4

模块化程序设计

- 本章在详细介绍函数的概念的基础上，阐述如何利用模块化的思想与方法，分析与分解系统、设计与实现功能独立、接口清晰的模块并组织成逻辑完整的中等规模程序，解决较为复杂的问题。
- 模块化程序设计或称模块化编程（modular programming），指的是将软件系统按照功能层层分解为若干独立的、可替换的、具有预定功能的模块，各模块之间通过接口（对输入与输出的描述）实现调用，互相协作解决问题。

例 4.1　用字符“*”作为树叶、用“#”作为树干画出一棵树

```
#include <stdio.h>
int main() {
    printf("    *\n");
    printf("   ***\n");
    printf("  *****\n");
    printf("*******\n");
    printf("    *\n");
    printf("   ***\n");
    printf("  *****\n");
    printf("*******\n");
    printf("    #\n");
    printf("    #\n");
    printf("    #\n");
    return 0;
}
```

- 精简程序代码
- 改善程序结构
- 增强程序通用性

精简程序代码

```
#include <stdio.h>
int main() {
    printf("    *\n");
    printf("   ***\n");
    printf("  *****\n");
    printf("*******\n");
    printf("    *\n");
    printf("   ***\n");
    printf("  *****\n");
    printf("*******\n");
    printf("    #\n");
    printf("    #\n");
    printf("    #\n");
    return 0;
}
```

重复的代码段可以写成函数

改善程序结构

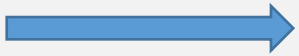
```
#include <stdio.h>
int main() {
    printf("    *\n");
    printf("   ***\n");
    printf("  *****\n");
    printf("*******\n");
    printf("    *\n");
    printf("   ***\n");
    printf("  *****\n");
    printf("*******\n");
    printf("    #\n");
    printf("    #\n");
    printf("    #\n");
    return 0;
}
```

```
void treetop() {
    printf("    *\n");
    printf("   ***\n");
    printf("  *****\n");
    printf("*******\n");
}
```

```
void treetrunk() {
printf("    #\n");
printf("    #\n");
printf("    #\n");
}
```

改善程序结构

```
#include <stdio.h>
int main() {
    printf("    *\n");
    printf("   ***\n");
    printf("  *****\n");
    printf(" *******\n");
    printf("    *\n");
    printf("   ***\n");
    printf("  *****\n");
    printf(" *******\n");
    printf("    #\n");
    printf("    #\n");
    printf("    #\n");
    return 0;
}
```



```
#include <stdio.h>
void treetop() {
    printf("    *\n");
    printf("   ***\n");
    printf("  *****\n");
    printf(" *******\n");
}
void treetrunk() {
    printf("    #\n");
    printf("    #\n");
    printf("    #\n");
}
int main() {
    treetop();
    treetop();
    treetrunk();
    return 0;
}
```

增加程序通用性

```
#include <stdio.h>
void treetop(char ch) {
printf("    %c\n",ch);
printf("  %c%c%c\n",ch,ch,ch);
printf(" %c%c%c%c%c\n",ch,ch,ch,ch,ch);
printf("%c%c%c%c%c%c%c\n",ch,ch,ch,ch,ch,ch,ch);
}
void treetrunk(char ch) {
    printf("    %c\n",ch);
    printf("    %c\n",ch);
    printf("    %c\n",ch);
}
int main() {
    char tree_leaf,tree_trunk;
    int tree_layers;
    printf("请输入一个字符表示树叶形状:");
    scanf("%c",&tree_leaf);
    printf("请输入一个字符表示树干形状:");
    scanf("%c",&tree_trunk);
    printf("请输入一个数字表示树的高度:");
    scanf("%d",&tree_layers);
    while(tree_layers--){
        treetop(tree_leaf);
    }
    treetrunk(tree_trunk);
    return 0;
}
```

通用的树叶函数

通用的树干函数

定义和输入相关参数

调用各函数生成最终结果

函数定义的一般形式与函数原型声明

一般形式

```
函数类型    函数名（形参类型说明表）
{
        函数的操作对象（数据）定义说明部分
        语句部分
}
```

合法标识符

函数返回值类型
缺省int型
无返回值void

函数体

```
int factorial(int n){    //计算n的阶乘
    int product,i;
    product=1;
    for(i=1;i<=n;i++)
        product*=i;
    return(product);
}
```

函数定义的一般形式与函数原型声明

函数声明

- 对被调用函数要求：
  - 必须是已存在的函数
  - 库函数：#include <*.h>
  - 用户自定义函数：函数类型说明
- 函数说明
  - 一般形式：函数类型 函数名(形参类型1 [形参名1],.....);
    或 函数类型 函数名(形参类型1，形参类型2，.....);
  - 作用：告诉编译系统函数类型、参数个数及类型，以便检验
  - 函数定义与函数说明不同
  - 函数说明位置：程序的数据说明部分（函数内或外）
  - 被调用函数定义出现在主调函数之前，可不作函数说明
  - 库函数声明包括在头文件（*.h）里，只需将头文件*.h作#include预处理即可。

函数定义的一般形式与函数原型声明

函数声明举例

```
#include <stdio.h>
int main()
{
    float add(float,float);
     //function declaration
    float a,b,c;
    scanf("%f,%f",&a,&b);
    c=add(a,b);
    printf("sum is %f",c);
    return 0;
}
float add(float x, float y)
{
    float z;
    z=x+y;
    return(z);
}
```

函数定义的一般形式与函数原型声明

函数声明举例

```
#include <stdio.h>
float add(float x, float y)
{
    float z;
    z=x+y;
    return(z);
}
int main()
{
    float a,b,c;
    scanf("%f,%f",&a,&b);
    c=add(a,b);
    printf("sum is %f",c);
    return 0;
}
```

被调函数出现在主调函数之前，可不做函数声明

函数定义的一般形式与函数原型声明

函数声明举例

```
#include <stdio.h>
int main()
{
    int a,b;
    int c;
    scanf("%d,%d",&a,&b);
    c=max(a,b);
    printf("Max is %d\n",c);
    return 0;
}
max(int x, int y)
{
    int z;
    z=x>y?x:y;
    return(z);
}
```

int型函数可不作函数声明

函数调用

调用形式：函数名(实参表);

说明：

- 实参与形参**个数相等**，**类型一致**，**按顺序一一对应**
- 实参表求值顺序，因系统而定

函数调用

```
#include<stdio.h>
main()
{
    int i=2,p1,p2;
    p1=f1(i,++i);
    p2=f2(i,i++);
    printf("p1=%d,p2=%d\n",p1,p2);
}
int f1(int a, int b)
{   int c;
    if(a>b)  c=1;
    else if(a==b)   c=0;
    else c=-1;
    return(c);
}
int f2(int a, int b)
{   int c;
    if(a>b)  c=1;
    else if(a==b)   c=0;
    else c=-1;
    return(c);
}
```

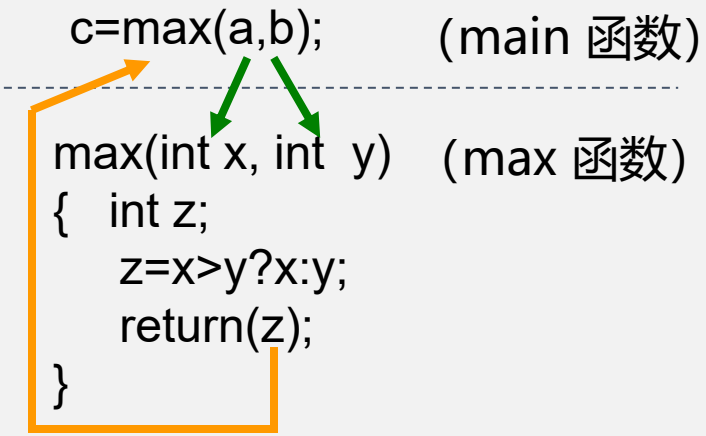
根据顺序不同会产生不同结果

函数参数及传递方式

★ 形参与实参

- ❖ 形式参数：定义函数时函数名后面括号中的变量名
- ❖ 实际参数：调用函数时函数名后面括号中的表达式

例 比较两个数并输出大者




```
int main()
{   int a,b,c;
    scanf("%d,%d",&a,&b);
    c=max(a,b);
    printf("Max is %d",c);
    return 0;
}
int max(int  x, int  y)
{   int z;
    z=x>y?x:y;
    return(z);
}
```

实参

形参

函数参数及传递方式

说明：

- 实参必须有确定的值，可以是**常量、变量或表达式**，但必须有**确定的值**
- 形参必须指定类型，各个形参之间要用**逗号**隔开
- 形参与实参**类型一致，个数相同**
- 若形参与实参类型不一致，自动按形参类型转换————**函数调用转换**
- 形参**不能在定义的同时进行初始化**
- 形参在函数**被调用前不占内存**，函数调用时为形参分配内存，**调用结束，内存释放**

函数参数及传递方式

例 4.2 字符串加密程序的实现

```
#include<stdio.h>
#define STEPS 3
#define LEN_INFO 16
char encrypt(char origin){
    char i='A'+(origin+STEPS-'A')%26;
    return i;
}
int main(){
    int i;
    char c,s[LEN_INFO]="SIX AM,RUN WEST";
    printf( "origin is:\t%s\n",s);
    printf( "encrypts:\t");
    for(i=0;i<LEN_INFO-1;i++){
        c=encrypt(s[i]);
        putchar(c);
    }
    printf("\n");
    return 0;
}
```

