

# C 语言

这次是真的从入门  
到退学//狗头保命

关山口男子技术学院

所有修电脑的里面最菜的那个

写的 C 语言入门书

面向 0 基础

首先，作者在此声明一点，关山口男子职业技术学院，又名华中科技大学，是一所正宗 985，正宗 985，正宗 985!!!（重要的事情说三遍）

这本书大概以这样的方式讲述 C 语言

1. C 语言的简介以及环境搭建
2. 认识 C 语言的运算符和表达式
3. C 语言的三种流程控制
4. 函数式编程
5. 宏定义语句和断言调试
6. 数组介绍和应用
7. 指针简介
8. 结构体的使用
9. 文件的操作

## 第一章 C 语言的环境搭建

你说像我们学计算机的，入门肯定就要学程序设计了。甭管你是 985 还是 211，一本还是二本，本科还是专科，入门课程绝对是程序设计。那么，我们为啥要拿 C 开刀呢？

从最早的计算机编程开始说起，自从图灵与冯诺伊曼提出计算机模型并设计出来了最早的计算机开始，我们就一直在探索如何利用计算机执行一系列操作。有一天，机器语言诞生了，这时人类可以用机器语言指挥计算机做想做的事。然鹅……机器语言有一个大大的弊端就是指令有限还难懂，相比于 01 串虽然强了一点，但指令不仅长，还 TM 非常难懂。后来呢，又有人发明了汇编语言。到现在，中国大部分大学的计算机专业必修课中，汇编语言的大名赫然在列。然鹅，如果你有个高年级的学长学姐，去问问他们做汇编实验的时候内心是个什么感受吧……那感觉真的是一言难尽……

C 语言作为一种高级语言，相比于汇编等初级语言有着完备的语法体系和相对简洁的指令结构。D.M.Ritchie 发明了它（虽然开始是为了电脑游戏哈哈但是确实“造福”人类）作为第五代语言。因为它的祖宗是 ALGOL。后面是 BCPL, B, FORTRAN?哎呀我也忘了，反正这就是第五代语言。

无论是嵌入式系统，还是游戏编程，亦或是强大的 UNIX, LINUX 都是用 C 写出来的。包括小时候的那些电子游戏机内核也是 C 语言。那么，如何搭建一个 C 语言的开发环境呢？

C 语言的编写软件有很多的嗷，例如 DEV-C++, VScode, code::blocks

等。个人更喜欢用 DEV-C++ 一点，而且环境相对好搭建，进入官网然后下载安装就行了，按着它的提示来。这里我推荐 CSDN 的一篇文章讲解 DEV-C++ 的安装的：CSDN 也是一个程序员互相交流的地方 [https://blog.csdn.net/qq\\_40160605/article/details/82940228?ops\\_request\\_misc=%257B%2522request%255Fid%2522%253A%2522159039848319725211958305%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request\\_id=159039848319725211958305&biz\\_id=0&utm\\_medium=distribute.pc\\_search\\_result.none-task-blog-2~all~first\\_rank\\_v2~rank\\_v25-3-82940228.first\\_rank\\_v2\\_rank\\_v25&utm\\_term=dev+c%2B%2B%E6%80%8E%E4%B9%88%E7%94%A8](https://blog.csdn.net/qq_40160605/article/details/82940228?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522159039848319725211958305%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=159039848319725211958305&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_v2~rank_v25-3-82940228.first_rank_v2_rank_v25&utm_term=dev+c%2B%2B%E6%80%8E%E4%B9%88%E7%94%A8)

我这里说一下 VScode 的配置吧。首先，VScode 为啥这么香呢，因为你只要在计算机内部搭建了有关环境，你啥子语言都可以开发（例如我的就可以开发 C, C++, JAVA, python, javascript, html 等）。但不配置相关环境，这个玩意就是个弟弟。所以，环境才是重点。

