High cohesion, low coupling.

C语言程序设计

C Programming

9

函数

理论课程

厦門大學

XIAMEN UNIVERSITY

信息学院 黄 炜

（特色化示范性软件学院） 博士/副教授

School of Informatics Dr. Wei Huang

内容要点

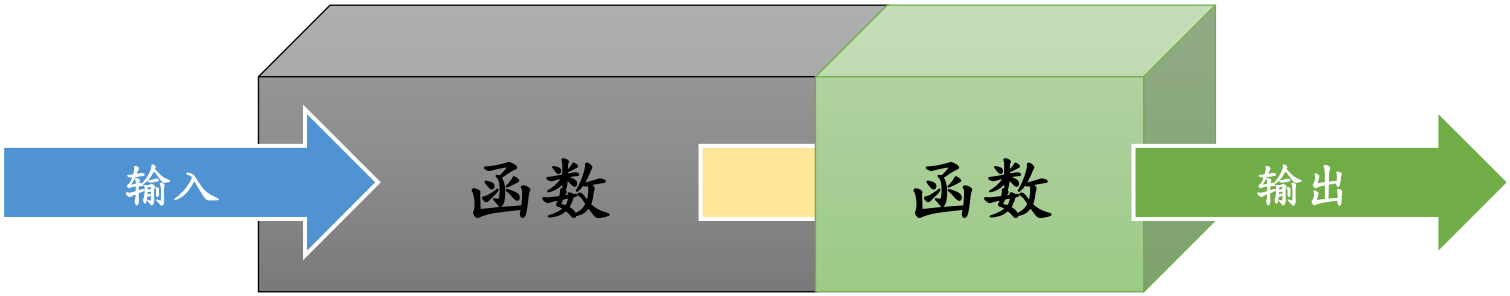
- 函数的组成
  - 声明、定义、参量和返回值
- 函数的调用
  - 调用的过程
  - 实际参数与形式参量
- 递归函数
- 函数的设计原则
- 函数的复用与分发

目录

函数（function）

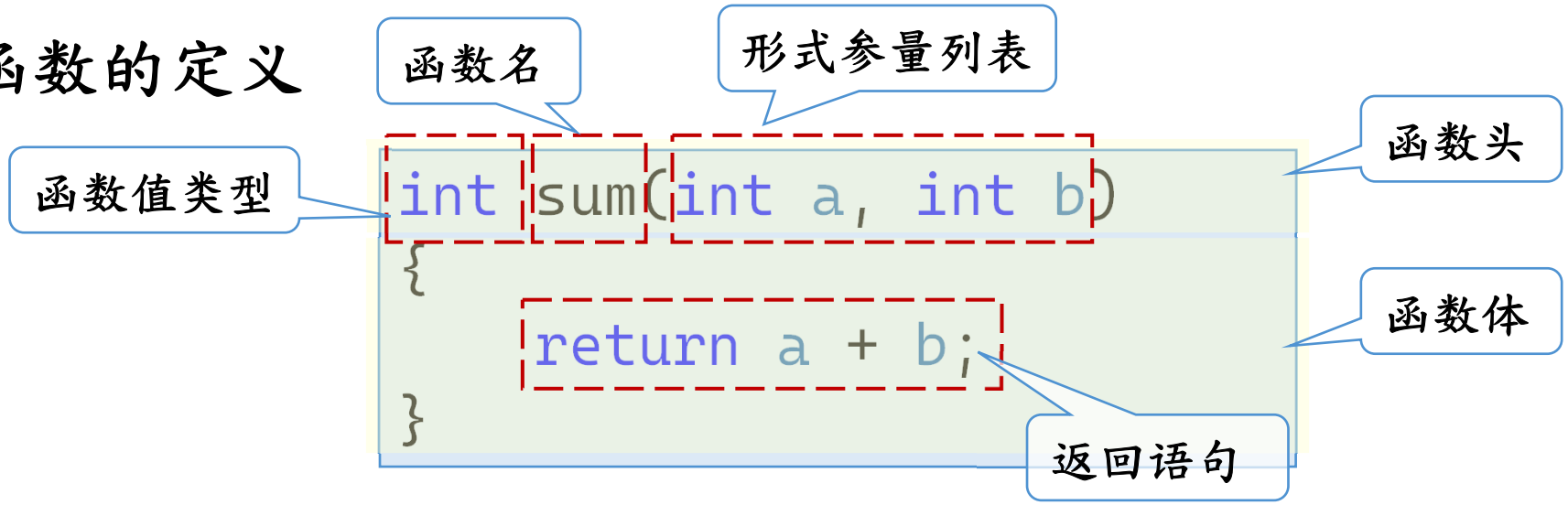
- 代数课程的函数是两个数集的多对一映射
  - 零个或多个输入、一个输出、过程
- 计算机程序的函数是用于完成特定任务的程序代码的自包含单元
- 程序员喜欢将函数视为黑盒子
  - 有利于将精力投入程序整体设计而不必兼顾细节