函数参数及传递方式

值传递方式

- 函数调用时,为形参分配单元,并将实参的值**复制**到形参中；调用结束，形参单元**被释放**，实参单元仍保留并维持原值

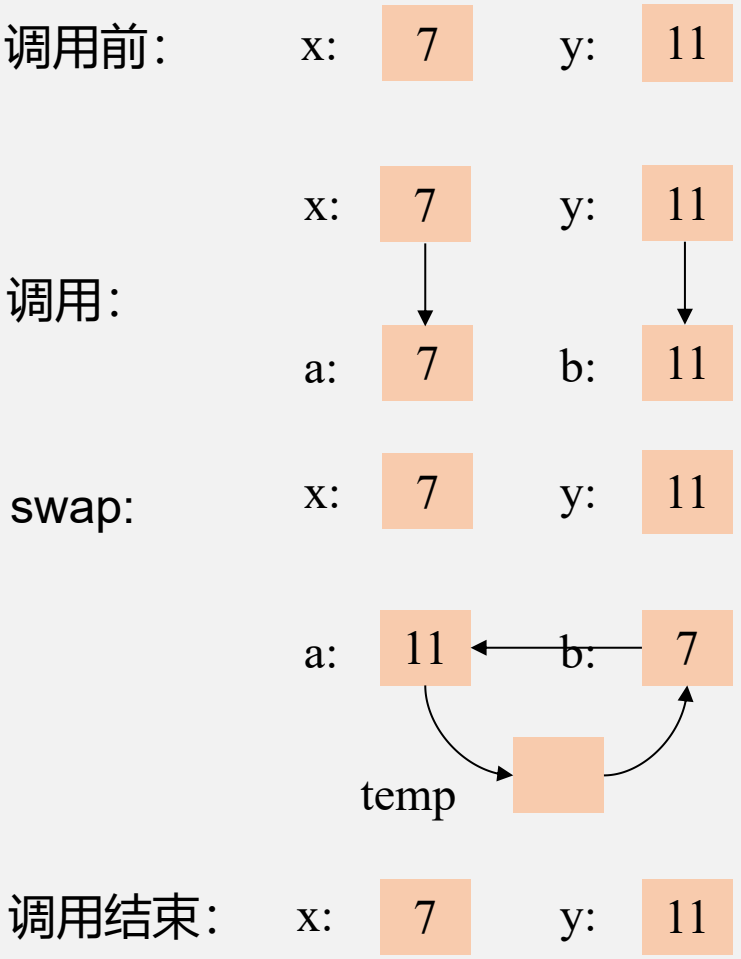
特点:

- 形参与实参占用**不同**的内存单元
- **“单向”**传递

函数参数及传递方式

例 4.3 变量交换函数

```
#include <stdio.h>
void swap(int a,int b)
{
    int temp;
    temp=a; a=b; b=temp;
}
int main()
{
    int x=7,y=11;
    printf("x=%d,y=%d\n",x,y);
    printf("swapped:\n");
    swap(x,y);
    printf("x=%d,y=%d\n",x,y);
    return 0;
}
```



函数参数及传递方式

例 4.3 变量交换函数

```
#include <stdio.h>
int main()
{
    void swap(int*);
    int a[2]={1,2};
    printf("a[0]=%d,a[1]=%d\n",a[0],a[1]);
    printf("swapped:\n");
    swap(a);
    printf("a[0]=%d,a[1]=%d\n",a[0],a[1]);
    return 0;
}
void swap(int b[])
{
    int temp;
    temp=b[0]; b[0]=b[1]; b[1]=temp;
}
```

函数参数及传递方式

地址传递方式

- 函数调用时，将数据的**存储地址**作为参数传递给形参

特点：

- 形参与实参占用**同样**的存储单元（此时形参本质是“**指针**”变量）
- **“双向”**传递
- 实参是地址常量或变量，形参是指针变量

函数参数及传递方式

函数的返回值

- 返回语句

形式：

```
return（表达式）；
return 表达式;
return;
```

功能：使程序控制从被调用函数返回到调用函数中，同时把返值带给调用函数.

说明：

- 函数中可有多个return语句
- 若无return语句，遇 } 时，自动返回调用函数
- 若函数类型与return语句中表达式值的类型不一致，按前者为准，自动转换 — **函数调用转换**
- void型函数

函数的模块化概念

```
#include<stdio.h>
int dingjiudian(int, float, double, struct guke);
int zuche(...);
int dingpiao(...);
int fukuan(...);
int main()
{
    ...
    dingjiudian(...);
    dingpiao(...);
    ...
}
int dingjiudian(int a, float b, double c, struct guke d)
{
    ...
    zuche(...);
    ...
    fukuan(...);
}
int zuche(...)
{
    ...
    fukuan(...);
}
int dingpiao(...)
{
    ...
    fukuan(...);
}
int fukuan(...)
{
    ...
}
```

数组作为函数参数

例 4.4 数组名作为实参

```
#include <stdio.h>
float average(float array[]);
int main() {
    float score[10]={100,56,83,94,67,90,83,77,76,58};
    printf(" 平均成绩为:%5.1f\n ",average(score));
    return 0;
}

float average(float array[]) {
    int i;
    float aver , sum=array[0];
    for(i=1;i<10;i++) sum=sum+array[i];
    aver=sum/10;
    return(aver);
}
```

数组名作为实参

形参是什么?

? 函数接口的问题

数组作为函数参数

```
#include<stdio.h>
float average(float array[ ], int n);
int main() {
    float score_1[10]={67,87,90,93,74,65,78,88,99,78};
    float score_2[5]={98,76,87,83,95};
    printf("aver...%5.1f\n", average(score_1,10));
    printf("aver...%5.1f\n", average(score_2,5));
    return 0;
}
float average(float array[ ], int n) {
    int i;
    float aver , sum=array[0];
    for(i=1; i<n; i++) sum=sum+array[i];
    aver=sum/n;
    return(aver);
}
```

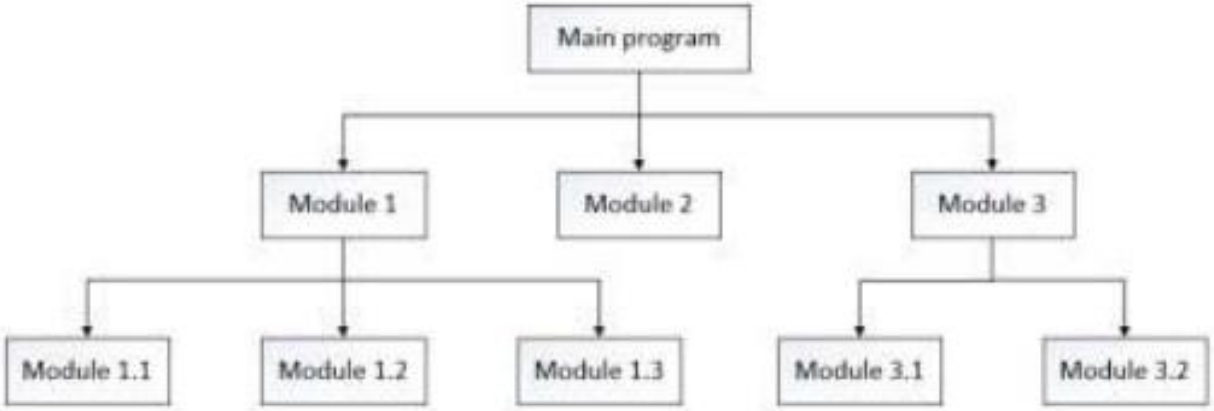
增加了数组长度作为实参

数组作为函数参数

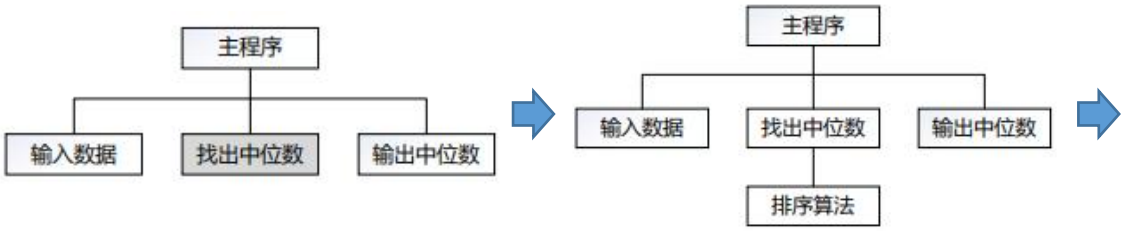
```
#include<stdio.h>
int max_value(int ary[][4], int, int);
int main() {
    int a[3][4]={{1,3,5,7},{2,4,6,8},{15,17,34,12}};
    printf("max is %d\n",max_value(a,3,4));
    return 0;
}
int max_value(int array[][4],int m,int n) {
int i,j,max;
max=array[0][0];
for(i=0; i<m; i++)
for(j=0;j<n;j++)
if(array[i][j]>max) max=array[i][j];
return(max);
}
```

二维数组需要增加两个参数

自顶向下设计


Main program
Module 1
Module 2
Module 3
Module 1.1
Module 1.2
Module 1.3
Module 3.1
Module 3.2

自顶向下设计-寻找中位数



- 输入数据函数：

  int input_data(int a[], int n);

- 找出中位数函数：

  double find_median(int a[], int n);

- 输出数据函数：

  void output_data(double m);

- 排序函数：

  void sort(int a[], int n);

自顶向下设计-寻找中位数

- 主函数：

```
#include <stdio.h>
int input_data(int a[], int n);
double find_median(int a[], int n);
void output_data(double m);
void sort(int a[], int n);
int main() {
    int a[10], n;
    double med;
    n=input_data(a, 10); //输入数据
    med=find_median(a, n); //找出中位数
    output_data(med); //输出数据
    return 0;
}
```

- 输入数据函数：

```
int input_data(int a[], int n){
    int i;
    for(i=0; i<n; i++) {
    a[i]=i+2;
    printf("%d ", a[i]);
    }
    return n;
}
```

- 找出中位数函数：

```
double find_median(int a[], int n) {
    double med;
    sort(a,n); //调用排序函数
    if(n%2 == 0) med = (a[n/2-1] + a[n/2])/2.0;
    else med = a[n/2];
    return med;
}
```