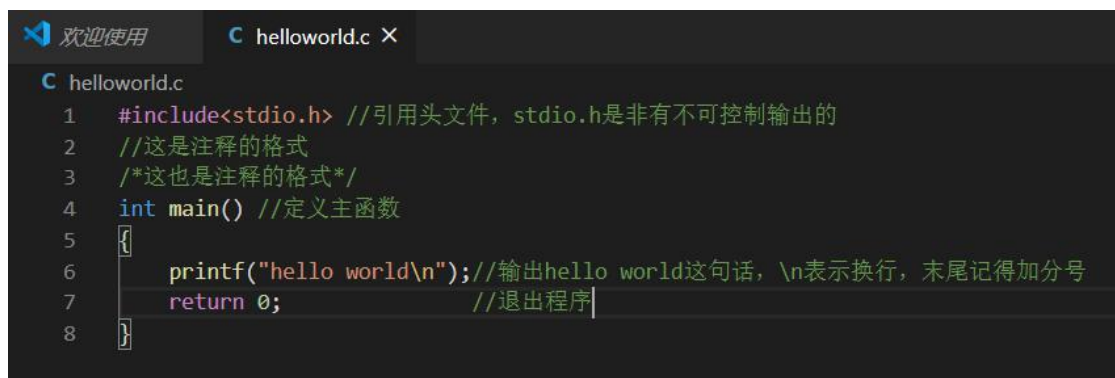
VScode 配置是要加装一些插件的，在插件搜索的图标里面可以搜到。我就装了这样一些插件啊：



要看不清楚图片可以放大看，就在左边的那一栏。

然后环境搭建，VS 只是一个写代码的工具。但环境搭建好了，就算没有 VS，你也可以写程序（文本文档写，然后 cmd 里面跑。不过不能 debug）。环境搭建要用到一个玩意叫做 mingw。可以进入官网去下载。下载好了以后呢点击安装，到最后会弹出一个有很多安装包的界面。这时你把带 g++ 的选上再装，然后添加到路径，这个搭建就算好了。如果我说的你听不懂，很正常。因为别人跟我说的時候我也听不懂。这里你一样可以上 CSDN 去看看，我就不放网址链接了好吧。

安装好环境后，我们来试着编一个 C 语言程序。这个程序将会输出 HELLO, WORLD（程序猿自然懂？//确信）

A screenshot of a code editor window. The title bar shows '欢迎使用' (Welcome) and 'C helloworld.c X'. The code is written in C and includes comments in Chinese. The code is as follows:

```
C helloworld.c
1  #include<stdio.h> //引用头文件，stdio.h是非有不可控制输出的
2  //这是注释的格式
3  /*这也是注释的格式*/
4  int main() //定义主函数
5  {
6      printf("hello world\n");//输出hello world这句话，\n表示换行，末尾记得加分号
7      return 0;           //退出程序
8  }
```

编写的 C 语言程序后缀名叫.C，例如这里的 helloworld.c，注意文件名不要出现空格。这个程序的结构如图所示，相信你能够看懂。

通过这个简单的例子，我们大概了解了 C 语言开发环境的安装以及一个基本 C 语言程序的结构。接下来让我们逐步学习 C 语言的语法内容，并一起完成一些相应的练习吧

## 第二章。运算符和表达式

对于每个萌新而言，运算符和表达式是最头疼但又不得不学的部分。因为没有运算符和表达式，就没有一门编程语言。我们这次大致上，就以这样的方式展开本章：

1. 二进制，十进制与十六进制
2. 对 01 串按位的运算符
3. C 语言的数据类型
4. C 语言的逻辑操作
5. C 语言的算术运算
6. C 语言的经典 I/O 语句：printf 和 scanf

