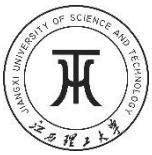

江西理工大学 一起来学

《C/C++语言程序设计》

第 3 版

C 语言划重点

习题参考答案与解析

章节例题分析与注释

江西理工大学一起来学 C/C++ 团队出品

前言

2021 · 01 · 08

各位软件工程相关专业及方向的同学们大家好，我们是一群对计算机科学、软件开发等领域具有浓烈兴趣的爱好者，我们希望能召集更多的同学加入学习计算机技术的行列。为此，我们特地创办“一起来学”栏目，来帮助同学们更好的学习 C/C++语言这门通识课程，同时弥补教学资源的不足，让编程的乐趣为更多人所知。

本栏目既可以作为学习这门课程的辅助资料，也是同学们复习期末考试的利器。

你可在本栏目中看到三个板块，它们分别是 C 语言划重点、参考答案与解析、课本例题分析。我们会将每章节知识的重点分析归纳总结放在小课堂板块供大家快速查阅、我们会弥补课本习题没有答案的缺陷，并附上我们个人的详细解答过程、我们还会把课本中的例题附上更加详细的注释帮助你理解它。

一起来学 C/C++ 团队

（排名不分先后）

特别感谢

陈璨 老师

主编

软件会计 204 史贝宁

主稿

软件会计 202 邬山峰

软件会计 203 余传钦

软件造价 181 马国智

人力资源 201 王得懿

软件开发 202 万鹏辉

为配合教学进度与同学们的复习情况，本栏目采取持续更新发布方式，截至第 3 版我们已经发布适用于 20 级的划重点、全书课后习题的参考答案与解析、1-7 章的例题分析注释。如果你觉得很棒，欢迎把资料分享给你的同学，也欢迎来我们群里为我们点赞！用爱发电，无法保证更新时效性，请同学们谅解。

请各位同学加入下方的 QQ 群，以获取最新更新与其他栏目的消息，同样的，亦可是学习的一片天地。

如果你也想加入我们的编写行列，或是创建你的一起来学团队，欢迎联系我们！

群号：239222545

资源导航

参考资料：

[1] 《C 语言程序设计》课后习题参考答案_李勃_邱晓红_主编_清华大学出版社[1-7 章参考答案] https://www.docin.com/p-553847848.html

[2] 大学 C++期末考试试卷(含答案)[12 章参考答案] https://wenku.baidu.com/view/4299ed26bcd126fff7050b1f.html

目录

C 语言划重点 2020-2021 学年第一学期

习题参考答案与解析 群文件可获取编程题源代码

章节例题分析与注释 群文件可获取例题源代码

C 语言划重点

与其说是划重点，不如说是考点速览，做不完所有的课后习题，请至少学会下面涉及到的题目
信息来源于任课教师，适用于 2020 级大一上学期期末，感谢所有老师

第一章

课后习题
选择：1、2、4
填空：1

考点注意
P5 合法标识符
P7 C 语言编译步骤，每一流程文件后缀名

第二章

课后习题
选择：1、2、5、6、13、15、19
填空：4、6

考点注意
合法表达式的判断
运算符优先级以及结合性，特别注意% / ++ --
&&以及||的短路效应
无需理会位运算
掌握赋值、逗号、条件表达式
P24 转义字符，特别注意右列

第三章

课后习题
选择：1、2、3、9、10
填空：2、3、4
编程：2

考点注意
printf 格式输出控制符，八进制与十六进制用%d
掌握%-m.n 的含义
scanf 格式输入控制符，%4d 与%d 等不同的结果

第四章

课后习题
选择：3、6、13、18
编程：1、7

考点注意
if 与 else 的配对原则
切勿使用 a<b<c 作为判断表达式
case 后必须为常量
case 语句使用后 break 跳出，否则继续往下执行

第五章

课后习题
选择：1、4、6、12、14
编程：3、5

考点注意
while 循环体内切勿忘记对循环记录变量作变化
k=0 是相等还是赋值要区分清楚
有\n 写答案要换行，没有则紧挨着写不要换行

第六章

课后习题
选择：1、3、5、7、10
填空：1、2、3、4
编程：1

考点注意
例题 6.6、6.7
有消息称考试编程题来源于第六章

第七章

课后习题
选择：1、5、7、8、11、13
编程：5

考点注意
汉诺塔问题可不看
函数的返回值是由函数定义时决定的
不同的变量类型带来的影响

第八章

课后习题
选择：3、4、9、13、15
填空：1、8、10
编程：1

考点注意
区分 int *p()以及 int (*p)()
区分 int *p[]以及 int (*p)[]
指针访问二维数组的方法
指针的指针不会考

第九章

课后习题
选择：1、2、3、4、8

考点注意
结构体的嵌套要注意分解

第十二章

课后习题
选择：1、3、4、12 记住答案

个人建议课后习题全刷

编者注

我国的高中教育与大学教育之间其实存在不小的鸿沟，大多数高中生经历十二年的语数外、政史地等科目的学习后进入大学，没有任何过渡便开始学习各个领域内的专业课。相信大家都已经经历了如此的冲击，我希望大家已经能够回过头来重新明白，高考只不过是进入大学的敲门砖而已，我们需要提炼过去的学习方法，加以改造来适应新的、面向社会的技能学习。

为了避免四年后充满悔恨，规划需要从每时每刻开始。什么是你所喜爱的，什么是你想去做的，每个时间点都应该有寻找答案的可能。

学习是一个人赖以生存的技能。如果你看过 Yjango 的学习观就会明白，人是如何靠学习走到食物链顶端的。

三个月的大一学习已经临近结束，现在你可能在奔波于考试科目复习，但最重要的事并不是考试科目的复习，我希望你找寻你所做事情的意义，找到真正属于自己的价值。

C 语言属于一门规律性比较强的学科，知识点不少，个人建议大家应该以语法入手，看懂每一个语句的具体含义和作用，能够举一反三。可以参考菜鸟教程的资料：

www.runoob.com/cprogramming/c-tutorial.html

当然看书也是不错的选择，只是你会被书的厚度吓到罢了。熟悉基本语法之后便可以掌握课后习题来巩固。

“课后习题都不会你不挂谁挂”。

B 站也有相关视频资源，但都比较长，出于时间考虑，已经不推荐大家刷视频了，看文字资料速度更快。

课后习题搞定之后可去资料库找往年题目写一写就可以等待考试了。

大家合理安排作息，祝大家考试必过！

《C/C++语言程序设计》习题参考答案与解析

我团队答案来自成员内校对、资料参考并经过代码验证，已尽可能保证其正确性。
如果您对答案有任何建议，欢迎联系我们。

第一章

邬山峰 史贝宁

答案速对 1-5 ABABC

一、单选题

1、A C 语言规定，函数开始和结束的标记是一对花括号；圆括号可用于改变运算次序，也可用于函数的参数表列；方括号用于数组的声明与使用；双引号内一般是字符串，单引号内一般是单个字符。

2、B 当程序运行时，计算机会首先寻找 main 函数的位置并从此处开始执行函数中的代码，直到 main 函数结束为止。程序中的其他函数都是“工具”，由 main 函数中的代码去使用它们。

3、A 根据上题可知 C 语言程序从 main 函数的开始处执行，一直到 main 函数的结尾处停止，故 B 选项错误；C 选项，函数是程序的基本单位，不可或缺；D 选项，标准库（stdio.h）文件中的函数提供输入输出函数 scanf 和 printf，C 语言本身不提供输入输出函数。

4、B C 语言对于 main 函数位置没有要求，因为无论 main 函数在哪个位置，程序总是从 main 函数开始，在 main 函数结尾处结束。

5、C 每一次编程之前我们需要分析编程的目的，然后思考如何用代码实现；蓝图构造好后就开始了题目所说的第一步编辑，即编辑代码；写好后需要编译，即将我们的高级语言编译成计算机可以识别的语言；然后进行第三步连接，即把刚才编译出来的结果和一些必要的文件（库代码和启动代码）打包在一起，就有了可执行文件；最后第四步运行即可。

二、填空题

大家请注意本书所有填空、分析题的答案对格式要求很严格，程序输出什么，你就必须填写什么，多了逗号，少了逗号，没有根据格式控制符填写的都算错。

1、// /* */

后者的/*和*/是多行注释，在这两个符号之间的代码不会被运行。前者是单行注释，一行中 // 之后的代码不会被运行，也称被注释掉了。

2、scanf( )

3、printf( )

三、判断题

1、√ 别问为什么，C 语言规定的，而且有且仅有一个 main 函数。

2、× 课本告诉我们 C 语言的基本组成单位应该是函数而不是语句。

3、× 注释是用来帮助自己理解代码含义的。C 程序中注释没有固定的位置，只是有固定的写法。原则上只要不破坏声明和语句的意思和结构，放在哪里都可以。一般可以在编辑器中看到正常语句和注释的颜色不一样。

```
/*
printf("您好，欢迎进入 C 语言世界！ ");
int x, y;
*/
int a, b;
// int c,d;
```

4、√ 没错，就是这样。刚才说了基本组成单位是什么来着?

四、简答题

1、

C 语言的主要特点：

①C 语言是结构化的语言；

②语言简洁、紧凑、使用方便、灵活；

③C 语言可以对硬件进行操作；

④数据类型丰富；

⑤运算符极其丰富；

⑥C 语言程序的可移植性好；

⑦C 语言生成的目标代码质量高，程序执行效率高；

⑧C 语言的语法灵活、限制不是十分严格。

C 语言程序的主要特点：

①函数是 C 语言程序结构的基本单位；

②C 语言程序只有一个主函数；

③C 语言程序的书写格式比较自由；
④C 语言中要使用声明语句，遵循先定义后使用原则；
⑤C 语言可带有编译预处理命令；
⑥C 语言中可使用注释信息，多行注释或单行注释；
⑦C 语言的标识符区分大小写；
⑧C 语言本身没有输入输出语句。

答案在课本的第 2 页和第 5 页。

2、

```
#include <stdio.h>
void main()
{
    printf("您好，欢迎进入 C 语言世界！ ");
}
```

现在感到好奇并不奇怪，随着学习的深入，你将会明白它们为什么要这么写。现在只需要照着打到电脑上，感受神奇的编程过程即可。

第二章

史贝宁

答案速对

1-5 CDCBC 6-10 ABDAB 11-15 CCDAB
16-20 CAABC 21-25 CDCAD 26-30 BCBBA

一、选择题

1、C 单引号内仅能放一个字符，而 C 选项的 abc 是字符串；双引号内可放任意数量字符；0x 开头为 16 进制表示的数；0 开头为 8 进制表示的数。

2、D 这是一个逗号表达式，整个逗号表达式的值是其中最后一个表达式的值。首先是两次赋值使得 a 的值是 3，b 的值是 5，然后 a++即 a 自增 1 变成 4，最后 a+b 即 4+5 等于 9。

3、C 变量变量，当然是可以被改变的；逗号运算符的优先级最低；C 语言中对大小写敏感，大小写不同即表示不同的名称。

4、B 将数学表达式转换成 C 语言的表达式要尤其注意计算时的优先级。每一个变量和常量之间例如 a 和 2 表示相乘关系必须使用运算符*；此外还要通过括号来改变运算时的顺序。

5、C 字符串常量应该用双引号括起来。

6、A 赋值号的左侧必须是单独的一个变量，不能是表达式，不然你赋给谁？C 选项是一个逗号表达式，而且其中的 a++不属于赋值表达式。

7、B 字符本身代表一个整数，参与运算的时候是所代表的整数去算，具体对应请看书附录的 ASCII 码表。A 这个字符代表 65，加 3 得 68 对应字符 D，那到底结果是整数还是字符呢？题目告诉我们 c 是一个字符变量所以最后输出字符 D。

8、D 记住。

9、A 考察数据类型的转换，不同类型的数据参与同一个计算的时候会自动转成别的类型，具体怎么转请参照课本 31 页。本题因 f 是双精度实型则别的 int 会直接转换成这个类型，即 double。

10、B 求余运算的除数必须是整数。

11、C void 和 int 都是内置标识符已经被用掉了，2nd 开头是数字所以不合格。具体规则请看课本第 6 页。

12、C 合法的字符常量，A 中原本以为是 8 进制数，即课本 24 页所示的\ddd 形式，但是它出现了 8，八进制怎么可能有数字 8 呢？直接 PASS；B 中单引号内出现了两个字符；D 是个字符串；C 是一个 16 进制的整数，同样使用了转义符\xhh 的形式。

13、D 赋值语句结合性从右向左，先算赋值号右侧的，a*4=16，然后算中间 a-=16 也就是 a=a-16 即 a=4-16 也就是-12，注意此时 a 从 4 变成了-12，最后算开头 a+=-12，即 a=a-12 即-12-12 最后 a 等于-24。

14、A 变量前面加(int)，就是把变量的类型强制转换成括号里的类型。实数强制转换成整数会丢失小数部分，所以最后两个整数部分相加得 12+13=25。

15、B 例如一个数 1234 要取出它得第二位即 2，挨个代入计算即可。1234%1000=234，234÷100=2.34，因为是整型变量，小数部分被舍弃，得到了 2。

16、C 先判断表达式 8>4？没错执行 a++，先把 a 得结果输出然后 a 再自增 1，最终结果为 8。

17、A 先算括号里的，括号里是个逗号表达式，一通计算猛如虎，最后问 c 结果是什么，发现 c 没有被赋值，还是初始值 2，算了个寂寞。

18、A 先算赋值号右侧得 b+1 为 7，最后 a%=7 也就是 a=14%7 结果为 0。

19、B 记得一定有括号，不然运算顺序就错了。

20、C sizeof 函数用于得知括号内事物所占用的内存数量，最终结果是个整数。

21、C 这个 a---b 到底是(a--)-b 还是 a-(--b)呢？按理来说无法确定，但是我使用了多个平台去运算发现结果是一样的，所以我大胆的给出结论，C 语言

会因自增自减运算符的优先级更高而优先匹配自增自减运算符。所以是(a--)-b 结果为 2。

22、D　别除非了赶紧看题，前三个选项都可以在课本第 24 页找到对应，D 选项里面出现了数字 8，装什么八进制？甚至还有 9。

23、C　输出--x 得结果，先减去 1 然后输出，那么就是 24-1=23 了。

24、A　符号&&是并计算，符号两侧得值如果都是真那么结果为真，输出 1，一个是假结果是假，输出 0。C 语言中定义 0 是假，非 0 就是真。所以本题左边得字符和右边的表达式都是成立得，结果自然为 1。

25、D　似曾相识的第 10 题。

26、B　a 除 b 是整型变量相除结果不要小数得 2，再加 1 就是 3，1 得小数部分也不要了。为什么？三个变量都是 int 型哦。

27、C　如何判断一个数是偶数，把它除以 2 看能整除不能。我们要判断，而不是赋值，所以用两个等于号来判断，这种判断的式子经常写在 if 的条件里。判断为真结果为 1，反之结果为 0。

28、B　这，不解析了吧。

29、B　3 大于 2 众人皆知结果为 1，1 不等于 1 吗这像话吗，不成立所以结果为 0。

30、A　我们来分析一下选项的优先级，A 选项分别为 2、3、5、11；B 选项分别为 9，14，7，12；C 选项分别为 14，9，5，3；D 选项分别为 2，6，11，3。数字越低优先级越高，所以答案显而易见。这些数字来自课本 54 页，你不会不知道吧？

二、填空题

1、sprt(pow(y,x)+log10(y))

课本第 424 页有各种数学函数的使用方法。还是要注意括号导致的计算优先级问题。sqrt 对括号内表达式开方，pow 是第一个参数的第二个参数次方，log10 是求括号内以 10 为底的对数。

2、36

a 先被赋值为 6，然后 a*4 算个寂寞，最后 a+30，也就是 6+30 得 36。

3、6（VC6 结果）9（其他编译器结果）

先算括号里的，把各自的 i 拿出来先进行相加得到结果 6，这是 VC6 的运算方式，其他编译器的运算方法是 2+3+4 得 9。如果问 i 最后等于多少，应该是 5。问赋值表达式的值其实就是问被赋了什么值。

4、3 3（VC6 的结果）4 3（其他编译器结果）

自增优先级高，先算右边的，先把 i 拿出来进行输出，然后对 i 自增 1 变成 4。注意此时 i 变成了 4，再算左边的，直接输出 i 为 4，所以结果是 4 和 3。因为 VC6 的编译器版本比较老，运算顺序可能和新版编译器结果不一样。

5、6

逗号之前是赋值表达式，逗号之后算了个寂寞。所以 x 仅被赋值为 6 了，就没了。

6、36

先算 2+4 为 6，原式等于 a=a*6，结果为 36。

7、int x=8,y=8;

不允许在变量定义初始化时使用连等 x=y=8。不是变量定义的时候可以这么写。

8、1

顺着算，i 变成 2，i 变成 3，算个寂寞，3||3-9 左右两侧都不是 0，或运算有 1（非零）结果就是 1，两个都是 1（非零）了那直接结果为 1。还是那句话，逗号表达式的值是里面最后一个式子的值。

9、E

要求输出一个%c 也就是字符，c 是字符 A，整数对应 65，加 4 那就是 69，对应字符 E，其实没必要这么麻烦，你字母加一个数直接往后推几个字母就好了。

10、7 9 6 8

a++和 b++都是先把值赋值后再自增，所以 x 和 y 分别是原来的 6 和 8，a 和 b 自增变成了 7 和 9。

11、6

0x12 是十六进制数，对应十进制的 18，18-12 得出结果。

12、66 96

一个是先加再输出，一个是先减再输出。

13、240

这个题和上个题都是在考字符对应的整数值是多少。查一下表，记住数字字母的常用位置，算一下就好。

14、1 1 -1 -1

本质是取余计算结果的符号和被除数相同，不知道？没好好看书吧少年！

15、5 2 6

整个表达式的值是 j++，先取 j 给到结果，然后 j 自增变成了 6，i 在此前自增变成了 2。

16、1

从左到右依次看，并计算结果为 1，大于计算结果为 1，或计算左侧为 1 右侧为 0 最终结果为 1。

17、!

答案是个英文感叹号，因为 65-32=33 对应的字符的确就是一个英文感叹号，题目要求%c 输出字符，这个百分号加个 c 是格式控制符，第三章的知识。

18、2

这是一个条件表达式，7>3 当然成立然后执行冒号左边的除法，整型舍弃小数部分。

19、(a<100&&a>0)&&(a%3==0||a%7==0)

三个且一个并，满足了 100 以内、正整数、被 3 或 7 整除这些条件。

20、A（VC6 的结果）B（其他编译器结果）

大写对应的整数比小写对应的整数要小，条件成立为 1，再加一个字符 A 那就是字符 B 啦。

至于 VC6 为什么是这样我就不得而知了。

写解析太累了……-_-

三、程序分析题

1、 2 7
2 7
4 9
4 4
12 12

前三个 printf 的答案很确定，后两行里面又出现了三个连加或者连减，按理来说无法确定怎么结合，不同的编译器可能有不同的处理方式，纠结它也没有意义，你会去写这样的代码吗？当然不会啦。

话是这么说，但是我发现很多个编译器的结果都是一样的，故我给出以下结论：多个自增结合性从右到左没有问题，先算右边的--y-x，得到 9-1-4 为 4，再算左边的，可能是因为自增自减运算符的优先级比较高，所以从左到右优先匹配自增自减运算符，即 y-- - x，得到 8-4 为 4。最后一个 printf 遵循相同的原则，结果为 12 和 12。

2、3 1 4 0
1
16

第二个它计算了两个式子，但是只有一个%d，可能是题目忘了，第二个式子的结果是-6。

3、100 d 68 D
97 a 49 1

题目中的\t 代表是 TAB,也就是制表,做分割作用，效果就像是很多个空格。题目本身前面的解析已经对这种字符加减数字做过解释了，就不赘述了。可能有个字符加减字符比较陌生，本质还是整数加减整数，看最后显示数字还是字符了。

4、0 1 0 1

表达式成立就输出 1，不成立就输出 0。

5、2 4 6 7
-0.580000 7.000000
4
6.580000
7.500000

在 print 的%f 格式控制符中，输出的小数位数有 6 位。但并不保证他们都是有效数字，具体在课本第 75 页。计算过程偷个懒，我就不写了，也不难。我详细你可以写得出来，如果有问题，可以直接来问我们。

6、3 3 1 （后两个数字是第二个 printf）
2 1
13 13 0
8
2 32

第二个 printf 在 VC6 可以运行,是上面的结果。在 VSCode 和 CodeBlocks 中无法通过编译，原因是赋值号左侧不能是--x, 在 DEVCPP 中第一行结果为 3、2 和 1。

我们按照最新 C 语言编译来把第二个 prinf 视作错误表达式，不去探讨他的运算过程。

这道题值得说的是 a 与 b 的四个位运算：十进制 4 和 9 进行位运算首先转换成二进制 0100 和 1001，异或运算，同 0 异 1，得 1101 转换为十进制为 13；或运算，有 1 则 1，全 0 为 0，得 1101 转换为十进制为 13；与运算，全 1 为 1，有 0 则 0，得 0000 转换为十进制为 0；最后对 b 即 0000 0010（一个字节为 8 位二进制数，C 语言中一般用 4 个字节存储一个 int 型数据，我这里简写为 8 个二进制数）左移 4 位,高位舍弃向左移动 4 位,低位补 0,得 0010 0000，结果为 32。

7、6
18 30
18
1

主要考察逗号表达式的最终结果是最后一个式子，先把 6 赋值给 x，然后 18 又赋值给 x，后面两个是算的寂寞，y 的值是 x*2-6 这个表达式的值。最后一个 printf 是输出一个判断是否等于的结果，等于的话就是 1 不等于就是 0。

8、1 1 0 0

注意运算的优先级和结合性即可。第一个是三个非

0 相并结果是 1，第二个是 1 取反为 0 等于 5 取反为 0 左右相等最终结果为 1，第三个从左往右开始 0 或 5 得 1，1 再并 0 最终得 0，第四个 0 并 4 得 0 或上 z=字符 8，因为字符 8 和整数 8 不相等，最终得 0。

四、改错题

1、①定义变量 x 后未对其进行初始化；可以把 printf 函数改成 scanf 函数进行赋值，也可以直接在函数定义处为函数赋值；

②变量 x 是 double 类型，输出时应用格式控制符%f；

③变量定义行末尾缺少了分号；

④char 为字符变量，字符串不能赋值给字符变量；

⑤定义变量并初始化时不可连等；

⑥变量名 3c 和 a+b 非法；

2、①short 类型即短整型的取值范围是-32678~32678，题中 38000 超越此范围会导致数据丢失；

②题目使用#define 定义了一个常量 PI，又对 PI 进行赋值，常量定义后不可改变；

③变量 y 是 double 类型，数值为实数，x%y 中的取余运算除数必须是整数；

④赋值运算符/=的左侧必须是单个变量，题目中（b+c）无法被赋值。

五、编程题

注意：编程题是没有标准答案的，代码中的各种变量名都可以按照自己的想法去定义，不唯一，只要整个程序运行后能够满足题目要求的条件即可。

1、

```
#include <stdio.h>
void main()
{
    int hour, min, sec;
    float v, s, lc_start, lc_end, l;
    printf("请输入开始、结束的里程数和所用时分秒:
");
    scanf("%f %f %d %d %d", &lc_start, &lc_end, &ho
ur, &min, &sec);
    l = lc_end - lc_start;
    s = hour + min / 60.0 + sec / 3600.0;
    v = l / s;
    printf("平均速度为: %5.2f km/h\n", v);
}
```

首先使用 scanf 获取我们所需要的五个数据，经过分析可知最终计算速度需要的是路程和时间两个数据，对路程进行最终减去开始的计算得到最终路程，对时间进行全部转化为小时的运算得到最终时间，最后路程除以时间就是平均速度，输出即可。

2、

```
#include <stdio.h>
#define PI 3.14
void main()
{
    float r, h, v;
    printf("请输入圆锥体底半径和高: ");
    scanf("%f,%f", &r, &h);
    v = (1.0 / 3) * PI * r * r * h;
    printf("体积为: %f", v);
}
```

首先要学会如何定义一个常量，此外要注意的就是三分之一在 C 语言中要写成 1.0/3，如果写成 1/3 那么按照整数运算会舍弃小数部分最后结果会变成 0，或者写成 1/3.0，或者两个都加小数，让系统认为这是一个实数计算而不是整数计算。

有关 printf 和 scanf 函数的使用在课本的第三章知识，如果你刚刚学完第二章，先理解代码的意思即可，具体如何写，学着学着就会明白。

第三章

余传钦

答案速对　1-5　DDDBC　6-10　ABCBD

一、选择题

1、D　printf("%d\n", z = (x % y, x / y)); z 的值由表达式 x/y 决定，故 z 的值为 3。

注意 printf 函数是从右往左计算表达式的(VC++6.0 至少是的)，但输出时是从左往右输出的。

2、D　A 选项，没有分号；B 选项，语法错误；C 选项，没有分号。

3、D　A 选项，语法错误；B、C 选项，没有使用取地址符“&”。

4、B　C1 赋值为单个字符 d，对应 ASCII 的数值为 98，c2 赋值为单个字符 e，对应 ASCII 的数值为 101；再根据 c2-32 的值对应 ASCII 表中 E 故选 B 选项。

5、C　printf 语句中，确定输出宽度为 10 位，精度为两位，向右靠，故选 C 项。

6、A　对输入数字没有做处理，故输出结果也不变咯。不要受 unsigned int 类型的干扰，它的范围你还记得吗?

