函数进阶和练习

中山大学计算机学院

讲课人：潘茂林

目录

CONTENTS

01 问题求解-最大公约数

02 汉诺塔（Hanoi Tower）

03 编译与程序布局

04 头文件初步

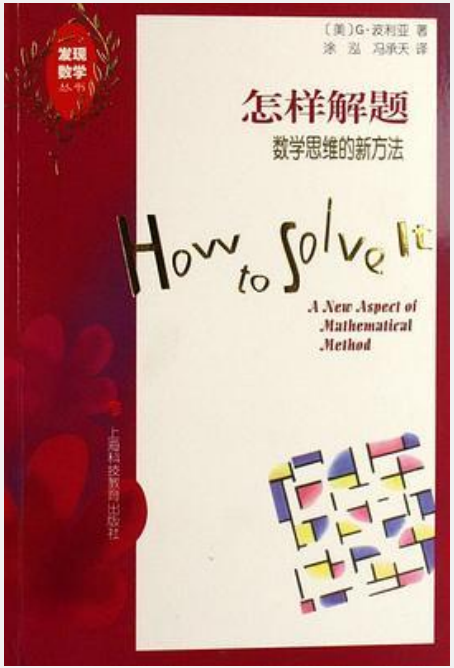
05 题库实战案例研究

问题求解

问题求解（*problem solving*）一般方法：

1. *What do I know about the problem?*
2. *What is the information that I have to process in order the find the solution?*
3. *What does the solution look like?*
4. *What sort of special cases exist?*
5. *How will I recognize that I have found the solution?*

计算问题的本质就是数学，这本书可能对你提升问题求解能力有所帮助!



作者: [美] G·波利亚

怎样解题 (豆瓣) (douban.com)

最大公约数 – 问题描述

最大公约数，也称**最大公因数（*greatest common divisor*）**，是指两个或多个整数共有约数中最大的一个。记为

$$\gcd(a, b)$$

1. 审题：从数学角度，理解最大公因素定义是解题的第一步
2. 输入：整数 a，b
3. 输出：整数 $g = \arg\max_{d}(a \,\%\, d == 0 \;\&\&\; b \,\%\, d == 0)$
4. 案例：$\gcd(3,5) = 1;\ \gcd(3,9) = 3;\ \gcd(8,12) = 4;$
5. 类似问题：求 n 以内的最大素数

最大公约数 – 算法设计

计算方法1：试除法

两个数的最大公约数有可能是其中较小的数。所以在按从大到小顺序找寻最大公约数时，循环变量i的初值从$\min(a,b)$开始依次递减直到 1，去寻找第一个能同时被两整数整除的自然数，并将其输出。（注意，1是最小的公约数，不用考虑无解的问题）

最大公约数 – 编程技巧

```c
/*gcd.c*/
#include<stdio.h>
#include<assert.h>

inline int min(int a,int b);
int gcd(int a,int b);

int main() {
    int a,b;
    assert(gcd(3,5)==1);
    assert(gcd(3,9)==3);
    assert(gcd(8,12)==4);
    scanf("%d%d",&a,&b);
    printf("%d\n",gcd(a,b));
    return 0;
}
```

转为内联函数

练习：修改12行 assert(gcd(3,9)==4);
运行程序，请解释输出的意义。

```c
int min(int a,int b){
    return (a > b)? b:a;
}

int gcd(int a,int b) {
    int n = min(a,b);
    int i;
    for(i=n; i>0; i--){
        if (!(a % i) && !(b % i))
            break;
    }
    return i;
}
```

程序要点：
① 先申明相关函数
② 对返回表达式这样的简单函数，添加 inline 修饰提升执行性能。
③ 使用 assert(表达式) 测试函数正确性

编程技巧 – assert 断言

如何确认自己编写的函数是对的，又不影响程序的提交?

使用 assert.h 中的

$$assert(expr)$$

当 expr 为真，这个语句不会产生任何输出
当 expr 为假，语句中止执行，并给出错误信息

关键是：在 #include<assert.h> 前添加 #define NDEBUG，例如：

```
#define NDEBUG
#include<assert.h>
```

所有 assert 语句就被编译器自动忽视了!