### 1. 二进制，十进制与十六进制

二进制最早的起源来自中国的周易，阴阳组成八卦，八卦表示万物恰恰契合了 01 表示字节，字节表示数据，数据构成程序的思想。

十六进制相比于十进制，多了六个数位，就用 ABCDEF 补齐，小写也可。常常在前面加上 0X 表示这是一个十六进制数

[https://blog.csdn.net/yuanxiang01/article/details/82503568?ops\\_request\\_misc=%257B%2522request%255Fid%2522%253A%2522159080888019725256720886%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request\\_id=159080888019725256720886&biz\\_id=0&utm\\_medium=distribute.pc\\_search\\_result.none-task-blog-2~all~first\\_rank\\_v2~rank\\_v28-2-82503568.pc\\_insert\\_v2&utm\\_t](https://blog.csdn.net/yuanxiang01/article/details/82503568?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522159080888019725256720886%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=159080888019725256720886&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_v2~rank_v28-2-82503568.pc_insert_v2&utm_t)

erm=%E4%BA%8C%E8%BF%9B%E5%88%B6%E5%8D%81%E8%BF%9B%E5%88%B6%E5%8D%81%E5%85%AD%E8%BF%9B%E5%88%B6%E8%BD%AC%E6%8D%A2%E8%A1%A8

这篇博客介绍他们的进位制互相转化。这个二进制化十进制，十进制转二进制数学老师应该讲过，这篇博客也讲得挺好。这里介绍一下二进制转十六进制

每八个空位构成一个字节。而很多情况下我们是两个两个地数。那么，就是  $2 \times 8 = 16$  个位子。四个一滑，就是四个字段。

0000=0, 0001=1, 0010=2, 0011=3, 0100=4, 0101=5, 0110=6, 0111=7, 1000=8, 1001=9, 1010=A, 1011=B, 1100=C, 1101=D, 1110=E, 1111=F

例如, 000110001000001=0X1C41

另外还有反码掩码补码的问题这里有解答

[https://blog.csdn.net/wn084/article/details/79963979?ops\\_request\\_misc=%26request\\_id=%26biz\\_id=102&utm\\_term=%E5%8F%8D%E7%A0%81%E8%A1%A5%E7%A0%81%E5%8E%9F%E7%A0%81%E6%80%8E%E4%B9%88%E8%BD%AC%E6%8D%A2&utm\\_medium=distribute.pc\\_search\\_result.none-task-blog-2~all~sobaiduweb~default-1-79963979](https://blog.csdn.net/wn084/article/details/79963979?ops_request_misc=%26request_id=%26biz_id=102&utm_term=%E5%8F%8D%E7%A0%81%E8%A1%A5%E7%A0%81%E5%8E%9F%E7%A0%81%E6%80%8E%E4%B9%88%E8%BD%AC%E6%8D%A2&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-1-79963979)

## 2. 对 01 串的按位运算

主要是逻辑运算。接触过离散数学的同学肯定是熟不能熟，没有接触过的同学这里只要有高中数学基础也可以听得懂。我们用 0 表示假而以 1 表示真，那么高中我们学过的真值表可以表示成什么呢？

$0 \& 0 = 0, 0 \& 1 = 0, 1 \& 1 = 1$  与

$0|0=0, 0|1=1, 1|1=1$  或

$!0=1, !1=0$  非

另外一个逻辑关系叫做异或，就是相同则真相异则假

$0^{\wedge}0=1, 0^{\wedge}1=0, 1^{\wedge}1=1$

### 3. C 语言的数据类型

C 语言的数据类型大致分为以下几种

整型 `int`，长整型 `long int`，超长整型 `long long int`，短整型 `short`，无

符号整型 `unsigned int`

浮点型 `float`，双精度浮点型 `double`

字符型 `char`

还有布尔型，学习了结构体我们自己还可以定义新的类型

### 4. C 语言的逻辑操作

刚才我们讲到了两个 0/1 串之间的操作，现在我们来描述两个事件或

者条件的操作。这时与或都要变成 `&&` 和 `||`

例如，要表示  $0 < x < 1$ ，就要写成

$x > 0 \&\& x < 1$

要表示色子取偶数点或者三点

$N \% 2 == 0 || n == 3$

### 5. C 语言的运算操作

加号 `+`，减号 `-`，乘号 `*`，除号 `/`，取余数 `%`，注释 `//`，宏定义 `#`，否定 `!`，

三目运算 `?:`：表选择，字符串 `"`，字符 `'`，判断等于 `==`，赋值 `=`，

不等关系 `<`, `>`, `!=`, `<=`, `>=`.