例如：printf()执行了将常量或变量格式化显示在界面的功能



函数的组成

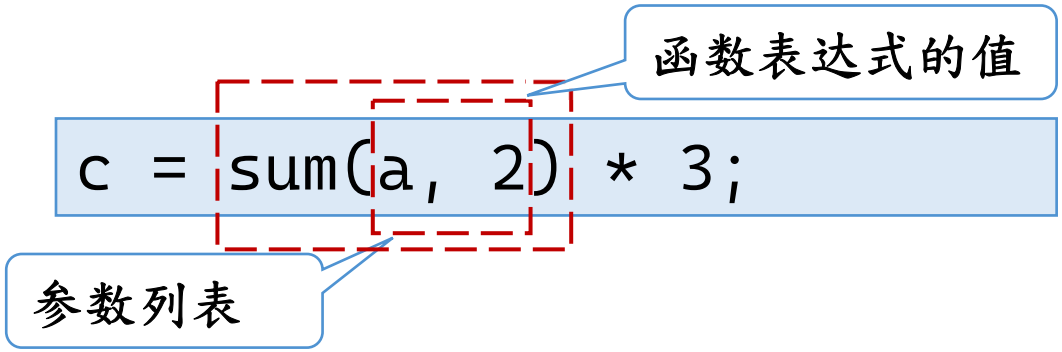
- 函数的定义



- 函数的声明

```
int sum(int a, int b);
```

- 函数的调用



```c
/* lethead1.c */
#include <stdio.h>
#define NAME "GIGATHINK, INC.
#define ADDRESS "101 Megabuck
#define PLACE "Megapolis, CA

int main(void)
{
    int count;
    for (count = 1; count <= 40; count++)
        putchar('*');
    putchar('\n');
    printf("%s\n", NAME);
    printf("%s\n", ADDRESS);
    printf("%s\n", PLACE);
    for (count = 1; count <= 40; count++)
        putchar('*');
    putchar('\n');
    return 0;
}
```

```
****************************************
GIGATHINK, INC.
101 Megabuck Plaza
Megapolis, CA 94904
****************************************
```

为了输出美观，上下的星号数量是一致的，应用宏对40命名。

```c
#define WIDTH 40
```

此二处的代码一致，说明存在冗余不够内聚，有经验的程序员将它们封装为函数

```c
/* lethead1.c */
#include <stdio.h>
#define NAME "GIGATHINK, INC."
#define ADDRESS "101 Megabuck Plaza"
#define PLACE "Megapolis, CA 94904"
#define WIDTH 40
void starbar(void);  /* prototype the function */

int main(void) {
    starbar();
    printf("%s\n", NAME);
    printf("%s\n", ADDRESS);
    printf("%s\n", PLACE);
    starbar();          /* use the function        */

    return 0;
}
```

函数的声明，函数应：先声明、有定义、再调用

函数的调用

```
void starbar(void)    /* define the function    */
{
    int count;

    for (count = 1; count <= WIDTH; count++)
        putchar('*');
    putchar('\n');
}
```

函数的定义
这是个没有参量的函数

```
****************************************
GIGATHINK, INC.
101 Megabuck Plaza
Megapolis, CA 94904
****************************************
```

函数的原型声明

- 格式 `<类型> <函数名>(<类型1> [参数名1] [, <类型2> [参数名2] ]...);`
  - 示例：参量名可以省略

```
void dibs(int, int, int);
```

```
void dibs(int x, int y, int z);
```

  - 旧版ANSI C允许只声明返回类型，勿用

```
int imax();        /* old-style declaration */
```

- 函数应先声明，有定义，后使用
  - 函数在调用前定义，可以省略声明
- 函数名在声明范围内是唯一的，而且不能和变量重名
  - 返回值、参量列表必须一致

函数的头部

- 格式 <类型> <函数名>(<类型1> [参数名1] [, <类型2> [参数名2]]...)
- 示例
  - 左侧为已废弃的形式（勿用），右上方正确，右下方错误

```
void dibs(int, char)
int num;
char ch;
```

```
void dibs(int x, int y, int z)
```

```
void dibs(int x, y, z)
```

函数返回语句

- 格式 `return <表达式>;`
  - 函数类型为void时可以用“return;”或不写返回语句
- 函数类型即返回值类型
  - 不存在参量类型相同而返回值类型不同的两个同名函数
  - 返回值由表达式得到，类型应与函数类型一致
- 返回语句可以不止一个
  - 但在运行中遇到第一个return语句终止

```c
/* lesser.c -- finds the lesser of two evils */
#include <stdio.h>
int imin(int, int);

int main(void)
{
    int evil1, evil2;

    printf("Enter a pair of integers (q to quit):\n");
    while (scanf("%d %d", &evil1, &evil2) == 2)
    {
        printf("The lesser of %d and %d is %d.\n",
               evil1, evil2, imin(evil1,evil2));
        printf("Enter a pair of integers (q to quit):\n");
    }
    printf("Bye.\n");

    return 0;
}
```

函数的声明可以不写参量名

函数构成表达式，作为一个函数的参数

```
int imin(int n,int m)
{
    int min;

    if (n < m)
        min = n;
    else
        min = m;

    return min;
}
```

```
Enter a pair of integers (q to quit):
34 24↲
The lesser of 34 and 24 is 24.
Enter a pair of integers (q to quit):
19 19↲
The lesser of 19 and 19 is 19.
Enter a pair of integers (q to quit):
q↲
Bye.
```