7、B　有赋值语句，n=6*4；其后的 n+6，n*2；并不是完整的赋值语句，并没有赋值于 n。

8、C　0xabc-0xabc=0。当结果为 0 时直接输出 0，不管你是什么进制的 0，而如果不是 0，那么就按照格式控制符控制的进制输出，这里要注意%x 和%X 控制输出的十六进制结果有大小写的区别。

9、B　A 选项中，b 没有赋值；B 选项，正确；C 选项，没有取值符；D 选项中，完整输入所需字符后没有终止输入，只有继续输入一个空白字符后，才能终止输入。

10、D　在格式控制符中，用一个%表示格式控制，如果想输出百分号的话，两个连着的百分号输出一个百分号，本题中计算机读取到两个百分号，然后向屏幕输出了一个百分号，接着继续读取到了 d 便输出了原样的 d。

二、填空题

1、一条语句 ；

第 69 页有原句。

2、小于 左 右端

参考第 77 页。

3、%%

第 74 页格式表最后一个。

4、普通字符 格式说明

课本第 71 页讲了三部分，格式控制符里的普通字符原样输出，而格式说明根据参数表列完成替换输出。

5、取地址　取 a 的地址

参考第 78 页。

6、从键盘上读入一个字符

参考第 70 页 getchar 函数介绍。这里介绍一下键盘缓冲区的概念，getchar 函数运行后会优先从键盘缓冲区里收入字符，如果键盘缓冲区为空，则等待用户输入，用户输入的字符会被送入键盘缓冲区中，里面包含了所有用户在键盘上按下的键位，包括回车。所以严格地说，getchar 函数是从键盘缓冲区读入一个字符。

7、花括号

在第 69 页复合语句的介绍。

8、f=68.000000

printf 函数格式控制%f 将会默认输出六位小数，但是依据不同的数据类型，其六位小数的有效数字不做保证。

9、n1=%d\nn2=%d

根据输出的结果推出要有换行符。

10、7,5,c=3

%f 的输入可以不需要输入浮点型，直接输入整型也可以完成输出；在 scanf 函数中，除格式说明外有其他字符，在输入数据时，这些普通字符要原样输入。

三、程序分析题

1、i=100,c=a,f=1.234000　i=1,c=.,f=23456.000000

系统将会自动按照 scanf 的格式控制符和普通字符来提取需要的内容，例如 100 给到 i，碰到了 a 就给到 c，碰到了实数就给到了 f；第二问中系统碰到了小数点，这也算一个字符，所以得到上述结果。

2、i=65535,j=65536

i = n++中先赋值 n 再自增；同理 i = n--;中先赋值再自减。

3、1234,123.5,12345.5

在调用 printf 函数输出数据时，当数据的实际位宽大于 printf 函数中的指定位宽时，按实际长度输出。第二个空根据课本第 73 页，对于数字长度超过的部分作四舍五入处理。

四、编程题

1、

```c
#include <stdio.h>
int main(void)
{
    int num1, num2;
    scanf("%d%d", &num1, &num2);
    int temp;

    temp = num1;
    num1 = num2;
    num2 = temp;

    printf("%d\t%d", num1, num2);

    return 0;
}
```

本题非常简单，仅需要交换两个变量的值即可，需要借助第三个变量来存储临时的值，在 C 语言中仅靠两个变量是无法单独完成交换的。

此外，我们建议各位准程序员在写代码时将有明显功能区分的代码进行分块，用空行来分割他们，以便于后期维护，写注释也是很重要的哦。

在本参考代码中，变量的定义并没有全部集中在函数开始时定义，其实在 C 语言的早期版本所有的变量必须在开始时全部定义，不允许函数中间突然定义一个变量，而现在的 C 语言版本是支持这样的操作的。

2、

```c
#include <stdio.h>
int main(void)
{
    int num;
    scanf("%d", &num);
    if (num > 999 && num < 10000)
    {
        int a, b, c, d;
        a = num / 1000;
        b = num % 1000 / 100;
        c = num % 100 / 10;
        d = num % 10;
        num = a + b * 10 + c * 100 + d * 1000;
        printf("%d", num);
    }
    else
        printf("输入的数字有误！");
    return 0;
}
```

本程序中，我们需要首先分别获取到这四位数字分别是什么，这样的算法其实我们在第二章的题目中就做过，相信你还记得。代码中将四位数字分别赋值到变量 abcd 中去，然后根据题目要求重新计算这个数字。这里，其实你还可以直接四个%d 直接连续输出这四个变量。

3、

```c
#include <stdio.h>
#include <math.h>
int main(void)
{
    float a, b, c;
    scanf("%f%f%f", &a, &b, &c);
    float s = (a + b + c) * 0.5;
    float area = sqrt(s * (s - a) * (s - b) * (s - c));
    printf("%f", area);
    return 0;
}
```

提前预告一下，这道题在第四章的编程题中会再次出现，到时候会让你使用 if 语句判断输入的三角形边长是否能构成三角形。

计算面积需要知道计算公式，如果你只知道有底和高的计算公式，那并不适合本程序，因为我们仅知道三条边长。百度得公式后按照题目要求计算然后输出即可。

4、

经过前三题的历练，本题的代码一定难不倒大家。

①两者皆可，getchar 函数可以读取一个字符，赋值给字符型变量或整型变量

②要输出字符对应的 ASCII 码时，则需要用到 printf 函数，用%d 作格式说明。

③不是，大多数情况下，字符型与整型可以通过 ASCII 码进行转换，但并不是绝对的，因为 ASCII 表仅有 255 个字符。

第四章

邬山峰

答案速对

1-5 CBBBD 6-10 DDABA 11-15 BBDAC 16-20 ACCBD 21-24 ACAA

一、选择题

1、C 别问，问就是 if 函数嵌套。

2、B 浮点型输出，小数位保留两位，不足补零。

3、B 这也没啥好说的，自由 C 语言，愣着干啥，画知识点。

4、B if else 基础知识，跳过。

5、D 逻辑且、或运算，如果符号前面的表达式真假，能直接确定整个表达式的真假，那么后面表达式将不被执行，所以 k++不做运算，类似于短路效应，书 P42 页有知识点解析。

6、D ABC 都是等于零输出 y，就 D 输出 x。（BD 辣么像肯定有一个是答案啦）

7、D D 是两条语句。

8、A 题目意思很明显，要满足 a>b 和 b>c 才能输出 1，不然输出 0。

9、B　连续使用关系运算符进行判断时要注意，如本题 a>b>c 实际上为（a>b）>c，a>b 为真输出 1 ，所以为对 1>c 的判断，在题中为假。所以 if 后面的表达式不执行。

10、A　15 对 3 求余等于 0 啦，所以直接进 case 0，然后 break 出来嘛，输出 m=m+1 结果=1，选 A。

11、B　5<5 是不成立的，所以转向 else，由于 if 里面是 x--所以在这里 x 值变成了 4，x++是在输出了 x=4 的值后再运算，所以选 B。

12、B　这题跟上题类似，9<10 所以进入 if 后面的表达式，输出 n 的值，但是因为在 if 判断里有++后缀，所以此时 n 的值要加 1。

13、D　又是一题基础题啦，A 没 default，B case 3 重复，C switch 后面少了一个括号，这些小错误在写程序的时候别犯。

14、A　因为 x=1，所以从 case 1 开始执行，而且由于没有 break 会一直执行 case 2 ，所以 a++进行了两次，值变为 2，b++进行一次变为 1。

15、C　x<0.0 不成立，进入下一个 if，x<10.0 成立所以 y=1.0/x=1/2=0.5 很简单是吧，最后注意一下浮点格式就行。

16、A　y= (x>0?1:x<0?-1:0) == (x>0?1:(x<0?-1:0))，原理来自书 49 页条件表达式的嵌套。回到本题，题目的功能是当 x>0 时 y=1；当 x=0 时 y=0；当 x<0 时 y=-1，直接 ABCD 代入不同的 x 即可。B 选项当 x 等 0 时不执行任何代码，y 无法被赋值；C 选项当 x 等 0 时 y 等-1 不满足题目；D 选项同 C 的原因。

17、C　++a<0 为假，而且&&一假全假，所以后面的 b--不会被执行（不懂的看书 43 页例 2.16 上面小字部分），so 正常执行输出。

18、C　A 的 case 后面必须是常量表达式，不能为浮点型，这是规定!!（小本本在哪，拿出来记啊）B 的 case 后面只能有一个整数；D 的 switch 后面没分号；

19、B　正常按顺序执行，因为后面会对 x 重新赋值，所以 if(c)前面的 if 语句都是烟雾弹，又因为 c=0 所以进入 else x=4，选 B。

20、D　y+z=0 啊!! 所以 if 判断里面为假直接进入 else。

21、A　猜猜这题的 else 是与哪个 if 配对的？答案是 if(!ok1)，然后!ok1 为 0 所以进入 else，又 ok2=0，所以 x=10 不执行，x=-1，选 A。

22、C　按顺序执行，x<5.0 为真，x!=2.0 为假，所以 if 为假，跳过进入下个 else if，y=1.0/x，选 C。

23、A　先进入 case 1 进入 switch（y）的 case 0，然后退出 switch（y），此时由于 switch（x）的 case 1 没有 break 语句，所以将继续进入 case 2，得出 a=2，b=1，选 A。

24、A　简单的 if else 语句啦，肯定不会错的对吧……

二、填空题

1、1

首先进入 printf("%d\n",a>100);（没问题吧，有问题的回去看书…… ），然后注意 a>100 是一个关系表达式，它的值为 0 或 1（问就是规定 ）。真为 1 假为 0，显然这里为 1。

2、5 8 8

x>z 为假对吧，所以 if 后面的表达式不被执行。但是在这里要注意：在 if 后面的表达式只有 y=x；y=x;x=z;z=y;他们是有分号隔开的，是三个单独的语句别弄混了。

3、4599

if（a）a 为非零整数，所以执行后面的 printf。

4、2

由题知 a!=c 所以 if 条件句为假，执行 else。

5、10 20 0

这里有逻辑或（||），所以 c 的值为 0 或 1。0 为假 1 为真。

6、2　1

这不是选择题 23 题变填空题嘛（蜜汁操作，值都不改一下），不会的回去看 23 题解析。

7、-4

相信执行到 if(m) x-=3;大家都会，重点在 if(x) x-=3;这后面表达式要不要执行，到这里时 x 值为-1。注意-1 也是非零的，非零即为真，所以 x-=3 还是要执行的。

8、3

这题主要在后面的条件表达式上面，按程序一步步判断求值就行。

9、No

c!=a+b 判断式为假，进入 else。

10、25

条件表达式 x>12 所以执行 x+10=25。

11、4545

由于题目为单分支 if 语句，所以主要条件满足则一直执行。

12、0

这题跟选择题第九题类似，错了的自觉回去再看一遍。

13、5.5

这题就是用结果逆推条件：z=a/2+b*x/y+1/2 看得懂吧？已知 a,b,z,y 求 x 很简单，要注意 1/2=0。

14、13

从 case 10 开始由于没有 break 所以会一直执行到结束。

15、3

简单的 if 判断，y<0 不成立，所以进入 else，输出 z+=1。

三、程序设计题

1、

```
#include <stdio.h>
int main()
{
    int n;
    scanf("%d", &n);
    if (n % 7 == 0)
        // 这里用 n 对 7 求余判断，为零则 if 语句判断里面为真输出 YES
        printf("YES");
    else
        printf("NO");
}
```

2、

```
#include <math.h>
#include <stdio.h>
int main()
{
    double a, b, c, s, area;
    scanf("%lf,%lf,%lf", &a, &b, &c);
    if (a < (b + c) || b < (a + c) || c < (a + b))
    //用逻辑或进行三角形两边之和一定要大于第三边的判断
    {
        s = (a + b + c) / 2;
        area = sqrt(s * (s - a) * (s - b) * (s - c));
        //这两行是三角形面积的计算代码
        printf("area=%lf\n", area);
    }
    return 0;
}
```

3、

程序一：优秀

多分支 if 结构只会输出第一个满足的条件的结果，有刹车。

程序二：优秀良好中等及格

这种程序里单分支 if 结构会造成只要条件满足就一直输出的情况，相当于飙车没刹车。

4、

```
#include <stdio.h>
void main()
{
    int x;
    scanf("%d", &x);
    switch (x)
    {
    case 1:
        printf("Monday");
        break;
    case 2:
        printf("Tuesday");
        break;
        //标准的 switch 结构，主要注意 switch 的格式
    case 3:
        printf("Wednesday");
        break;
    case 4:
        printf("Thursday");
        break;
    case 5:
        printf("Friday");
        break;
    case 6:
        printf("Saturday");
        break;
    case 7:
        printf("Sunday");
        break;
    default:
        printf("Error");
    }
```

```
}
```

5、

```
#include <stdio.h>
void main()
{
    int m, n;
    scanf("%d", &m);
    switch (m / 10)
    //if 改 switch 主要在 switch(n)的 n 上面
    //改好 n 就直接按标准 switch 格式输出就行
    {
    case 0:
    case 1:
    case 2:
        n = 1;
        break;
    case 3:
        n = 2;
        break;
    case 4:
        n = 3;
        break;
    case 5:
        n = 4;
        break;
    default:
        n = 5;
    }
    printf("%d", n);
}
```

6、①不嵌套的 if 语句

```
#include <stdio.h>
void main()
{
    float x, y;
    scanf("%f", &x);
    if (x < 0)
        y = x;
    if (x >= 0 && x < 10)
        //不嵌套 if 语句就在条件里面加一个逻辑且（&&）来保证 x 范围
        y = x - 10;
    if (x >= 10)
        y = x + 10;
    printf("%f", y);
}
```

②嵌套的 if 语句

```
#include <stdio.h>
void main()
{
    float x, y;
    scanf("%f", &x);
    if (x >= 0)
        if (x < 0)
            y = x - 10;
        else
            y = x + 10;
    else
        y = x;
    printf("y=%f", y);
}
```

嵌套的 if 语句很容易理解，就是 if 里面有 if，注意搞好条件判断就可以了。

③多分支 if 语句

```
#include <stdio.h>
void main()
{
    float x, y;
    scanf("%f", &x);
    if (x < 0)
        y = x;
    else if (x < 10)
    //标准多分支 if 语句,注意 else 和 if 的搭配和条件的范围改变
        y = x - 10;
    else
        y = x + 10;
    printf("%f", y);
}
```

7、①使用多分支 if 语句

```
#include <stdio.h>
int main()
//按照题目要求使用标准多分支 if 语句
{
```

```
    int n;
    scanf("%d", &n);
    if (n < 60)
        printf("E");
    else if (n < 70)
        printf("D");
    else if (n < 80)
        printf("C");
    else if (n < 90)
        printf("B");
    else if (n <= 100)
        printf("A");
    else
        printf("Error!");
}
```

②使用 switch 语句

```
#include <stdio.h>
//标准 switch 语句，只是要注意 100 成绩判定的问题
int main()
//在 switch 语句没想到什么好办法解决
//干脆在外面加了一个 if 语句限制
{
    int x;
    scanf("%d", &x);
    if (x > 0 && x <= 100)
        switch (x / 10)
        {
        case 0:
            printf("E");
            break;
        case 1:
            printf("E");
            break;
        case 2:
            printf("E");
            break;
        case 3:
            printf("E");
            break;
        case 4:
            printf("E");
            break;
        case 5:
            printf("E");
            break;
        case 6:
            printf("D");
            break;
        case 7:
            printf("C");
            break;
        case 8:
            printf("B");
            break;
        case 9:
            printf("A");
            break;
        case 10:
            printf("A");
            break;
        default:
            printf("Error");
        }
    else
        printf("Error");
}
```

8、

```
#include <stdio.h>
//多分支 if 语句或者 switch 语句都可以,这里用的多分支 if
int main()
//整体来说这些编程题都不是很难，大多前面都有例题借鉴
{
    int x;
    scanf("%d", &x);
    if (x == 0)
        printf("");
    else if (x == 1)
        printf("高等数学");
    else if (x == 2)
        printf("程序设计");
    else if (x == 3)
        printf("高等数学");
```

```
    else if (x == 4)
        printf("程序设计");
    else if (x == 5)
        printf("外语");
    else if (x == 6)
        printf("政治");
    else
        printf("输入错误");
}
```

第五章

余传钦

答案快对

1-5 CADCD　6-10 B "A A/B" ABA　11-14 BCBC

一、选择题

1、C　k 的初始值为 10；而 while 的循环条件为 k=0；不符合循环条件，所以循环体语句不执行。

2、A　初始值 a = 1, b = 2, c = 2，循环条件中 a<b<c; 先看 a<b 为真，故其值为 1，1 小于 c，所以继续向下执行，得到 a= 2, b = 1, c = 1。此时 a<b 为假，其值为 0；0 <1，继续执行循环，经过第二次循环，a = 1, b=2, c=0，故选 A。

3、D　x=y=0；根据执行语句中，先 y++后 x=x+（++y），逐步计算，可得第一次循环结束，y=2,x=2; 第二次结束 y=4,x=6；第三次结束 y=6,x=12；第四次结束 y=8,x=20。

4、C　让我们跟着程序走一遍，首先 0<=2 成立，n 变成 1，然后 1<=2 成立，n 变成 2，然后 2<=2 成立，n 变成 3，然后 3<=0 不成立，跳出循环，别忘了 n 变成了 4。

5、D　while 语句中有语句，而且没有判断，我们将其默认为循环条件永远成立，故而该循环不会终止，在下面程序运行后，if（t<3）break; 中可知，输入时 t=1，break 语句跳出循环。

6、B　根据题中所述，当 a=b 时跳出循环可知要继续循环则需要 a!=b。

7、A　A/B

第一空中，需要将一个字符赋值于字符变量 c，根据赋值语句的使用规则可知选 A，第二空中，n:m 与 m:n 在语法都没有错误，区别在于前者输出最大值，后者输出最小值。

8、A　代码中首先用 while 语句依次读取题目所输入的 2473，每次读入一个字符后开始以下循环：switch 语句中有 c='2'，将字符 2 赋值于字符变量 c，字符 2 所代表的整数是 50，其不属于 case 0,1,2,3 中的任何一条，故跳至 default 执行，即输出 c+2 这个字符，刚才已经知道 c 对应的整数是 50，加 2 后得 52，根据 ASCII 表得知 52 对应的字符是'4'，所以输出了 4，因为 while 语句循环 4 次，答案为 4444。

9、B　由题意可知，当 a 输入值为 0 时，程序结束，当 a 为其他值时程序执行，由此判断选 B。

10、A　y 的初始值为 10，判断语句为--y，故当 y 值为 1，--y 为 0，程序终止执行，在输出语句中，输出值为--y 可知输出值为-1。

11、B　根据循环条件，b--<0 ,在执行完第一条语句后 b = 9，在 while 中 b=8,此时 a = 2。

12、C　判断语句中，y=123，赋值语句，不做判断依据，默认为 1；其后 x< 4;推出当 x 自增为 4 是循环终止，可知循环 4 次。

13、B　A 选项当 I>100 时跳出循环，但 I=I%100+1，执行过程中，I 的值不可能大于 100；C 选项 k 初始值为 0，其后每次自增为 1 ，可知 k 值永远大于 0，故循环不终止；D 选项中 while(s) ；执行一个空语句，s 的值不变，循环一直执行。

14、C　I++<4 与++I<4 的区别是，前者是 I 先与 4 比较大小，后自增；后者是 I 先自增后与 4 比较大小。

二、填空题

1、{r = m; m = n; n = r}　m%n

用辗转相除算法求最大公约数，由后面 m%n 可知，m>n，故当 m<n 时，将 m 与 n 的值互换。

2、3

a-=2+b 等价于 a=a-(2+b)。

3、a=-5

依次计算得 a=-5，注意 printf 中的 a=也要原样输出。

4、i%3==2&&i%5==3&&i%7==2　j%5==0

第二空中判断循环个数，用 j 记录，当输出五个字符时，自动换行，但由于输出个数不确定，所以用 j%5==0 做判断更为稳妥。

5、i==j　i!=k&&j!=k

因为需要三个不同的数字，所以要判断三个数字是否相等。

6、j=1　i+j+k==8

i 模拟红球个数，j 模拟白球个数，k 模拟黑球个数；因为必须有白球所以 j 起始值为 1；最后计算总共拿

出的球个数为 8。注意本题莫名其妙的出现了一个变量 s，题目没有定义它，这里是题目的问题，我们可以补充定义，也可以把 s 改成 8。

7、8

s=s+(i++) 且当 s 的值为 7 的倍数时，++i 不执行；可得 i 在 s+=i++中执行 4 次，在 else++i 中执行三次，所以 i 最后为 8。

	s	i
第 1 次循环	3+1=4	自增 2 次为 3
第 2 次循环	4+3=7	自增 1 次为 4
第 3 次循环	7+4=11	自增 2 次为 6
第 4 次循环	11+6=17	自增 2 次为 8

8、k&&i<=500　k/10　continue

当判断语句中 k 值不为 0 时且 k 小于等于 500 时，程序执行，各位数之和，我们可以从最后一位数开始一个一个加，所以执行语句为 s=s+k%10;k/10; 最后当 s 的值不为 5 时，我们需要将其舍去，不进行计数，所以用 continue 跳过其后的语句。

三、编程题

1、

```
#include <stdio.h>
int main(void)
{
    char c;
    int i;
    while ((c = getchar()) != '\n')
    {
        if ('A' <= c && c <= 'W')
            for (i = 0; i < 3; i++)
            {
                c += 1;
                putchar(c);
            }
        else if ('a' <= c && c <= 'w')
            for (i = 0; i < 3; i++)
            {
                c += 1;
                putchar(c);
            }
        else if ('X' <= c && c <= 'Z')
            for (i = 0; i < 3; i++)
            {
                if (c == 'Z')
                    c = 'A';
                else
                    c += 1;
                putchar(c);
            }
        else if ('x' <= c && c <= 'z')
            for (i = 0; i < 3; i++)
            {
                if (c == 'z')
                    c = 'a';
                else
                    c += 1;
                putchar(c);
            }
        else
            putchar(c);
    }
    return 0;
}
```

首先分析题意，输入一串字符，用 c=getchar(), 循环读取输入的字符，当读取到换行时停止读取；英文字母利用 ASCII 表所对应的，将各个字母加 1 就可以得到下一个。我们使用 for 循环语句循环三次即可实现题目所要求的把一个字母替换成其后三个字母。

这里要注意的问题是当循环到 X、Y、Z 这些临界点时，因为 ASCII 码表不是连续的，所以需要我们手动判断当 c=字符 z 时，让 c=字符 a，这是解决问题的思路。程序开发过程中经常会出现意外情况，我们要统筹兼顾，把所有情况都尽量处理到。

2、

```
#include <stdio.h>
int main(void)
{
    char newNum[10000]; //储存转换的字符
    int num;            //需要转换的数
    int R;              //转换进制
    int i = 0;          //记录数组
    printf("请输入要转换的进制 R(R 在 2~16): ");
    scanf("%d", &R);
```

```
    if (2 <= R && R <= 16)
    {
        printf("请输入需要转换的数字: ");
        scanf("%d", &num);
        while (num != 0)
        {
            i++;
            newNum[i] = num % R;
            num = num / R;

            char Hex = 'A';
            if (newNum[i] > 9)
                newNum[i] = Hex + (newNum[i] - 10);
            else
                newNum[i] = newNum[i] + '0';
        }
        for (int j = i; j > 0; j--)
            printf("%c", newNum[j]);
    }
    else
        printf("输入有误! ");
}
```

分析题意，需要将一个十进制数，转换成一个其他进制数，所以在程序开始先要求输入要转换的进制，且限制范围，不在范围内主动报错；在输入正确的值之后，在用我们转换进制的算法，用数字除以 R，保留商，继续用余数除以 R，直至商为 0，在用数组收入各个余数，注意输出时需要逆序输出。当 R>10 时，就需要引入字母，不妨定义一个字符变量，也可以再根据需要用 newNum[i] = Hex + (newNum[i] - 10); 引入其他字母。在用上述算法继续执行程序。

3、

```
#include <stdio.h>
int main(void)
{
    float money;
    printf("请输入您有的多少钱: ");
    scanf("%f", &money);
    int m = money * 100; //数据类型转换 ,以及将输入错误的数字纠正
    int a[10];
    int i = 0, n;
    int type[10] = {10000, 5000, 1000, 500, 200, 100, 50, 10, 5, 1};
    while (m != 0)
    {
        n = type[i];
        int b = m / n;
        a[i] = b;
        m = m % n;
        i++;
    }
    printf("%.2f = 100*%d + 50*%d + 10*%d + 5*%d + 2*%d + 1*%d + 0.5*%d + 0.1*%d + 0.05*%d + 0.01*%d",
           money, a[0], a[1], a[2], a[3], a[4], a[5], a[6], a[7], a[8], a[9]);
}
```

先获取一个金额，可以参考“提取一个整数的各个数位的数字”的算法，所以需要先将这的小数转换为整型，所以先乘以 100，并且转换数据类型；除 n 倒取余，例如：用金额乘以 100 后的数字，除以 100，取出商存入 a[0]，再去余数除以 50，依次进行下去，最终获取各种面值货币的数目。

4、

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(void)
{
    int a[20];
    int max = 0, min = 0, sum = 0;
    float aver = 0;
    srand((int)time(0));
    for (int i = 0; i < 20; i++)
    {
        a[i] = rand() % 41 + 10;
        sum = sum + a[i];
        min = a[0];
        if (min > a[i])
            min = a[i];
        if (max < a[i])
            max = a[i];
    }
```

```
    aver = sum / 20;
    for (int i = 0; i < 20; i++)
    {
        printf("%d\t", a[i]);
        if ((i + 1) % 5 == 0)
            printf("\n");
    }
    printf("min = %d\tmax = %d\tsum = %d\taver = %.
2f", min, max, sum, aver);
    return 0;
}
```

这里需要介绍三种新的函数，srand，rand，time。使用它们需要包含新的头文件：

```
#include<stdlib.h> // srand 和 rand 函数
#include<time.h> // time 函数
```

头文件中还有其他函数，这里就不一一介绍了，有兴趣可以去查阅下。

rand()函数返回一个从 0~32767 之间的随机整数，如果要指定范围，例如取[m,n]范围内的随机数，则用公式 rand()%(n-m+1)+m 表示。rand()函数是以种子(seed)为基准，以某个递推公式推算出来的一系列数(随机序列)，但不是真正意义上的随机整数。当计算机开机后，这个种子的值就已经确认了，关机重启后种子的值也不会变。也就是说一般计算机只用这个函数，所得到的随机数是一样的（伪随机数）。可以这样认为：一个种子对应范围内的一组随机数，种子改变随机数改变。所以就要用到 srand 函数。

函数原型： void srand(int a)

功能：初始化随机种子产生器，即将种子的值改为 a。如果要产生随机种子，可以用下面这个语句

```
srand(time(0));
```

其中 time()函数是包含在头文件 time.h 中其功能是返回一个从 1970.1.1 00.00.00 到现在的秒数。因为每次运行的时间不同，因此产生的随机种子也不同，这样就保证运行时可以得到不同的随机序列。

再分析这道题，需要 20 个随机数，用 srand ((int) time(0));初始化种子，以便得到不同的随机数，返回值是 int 类型。套用公式 rand()%(n-m+1)+m 来确定[10,50]范围的数。本题为以下语句：

```
a[i] = rand() % 41 + 10;
```

5、

```
#include <stdio.h>
int main(void)
{
    int num;
    printf("请输入一个正整数: ");
    scanf("%d", &num);
    if ((num % 7) == 0 && (num % 5) == 0)
        printf("yes");
    else
        printf("no");
}
```

这题需要判断正整数是否既是 7 的倍数，也是 5 的倍数;用 if 语句做一个判断。用逻辑与&&连接，同时成立时，输出 yes，有一个不成立的，则输出 no。

6、

```
#include <ctype.h>
#include <stdio.h>
int main(void)
{
    char ch;
    int countNum = 0, countLetter = 0; //定义记录数
字与字母的变量
    printf("请输入您要输入的字符串! \n");
    while ((ch = getchar()) != '\n')
    {
        if (isdigit(ch)) //数字
            countNum++;
        else if (isalpha(ch)) //字母
            countLetter++;
    }
    printf("字母有%d 个\t 数字有%d 个
", countLetter, countNum);
}
```

这里需要介绍#include<ctype.h>头文件， 这个头文件是对字符类型进行处理，拥有一些对字符处理的功能函数，这里介绍四种函数：sialpha 是否是英文字母；isdight 是否是数字；islower 是否是小写字母；issupper 是否是大写字母。其他函数有兴趣的可以去查阅下。

分析本题，需要连续输入一串字符，但个数不确定，所以用 while ((ch = getchar())!='\n')语句是一种很好的选择，函数的使用方法参考上例。

7、

```
#include <stdio.h>
```

```c
#include <string.h>
int main(void)
{
    char str[1000];
    str[0] = ' ';
    gets(&str[1]);
    int lspace = 0, maxfirst = 0, length = 0;
    //maxfirst:最长单词第一个字符下标,lspace:最近出现的空格或'\0'
    for (int i = 0; i <= strlen(str); i++)
    {
        if (str[i] == ' ' || str[i] == '\0')
        {
            if (length < i - lspace)
            {
                length = i - lspace;
                maxfirst = i - length;
            }
            lspace = i;
        }
    }
    printf("最长单词是:");
    for (int i = maxfirst + 1; str[i] != ' ' && str[i] != '\0'; i++)
    {
        printf("%c", str[i]);
    }
    return 0;
}
```

分析题目，在输入一句英文后，以空格作为分界，在检测到空格时终止记录，不妨设置变量 i 记录来记录从 开始循环 到 检测到空格 时的字符个数，记为单词长度，记录数组下标；重新记录下一个循环，如果下一个单词比上一个长，则重新记录新的单词，并更新数组下标下标，在这里建议数组第一个储存为空格，否则在输出过程中会出现问题，原因是在储存数组下表时是以空格为首字母，当输出时，需要将其省略，如果数组的首位不是空格则会出现：当第一个单词最大时，输出单词会自动去除首位单词。

8、

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(void)
{
    int i, n, A, B, C, D, E;
    printf("请输入参与选举的人数: ");
    scanf("%d", &n);
    srand((int)time(0));
    int ballot[n]; //定义选举票
    for (i = 0; i < n; i++)
    {
        ballot[i] = rand() % 5 + 1;
        switch (ballot[i])
        {
        case 1:
            A++;
            break;
        case 2:
            B++;
            break;
        case 3:
            C++;
            break;
        case 4:
            D++;
            break;
        default:
            E++;
            break;
        }
    }
    printf("A 的票数为:%d\tB 的票数为:%d\tC 的票数为:%d\tD 的票数为:%d\t 无效的票数为:%d\t", A, B, C, D, E);
}
```

这题中同样也用到了随机数，不妨用 1~4 来模拟 A、B、C、D 的投票，用 5 来模拟废弃的票，因此随机数范围设为 1~5；随机数的产生就不再赘述了，参考上题。

这里定义了一个动态数组，根据 C 语言的定义，数组设置的数组长度必须是一个确定的数，所以对 n 必须要先对其赋值，再定义数组。再利用 switch 语

句，分别统计 A、B、C、D 以及无效的票数。

9、

```c
#include <stdio.h>
int main(void)
{
    int start = 0, m;
    printf("请输入一个奇数: ");
    scanf("%d", &m);
    start = m * (m - 1) + 1;
    printf("m^3=");
    for (int i = 0; i < m; i++)
    {
        if (i == m - 1)
            printf("%d", start + i * 2);
        else
            printf("%d+", start + i * 2);
    }
}
```

这题看似有点复杂，实际上只要理清楚自然数 m，与连续奇数中任意一个的关系即可解决，在这题中，我们不难发现，m 与连续奇数中的首个数字之间存在关系，不妨用 start 来表示首位数字，则有 start=m*(m-1)+1；但是输出的数字显然是不确定的，所以必须要使用一个循环来输出各个数字，数字个数为 m，依次增加 2，在如上写出程序即可。

10、

(a)

```c
#include <stdio.h>
int main(void)
{
    char ch = 'a';
    int i, j;
    for (i = 0; i < 26; i++)
    {
        for (j = 0; j < i + 1; j++)
        {
            printf("%c", ch + j);
        }
        printf("\n");
    }
}
```