## 6. printf, scanf

这里我不打算再讲下去，放一篇博客自己看看吧，注意%后面跟什么

[https://blog.csdn.net/qq\\_18671205/article/details/88756193?ops\\_request\\_misc=%26amp;request\\_id=%26amp;biz\\_id=102&utm\\_term=printf%E5%92%8Cscanf%E7%9A%84%E5%8C%BA%E5%88%AB&utm\\_medium=distribute.pc\\_search\\_result.none-task-blog-2~all~sobaiduweb~default-0-88756193](https://blog.csdn.net/qq_18671205/article/details/88756193?ops_request_misc=%26amp;request_id=%26amp;biz_id=102&utm_term=printf%E5%92%8Cscanf%E7%9A%84%E5%8C%BA%E5%88%AB&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-88756193)

这里搞几个基础练习

1. 复述常用的 C 语言数据类型

2. 将十进制化二进制

12, 1024, 1100, 12345, 5982310

3. 将十六进制化二进制

0XABCD, 0X2541, 0X0012, 0X00FF

4. 将二进制化十六进制

0011110001000001, 0001001011100010

5. 写出 (1) 的反码, (2) 的补码

00100011, 10010011

### 第三章。C 语言的流程控制

C 语言的最核心语法就是流程控制。流程一般有三类，顺序结构，选择结构和循环结构。对应的有一些语句：if, else, switch, while, do-while, for, 下面我们会逐一讲解它们的使用方法。

1. 流程图与伪代码，顺序结构示例：交换两数值
2. if 语句和 if-else 语句
3. switch 语句的使用，以及为什么很多程序员不喜欢 switch?
4. while 和 do-while 语句的区别
5. for 循环
6. 习题示例

#### 1. 流程图和伪代码

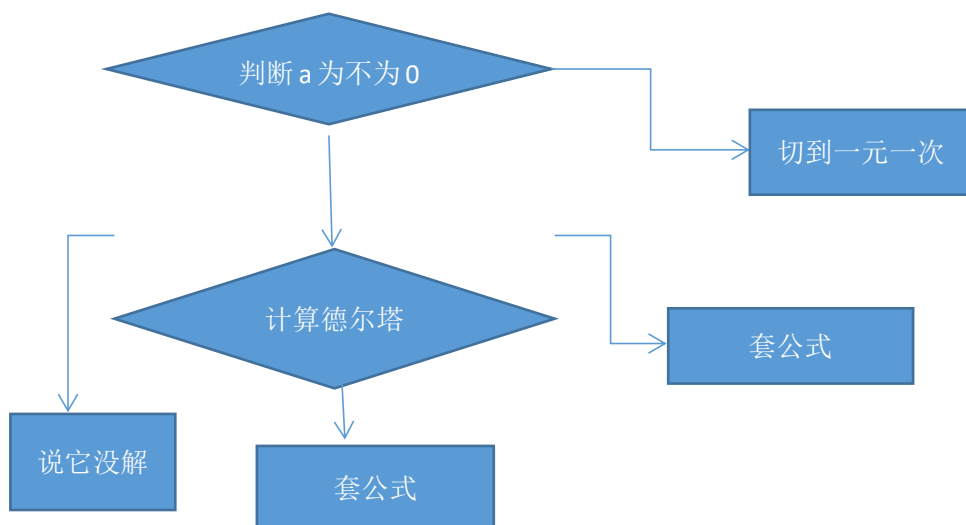
我们知道，算法是一些执行操作的指令集合。但是理解它到底是啥，要执行些什么操作，我们表示疑惑。我们不妨拿个经典例子出来鞭尸//奸笑。我们的这个例子就是非常熟悉的求一元二次方程  $ax^2 + bx + c = 0$  的过程。相信有初中数学基础的童鞋应该不陌生它的求法。如果我们用算法描述它，将会是这个亚子：