- 输出数据函数：

```
void output_data(double m){
    printf("中位数是%.1f", m);
}
```

- 排序函数：

```
void sort(int a[], int n) {
//先空着，吃完饭回来再写
}
```

变量的作用域与生存期

变量是对程序中数据的存储空间的抽象

- 变量的属性
  - 数据类型：变量所持有的数据的性质（操作属性）
  - 存储属性
    - 存储器类型：寄存器、静态存储区、动态存储区
    - **生存期**：变量在某一时刻存在-------**静态变量**与**动态变量**
    - **作用域**：变量在某区域内有效-------**局部变量**与**全局变量**
- 变量的存储类型
  - auto ----- 自动型
  - register ----- 寄存器型
  - static ----- 静态型
  - extern ----- 外部型
- 变量定义格式: [存储类型] 数据类型 变量表;

局部变量与全局变量

局部变量（内部变量）

定义：**在函数内定义，只在本函数内有效**

说明：

- main中定义的变量只在main中有效
- 不同函数中同名变量，占不同内存单元
- 形参属于局部变量
- 可定义在复合语句中有效的变量
- 局部变量可用存储类型：**auto　register　static**　（默认为auto）

局部变量与全局变量

```
float  f1(int a)
{  int b,c;
    .......
}
```

a,b,c有效

```
char f2(int x,int y)
{   int i,j;
    ......
}
```

x,y,i,j有效

```
main()
{  int m,n;
   .......
}
```

m,n有效

局部变量与全局变量

例 4.5 不同函数中同名变量

```
int main()
{
    int a,b;
    a=3;
    b=4;
    printf("main:a=%d,b=%d\n",a,b);
    sub();
    printf("main:a=%d,b=%d\n",a,b);
    return 0;
}
sub(){
    int a,b;
    a=6;
    b=7;
    printf("sub:a=%d,b=%d\n",a,b);
}
```

局部变量与全局变量

全局变量---外部变量

定义：**在函数外定义，可为本文件所有函数共用**

说明：

- 有效范围：从**定义变量的位置开始到本源文件结束**，及有**extern**说明的其它源文件
- 外部变量说明： extern 数据类型 变量表;
- 外部变量定义与外部变量说明不同

应尽量少使用全局变量：

- **全局变量在程序全部执行过程中占用存储单元**
- **降低了函数的通用性、可靠性，可移植性**
- **降低程序清晰性，容易出错**

局部变量与全局变量

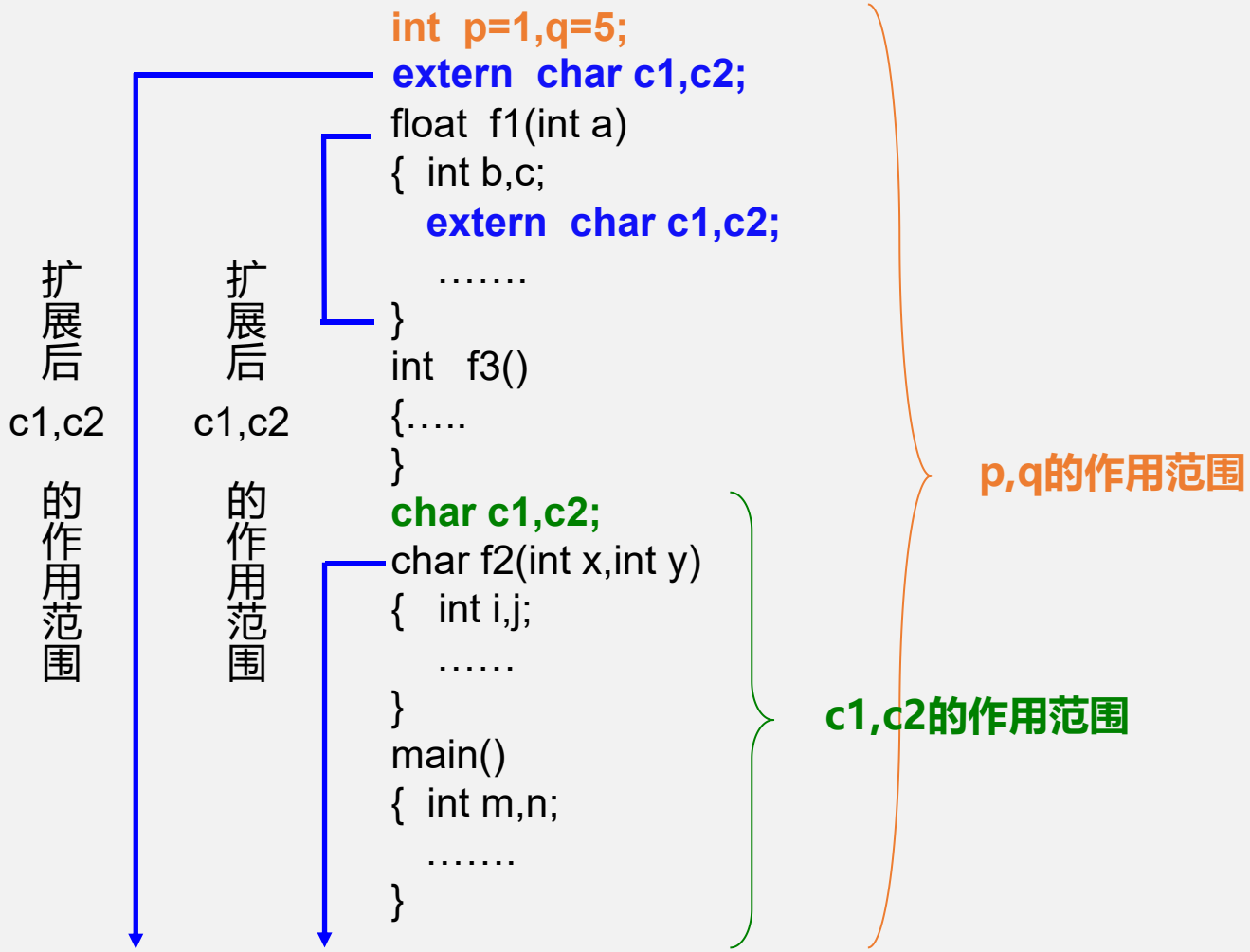
```
float max,min;
float average(float  array[],  int  n)
{
    int i;   float  sum=array[0];
    max=min=array[0];
    for(i=1;i<n;i++)
    {
        if(array[i]>max)  max=array[i];
        else if(array[i]<min)  min=array[i];
        sum+=array[i];
    }
    return(sum/n);
}
int main()
{
    int i;
    float ave,score[10];
    ave=average(score,10);
    printf("max=%6.2f\nmin=%6.2f\n
            average=%6.2f\n",max,min,ave);
    return 0;
}
```

max

min

作用域

局部变量与全局变量


int p=1,q=5;
extern char c1,c2;
float f1(int a)
{ int b,c;
extern char c1,c2;
.....
}
int f3()
{.....
}
char c1,c2;
char f2(int x,int y)
{ int i,j;
.....
}
main()
{ int m,n;
.....
}
扩展后c1,c2的作用范围
扩展后c1,c2的作用范围
p,q的作用范围
c1,c2的作用范围

局部变量与全局变量

例 4.6 全局变量与局部变量

```
int a=3,b=5;
max(int  a, int b)
{   int c;
    c=a>b?a:b;
    return(c);
}
main()
{
    int a=8;
    printf("max=%d",max(a,b));
}
```

运行结果：max=8

局部变量与全局变量

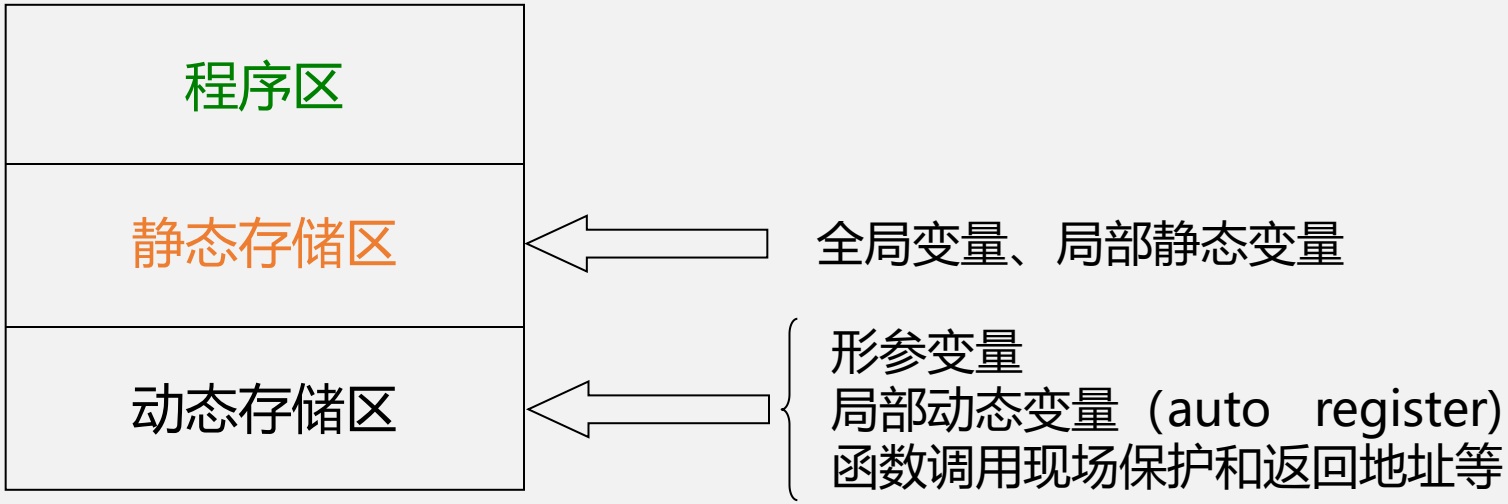
例 4.7 外部变量的副作用

```
#include<stdio.h>
int  i;
int main()
{
    void  prt();
    for(i=0;i<5;i++)
        prt();
    return 0;
}
void  prt()
{   for(i=0;i<5;i++)
            printf("%c",'*');
     printf("\n");
}
```