题中意思明确最后一行输出 26 个字母，所以控制行数为 26 行，每行从 a 开始，依次加一依次循环即可。

(b)

```c
#include <stdio.h>
int main(void)
{
    int i, j, m;
    int num = 0;
    printf("请输入循环层数");
    scanf("%d", &m);
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < 2 * i + 1; j++)
        {
            printf("%d", num);
            num++;
            if (num == 10)
                num = 0;
        }
        printf("\n");
    }
}
```

分析输出结果，输出的层数是不确定的，因此可以自己定义一个；数字是从 0~9，超过 9 时，又重新变为 0，重新循环；注意定义的 num 需要初始化，如果不初始化，会出现不可预见的错误。

第六章

史贝宁

答案速对 1-5 BCDAB 6-10 CBAAD

一、选择题

1、B A 选项中的圆括号不可作为下标，必须用方括号；B 是一个标准的整型数组定义，代表有 10 个元素；C 中定义数组时下标为变量 k，C 语言中不允许定义动态数组，方括号内必须是一个常量，你可能会问 k 不是已经等于 5 了吗？但是计算机在进入函数的时候就会去给数组分配内存，此时代码还没有执行，k 还没有被赋值；D 中的第一个元素内有两个字符 a 和版，字符数组中一个元素仅能放置一个字符。

2、C 和前一道题的解释是一样的，只不过这道题

问的是二维数组罢了。

3、D　第一个下标为 10，数组定义时方括号里的代表数组内的元素个数，第一个元素的序号在调用的时候是 0，一共 10 个元素，那最后一个元素的序号是 9，直接 a[10]会导致数组越界发生错误，因为不存在序号为 10 的元素；不能用圆括号，元素的序号也不是小数。最后一个方括号内为 10-10 这个表达式，也就是 0，代表序号为 0 的第一个元素。

4、D（VC6 的结果）A（其他编译器结果）

代码定义了有 2 个元素的数组 a，并给序号为 0 的元素赋值为 1，第二个元素没有被赋值，初值为 0，计算 k 被赋值为了 a[k]也就是 a[1]乘以 a[0]，也就是 0 乘 1 得 0。

5、B　所有的字符数量包括空格在内一共是 12 个，再加上系统最后补上的\0 结束标志一共占用 13 个字节。

6、C　循环三次，每次循环输出不同的数组元素，他们分别是 x[0][2]、x[1][1]、x[2][0]，对应后写出答案。补充一下，二维数组的第一个方括号是代表行，第二个方括号是代表列。行和列的序号同样是从 0 开始一定要记得。

7、B　gets 函数的参数必须为一个数组的地址，选项 A 的参数是 a，a 即数组名，数组名可代表这个数组的地址；C 和 D 都是取地址自然没有问题；B 选项所代表的是一个元素的值，不符合我们参数要求是地址的条件。

8、A　先来看一下这道题的代码高亮，

```
printf("%d\n", strlen("\t\"\065\xff\n"));
```

貌似这高亮也看不出有几个字符哈，那我们分析一下把：strlen 函数是取长度，我们就看这个字符串里有几个字符即可，\t、\"、\065、\xff、\n 你数一下一共有五个没问题。\开头的一般都是转义字符，课本 24 页列举了很多，虽然他们看起来不算一个字符，但其实算一个。

9、A　数组 c 是一个有着两行四列的字符数组，程序首先把字符串 you 送进了 c 中，没加下标，默认送进第一行，也就是序号为 0 的行，又把字符串 me 送进了序号为 1 的行，也就是第二行，最后又把字符&送进了第一行第四列中，此时，数组 c 中第一行是 you&，第二行是 me 空空，输出数组 c 便是 A。

10、D　gets 和 puts 函数由头文件 string.h 提供。

二、填空题

1、按行顺序存放

课本 141 页的原话。

2、0 9

记住数组的第一个元素序号为 0。

3、0

这是一个三行四列的二维数组，初始化的时候给第一行第一列下标 0 0，第二行第一列下标 1 0，第三行第一列下标 2 0 赋值了，题目问你第二行第二列下标 1 1，并没有被赋值，结果为初始值 0。

4、1 2

一个字符占据 1 个字节，注意中文字符占据 2 个字节，整数占据 4 个字节（各编译器分配的空间不同，大多数是 4），存储字符串 a 意味着会被存在字符数组里，a 本身占一个字节，还有字符串的结束标志\0 也占据一个所以最后结果为 2 个字节。

5、6　这和选择第 8 题基本一样，只不过字符串不一样，看看高亮，数一下有几个字符就可以了：

```
printf("%d\n", strlen("Ab\123\\%%"));
```

开始数数，A、b、\123、\\、%、%一共由 6 个本题结束。

6、gets 可接收空格；对回车符处理不同；返回值不同；可接收数据类型不同

这道题题目写的是输出，写错了，这两个函数都是输入函数，所以各位把题目改成输入。以下是对答案的具体解释，来自百度。

①gets 可以接收空格；而 scanf 遇到空格、回车和 Tab 键都会认为输入结束，所以它不能接收空格。

例如：如果输入为"hello world"时，gets 的运行结果是"hello world"。而如果用 scanf 则只能输出 hello。

②scanf 对末尾回车符的处理：把回车符保留在缓存中。gets 对末尾回车符的处理：接收回车，但把回车替换为\0。

③gets 的返回值为 char*型，当读入成功时会返回输入的字符串指针地址，出错时返回 NULL；scanf 返回值为 int 型，返回实际成功赋值的变量个数，当遇到文件结尾标识时返回 EOF。

④gets 函数仅用于读入字符串；scanf 为格式化输出函数，可以读入任意 C 语言基础类型的变量值，而不是仅限于字符串(char*)类型。

7、strcpy()

字符串常量在定义时可以直接初始化到字符数组中去，但在非定义时不能这么做，不可写 a="abc"此类代码，而应该使用 strcpy 函数来实现，也就是：

```
char a[] = "bcd";
strcpy(a, "abc");
```

8、&a[i] 或 a+i

第三章 scanf 和第六章数组知识，本题循环为数组元素赋值，scanf 接收到数据后需要给到某地址，每次赋值的元素序号不同，用 i 来表示变化中的下标，所以最后&取地址符号加上 a[i]即可。

因为 i 最初等于了 0，序号和我们数组从 0 开始的规则一致。也可以填 a+i，因为 a 本身代表数组的首地址，+1 意味着下一个元素的地址，同样符合条件。

不知道大家注意到了没有，这道题缺少了 i++;

9、k)-1

第一个空很简单，因为下面用到了变量 k 所以需要在最开始去定义一番。

第二个空可能这道题原本是想让我们去填 str，但是打错了，那我们只能填个右括号了。

填完之后发现 strlen 是让程序获取 str 这个数组的长度，不包含最后的\0，假如是对字符串 123456789 进行操作，strlen 结果是 9，而数组的下标最大是 8，所以我们要再减去 1 以免后面的代码在循环到 9 时导致数组越界。

得到的 8 赋值给 j 来确定需要循环的次数，从 0 开始循环到 8 一共 9 次，满足假设的条件。

10、9, 8

sizeof 函数获取的是数组的长度，包括最后的\0，所以一共是 9 个；strlen 函数获取的是数组里字符串的长度，不包含\0，所以是 8 个。请记得最后输出的结果中间是有逗号的，格式控制符里的逗号和空格都应该被原样输出。

三、程序分析题

1、1 3 7 15

循环 4 次，每次循环根据表达式计算的结果进行输出，核心上还是考数组下标的使用，没有什么难度吧。

2、1 2 3

0 5 6

0 0 9

本题计算时把数组中的内容写在草稿纸上跟着代码运算即可，非常需要注意的是，本题的代码缩进会干扰你的判断，正确的缩进是这样的：

```
#include <stdio.h>
void main()
{
    int i, j, a[][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    for (i = 0; i < 3; i++)
        for (j = i + 1; j < 3; j++)
            a[j][i] = 0;
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
            printf("%d ", a[i][j]);
        printf("\n");
    }
}
```

你可能光看题目会认为有 for 里套了三个 for，但是不是的，for 循环语句后面如果没有花括号括起来，那么这个 for 循环语句的循环体就只有它后面的一条语句。这个原则同样适用于 if。

接下来单独解释以下每个 for 的作用。

第一个 for 循环循环 3 次，在这次循环中：

i=0 时 j=1，j=2 循环成立

i=1 时 j=2 循环成立

i=2 时 啥也没有

第二个 for 循环把以上成立的情况中 a[j][i]赋值为 0，也就是有 3 个数字被赋值为 0 了；

第三个 for 循环也是循环三次，每次改变行数：

第四个 for 循环循环三次，每次用 i 这个固定的行，输出变动的 j 来输出每行数组元素的值。

第三和第四个 for 其实就是一组输出二维数组的标准 for 循环。

3、把输入的字符串中的数字单独输出

首先 ctype.h 其实就是 string.h 文件，两者功能一致；

代码中用到了 gets 函数但是题目没有给出输入的内容，所以答案用文字描述了会产生的结果。

回头看程序，先定义了 c 和 d 两个长度为 80 的字符数组，两个整型变量，获取一个字符串给到 s 后，循环判断 s 内每个字符如果是数字 0-9 之间的字符就把他送到 d 数组里面去，最后输出 d 中的字符串即可。

4、改错题请看代码下面的代码修改

①有 4 处修改

```
#include <stdio.h>
#include <string.h>
void main()
{
    char str1[] = "abcdefg";
```

```
    char str2[] = "abcdefg";
    // 前两行去掉了原来的花括号，可去可不去
    if (strcmp(str1, str2) == 0)
    // 条件使用 strcmp 函数，比较一致会返回 0
    // str1==str2 是地址的比较 当然不一样
        printf("yes");
    else
        printf("no");
}
```

因为改用了 strcmp 函数所以添加了 string.h 头文件，字符数组在初始化定义时直接等于字符串即可，不用加什么花括号。

strcmp 函数的具体用法可以看课本 151 页，了解不同情况的结果是什么。

②有 2 处修改

```
    int i;
    char c1[] = "How are you?";
    printf("%s", c1);
```

去掉了花括号，不去也是可以的，c1 的方括号必须去掉，要输出字符数组的所有字符，直接给首地址也就是数组名即可，加一个空下标是错误写法。

③有 2 处修改

```
#include <stdio.h>
void main()
{
    int m[][3] = {1, 4, 7, 2, 5, 8, 3, 6, 9};
    int i, j, k = 1;
    for (i = 0; i < 3; i++)
    {
        printf("%d ", m[i][k]);
    }
}
```

stdio 题目是 studio，很多同学也经常打错；

k 应该被赋值为 1 而不是 2，因为要输出的是 4 5 6，序号为 1 的列，也就是第二列；

m 下标应该先 i 后 k，也是因为要输出的是不同行的第二行也就是序号为 1 的列的内容。

当然你也可以依然让 k=2，改成 m[i][k-1]。

四、编程题

1、

```
#include <stdio.h>
void main()
{
    int s[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int max, min, i;
    max = s[0];
    min = s[0];
    // 先把最大最小值设定为第一个元素
    for (i = 0; i < 10; i++) // 循环 10 次
    {
        max = max > s[i] ? max : s[i];
        min = min < s[i] ? min : s[i];
        // 我使用了条件表达式，if 也可以的
        // 每次循环让数组元素和 max 与 min 进行对比
    }
    printf("max=%d , min=%d", max, min);
}
```

max 和 min 变量最初被设定成第几个元素都是可以的，毕竟我们待会会一个元素一个元素的去扫描，挨个对比，原理也是挨个对比。

要是对比发现比 max 大了那就把 max 等于它，对比发现比 min 小了，那就把 min 等于它。扫描完成后的 max 和 min 一定是这组数据中的最大值和最小值。

2、

```
#include <stdio.h>
int main()
{
    int a[15] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19,
 21, 23, 25, 27, 29};
    int min = 0, max = 14, mid, n; //max 为数列长度
    printf("请输入您要查找的数:\n");
    scanf("%d", &n);
    while (min + 1 != max)
    {
        mid = (min + max) / 2;
        if (n > a[mid])
            min = mid;
        else if (n < a[mid])
            max = mid;
        else
        {
            printf("输入的数在数列的第%d 位
\n", mid + 1);
```

```
            break;
        }
    }
    if (n == a[max])
    {
        max += 1;
        printf("\n输入的数在数列的第%d位\n", max);
    }
    else if (n == a[min])
    {
        min += 1;
        printf("\n输入的数在数列的第%d位\n", min);
    }
    else if (n != a[mid])
        printf("\n输入的数不在数列中");
    return 0;
}
```

首先我们需要清楚折半查找法的原理，例如在上述程序的 15 个数中，我想寻找 11 这个数。我先用这组数列里最中间的那个数也就是第 8 个数 15 与 11 做比较，发现 11 小于这个数，此时中间右侧的数就不用去比较了，我们只看左边剩下的 7 个数，还是取中间的数也就是第四个数 7 与 11 进行比较，发现 11 比这个数大，此时我们就只需要看 7 右侧和 15 左侧的数了，依然寻找中间的数正好是 11，完成比较。

在用 C 语言完成这个过程时，我们根据实际情况需要考虑 2 种情况，那就是要找的数存在、要找的数不存在。

整个程序贯穿的核心算法是 min+max 然后除 2 来不断获取新的中间值。所以在 while 中，你会发现 max 和 min 根据不同的情况被赋予了不同的值。以此来获取下一个中间值，原理也是很简单的，一组数列首尾位置相加除 2 就是中间数的位置。

当要找的数据存在时，我们循环查找中间数进行对比，总能循环到那个相等的情况，输出存在即可。

当要找的那个数不存在时，我们会发现程序会循环到只剩两个数的时候就不能再取中间值了，这个时候就要判断 min 或 max 是否和要对比的值相等。相等的话意味着存在，不相等意味着没有这个数存在。

整个程序逻辑清晰，简单易懂，可百度搜索折半搜索法获取更多信息。需要说的是，代码所使用的算法并不是唯一的，所谓条条大路通罗马，如果你有更好的算法也可以发出来和我们一起研究。计算机学科里还有很多有趣的算法，等你来探索~

3、

```
#include <stdio.h>
int main()
{
    int s[3][3] = {8, 4, 5, 19, 8, 9, 31, 11, 22};
    int i, j, k, min, min_h = 0, min_l = 0, staus;
    for (i = 0; i < 3; i++)
    {
        min = s[i][0];
        // 默认最小为该行第一个元素
        min_h = i;
        min_l = 0;
        for (j = 0; j < 3; j++)
        { // 寻找序号为 i 的行中最小值
            if (min > s[i][j])
            {
                min = s[i][j];
                min_h = i;
                min_l = j;
            } // 及时记录最小值的坐标
        }     // 执行到这里已确定行最小所在的列
        for (k = 0; k < 3; k++)
        { // 锁定列后逐行对比这个数是不是这列最大的
            if (min < s[k][min_l])
            {   // 如果小于别的数了 说明不是最大的
                staus = 1;
                // 让 temp 变量等于 1 说明本情况失败
                break;
                // 跳出，进行下种情况判断
            }
        }
        if (staus = 0)
        // 如果发现了成功情况则跳出
            break;
    }
    printf("Andian=%d Hang=%d Lie=%d", s[min_h][min
_l], min_h, min_l);
    return 0;
```

}

做编程题首先需要明确编程目标和思路。题目要求我们寻找一个整型二维数组的鞍点，并定义说鞍点是指某个元素是其所在行中最小、所在列中最大的元素。所以此题有两种思路，第一种是先找行中最小，再去看其是不是列中最大，第二种先找列中最小，再去看其是不是行中最小，本质上没有区别，只是查找顺序不一样。

在给的答案中采用的是先找行最小的算法，程序定义的数组 s 是任意给的数字，正确的鞍点应该是 11 这个数，我们首先需要让程序进行三次循环，因为我们的二维数组有三行三列，因为是先查找行最小，所以必须保证查找三次，循环次数也就是你的行数。(如果先查找列则循环次数与列数相同) 每次查找时，先假定最小值是该行第一个元素，然后使用 for 循环进行循环比对，发现即改变，同时必须记录最小值的行列坐标，以便后续的使用。

找到最小值后我们得到了它的行列坐标，接下来要做的事情就是看这个数在它所在的列是不是最大的，同样使用 for 循环进行循环比对，在使用数组下标时，因为列已经固定，所以行坐标可使用循环记录变量，列坐标可以直接使用刚才记录得到的最小值的列坐标。如果发现列中还有比他小的数，则令 staus 变量=1，并跳出循环，已经不成立了就不用再找了。行列都查找完后迎来一个 if 判断 stats，如果刚才的列查找满足条件则不会被改变，所以如果 staus 是 1 则说明不符合情况需要继续循环，如果是 0 则说明符合条件，找到了我们需要的鞍点，则可以停止循环进行最后的输出了。

记得按照题目要求输入鞍点及其所在的行和列信息。

第七章

邬山峰

答案速对 1-5 BCCAA 6-10 DDBBD 11-14 AABB

一、选择题

1、B　在 C 语言中实参可以为任何类型，但是形参类型需要与实参保持一致，不然编译器不给通过。

2、C　函数声明实际上还是一条语句（所以还是要分号结束滴）而且函数声明中要向编译器提供函数返回值类型、函数名、参数个数、函数体等的定义，所以参数变量的类型也是要说明的。

3、C　定义函数时，形参的类型说明放在形参表列内说明，或放在函数定义的第二行，函数体花括号“{”之前，所以选项 A 错；return 后面的值可以是一个表达式，选项 B 错；实参与形参的类型应相同或赋值兼容，如果实参为整型而形参为实型，或者相反，则按不同类型数值的赋值规则进行转换，以形参类型为准，选项 D 错。

4、A　函数中的局部变量，如果不专门声明为 static 存储类别，都是动态的分配存储空间的，数据存储在动态存储区中。这类变量叫做自动变量，自动变量可以用关键字 auto 作为存储类别的声明，实际上关键字 auto 是可以省略的。

5、A　在 C 语言中，实参和形参必须是地址常量或变量。而用数组名作为函数的参数则是一个典型的地址传递方式，在数组名作为函数参数时进行的传送只是地址的传送，也就是说把数组的首地址赋予形参数组，实际上是形参数组和实参数组为同一数组，共用同一段内存空间。

6、D　函数调用是作为另一个函数调用的实参，因为在使用被调函数时，是为了使用被调函数的结果，所以在引用被调函数时，实际上被调函数已经为一个定值，为实参。

7、D　规定就记就完事了。如果函数值的类型和 return 语句中表达式的值不一致，则以函数类型值为准。对于数值型数据，可以自动进行转换，即函数类型决定返回值类型。

8、B　简单变量作为实参时，它的对应形参也是简单变量，它们之间的数据传递是单向的值传递，即数据只能由实参传到形参形参值的改变不影响实参值。

9、B　在一个 C 程序中，若要定义一个只允许本源程序文件中所有函数使用的全局变量，则该变量需要定义的存储类别是静态变量 static。全局变量(外部变量)的说明之前再冠以 static 就构成了静态的全局变量。全局变量本身就是静态存储方式， 静态全局变量当然也是静态存储方式。这两者在存储方式上并无不同。这两者的区别虽在于非静态全局变量的作用域是整个源程序，当一个源程序由多个源文件组成时，非静态的全局变量在各个源文件中都是有效的。而静态全局变量则限制了其作用域， 即只在定义该变量的源文件内有效，在同一源程序的其它源文件中不能使用它。由于静态全局变量的作用域局限于一个源文件内，只能为该源文件内的函数公用，因此可以避免在其它源文件中引起错误。

10、D　C 语言在函数中说明的变量为局部变量，只在函数内起作用不会影响到其他函数。在不同函数中使用相同的变量名不代表是同一变量，选项 A 正确。在函数定义时声明的参数只在函数内部起作用，是函数的局部变量，选项 B 正确。在一个函数中定义的变量是这个函数的局部变量，所以只在这个函数内起作用，选项 C 正确。复合语句中定义的变量其作用域是这个复合语句，不会扩大到整个函数，所以选项 D 错误。

11、A　函数的数据类型是指函数的返回值类型，因为函数本身没有类型。函数相当于白纸，函数返回值即纸上的内容，是试卷还是答案全由内容决定。

12、A　A.静态变量会被分配在静态存储区内的存储单元，在系统运行期间都不释放，它的生存期为整个源程序，且它只在定义它的作用域内可见；B. 动态变量在函数结束或复合语句结束时，存储空间会自动释放，所以它不会一直存在，排除； C. 外部变量其实与静态变量很像，生存期也为整个源程序，但是外部变量是文件作用域，意思是在整个文件中它均是可见的；D. 内部变量的作用域是在定义它的函数中的，函数结束自动释放，所以它虽然是在定义它的函数内可见，但它无法在整个程序运行期间存在。

13、B　这个没什么说的，规定……，而且定义嵌套会使定义不清晰，最后鬼知道你定义了什么。

14、B　程序开始定义了一个整型函数，目的在于计算整数 n/10>0 的次数，然后 main 引用了定义的 digits 函数并对 n 赋 824，然后把数带进去算就行。

二、填空题

1、库函数 用户自定义函数

从函数定义角度看，函数分为库函数和用户自定义函数两种。C 语言的函数兼有其他语言中的函数和过程两种功能，从这个角度看，又可以分为有返回值函数和无返回值函数两种。从主调函数和被调函数之间数据传送的角度又可以分为有参函数和无参函数两种。

2、顺序 类型

首先个数肯定要相同，不然你定义出来干嘛，浪费时间浪费空间。然后顺序也要对应上，类型的话，因为实参要向形参传递数值的，类型不同很容易会造成数据丢失等一系列问题，而且编译器也不会给通过（到时候等待你的将是大大的报错）。

3、定义 调用

4、fmax(a,N)　s[k]=s[p]

写这种程序填空题我们一定要知道题目在写什么，这第一个空很明显是要输出一个东西，而这个程序目的是为了显示具有 n 个元素数组 s 中的最大元素，但是我们可以看到，实现显示元素的代码在子函数 fmax 中,且第一空的位置已到达 main 函数的结尾处，所以这第一个空显然是要引用 fmax 并且把实参向形参传递。

5、age(n-1)+2　age(n)

首先明确 age 函数是为了求第 n 个学生的年龄(这个没问题吧？都看得懂吧……)，然后它在 if 里面先定义了 n=1 时 age 为 10(c 是 age 函数返回值)，所以位于 else 后的①空应该是要求 n>=2 时的年龄，题目说了一个比一个大一岁，写表达式嘛(都说到这里应该会了吧)，这里给了两种答案，第一种是典型是数学思维答案，第二种是用了函数嵌套调用。然后②，题目要求我们求第五个孩子年龄，引用 age 函数并赋 n=5。最后 age 参数写 n 或写 5 都行。

6、prt(c,n-1)　prt(32,n-i)　prt(42,i)

首先在前面打过很多次三角形的我们知道打三角形肯定要循环(这里说的是比较方便，你要直接打当我没说)，但是在这里我们发现没有我们之前学的任意循环结构，所以在这里我们要用函数嵌套的方式来达到循环的功能。

首先我们知道子函数肯定是输出字符的，所以我们要在子函数内实现嵌套，然后①空是在 if(n>o)内的，所以填 prt(c,n-1)：因为后面有个 n-1，所以可以达到一种循环，n=1 循环一次，2 循环两次……。然后②空是输出空格，32 为空格的 ASCLL 码值。③是输出*号，因为等边三角形嘛，每行*号个数与行数相同，所以用 i，42 为*的 ASCLL 码值。（最后可能看上去不是等边三角形，但是那是题目和缩进问题……没办法了）。

三、分析题

1、hlo

首先我们先看 main 函数，是赋 i 为 0，并引用子函数 func1，那我们带着 i=0 进入 func1，要注意的是前面还有一个全局变量 st[ ]字符数组。进入 func1 首先打印 st[0](这就是 h)，向下，因为 0<3 所以进入 if 后面的表达式，i+=2，然后引用 func2，所以我们就带着 i=2 进入 func2，打印 st[2](这是 l)，然后因为 2<3 的原因，就进入 if 再循环一遍 func1，此时 i=4，打印 st[4](这是 o)，结束(因为此时 i=4>3 不

能再进入 if 后的表达式了)。

2、3

进入 main 函数(以后养成习惯先看 main 函数，因为程序是从 main 开始的)，我们注意到在 for 循环后面调用了子函数 f(i)，那我们就把 for 循环里面每个 i 的值一个个带进 f(i)函数内，首先 i=1 时，直接 return 1 结束，然后 i=2 时进入子函数的 else 内，注意这里是进行了函数调用的嵌套，在这里因为 n=2 的原因所以相当于返回 f(1)+1=2 的值，所以 f(2)=2;最后加起来就行，1+2=3 搞定!