①判断  $a$  为不为 0，是则解一元一次方程。若  $b=0, c=0$  则无穷解， $b=0, c \neq 0$  则无解。

②计算德尔塔，看它的取值。大于 0 两个解，等于 0 一个解（暂且这么说吧，因为实际上是两根相等），小于 0 无解（如果你还不认识复数与共轭虚数根你可以这么理解）

③套公式出结果

如果我们要画一个流程图，它大概是这个亚子：



这就是它的一个流程（虽然框图似乎没有画好但是我也懒得修了凑合看把哈哈哈）这就是我们的一个流程图了。伪代码又是啥呢？用自己的记号来描述算法的半成品代码。就看你个人的喜好了。

那么，顺序结构我们就来思考一个问题啊：现在一个玻璃杯里装的是冰可乐，保温杯里面装的是菊花加枸杞。怎么把这两杯里面的东西换过来呢？这个时候你需要一个空杯子，把冰可乐转移到空塑料杯里面，再把菊花加到玻璃杯里面，最后把塑料杯的可乐倒进保温杯。塑料杯里还是啥都没装，但是实现了我们的效果。

现在我们抽象成一个问题： $a=1, b=2$  现在想交换  $ab$  的值，怎么办？

我们编一个程序吧：

```

1  #include<stdio.h> //引用头文件，stdio.h是非有不可控制输出的
2  //这是交换两个变量的值
3  int main() //定义主函数
4  {
5      int a=1,b=2;//这就是我们定义的两个变量
6      int c;//定义了一个空杯子
7      printf("start:%d %d\n",a,b);//输出初始状况
8      c=a;
9      a=b;
10     b=c;//换水，一次赋值就代表把一个杯子里的水倒进另一个杯子
11     printf("end:%d %d",a,b);//输出末尾状况
12     return 0;
13 }
14

```

我们运行试试看：

```

start:1 2
end:2 1

```

结果符合我们的要求

## 2. if 语句和 if-else 语句

我们使用 if 语句来对一件事进行判断。在程序设计里面也是如此。

那么，if 语句应该如何使用呢？if 当然可以直接用。但是我们在使用 if 时，常常还与 else 连用表判断。

我们可以举个例子来说明一下 if 的使用方法。大家都知道，在大学里面，有一门课低于 60 就叫挂科。我们现在就写一个判断你挂没挂的程序（我上学期 C 语言差点挂掉//狗头）

```

#include<stdio.h> //引用头文件，stdio.h是非有不可控制输出的
//这是判断你挂科与否的程序
int main() //定义主函数
{
    int score;
    scanf("%d",&score);//输入分数
    if(score>=60){//判断分数是否大于60，当你的执行语句只有一句可以不要括号，这里为了通用性都加上了
        printf("good,you passed the exam");//输出你过了
    }
    else{//否则
        printf("go back to learn and take another exam");//输出gun回去重修一年再来补考吧
    }
    return 0;
}

```

我们可以运来试一试

```
56
go back to learn and take another exam
```

```
69
good, you passed the exam
```

```
-----
Process exited after 10.27 seconds with return value 0
请按任意键继续. . .
```