目录

程序的流程

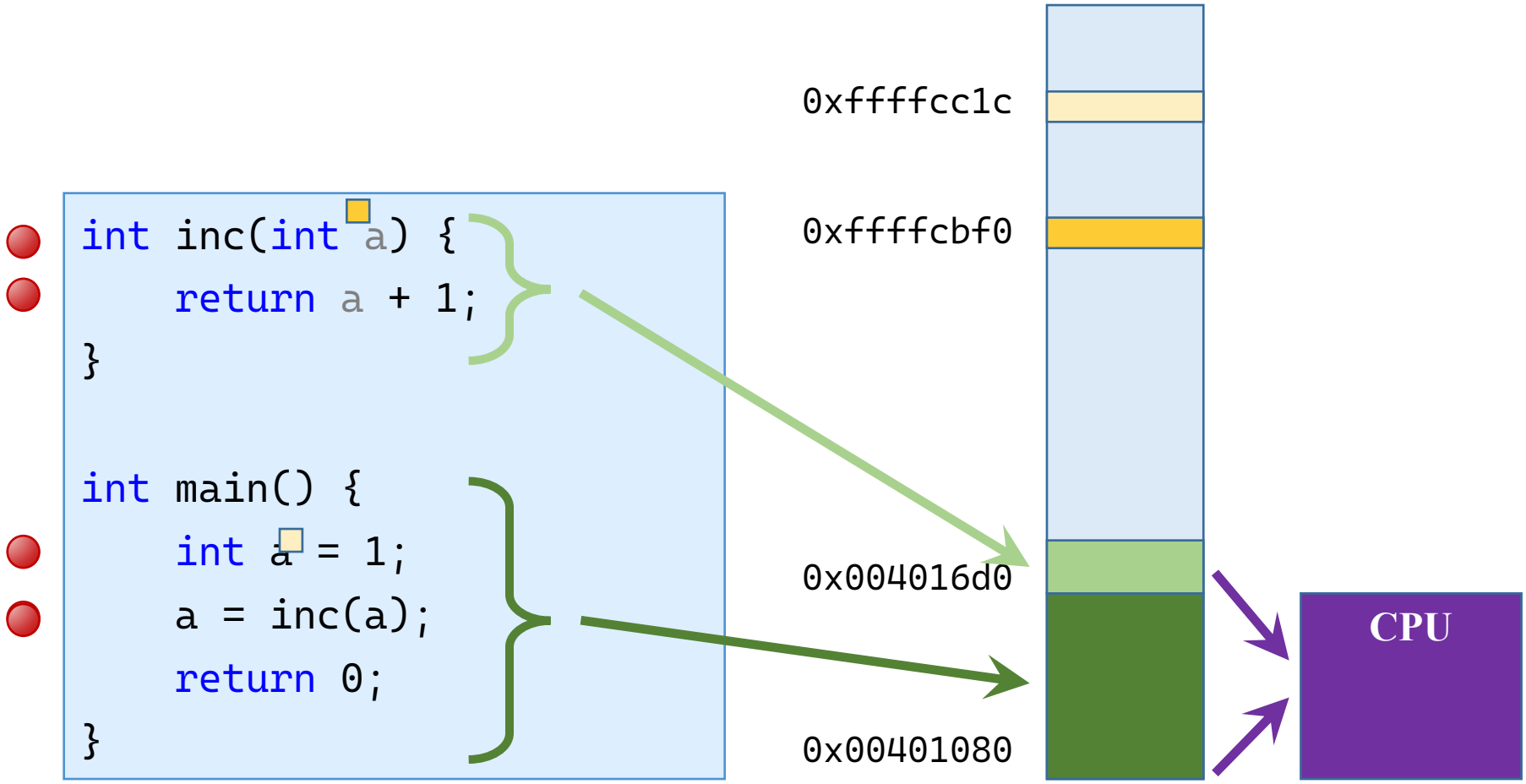
- 函数调用时，程序流程发生改变
  - 执行函数的第一条语句
- 函数调用中，程序按顺序执行
  - 执行到首次遇到返回语句时退出
  - 如未遇到返回语句且返回类型为void时，执行到函数最后一条语句退出
- 函数退出后，控制权交还给调用前的语句

函数调用时的操作

- 函数调用时
  - 计算所有参数列表的值
    - 如果列表中的参数有副作用，结果不可控
  - 为每个参量开辟内存空间
  - 将每个参数的值以此赋值给对应的形式参量
- 函数退出时（当函数类型不为void）
  - 计算返回语句中表达式的值，作为函数的值传给外部
  - 销毁函数内声明的变量（含形式参量，除静态变量）