3、26

在这个程序中，有定义的全局变量 int x=3，但是我们发现在子函数 incre 中也有一个 int x=1，在这里我们要注意同名全局变量和局部变量的定义——在定义局部变量的子函数中是以定义的局部变量为准的，但全局变量不改变，而且局部变量作用域只在定义它的函数内。然后我们来看 main 函数，由于全局变量 x 定义为 3，所以 for(i=1;i<x;i++)可以写为 for(i=1;i<3;i++)所以可以知道 incre()被调用了两次。

再看 incre 函数，这里要注意定义的局部变量 x 前面还有 static 静态变量前缀，所以第一次调用返回打印 2 应该没什么问题，但是因为有 static 前缀的原因，在第二次调用的时候 x 的值保留的是第一次调用的结果——x=2，所以第二次调用 incre 函数返回值是 6。static 静态变量会保留函数值不释放，你可以试试去掉 static，那么结果就会变为 22 。

4、21

进入 main 函数，调用 func 子函数，这题的难点在于在 func 函数的调用中出现了((x--,y++,x+y),z--),首先(x--,y++,x+y)是一个逗号表达式，取最后一个表达式的值作为逗号表达式的值，然后因为都是后缀的原因，先进行运算再进行增减，最后正常运算流程走下去就行了，得出 13，结束。

5、5, 25

首先进入 main 函数，一直到调用 num()之前都是常规操作没问题。然后我们再看 num()函数，首先注意它类型标识符是 void，本来它应该不返回任何值回主函数的(空类型函数嘛，小声逼逼，那结果不应该是 2，13 嘛)，但是(嗯，又来但是了（狗头）)这里的局部变量 x，y 前面有 extern ，就把前面定义的全局变量 int x,y;的作用域扩展到了 void num()中，在这函数里的 x,y 就变成了全局变量，这里 x，y 的值改变会影响 main 函数 x,y 的值(可以试试去掉 extern，那么结果就会变成 2，13)，又因为运算顺序的问题，最后的结果为 5，25 。嗯，就酱。

四、编程题

1、

```c
#include <stdio.h>
int n, i;
int a(int n)
{
    int c;
    if (n == 1)
        c = 0;
    else if (n == 2)
        c = 1;
    if (n >= 3)
        c = a(n - 1) + a(n - 2);
    return c;
}
int main()
{
    int n, i;
    n = 40;
    for (i = 1; i <= n; i++)
    {
        printf("%d\t", a(i));
    }

    return 0;
}
```

首先我们要知道斐波那契数列是第一项为 0，第二项为 1，然后从第三项开始每一项等于前两项的和。然后递归函数实际上还是函数嵌套，所以这里我们在子函数 a()中用函数嵌套的方式来实现从第三项开始每项等于前两项的和，最后在 main 主函数制表打印就 ok。

2、

```c
#include <stdio.h>
int s(int a, int b)
{
    int c;
    c = a * b;
    return c;
}
```

```
int main()
{
    int l, w, h, v;
    scanf("%d,%d,%d", &l, &w, &h);
    v = s(l, w) * h;
    printf("%d\n", v);
    printf("%d\n", s(l, w));
    printf("%d\n", s(l, h));
    printf("%d\n", s(w, h));
}
```

解析：这题蛮简单的，用了一个子函数计算面积，在 main 函数调用，然后底面积乘高计算体积。

3、

```
#include <stdio.h>
int exchange(int a[5][5])
{
    int i, j;

    for (i = 0; i < 5; i++)
    {
        for (j = 0; j < 5; j++)
            printf("%d\t", a[j][i]);
        printf("\n");
    }
}
int main()
{
    int a[5][5] = {{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10
}, {11, 12, 13, 14, 15}, {16, 17, 18, 19, 20}, {21,
 22, 23, 24, 25}};
    exchange(a);
}
```

二维数组行列互换把行列交换打印就 ok……，应一下本章主题加了一个 exchange 子函数。

4、

```
#include <math.h>
#include <stdio.h>
#include <string.h>
int main()
{
    char array[100];
    void fun(char array[100]);
    gets(array);
    fun(array);
}
void fun(char array[100])
{
    int sum[100], i, c = 0, l;
    l = strlen(array);
    for (i = 0; i < l; i++)
    {
        if ('0' < array[l - 1 - i] && array[l - 1 -
 i] <= '9')
            sum[i] = pow(16, i) * (array[l - 1 - i]
 - '0');
        if ('a' <= array[l - 1 - i] && array[l - 1
- i] <= 'f')
            sum[i] = pow(16, i) * ((array[l - 1 - i
] - 'a') + 10);
        if ('A' <= array[l - 1 - i] && array[l - 1
- i] <= 'F')
            sum[i] = pow(16, i) * ((array[l - 1 - i
] - 'A') + 10);
    }
    for (i = 0; i < strlen(array); i++)
    {
        c += sum[i];
    }
    printf("%d", c);
}
```

首先我们都应该知道 16 进制转 10 进制怎么转吧？就是从右往左每位上的数依次乘 16 的 i(第 n 位-1)次方。然后我们就通过 math 函数库里的 pow 函数来实现 16 的 i 次方的问题，然后由于我这里是从 i=0 开始的，所以我通过 array[l-i-1]（这里要注意因为 array 是字符数组，加上我们是从 i=0 开始的，所以最后一个字符要在实际长度 1 的基础上减一）的方式来输出最右项，然后判断是字母还是数字，输出相应的数字去与 16 的 i 次方相乘，每次运算我是放进了 sum 这个整形数组里面，最后通过把 l（输入字符实际长度）个 sum 中的元素全部相加就得出了转换的十进制结果。

5、

```
#include <stdio.h>
```

```c
#include <string.h>
void func(char a[100])
{
    int i, l, m;
    for (i = 0; a[i] != '\0'; i++)
        l++; //这里是求出输入字符串实际长度
    printf("%d\n", l);
    for (i = 0; i <= l / 2; i++) //循环
    {
        m = a[i];
        a[i] = a[l - 1 - i]; //注意由于字符数组最后一项是'\0',所以还要减一
        a[l - i - 1] = m;
    }
    puts(a);
}
int main()
{
    char a[100];
    gets(a);
    func(a);
}
```

首先构思怎么写一个交换顺序的函数——把第一项和最后一项交换然后第二项和倒数第二项……这样一直循环交换 L(L 为输入字符的实际长度)/2 次（这个应该看得懂吧？一次交换涉及两项，所以实际交换 L/2 次就能完成所有交换），最后打出相应函数程序实现就行。

第八章

史贝宁

答案速对 1-5 CBACA 6-10 CBDAB 11-15 CDCAD

一、选择题

1、C　题目里面的二维数组的九个数字，我建议大家先以矩阵的形式写在题目旁边，方便查找行和列。

然后题目定义了一个指针变量 p，并指向了二维数组 1 行 1 列的数字，也就是第 2 行第 2 列的数字 5。接下来开始了 for 循环，根据题目得知循环 2 次，每次输出一个整数，第一次输出 p[0]，第二次输出 p[2]，课本告诉我们指针变量也是可以加下标的，p[5]和*(p+5)是等价的。

所以本题第一次输出的是 p 指向的数字本身 5，第二次输出的是 p 指向地址加 2 后为第三行第一列的数字，地址加几就往后数几个元素，加 0 就是它本身嘛，很好理解，然后取内容便是 3，最终得到答案 53。

2、B　题目定义了一个有 10 个元素的字符数组，并初始化了值，其中第十个元素是 0 需要大家注意，因为这个 0 是没加单引号的。还定义了一个指针变量 p。使得 p 指向 a+8，也就是数组 a 的第八个元素地址，最后输出的时候输出的是 p-3，最后也就是 a+5 啦，从 a 这个字符数组的第六个元素开始输出，因为 printf 的格式控制符是%s，所以从这个元素之后的所有元素都需要进行输出，答案就是 6789，为什么没 0 一开始就已经解释过了。

3、A　首先定义了有 12 个元素的字符数组 a，并把指针变量 p 指向了 a+5，也就是第六个元素的地址。指针变量 q 定义时指向 NULL。接下来 q 地址的值被赋为 p+5 的值，但是我们发现 q 本身的地址是 NULL，哪来的地址给它存数据呢？程序便会出现错误，于是选择运行后报错。

4、C　我们需要表示 a[1]的地址，A 选项为指针变量 p+1，指针变量是可以加 1 的没问题，加 1 表示的是当前地址向后移一个存储单位（比如整型数组每个元素占据 4 个字节，就会往后移动 4 个字节），也正是数组序号为 1 的元素地址；

a 为数组名，同时也是数组的首地址，对地址加 1 则会变成下一个元素的地址，成立；

a 刚才说了是数组名和数组的首地址，但是它是一个地址常量，常量是不能改变的；

p 是指针变量可以使用自增运算，自增后跳转到下一个元素的地址。

5、A　题目声明了 long 类型的变量 a，和指向 long 类型的指针变量 p。

我们看到 A 选项直接给*p 赋值，请注意，此时此刻*p 代表的是 p 所指地址的内容，定义指针变量 p 之后并没有让他指向任何地址，所以这个用法是完全错误的。后面的 scanf 没什么问题。

B 选项中使用了 malloc 函数，在第九章我们会学到它的使用，本选项中它的作用是向系统申请 8 个字节的内存，然后(long *)是把 malloc 函数的返回值强制转换为长整数指针型，结果就是 p 指向了一个长整型数据的地址；

C 选项中 p 被赋值为了 a 变量取地址，p 本身就是指针变量当然可以被赋值为地址，然后被 scanf 函

数使用此地址去获取内容；

D 选项是常规的取变量地址用法，大家见的很多了。

6、C　先看看这串代码做了什么，首先，定义了三行三列的二维数组 t，定义了有三个元素的整型指针数组 pt，还有一个整型变量 k。然后开始 for 循环，很明显会循环三次，这三次会分别执行以下语句：

```
pt[0] = &t[0][0];
pt[1] = &t[1][0];
pt[2] = &t[2][0];
```

分别让 pt 指针数组里的三个元素指向了 t 的不同元素。题目问的是*(*(pt+1)+2)表示的数组元素是哪一个，我们一层一层看：

pt：代表 pt 这个指针数组的首地址，也可以理解为这个一维数组中序号为 0 的元素的地址。

pt+1：代表 pt 这个指针数组中序号为 1 的元素的地址。

*(pt+1)：括住前面加*就是取这个地址的内容，指针数组元素的内容依然是地址，此时代表的应该是 t[1][0]这个元素的地址。

*(pt+1)+2：让 t[1][0]这个元素的地址再加 2，因为是二维数组，我们刚才已经确定了行数为 1，现在让列加 2 就行了，往后挨个加嘛，得到最后表示的地址就是 t[1][2]。

((pt+1)+2)：最后括住又取内容，所以结果就是 t[1][2]所代表的元素内容了。

这里二维数组的指针操作需要你对课本 213 页的内容有足够的理解，多看多思考，加油！

7、B　这道题目就很有趣了，题目忘记给你把 if 括号里面的表达式去掉，答案直接告诉你了，哈哈哈。别开心了，你会吗？接下来来看看这道题如果不给答案怎么做哈。

程序的目的是把数组元素之中的最大值放在数组 a 的第一个元素之中，具体的流程我们看了代码以后就可以发现是循环十次，每次判断数组里面的元素是否比第一个元素大，如果打的话就让第一个元素等于它，最后输出这个最大的元素。整个流程很好理解，但是题目使用了指针变量来干扰你的理解。没关系，也很简单。

题目定义了一个整型数组 a，里面有十个元素，放了十个数字，还定义了一个指针变量 p，初始化指向了数组 a 的首地址，还有一个用来记录循环变量的 i。

接下来明白 a、p、*a、*p 是什么就可以了。a 是数组 a 的首地址，p 是指向 a 数组的首地址，现在你可以把 p 和 a 理解成等价的，*a 是取 a 这个地址的内容也就是第一个元素，你可以理解成 a[0]，*p 同样。随着循环的进行，i++，p++，所以*p 会随着循环表示数组 a 中的不同元素。

在 if 的条件里应该写如果数组 a 中的不同元素大于第一个元素，那么正好就是*p 来表示不同的元素，大于，第一个元素用 a[0]来表示。

如果 if 判断成功，则让该元素的内容赋值给第一个元素，*a=*p 刚好实现我们的意图，本题结束，不知道我讲的清楚不？哈，如果不清楚欢迎来问。指针本来就比较难理解，但是不能直接放弃，理解之后还是很简单的哦。

8、D　解释一下代码：a 为定义为全局变量的二维数组，p 也是全局指针变量，指向整型地址。f 是一个函数，它有两个变量，第一个是指针型，第二个是整数二维数组型。

接下来我们跟着 main 函数开始执行喽，malloc 函数会在第九章学到，它的作用刚才已经讲了，就是让系统分配一块内存出来，分配多大看函数的实参，sizeof 就是取内存大小，即取 int 的内存大小，也就是说 malloc 要分配一块内存用来存整数型数据，函数之前的(int *)是要强制把 malloc 函数的返回值转换为指向整数型变量的指针地址，最后赋值给指针变量 p。

接着调用函数 f，实参为变量 p 和数组 a。进入 f 函数后，把数组行列分别为 1、1 的元素赋值给了指针变量 p，也就是把 5 这个数据存到了指针变量 p 所代表的地址中。回到 main 函数继续执行，输出指针变量 p 的内容，那就是 5 喽。

9、A　题目定义了一个指针变量 st 指向字符串，然后初始化指针地址为一个字符串的首地址。

A 选项来个字符型数组 a，指向字符串的指针变量 p，然后用 strcpy 函数把第二个参数：st 本身就是个地址，加下标没问题，就相当于是*(st+4)，最后又来了个&取地址。也就把 st+4 之后的字符串赋值给了 a+1 去。验证一下数组 a 有足够的空间存放 are you 内容。

B 选项出现了对地址常量 a 进行自增操作，这当然是不允许的，常量不允许被改变。

C 选项通过 strcpy 函数把 st 里面的字符串给到了字符数组 a 之中，该函数的第一个参数是数组 a 一共有 11 个元素，第二个参数可以是字符串也可以是地址，这里字符串一共有 12 个字符（包括\0）。

我们发现数组 a 没有足够的元素个数来接受第二个参数送来的字符，会占据数组 a 第 11 个元素之后的内存空间，这将导致程序出现数组越界的未知错误，而程序看起来是一切正常的。

D 选项定义数组时没有给定元素个数，系统将不知道为其分配多少内存，编译无法通过。记住 C 语言本身时无法定义动态数组的。

10、B　p 指向变量 a 的地址，a 被赋值为 p 取内容也就是 a 的值再加 b，也就是 10+1 为 11。

11、C　不能进行运算“+”运算。类型相同的两个指针变量之间可以进行<（小于）、=（等于）、-（减法）运算。

<（小于）运算在两个同类型的指针间可以比较大小，比较原则应该是按照实际内存的高低位比较的

=（等于）是对于类型相同的两个指针变量之间常规运算。

-（减法）运算两个相同指针变量相减可以获得在之间相隔的同类型元素个数（在某个类型的数组中的应用）。

+（加法）运算是不可以的，因为两个指针相加什么都得不到，所以规定不允许相加。

12、D　首先我们有了一个字符数组 s，和指针变量 p，接下来开始 for 循环了，第一次循环先让 p 指向 s+1，也就是元素 B 的地址，判断 p 小于 s+4 也就是在数组范围内成立的话，就以%s 格式输出 p，也就是说会输出 p 所指向指针所代表内容和其之后的内容直到遇到\0，那么就是 BCD，然后执行 p++，自然 p 的地址后移一个，那么输出也会变成 CD，最后是输出一个 D。

13、C　程序定义了一个字符数组 a，里面有十个元素，然后定义了一个字符指针变量 p，指向 a[5]的地址。输出*--p，自减的优先级比*高，所以先自减让 p 指向 a[4]然后再取其内容输出结果就是 5 了。

14、A　这个题呀，他就是想用不同的参数名来干扰大家。不怕，很简单，搞清楚关系别脸盲就可以作对。

首先程序定义了一个函数 fun，它有两个参数，两个整型指针。

进入 main 函数开始执行了，先在 main 函数内定义了两个变量一个 x 一个 y 分别赋值为 1 和 2。

调用 fun 函数，实参给的是 main 函数中 y 的地址和 x 的地址，接下来我们进入 fun 函数。

参数传进来以后，fun 函数的 x 参数指向 main 中的 y 地址，y 参数指向 main 中的 x 值。故 x 参数内容为 2，y 参数内容为 1。

一进来就是 printf 输出 x 和 y 的内容，那就输出呗，2 和 1，然后让 x 的地址内容等于 3，让 y 的地址内容等于 4。好了，还记得参数 x 的地址是 main 里谁的地址吗？是 y！参数 y 呢？是 x!

所以回到 main 函数中，main 函数的变量 x 变成了 4，变量 y 变成了 3，最后输出 x 和 y，就是 4 和 3。

哈哈哈，有点绕，看不懂就多看几遍。

15、D　A 选项正确对应的应该是 int *f；B 选项正确对应的应该是 int (*f)[]；C 选项正确对应的应该是 int (*f)()。

二、填空题

1、定义了一个返回值为整型指针的函数

定义了一个指向返回值为整型的函数的指针变量

第一空中，因为括号的优先级比较高，所以优先匹配（）说明 f 是个函数，然后是前面的 int *，这是这个函数的返回值类型；第二空中，两个括号分别匹配那么运算是从左到右的，int (*f)()说明这是一个指针变量 f，后面还有个括号，继续说明这是一个指向返回值类型为整型的函数指针。

2、地址 NULL

指针变量存储的是内存地址，除了放地址，还可以放 NULL 值，也就是空地址。

3、0

字符串指针的定义其实就是让系统定义了一个字符数组，然后把字符数组的首地址给到指针变量。既然字符串指针和字符数组是一个东西，那么字符串结束标志\0 也是存在的，题目最后输出了 p[4]，那这个字符串里序号为 4 的元素正好是结束符\0，这个字符在 ASCII 码表中是 0。

4、int *z

想要用形参把数据传回，那么参与计算的必须是指针变量，通过指针变量才能够让数据在不同的函数之间传递。故我们根据函数体内的*z=x+y 语句判断这里的形参应该定义为 int *z。

如果你无法理解我所说的内容，那可能是因为你看书不太仔细，这些内容在书上都是找的到出处的，本题内容涉及课本 8.2.5 节内容。

5、*(p+5)或 p[5]

在数组 w 中，元素 98 的序号是 5，那么通过指

针变量 p 来调用它，则从 w 的首地址向后加 5 个单位即可。代码中已经让 p 指向了 w 的首地址，那么我们直接使用即可。注意指针变量 p 加下标等价于加下标后取其内容。

6、a+i　&a[i]　*(a+i)

本题只是单纯的使用字符数组的名字 a 来表示元素地址，首先你必须知道字符数组的名字 a 就代表了这个数组的首地址，也就是第一个元素的地址。然后你就可以用它加整数来表示别的元素地址或内容了。一定要区分题目问的是地址还是内容，如果是内容记得要括住加*来取这个地址的内容。

7、*(a+i)+j　&a[i][j]　（a[i]+j 也行）

i 行 0 列的元素地址　地址常量

大家把 213 页的表格掌握了，记得不要去背而是要理解，这道题就易如反掌了。针对二维数组指针的理解，首先我觉得大家应该先把二维数组理解成一维数组的每个元素还是一个一维数组，这样理解的话，二维数组的指针里 a+i 为什么要先取内容以后才能继续进行列操作的问题就迎刃而解了。因为你不取内容的话，始终进不去内层的一维数组呀~

8、4

先取序号为 1 的元素地址，然后再加 2，相当这个地址就是 a+3，那么最后指向数组的第，注意是第几个元素。

9、指针数组

好简单哦~

10、含有 4 个整型指针变量的指针数组

一个指向有 4 个元素的一维数组的行指针变量很简单了，前者就是指向一个指针数组的地址，后者呢就是指向一个一维数组的地址。叫它行指针变量，就是因为二维数组可以看作是一维数组的嵌套。氦，听不懂了？第七题讲过了什么叫一维数组的元素依然是一维数组哦。

三、编程题

1、

```c
#include <stdio.h>
int main()
{
    char *str = "YiQiLaiXue";
    int i = 0;
    while (*(str + i) != '\0')
    {
        i++;
    }
    printf("字符串长度为%d", i);
}
```

我先自己定义了一个字符串指针 str，在 C 语言中定义一个字符串指针相当于定义了一个字符数组，然后，系统把字符数组的首地址给到指针变量，所以我们可以通过'\0'结束符来判断一个字符串的结束。由此，我们可以用 while 来循环判断是否检测到了结束符，如果没有，则计数+1，如果有，则跳出循环输出我们的字符串长度。

本程序中的 i 既是字符串长度的记录变量，同时又是 str 字符串指针的偏移量，用它来实现取不同字符的地址。

2、

```c
#include <stdio.h>
int main()
{
    char *monthname[12] = {"January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"};
    int i;
    scanf("%d", &i);
    if (i >= 1 && i <= 12)
        printf("%s", monthname[i - 1]);
    else
        printf("error");
}
```

在本程序中我创建了一个有 12 个元素的指针数组，每一个元素存储了每一个月份英文名的字符串地址，这么做，能够让我们很方便的通过不同地址来使用不同的字符串内容。

按照题目要求输入一个整数，接下来继续判断输入的整数是否在我们 12 个月份的范围之内，如果在就通过地址输出，代码中我使用了指针变量加下标的形式，当然你也可以使用*(monthname+i-1)来达到同样的效果，不在范围内则输出错误信息。

3、

```c
#include <stdio.h>
int main()
{
    char str[] = "software", *p;
    int i;
```

```
    for (p = str; p < str + 8; p += 2)
    {
        printf("%c", *p);
    }
}
```

题目说建立一个字符数组那我啪的一下很快阿就建立了一个字符数组 char，然后把 software 这个字符串给存了进去，还顺便定义了一个字符型指针变量 p，因为题目说要求你讲武德，要用指针操作。

这好吗？这当然很好，要间隔的从第一个字符开始输出，那么我们使用指针的话，每次输出地址之后让地址往后移 2 个不就行了，很简单。所以我们采用了 for 循环，最关键的就是 for 循环里面的三个语句了，先让 str 也就是字符数组的首地址给到指针变量 p，现在指针变量 p 就指向了字符 s，然后每次循环的判断条件是 p 小于 str+8，在字符串范围内咱就继续循环，然后 p+=2，你懂的，往后移动 2 位，因为我们要间隔输出嘛。每次循环输出就行了，不过你要记得输出的时候可不是输出 p 而是要输出 p 的内容，使用*取内容运算符。

4、

```
#include <stdio.h>
#include <string.h>

char *copystr(char str[], int m); // 被调函数声明
char str2[100]; // 定义一个字符数组的全局变量

int main() // 主函数
{
    char str[100]; // 用于接受字符串
    int m; // 用于接收 m
    gets(str); // 接受字符串给到 str
    scanf("%d", &m);
    printf("%s", copystr(str, m));
    // 函数返回值是字符指针型 可直接%s 进行输出
}
char *copystr(char str[], int m) // 该函数有 2 参数 一个字符串 一个 m
{
    char *p = str, *p2 = str2; // 使用指针变量来方便访问元素
    while (*(p + m - 1) != '0') // 以结束符\0 作判断标准
    {
        *p2 = *(p + m - 1); // 开始复制字符串到全局变量的 str2 数组中
        p2++;
        m++;
    }
    return str2; // 函数返回复制好的数组首地址
}
```

因为本题使用了两个函数并且涉及函数之间的数据传递，所以比前几道题要复杂一些。只要你熟练掌握了第七章和第八章的知识，完全是 soeasy。

在主函数中输入字符串和输出 m 的数值，那么我们就需要一个字符数组来存放输入的字符串和一个整型变量来存输入的 m，因为 C 语言中是不支持动态数组定义的，所以我们定义数组的时候就稍微定义的大一点，这是在做题，所以输入的时候可以不越界，如果是真正的写程序，就必须使用 malloc 来动态定义数组了。

在被调函数中完成复制操作，那么复制操作就是从一个字符数组复制到另一个字符数组中，因为这是第八章，所以只能要用指针来做的步骤就一定要用指针来完成。那么很显然，能使用指针的步骤就是复制的步骤，他涉及了使用指针来访问字符数组的不同元素。

如上面的代码所示，在被调函数 copystr 中，指针变量 p 和 p2 分别指向了待复制数组 str 和要复制的数组 str2，通过对指针变量的运算来让不同的元素赋值到 str2 之中去。我个人把这个函数的返回值定义为了字符指针型，返回一个指针来方便 printf 直接输出地址的结果。但其实因为 str2 是我们定义的全局变量，则 copystr 函数可以没有返回值，因为该函数执行完毕后全局变量 str2 的内容已经赋值好了，直接在 main 中的 printf 中输出 str2 就可以，这完全是个人习惯，大家怎么喜欢怎么来，能够实现我们的题目目标就可以。

5、

```
#include <stdio.h>
int main() // 主函数
{
    int num[10] = {5, 7, 8, 3, 2, 1, 6, 4, 9, 10};
    int i, j, temp, m, n, *p;

    for (i = 0; i < 10; i++) // 升序重新排列
```

```
        for (j = i + 1; j < 10; j++)
        {
            if (num[i] > num[j])
            {
                temp = num[i]; // 符合条件数据交换
                num[i] = num[j];
                num[j] = temp;
            }
        }

    scanf("%d %d", &m, &n); // 输入位置和 n 个数
    p = num; // 使用指针变量
    for (i = m - 1; i < m + n - 2; i++) // 倒序排序
        for (j = i + 1; j < m + n - 1; j++)
        {
            if (*(p + i) < *(p + j))
            {
                temp = *(p + i); // 符合条件数据交
换
                *(p + i) = *(p + j);
                *(p + j) = temp;
            }
        }

    for (i = 0; i < 10; i++) // 输出数列
        printf("%d ", num[i]);
}
```

别看代码挺长的，但其实其中的思想并不复杂。我们把题目拆开，就能得到下面三个小问题：

你需要设一个数列有 10 个数，乱序的，很简单，那么我就定义了一个整型数组，里面的元素我都是乱着顺序写的；

你需要先把他们按照升序排列，两个 for 循环解决排序问题我们在之前的章节已经使用的很习惯了，应该理解上是没有问题的。实际上就是先拿出一个元素来，然后用后面的所有元素进行对比，如果后面的比这个数大了，进行交换最后就是降序排列，如果后面的数比这个数小了，进行交换最后就是升序排列。

接着你需要两个数据，从指定的位置开始的 n 个数，这个指定的位置和 n 个数都需要用变量存储，你可以直接写在代码里，也可以像我一样 scanf 让用户输入。

然后你需要把指定位置开始的 n 个数进行逆序排列，就是降序呗。这里题目要求我们使用指针变量来操作，那么我们就勉为其难的定义了一个整型指针变量 p，把它指向数组的地址 num。然后就可以通过 p 来操作数组了。降序操作就和上面升序的代码差不多，只不过是 if 里面的条件不一样，一个是数组操作一个是指针操作。

最后再来一个 for 循环输出数列结果就可以了。一气呵成，写代码最重要的是前期的规划，而不是一边写一边东拼西凑。

6、

```
#include <stdio.h>
int main() // 主函数
{
    int a[6] = {2, 4, 6, 8, 10, 12}, *p[3], i, j;
    p[0] = a;
    p[1] = a + 2;
    p[2] = a + 4;
    // 行顺序输出
    for (i = 0; i < 3; i++)
        for (j = 0; j < 2; j++)
            printf("%d ", *(p[i] + j));
    // 列顺序输出
    printf("\n");
    for (i = 0; i < 2; i++)
        for (j = 0; j < 3; j++)
            printf("%d ", *(p[j] + i));
}
```

本题的难点在于理解如何用指针数组加一个一维数组来组成一个二维数组，众所周知二维数组之中的元素可以理解成行和列。而指针数组里面的元素都是一个个的地址，那么我们可以让指针数组里面的元素，指向一维数组中不同的元素，这样不久人为的让一维数组有了行有了列嘛！

于是我们定义了一个元素个数为 6 的整型数组 a，按照题目要求赋值，同时还需要定义一个元素个数为 3 的指针数组，此时，我们将指针数组内的各个元素分别指向一维数组 a 中的第 1、第 3、第 5 个元素，那么我们就可以得到一个三行二列的二维数组。

接下来按照行顺序输出，一行一行的输出，表现在一维数组就是按照正常顺序输出。刚才我们已经通过指针数组内的元素获得了行指针变量，即 p

加下标就是每一行的指针，在此基础上加 1 或 2 就能访问到列的内容，使用 for 循环进行输出即可。

按照列输出同理，还是记得找到正确元素的地址最后输出的时候要取地址的内容，不然你输出的依然是地址而不是元素的内容。

7、

```c
#include <stdio.h>
int main() // 主函数
{
    int data[10], i, j, *p, temp;
    for (i = 0; i < 10; i++)
        scanf("%d", &data[i]);

    p = data;
    for (i = 0; i < 10; i++)
        for (j = i + 1; j < 10; j++)
            if (*(p + i) > *(p + j))
            {
                temp = *(p + i);
                *(p + i) = *(p + j);
                *(p + j) = temp;
            }

    for (i = 0; i < 10; i++)
        printf("%d ", data[i]);
}
```

本题毫无新意，常规的循环输入数据到数组，循环输出数据的数据，甚至用指针排序在前面的题都已经做过了，大家可以多写写熟悉一下。

8、

```c
#include <stdio.h>
void findnice(int (*p)[5]); // 函数声明
int main() // 主函数
{
    int data[10][5] = {{13, 31, 45, 67, 88},
                       {91, 34, 54, 78, 11},
                       {99, 14, 28, 95, 23},
                       {54, 85, 13, 43, 95},
                       {12, 94, 15, 81, 74},
                       {11, 84, 29, 37, 18},
                       {18, 48, 23, 95, 23},
                       {13, 29, 98, 24, 28},
                       {23, 49, 59, 48, 84},
                       {28, 84, 27, 10, 18}};
    findnice(data); // 调用函数查找和输出
}
void findnice(int (*p)[5]) // 函数定义
{
    int max, i, j, h, l, (*ptr)[5];
    // max h l 记录要输出的内容 i j 为循环记录
    for (i = 0; i < 5; i++) // 5 列 寻找每列最大值
    {
        ptr = p; // 每次循环先让 ptr 指向 p 首地址
        max = *(*(p + 0) + i);
        h = 0;
        l = i; // 给 max 默认赋值为第一个数
        for (j = 0; j < 10; j++) // 循环比较
        {
            if (*(*(ptr + j) + i) > max)
            {
                max = *(*(ptr + j) + i);
                h = j;
                l = i; // 符合条件则记录
            }
        }
        printf("max=%d h=%d l=%d\n", max, h, l);
        // 输出每列找到的数据
    }
}
```

阿哈，这道题还是让人收获颇丰的，自己写完之后能够更好的掌握二维数组的传参和使用。真是妙蛙种子吃了妙脆角走进了米奇妙妙屋妙到家了！

话不多说，我们先来把这 10 个学生的 5 门成绩写到数据里，如代码所示，定义了一个 10 行 5 列的二维数组来存储这个数据，我们写入数据的时候是可以换行操作的，最后到分号结束，这便是 C 语言使用分号结束一条语句的好处。

题目要求之后的查找每门成绩的最大值、输出最大值和它数据的行列值都在被调函数中进行，那我们就声明并定义一个函数，程序中将函数定义为了 void 无返回值、名称为 findnice，参数为 int (*p)[5] 因为题目要求要使用指针传递参数。

书上 216 页第二点明确告诉我们如果要将一个二维数组的地址传递给函数，那么这个函数的形参必须是一个指向一维数组的行指针变量。所以我的

参数会这么写，后面下标为 5 是因为我们的二维数组是 10 行 5 列，本应把其看作是有十个元素的一维数组，然后每个元素仍是一个一维数组，每个元素里还有五个元素。最关键的是这样操作才能实现指针对二维数组的正确操作。

来到函数的定义部分，首先我们需要找到每一列也就是每一门成绩的最大值，使用循环查找即可，先定义一个 max，h，l 来记录，初始化 max 为这一列的第一个数据，接着使用 for 循环进行对比即可。比对过程需要使用指针进行操作。

我为什么又定义了一个 ptr 指针变量呢，是因为我们有 5 门课需要找最大值，所以每次循环的时候我都需要让指针指向这个数组的第一行去，如果你不再定义一个 ptr 出来，只用 p，那你就很难回到首地址去进行下一次循环了。

在循环对比过程中最容易犯难的就是如何使用指针进行二维数组的取内容操作了，现在我们拥有的是一个指向一维数组的行指针变量，假设 ptr+1 指向二维数组的第二行，然后我们想取这个二维数组第二行里面的第三列，则应该先进入列操作，即*(ptr+1)这样就进来列操作了，再+3 就得到了第二行第三列的地址，最后再括住取内容就可以得到内容了，最终为*(*(ptr+1)+2)。