实现需求

### 3. switch 语句和多重 if-else

我们的 if-else 语句里面可以有很多层，形成 if-else if...-else 的结构。

它适合于多条件判断问题。而对于离散型条件判断，我们为了简化他，

又特意加了一条语句就是 switch。这个 switch 语句在 Python 语言里面

是没有的。当然，switch 可以做的 if-else 也能做就是代码量可能长了

些。但是 if-else 可以做的用 switch 就不那么容易了。

为了解释两种语句的语法规则，我们看一个例子。这个例子是做一个

选择题。1234 就表示选项 ABCD。

```
#include<stdio.h>
//这是一个选择题程序，我们先用if语句实现
int main()
{
    printf("现在我们来做一个选择题，1234表示选项ABCD\n");
    printf("下列哪一项不是三国时期的人物（）\nA.曹操\nB.诸葛亮\nC.刘禅\nD.孙权\n"); //这题答案选A
    //曹操这个魏武帝的称号是曹丕追赠的
    int choice; //表示选项
    scanf("%d",&choice);
    if(choice==1){
        printf("right");
    } //这里为了讲解，就对每一个选项进行分析
    else if(choice==2){
        printf("wrong");
    }
    else if(choice==3){
        printf("wrong");
    }
    else{
        printf("wrong");
    }
    return 0;
}
```

```
现在我们来做一个选择题，1234表示选项ABCD
下列哪一项不是三国时期的人物（）
```

```
A. 曹操
B. 诸葛亮
C. 刘禅
D. 孙权
```

```
1
```

```
right
```

```

#include<stdio.h>
//这是一个选择题程序，我们用switch语句实现
int main()
{
    printf("现在我们来做一个选择题，1234表示选项ABCD\n");
    printf("下列哪一项不是三国时期的人物（ ）\nA. 曹操\nB. 诸葛亮\nC. 刘禅\nD. 孙权\n"); //这题答案选A
    //曹操这个魏武帝的称号是曹丕追赠的
    int choice; //表示选项
    scanf("%d",&choice);
    switch(choice){ //判断变量的离散取值
        case 1:printf("right");break; //表示变量取值1时输出正确提示，然后要有break
        case 2:printf("wrong");break;
        case 3:printf("wrong");break;
        case 4:printf("wrong");break;
    }
    return 0;
}

```

相信各位已经大致看懂了它们的语法结构。关于 switch 还想多说两句，这里的判断是基于离散型随机变量的取值（概率论老师突然附体），但取值不局限于整数，也可以是字符。执行操作以后记得带 break，否则它执行完以后会再执行下一条语句，就从选择结构变成了顺序结构了。不信的话或者觉得难以理解的话也很简单，你可以把上面这个例子 case1 的 break 去掉看看是个什么效果。

另外就是有一天我看到一篇文章，说现在的 C/C++ 程序员都并不太喜欢用 switch 反而喜欢 if-else。首先我觉得 switch 其实用起来也挺方便的，可以减少代码量。但是，if 判断的是不等式，switch 判断的是取值。相对来说，if 是离散连续通吃，而 switch 只能判断离散，这一点上就落后于 if 语句。另外也是像刚才提到的，有很多人写 switch 语句没有 break 掉就会导致一批错误。

#### 4. while 和 do-while

while 和 do-while 都是用于循环的。不同的是，一个是先判断在执行，一个是先执行后判断。它们到底是个啥子区别，又有些什么联系，我们编写一个实例看看叭

```

#include<stdio.h>
//现在我们试一试while语句的用法
int main()
{
    int i=2;
    while(i<10){
        i++;
        printf("%d ",i);
    }
    //这是while的格式
    i=2;
    printf("\n|*****\n");
    do{
        i++;
        printf("%d ",i);
    }while(i<10);//这是DO-WHILE的格式，记得打分号
    return 0;
}

```

```

3 4 5 6 7 8 9 10
*****
3 4 5 6 7 8 9 10

```

我们看看它们的运行结果如何： 好吧并没有什么卵

区别//尬笑。这也说明，我们的 while 与 do-while 是可以等价代换的。

## 5. for 语句的灵活使用

For 语句是一种改良过的 while 语句，是为数不多的三目运算符之一（之前就接触过（?:））它的背景来源于语句结构 while 里面的

```
int i=0;
```

```
while(i<n){
```

```
    {do:.....;}
```

```
    i++;
```

```
}
```

这样一个结构。For 语句的基本语法形式像这样：for(计数变量初值；

条件；计数变量增加)。如果你表示看这玩意不太习惯，你也可以看



下面的例子。当年高斯用了一分钟从 1 加到 100，今天我计算机用 10 秒加到 100，不是问题//卢本伟牛逼（破音）

```
#include<stdio.h>
//现在我们试一试for语句的用法
int main()
{
    int i;//规定计数变量
    int sum=0;//规定求和变量
    for(i=0;i<101;i++){
        //在C89中，计数变量不能在循环体内定义，但是C99以后就可以了
        //这里为什么是101，这是因为i只加到100就完了，但100小于101是明显的。
        //当然，我们也可以改成i<=100
        sum+=i;
    }
    printf("%d",sum);
    return 0;
}
```