最大公约数 – 算法设计2

计算方法2：辗转相除法

其算法定义是：

$$\gcd(a,b)=\begin{cases}a & where\ b\%a==0\\ \gcd(b\%a,a) & where\ b\%a\ !=0\end{cases}$$

例如：$\gcd(98,63)=\gcd(63,98)=\gcd(35,63)$
$=\gcd(28,35)=\gcd(7,28)=7$

这么巧妙的方法是如何想出来的?
经典算法，出于**欧几里得**的《几何原本》，你的任务就是背诵经典，并用C语言实现它。
因为它是许多整数问题的基础函数，如计算最小公倍数等等。

课堂练习 - 最大公约数

计算方法2：辗转相除法

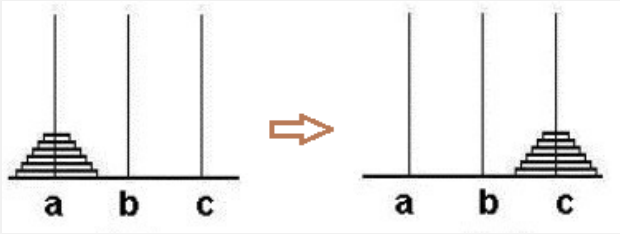
任务要求：

- 将 gcd 函数使用辗转相除法实现
- 打印 gcd(98,63) 的计算过程

递归经典：汉诺塔（Hanoi Tower）

- 问题描述

  源于印度一个古老传说。大梵天创造世界的时候做了三根金刚石柱子，在一根柱子上从下往上按照大小顺序摞着64片黄金圆盘。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，任何时候，在小圆盘上都不能放大圆盘，且在三根柱子之间一次只能移动一个圆盘。问应该如何操作？



汉诺塔求解过程分析

- 问题求解
  - 申明问题

  void hanoi(int n, int a, int b, int c);
    - a,b,c 代表柱子编号，用字母表示。分别代表 source，mid，destination
    - 将 n 个从大到小的盘子从source上，使用 mid 周转，最后到 dest 上

  void move(int n, int a, int b)

    将第 n 号盘子从 a 号柱子搬到 b 号柱子

这些例子表明，正确理解并抽象问题是问题求解的关键之一！

汉诺塔求解过程分析

- 问题求解
  - 最小子问题

hanoi(1,'a','b','c') → move(1, 'a', 'c')

  - 验证 hanoi(2 ,'a','b','c')

hanoi(1,'a','c','b') → move(1, 'a', 'b') 周转
move(2, 'a', 'c')
hanoi(1,'b','a','c') → move(1, 'b', 'c') 盖在 2 上

  - 分解问题

hanoi(n,'a','b','c');
hanoi(n-1, a, c, b); 以 c 为中转，把 n-1 个盘子从 a柱 移到 周转柱子 上
move(n, a, c); 把 a上的编号为n（最大）的盘子移到 目标柱子 上
hanoi(n-1, b, a, c); 以 a 为中转，把 n-1个盘子从 周转柱子 移到 目标柱子 上

参考程序：汉诺塔

```
void hanoi(int n,int a,int b,int c){
    if (n==1) {
        move(n,a,c);
        return;
    }
    hanoi(n-1, a, c, b);
    move(n,a,c);
    hanoi(n-1, b, a, c);
}
```

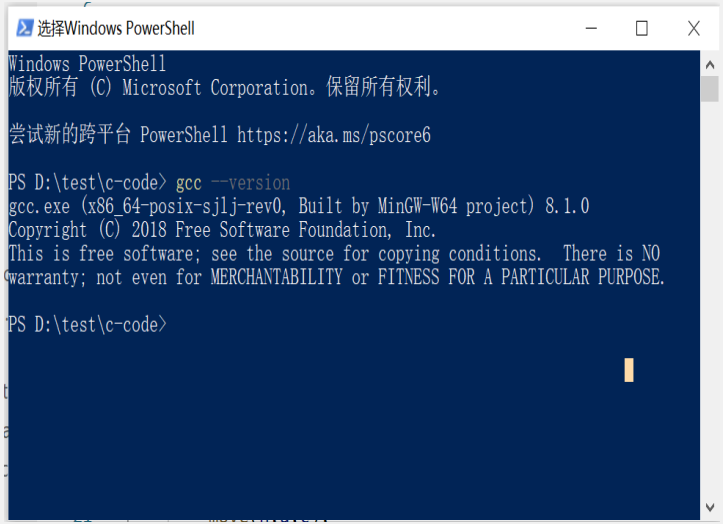
这么简单？请输入 1，2，3 分别测试，并用自然语言解释 3 结果。

注意：右边程序在测评系统中是不可以修改的，甚至（不给钱）不给你观看。

```
#include <stdio.h>

//声明函数，其中 a,b,c 是柱子编号，用'a','b','c'表示
void hanoi(int n,int a,int b,int c);
void move(int n,int a, int b);

int main()
{
    int n;
    scanf("%d",&n);
    hanoi(n,'a','b','c');
    return 0;
}

void move(int n, int a, int b) {
    printf("move %d from %c to %c\n",n,a,b);
}
```