你可能会疑惑为什么要从行操作转换到列操作，其实就是把他理解成一维数组的元素还是一维数组。因为二维数组可以这么表示：{{1，2}，{3，4}，{5，6}}，这就是一个 3 行 2 列的二维数组对吧，但是你可以把它看成是有三个元素的一维数组，没毛病把，ptr 就是这个一维数组的地址，ptr+1 就是{3,4}对吧，那你想取 4 的话不得先让人家进去才能取？所以你得*(ptr+1)哎就进来了，进来以后再加 1 不就取到元素 4 的地址了么，最后取个内容，是不是就直接明白了？我觉得我讲的挺清楚喽，如果还不明白的话直接来问我吧~

第九章

王得懿

答案速对 1-5 CBBCC 6-10 DA？DB

一、选择题

1、C 通过代码可知，有一个结构体 data，有一个结构体 person。在结构体 person 中有一个 data 型的结构体成员。也就是 person 里面有 data 这个结构体变量。题目定义了一个 person 型结构体变量 a，我们可以通过 a.name、a.sex 访问其中的 name 和 sex 成员，还可以通过 a.birthday 进一步进入 data 结构体中，然后 a.birthday.year 即可访问到 data 结构体中的 year 变量。

2、B p 中存有 data 的地址，由定义*p=data 得 B 等价于 data.a 还可以写为 p->a，为正确引用。

3、B 枚举 enum season {spring, summer, autumn, winter}；没有等号，没有引号，参考书上对枚举的定义。

4、C sizeof 用于计算所占字节数,其中 int 和 float 字节长度都为 4，共用体 union 中取字节长度最大的 double，长度为 8 个字节，2 x 8=16。

5、C 可以使用 typedef 来为类型取一个新的名字。此处 STU 为新的类型名，可以用这个新的数据类型来直接定义结构变量。

6、D p 指向结构体数组 x，结构体数组中 x[0]的 a 为 99，b 为指向 m1 数组的指针变量。同理得 x[1]中的元素 ++p 为先加再使用，p++为先使用后加。对 A：p 先加 1 即变成指向 x[1]，引用其中的 b 即 m2 然后指向 m2 的首元素，进行*运算后取出 m2 的首元素得 100；对 B：p 先加 1 即变成指向 x[1]，引用其中的 a 即 100；对 C：先取到 99 后自增 1 得 100；对 D：p 自增后指向 x[1]，取 b 地址得到 m2 的首地址，依然是个地址，而不是数据 100，所以错误。

7、A 声明结构体需要在花括号后加分号。

8、？ 结构体变量所占内存长度是其中最大字段大小的整数倍，共用体变量所占的内存长度等于最长的成员变量的长度。例如结构体中含有 char a;int b;此时结构体变量所占内存为 8，本题无答案。

9、D 对 A：将字符串赋入结构体数组 pup[1]中的 name 数组中，此处调用即为 name 数组的首地址，不需要取地址符；对 BC：数值赋入结构体数组 pup[0]中的整形变量中，都是正常使用；而 D 缺少取地址符。充分理解指针指向数组，即指针存有数组的首元素地址。

10、B 本题结构体里面套了一个共用体，结构体名为 st，结构体变量为 s，共用体名为 un，共用体变量为 h，我们只能通过变量名来引用成员，通过名来定义变量。所以含 un 的、含有 st 的为错误选项，我们引用共用体中的 h 成员正确应该是 s.h.h，而 p 指针恰好指向 s 的地址，则*p 可以等价于 s。

二、填空题

1、max=i;　min=i;　stu[max].score　stu[min].score

以第二个元素为起点，对之后的元素分别分析，即与第一个元素比较，如果值更大或更小，则将对应的数组名赋给 max 或 min，再用更大或者更小的作为基准继续与之后的值进行比较，从而将最大值或最小值找出，妙啊！

2、head　p　p->next

题中第一行定义应为 struct node *head,*p;

注意题目要求输入链表内容时与输入的顺序相反，本题的确创建了链表，我们为了实现相反的顺序，肯定就是要输出的时候，从第一个数据到最后一个数据输出后正好和输入的顺序时相反，那我们这条链条里面的内容，第一个数据一定必须是我们输入的最后一个数据，所以，创建链表时，我们应该从尾部开始创建，最后输入的数据成为我们的链表 head 所指向的数据。你可以模拟一个 head 节点不断向后退的过程，然后最后从 head 节点开始输出链表就可以实现我们想要的效果。题目中使用 head 和 p 来创建这个倒序链表一开始很可能会干扰你，习惯就好。

三、分析题

1、41, 50, 21

首先需要明确++为先加再使用，p++为先使用后加。此处为++p->x，->运算符比++运算符的优先级要更高，所以先完成 p->x 的运算，p 本来表示结构体数组 a 的首地址，取到成员 x 的内容为 40，后加 1 则变成 41，这是第一个输出语句；第二个输出语句为先让地址自增，则 p 指向了结构体数组 a 的第二个元素，再取成员 x 则取到了 50；第三个输出语句先算括号里面的，同样的->的优先级更高，取成员后我们得到了 dt[1]的地址，再取内容为 20，前面还有个自增别忘记，所以最后是 21。

2、5

此处为枚举的应用，枚举中的值所对应的整数值一般从 0 开始，当其中某个变量被赋了某个值时，则从修改的值往后，后面的变量递增，例如本题各个变量分别对应 0、3、4、5、6。

3、6

共用体，任何时候只能有一个成员带有值。共用体提供了一种使用相同的内存位置的有效方式。此处有两个共用体，x 和 y，先让 x.a=3 后 x.b 也会等于 3，所以 y.b 变成 5，于是 y.a 值继续变成 6 后，y.b 相同为 6。

四、编程题

1、

```c
#include <stdio.h>
struct baobiao
{
    char name[20];
    double j;
    double f;
    double z;
};
int main()
{
    int i;
    struct baobiao a[3] = {{"Zhao", 240.00, 400.00,
 75.00}, //定义的同时声明
                    {"Qian", 360.00, 120.00, 50.00},
                      {"sum", 560.00, 0.00, 80.00}};
    for (i = 0; i < 3; i++)
    {
        printf("%s\n", a[i].name);
//输出字符串
        printf("%lf\n", a[i].f + a[i].j - a[i].z);
//进行实发数的计算
    }
    return 0;
}
```

2、

```c
#include <stdio.h>
struct in
{
    int x;          //存学生学号
    char name[20]; //存学生名字
    int score[5];  //存学生成绩
};
void shuru(struct in *p)
{
    int i, j;
    for (i = 0; i < 20; i++)
    {
        printf("请输入第%d 同学的学号:\n", i + 1);
        scanf("%d", &p[i].x);
        printf("请输入第%d 同学的名字:\n", i + 1);
```

```
        scanf("%s", p[i].name);
        for (j = 0; j < 5; ++j)
        {
            printf("请输入第%d 同学的第%d 门成
绩:\n", i + 1, j + 1);
            scanf("%d", &p[i].score[j]);
        }
    }
}
void zongfen(struct in *p)
{
    int sum = 0, j;
    for (j = 0; j < 5; j++)
    {
        sum += p->score[j];
    }
    printf("总分:%d\n", sum);
}
void shuchu(struct in *p)
{
    int i, j;
    for (i = 0; i < 20; i++)
    {
        printf("输出第%d 同学的学号:", i + 1);
        printf("%d\n", p[i].x);
        printf("输出第%d 同学的名字:", i + 1);
        printf("%s\n", p[i].name);
        printf("输出第%d 同学的成绩:\n", i + 1);
        for (j = 0; j < 5; ++j)
        {
            printf("%d ", p[i].score[j]);
        }
        zongfen(p + i); //引用总分函数计算总分并输
出
    }
}
int main()
{
    struct in s[20];
    shuru(s);
    shuchu(s);
    return 0;
}
```

3、

```
#include <malloc.h>
//这里需要 malloc 用来创建动态内存
#include <stdio.h>
struct ll //声明结构体
{
    int data;
    struct ll *next;
};
int main()
{
    int a, i = 0;
    struct ll *head = (struct ll *)malloc(sizeof(st
ruct ll));
    //头结点需要动态的创建
    struct ll *Tail = head;

    //创建增加结点用的临时结点
    struct ll *s2 = head;

    //创建两个比较大小并插入的临时结点
    struct ll *s1 = head;
    while (scanf("%d", &a) != EOF)
    {
        if (a == 0) //题中的以 0 为结束
            break;
        struct ll *d = (struct ll *)malloc(sizeof(s
truct ll));
        d->data = a; //创建有效结点将值赋进去
        if (i != 0)  //比较需要从第二个数开始
        {
            s2 = s2->next;
            //s2 为前结点此步将 s2 提前与 s1 刚好相邻
            while (s2 != NULL)
            {
                if (s2->data > d->data)
                //当发现 s2 的值大于输入的值时 c
                {
                    s1->next = d;
                    //将新的结点放入 s2 与 s1 中间
                    d->next = s2;
```

```
                    break;
                }
                s1 = s1->next;
                s2 = s2->next;
            }
            if (s2 == NULL)
            //当发现 s2 等于空指针时，data 的值比前面的都大
            {
                Tail->next = d;
                //所以对其进行正常的结点增加操作
                d->next = NULL;
                Tail = d;
            }
            s1 = s2 = head;
        }
        else
        {   //对于首个结点无需比较大小
            Tail->next = d;
            //Tail 与头结点指向相同，Tail 中存有首结点的地址即头结点存有
            d->next = NULL;
            //将有效结点存有的地址初始化(未初始化将不能随意更改数据 参考野指针)
            Tail = d;
            //将 d 赋予 Tail 此时有 head 存有 Tail 的地址，后 Tail 作为临时结点，不断向前移动。
        }   //当存完时最后一个结点存有的地址将为 NULL
        i++;
         //此步 i 可以为非 0 的任意值
    }
    head = head->next;
    //将不含有效数据的头结点向前移
    while (head != NULL)
    {

        printf("%d ", head->data);
        //输入首个结点的数据
        head = head->next;
        //向前移
    }
    return 0;
}
```

链表比较难理解，必须多练多看。

此题考查对链表的应用,并且用插入结点方法建立有序的链表。

首先创建结构体、链表头结点、临时结点以创建链表。后在输入第二个有效数据后，与前面的数据进行比较，此时需要两个临时比较结点，在比较完后插入两个临时结点之间。

当比较完后无需插入的情况，需要和正常增加结点的操作相同。

4、

```
#include <stdio.h>
struct alltime
{
    int year, month, day;
    double h, m, s;
}; // 使用结构体保存日期时间数据
int calcday(int year, int month, int day, int r)
{ // 专门写了个函数来计算日期之间的天数
    int count = 0, yeartemp;
    month--;
    if ((r == 1)) // 同样有对闰年问题的处理
        while (month != 0)
        {
            switch (month)
            {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                count += 31;
                break;
            case 2:
                count += 29;
                break;
            case 4:
            case 6:
            case 9:
```

```
            case 11:
                count += 30;
                break;
            };
            month--;
        }
    else
        while (month != 0)
        {
            switch (month)
            {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                count += 31;
                break;
            case 2:
                count += 28;
                break;
            case 4:
            case 6:
            case 9:
            case 11:
                count += 30;
                break;
            }
            month--;
        };
        // 先把参数里面的月份和天数加起来，下面再整体加年天数
    yeartemp = year;
    while (yeartemp != 1900)
    {
        if ((yeartemp % 4 == 0 && yeartemp % 100 != 0) || (yeartemp % 400 == 0 && yeartemp % 3200 != 0) || yeartemp % 172800 == 0)
            if (yeartemp != year)
                count += 366;
            else
                count += 365;
        else
            count += 365;
        yeartemp--;
    }
    count += day - 1;
    // 最后减去 1 是为了抵消 1900 年 1 月 1 日
    return count;
}
int main()
{
    struct alltime time, totime;
    // time 用来放用户输入，totime 放间隔
    float tyear, tday; // 待会算年份精确值
    printf("请输入 1900 之后的年月日时(24 小时)分秒
");
    scanf("%d %d %d %lf %lf %lf", &time.year, &time.month, &time.day, &time.h, &time.m, &time.s);
    if (time.year >= 1900) // 只考虑了流逝的时间
        if ((time.year % 4 == 0 && time.year % 100 != 0) || (time.year % 400 == 0 && time.year % 3200 != 0) || time.year % 172800 == 0)
        { // 上面这一长串都是用来判断是否是闰年的
            printf("你所计算的年份是闰年\n");
            printf("距离 1900 年始流逝了: \n");
            tday = calcday(1900, time.month, time.day, 1);
            tyear = tday / 366.0; // 这些是把当前过去的天数除总天数得到今年过了多久
            // 本程序只精确年和时间，月份和天大家可自己试试精确
            totime.year = time.year - 1900;
            // 算年间隔
            totime.month = 12 * totime.year + time.month;
            // 算月间隔，没有做对后面的天作精确处理
            totime.day = calcday(time.year, time.month, time.day, 1);
            // 先把间隔的天数算出来 再算后面的秒，分钟和小时就很好算了
```

```
            totime.s = totime.day * 24.0 * 60.0 * 6
0.0 + time.h * 60.0 * 60.0 + time.m * 60.0 + time.s
;
            totime.m = totime.s / 60.0;
            totime.h = totime.m / 60.0;
            printf("%f 年\n%d 月\n%d 日\n%f 时\n%f 分
\n%f 秒
\n", totime.year + tyear, totime.month, totime.day,
 totime.h, totime.m, totime.s);
        }
        else
        {
            printf("距离 1900 年始流逝了: \n");
            tday = calcday(1900, time.month, time.d
ay, 1);
            tyear = tday / 365.0;

            totime.year = time.year - 1900;
            totime.month = 12 * totime.year + time.
month;
            totime.day = calcday(time.year, time.mo
nth, time.day, 1);
            totime.s = totime.day * 24.0 * 60.0 * 6
0.0 + time.h * 60.0 * 60.0 + time.m * 60.0 + time.s
;
            totime.m = totime.s / 60.0;
            totime.h = totime.m / 60.0;
            printf("%f 年\n%d 月\n%d 日\n%f 时\n%f 分
\n%f 秒
\n", totime.year + tyear, totime.month, totime.day,
 totime.h, totime.m, totime.s);
        }
}
```

本题使用结构体来实现计算两个日期之间的时间差，分别用不同的单位表示，为了简便，我这里只精确了年、时分秒，如果你感兴趣也可以自己去精确到月份和天。值得一说的是闰年的判断方法、以及如何在有闰年的情况累加流逝的天数，具体的解释已经写在了代码中。

第十章

史贝宁

一、填空题

1、流式文件

课本第 295 页是一条概念，感觉这种概念性的题目考试总会出几道来玩。

2、键盘　显示器

这个在书上也有，我去找找，在第 298 页的最上面，括号里面就是答案了。

3、wb+

请注意了，题目说要打开一个新的二进制文件，即可以读又可以写，那么我先告诉大家文件打开方式字符串是由三个字符组成的。

第一个字符是 r、w、a，分别是读、写、追加。读嘛，就得打开一个存在的文件去读。

写嘛，就不是打开去写了，而是新建一个去写，如果文件已经存在了，那就删了以后重新建立一个写。

追加，就是打开一个存在的文件，继续写。

第二个字符是 t、b，分别是文本、二进制文件。

文本文件嘛，里面存的是 ASCII 码，看得懂；

二进制文件，里面存的是 0 和 1，不太容易看懂；

第三个字符是+，就是读写的意思。

如果第一个是 r 最后+了，那么即可读又可写，如果第一个是 w 最后+了，那么写了之后也可读，如果第一个是 a 最后+了，同样写了之后也可读。

以上三个字符可以任意搭配，来实现你想要实现的文件操作，具体搭配可以看课本第 296 和 297 页的表格。那么这道题就不用说了吧，抓住题目的新和可读可写就 ok。

4、NULL

成功会返回这个文件的指针，失败则空地址 NULL。

5、读或读写

你想读一个文件，那么必须有读这个属性。

6、把位置指针从文件末尾处后退 30 个字节

fp 是文件指针，-是后退，30L 是 30 个字节，加 L 是为了说明 30 是个长整数型，2 是从文件末尾开始。

7、rewind　fseek　ftell

课本是个好东西，多翻多看 302 页真香。

8、0

课本 303 页的原话，在执行 fopen 函数时，ferror 的值会自动置 0。

二、简答题

1、 在 C 语言上，文件型指针就是指 FILE 类型的

指针，它指向一个文件类型的结构，结构里包含着该文件的各种属性。

首先定义一个文件类型的指针，然后使用 fopen 函数获取要打开文件的指针，将函数返回的文件指针赋值给指针变量。之后便可将此文件指针作为文件操作函数的参数，来访问文件。

2、 为了建立文件的各种信息，获取文件指针以便后续操作。

对于操作系统而言，文件操作的最高级别的目的，就是对于文件的独享权。没有打开文件之前，文件只是存在于硬盘上的数据集合。

当进行打开文件操作时，操作系统会建立文件登记表中记录该文件处于打开状态，除了申请进行打开文件操作的进程，其他程序不能对文件进行访问和写入。

同理，进行关闭文件操作时，操作系统会在文件登记表中清除该文件的打开状态，此时，其他的程序，才能获得文件的完全控制权。

及时关闭文件还可防止文件数据丢失。

可通过 fopen 函数和 fclose 函数打开或关闭文件。

三、编程题

1、

```c
#include <stdio.h>
int main() // 主函数
{
    FILE *fp;
    char ch;
    if ((fp = fopen("C:\\vc1.txt", "w+")) == NULL)
    {
        printf("Cannot open file!");
        getchar();
        return 0;
    }
    ch = getchar();
    while (ch != '@')
    {
        if (ch >= 65 && ch <= 90)
            ch += 32;
        fputc(ch, fp);
        ch = getchar();
    }
    fclose(fp);
}
```

涉及到文件的操作都需要使用 FILE 类型来定义一个指针，后续通过这个指针来操作文件。

首先我们需要打开一个文件，使用 fopen 函数，题目要求是 C 盘下的一个 vc6.txt 文件，你在运行期间如果直接调试很可能会打开文件失败，这是因为 C 盘写文件并不是所有软件都有这个权限的，解决办法就是先把可执行文件编译出来，然后找到它右键，选择以管理员权限打开即可。

文件打开方式的几种组合前面的解析已经说过了，判断 fopen 返回给 fp 指针的值是否是 NULL，如果是说明文件打开失败了，输出提示信息，成功打开文件之后就可以进行相关文件操作了。

接下来的事情就是获取字符、判断是否结束、判断大写如果是则转换小写、把字符使用函数 fputc 写入文件之中，循环以上该过程即可完成本题。

最后记得 fclose 把文件关闭，避免数据丢失。

2、

```c
#include <stdio.h>
int main() // 主函数
{
    FILE *fp1, *fp2;
    char ch;
    if ((fp1 = fopen(".\\vc1.txt", "rt")) == NULL)
    {
        printf("Cannot open file!");
        getchar();
        return 0;
    }
    if ((fp2 = fopen(".\\vc2.txt", "at+")) == NULL)
    {
        printf("Cannot open file!");
        getchar();
        return 0;
    }
    ch = fgetc(fp1);
    while (ch != EOF)
    {
        fputc(ch, fp2);
        ch = fgetc(fp1);
    }
    fclose(fp1);
```

```
    fclose(fp2);
}
```

题目要求要把一个文件的内容连接到另一个文件中去，所谓连接，就是继续在第二个文件后面输入内容，那么很明显第二个文件必须使用追加方式打开，第一个文件可读就行。

那么我们定义了两个文件指针分别指向两个文件，接下来要做的事情就是每从 fp1 中获取一个字符，判断是否获取到了字符，把获取到的字符写入第二个文件之中，以上循环即可完成连接的过程。

同样的，记得最后关闭。

3、

```
#include <stdio.h>
struct student
{
    int num;
    char name[10];
    int age;
} stu[20];
int main() // 主函数
{
    FILE *fp;
    int i;
    if ((fp = fopen(".\\data.txt", "wt+")) == NULL)
    {
        printf("cannot open file!");
        getchar();
        return 0;
    }
    for (i = 0; i < 20; i++)
        scanf("%d %s %d", &stu[i].num, &stu[i].name
, &stu[i].age);
    fwrite(&stu, sizeof(struct student), 20, fp);
}
```

因为程序本身没有给我们文件，所以我们自己来创造一个文件吧，这组代码的目的是写入一个有 20 个学生数据的文件，我随便输入了以下的内容：

```
01 a 1
02 b 2
03 c 3
04 d 4
05 e 5
06 f 6
07 e 7
08 h 8
09 i 9
10 d 10
11 j 11
12 o 12
13 j 13
14 h 14
15 n 15
16 o 16
17 i 17
18 n 18
19 b 19
20 i 20
```

接下来我们来实现题目的操作：

```
#include <stdio.h>

struct student
{
    int num;
    char name[10];
    int age;
} stu[20], *p;

int main() // 主函数
{
    FILE *fp;

    if ((fp = fopen(".\\data.txt", "rt")) == NULL)
    {
        printf("cannot open file!");
        getchar();
        return 0;
    }
    fread(stu, sizeof(struct student), 20, fp);
    p = stu;
    for (p += 1; p < stu + 10; p += 2)
    {
        printf("%d %s %d\n", p->num, p->name, p->ag
e);
    }
```

```
}
```

得到输出结果为：

```
2 b 2
4 d 4
6 f 6
8 h 8
10 d 10
```

好的，我们来说说本题的思路。先说刚才的第一段代码，在这段代码中，因为题目并没有给我存放有 20 个学生的数据给我去直接读取，所以我需要自己去写入一个有 20 个学生的数据，方法也很简单，创建一个结构体，里面有学生的学号姓名和年龄的变量，弄他 20 个，也就是结构体数组。

接着是打开文件的常规操作，这里打开方式是 wt+，然后循环 20 次每次咱自己输入数据，存放到结构体的数组中去。最后使用 fwrite 函数来把数据写入到文件中去。

好了现在我们有文件数据就可以继续做题了。题目要求让我们读取其中 2、4、6、8、10 的序号的学生数据显示在屏幕上，那好，我们先把所有的数据都存到结构体数组里，然后对应序号给他显示出来不久行了，多简单的事情对吧。fread 直接给他读取 20 次存到 stu 这个结构体数组中，接着你即可用指针操作来输出相应的序号，也可以用常规的操作，我使用了看起来非常高大上的指针操作。

4、

```
#include <stdio.h>
struct student
{
    int num, eng, math, comp, ping;

} stu[4];
int main() // 主函数
{
    FILE *fp;
    int i;
    if ((fp = fopen(".\\data.txt", "wt+")) == NULL)
    {
        printf("cannot open file!");
        getchar();
        return 0;
    }
    for (i = 0; i < 4; i++)
        scanf("%d %d %d %d %d", &stu[i].num, &stu[i].eng, &stu[i].math, &stu[i].comp, &stu[i].ping);
    fwrite(&stu, sizeof(struct student), 4, fp);
    fclose(fp);
}
```

和上一道题一样先自己构造一些输入，以下是我随便输入的：

```
1 12 13 14 84
2 34 84 28 48
3 84 47 84 28
4 84 84 85 99
```

然后我们就可以做题按照平时成绩来降序，然后存储进文件了。

```
#include <stdio.h>
struct student
{
    int num, eng, math, comp, ping;

} stu[4], *p, *q;
void change(struct student *p, struct student *q)
{
    struct student temp;
    temp = *p;
    *p = *q;
    *q = temp;
}
int main() // 主函数
{
    FILE *fp;
    int i;
    if ((fp = fopen(".\\data.txt", "rt+")) == NULL)
    {
        printf("cannot open file!");
        getchar();
        return 0;
    }
    fread(stu, sizeof(struct student), 4, fp);

    for (p = stu; p < stu + 4; p++)
    {
        for (q = p + 1; q < stu + 4; q++)
            if (p->ping < q->ping)
```

```
                change(p, q);
    }
    for (p = stu; p < stu + 4; p++)
        printf("%d %d %d %d %d\n", p->num, p->eng,
p->math, p->comp, p->ping);

    rewind();
    fwrite(stu, sizeof(struct student), 4, fp);
    fclose(fp);
}
```

最终的输出排序结果是这样的：

```
4 84 84 85 99
1 12 13 14 84
2 34 84 28 48
3 84 47 84 28
```

好的，前面自己弄点数据的过程我就不赘述了，在对数据进行排序的过程中，我构造了一个 change 函数，用来交换结构体数组之中元素的数据，第九章的知识告诉我们相同的结构体变量之间是可以相互赋值的，赋值之后结构体变量内的各种变量值也就相等了。

显示 fread 读取我们之前搞定的数据，然后就是排序，排序的时候用到了两个 for 循环，里面我都是通过指针来操作循环的，然后排序完成后进行输出展示，输出完成后 rewind 把文件内指针搞到首部，最后写入文件中即可。

马国智

以下代码使用 C++语言编写

三、编程题

one.cpp

```
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
/*
* 首先我们定义一个路径变量，定义变量的好处是方便程序员修改
* 处理好输入遇到@就退出，msg 是一个变量有可读字符（非@）就加在 msg 中
*/
void one()
{
    string filePath = "d:\\vc1.txt";
    ofstream out(filePath);
    string msg;
    while (1)
    {
        char a = '\0';
        cin >> a;
        if (a == '@')
        {
            break;
        }
        msg += a;
    }
    out << msg << endl;
    out.close(); // 清理资源
}
```

two.cpp

```
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
/*
* 掌握好 ifstream、ofstream 就可以了
*/
void two()
{
    string inPath = "d:\\vc1.txt";
    ifstream in(inPath.c_str());
    string outPath = "d:\\vc2.txt";
    ofstream out(outPath);
    string msg;
    while (in >> msg)
    { // 一行一行的读入
        out << msg << endl;
    }
    in.close();
    out.close();
}
```

three.cpp

```cpp
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
/* constexpr 和 const 是一样的，C++11 的新特性，优化比 const 更好一些
*  如果所在的编译器不支持这个可以改成 const
*  简单的 if-else 语句
*/
constexpr int N = 5;
void three()
{
    string path = "d:\\vc3.txt";

    // 如果原始没有数据，可以在这里输入
    //ofstream out(path);
    //for (int i = 0; i < N; ++i) { // 原文是要 20 个，这里从略写
    //  string msg;
    //  cout << "请输入学号、姓名、年龄" << endl;
    //  getline(std::cin, msg);
    //  out << msg << endl;
    //}
    ifstream in(path.c_str());
    if (in.fail())
    {
        cerr << "打开输入文件出错.\n";
        return;
    }
    char buff[64];
    int i = 0;
    while (in.getline(buff, sizeof(buff)))
    { // 一行一行的读，buff 大小跟根据实际大小来求解
        if (i == 2 || i == 4 || i == 6 || i == 8 || i == 10)
        {
            cout << "第" << i << "条信息" << buff << endl;
        }
        ++i;
    }
    in.close();
    //out.close();
}
```

four.cpp

```cpp
#include <algorithm>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
using namespace std;

/*
* 这道题明显是一道数据库的题目，这么出题，我不知道有啥意义
* 代码略显复杂，用到了很多 C++新特性的知识，见谅。
*/

// 注意题干是按平时成绩排名
void four()
{
    string path = "d:\\vc4.txt";
    ifstream in(path);
    char buff[64];
    vector<pair<int, string>> guard; // int 保存平时成绩,string 保存信息
    while (in.getline(buff, sizeof(buff)))
    {
        string msg = buff;
        //cout << buff;
        // 在第四个空格之后就是我们想要的平时成绩，所以只要找到第 4 个空格后面的分数就行了
        string tmp;
        // foreach 循环
        int num = 0;
        for (int i = 0; i < msg.size(); ++i)
        {
            if (msg[i] == ' ')
            {
                // 防止多余的空格
                while (msg[i++] == ' ')
                {
                }
```

```
                --i; // 因为多加了一个 i
                ++num;
            }
            if (num == 4)
            {
                tmp = msg.substr(i, msg.size() - i)
;
                guard.emplace_back(stoi(tmp), msg);
                break;
            }
        }
    }
    // lhs 左手边 rhs 右手边，algorithm 头文件里的排序函数,lambda 表达式,
    // 这个 lambda 很重要,写起来让你的代码更得心应手
    sort(guard.begin(), guard.end(), [](const pair<int, string> &lhs, const pair<int, string> &rhs) {
        return lhs.first > rhs.first;
    });

    string outPath = "d:\\vc4_1.txt";
    ofstream out(outPath);
    for (auto val : guard)
    {
        out << val.second << endl;
    }
    in.close();
}
```

第十一章

余传钦

答案速对 1-5 CDBCD 6-10 BCBBA

一、选择题

1、C C 程序在编译过程中对预处理命令进行处理。

2、D 宏名一般用大写，但不是强制规定。

3、B 在本章开篇第三段有介绍，预处理命令与 C 语句是有区别的，而宏定义是不需要在行末加分号；C 选项在进行宏定义时，可以应用已经定义的宏名，可以层层置换。

4、C 课本 323 页顶部。

5、D 系统在进行编译时，系统会自动在编译前，对预处理处理。

6、B 主要是计算出 t 的值，MIN 的结果是 10，最终结果是 10*10。

7、C 先展开 NUM =（2+1+1）* 2 + 1 / 2；算出 NUM 的值为 8；注意 i 的初始值为 1；到 i= 8；共循环 8 次。

8、B 分析程序执行 a = (2.84 + 5 * 2)；pritnf ("%d", (int)(a));强制转换，将浮点型转换为整型输出值为 12；

9、B 分析程序，sum = m + n + m + n * 2；算出 sum= 10；

10、A 分析程序，MAX 得出是 5 再乘 100；得出 t 值为 500；

二、填空题

1、从宏名定义开始到文件结束

2、#undef

3、文件包含

4、#include

5、1000, 40

i1 = 1000/5*5 = 1000，i2 = 1000/(5*5) = 40；

6、c

7、11

NY = 2 + 3 * 2 + 3 = 11

8、70

7*10 = 70

9、MIN

a=5, b= 6; a<b; 所以输出最后一条语句

10、0, 1

1

因为之前已经定义过 TEST，所以先执行程序段 printf("%d, %d\n",x, y)；再执行 printf（"%d\n",z)。

三、分析题

1、OK!OK!OK!