5050

[Done] exited with code=0 in 1.119 seconds nice, 下面自己试试用另外

两种方法写循环体

6. 现在我们看一些习题来巩固一下刚才所学的知识叭

首先第一个问题就是求方程的解这样一个问题。公式里面要用到根号，可以从 math.h 头文件里面引用。大家先思考一下，我再来说说我

的代码。

请输入方程的三个系数:	请输入方程的三个系数:
1	1
2	2
0	1
x1=0.00, x2=-2.00	x1=x2=-1.00

请输入方程的三个系数:

1

2

2

此方程无解



```

#include<stdio.h>
#include<math.h>
int main()
{
    printf("请输入方程的三个系数: \n");
    int a,b,c;
    scanf("%d%d%d",&a,&b,&c);
    if(a==0)printf("对不起, 这不是个一元二次方程。 \n");
    else{
        int delta=b*b-4*a*c;
        float x1,x2;
        if(delta>0){
            x1=(-1*b+sqrt(delta))/(2*a);
            x2=(-1*b-sqrt(delta))/(2*a);
            printf("x1=%.2f,x2=%.2f\n",x1,x2);
        }
        else if(delta=0){
            x1=-1*b/(2*a);
            printf("x1=x2=%.2f",x1);
        }
        else printf("此方程无解\n");
    }
    return 0;
}

```

上面的代码有个地方有点小问题会造成 BUG, 细心的看官发现是哪里了吗? 对了就是 `else if` 那一行, 并不是 `=`, 而是 `==`。否则的话它就会只有两个解或者无解哦。(这一赋值指令在条件框里面出现, 在 JAVA 里面会直接给他报错的)

接下来这个题目倒是不难, 求所有的三位数, 满足平方的后三位等于自己。还是先想想怎么写, 稍后我再放代码。

```

376
625

```

```

#include<stdio.h>
int main()
{
    int a;
    for(a=100;a<1000;a++){//因为按照题意，这个数得是个三位数。
        //如果没有数论基础来找到更好的算法，干脆遍历完得了
        int b=(a*a)%1000; //平方除以1000的余数
        if(b==a)printf("%d\n",a);//判断
    }
    return 0;
}

```

下面这个问题就更加直截了当了，让你打印杨辉三角前 10 行。不过如果有狠人一个一个打印，我也是佩服宁的胆量

预习了数组的可以先做，控制空格你们再来想办法吧。

这个问题十分经典，在 CSDN 上面还可以找到源代码。

```

#include <stdio.h>
int main() {
    int i,j,n;
    scanf("%d",&n);
    int a[n][n];
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            a[i][j]=1;
    } //给数组赋初值
    for(i=0;i<n;i++)
    {
        for(j=0;j<i+1;j++) //注意:这里是i+1而不是i, i值j可以取
        {
            if(i==j||j==0)
                a[i][j]=1;
            else
                a[i][j]=a[i-1][j-1]+a[i-1][j];
        }
    }
    for(i=0;i<n;i++)
    {
        for(j=0;j<i+1;j++)
        {
            if(i==j)
                printf("%d\n",a[i][j]);
            else
                printf("%d ",a[i][j]);
        }
    }
    return 0;
}

```

如果你自认为数理基础还不错，我推荐一下这道题：

用牛顿法给出方程  $3x^3 - 4x^2 - 5x + 13 = 0$  的数值解，精度要求  $1.0e-3$

我们可以回忆一下牛顿法的过程是什么，就是求切线与  $x$  轴交点然后作为下一次迭代的值嘛。切线最主要的一点恐怕就是斜率。对这个函数求导（如果你不明白什么是导数，可以看看高二的数学书里面有写）