手工编译环境准备

- 设置环境变量
  - 在window 10左下角输入“编辑系统环境变量”
  - 点击“环境变量(N)...”按钮
  - 选择 path 属性，点击“编辑(E)...”按钮
  - 将 ”C:\Program Files (x86)\Dev-Cpp\MinGW64\bin”添加入 path
  - 点击按钮“确定”
- 检查确认环境
  - 使用文件浏览器，打开一个包含c程序的目录
  - 在文件浏览器路径区输入“powershell”或 “cmd”
  - 在命令行程序中输入 “gcc –version”



编译过程

- 预处理阶段
  - 处理一个 c 文件分为6个小步骤：
  1. 将文件映射到96个字符的构成的文件
  2. 处理续行符号
  3. 分解文件为注释，空白字符，预处理标记，标识符，数字常量，字面量，运算符等元素。用一个空格代替注释段落
  4. 执行预处理。并递归地按1-4步骤**处理 #include 文件**
  5. 将字符常量和字符串字面量转为平台执行编码
  6. 链接相邻的字符串
  - 例如：gcc –E hello.c

翻译阶段 - cppreference.com

- 编译阶段
  - 按照**语法和语义分析**记号，并将它们翻译成翻译单元。
  - 例如：gcc –c hello.c
- 链接阶段
  - 将多个翻译单元中**未解决的符号**，给出具体地址，组成成一个可执行文件。
  - 例如：gcc main.c hanoi.c

案例：汉诺塔程序的布局与编译（all-in-one）

```c
#include <stdio.h>

//声明函数，其中 a,b,c 是柱子编号，用'a','b','c'表示
void hanoi(int n,int a,int b,int c);
void move(int n,int a, int b);

int main()
{
    int n;
    scanf("%d",&n);
    hanoi(n,'a','b','c');
    return 0;
}

void move(int n, int a, int b) {
    printf("move %d from %c to %c\n",n,a,b);
}
```

```c
void hanoi(int n,int a,int b,int c){
    if (n==1) {
        move(n,a,c);
        return;
    }
    hanoi(n-1, a, c, b);
    move(n,a,c);
    hanoi(n-1, b, a, c);
}
```

编译指令： gcc hanoi-all-in-one.c

案例：汉诺塔程序的布局与编译（embed）

```c
#include <stdio.h>

void hanoi(int n,int a,int b,int c);
void move(int n,int a, int b);
#include “hanoi.h”

int main()
{
    int n;
    scanf("%d",&n);
    hanoi(n,'a','b','c');
    return 0;
}

void move(int n, int a, int b) {
    printf("move %d from %c to %c\n",
n,a,b);
}
```

hanoi-embed.c

```c
void hanoi(int n,int a,int b,int c){
    if (n==1) {
        move(n,a,c);
        return;
    }
    hanoi(n-1, a, c, b);
    move(n,a,c);
    hanoi(n-1, b, a, c);
}
```

hanoi.h **注意，该文件不能独立编译**

编译指令：gcc hanoi-embed.c
在线测试中常见的布局形式

案例：汉诺塔程序的布局与编译（normal）

```c
#include <stdio.h>
#include “hanoi.h”

int main()
{
    int n;
    scanf("%d",&n);
    hanoi(n,'a','b','c');
    return 0;
}

void move(int n, int a, int b) {
    printf("move %d from %c to %c\n",
n,a,b);
}
```

main.c　　**编译指令：** gcc –c main.c

```c
//a,b,c 是柱子编号，用'a','b','c'表示
void hanoi(int n,int a,int b,int c);
void move(int n,int a, int b);
```

hanoi.h 仅用于包含在源文件中

```c
#include “hanoi.h”

void hanoi(int n,int a,int b,int c){
    if (n==1) {
        move(n,a,c);
        return;
    }
    hanoi(n-1, a, c, b);
    move(n,a,c);
    hanoi(n-1, b, a, c);
}
```

hanoi.c　　**编译指令：** gcc –c hanoi.c

案例：汉诺塔程序编译与链接（normal）

```c
#include <stdio.h>
#include “hanoi.h”

int main()
{
    int n;
    scanf("%d",&n);
    hanoi(n,'a','b','c');
    return 0;
}

void move(int n, int a, int b) {
    printf("move %d from %c to %c\n",
n,a,b);
}
```

main.c

编译指令： gcc –c main.c
产生 main.o
其中 scanf,hanoi,printf 有申明，无定义。
move 有定义且有外部链接

```c
//a,b,c 是柱子编号，用'a','b','c'表示
void hanoi(int n,int a,int b,int c);
void move(int n,int a, int b);
```