循环中 i 初始值为 0；循环 3 次，在执行预处理中的 printf("OK!")；之后再输出换行。

2、10

DEBUG 初始值为 1，所以执行语句 printf("%d\n", a);

3、2

输出较大值。

四、编程题

1、

```
#include <stdio.h>
```

```c
#define DIVIDE(a, b) a % b

int main(void)
{
    int a, b, t;
    scanf("%d %d", &a, &b);
    t = DIVIDE(a, b);
    printf("%d", t);
    return 0;
}
```

2、

```c
#include <stdio.h>
#define MAX(a, b, c) a > b ? (a > c ? a : c) : (b > c ? b : c)

int main(void)
{
    int a, b, c;
    scanf("%d%d%d", &a, &b, &c);
    int t = MAX(a, b, c);
    printf("%d", t);

    return 0;
}
```

3、

```c
#include <stdio.h>
#define ABS(m) a > 0 ? 1 : 0

int main(void)
{
    int a;
    scanf("%d", &a);
    if (ABS(a))
        printf("true");
    else
        printf("false");
    return 0;
}
```

4、

```c
#include <stdio.h>
#define PI 3.14159
#define VOL(r) (PI * r * r * r)

int main(void)
{
    int r;
    scanf("%d", &r);
    double V = VOL(r) * 4.0 / 3.0;
    printf("%.2f", V);

    return 0;
}
```

注意：4.0/3.0 不可以写成 4/3。

5、

```c
#include <stdio.h>
#define MIN(a, b) a < b ? a : b

int main(void)
{
    int num[10], min = 0;
    int i;
    for (i = 0; i < 10; i++)
    {
        scanf("%d", &num[i]);
    }
    for (i = 0; i < 9; i++)
    {
        min = MIN(num[i], num[i + 1]);
    }
    printf("%d", min);
    return 0;
}
```

注意：最后一个循环中 i<9,而不是 10。

马国智

以下代码使用 C++语言编写

四、编程题

```c
#define one(a, b) a % b;

#define two(a, b, c) (a > b ? (a > c ? a : c) : (b > c ? b : c))

#define three(a) (a > 0 ? 1 : 0)
```

```
#define four() (4 * 3.14 * 3 * 3)

// 第五题调用的方法
#define min(a, b) (a < b ? a : b)

int twoFuntion(int a, int b, int c)
{
    a = a > b ? a : b;
    return a > c ? a : c;
}
#include <iostream>
using namespace std;

void testThree()
{
    int num = 0;
    // 输入一个数字
    cout << "请输入一个数字" << endl;
    cin >> num;
    if (three(num) == 1)
    {
        cout << "数为正数" << endl;
    }
    else
    {
        cout << "数为负数" << endl;
    }
}

void testFive()
{
    int arr[5] = {5, 4, 3, 6, 10};
    int min = INT_MAX; // #define INT_MAX   2147483
647 语言自带的
    // 防止数组下标越界，所以要减个 1
    for (int i = 0; i < 5; ++i)
    {
        min = min(arr[i], min);
    }
    cout << "最小值为:" << min << endl;
}
```

```
int main()
{
    int ans = 0;
    // 调用的方法
    //ans = one(2, 3);
    //ans = two(30,31,90);
    //ans = twoFuntion(30, 31, 90);

    // 第三题调用方法
    //testThree();

    // 第四题
    //ans = four();

    // 第五题
    testFive();

    // 输出最后的结果
    cout << ans;
    return 0;
}
```

main.cpp

```
// 链接程序
extern void one();
extern void two();
extern void three();
extern void four();
int main()
{
    four(); // 用哪个函数调用哪个就行
    return 0;
}
```

第十二章

万鹏辉

答案速对

1-5　BDABC　6-10　BDAAC

11-15　DDDDB　16-20　CBDDA

一、选择题

1、B　面向对象程序设计具有抽象性、封装性、继承性、多态性四个基本特性。数据相关性是指数据之间存在某种关系。

2、D　函数的模板实际上是建立一个通用函数，不具体指定函数类型和形参类型，用一个虚拟的类型代表，这样只需要定义一次，就可以处理多种不同类型的数据，也就完成了对某一类数据的处理算法应用到另一类数据的处理中。

3、A　C++在最初也被叫做带类的 C，这也是 C++与 C 的根本不同。

4、B　动态内存分配会在函数或程序结束时自动释放内存，更合理的使用内存。

5、C　const 关键字可以限定一个变量不允许被改变，一定程度的提高了程序的安全性，而可维护性包括两个方面，而其中之一就是在预防和纠正方面的难易程度，安全性越强，就可以更好的预防，可维护性越强。

6、B　函数重载可以在同一函数名的情况下改变形参而改变函数内容，更方便，可读性也越强。

7、D　name 表示 name[]数组的首原素地址，所以是地址调用。

8、A

9、A　const 修饰类的成员函数，则该成员函数不能修改类中任何非 const 成员函数。一般写在函数的最后来修饰。

10、C　构造函数无返回类型。

11、D　this 是一个指针，它时时刻刻指向你这个实例本身，类型应该为本身类型的指针。

12、D　内存泄漏是指程序中已动态分配的堆内存由于某种原因程序未释放或无法释放，造成系统内存的浪费。

马国智：第十二题我认为没有一个是对的，A 和 B 不去解释了，C 选项是栈是由系统分配的，作用域结束后系统回回收栈的内存比如这样的一个{}就是一个作用域，}走掉之后，{}栈的内存就被回收了。

第四个选项，是初学 C++最容易犯的错误，因为听了太多 C++会产生内存泄露，我刚学那会对这个也是半蒙的，也在思考到底什么是内存泄露？我对内存泄露的理解是这样的，比如我的服务器是要一直跑着的有一个死循环在跑着，而 p 的值显然是在 doSomething 中用不到的，实际上 p 也是一点用也没有了，但是他一直在占着这块内存，没有 delete 释放掉，所以这是内存泄露。而不是进程（可执行的程序）都退出了没有 delete 才是内存泄露。

当你的进程（一个可执行的文件）退出了，操作系统会自动回收给进程分配的资源，你的代码中 new 的资源就算没有 delete 变量，操作系统是会帮你直接回收的。但有一种情况特殊，就是创建了内核的某个对象没有及时的释放，造成了操作系统的内存泄露，这个解决办法就只有重启了。综上所述，我认为没有合理答案，如果强行选答案的 D 还较为合理，这是刚找的博客链接：
https://blog.csdn.net/wallezhe/article/details/69385474

13、D　首先易知是一个函数，因为可以在外部直接调用所以是公有函数。

14、D　作用域运算符的功能是标识某个成员属于哪个类，避免出现二义性。

15、B　静态成员函数，普通的成员函数一般都隐含了一个 this 指针，this 指针指向类的对象本身，因为普通成员函数总是具体的属于某个类的具体对象的。通常情况下，this 是缺省的。如函数 fn（）实际上是 this->fn（）。但是与普通函数相比，静态成员函数由于不是与任何的对象相联系，因此它不具有 this 指针。

16、C　C++形式接口，只需要做到代码级重用，那么可以选择接口继承或者函数重载等方法。

17、B　公有继承的派生类对象可以访问基类的公有成员和受保护成员，亲测可以。

18、D　设置虚基类可以通过使多个派生类的基类副本只有一个，消除了二义性。

19、D　纯虚函数是一个没有具体实现的虚函数，抽象类指至少包括 1 个纯虚函数的类，其纯虚函数的实现在派生类中给出，抽象类不能被实例化，也就是不能声明一个抽象类的对象。

20、A　友元函数是可以直接访问类的私有成员的非成员函数。它是定义在类外的普通函数，它不属于任何类，但需要在类的定义中加以声明，声明时只需在友元的名称前加上关键字 friend，显然运算符重载不是友元函数。

二、填空题

1、继承　多态

2、对象

3、封装　继承

4、

第一空：

Journal default constructor

《Computer Design》default constructor

Subscribing 《Computer Design》
Reading 《Computer Design》
第二空：
《Computer Design》default destructor
Journal default destructor

第一空，主函数内第一句,定义对象 journal1 时运行 CComputerDesign 类的构造函数，先运行基类的构造函数，输出 Journal default constructor 再运行自身的构造函数，输出<<Computer Design>>default constructor；主函数内第二句，定义了一个 CJournal 类的指针（不会运行构造函数）；主函数内第三句，执行对象 journal1 中的 subscribe（）函数，输出 Subscribing <<Computer Dsign>>；主函数内第四局同理，输出 Reading<<Computer Design>>，c++派生类要是有和基类相同的函数名，派生类的对象调用这个函数是调用派生类重写的函数。

第二空，运行析构函数的结果，由于是放入堆中，先运行自身的再基类的。

三、编程题
以下答案来自参考资料[2]
后文有马国智学长提供的答案。
1、

```
#include <iostream>
#include <string.h>
using namespace std;
class CCourse
{
private:
    long no;
    char *p_name;
    float credit;

public:
    CCourse(long no_val, char *p_val, float credit_
val);
    CCourse(const CCourse &r_course);
    ~CCourse() { delete p_name; }
    void print() const;
};
```

```
CCourse::CCourse(long no_val, char *p_val, float cr
edit_val)
{
    no = no_val;
    p_name = new char[strlen(p_val) + 1];
    strcpy(p_name, p_val);
    credit = credit_val;
}

CCourse::CCourse(const CCourse &r_course)
{
    no = r_course.no;
    p_name = new char[strlen(r_course.p_name) + 1];
    strcpy(p_name, r_course.p_name);
    credit = r_course.credit;
}

void CCourse::print() const
{
    cout << "Course number: " << no << endl;
    cout << "Course name: " << p_name << endl;
    cout << "Course credit: " << credit << endl;
}
```

2、

```
//在 class CCourse 定义中增加一条：
private:
static int total_course;

//在类外部增加一条：
int CCourse::total_course = 0;
//在 CCourse 类的构造函数中增加一条：
total_course++;
//在 CCourse 类的拷贝构造函数中增加一条：
total_course++;
//在 CCourse 类的析构函数中增加一条：
total_course--;

//在 class CCourse 定义中增加一条：
public:
static getTotalCourse() { return total_course; }
//在 class CCourse 定义中增加一条：
```

```
friend char *getCourseName(const CCourse &r_course)
;

//在类外部定义:
char *getCourseName(const CCourse &r_course)
{
    return r_course.p_name;
}
```

3、

```
//在 class CCourse 定义中增加一条:
public:
bool operator<(const CCourse &r_course);

//在类外部定义:
bool CCourse::operator<(const CCourse &r_course)
{
    if (credit < r_course.credit)
        return true;
    else
        return false;
}

//在 class CCourse 定义中增加一条:
public:
bool operator>=(const CCourse &r_course);

//在类外部定义:
bool CCourse::operator>=(const CCourse &r_course)
{
    if (credit >= r_course.credit)
        return true;
    else
        return false;
}
```

4、

```
class CHLP : public CCourse
{
private:
    char *p_openby;

public:
    CHLP(long no_val, char *p_val, float credit_val
, char *p_open) : CCourse(no_val, p_val, credit_val
)
    {
        p_openby = new char[strlen(p_open) + 1];
        strcpy(p_openby, p_open);
    }
    ~CHLP() { delete p_openby; }
    void studyFor() { cout << "Study for structured
 programming" << endl; }
};

class COOP : public CCourse
{
private:
    char *p_openby;

public:
    COOP(long no_val, char *p_val, float credit_val
, char *p_open) : CCourse(no_val, p_val, credit_val
)
    {
        p_openby = new char[strlen(p_open) + 1];
        strcpy(p_openby, p_open);
    }
    ~COOP() { delete p_openby; }
    void studyFor() { cout << "Study for object ori
ented programming" << endl; }
};
```

5、

```
//在 class CCourse 定义中增加一条:
public:
virtual void studyFor() { cout << "study for degree
\n"; }

//增加:
#include <stdlib.h>

//主函数可定义为:
void main()
{
    char choice, instructor[10];
```

```
    float credit;
    long id;
    CCourse *p_course;

    cout << "Select course:\n";
    cout << "1. for High Level Language Programming
\n";
    cout << "2. for Object Oriented Programming\n";
    cin >> choice;
    cout << "Enter course number: ";
    cin >> id;
    cout << "Enter credit: ";
    cin >> credit;
    cout << "Enter instructor name: ";
    cin >> instructor;

    switch (choice)
    {
    case '1':
        p_course = new CHLP(id, "高级语言程序设计
", credit, instructor);
        break;
    case '2':
        p_course = new COOP(id, "面向对象程序设计
", credit, instructor);
        break;
    default:
        exit(0);
    }
    p_course->studyFor();
    delete p_course;
}
```

6、

```
#include <iostream>
using namespace std;

template <class T>
void swap(T &a, T &b)
{
    T temp;

    temp = a;
    a = b;
    b = temp;
}

template <class T>
void bubbleSort(T a[], int n)
{
    int i, j;

    for (i = 1; i < n; i++)
        for (j = 0; j < n - i; j++)
            if (a[j] > a[j + 1])
                swap(a[j], a[j + 1]);
}

template <class T1>
void print(T1 a[], int n)
{
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << endl;
}

void main()
{
    int a[] = {2, 3, 1, 6, 43, 22};
    double b[] = {2.3, 3.2, 1.6, -6.0, 4.3, 2.2};

    print(a, 6);
    bubbleSort(a, 6);
    print(a, 6);

    print(b, 6);
    bubbleSort(b, 6);
    print(b, 6);
```

马国智

以下代码使用 C++语言编写

三、编程题

main.cpp

```
#include "CGoods.h"
```

```
#include <iostream>
using namespace std;
template <typename T>
static void Swap(T &lhs, T &rhs)
{ // 通过引用来修改值
    int tem = lhs;
    lhs = rhs;
    rhs = tem;
}

// 第二个模板参数是数组的个数
template <int size, typename T1>
void BubbleSort(T1 (&arr)[size])
{
    for (int i = 0; i < size; ++i)
    {
        for (int j = i + 1; j < size; ++j)
        {
            if (arr[i] > arr[j])
            {
                Swap(arr[i], arr[j]);
            }
        }
    }
}

int main()
{
    // 做一个 const 的强制转换，有的 ide 会把 char*这要的字符串看成 const char*
    // 变量都是随意写的。
    /*CGoods cg1(996,const_cast<char*>("book"),20,5,20,40);
    CGoods cg2(997, const_cast<char*>("mouse"), 120, 5,20,40);
    cout << "cg1.getName:" << getPName(cg1)<<endl;
    if (cg1 < cg2) {
        cout << "cg2 大于 cg1" << endl;
    }
    else {
        cout << "cg1 大于 cg2" << endl;
    }*/
    int arr[5] = {10, 8, 43, 15, 21};
    BubbleSort<5, int>(arr); // 要显式的调用
    // 验证一下
    for (int i = 0; i < 5; ++i)
    {
        cout << arr[i] << ' ';
    }
    return 0;
}
```

CClothes.cpp

```
#include "CClothes.h"
#include <iostream>

using namespace std;
void CClothes::usedFor()
{
    cout << "商品信息...." << endl;
}
```

CClothes.h

```
#pragma once
#include "CGoods.h"
class CClothes : public CGoods
{
public:
    // CGoods 好好研究一下，我相信仔细研究的读者一定也会写了。读者这里可以自行解决
    // 要学好编程，一定要多写多练多总结。如果为了考试，这么多东西记也是记不住的。
    //CGoods::CGoods(int longNo,char* p_name, int price,int count, long total_goods,double total_price)
    void usedFor();

private:
    char *p_brand;
};
```

CFood.cpp

```
#include "CFood.h"
#include <iostream>
```

```
using namespace std;
void CFood::usedFor()
{
    cout << "商品信息...." << endl;
}
```

CFood.h

```
#pragma once
#include "CGoods.h"
class CFood : public CGoods
{
public:
    void usedFor();

private:
    char *p_brand;
};
```

CGoods.cpp

```
#include "CGoods.h"

char *getPName(CGoods &cg)
{
    return cg.p_name;
}

// 尽管变量名字是一样的，但一个是表示函数变量名。一个是类的成员变量名
CGoods::CGoods(int longNo, char *p_name, int price, int count, long total_goods,
               double total_price)
    : longNo(longNo), p_name(p_name), price(price), count(count),
      total_goods(total_goods), total_price(total_price)

{
}
CGoods::CGoods(CGoods &cg)
{
    // this 表示当前的类对象
    this->longNo = cg.longNo;
    this->p_name = cg.p_name;
    this->count = cg.count;
    this->price = cg.price;
    this->total_price = cg.total_goods;
    this->total_goods = cg.total_goods;
}
CGoods::~CGoods()
{
}

bool CGoods::operator<(const CGoods &cg)
{
    return this->price < cg.price;
}

bool CGoods::operator>=(const CGoods &cg)
{
    return this->price > cg.price;
}
```

CGoods.h

```
#pragma once // 头文件的预处理
class CGoods
{
    // 动态联编使用虚函数
public:
    CGoods() = default; // 让编译器自动生成一个构造函数
    CGoods(int longNo, char *p_name, int price, int count, long total_goods,
           double total_price);
    CGoods(CGoods &cg);
    friend char *getPName(CGoods &cg);
    int getCount() const { return count; } // 又不改变值用 const，能用 const 就用 const
    bool operator<(const CGoods &cg);
    bool operator>=(const CGoods &cg);

    long getTotalPrice() const { return total_price; }
```

```
    double getTotalGoods() const { return total_goo
ds; }

    ~CGoods();

private:
    double price;
    int longNo;
    char *p_name;
    int count;
    long total_goods;
    double total_price;
};
```

感谢你看到这里，江理一起来学倾情奉献。

《C/C++语言程序设计》章节例题分析与注释

史贝宁

群文件可下载例题源代码

第一章

例 1.1

```c
#include<stdio.h>
void main()
{
    printf("programming is interesting!\n");
}
```

例 1.2

```c
#include <stdio.h>
int fab(int n); //函数被写在了 main 之后，需要提前声明你定义了一个新的函数
void main()
{
    int n, i; //书上有这个 i，但是这个 i 在整个程序中没有被用到
    printf("请输入几个月整数值: ");
    scanf("%d", &n);
    printf("num=%d", fab(n)); //直接输出 fab(n)到%d 的位置，因为 fab 函数是一个整数型，返回值是整数，而%d 表示整数，故符合
    scanf("%d", &n);
}

int fab(int n) // 这是我们新定义的函数，用于计算递归问题
{
    if (n == 1 || n == 2)
        return 1; // 满足条件执行后面的 return，即返回给调用它的函数 scanf 一个整数 1
    else
        return fab(n - 1) + fab(n - 2); // 不满足条件则会转到 else 后继续执行，可以看到这里依然调用了 fab 函数
    // 可谓是自己调用自己一直算算算，这就是用编程语言实现递归，想算后面的必须知道前面的。
```

```
    // 用数学来表达就是 f(12)=f(11)+f(10) 而 f(11)=f(10)+f(9) 一直递推 最后可写成很多个 f(1)或 f(2)相加
}
```

1.5 编程实践

任务一

```
#include<stdio.h>
int main() // 有的是 void 即无返回值 有的是 int 整数型 视程序所需要的情况而定
{
    printf("           *          ");
    printf("          ***         ");
    printf("         *****        ");
    printf("        *******       ");
    printf("       *********      ");
    printf("      ***********     ");
    return 0; // 因为定义成了 int，所以我们要返回一个整数
}
```

任务二

```
#include <stdio.h>
void main()
{
    int fahr, cel;
    int lower, upper, step;
    lower = 0;
    upper = 300;
    step = 20;
    printf("\t==华氏和摄氏温度对照表==\n"); //\t 是水平制表符
    printf("\n\n 华氏温度   摄氏温度\n------------\n");
    fahr = lower;
    while (fahr <= upper) // 循环语句，当满足括号内的语句时就执行下面花括号内的语句
    {
```

```
        cel = 5 * (fahr - 32) / 9; // 运用数学关系进行运算
        printf("%d\t\t%d\n", fahr, cel); // 输出对应的华氏温度和摄氏温度
        fahr = fahr + step; // 对华氏温度进行加 20 的操作 以便于下一个数的循环
    }
}
```

第二章

例 2.1

```
#include <stdio.h>
#define PI 3.14 // 本节知识点是要学会定义常量
void main()
{
    float r, s, v;
    printf("请输入半径 r: ");
    scanf("%f", &r);
    s = 4 * PI * r * r;
    v = (4.0 / 3) * PI * r * r * r; // 数据类型为 float 单精度实数型，也就是有小数
    // 如果这里输入 4 则 4/3 结果为 double 型 1.0 输入 4.0 则为 double 型 1.33333333333333
    // 因为 4/3 是两个整数相除，结果仍为整数，系统在发现前后数据结构不同的情况下会自动进行类型转换
    // 在课本 2.3.1 节可知 int 会直接转换为 double 型，转换过程中原有结果的小数点会直接丢失
    printf("s=%f,v=%f\n", s, v);
}
```

例 2.2

```
#include<stdio.h>
void main()
{
    int a = 30000; // 数据长度 4B 取值范围 -2147483648 ~ 2147483647
    short int b = 20000; // 数据长度 2B 取值范围 -32768~32767
    long int c = 1234567890; // 数据长度 4B 取值范围-2147483648 ~ 2147483647 可知 int 如果不声明具体类型则为 long
```

```
    unsigned int d = 25; // 数据长度 4B 取值范围 0 ~ 4294967295 没有了复数 整数可取的更多了
    // 数据长度越长 可存储的值就越精确 当然占用的存储空间也就越大
    printf("%d\t%d\t%d\t%u\n", a, b, c, d);
}
```

例 2.3

```
#include<stdio.h>
void main()
{
    char c1 = 'A', c2 = 'a'; // 定义两个字符变量 注意区分字符用'' 字符串用""
    int x = 89, y = 56; // 定义两个整数型变量
    printf("%c\t%c\n", c1, c2); // %c 代表以字符形式输出后面对应的 c1 和 c2
    printf("%d\t%d\n", c1, c2); // %d 代表以整数形式输出后面对应的 c1 和 c2
    printf("%d\t%d\n", x, y); // 同上
    printf("%c\t%c\n", x, y);
    // 这里需要理解的是字符'A'与整数 65 是等价的
    // 因为字符其实在系统内是用一种叫 ASCII 码来对应的，可见课本附录
    // 所以系统会自动根据你想要输出的类型 输出整数 还是输出整数所对应的字符
    // 此外系统根据数据长度来判断存储的是整数还是字符
    // 字符'A'的 ASCII 码：0100 0001 一个单位长度 1 B=8 个二进制数
    // 整数 65 的 ASCII 码：0000 0000 0100 0001 两个单位长度
}
```

例 2.4

```
#include<stdio.h>
void main()
{
    float x = 256.012345678, m; // 定义一个单精度实数型变量 float 也就是浮点的意思嘛 还定义了一个 m 只不过没有赋予值
    double y = 1234567890.1256789, n; // 定义一个双精度实数型变量 double 双嘛 还定义了一个 n 也没有赋予初值
    m = x + y; // 先计算 x+y 的值 也即单精度加双精度 系统会自动变成双精度加双精度 然后赋给 m 这个单精度 m 精度低 所以会丢失一部分数据
    n = x + y; // 这个的话 n 是个双精度 所以最后的数据计算出来是什么 n 里面就是什么 大概率不会丢失
```

```
    printf("x=%f\n", x); // %f 意思是让 x 以浮点数的形式输出 也就是有小数点的形式 发现结果与初值不同 数据发生了丢失
    // 因为单精度只提供七位有效数字 往后面的就不能保证了
    printf("y=%f\n", y); // 这个没毛病 双精度能提供 15-16 位有效数字
    printf("m=%f\n", m); // 这个也会发生数据丢失 前面提到了 m 是单精度
    printf("n=%f\n", n); // 这个没毛病 双精度 有效数字很多 放得下
}
```

例 2.5

```
#include<stdio.h>
void main()
{
    double l; // 定义一个叫 l 的双精度实数型变量 注意 l 很像 1 要区分一下 1 是不能当作变量名称的
    int e = 5; // 整数型变量
    float h = 6.24; // 单精度实数型变量
    double f = 69.5; // 双精度实数型变量
    char d = 'D'; // 字符变量
    l = (8 - d) + e * h + f / e; // l 是最开始定义的双精度实数型变量 先运算括号 然后乘除 然后加减
    // d 是个字符变量 如果参与了运算 则把字符所对应的整数值拿来运算 D 对应的 ASCII 码是 68
    printf("%f\n", l); // 算完输出 以%f 就是浮点型输出 带小数那种形式就是
}
```

例 2.6

```
#include<stdio.h> // 这串代码可读性很差 因为变量用 abcd 这些字母谁知道是啥意思啊 不直观 以后大家自己写代码还是写的直观一些
void main()
{
    short int t;
    char b;
    long h;
    float f;
    double d;
    int e;
```

```
    t = 65; // 短整数
    b = 'A'; // 字符变量
    f = 12.64; // 单精度
    e = 100; // 整数
    d = e; // 双精度 = 整数 即 int 转换为 double
    e = f; // 整数 = 单精度 即 float 转换为 int
    h = t + b; // 长整数 = 短整数 + 字符型 即 short 和 char 转换为 int 然后赋值再转换为 long int 型
    f = b; // 单精度 = 字符 即 char 到 float 型
    b = t; // 字符 = 短整数 即 short 到 char 型 把整数对应的字符赋给 b 这个字符变量 这里大家应该理解字符和整数是等价的了吧
    printf("%f\t%d\t%ld\t%f\t%c\n", d, e, h, f, b); // 分别输出即可
    // 其中%ld 是按照长整型或双精度型输出 %hd 就是短整型 后面第三章会学到
}
```

例 2.7

```
#include<stdio.h>
void main() // 本例旨在让大家学会强制类型转换 即在变量名称前加上(类型) 例如 (int)
{
    int a, b, c, d;
    float x = 6.46, y = 8.57, z = 7.68;
    a = (int)x; // 将 float 单精度变量 x 强制转换成 int 型赋值给整数型变量 a
    b = (int)x + (int)y + (int)z; // 分别强制转换 然后加起来 赋值
    c = (int)(x + y + z); // 括号永远有最先优先权 先加起来括号的 然后再把结果强制转换 再赋值
    d = (int)x + y + z; // 只强制转换 x 为 int 然后加起来 最后因为 y z 是单精度 数据会丢失 只剩下一个整数
    printf("%d\t%d\t%d\t%d\n", a, b, c, d);
    printf("%f\t%f\t%f\n", x, y, z);
}
```

例 2.8

```
#include<stdio.h>
void main()
{
```

```
    int a = +7; // 很简单 一个正数赋值给变量
    int b = -9; // 一个负数负值给变量 负号不能省略 废话 正号可以省略 这是正话
    printf("%d\t%d\t%d\n", a, b, -a * b); // 输出结果 7    -9    63
    // 再次强调这个\t 是水平制表符的意思 就是让你的输出结果在\t 的位置来点空格不至于挤在一起
}
```

例 2.9

```
#include<stdio.h>
void main()
{
    int a = 6, b = 7;
    int i, j;
    i = a++; // 此处先让 i=a 然后 a 再自增 即 i=6 a=7
    j = ++b; // 此处先让 b 自增 然后再让 j=自增后的 b 即 b=8 j=8
    printf("%d\t%d\n", -i++, -(++j)); // 输出-i 即 -6 然后 i 自增 i=7 接着先 j 自增 j=9 输出-j 即 -9
    printf("%d\t%d\t%d\t%d\n", a, b, i, j); // 现在 a=7 b=8 i=7 j=9
    // 搞清楚 ++i 就是先自增 再运算别的 然后 i++ 就是先运算别的 再自增就好
    // 所谓自增就是 i=i+1
}
```