运行结果：*****

动态变量与静态变量

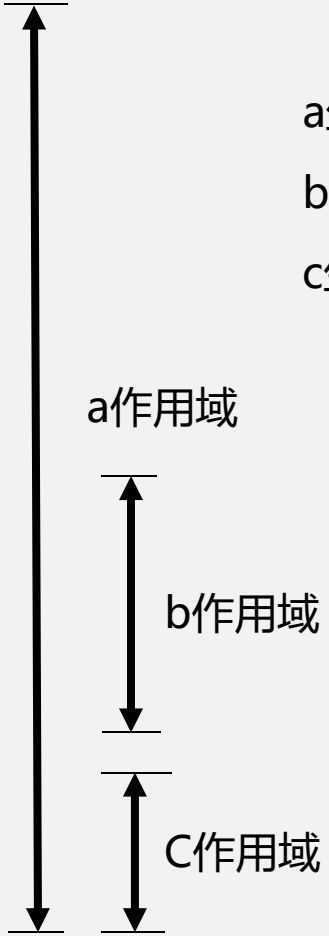
- 存储方式
  - 静态存储：程序运行期间分配固定存储空间
  - 动态存储：程序运行期间根据需要动态分配存储空间
- 内存用户区
- 生存期
  - 静态变量:从程序开始执行到程序结束
  - 动态变量:从包含该变量定义的函数开始执行至函数执行结束

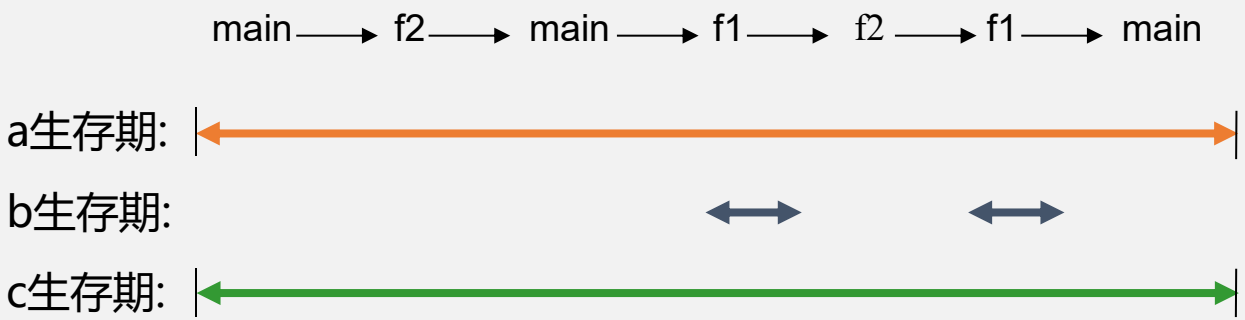


动态变量与静态变量

例 4.8 a、b、c的作用域和生存期

```
int a;
main( )
{    .....
     .....
       f2;
     .....
       f1;
     .....
}
f1( )
{   auto int b;
     ......
      f2;
     .....
}
f2( )
{  static int c;
     ......
}
```





动态变量与静态变量

例 4.9 同名变量

```
#include<stdio.h>
int main()
{    int x=1;
     void  prt(void);
      {
           int x=3;
           prt();
           printf("2nd x=%d\n",x);
      }
      printf("1st x=%d\n",x);
      return 0;
}
void  prt(void)
{     int x=5;
      printf("3th  x=%d\n",x);
}
```

运行结果：

3th x=5

2nd x=3

1st x=1

动态变量与静态变量

例 4.10 局部静态变量值具有可继承性

```
int main()
{
    void  increment(void);
    increment();
    increment();
    increment();
    return 0;
}
void  increment(void)
{   int x=0;
     x++;
     printf("%d\n",x);
}
```

运行结果：
1
1
1

```
int main()
{
    void  increment(void);
    increment();
    increment();
    increment();
    return 0;
}
void  increment(void)
{   static int x=0;
     x++;
     printf("%d\n",x);
}
```

运行结果：
1
2
3

动态变量与静态变量

例 4.11 变量的寿命与可见性

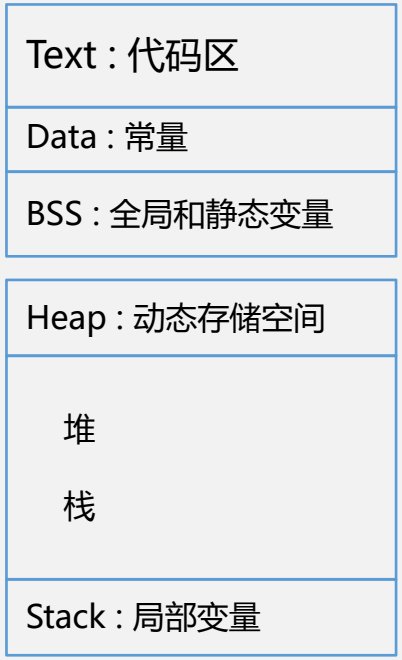
```
#include <stdio.h>
int i=1;
main()
{    static int a;
     register int b=-10;
     int c=0;
     printf("-----MAIN------\n");
     printf("i:%d a:%d b:%d c:%d\n",i,a,b,c);
     c=c+8;
     other();
     printf("-----MAIN------\n");
     printf("i:%d a:%d b:%d c:%d\n",i,a,b,c);
     i=i+10;
     other();
}
other()
{    static int a=2;
     static int b;
     int c=10;
     a=a+2;    i=i+32;   c=c+5;
     printf("-----OTHER------\n");
     printf("i:%d a:%d b:%d c:%d\n",i,a,b,c);
     b=a;
}
```

```
-------MAIN------
i:1  a:0  b:-10  c:0
------OTHER------
i:33  a:4  b:0  c:15
-------MAIN-----
i:33  a:0  b:-10  c:8
-------OTHER-------
i:75  a:6  b:4  c:15
```

动态变量与静态变量

例 4.12 程序存储空间

```
#include <stdio.h>
double glb;
int max(int x,int y) {
    static st;
    st+=x;
    printf("x 的地址%p，st 的地址%p\n",&x,&st);
    return st>y?st:y;
}
int main()
{
    int i;
    char str[8]="USTC";
    printf("glb 的地址%p，i 的地址%p\n",&glb,&i);
    printf("str 的地址%p，\"USTC\"的地址%p\n",str,"USTC");
    for(i=1;i<3;i++)
    printf("%d\n",max(i,i*i));
    printf("main 的地址%p，max 的地址%p\n",main,max);
    return 0;
}
```

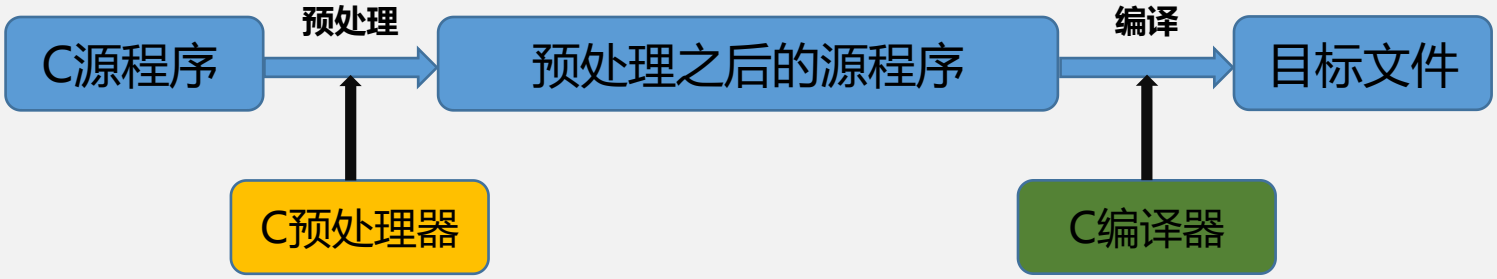


内部函数与外部函数

- 函数本质上是全局的，因为一个函数要被另外的函数调用，但是也可以指定函数不能被其他文件调用，根据函数能否被其他源文件调用，将函数区分为内部函数和外部函数。
- 外部函数　extern int fun (int a, int b)
  - 外部函数fun可以为其他文件调用。
  - C语言规定，如果在定义函数时省略extern，则默认为外部函数。
- 内部函数　static int fun(int a,int b)
  - 如果一个函数只能被本文件中其他函数所调用，它称为内部函数。在定义内部函数时，在函数名和函数类型的前面加static。
  - 使用内部函数，可以使函数只局限于所在文件，如果在不同的文件中有同名的内部函数,互不干扰。这样不同的人可以分别编写不同的函数，而不必担心所用函数是否会与其他文件中函数同名，通常把只能由同一文件使用的函数和外部变量放在一个文件中，在它们前面都static使之局部化，其他文件不能引用。

文件包含

作用：对源程序编译之前做一些处理,生成扩展C源程序



种类

- 宏定义 #define
- 文件包含 #include
- 条件编译 #ifdef #endif

格式：

- ''#''开头
- 占单独书写行
- 语句尾不加分号

文件包含

- 一个#include 命令只能包含一个文件，如果有多个文件要包含，则要分别使用多个#include 命令实现
- 文件包含允许嵌套，即一个被包含的文件中还可以再包含其他文件
- 文件包含中的文件通常是以.h 作为扩展名的头文件，但也可以是任何其他文本文件，例如.c 的源程序文件、.txt 文本文件等
- 由于被含文件的全部内容都会原原本本地复制展开在当前文件源代码中，因此，所包含文件的内容必须符合 C 语言的语法，否则编译时会报错
- 同一个头文件可以被多次包含，效果和一次包含相同，因为头文件在代码层面有防止重复引入的机制