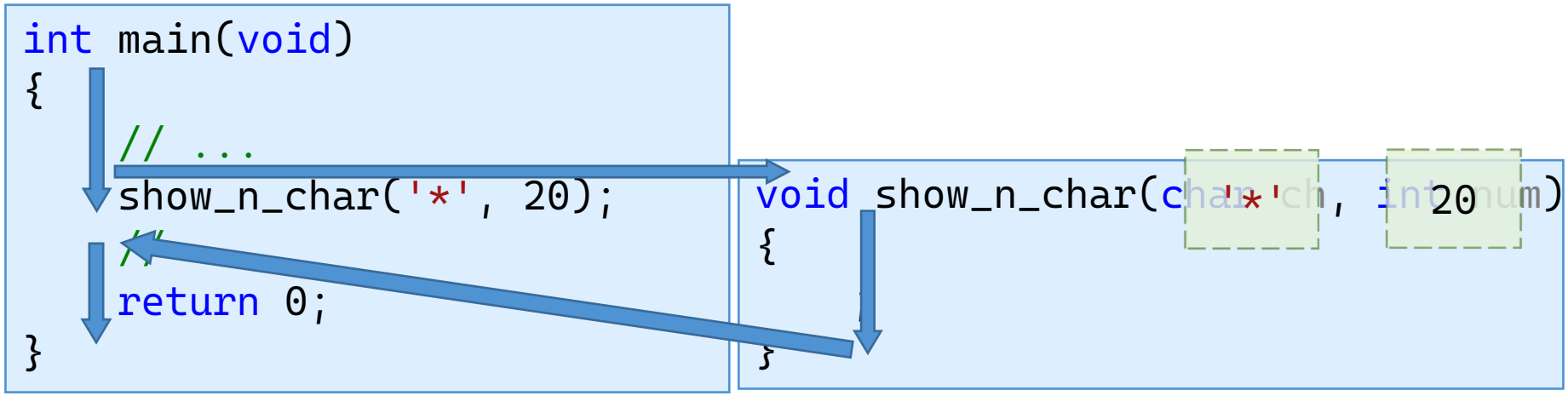
函数调用（Function call）

- 函数调用的执行过程和内存内容



函数参量和参数

- 函数参量称为形式参量
  - 函数调用者向被调用者传递的信息的量
- 实际参数
  - 函数在调用时通过实际参数对形式参量进行赋值



```c
/* lethead2.c */
#include <stdio.h>
#include <string.h>                 /* for strlen() */
#define NAME "GIGATHINK, INC."
#define ADDRESS "101 Megabuck Plaza"
#define PLACE "Megapolis, CA 94904"
#define WIDTH 40
#define SPACE ' '
void show_n_char(char ch, int num);
int main(void) {
    int spaces;
    show_n_char('*', WIDTH); /* using constants as arguments */
    putchar('\n');
    show_n_char(SPACE, 12);  /* using constants as arguments */
    printf("%s\n", NAME);
    spaces = (WIDTH - strlen(ADDRESS)) / 2;
    /* Let the program calculate how many spaces to skip */
    show_n_char(SPACE, spaces); /* use a variable as argument */
```

打印星号的函数与具体符号、长度搅在一起，是为耦合。有经验的程序员将符号、长度提取为参量，降低耦合度。

函数的声明，这是个有参量的函数。

使用宏代替空格字符将便于阅读

居中对齐的缩进计算

```
    printf("%s\n", ADDRESS);
    show_n_char(SPACE, (WIDTH - strlen(PLACE)) / 2);
    /* an expression as argument    */
    printf("%s\n", PLACE);
    show_n_char('*', WIDTH);
    putchar('\n');
    return 0;
}
/* show_n_char() definition */
void show_n_char(char ch, int num) {
    int count;
    for (count = 1; count <= num; count++)
        putchar(ch);
}
```

这只是表达式作为参数的示例。有经验的程序员不会在程序中使用不同风格的做法。

```
*****************************************
            GIGATHINK, INC.
          101 Megabuck Plaza
         Megapolis, CA 94904
*****************************************
```

黑盒子观点

- 黑盒子里的一切操作对调用函数而言是不可见的
- 函数内部的变量，不影响外部同名变量
- 函数的参量值变化，不影响外部参数

函数参量：无参量和不确定参量

- 无参量
  - 参量列表为空表示无参量

```
int imax();
```

  - 参量列表显式用void记为无参量

```
int imax(void);
```

- 不确定参量

```
int imax(int, ...);
```

  - 读取参量通过 va_arg

```c
/* misuse.c -- uses a function incorrectly */
#include <stdio.h>
int imax();       /* old-style declaration */

int main(void)
{
    printf("The maximum of %d and %d is %d.\n",
           3, 5, imax(3));
    printf("The maximum of %d and %d is %d.\n",
           3, 5, imax(3.0, 5.0));
    return 0;
}

int imax(n, m)
int n, m;
{
    return (n > m ? n : m);
}
```

旧式声明不写参量，在调用时发生错误却无法产生编译错误

```
The maximum of 3 and 5 is 3.
The maximum of 3 and 5 is 1074266112.
```

```c
/* proto.c -- uses a function prototype */
#include <stdio.h>
int imax(int, int);          /* prototype */
int main(void) {
    printf("The maximum of %d and %d is %d.\n",
           3, 5, imax(3));
    printf("The maximum of %d and %d is %d.\n",
           3, 5, imax(3.0, 5.0));
    return 0;
}
int imax(int n, int m) {
    return (n > m ? n : m);
}
```

新式声明写明参量，参数不匹配产生编译错误

在调用时使用了错误但兼容数据的类型，先做强制类型转换而非编译错误

```
proto.c: In function ‘main’:
proto.c:7:12: error: too few arguments to function ‘imax’
             3, 5, imax(3));
             ^
proto.c:3:5: note: declared here
 int imax(int, int);          /* prototype */
```

目录

递归函数（Recursive Functions）

- 递归函数是调用自身的函数
  - 每一级的函数调用都有自己的变量，但共用一个代码段
  - 每次调用都会有一次返回
- 函数的调用顺序
  - 递归调用之前顺序执行
  - 调用语句之后逆序执行

