长度
    int num[N], sum[N]; // num表示N个整数, sum存放对应整数的各位数字的和
    scanf("%d", &n);
    for(i=0; i<n; i++)
    {
        scanf("%d", &num[i]);
        sum[i]=bitsSum(num[i]);
    }
    sort(sum, n); // 从大到小排序
    output(sum, n); // 输出
    return 0;
}
```

例: 数字排序 (子函数: 计算整数的各位之和, 输出)

// 计算整数a的各位之和

```
int bitsSum(int a)
```

```
{  
    int sum=0;  
    while(a)  
    {  
        sum += a%10;  
        a /= 10;  
    }  
    return sum;  
}
```

// 输出

```
void output(int a[], int n)
```

```
{  
    int i;  
    for(i=0;i<n;i++) printf("%d",a[i]);  
    printf("\n");  
}
```

例：数字排序 (排序子函数—方法 1: 冒泡排序)

// 从大到小排序，冒泡

```
void sort(int a[], int n)
{
    int i,j,flag,temp;
    for(j = 1; j <= n-1; j++) // 第j趟比较
    {
        flag=0;
        for(i = 0; i < n - j; i++) // 相邻两数比较
        {
            if (a[i] < a[i+1]) // 交换
            {
                temp = a[i]; a[i] = a[i+1]; a[i+1] = temp;
                flag=1;
            }
        }
        if(!flag) break;
    }
}
```

例：数字排序 (排序子函数—方法 2: 选择法排序)

从大到小**选择法排序**: 先将 n 个数中最大的数与 $a[0]$ 对换;

再将 $a[1] \sim a[n-1]$ 对换, \dots , 每比较一轮, 找出一个未经排序的数中最大的一个。共比较 $n-1$ 轮。

// 从大到小排序, 选择

```
void sort(int a[], int n)
{
    int i, j, k, temp;
    for(j = 0; j <= n-2; j++) // 第j趟比较
    {
        k=j; // 未经排序的数中, 最大元素的下标
        for(i = j+1; i < n; i++) // 未经排序的数从j+1开始
        {
            if (a[i] > a[k]) k=i;
        }
        if(k!=j) // 未经排序数中的最大元素与a[j]交换, a[j]及其以前元素是已经排好的数据
        {
            temp = a[j]; a[j] = a[k]; a[k] = temp;
        }
    }
}
```

例: 数字排序 (联动输出: 修改主程序)

给定 n 个整数, 请计算每个整数各位数字和, 按各位数字和从大到小的顺序输出。

要求联动输出: 整数 该整数的各位之和

```
#include <stdio.h>

#define N 1000 // 估计的数组大小

int bitsSum(int a); // 计算整数a的各位之和

void sort(int a[], int b[], int n); // a从大到小排序, b联动改变

void output(int a[], int b[], int n); // 输出a,b

int main()
{
    int i, n; // n是实际数组长度
    int num[N], sum[N]; // num表示N个整数, sum存放对应整数的各位数字的和
    scanf("%d", &n);
    for(i=0; i<n; i++)
    {
        scanf("%d", &num[i]);
        sum[i]=bitsSum(num[i]);
    }
    sort(sum, num, n); // sum从大到小排序, num联动改变
    output(num, sum, n); // 输出num, sum
    return 0;
}
```

例: 数字排序 (联动输出: 修改排序子函数)

```
// a从大到小排序, b联动改变
void sort(int a[], int b[],int n)
{
    int i,j,flag,temp;
    for(j = 1; j <= n-1; j++) // 第j趟比较
    {
        flag=0;
        for(i = 0; i < n - j; i++) // 相邻两数比较
        {
            if (a[i] < a[i+1]) // 同时交换a和b
            {
                temp = a[i]; a[i] = a[i+1]; a[i+1] = temp;
                temp = b[i]; b[i] = b[i+1]; b[i+1] = temp;
                flag=1;
            }
        }
        if(!flag) break;
    }
}
```

例: 数字排序 (联动输出: 修改输出子函数)

```
// 输出a,b  
void output(int a[], int b[], int n)  
{  
    int i;  
    for(i=0;i<n;i++) printf("%d_ %d", a[i], b[i]);  
    printf("\n");  
}
```


例：数字排序 (联动输出：修改主程序, 二维数组)

给定 n 个整数, 请计算每个整数各位数字和, 按各位数字和从大到小的顺序输出。

要求联动输出：整数 该整数的各位之和

```
#include <stdio.h>

#define N 1000 // 估计的数组大小

int bitsSum(int a); // 计算整数a的各位之和

void sort(int a[][2], int n); // 按a的第0列从大到小排序

void output(int a[][2], int n); // 输出a

int main()
{
    int i, n; // n是实际数组长度
    int num[N][2]; // 第0列表示整数, 第1列是该整数的各位数字的和
    scanf("%d", &n);
    for(i=0; i<n; i++)
    {
        scanf("%d", &num[i][0]);
        num[i][1]=bitsSum(num[i][0]);
    }
    sort(num, n); // sum从大到小排序, num联动改变
    output(num, n); // 输出num
    return 0;
}
```

例: 数字排序 (联动输出: 修改排序子函数, 二维数组)

```
// 按a的第0列从大到小排序
void sort(int a[][2],int n)
{
    int i,j,flag,temp;
    for(j = 1; j <= n-1; j++) // 第j趟比较
    {
        flag=0;
        for(i = 0; i < n - j; i++) // 相邻两数比较
        {
            if (a[i][0] < a[i+1][0]) // 同时交换a的第0列和第1列
            {
                temp = a[i][0]; a[i][0] = a[i+1][0]; a[i+1][0] = temp;
                temp = a[i][1]; a[i][1] = a[i+1][1]; a[i+1][1] = temp;
                flag=1;
            }
        }
        if(!flag) break;
    }
}
```

例: 数字排序 (联动输出: 修改输出子函数, 二维数组)

```
// 输出a
```

```
void output(int a[][2], int n)
{
    int i;
    for(i=0;i<n;i++) printf("%d_□%d",a[i][0],a[i][1]);
    printf("\n");
}
```

模块化程序设计的优点

综上,程序功能的变化,仅修改相应子函数即可,逻辑清晰。

因此,功能分解是进行复杂程序设计的有效手段。

欢迎批评指正！