文件包含

例 4.13 自定义的头文件

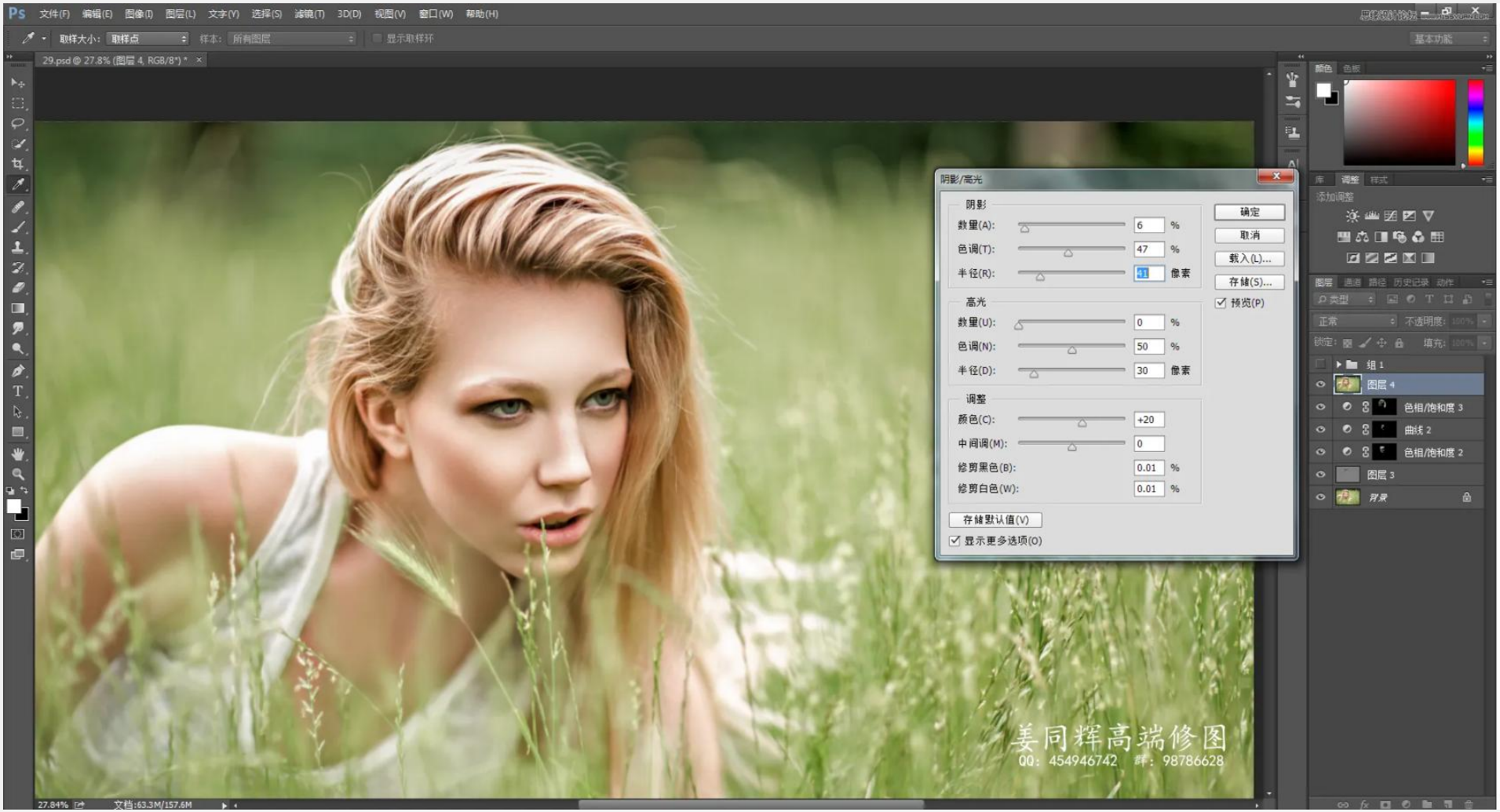
```
/* my.c */
int sum(int m, int n) {
    int i, sum = 0;
    for (i = m; i<= n; i++) sum += i;
    return sum;
}
```

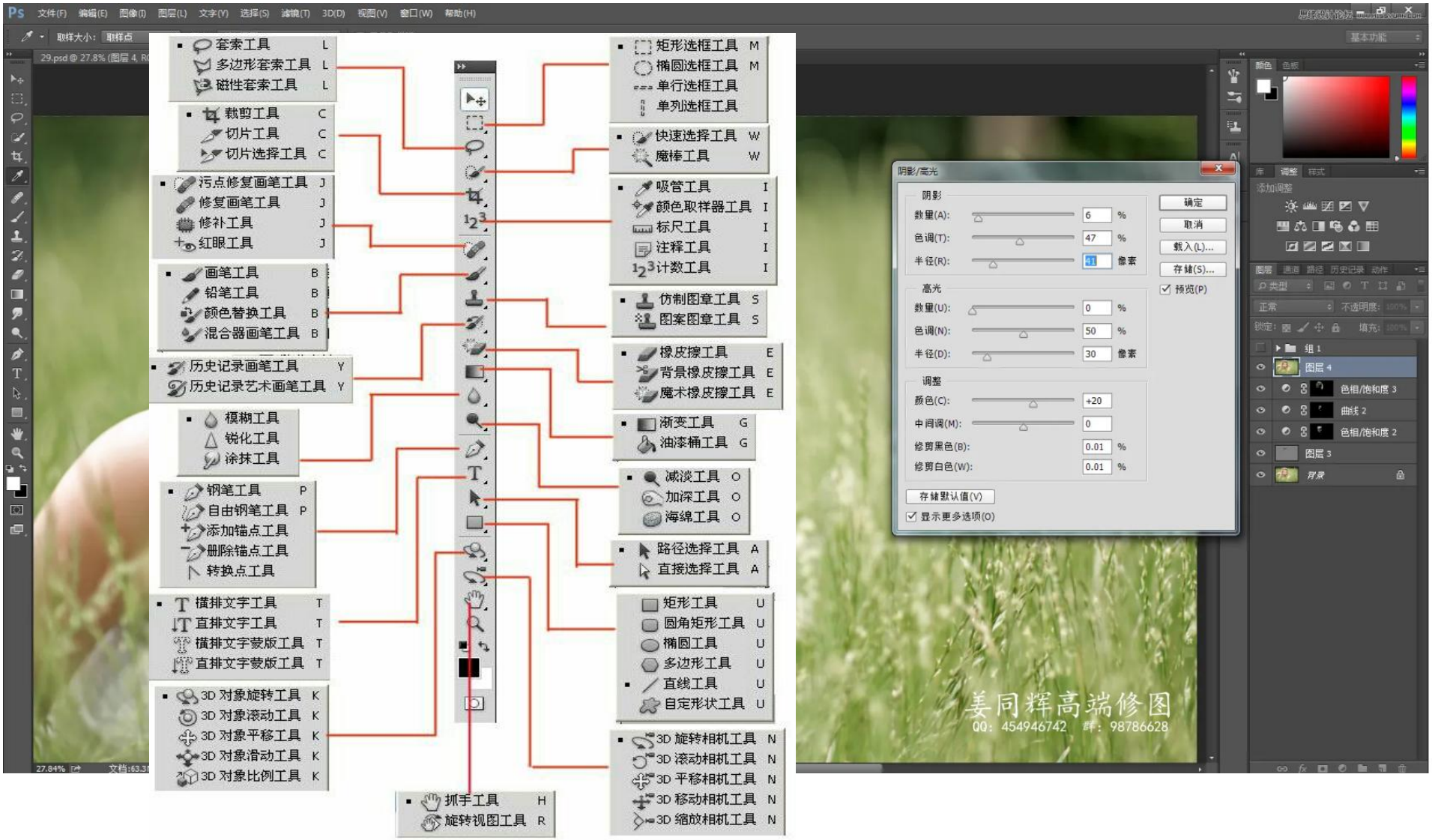
```
/* my.h */
int sum(int m, int n);
```

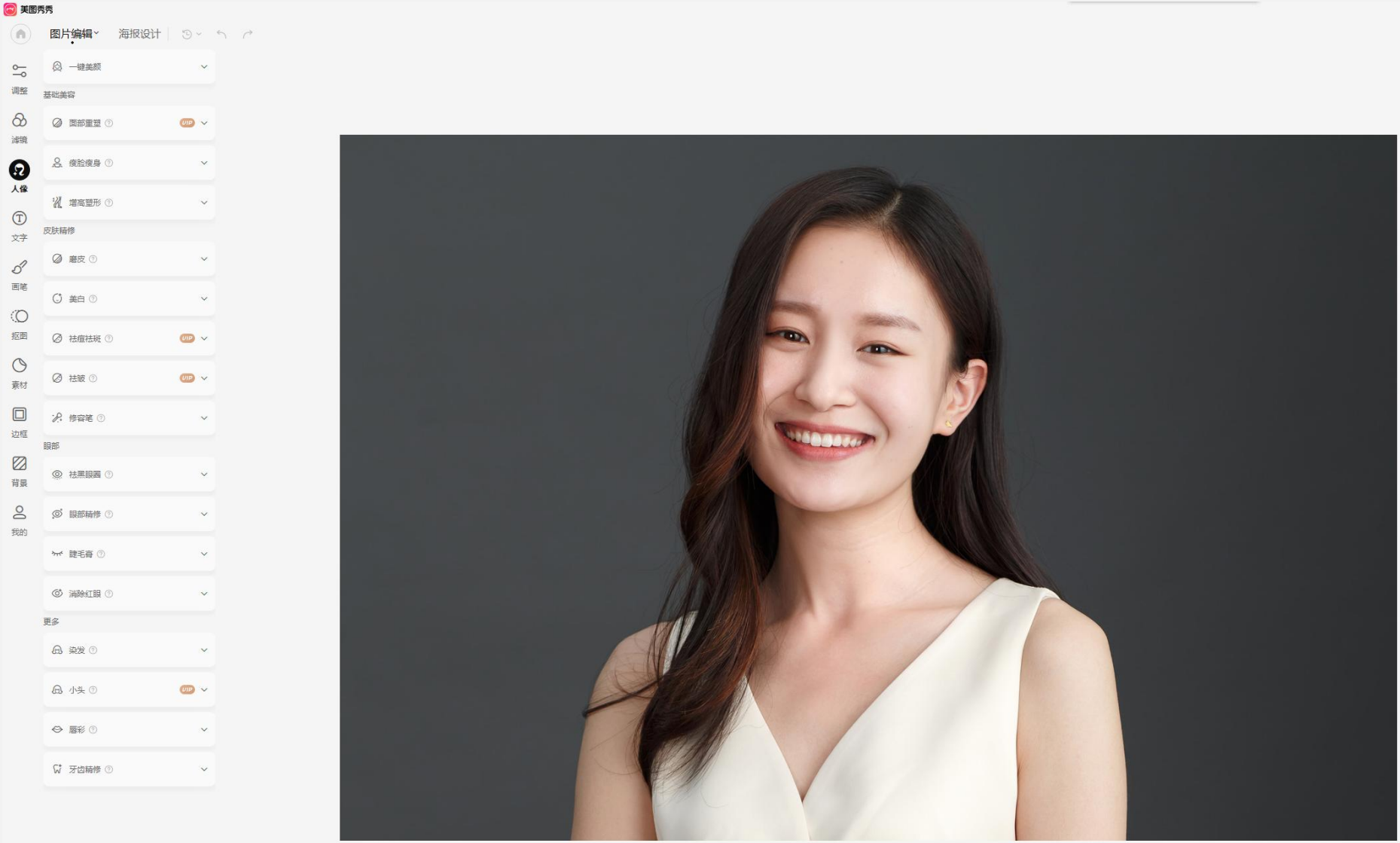
```
/* main.c */
#include <stdio.h>
#include "my.c"
#include "my.h"
int main() {
    printf("%d\n", sum(1, 100));
    return 0;
}
```