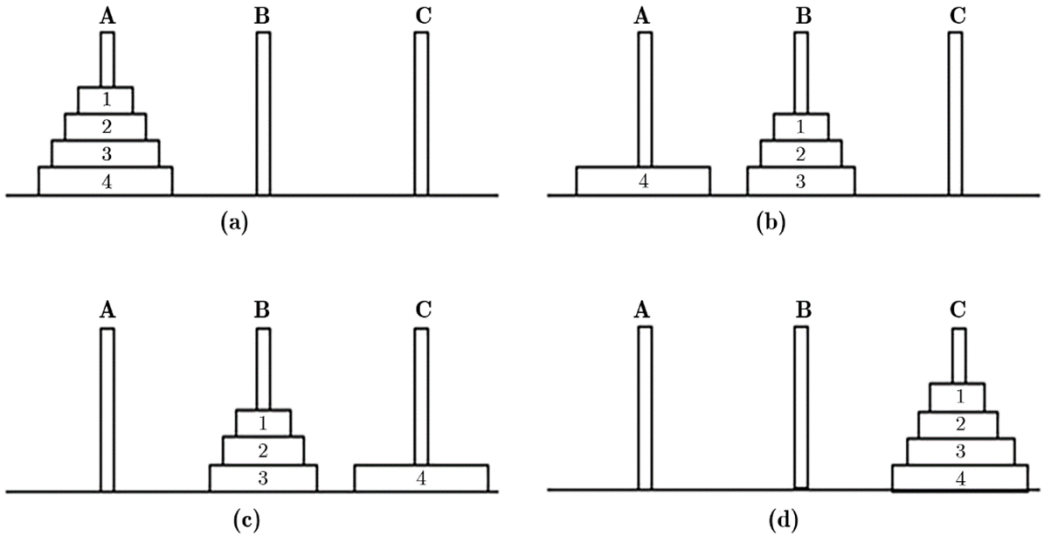
参数	各层变量n的值			
	第1层	第2层	第3层	第4层
调用第1层之后	1	*	*	*
调用第2层之后	1	2	*	*
调用第3层之后	1	2	3	*
调用第4层之后	1	2	3	4
第4层返回之后	1	2	3	*
第3层返回之后	1	2	*	*
第2层返回之后	1	*	*	*
第1层返回之后	*	*	*	*

```c
/* recur.c -- recursion illustration */
#include <stdio.h>
void up_and_down(int);

int main(void)
{
    up_and_down(1);
    return 0;
}

void up_and_down(int n)
{
    printf("Level %d: n location %p\n", n, &n); // 1
    if (n < 4)
        up_and_down(n+1);
    printf("LEVEL %d: n location %p\n", n, &n); // 2
}
```

```
Level 1: n location 0018F9D4
Level 2: n location 0018F8FC
Level 3: n location 0018F824
Level 4: n location 0018F74C
LEVEL 4: n location 0018F74C
LEVEL 3: n location 0018F824
LEVEL 2: n location 0018F8FC
LEVEL 1: n location 0018F9D4
```

尾递归

- 恰在返回语句之前的递归称为尾递归
- 示例
  - 阶乘 $f(n) = n!;\ f(n) = n \cdot (n-1)! = n \cdot f(n-1)$
  - 斐波那契数列 $F(0) = 0;\ F(1) = 1;\ F(n) = F(n-1) + F(n-2)$
  - 汉诺塔
    - **1~*n*-1 : A→B**
    - ***n*** **: A→C**
    - **1~*n*-1 : B→C**



```c
// factor.c -- uses loops and recursion to calculate factorials
#include <stdio.h>
long fact(int n);
long rfact(int n);
int main(void) {
    int num;
    printf("This program calculates factorials.\n");
    printf("Enter a value in the range 0-12 (q to quit):\n");
    while (scanf("%d", &num) == 1) {
        if (num < 0)
            printf("No negative numbers, please.\n");
        else if (num > 12)
            printf("Keep input under 13.\n");
        else {
            printf("loop: %d factorial = %ld\n", num, fact(num));
            printf("recursion: %d factorial = %ld\n", num, rfact(num));
        }
        printf("Enter a value in the range 0-12 (q to quit):\n");
    }
```

```
    printf("Bye.\n");
    return 0;
}
long fact(int n) {      // loop-based function
    long ans;
    for (ans = 1; n > 1; n--)
        ans *= n;
    return ans;
}
long rfact(int n) {     // recursive version
    long ans;
    if (n > 0)
        ans= n * rfact(n-1);
    else
        ans = 1;
    return ans;
}
```

递归调用

递归函数在某些情况下不递归调用，才能终止

```
This program calculates factorials.
Enter a value in the range 0-12 (q to quit):
5↲
loop: 5 factorial = 120
recursion: 5 factorial = 120
Enter a value in the range 0-12 (q to quit):
q↲
Bye.
```

```c
/* binary.c -- prints integer in binary form */
#include <stdio.h>
void to_binary(unsigned long n);

int main(void) {
    unsigned long number;
    printf("Enter an integer (q to quit):\n");
    while (scanf("%lu", &number) == 1) {
        printf("Binary equivalent: ");
        to_binary(number);
        putchar('\n');
        printf("Enter an integer (q to quit):\n");
    }
    printf("Done.\n");

    return 0;
}
```

```
void to_binary(unsigned long n)  /* recursive function */
{
    int r;

    r = n % 2;
    if (n >= 2)
        to_binary(n / 2);
     putchar(r == 0 ? '0' : '1');

    return;
}
```

递归函数在某些情况下不递归调用，才能终止

```
Enter an integer (q to quit):
49↲
Binary equivalent: 110001
Enter an integer (q to quit):
-3↲
Binary equivalent:
11111111111111111111111111111101
Enter an integer (q to quit):
q↲
Done.
```

递归的优缺点

- 优点
  - 为某些编程问题提供了最简单的解决方法，结构优美
- 缺点
  - 递归算法耗费计算机内存资源，限制计算规模，效率较低
- 解决方案
  - 自己实现“栈”，选择最少的Push和Pop操作
  - 设计矩阵保存已知的计算结果，免得下次再算