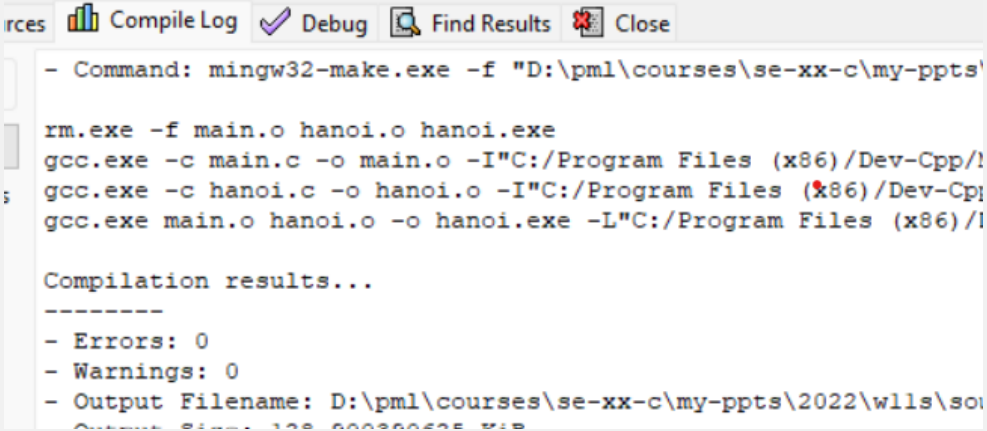
hanoi.h 仅用于包含在源文件中

```c
#include “hanoi.h”

void hanoi(int n,int a,int b,int c){
    if (n==1) {
        move(n,a,c);
        return;
    }
    hanoi(n-1, a, c, b);
    move(n,a,c);
    hanoi(n-1, b, a, c);
}
```

hanoi.c

编译指令： gcc –c hanoi.c
产生 Hanoi.o
其中 move 有申明，无定义
hanoi 有定义且有外部链接

案例：汉诺塔程序编译与链接过程



1. rm -f 强制删除文件
2. gcc -c 编译 main.c
3. gcc -c 编译 Hanoi.c
4. gcc 链接 main.o 和 hanoi.o 生成 hanoi.exe

头文件（Head File）

- 头文件是扩展名为 .h 的文件
- 头文件通常包含：
  - 共享的函数申明
  - 全局变量申明与初始化
  - 注意：头文件中函数与变量不能用 static 修饰
  - 定义的预处理符号或宏
- 头文件通过 #include 编译预处理指令添加到源代码文件中
  - 例如：源代码中需要使用内置库的求绝对值这个函数
  - 第一种方法，使用标准的头文件：#include <math.h>
  - 第二种方法，自己给出申明函数：double fabs (double);

预处理指令

- 预处理指令控制预处理器的行为。每个指令占据一行，且拥有下列格式
  - 必须以 # 字符开头
  - 预处理指令（define、undef、include、if、ifdef、ifndef、else、elif、elifdef、endif、line、error、pragma 之一）
- 预处理器拥有控制源文件翻译能力
  - 有条件编译源文件的某些部分（由 #if、#ifdef、#ifndef、#else、#elif、elifdef、和 #endif 指令控制）。
  - 替换文本宏，可以连接或加引标识符（以指令 #define 和 #undef ，运算符 # 和 ## 控制）。
  - 包含其他文件（以指令 #include 控制）。
  - 导致错误（以指令 #error 控制）。

替换文本宏

```c
#include <stdio.h>

// 制造函数工厂并使用之
#define FUNCTION(name, a) int fun_##name(int x) { return (a)*x;}

FUNCTION(quadruple, 4)
FUNCTION(double, 2)

#undef FUNCTION
#define FUNCTION 34
#define OUTPUT(a) puts( #a )

int main(void)
{
    printf("quadruple(13): %d\n", fun_quadruple(13) );
    printf("double(21): %d\n", fun_double(21) );
    printf("%d\n", FUNCTION);
    OUTPUT(million);                // 注意缺少引号
}
```

只引用一次头文件

- 如果一个头文件被引用两次，编译器会处理两次头文件的内容，这可能产生错误。为了防止这种情况，标准的做法是把文件的整个内容放在条件编译语句中，如下：

```
#ifndef HEADER_FILE
#define HEADER_FILE

the entire header file file

#endif
```