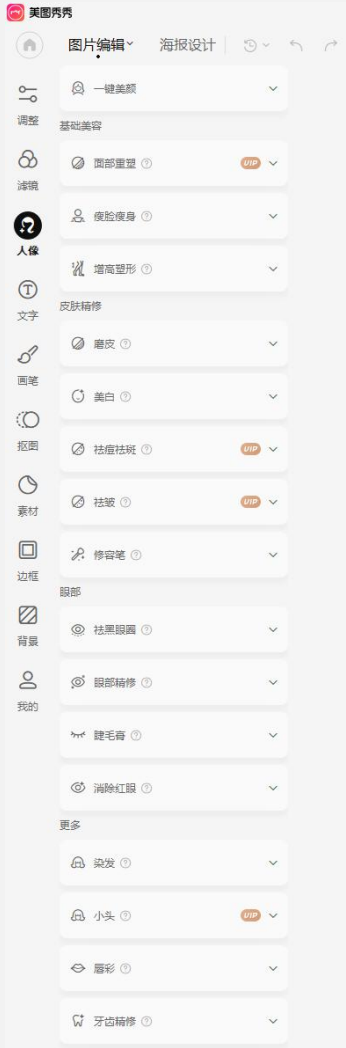
库函数

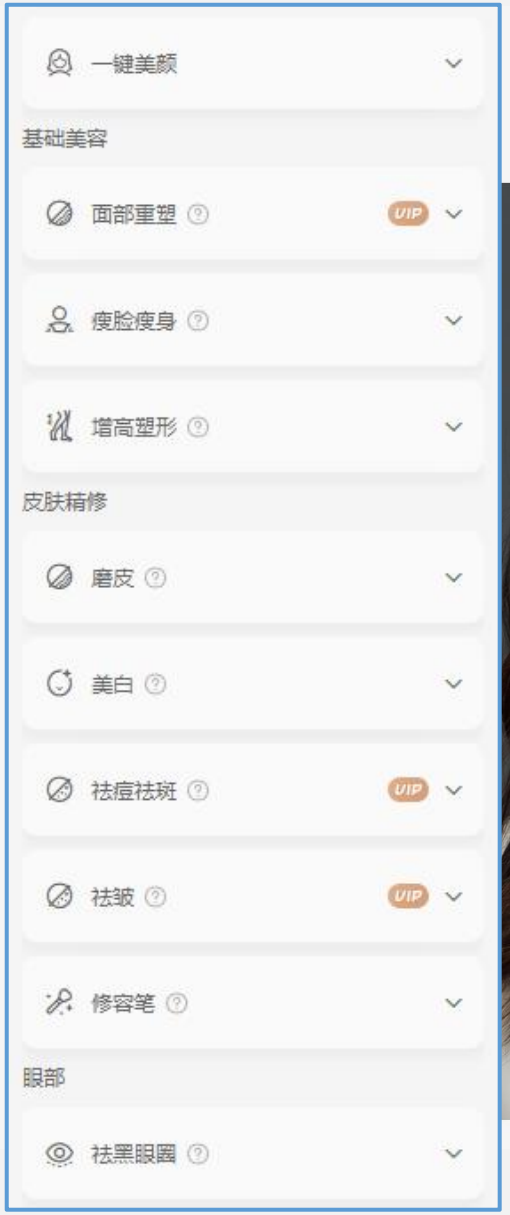
- C编译系统为方便用户使用而提供的已经编写好的公共函数
- ANSI C常见的库函数：
  - 输入输出函数<stdio.h>
  - 数学函数<math.h>
  - 字符串函数<string.h>
  - 字符函数 <ctype.h>
  - 功能库函数<stdlib.h>
  - 时间日期函数<time.h>
- 在C语言中要使用库函数，应当使用编译预处理命令“#include”把与该库函数相关的头文件包含进来


阴影/高光
阴影
数量(A): 6 %
色调(T): 47 %
半径(R): 41 像素
高光
数量(U): 0 %
色调(N): 50 %
半径(D): 30 像素
调整
颜色(C): +20
中间调(M): 0
修剪黑色(B): 0.01 %
修剪白色(W): 0.01 %
确定
取消
载入(L)...
存储(S)...
预览(P)
存储默认值(V)
显示更多选项(O)
姜同辉高端修图
QQ：454946742 群：98786628


套索工具 L
多边形套索工具 L
磁性套索工具 L
裁剪工具 C
切片工具 C
切片选择工具 C
污点修复画笔工具 J
修复画笔工具 J
修补工具 J
红眼工具 J
画笔工具 B
铅笔工具 B
颜色替换工具 B
混合器画笔工具 B
历史记录画笔工具 Y
历史记录艺术画笔工具 Y
模糊工具
锐化工具
涂抹工具
钢笔工具 P
自由钢笔工具 P
添加锚点工具
删除锚点工具
转换点工具
横排文字工具 T
直排文字工具 T
横排文字蒙版工具 T
直排文字蒙版工具 T
3D 对象旋转工具 K
3D 对象滚动工具 K
3D 对象平移工具 K
3D 对象滑动工具 K
3D 对象比例工具 K
抓手工具 H
旋转视图工具 R
矩形选框工具 M
椭圆选框工具 M
单行选框工具
单列选框工具
快速选择工具 W
魔棒工具 W
吸管工具 I
颜色取样器工具 I
标尺工具 I
注释工具 I
计数工具 I
仿制图章工具 S
图案图章工具 S
橡皮擦工具 E
背景橡皮擦工具 E
魔术橡皮擦工具 E
渐变工具 G
油漆桶工具 G
减淡工具 O
加深工具 O
海绵工具 O
路径选择工具 A
直接选择工具 A
矩形工具 U
圆角矩形工具 U
椭圆工具 U
多边形工具 U
直线工具 U
自定形状工具 U
3D 旋转相机工具 N
3D 滚动相机工具 N
3D 平移相机工具 N
3D 移动相机工具 N
3D 缩放相机工具 N
阴影/高光
阴影
数量(A): 6 %
色调(T): 47 %
半径(R): 41 像素
高光
数量(U): 0 %
色调(N): 50 %
半径(D): 30 像素
调整
颜色(C): +20
中间调(M): 0
修剪黑色(B): 0.01 %
修剪白色(W): 0.01 %
确定
取消
载入(L)...
存储(S)...
预览(P)
存储默认值(V)
显示更多选项(O)
姜同辉高端修图
QQ：454946742 群：98786628


美图秀秀
图片编辑
海报设计
一键美颜
基础美容
面部重塑
瘦脸瘦身
增高塑形
皮肤精修
磨皮
美白
祛痘祛斑
祛皱
修容笔
眼部
祛黑眼圈
眼部精修
睫毛膏
消除红眼
更多
染发
小头
唇彩
牙齿精修
调整
滤镜
人像
文字
画笔
抠图
素材
边框
背景
我的


美图秀秀
图片编辑
海报设计
一键美颜
基础美容
面部重塑
瘦脸瘦身
增高塑形
皮肤精修
磨皮
美白
祛痘祛斑
祛皱
修容笔
眼部
祛黑眼圈
眼部精修
睫毛膏
消除红眼
更多
染发
小头
唇彩
牙齿精修
调整
滤镜
人像
文字
画笔
抠图
素材
边框
背景
我的


一键美颜
基础美容
面部重塑
VIP
瘦脸瘦身
增高塑形
皮肤精修
磨皮
美白
祛痘祛斑
VIP
祛皱
VIP
修容笔
眼部
祛黑眼圈

库函数

printf（）和scanf（）

返回值：printf()函数若执行成功则返回输出的字符数，输出出错则返回负值。

返回值：执行成功，该函数返回成功匹配和赋值的个数。如果到达文件末尾或发生读错误，则返回 EOF。

例 4.14 printf

```
#include<stdio.h>
int main()
{
    int num=12345;
    printf("num=%d,printf=%d",num,printf("%d\n",num));
    return 0;
}
```

库函数

常用的字符串处理函数

包含在**string.h**和**stdio.h**头文件里

字符串输出函数**puts**

格式：puts(字符数组数组名)

功能：向显示器输出字符串（

说明：字符数组必须以 '\0'

字符串输入函数**gets**

格式：gets(字符数组数组名)

功能：从键盘输入一以回车结

并自动加 '\0'

说明：输入串长度应小于字符

例
```
#include <stdio.h>
main( )
{
    char string[80];
    printf( "Input a string:" );
    gets(string);
    puts(string);
}
```
输入：How are you?