目录

函数设计需要遵循原则

- 软件程序需要维护
  - OJ题AC后一般不再见，软件需要不断修正错误，增加功能
- 函数设计需要遵循原则
  - 结构清晰的程序，成员专注于本自己的模块，不相互干扰
  - 代码自明的程序，减少沟通，提高效率

你调用队友负责的某个函数，一会儿需要打电话问他参数是干嘛用的，一会儿又要问他能不能改某个值。

你调用队友负责的某个函数，当交作业的前十分钟他兴冲冲地跑来告诉你，他把参数列表改得更合理了。

- 好的程序不是设计来的，而是进化来的
  - 写出优秀的代码需要：先写代码、再打磨代码

函数设计原则：单一职责

- 单一职责：一个函数只完成一个任务或一方面功能
  - 函数要么执行操作，要么返回结果，不可兼顾。

```
void process(int* in, int* out) {
    scanf("%d", in);
    *out = *in * 2;
}
```

```
void read_input(int* in) {
    scanf("%d", in);
}
int double_value(int in) {
    return input * 2;
}
```

  - 勿用标识参量：标识说明函数不止做一件事，应拆分。

```
const char* getName(int isFullName) {
    if (isFullName)
        return "John Smith";
    else
        return "Smith";
}
```

```
const char* getFamilyName() {
    return "John Smith";
}
const char* getFullName() {
    return "Smith";
}
```

函数设计原则：单一职责

- 单一职责原则
  - 函数应无副作用，因为副作用是一种谎言，违背单一职责
    - 对变量做出未能预期的改动，导致时序耦合和顺序依赖

```
int checkPassword(const char* account, char* password) {
    const char* decPass = decrypt(password);
    if (strcmp(account, decPass) == 0) {
        SessionInitialize();
        return 1;
    }
    return 0;
}
```

```
int checkPassword(const char* account, char* password) {
    const char* decPass = decrypt(password);
    return strcmp(account, decPass) == 0;
}
void handleAuthenticationSuccess(void) {
    SessionInitialize();
}
void authenticate(const char* account, char* password) {
    if (checkPassword(account, password))
        handleAuthenticationSuccess();
}
```

函数设计原则：命名清晰准确

- 命名清晰反映功能描述
  - 函数名使用动词或动宾短语，参量名使用名词

```
void write(const char* name);
void writeField(const char* name);
int assertExpectedEqualsActual(int expected, int actual);
```

函数设计原则：输入输出接口

- 函数参量应尽量少：参数多用着麻烦，内聚有问题
  - 有足够理由才能使用3个以上的参数（考虑结构体）
  - 参量应按重要性降序排列，不需修改的参量加 const 标记

```
int countpass(int passline, float math, float english, float science) {
    int count = 0;
    count += math >= passline;
    count += english >= pass
    count += science >= pass
    return count;
}
```

```
typedef struct {
    float math;
    float english;
    float science;
} Score;
int countpass(Score scores, int passline) {
    int count = 0;
    count += scores.math >= passline;
    count += scores.english >= passline;
    count += scores.science >= passline;
    return count;
}
```

函数设计原则：输入输出接口

- 函数参量应尽量少：参数多用着麻烦，内聚有问题
  - 应对输入参数进行有效性检查，确保参数符合预期。

```
int divide(const int numerator, const int denominator) {
    if (denominator == 0) {
        printf("Error: Division by zero.\n");
        return -1;
    }
    return numerator / denominator;
}
```

函数设计原则：结构化编程

- 避免使用全局变量
  - 尽量避免不必要的全局变量，减少耦合和潜在的副作用

```c
#include <stdio.h>
void format();
int num = 0;
int main() {
    int m;
    for (m = 1; m * m <= 200; m++) {
        printf("%d", m * m);
        format();
    }
    return 0;
}
void format() {
    num++;
    printf(num % 5 == 0?"\n":" ");
    return;
}
```

```c
#include <stdio.h>

// 将num作为参数传递给format函数
void format(int* num) {
    (*num)++;
    printf(*num % 5 == 0?"\n":" ");
}

int main() {
    int m, num = 0;
    for (m = 1; m * m <= 200; m++) {
        printf("%d", m * m);
        format(&num);
    }
    return 0;
}
```

函数设计原则：结构化编程

- 层次清晰，在同一个抽象层级
  - 避免将顶层逻辑与底层细节混合在一起。
  - 应分离到不同函数中，代码更模块化，更容易理解和维护。

```
void initBoard() {
    int x, y;
    for (x = 0; x < SIZE; x++)
        for (y = 0; y < SIZE; y++)
            board[x][y] = 0;
    addRandom();
    addRandom();
}
```

```
void cleanBoard() {
    int x, y;
    for (x = 0; x < SIZE; x++)
        for (y = 0; y < SIZE; y++)
            board[x][y] = 0;
}
void initBoard() {
    cleanBoard();
    addRandom();
    addRandom();
}
```

- 控制函数规模
  - 数体规模应尽量小，避免超过一至两屏（每屏25行40列）。

函数设计原则：高内聚低耦合

- 低耦合必然意味着高内聚。
  - 内聚是就其中任何一个模块的内部特征而言的。
  - 耦合是就多个模块组成的系统中各模块关联关系而言的。
- 避免内容耦合
  - 当起心动念想要复制粘贴时，可能耦合出问题了
    - 例如：2048的上下左右走