中山大學
SUN YAT-SEN UNIVERSITY

实战案例1- Cat

Description

小沁是一只可爱的很胖的猫咪，你现在要陪它玩。你每次有三个选择：摸猫猫头，喂猫粮，什么也不做，分别用数字代号2、1、0表示，它们的效果如下。

- 摸猫猫头：小沁的 `心情值` +1；
- 喂猫粮：小沁的 `心情值` +2， `体重值` 翻倍；
- 什么也不做：小沁的 `心情值` -1， `体重值` 减少四分之一（即变为原来的0.75倍）。

小沁的初始 `心情值` 是1，初始 `体重值` 是100.0。每当小沁的 `心情值` 改变时，根据**改变后的** `心情值` 输出相应语句：

- `心情值` 大于0：输出 `Little Qin is happy! XD`；
- `心情值` 等于0：输出 `Little Qin wants to play with you~ : )`；
- `心情值` 小于0：输出 `Little Qin is sad... : (`。

同理，每当小沁的 `体重值` 改变时，根据**改变后的** `体重值` 输出相应语句：

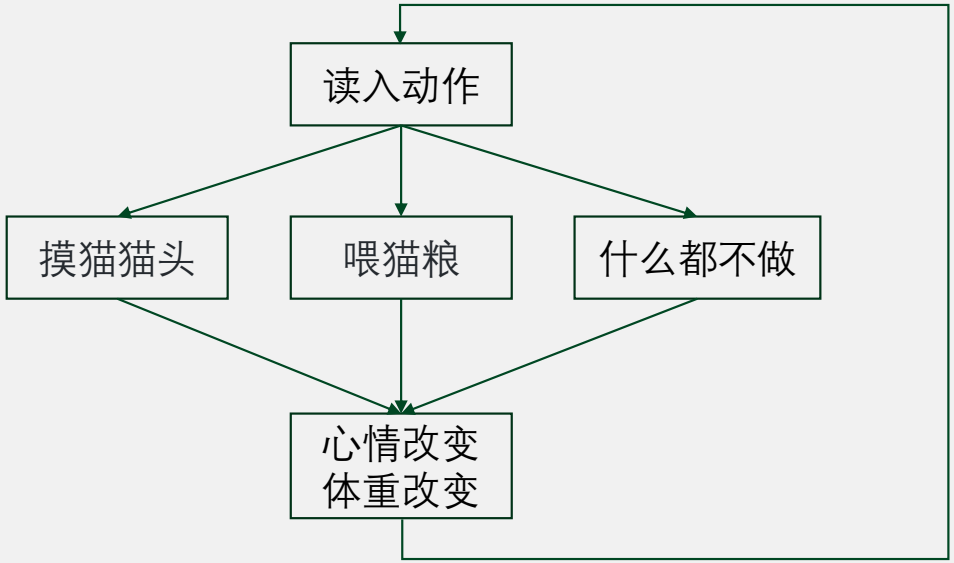
- `体重值` 小于50.0：输出 `Cute!`；
- `体重值` 在50.0和200.0之间（**包含两端**）：输出 `So cute!`；
- `体重值` 大于200.0：输出 `Extremely CUTE!`；

现在要求你根据输入的动作序列（包含7个动作），模拟陪小沁玩的过程，并输出一系列对应的语句。

实战案例 - Cat分析

分析程序的执行流程，我们可以发现，对猫做的三个动作，都会改变猫的心情和体重，只是改变的数值不同。而且，改完以后都同样要依据体重和心情来输出语句。

这种就是适合使用函数的情况。



实战案例 - cat 编程

```c
int rua(int mood)
{
    int new_mood = change_mood(mood, 1);
    return new_mood;
}

int feed(int mood)
{
    int new_mood = change_mood(mood, 2);
    change_weight(2.0);
    return new_mood;
}

int do_nothing(int mood)
{
    int new_mood = change_mood(mood, -1);
    change_weight(0.75);
    return new_mood;
}
```

```c
extern float weight;

int change_mood(int mood, int val)
{
    int new_mood = mood + val;

    if (new_mood > 0)
        printf("Little Qin is happy! XD\n");
    else if (new_mood < 0)
        printf("Little Qin is sad... : (\n");
    else
        printf("Little Qin wants to play with you~ : )\n");

    return new_mood;
}

void change_weight(float factor)
{
    weight *= factor;

    if (weight < 50.0)
        printf("Cute!\n");
    else if (weight >= 50.0 && weight <= 200.0)
        printf("So cute!\n");
    else if (weight > 200.0)
        printf("Extremely CUTE!\n");
}
```

中山大學
SUN YAT-SEN UNIVERSITY

谢 谢

中山大学计算机学院

编制人：课题组