例 2.10

```
#include<stdio.h>
void main()
{
    int a = 5;
    int f, h, g;
    f = 18 - a++; // 先计算 后自增 18-5 然后 5+1
    printf("f=%d", f);
    printf("a=%d\n", a);
    h = ++a + 6; // 先自增 后计算 6+1 然后 6+7
    printf("h=%d", h);
```

```
    printf("a=%d\n",a);
    g = ++a + a++; //先自增 7+1 然后两个 a 相加为 16 然后 a 再自增 为 9
    // 这里不同的编译器结果不一样 大家知道怎么用这个自增自减就好了
    printf("g=%d", g);
    printf("a=%d\n",a);
}
```

例 2.11

```
#include<stdio.h>
void main()
{
    int a = 6, b = 5, x, y;
    x = (a++) + (a++) + (a++); // 这段程序想告诉我们自增的重复使用会有副作用
    y = (++b) + (++b) + (++b); // 比如我用 VS 执行的结果是 21 22 和书上两种情况还不一样
    // 所以没事别用两个及以上的自增自减 就这么简单
    printf("%d\t%d\n", x, y);
}
```

例 2.12

很简单，就不赘述了。

需要注意的是求余运算符%，参与运算的数据必须是整数，取商最大时的余数，余数的符号与被除数的符号相同。例如-7%3 结果为-1。

例 2.13

```
#include<stdio.h>
#include<math.h>
void main()
{
    float x, y, z;
    printf("请输入实型变量 x 和 y 的值，x 不等于 y: \n");
    scanf("%f %f", &x, &y); // 获取用户输入的两个数 这两个数可以是空格分开的 也可以是回车分开的 分别存储到 x 和 y 中
    z = fabs(y) / (2 * x + 4 * pow(y, x)); // fab 和 pow 函数是 math.h 提供的两个数学函数 每个函数参数里面写什么都会有文档告诉你的 不知道就查或百度
    printf("z=%f\n", z);
```

```
}
```

例 2.14

```
#include <stdio.h>
void main()
{
    int x, y, i, j;
    x = 6;
    y = 7;
    i = 8;
    j = 9;
    printf("%d ", x + y > i + j);         // 一旦引入关系运算符 最后输出的结果一定是关系运算符的结果 即 0 或 非 0
    printf("%d ", (x = 6) != (i = 8));    // 实际计算 6!=8 吗? 是的 输出 1 关系运算符两侧的表达式都是用来判断关系的 在这里他们不进行输出 只参与运算
    printf("%d ", (x == 6) != (i == 8)); // 括号里也是关系运算符 实际计算 1 不等于 1 吗? 结果是假 0
    printf("%d ", (x <= y) == (i <= j));
    printf("%d ", 'A' > 'a');
}
```

例 2.15

```
#include <stdio.h>
void main()
{
    int a, b; // 创建整型变量 a 和 b
    double i, j; // 创建双精度实数型变量 i 和 j
    printf("请输入整型变量 a、b 的值: \n");
    scanf("%d %d", &a, &b); // 获取用户输入的两个整数 %d 为获取整数
    // &a 其实是将用户输入的数据 存到&号之后的变量之中 &为获取变量对应的内存地址 存到这个变量 (变量的地址) 里
    printf("请输入实型变量 i、j 的值: \n");
    scanf("%f %f", &i, &j); // 同上
    printf("%d\t", a > b); // 结果为关系运算的结果 0 或 非 0 即 假或真
    printf("%d\t", 'g' > 'X'); // 同上 在 ASCII 码中 g 为 103 X 为 88 故 关系表达式结果为 1
```

```
    printf("%d\t", b / a * a == b); // 同下
    printf("%d\t", j / i * i == j); // 同样是关系表达式
    printf("%d\n", 0 < a < 100); // 注意此处不和数学一样表示区间 而是关系表达式 表示 a 同时满足两个不等式
}
```

例 2.16

```
#include <stdio.h>
void main()
{
    int s, r, t; // 先定义了三个变量 整数型的
    float x; // 然后定义了一个单精度实型变量
    s = 0;
    r = 4;
    t = 0;
    x = 6.5;
    // 三种逻辑分别是 非 与 或 分别是这样的 非与原来相反 与的话全真为真 一假为假 或的话一真为真
    printf("%d\t", s && r && t); // 逻辑运算的结果同样是真或假 即 0 或 非 0 本行计算 s 与 r 结果为 0 r 与 t 已经无需运算 一侧已经假了 必定为假
    printf("%d\t", s || r || t); // 逻辑或运算 0 或 4 当然是非 0 结果为真 t 为 0 但是前式为真 ok 整个式子为真
    printf("%d\t", !s); // s 本是 0 即 假 然后非一下 就变成了真
    printf("%d\t", r > s && ('s' || 'r')); // 先算括号内为真 再算大于号为真 最后全真为真 真!
    // 这里还涉及到书后面的运算顺序 要知道 或与的优先级是比较低的 而逻辑非的优先级比较高
    printf("%d\t", !(x > 3) && (r <= x)); // x 大于 3 成立 再非 假 逻辑与 一假为假 所以假
}
```

例 2.17

```
#include <stdio.h>
void main()
{
    int x, y, z, i, j; // 本节主要是学会复合赋值符号
    x = 3;
    y = 5;
```

```
    z = 7;
    i = j = 9; // 以上四种是直接赋值不参杂运算
    y += 3; // 等同于 y=y+3
    z %= x; // 等同于 z=z%x
    i *= x + 6; // 等同于 i=i*(x+6) 注意是有括号的
    x -= j / 4; // 等同于 x=x-(j/4)
    printf("%d\t%d\t%d\t%d\t", x, y, z, i);
}
```

例 2.18

```
#include <stdio.h>
void main()
{
    int x = 22, y = 93;
    // 0001 0110 这是 x=22 对应的二进制
    // 0101 1101 这是 y=93 对应的二进制
    printf("%d\t", x & y); // 按位与 按照每一位进行逻辑与运算 全真为真 结果为 0001 0100 对应十进制 20
    printf("%d\t", x | y); // 按位或 按照每一位进行逻辑或运算 一真为真 结果为 0101 1111 对应十进制 95
    printf("%d\t", x ^ y); // 按位异或 按照每一位进行同 0 异 1 运算 结果为 0100 1011 对应十进制 75
    printf("%d\t", ~x); // 按位取反 只能给一个数据使用 原来的 0 变成 1 1 变成 0 即可 结果为 1110 1001 首位 1 则负 0 则正
    // 如果判断出是负数 则这个负数的值是各位取反再加 1 即 0001 0111 结果为 23 加上负号 结果为-23
}
```

例 2.19

```
#include <stdio.h>
void main()
{
    int x = 45;
    // 0010 1101 这是 x=45 对应的二进制
    printf("%d\t", x);
    printf("%d\t", x >> 2); // 0010 1101 右移两位 就是舍去最后两位 变成 0000 1011 高位用 0 补齐
```

```
    printf("%d\t", x << 2); // 0010 1101 左移两位 就是舍去最前两位 变成 1011 0100 低位用 0 补齐
    // 一个数左移 1 位相当于该数乘以 2，左移两位相当于该数乘以 2 的 2 次，以此类推，前提是舍弃的高位中没有 1
}
```

例 2.20

```
#include <stdio.h> // 输入一个无符号整数，输出该数从右端开始的第 4-7 位组成的数
void main()
{
    unsigned int x, y, z;
    scanf("%d", &x);
    // 假如输入了 93 其二进制为 0101 1101
    // 目的是输出该数从右端开始的第 4-7 位组成的数，那么这几位数是从左往右开始的 1011 十进制结果为 11
    x >>= 3; // 将二进制 x 向右移位 3 个即可得到我们想要的从右端开始的 4-7 位数
    printf("%d", x);
    y = 15; // 15 的二进制数为 0000 1111 是专门构造的这个数
    z = x & y; // 让 1011 与 1111 进行按位与运算 结果为 1011
    // 可能用 93 来解释不太好说明构造这个 15 是用来干嘛的
    // 加入我们输入 999 这个数 二进制为 0011 1110 0111 我们需要的数是 1100 结果为 12
    // 按照程序流程 先右移 3 位 得到二进制为 0111 1100，此时这个数和我们需要的相比 还多了高位的四个数
    // 为了去掉高位的这四个数 同时还不影响后四位 就构造了一个 0000 1111 的二进制数
    // 通过进行 按位与运算 全 1 则 1 有 0 则 0 可得 0000 1100 结果为 12
    printf("result=%d", z);
}
```

例 2.21

```
#include <stdio.h>
void main()
{
    int x, y, min;
    printf("请输入两个整数: ");
    scanf("%d%d", &x, &y);
```

```
    min = (x < y) ? x : y; // 本节理解本条即可 先判断? 前的条件 成立的话 整个式子取 x 否则取 y
    // 注意 x 和 y 的数值类型如果不同 取两者中更高的那个 比如 int 和 double 取 double 输出时使用格式必须一致
    printf("两个数最小的是%d", min);
}
```

例 2.22

```
#include <stdio.h>
void main()
{
    int x, y, i, j, exp1, exp2;
    x = 4, y = 4, i = 8, j = 16;
    exp1 = (x + 6, x - 6, x / 6, x * 6); // 逗号表达式中从左到右以此计算 最后一个表达式为整个逗号表达式的值
    printf("%d\t%d\n",x,exp1); // 输出结果为 4  24
    exp2 = ((x = x + 2), y + i, x + j); // 第一个表达式计算后 x 被赋值 导致第三个表达式的 x 为 6
    printf("%d\t%d", x, exp2); // 输出结果为 6  22
}
```

例 2.23

```
#include <stdio.h> // 就是用这个语句来知道知道某个数据类型占几个字节数，因为每个编译系统结果可能不同
void main()
{
    printf("sizeof(char):%d\n", sizeof(char));
    printf("sizeof(short int):%d\n", sizeof(short int));
    printf("sizeof(int):%d\n", sizeof(int));
    printf("sizeof(unsigned int):%d\n", sizeof(unsigned int));
    printf("sizeof(long int):%d\n", sizeof(long int));
    printf("sizeof(float):%d\n", sizeof(float));
    printf("sizeof(double):%d\n", sizeof(double));
    printf("sizeof(long double):%d\n", sizeof(long double));
}
```

例 2.24

```c
#include <stdio.h>
void main()
{
    int a, b, c;
    printf("请输入整型变量 a、b 的值: ");
    scanf("%d %d", &a, &b);
    c = sizeof(a) + (a - 3) * b + a / 2 - 4; // 详细的优先级见表 2-9 慢慢就熟悉了
    printf("%d\n", c);
}
```

例 2.25

```c
#include <stdio.h>
void main()
{
    int s = 8, r = 3, t = 12;
    int res;
    double i = 4.5, j = 3.6, f;
    f = (t << 2) - s % r + (s += r *= t) + (!t && i + 2) - j;
    // 先括号 然后一些特殊的单目运算符 然后四则运算 位运算 关系运算 位运算 逻辑 三目 双目 逗号
    // 在 VC6 中结果为 86 但是在别的有些编译器中可能为 80
    res = t << 2;
    printf("%d\n", res);
    printf("%f", f);
}
```

第三章

例 3.5

```c
#include <stdio.h>
void main()
```

```
{
    char m, n;
    m = 'a';
    n = 'b';
    (m >= n) ? putchar(m) : putchar(n); // 这里用到了上一单元的条件表达式
    // 当 ? 前的表达式成立 执行 : 前的函数，否则执行 : 后的函数、
    // 我们要建立一个概念 函数括号里面的参数不是随随便便什么都可以填进去的
    // 根据 stdio.h 头文件的定义 int putchar(int _Ch); 说明该函数返回一个整数型 参数为整数
    // 这里我们参数给到的是一个字符型 众所周知 字符可与整数等价 所以程序运行正常
    putchar('\n'); // 可以用这个命令直接输出一个换行符
}
```

例 3.6

```
#include <stdio.h>
void main()
{
    char ch;
    ch = getchar(); // 获取键盘输入的字符赋值给变量 ch，注意是从标准缓冲区中读取输入的字符 只读取第一个
    putchar(ch); // 然后输出这个字符
    // 如果这个时候再 getchar 程序会直接结束，并没有停下来让你输入，因为刚才你输入的是一个字符加一个回车
    // 你所输入的每个字符都被记录在了一个叫做标准缓冲区的地方 所以第二个 getchar 就会 get 这个回车 然后后面没代码了就结束程序
    // 我们可以使用 fflush(stdin); 这条代码来清空标准缓冲区的所有内容，这样 getchar 就会等待我们输入内容了
}
```

例 3.7

```
#include <stdio.h> // 本节我们就来专门认识 printf 这个函数了
void main()
{
    int x = 97, y = 98;
    printf("%d %d\n", x, y); // 首先函数后面括号里面的全是这个函数的参数 用逗号分隔
    printf("%4d,%-4d\n", x, y); // 第一个参数是要输出的内容 所有橙色%号开头的 都叫格式控制符 这个符号将被后面的参数在输出时替代
```

```
    printf("%c,%c\n", x, y); // 后面有两个参数 分别对应第一个参数内的两个格式控制符
    printf("x=%d,y=%d", x, y); // 第一个参数双引号之内的绿色字符都会原样输出的
    // 课本说的很详细 大家主要看课本
}
```

我的补充

```
#include <stdio.h> // 熟悉几种基本的格式控制符 d 整数 f 小数 c 单个字符 s 字符串 % 输出百分号 u 十进制无符号整数
void main()
{
    printf("%5d\n", 789);          // 五位宽度 右对齐
    printf("%-5d\n", 789);         // 五位宽度 左对齐
    printf("%+5d\n", 789);         // 带正号的五位宽度 右对齐
    printf("%8.4f\n", 1.2312345);  // 八位宽度取四个小数 右对齐
    printf("%8.7f\n", 1.23123);    // 八位宽度取七位小数 右对齐
    printf("%7.5s\n", "chinese");  // 七位宽度取五位字符 右对齐
    printf("%-7.2s\n", "chinese"); // 七位宽度取两位字符 左对齐
    long n = 123456;
    printf("%ld\n", n);
    // printf("%hd\n", n); 这里注释掉是因为编译系统会报错: %hd 与 int 不符
}
```

```
#include <stdio.h> // 关于二进制的存储方式 https://www.cnblogs.com/guziteng1/p/11530072.html
void main()
{
    printf("%o\n", -1);
    // 二进制原码 10000000 00000000 00000000 00000001
    // 二进制反码 11111111 11111111 11111111 11111110
    // 二进制补码 11111111 11111111 11111111 11111111
    // 八进制计算 011 111 111 111 111 111 111 111 111 111 111
    // 得八进制为   3   7   7   7   7   7   7   7   7   7   7
```

```
    // 八进制形式输出不带符号 即将符号位也一起作为八进制的一部分输出 不会输出带负号的八进制整数
    printf("%x,%X\n", -1, -1);
    // 小写 x 就是输出小写的十六进制 大写 X 就是输出大写的十六进制 因为十六进制有字母 所以分了个大小写
    // 上面十六进制计算的话 1111 1111 1111 1111 1111 1111 1111 1111
    // 所以输出结果为 FFFFFFFF
    printf("%u\n", -1);
    // -1 以补码存储在计算机中 直接输出%u 的 -1 则将其补码转换为十进制无符号进行输出
    printf("%e\n", 1230.4567890);
    // 标准宽度 也就是默认宽度是 1.111111e+111 共十三位
    printf("%20.4e\n", 1230.4567890);
    // 此处的 m.n 指的是整个数据至少占 m 位，n 为小数点后的小数位数 数据不足则在最前以空格补齐
    float n = 123.456;
    printf("%f,%e,%g", n,n,n); // g 控制符就是从 f 和 e 里选一个输出宽度最小的输出 且不输出无意义的 0
}
```

例 3.8

```
#include <stdio.h>
void main()
{
    int a, b, c, sum;
    scanf("%d,%d,%d", &a, &b, &c); // 第一个参数同样是格式控制符 你获取了用户输入的一段文本 必须是按照这个格式的
    // 然后计算机把这串字符里面的三个%d 分别赋值给后面的三个参数 &就是取变量的内存地址 存到内存中去
    if (b > c) // 如果条件成立执行下面的 然后跳过 else 去 printf
        sum = a + b;
    else // 条件不成立就执行下面的所有
        sum = a + c;
    printf("%d", sum);
}
```

Scanf 的使用说明大家看书就好，然后自己试一下，比较容易理解。

例 3.10

```c
#include <math.h>
#include <stdio.h>
void main()
{
    float a, b, c, l, s;
    float t;
    scanf("%f,%f,%f", &a, &b, &c); // 按照格式获取用户输入的数据存到变量的内存地址中
    if (a != b)                    // 第四章会学到的 if 语句，即上底不等于下底时
    // 因为需要判断直角梯形的斜边长度 所以为了用勾股定理 必须让 a 是长底 b 是短底
    {
        if (a < b) // 如果 a 确实输入的时候是小于 b 的
        {
            t = a;
            a = b;
            b = t; // 引入一个临时变量 t 来完成 a、b 变量值的互换
        } //
        s = (a + b) * c / 2.0f; // 计算面积
        t = (float)sqrt((a - b) * (a - b) + c * c); // 计算斜边 这里用的是 a-b 为了不产生负数 所以上面代码做了处理
        l = a + b + t + c; // 周长加起来
        printf("%lf,%lf", l, s);
    }
}
```

一、你可能会问为什么是除 2.0f 而不是直接 2.0?

定义一个单精度浮点型变量时如果写成如下语句:

float a = 0.0;

因为 C 语言中默认的浮点型常量类型为 double 类型，所以这样写程序执行时会存在一个将双精度浮点型变量转化为单精度浮点型变量的过程，这个过程可以通过如下的方式去除掉。

float a = 0.0f;

虽然这个转化的过程可以被某些高端的编译软件过滤掉，但是第二种写法是更稳妥的方式。

二、%lf 是什么格式控制符?

%f 是单精度，%lf 是双精度。

例 3.11

```c
#include <math.h>
#include <stdio.h>
void main()
{
    float a, b, c, disc, x1, x2, p, q; // disc 为 △
    scanf("a=%f,b=%f,c=%f", &a, &b, &c);
    if (a == 0)
    {
        printf("参数 a 不能为零\n");
        return; // 返回 不继续执行
    }
    disc = b * b - 4 * a * c; // 很熟悉的判别式
    if (disc < 0)
    {
        printf("此一元二次方程无解");
        return;
    }
    p = -b / (2.0f * a); // 此处使用 p 和 q 分别计算求根公式
    q = (float)sqrt(disc) / (2.0f * a);
    x1 = p + q;
    x2 = p - q;
    printf("x1=%6.2f\nx2=%6.2f\n", x1, x2); // 以 6 位宽度 2 位小数输出结果
}
```

第四章

例 4.2

```c
#include <stdio.h>
```

```c
void main()
{
    char ch;
    printf("请输入一个小写字母: ");
    scanf("%c", &ch);
    if (ch > 'A' && ch < 'z') // 这里的判断条件是看输入的东西是不是字母
    // 判断的也不是很严谨其实 因为 65-122 中间还有一些符号不是字母 比如[
    // 翻翻书后的 ASCII 表就可以知道大写字母是 65-90 小写字母是 97-122
    {
        ch = ch - 32; // 大小写所对应的整数关系正好是差 32 所以这里用了减号
        printf("转换后的大写字母为:%c",ch);
    }
}
```

例 4.4

```c
#include <stdio.h>
void main() // 本程序的目的是在题目给出的条件下 把输入的两个整数合在一起
{
    int a, b, c, k;
    printf("请输入两个正数: ");
    scanf("%d,%d", &a, &b);
    if (a < 0|| b < 0 || a > 31 || b > 999) // 首先要判断输入的整数是否符合我们的条件
    // 括号内的条件如果满足就意味着不符合我们的题目要求
    {
        c = -1; // if 判断成立后进入此部分
        printf("输入的数据有误"); // 输出后就直接跳过 else 去到最后的 printf
    }
    else // if 括号条件不成立 说明符合题目要求 开始执行 else 的花括号内的代码
    {
        if (b < 10)
            k = 10;
```

```
        else if (b < 100)
            k = 100;
        else if (b < 1000) // 这里判断一下 b 是几位数 根据情况把 k 的值改变
            k = 1000; // 因为你要把两个数合起来 最高位你总得知道是哪一位对吧
        c = a * k + b; // 然后让 a 直接乘这个 k 再加上 b 就能实现我们的功能了
    }
    printf("\na=%2d,b=%3d,c=%5d", a, b, c);
}
```

例 4.5

```
#include <stdio.h>
void main()
{
    int a, b, c;
    a = 5, b = 3, c = 0;
    if (c) // 第一个 if 判断 c 的值 是 1 则执行下一个 if 是 0 则执行 else 后的代码
        if (a > b) // 课本上专门不缩进 让大家判断 else 到底和哪个 if 匹配
            printf("\n max=%d", a);
        else
            printf("\n max=%d", b);
    else
        printf("\n c=%d", c); // 一般现代化的代码工具 格式化代码后都会自动缩进的 很方便
}
```

例 4.8

```
#include <stdio.h>
void main()
{
    int n;
    printf("请输入月份: ");
    scanf("%d", &n);
```

```
    switch ((int)((n - 1) / 3)) // 运用数学表达式巧妙的对输入的月份进行分析 减少运算量 这里算出数字多少 就跳转到 case 几
    {                           // 上面的(int)是强制类型转换 舍弃掉小数
    case 0:                     // 1 2 3 月 减 1 除 3 整数部分只能是 0
        printf("%d 月份是春季! ", n);
        break; // break 用于跳出整个 switch 语句 继续执行后面的语句
    case 1:    // 4 5 6 月 减 1 除 3 整数部分肯定是 1
        printf("%d 月份是夏季! ", n);
        break;
    case 2:
        printf("%d 月份是秋季! ", n);
        break;
    case 3:
        printf("%d 月份是冬季! ", n);
        break;
    default: // 如果算出来的值都不是上面的情况 那么跳转到这里
        printf("你输入的有误");
        break;
    }
}
```

例 4.9

```
#include <stdio.h>
void main()
{
    int m, d;
    printf("请输入平年的月份: ");
    scanf("%d", &m);
    switch (m)
    {
    case 1:
    case 3:
```

```
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        d = 31; // 这个例子主要目的是让各位明白多个 case 可以用一种情况 前面的空白就行
        break;
    case 4:
    case 6:
    case 9:
    case 11:
    case 2:
        d = 28;
        break;

    default:
        d = -1;
        break;
    }
    if (d == -1)
        printf("输入月份有误");
    else
        printf("平年%d 月份有%d 天", m, d);
}
```

例 4.11

```
#include <math.h>
#include <stdio.h>
void main()
{
    float a, b, c, pbs, x1, x2, p, q;
```

```
    printf("请依次输入二次方程的系数: \n");
    scanf("%f,%f,%f", &a, &b, &c);
    pbs = b * b - 4 * a * c; // 判别式应该很熟悉吧
    if (pbs > 0) // 然后开始了三种大于小于等于 0 的判断
    {
        x1 = (-b + sqrt(pbs)) / (2 * a);
        x2 = (-b - sqrt(pbs)) / (2 * a);
        printf("两个不相等的实数根为: x1=%5.4f,x2=%5.4f\n", x1, x2);
    }
    else if (pbs == 0)
    {
        x1 = -b / (2 * a);
        printf("两个相等的实数根为: x1=x2=%5.4f\n", x1);
    }
    else if (pbs < 0)
    {
        p = -b / (2 * a);
        q = sqrt(-pbs) / (2 * a);
        printf("两个不相等的虚根为: x1=%5.4f+%5.4fi,x2=%5.4f-%5.4fi\n", p, q, p, q);
        // 你可能只看书看不懂这个%5.4fi 的 i 是个啥 其实这是虚根的虚数标志 还记得高中数学选择题第二题的虚数吗? 是不是有个 i?
        // 你看代码显示是绿色的, 妥妥的原样输出。
    }
}
```

4.12 简单

4.13 涉及第五章这个就略掉了

编程实践

```
#include <stdio.h>
void main()
{
```

```
    double y;
    scanf("%lf", &y); // 获取用户输入的工资 双精度实数型
    double x, c; // 定义变量 x 存储税款 c 变量存储个人所得税起征点
    c = 3500; // 起征点为 3500
    if (y <= c) // 小于 3500 不征税
        x = 0;
    else if (y < c + 1500) // 执行到这一步说明肯定大于 3500 如果小于 3500+1500 则继续下面的语句
        x = (y - c) * 0.03;
    else if (y <= c + 4500) // 类同
        x = 1500 * 0.03 + (y - c - 1500) * 0.10;
    else if (y <= c + 9000)
        x = 1500 * 0.03 + 3000 * 0.10 + (y - c - 4500) * 0.20;
    else if (y <= c + 35000)
        x = 1500 * 0.03 + 3000 * 0.10 + 4500 * 0.20 + (y - c - 9000) * 0.25;
    else if (y <= c + 55000)
        x = 1500 * 0.03 + 3000 * 0.10 + 4500 * 0.20 + 9000 * 0.25 + (y - c - 35000) * 0.30;
    else if (y <= c + 80000)
        x = 1500 * 0.03 + 3000 * 0.10 + 4500 * 0.20 + 9000 * 0.25 + 26000 * 0.30 + (y - c - 55000) * 0.35;
    else
        x = 1500 * 0.03 + 3000 * 0.10 + 4500 * 0.20 + 9000 * 0.25 + 26000 * 0.30 + 80000 * 0.35 + (y - c - 80000) * 0.45;
    // 以上的所有 if 和 else 都是为了判断工资位于哪个区间
    // 知道区间以后就可以按照超额累进税率进行计算了 可以画数轴帮助自己理解这个税是怎么征收的
    printf("您的工资为: %9.2f 应缴纳个人所得税为: %9.2f", y, x);
    // 现在我国个人所得税已经改革了，请你自己编写一个适用于改革后的个人所得税计算工具!
}
```

第五章

例 5.1

```
#include<stdio.h>
```

```c
int main()
{
    int i = 1;
    int sum = 0;
    while (i<=100) // 本节主要学习 while 这个循环命令
    // 只要括号里面的式子一直是满足的，下面这个花括号里的所有代码就会一直循环执行 每次循环结束都会判断 成立继续 不成立跳出循环
    {
        sum += i; // 等同于 sum=sum+i 根据条件 i<=100 因为 i 最初赋值了 1 那么就是 1~100 循环 100 次 则会一直从 1 加到 100
                  // 如果我最开始让 i=0 条件是 i<100 也是循环 100 次，只不过是从 0+1+2+…+99 要注意这一点 以后会常分析
        i++; // 每一次循环加完之后 i 都会自增 要不然怎么能 1+2+3....呢?
             // 在编程循环语句中，总必须有一个变量来记录循环次数，因为我们会用到这个次数来实现一些操作 所以这个 i 是很重要的 也总是以 i++ 自增的形式出现
    }
    printf("sum=%d\n", sum);
}
```

例 5.2

```c
#include <stdio.h>
int main()
{
    float x, sum = 0;
    float s;
    int i = 1;
    printf("请输入数据: \n");
    while (i <= 20) // i 初值是 1 条件是 i<=20 则整个过程循环完毕后 i 会等于 21
    {
        scanf("%f", &x); // 每次循环用户需要输入一个数
        sum += x; // 把用户输入的这个数加入到 sum 中
        i++; // 让 i 自增
    }
    s = sum / (i - 1); // 这里不用 i-1 直接写个 20 也行，但是你必须明白 i-1 的意义是什么
    // 日后写程序的时候很多时候你是不知道用户输入了多少个数字这种情况的 就必须使用循环记录变量 i 来操作
```

```
    printf("平均值为: %f\n", s);
}
```

例 5.3 看了 5.4 之后自己写写

例 5.4

```
#include <stdio.h>
int main()
{
    float x, sum = 0;
    float s; // 变量定义和用 while 的时候是一样的 没什么要说的
    int i = 0; // 哎 你要注意了 这里的 i 可不是 1 了，初始化为了 0
    printf("请输入数据: \n");
    do // do-while 语句 先执行花括号的代码一遍 然后判断 成立的话继续循环 不成立的话跳出
    {
        scanf("%f", &x); // 获取用户输入的数据
        sum += x; // 把用户输入的数据添加到 sum 中
        i++; // i 自增
    } while (i < 20); // 哎 你必须要注意了 这里的判别式不是<=而是< 请你自己分析一下不同符号的循环次数
    s = sum / i; // 这里的 i 直接使用，因为他就是 20，不用减一 为什么呢 请你分析一下循环次数和 i 条件式的关系
    printf("%f\n", s);
}
```