输出：How are you ?

库函数

常用的字符串处理函数

字符串长度函数**strlen**

格式：strlen(字符数组数组名)

功能：计算字符串长度

返值：返回字符串实际长度，不包括 ‘\0’ 在内

字符串连接函数**strcat**

格式：strcat(字符数组1,字符数组2)

功能：把字符数组2连到字符数组1后面

返值：返回字符数组1的首地址

说明：①字符数组1必须足够大

②连接前,两串均以 ‘\0’ 结束;连接后,串1的 ‘\0’ 取消,

新串最后加 ‘\0’

库函数

常用的字符串处理函数

字符串比较函数**strcmp**

格式：strcmp(字符串1,字符串2)

功能：比较两个字符串

比较规则：对两串从左向右逐个字符比较（ASCII码），

直到遇到不同字符或 ‘\0’ 为止

返值：返回int型整数，a. 若字符串1< 字符串2， 返回负整数

b. 若字符串1> 字符串2， 返回正整数

c. 若字符串1== 字符串2，返回零

说明：**字符串比较不能用“==”,必须用strcmp**

库函数

常用的字符串处理函数

字符串拷贝函数**strcpy**

格式：strcpy(字符数组1,字符数组2/字符串常量)

功能：将字符数组2中的字符串，拷贝到字符数组1中去

返值：返回字符数组1的首地址

说明：①字符数组1一般比字符数组2长

②拷贝时‘\0’一同拷贝

③不能使用赋值语句为一个字符数组赋值

库函数

常用的字符串处理函数

字符串拷贝函数**strncpy**

格式：strcnpy(字符数组1,字符数组2,非负整型变量n)

功能：将字符串2中的n个字符拷贝到字符数组1中去

返值：返回字符数组1的首地址

说明：①参数n必须小于等于字符数组1的长度

②满足条件①的情况下，如果n>字符数组2长度，则将字符数组2全部复制到字符数组1中（包括'\0'）.

库函数

例 4.15 对一个文本中某个单词进行统计.

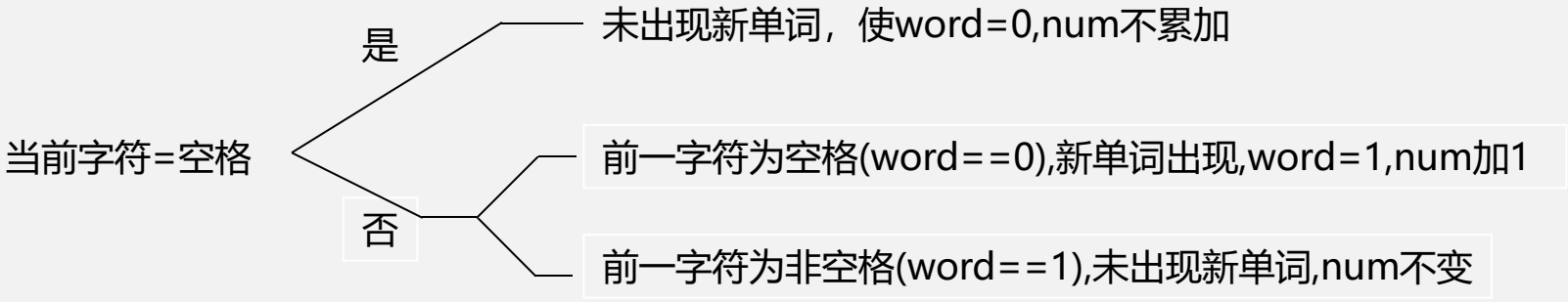
```
#include<stdio.h>
#include<string.h>
int main()
{
    int i=0,sum=0,length;
    char word[20];
    char text[]="10 plus 5 is 15,10 minus 5 is 5,10 multiplied by 5 is 50,10
divided by 5 is 2.";
    char temp[20]={0};
    printf("please input the word whick you want to count:");
    gets(word);
    length=strlen(word);
    while(text[i]!='\0')
    {
        strncpy(temp,text+i,length);
        if(strcmp(word,temp)==0)
        {
            sum++;
            i+=length;
        }
        i++;
    }
    printf("the word \"%s\" appears %d times in the text.\n",word,sum);
    return 0;
}
```

求目标单词的长度

根据目标单词利用strncpy
逐个向后匹配查找

库函数

另一种统计单词的方法




```
#include <stdio.h>
int main()
{
    char string[81];
    int i,num=0,word=0;
    char c;
    gets(string);
    for(i=0;(c=string[i])!='\0';i++)
      if(c==' ') word=0;
      else if(word==0) {word=1;num++;}
    printf("There are %d words in the line\n",num);
    return 0;
}
```

库函数

随机数库函数

```
int  rand( void ) ;
```

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int a = rand();
    printf("%d\n",a);
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int i;
    for (i=5; i>0; i--)
        printf("%d\t", rand());
    return 0;
}
```

随机？

伪随机？

库函数

随机数库函数

```
int  rand( void ) ;      void  srand( unsigned int seed );
time_t  time( time_t *seconds );
```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    int i;
    srand((unsigned)time(NULL));
    for (i=5; i>0; i--)
        printf("%d\t", rand());
    return 0;
}
```

用time函数做“种子”

递归

函数定义不可嵌套，但可以嵌

递归：函数自己调用自己

递归

```
#include <stdio.h>
void My_Small_Carp_Recursion(int depth)
{
    printf("抱着");
    if (depth==0) printf("我的小鲤鱼");
    else My_Small_Carp_Recursion(--depth);
    printf("的我");
}
int main()
{
    int layers;
    printf("层数:");
    scanf("%d",&layers);
    printf("吓得我抱起了\n");
    My_Small_Carp_Recursion(layers);
    putchar('\n');
    return 0;
}
```

递归的产生（自己调用自己）

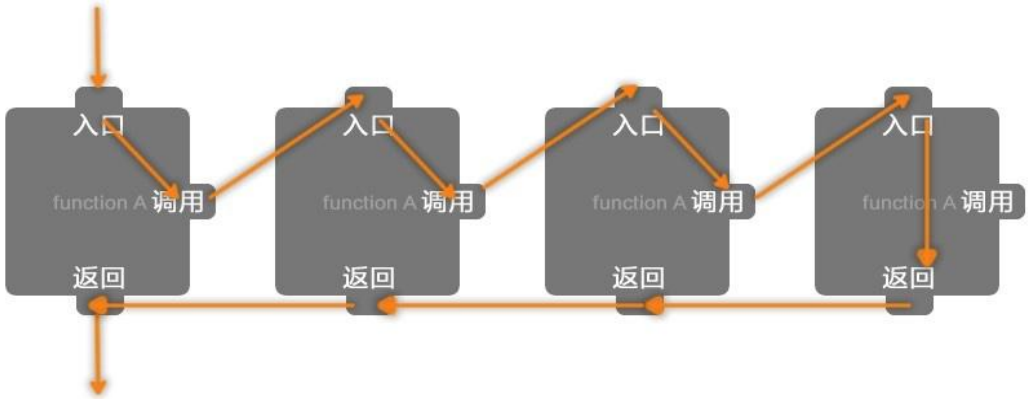
递归

- 适合递归的情况：

  （1） 问题具有一个或多个递归终止的条件；

  （2） 问题具备能持续缩小问题规模的参数　f(n) — f(n-1)

- C编译系统对递归函数的自调用次数没有限制
- 每调用函数一次，在内存堆栈区分配空间，用于存放函数变量、返回值等信息，所以递归次数过多，可能引起堆栈溢出



递归

例 4.16 采用递归求阶乘。 $n!=\begin{cases}1 & (n=0,1)\\ n\cdot(n-1)! & (n>1)\end{cases}$

```
#include <stdio.h>
int fac(int n)
{   int f;
    if(n<0)  printf("n<0,data error!");
    else if(n==0||n==1)  f=1;
    else f=fac(n-1)*n;
    return(f);
}
int main()
{   int n, y;
    printf("Input a integer number:");
    scanf("%d",&n);
    y=fac(n);
    printf("%d! =%15d",n,y);
    return 0;
}
```

递归的产生（自己调用自己）

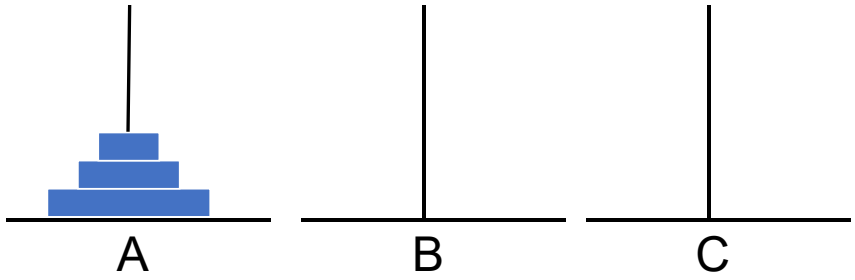
递归

例 4.17 汉诺塔问题

有三个柱和n个大小各不相同的盘子，开始时，所有盘子以塔状叠放在柱A上,要求按一定规则，将柱A上的所有盘子借助于柱B移动到柱C上。移动规则如下：

（1）一次只能移动一个盘子。

（2）任何时候不能把盘子放在比它小的盘子的上面。



递归

例 4.17 汉诺塔问题

汉诺塔问题递归过程的函数描述

有n个盘子的汉诺塔问题的函数：hanoi(n,'A','B','C');