函数设计原则：高内聚低耦合

扩展内容

- 耦合性的六个级别

级别	说明
无直接耦合	指两个模块之间没有直接关系
数据耦合	指两个模块之间有调用关系，传递的是简单的数据值，相当于高级语言的值传递
标记耦合	指两个模块之间传递的是数据结构，如高级语言中的数组名、记录名、文件名等名字即标记，其实传递的是这个数据结构的地址
控制耦合	指一个模块调用另一个模块时，传递的是控制变量（如开关、标志等），被调模块通过该控制变量的值选择执行块内某一功能
公共耦合	指通过一个公共数据环境相互作用的那些模块间的耦合。公共耦合的复杂程序随耦合模块的个数增加而增加
内容耦合	这是最高程度的耦合，也是最差的耦合。当一个模块直接使用另一个模块的内部数据，或通过非正常入口而转入另一个模块内部

函数设计原则：高内聚低耦合

扩展内容

- 内聚性的六个级别

级别	说明
偶然内聚	指一个模块内的各处理元素之间没有任何联系
逻辑内聚	指模块内执行几个逻辑上相似的功能，通过参数确定该模块完成哪一个功能
时间内聚	把需要同时执行的动作组合在一起形成的模块为时间内聚模块
通信内聚	指模块内所有处理元素都在同一个数据结构上操作（有时称之为信息内聚），或者指各处理使用相同的输入数据或者产生相同的输出数据
顺序内聚	指一个模块中各个处理元素都密切相关于同一功能且必须顺序执行，前一功能元素输出就是下一功能元素的输入
功能内聚	这是最强的内聚，指模块内所有元素共同完成一个功能，缺一不可。与其他模块的耦合是最弱的

目录

函数复用

- 书写函数提供他人使用，需要提供声明和实现
- 共享函数的声明
  - 函数原型：通常存于头文件（*.h）中
- 共享函数的实现
  - 有的程序员开放源代码共享
    - 将函数源代码归类于源代码文件（*.c）中
  - 有的程序员掩盖技术秘密
    - 将源代码编译为目标文件（*.obj），打包为库（*.lib）文件

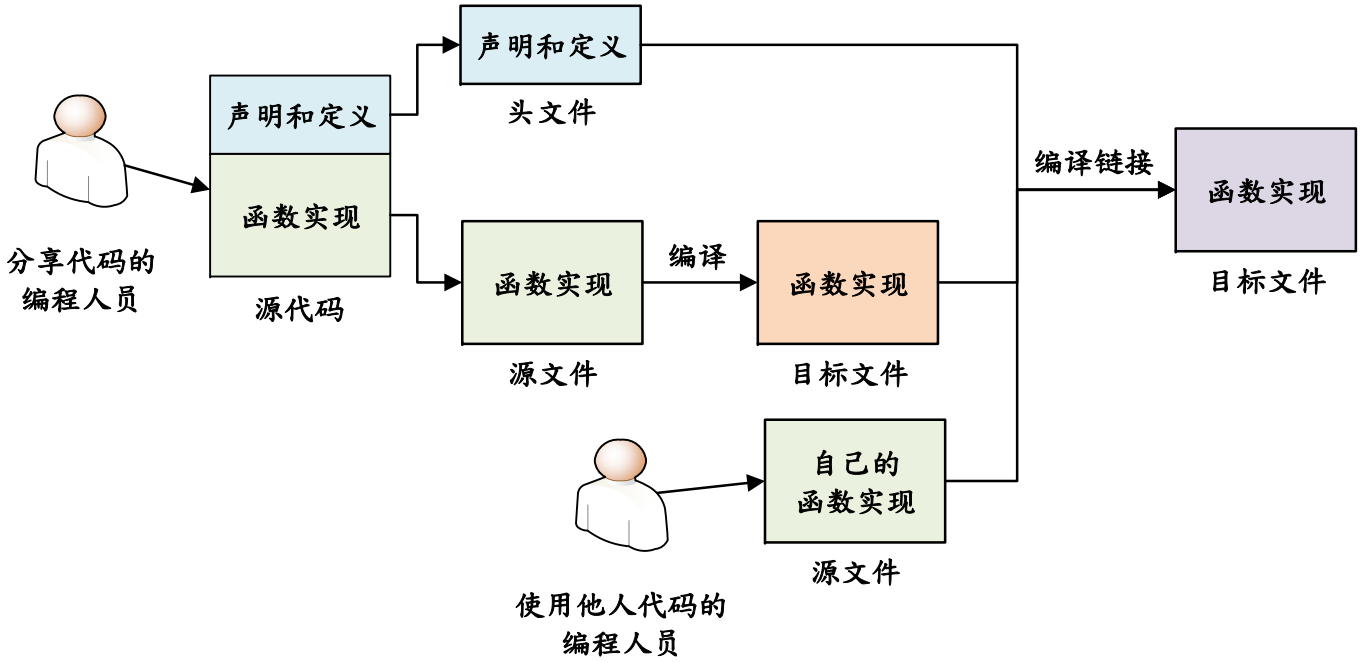
函数复用

- 编译命令行

编译目的	GCC	Visual Studio
通过命令行将代码编译为目标文件	`gcc -c file2.c -o file2.o`	`cl /c file2.c /Fofile2.obj`
利用目标文件将代码编译为可执行文件	`gcc file1.c file2.o`	`cl file1.c file2.obj`
将多个代码编译为可执行文件	`gcc file1.c file2.c`	`cl file1.c file2.c`
生成库文件	`ar cr ext.a ext.o ext2.o`	