例 5.5 看了 5.6 之后自己写写

例 5.6

```
#include <stdio.h>
int main()
{
    float x, sum = 0;
    float s;
```

```
    int i;                    // 哎 你要注意了 这里的 i 没给初始值
    for (i = 0; i < 20; i++) // for 循环先执行第一个参数 即赋值 整个循环只执行一次，然后判断条件，开始循环，每循环一次执行第三个参数 执行完再次判断条件…
    {
        scanf("%f", &x);
        sum += x;
    }
    s = sum / i; // 这里的 i=0 和 i<20 配套 循环 20 次
    printf("%f", s); // 书上写的是总和为 很明显错了 这是求平均数 这本书小细节错误不少
}
```

例 5.7

```
#include <stdio.h>
int main()
{
    int i, x, n, s;
    printf("请输入 n 的值: \n");
    scanf("%d", &n);
    for (i = 1, s = 0; i < = n; i++, s += x)
    { // 主要是让大家明白 for 循环的执行过程是怎么样的
      // 先是第一个分号前的语句，然后判断两个分号之间的条件 成立则开始循环 完成循环后执行最后一个分号后的语句 再次判断循环
      // 可以通过逗号添加多条语句
        x = 2 * i;
    }
    printf("s=%d\n", s);
}
```

5.2.3

```
#include<stdio.h>
```

```c
int main()
{
    int n, i, j;
    while (scanf("%d",&n)==1) // 这个 1 不是输入的 n 等于 1，而是 scanf 这个函数的返回值是 1，即获取到了输入
    {
        for ( i = 0; i < n; i++){ //最外层循环 n 次 条件为 i<n 且 i 从 0 开始 即 n 次
            for (j = 0; j < n-i; j++) //每外层循环时 循环键入空格 键入数量为 n-i 次（通过画图分析得到）
                putchar(' '); // 键入一个空格
            for (j = 0; j < i * 2 + 1;j++) // 最外层循环时 空格键入完毕后 开始循环键入* 数量为 i*2+1
            // 表达式 i*2+1 可由数学分析得到 假设 n=3 则 i 分别=0 1 2 时 *数量分别是 1 3 5
                putchar('*'); // 键入一个*号
            putchar('\n'); // 一行操作完成后换行 跳转到 for 循环头开始下一次循环
        }

    }
    return 0;
}
```

例 5.8

```c
#include <stdio.h>
int main()
{
    int i = 1, sum = 0;
    suibian:if(i<=100) // 本节主要学习 goto 语句 goto 后面加要跳转到的名字即可 这个名字可以自定义 不用非得是 loop
    {
        sum = sum + i;
        i++;
        goto suibian; // 要注意前后两个名字必须是一样的 不然跳到哪里去?
    }
    printf("%d\n", sum);
```

```
}
```

Break 语句很简单，就是跳出一层循环，各位可以自己写写代码感受一下。

例 5.9

```
#include <stdio.h>
int main()
{
    int x, y;
    for (x = 1, y = 1; x <= 100; x++)
    {
        if (y >= 20)
            break; // 跳出循环
        if (y % 3 == 1)
        {
            y += 3;
            continue; // 提前结束本次循环
            y -= 5;
        }
    }
    printf("x=%d\n", x);
}
```

例 5.10

```
#include <stdio.h>
int main()
{
    int i = 1, j;
    while (i <= 9) // 需要输出九行内容
    {
```

```
        for (j = 1; j <= i; j++) // 循环输出每行的乘法 根据乘法表规则 第一行需要一个乘法 第二行两个 以此类推
        {                        // 所以这里的判断条件就是 j<=i 过程中会是 1<=1 就循环 1 次 1<=2 就循环 2 次
                                 // for 在 while 的每次循环中都会重新执行所有语句的
            printf("%d×%d=%-2d ", j, i, j * i);
        }
        printf("\n");
        i++;
    }
}
```

例 5.11

```
#include <stdio.h>
int main()
{
    char c;    // 定义一个字符变量用于判断获取到的字符是什么
    int i = 0; // 初始化循环记录变量
    printf("请输入字符串以回车确认: \n");
    c = getchar();
    // 先介绍 getchar，用户会输入一段字符然后回车，这段字符会被存储在键盘的缓冲区中
    // 而 getchar 就会从缓冲区中 以从左到右的顺序一个一个的提取字符，提取过的字符就没了
    // 比如我输入了一段字符是 nicaicaiwoshuruleshenme\n 最后那个\n 是回车算一个字符
    // 这段字符就会存在缓冲区里面 赋值给 c 的第一个字符就是最左边的 n
    while (c != '\n') // 这里循环开始先判断 c 是不是回车 如果不是 就让 i 自增记录加 1 如果是回车那么不满足条件跳出循环
    {
        i++;
        c = getchar(); // 自增之后再次获取缓冲区里面的字符 比如循环第一次这里 c 会被赋值为 i 第二次循环会被赋值为 c
    }
    printf("你所输入的字符共有%d 个", i); // 经过循环语句的循环判断和自增 i 就会是我们最终的数据 输出即可
}
```

5.12 5.13 学会了上面的知识就可以自己编写了 不看书试试写一个出来?

例 5.14

```c
#include <math.h>
#include <stdio.h>
int main() // 关键是要分析程序如何实现利用线索求解
{
    int dNum;               // 三四位和五六位是相同的，就是 31xxyy 并且 xxyy 开根号是一个整数
    int num, i, j;          // 简单分析一下可能的情况很少 就从 1100 1111 1122 1133 一直试下去就好了
    for (i = 1; i <= 9; i++) // 第一个循环 让 xxyy 中的 xx 一直从 11 到 99
    {
        for (j = 0; j <= 9; j++) // 第二个循环 让 xxyy 中 xx 和 yy 组合起来
        {
            dNum = (i * 10 + i) * 100 + j * 10 + j; // i*10+i 是为了形成 11 这种形式 再*100 是为了加上后面的 yy 组成 xxyy
            num = (int)sqrt((float)dNum);           // 对 dnum 进行开方
            if (num * num == dNum)                  // 验证一下 如果开方出来的再平方和 dnum 一样 那就是这个数了
                printf("31%d\n", dNum);
        }
    }
}
```

例 5.15

```c
#include <stdio.h>
int main() // 本程序思路书上给的很清晰，首先确定各个量的取值范围
{
    int cocks, hens, chicks;
    for (cocks = 0; cocks <= 20; cocks++) // cocks 的取值是 0 到 20，循环 20 次，先确认 cocks 是多少
    {
        for (hens = 0; hens <= 100 - 5 * cocks; hens++) // 确认 cocks 后再确认 hens，根据 cocks 的取值动态改变 hens 的取值
```

```
        {
            chicks = 100 - hens - cocks; // 两个值都取完了 算出剩下的 chick 占多少
            if (5 * cocks + 3 * hens + chicks / 3.0 == 100) // 三个值都搞定判断是否满足题目要求的表达式 满足就输出 不满足就继续循环
                printf("cocks=%d,hens=%d,chicks=%d\n", cocks, hens, chicks);
        }
    }
} // 编程中处理多元方程组经常采用这种方法来确定
```

编程实践

```
#include <math.h> // 哥德巴赫猜想：任何大于 2 的偶数都是两个素数之和 比如 22=5+17 其中 5 和 17 为素数 除了 1 和其本身外没有其他因数
#include <stdio.h> // 感谢软件开发项目组李沛熹、ACM 算法工作室周波延的指点
int main()
{
    int c, d, i, j, k, t, x; // c 上限 d 下限 i 用于循环记录 j 记录第一个素数 k 记录第二个素数 t 用于辅助判断 x 用于判断是否为素数
    printf("请输入区间上下限: ");
    scanf("%d,%d", &c, &d);
    printf("在区间[%d,%d]中验证哥德巴赫猜想\n", c, d);
    if (c % 2)                  // 猜想的前提是任何大于 2 的偶数 这里对 c 上限进行修正 如果奇数/2 取余记结果会是非 0 因为后面循环是以加 2 为单位的 必须全是偶数
        c++;                    // 判断输入了个奇数，自增变成偶数
    for (i = c; i <= d; i += 2) // 让 i=上限 循环条件为小于等于下限 每次循环递增 2 算的都是偶数嘛 所以递增 2
    {                           // 嗯从上限的第一个偶数开始循环
        j = 1;                  // 初始化第一个素数为 1 所有数都成立的是 1*数本身=数本身 我们要找的是别的素数
        while (j < i / 2)       // 开始循环寻找两个素数 循环条件为该素数小于这个数的一半 因为猜想为两个素数相加 必有一大一小 如果不做限制则会出现重复查找 比如寻找 22 的素数 先找 5 又找 17 就浪费资源
        {
            j = j + 2;                      // 将会取到奇数 3 5 7 9 11 13 等等 里面有素数 也有不是素数的 一会会做一个判断它是不是素数
            k = i - j;                      // 用要计算的数减去取的第一个素数 得到 k 为第二个数
            t = 0;                          // 预先设 t 为 0 这个 t 可以把它当作是 0 的时候是素数 1 的时候不是素数（看程序怎么写了 本例是这样的）
            for (x = 3; x <= sqrt(k); x += 2) // 一般来说找一个数的因数是从 1 开始递增 然后除它看余数是不是 0 但是这种算法很浪费资源
```

```
            {                                   // 比如我们找 9 的因数 实际上只需要从 1 找到 3 也就是他的算术平方根就可以找全了 再比如 100 只需要从 1 找到 10 而 1 不需要找 所以程序直接从 3 开始 且每次增 2
                if ((j * k) % x == 0)           // 我们需要检查我们找到的两个数是不是都是素数 怎样就能一块知道两个数了呢 那就是判断这两个数相乘能否被 k 的因数整除 因为如果其中一个数可被某因数整除 jk 的乘积也必定会
                // 但是两个素数的乘积必定是合数 本例的这种检测算法虽然可以找到一组，但理论上也可能把唯一的一组素数给 pass 掉 虽然实际上并不会 所以还仍待优化
                {
                    t = 1; // 那么责令 t 状态为 1
                    break; // 跳出 for 循环开始找下一个 j
                }
            }
            if (t == 0) // 经上述 for 循环不符合素数条件的不输出 符合的输出
            {
                printf("%d=%d+%d\n", i, j, k); // 符合条件的进行输出
                break;                          // 跳出 while 循环 开始下一个偶数的旅程
            }
        }
    }
    if (t == 1) // 如果程序的区间只执行一次循环的话这个判断还有点用 如果执行多个循环这个反例如果存在则会被后面 t=0 时吞没掉
                // 所以课本不是上帝 课本也会出错的 这个 if 最好放在 for 循环里 然后加一个循环判断标识符 全部循环完了一个也没有的话 再及时输出无法分解的情况
        printf("找到反例无法分解");
    else
        printf("哥德巴赫猜想在区间[%d,%d]中正确\n", c, d);
}
```

第六章

例 6.1

```
#include <stdio.h> // 有一个整型数组 计算其中的正数和及正数平均数
void main()
{
```

```
    int a[10] = {15, -20, 30, 70, -60, 88, 90, 17, -10, 46}; // 定义了一个变量 a 存在数组 元素个数为 10 花括号内分别为各个元素的值
    int sum, aver, num, i; // 定义一些辅助变量
    num = 0;
    sum = 0; // 初始化变量值
    for (i = 0; i < 10; i++) // 循环十次 因为数组内有十个元素
        if (a[i] > 0) // 每次循环都会判断 数组的这个元素是否大于 0 即是否为正数
        {
            num++; // 是正数 令 num 自增 这个 num 是记录正数个数的 用来最后算正数的平均数
            sum += a[i]; // sum 则加上这个数 所以 sum 是记录正数之和的
        }
    if (num != 0) // 循环十次后来到这里 只要正数个数不为零 就算 aver 即平均数
        aver = sum / num; // 用正数之和除以正数个数
    else
        aver = 0; // 否则正数为 0 个 平均数直接等于 0
    printf("sum=%d,average=%d\n", sum, aver); // 最后输出我们得知的 sum 和 num 整个程序逻辑清晰 容易理解
}
```

例 6.2

```
#include <stdio.h> // 求斐波那契 Fibonacci 数列前 20 项
void main()        // 前两项分别是 1 和 1 后面的项是前面项的和 使用数组可以很好的处理这个情况
{
    int i;                  // 循环记录辅助变量
    int f[20] = {1, 1};     // 定义整形数组变量 f 用来存放前 20 项的值
    for (i = 2; i < 20; i++) // 这个循环用于把第 3-20 项的值求出来给到数组里
        f[i] = f[i - 1] + f[i - 2];
    for (i = 0; i < 20; i++) // 这个循环用于输出所有的项
    {
        if (i % 5 == 0) // 这里是为了让输出的时候换个行 别太长
            printf("\n");
        printf("%12d", f[i]);
    }
```

```
}
```

例 6.3

```
#include <stdio.h>
void main()
{
    int i, j, m, a[20];
    for (i = 0; i < 10; i++) // 循环输入十个数存储到数组中去
        scanf("%d", &a[i]);
    for (i = 0; i < 9; i++) // 循环九次
        for (j = i + 1; j < 10; j++) // 循环九次
            if (a[j] < a[i]) // 每次循环的时候 j 总比 i 大 1 所以大循环 a[i]会依次跟小循环的 a[j]进行对比 一旦后面的比它小 那么
            {
                m = a[j]; // 先把后面的数临时保存给 m
                a[j] = a[i]; // 让后面的数等于前面更大的数
                a[i] = m; // 再把刚才临时保存在 m 的大数赋值给前面的数
                // 这样就完成了两个数字之间的交换 为什么不直接交换呢 各位思考一下
            }
    for (i = 0; i < 10; i++) // 最后输出一下
        printf("%5d", a[i]);
} // 输入: 1 2 3 6 8 23 5 6 8 9
// 输出: 1    2    3    5    6    6    8    8    9    23
```

例 6.4

```
#include <stdio.h> // 本程序是让 n 个整数 使其前面各数顺序向后移动 m 个位置 最后 m 个变成前面的 m 个数 例如 12345 m=2 变成 34512
void main()
{
    int number[20], n, m, i; // 设定数组元素个数为 20 并定义几个变量一会使用
    printf("the total numbers is:");
    scanf("%d", &n); // 提示输入所有的数字个数
    if (n > 20)      // 如果导致数组越界则返回
```

```
        return;
    printf("back m:");
    scanf("%d", &m); // 提示输入 m 的值
    if (m > n)       // 如果 m 直接超出 n 的个数 返回
        return;
    for (i = 0; i < n; i++)      // 开始循环接收用户输入的每一个数字
        scanf("%d", &number[i]); // 存储到数组之内
    int p, array_end;            // 定义两个变量
    for (; m > 0; m--)           // 循环 m 次 因为我们需要移动 m 次数字 注意这里所使用的 m-- 完全是根据 m 的实际情况来使用的
    {
        array_end = number[n - 1];     //  数组的最后一个数的下标并不是数组的总元素 因为数组第一个元素的下标是 0 所以这里先把最后一个数给到临时变量
        for (p = n - 1; p > 0; p--)    // 紧接着开始循环 最后一个数已经被存储在临时变量 可以对其进行覆盖数值的操作了 需要循环 n-1 次
            number[p] = number[p - 1]; // 每次循环都将前面的数 赋值给 后面的数 完成一次一次的位数移动
        number[0] = array_end;         // 最后内层 for 循环完毕后第一位空出 再把临时变量赋给第一位
        // 这样一次完整的循环 就成功的把所有位数向右移动了一次
        // 这个过程需要各位用实际的例子去理解，自己列几个数字 然后把自己当成计算机 去执行一下就明白了 脑中会有相应的模型的
    }
    for (i = 0; i < n; i++) // 最后输出调整完毕的数字
        printf("%d ", number[i]);
}
```

例 6.5

```
#include <stdio.h> // 本例是为了让我们了解多维数组的使用 实现两个多维数组的数据交换 具体的功能是转置 也就是行变成列 列变成行
#define M 2 // 之所以要定义两个常量 是因为一会转制的过程中这个 M 和 N 会被频繁用到 不用常量直接输入整数也是可以的
#define N 3
void main()
{
    int a[M][N] = {{1, 2, 3}, {4, 5, 6}}, b[N][M]; // 首先定义一个二维数组 a 并赋值是个两行三列的数据 并再定义一个三行两列的数组
    int i, j; // 定义循环记录变量
    printf("array a:\n"); // 告诉用户以下输出数组 a 的数据
    for (i = 0; i < M; i++) // 输出数组的数据总是需要循环语句来参与的 不然那么多位置一次怎么能输出的完呢
```

```
    {
        for (j = 0; j < N; j++) // 二维数组 所以还需要内嵌一个循环语句
        {
            printf("%5d", a[i][j]); // 以 5 个数字长度默认右对齐的方式输出数组数据
            b[j][i] = a[i][j]; // 注意 关键功能实现在这里 把 a 的[i][j]赋值给 b 的[j][i] 比如 a[1][3]给到 b[3][1] 完美符合我们的转置要求
        }
        printf("\n"); // 一个元素循环完毕后换个行 然后开始循环下一元素
    }
    printf("array b:\n"); // a 数组循环完毕后换行 为了好看 精干 一些
    for (i = 0; i < N; i++) // 以下操作就是循环输出 b 数据的内容 和上面完全一模一样就不解释了
    {
        for (j = 0; j < M; j++)
        {
            printf("%5d", b[i][j]);
        }
        printf("\n");
    }
}
```

例 6.6

```
#include <stdio.h> // 输出杨辉三角形 算法的本质是
// 每行首列元素以及对角线元素的值为 1 其余元素的值等于上一行同列元素与上一行前一列元素的和；每行元素递增 1
#define n 6 // 常量定义了一个行数
void main()
{
    int a[n][n]; // 定义了 6*6 的二维数组
    int i, j; // 依然是循环记录变量
    for (i = 0; i < n; i++) // 开始循环干嘛呢不知道一会往下看 循环 n 次
    {
        a[i][0] = 1; // 让每个元素的第一个元素为 1
        a[i][i] = 1; // 实现每行最后一个元素为 1
```

```
        // 数组是从 0 开始的 循环也是从 0 开始的 数据不会出错
    }
    for (i = 2; i < n; i++) // 循环 4 次 前两行不需要赋值了
        for (j = 1; j < i; j++) // 需要的是给空缺的地方赋值 这里的循环条件就很巧妙了 正好和缺的地方数量一样
        // j 一定是从 1 开始的 1 代表一行内的第二个数 第三行的第二个数是肯定空缺的 循环从这里开始
        // 知道一个循环从哪里开始之后就必须根据已有的变量 来找寻终止循环的条件了 这里选择了 j<i
        // 寻找这个条件的时候 其实大概率第一次循环满足了 后面的循环也就满足了 没有那么难寻找的
            a[i][j] = a[i - 1][j - 1] + a[i - 1][j]; // 根据算法本质 让上一行上一列的数加上上一行本列的数 赋值给要给的
    for (i = 0; i < n; i++) // 最后循环输出数组 展示结果 ok
    {
        for (j = 0; j <= i; j++)
            printf("%5d", a[i][j]);
        printf("\n");
    }
}
```

例 6.7

```
#include <stdio.h> // 求出二维数组中的最大行和
// 数组中 行下标相同的元素之和称为行和，各行中最大值为最大行和。
void main()
{
    int a[][4] = {23, 14, 563, 657, 54, 95, -98, 0, 99, 108, 777, 10}; // 随便的给了一些数据 赋值给数组
    int b[3], i, j, rowmax;                                            // 定义了一个只有 3 个元素的数组 几个变量
    int sum = 0;                                                       // sum 估计是最后求和用的
    for (i = 0; i < 3; i++)                                            // 开始循环了 三次 很明显
    {
        rowmax = a[i][0];          // 假如这行里 最大值是第一个元素
        for (j = 1; j < 4; j++)    // 循环三次 从 1 开始是因为要把第二个元素和第一个元素进行对比
            if (a[i][j] > rowmax) // 如果这个元素大于 rowmax 了
                rowmax = a[i][j]; // 那就把大数换进来
        b[i] = rowmax;             // 最后最大的数赋值给 b 的元素
```

```
    }                           // 循环三次以后 b 变量三个元素就满了 分别对应 a 数组每行里面最大的数
    printf("二维数组 a:\n");
    for (i = 0; i < 3; i++) // 又开始循环了 也是三次
    {
        for (j = 0; j < 4; j++) // 就是输出一下数组内容 不说了
            printf("%5d", a[i][j]);
        printf("\n");
    }
    printf("一维数组 b:\n"); // 嗯 也不说了
    for (i = 0; i < 3; i++)
    {
        printf("%5d", b[i]);
        sum += sum + b[i]; // 循环输出 b 的时候呢 多了这一步 sum=sum+sum+b[i]
    }
    printf("\nthe sum is:%5d", sum);
    printf("\n");
}
```

例 6.8

```
#include <stdio.h>
// 本题目标是把一串字符串倒序输出
void main()
{
    char ch, str[] = "The c programming language!"; // 定义一个字符数组并给出数据
    int i, n; // 定义必要的变量
    n = sizeof(str) - 1; // 减去 1 的原因是因为 sizeof 获取的字符长度包含了\0
    for (i = 0; i < n / 2; i++)
    {
        ch = str[i]; // ch 是中间变量
        str[i] = str[n - i - 1]; // 这里的 for 循环书上有解释原因
        str[n - i - 1] = ch;
```

```
    }
    printf("%s\n", str);
}
```

例 6.9

```
#include <stdio.h>
#include <string.h>
// 编写函数实现 strcpy 的功能，strcpy 是把一个数组的字符给到另一个数组中去
void main()
{
    char s1[20], s2[20]; // 先定义两个字符数组
    int i;
    printf("Input s2:\n");
    gets(s2); // 获取用户手动输入的字符串
    for (i = 0; s2[i] != '\0'; i++)
    {
        s1[i] = s2[i]; // 通过 for 循环挨个赋值
    }
    s1[i] = '\0'; // 最后补充上一个\0 结束符号
    printf("Output s1:\n");
    puts(s1);
}
```

例 6.10

```
#include <stdio.h>
#include <string.h>
// 寻找三个字符串中最大的那个 也就是最长的那个 程序设计思路课本已经写的很详细了
void main()
{
    char str[3][18], maxstring[18]; // 创建二维数组 一行一组数据 一维数组用来存放最大的
    int i;
```

```
    for (i = 0; i < 3; i++) // 循环输入三组字符串
        gets(str[i]);
    if (strcmp(str[0], str[1]) > 0) // 使用 strcmp 函数判断
        strcpy(maxstring, str[0]);  // 如果成立则说明第一行的数据更长 赋值给一维数组专门用来存储他
    else
        strcpy(maxstring, str[1]);
    if (strcmp(str[2], maxstring) > 0) // 同理
        strcpy(maxstring, str[2]);
    printf("\n the max string is:%s\n", maxstring); // 最后输出最大的即可
}
```

第七章

例 7.1

```
#include <stdio.h>
void main() // 这是我们的 main 主函数 你会发现和下面的 printstar 和 print_message 函数同级别
// 最主要的区别就在于我们的程序在执行的时候会优先找到 main 函数执行 这是第一章节的知识点了
{
    void printstart();
    void print_message(); // 这是两条函数声明 告诉系统这两个函数是存在的
    // 之所以要优先声明 是因为使用它的时候系统会不认识他导致报错 可以理解为 提前报备
    printstar(); // 这三行就直接对用户自定义函数进行了使用
    print_message();
    printstar();
}
void printstar() // 那么我们调用函数后究竟下一步要执行什么事情 就在这个函数体里面写明
{
    printf("************\n");
}
void print_message() // 同样的 写明函数体里面具体的代码即可
{
```

```
    printf("How do you do!\n");
}
// 函数就像是一个又一个的工具，被 main 主函数所调用、所使用
// 上面的两个函数体叫做对函数的定义，我们可以把它直接放在 include 那行之后，这样的话就不用声明了。
```

例 7.2

```
#include <stdio.h>
float circumference(float x) // 与上一例正好相反，本题在 main 函数前定义了这个函数
// 注意以下定义一个新函数的格式 类型 函数名（参数表列）
{
    float y;
    y = 2 * 3.14 * x; // 花括号内的都是这个函数的具体代码 表明这个函数做了什么
    return (y);
}
void main() // 又来到了我们熟悉的主函数
{
    float r, s;
    printf("请输入半径值: ");
    scanf("%f", &r);
    s = circumference(r); // 在这里进行函数的调用，r 是传递过去的参数
    printf("周长是%f\n", s);
}
```

例 7.3

```
#include <stdio.h>
long jc(int k) // 刚开始就定义了一个长整型的函数，名字是 jc（阶乘）好记嘛，有一个参数 k，是整数型的
{
    long p;
    int i;
    p = 1;
    for (i = 1; i <= k; i++) // 在这个函数内 会进行 k 次循环
```

```
        p = p * i;
    return p; // 循环完毕后就完成了从 1 到 k 的连续乘法
}
void main()
{
    long jc_sum = 0;
    int i;
    for (i = 1; i <= 8; i++) // 要计算阶乘的和 当然就需要循环把每一个阶乘的结果相加起来
        jc_sum += jc(i); // 这里对 Jc 函数进行了调用
    printf("%ld", jc_sum);
}
```

例 7.4

```
#include <stdio.h>
long cub(int x) // 很简单的程序 先定义了一个用于计算某数立方的函数 定义方法就不说了
{
    long y;
    y = x * x * x; // 这是最主要的功能
    return y; // 然后通过 return 把 y 的值返回到主调函数内
}
void main()
{
    int num;
    long cub_num;
    printf("请输入一个整数: \n");
    scanf("%d", &num);
    cub_num = cub(num); // 变量接受到了函数的返回值，存储到了变量内
    printf("%d 的立方值是%ld", num, cub_num); // 进行输出即可
}
```

例 7.5

```c
#include <stdio.h>          // 主要是帮助大家理解以下函数返回值的概念
void swap(float x, float y) // 定义了一个 swap 函数，有两个参数，返回值是 void 也就是没有返回值
{
    float temp;
    temp = x;
    x = y;    // 在这个函数中，作用是把参数里面的 x 和 y 的值进行互换
    y = temp; // 但是大家要注意了，参数被传输过来以后 这两个参数就相当于是这个函数的内部变量了
    // 任何对参数变量内容的改变都仅限于这个函数之中
    printf("x=%.2f y=%.2f\n", x, y);
}
void main()
{
    float x = 9.3, y = 4.6;
    swap(x, y); // 调用了 swap，传出了参数，然后输出了 x 和 y
    // 然而当我们再次输出 x 和 y 的时候值依然没有成功交换
    // 这是因为这俩个 x、y 其实是不一样的 一组是主函数里的 一组是 swap 函数的参数变量
    // 从而说明函数里面的参数变量的改变并不会影响其他函数内的变量
    // 如果你想通过 swap 函数把 main 里面的 x 和 y 进行交换，那这种数值传递方式是不可行的 应该使用地址传参 第八章学习
    printf("x=%.2f y=%.2f\n", x, y);
}
```

例 7.6

```c
#include <stdio.h>
void nzp(int a[5]) // 无返回值的 nzp 函数带有一个参数 为整型变量 其是个数为 5 的数组 a
{
    int i;
    printf("\nvalues of array a are:\n");
    for (i = 0; i < 5; i++) // 这个循环的目的是为了把负数元素全部改成 0
    {
        if (a[i] < 0)
            a[i] = 0;
```

```
        printf("%d ", a[i]);
        // 其实就是想告诉你数组传参 是唯一的 不论在哪个函数修改都会导致这个数组内容的改变
    }
}
void main()
{
    int b[5], i;
    printf("\ninput 5 numbers:\n");
    for (i = 0; i < 5; i++) // 先让你输入五个数字
        scanf("%d", &b[i]);
    printf("initial values of array b are:\n");
    for (i = 0; i < 5; i++) // 先输出你的输入的五个数字
        printf("%d ", b[i]);
    nzp(b); // 调用函数取代负数元素
    printf("\nlast values of array b are:\n");
    for (i = 0; i < 5; i++) // 在输出依次让你看看结果 哎嘿 发现的确发生了变化
        printf("%d ", b[i]);
}
```

例 7.7

```
#include <stdio.h>
void mergestr(char s1[], char s2[], char s3[]) // 一个 mergestr 函数拥有三个数组参数
{
    int i, j;
    for (i = 0; s1[i] != '\0'; i++) // 先把第一个数组中的所有字符转移到参数 3 之中
        s3[i] = s1[i];
    for (j = 0; s2[j] != '\0'; j++) // 再把第二个数组中的所有字符转移到参数 3 之中
        s3[i + j] = s2[j]; // 下标要注意是在 i 的基础上 连接到后面的元素去 而不是覆盖
    s3[i + j] = '\0'; // 自己手动把最后一个字符改成结束字符
}
void main()
```

```
{
    char str1[] = {"Hello "};
    char str2[] = {"China!"};
    char str3[40];
    mergestr(str1, str2, str3); // 写函数的好处就是让整个程序看起来简单易懂
    printf("%s\n", str3);
}
```