当n=1时, 直接从 A 移到 C, 问题结束。移动过程用如下函数描述：

```
move('A','C');
```

若n>1时, 则必须经过如下三个步骤：

第一步： 按照移动规则，把A上面的 n-1 个盘子，移到B，此时C为中间柱。

```
hanoi(n-1,'A','C','B');
```

第二步: 将A上仅有的一只盘子 (当前最大的一只) 直接移到柱B上。

```
move('A','C');
```

第三步: 用第一步所述方法, 将B柱上的n-1个盘子移到C柱上，此时A为中间柱。

```
hanoi(n-1,'B','A','C');
```

递归

例 4.17 汉诺塔问题

```
#include<stdio.h>
void move(char getone, char putone)
{
    printf("%c--->%c\n",getone,putone);
}
void hanoi(int n,char one,char two,char three)
{
    if(n==1)  move(one,three);
    else
    {
        hanoi(n-1,one,three,two);
        move(one,three);
        hanoi(n-1,two,one,three);
    }
}
int main()
{   int m;
    printf("Input the number of disks:");
    scanf("%d",&m);
    while(m){
        printf("The steps to moving %3d disks:\n",m);
        hanoi(m,'A','B','C');
        printf("Input the number of disks:");
        scanf("%d",&m);
    }
    return 0;
}
```

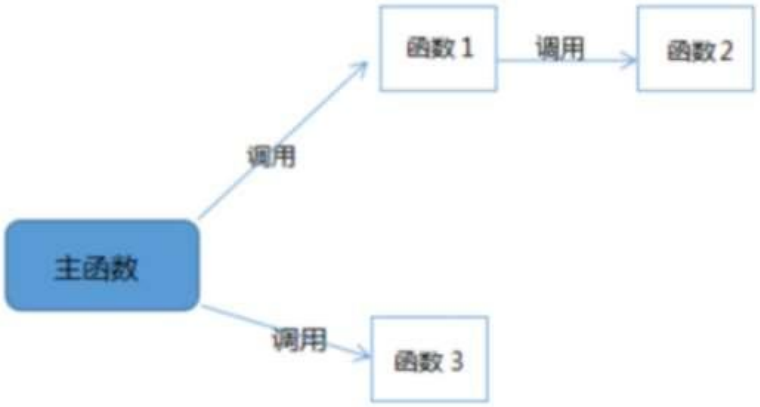
输出移动步骤的函数

第一步和第三步均为递归

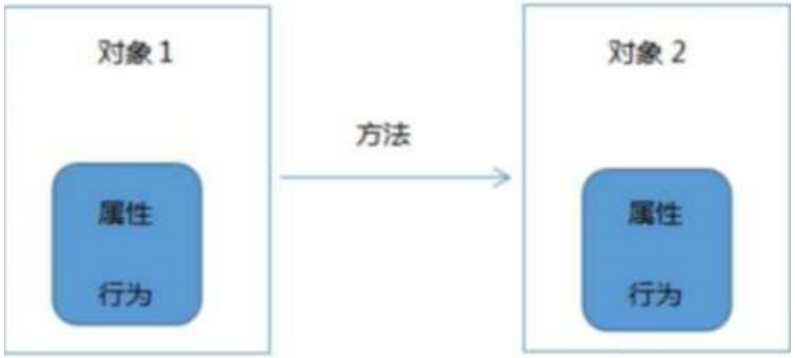
任何计算机程序都是对特定数据进行特定处理的过程。当利用计算机解决问题时，不外乎要做两件事情：

- 将问题中要处理的数据表示出来。这可以借助编程语言提供的基本数据类型、复杂类型构造手段以及更高级的逻辑数据结构等来实现。
- 设计处理这些数据的算法过程，并利用编程语言提供的各种语句编制成可以一步一步执行的操作序列

- **面向过程**



- **面向对象**



数据的排序和查找

- **生成随机数**：利用随机数库函数产生随机的数据并存储在数组中。
- **数据排序**：按照几种常用的排序算法思想对数组进行排序，分升序与降序两种情况进行练习。
- **数据查询**：分别按照普通查找、二分查找方法在数组中查找给定的数据。

数据的排序和查找

选择排序

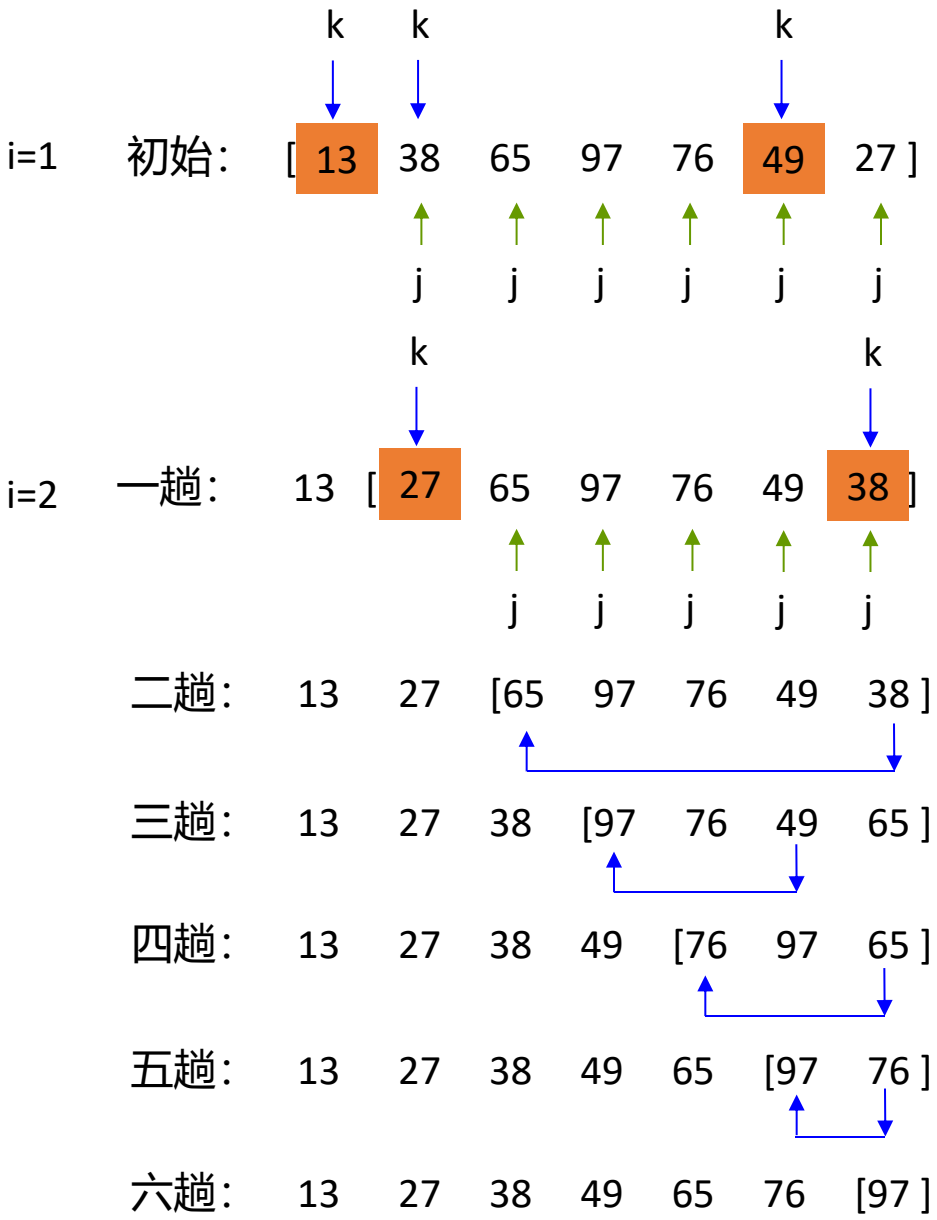
（1）首先通过n-1次比较，从n个数中找出最小的， 将它与第一个数交换—第一趟选择排序，结果最小的数被安置在第一个元素位置上。

（2）再通过n-2次比较，从剩余的n-1个数中找出关键字次小的记录，将它与第二个数交换—第二趟选择排序。

（3）重复上述过程，共经过n-1趟排序后，排序结束。

数据的排序和查找

选择排序



数据的排序和查找

选择排序

```
void SelectSort(int a[],int num)
{
    int i,j,k,temp;
    for(i=0;i<num-1;i++)
    {  k=i;
       for(j=i+1;j<=num-1;j++)
          if(a[j]<a[k])  k=j;
       if(i!=k)
       {temp=a[i];a[i]=a[k];a[k]=temp;}
    }
}
```

每一轮确定k的位置

每轮将k对应的元素
与该轮第一个数交换

数据的排序和查找

折半查找

```
int bin_search(int a[],int n,int key)
{
    int low,high,mid;
    low=0;
    high=n-1;
    while(low<=high)
    {
        mid =(low + high)/2;
        if(a[mid]==key)  return mid;
        if(a[mid]<key)  low=mid+1;
        if(a[mid]>key)  high=mid-1;
    }
    return -1;
}
```

确定中间位后根据情况判断下轮是查找前半区还是后半区

没找到

模块化程序设计的主要目标和步骤

主要目标

(1) 使程序结构更清晰，提高程序的可读性，便于交流；

(2) 降低代码间的耦合，便于独立开发；

(3) 提高代码的重用性，减少程序中的重复代码，使程序更简洁；

(4) 控制局部代码规模，便于编码、调试与维护

主要步骤

(1) 分析问题，明确程序的总体任务；

(2) 对任务进行逐层分解，直至大小合适的模块；

(3) 设计模块间的接口与模块的算法流程；

(4) 编码实现模块并通过调用组成程序；

(5) 测试模块与程序。

能力要求总结

(1) 较为全面地掌握模块化程序设计的思想与方法，初步了解软件工程过程；

(2) 养成先设计再实现的良好编程习惯，注重程序规范性、完善性；

(3) 能熟练定义与调用函数，有一定的算法优化能力；

(4) 能熟练编写基本的排序算法与字符串处理函数；

(5) 理解递归的逻辑，能编写简单的递归程序。

(6) 能编写包含多个函数的单文件程序，解决常见的数据处理与计算问题。