分发过程（GCC）

- 在提供软件包的一方将源代码编译为目标文件
  - 将得到的ext.o与ext.h分发出去
- 使用软件包的一方
  - 包含文件头



```c
// ext.c -- an external code
#include <stdio.h>

int print_msg(const char* message)
{
    int ret;
    ret = printf("In ext.c, print_msg(\"%s\")\n",
message);
    printf("In ext.c, print_msg: RETURN %d\n", ret);
    return ret;
}

void prt_msg(const char* message)
{
    printf("In ext.c, prt_msg(\"%s\")\n", message);
}
```

函数的实现
在此处包含或不包含原型
声明都可以
（不会造成编译错误即可）

```c
// ext.h -- a header file of ext.c
int print_msg(const char* message);
void prt_msg(const char* message);
```

函数的设计者应提供函数的原型声明

```c
// main.c - a code to test ext.c
#include <stdio.h>
#include "ext.h"
int main()
{
    int getret;
    printf("In main.c, main()\n");
    printf("In main.c, CALL print_msg()\n");
    getret = print_msg("This is a message");
    printf("In main.c, print_msg() RETURN %d\n", getret);
    printf("In main.c, CALL prt_msg()\n");
    prt_msg("This is a message");
    printf("In main.c, main()\n");
    return 0;
}
```

使用者包含该头文件，
使用函数的原型声明

声明可以直接用

```
In main.c, main()
In main.c, CALL print_msg()
In ext.c, print_msg("This is a message")
In ext.c, print_msg: RETURN 41
In main.c, print_msg() RETURN 41
In main.c, CALL prt_msg()
In ext.c, prt_msg("This is a message")
In main.c, main()
In main.c, CALL print_msg2()
In ext.c, print_msg2("This is a message")
In ext2.c, print_msg2: RETURN 42
In main.c, print_msg() RETURN 42
In main.c, CALL prt_msg2()
In ext2.c, prt_msg2("This is a message")
In main.c, main()
```

```c
/* usehotel.c -- room rate program */
/* compile with  Listing 9.10      */
#include <stdio.h>
#include "hotel.h" /* defines constants, declares functions */
int main(void) {
    int nights;
    double hotel_rate;
    int code;
    while ((code = menu()) != QUIT) {
        switch (code) {
        case 1: hotel_rate = HOTEL1; break;
        case 2: hotel_rate = HOTEL2; break;
        case 3: hotel_rate = HOTEL3; break;
        case 4: hotel_rate = HOTEL4; break;
        default: hotel_rate = 0.0;
            printf("Oops!\n"); break;
        }
        nights = getnights();
        showprice(hotel_rate, nights);
    }
    printf("Thank you and goodbye.\n");
    return 0;
}
```

```c
/* hotel.c -- hotel management functions */
#include <stdio.h>
#include "hotel.h"
int menu(void) {
    int code, status;
    printf("\n%s%s\n", STARS, STARS);
    printf("Enter the number of the desired hotel:\n");
    printf("1) Fairfield Arms           2) Hotel Olympic\n");
    printf("3) Chertworthy Plaza        4) The Stockton\n");
    printf("5) quit\n");
    printf("%s%s\n", STARS, STARS);
    while ((status = scanf("%d", &code)) != 1  ||
           (code < 1 || code > 5)) {
        if (status != 1)
            scanf("%*s");   // dispose of non-integer input
        printf("Enter an integer from 1 to 5, please.\n");
    }
    return code;
}
```

```c
int getnights(void) {
    int nights;
    printf("How many nights are needed? ");
    while (scanf("%d", &nights) != 1) {
        scanf("%*s");     // dispose of non-integer input
        printf("Please enter an integer, such as 2.\n");
    }
    return nights;
}
void showprice(double rate, int nights)
{
    int n;
    double total = 0.0;
    double factor = 1.0;
    for (n = 1; n <= nights; n++, factor *= DISCOUNT)
        total += rate * factor;
    printf("The total cost will be $%0.2f.\n", total);
}
```

```c
/* hotel.h -- constants and declarations for hotel.c */
#define QUIT        5
#define HOTEL1    180.00
#define HOTEL2    225.00
#define HOTEL3    255.00
#define HOTEL4    355.00
#define DISCOUNT    0.95
#define STARS "*************************************"

// shows list of choices
int menu(void);

// returns number of nights desired
int getnights(void);

// calculates price from rate, nights
// and displays result
void showprice(double rate, int nights);
```

头应该包含：宏定义、常数，全局变量或函数声明。

```
**********************************************************************
Enter the number of the desired hotel:
1) Fairfield Arms           2) Hotel Olympic
3) Chertworthy Plaza        4) The Stockton
5) quit
**********************************************************************
2↵
How many nights are needed? 3↵
The total cost will be $641.81.

**********************************************************************
Enter the number of the desired hotel:
1) Fairfield Arms           2) Hotel Olympic
3) Chertworthy Plaza        4) The Stockton
5) quit
**********************************************************************
3↵
How many nights are needed? a↵
Please enter an integer, such as 2.
2↵
The total cost will be $497.25.

**********************************************************************
Enter the number of the desired hotel:
1) Fairfield Arms           2) Hotel Olympic
3) Chertworthy Plaza        4) The Stockton
5) quit
**********************************************************************
5↵
Thank you and goodbye.
```

C语言程序设计

C Programming

9

谢谢观看

理论课程

厦門大學

XIAMEN UNIVERSITY

信息学院 黄 炜

(特色化示范性软件学院) 博士,副教授

School of Informatics Dr. Wei Huang