$y' = 9x^2 - 8x - 5$ ，迭代公式就是新值等于老值减去函数值比上导值

```
#include <stdio.h>
#include <math.h>
//这是函数定义，在第五章会讲到。这里就是说我定义了函数的表达式可以直接用
float y(x){
    return 3*x*x*x-4*x*x-5*x+13;
}
float u(x){
    return 9*x*x-8*x-5;
}
int main() {
    float x=4;
    float newi=5; //随机设定初值
    while(fabs(newi-x)>0.001){ //fabs是绝对值
        x=newi;
        newi=x-y(x)/u(x); //迭代
    }
    printf("%.3f",x);
    return 0;
}
```

结果 4.417，挺好的。

后面几个练习我就不告诉你们答案了，你们可以自己在 CSDN 里面搜索。里面有很多好的文章和经典解法

第一题：打印 1000 以内的所有质数

第二题：寻找所有的水仙花数，定义自己百度我就不说了

如果你们觉得还不够带劲，VERY GOOD，可以说明你们比我大一那会儿强。这个题目留给你们：一辆卡车违反交通规则，撞人后逃跑。现场有三人目击该事件，但都没有记住车号，只记下车号的一些特征。

1. 牌照的前两位数字是相同的；
2. 牌照的后两位数字是相同的，但与前两位不同。
3. 四位的车号刚好是一个整数的平方。

现在要你求出这个车牌号到底是多少

提示：有一个有技巧的办法可以筛掉一大批，但是要有数论基础；另外一种办法就简单点，比较暴力，但是时间长，看你学的怎么样吧，反正做得出来就不错了

程序设计中流程控制是最基础也是最核心的部分，学会了它就可以编写很多有意思的东西了。当然，这一部分你如果理解起来还有一些困难，或者说，我讲的不够清楚，没关系。推荐看看那本 *C PRIMER PLUS*，虽然顺序很奇怪，但是确实是 C 语言学习的好书。另外可以进 CSDN 或者 LEETCODE 看一些有意思的题目，体会那种思想。

## 第四章。宏定义简介

这一章的话只是简单提一下吧，毕竟感觉用途不大

1. 宏定义：定义某个常量的值，使之能够在每个函数里面都可以使用
2. 带参宏定义：宏定义的字面量里面含有参数，相当于一个函数吧
3. 静态断言调试：排查 BUG 的手段之一。引用头文件 `assert.h` 实现

我们不举太多例子，有关静态断言调试的可以自己去看网上查阅一下，也可以看看 C PRIMER PLUS，这一节不讲没有太大影响

无参宏定义：

```
#include <stdio.h>
#define pai 3.1415926535
int main(){
    int r=3;
    printf("area: %.2f, circle: %.2f", pai*r*r, 2*pai*r);
    return 0;
}
```

area:28.27,circle:18.85 这个程序定义了圆周率，求解面积和周长

含参宏定义：

```
#include <stdio.h>
#define MAX(a,b) (a>b)?a:b
int main(){
    int x,y,max;
    printf("Plasy input two numbers :");
    scanf("%d %d",&x,&y);
    max=MAX(x,y);
    printf("The max is: %d\n\n",max);
}
```

## 第五章. 函数式编程

之前我们使用的 `printf`, `scanf` 等都属于函数。包括 `math.h` 库中的 `sqrt`, `pow` 等函数都是自身携带的函数。那么我们能不能写自己的函数呢？

写代码时我们可能会碰上这样一些情况；代码中可能出现了一些重复片段，如果我们能够把重复片段集中起来，这样我们的代码行数和开发效率是不是就大大提高了呢？

今天我们就来聊聊函数式编程思想

1. 函数式编程思想与示例
2. 一些计算机玄学思想
3. 函数的参数：形参/实参（容易糊的概念）
4. 函数的返回值与调用
5. 习题示例

### 1. 函数式编程思想

那么什么是函数式编程呢？又是什么时候兴起的呢？

函数式编程，是一种能够减轻代码量，随时调用的一种灵活语言结构。

函数式编程的语言最早是由阿隆佐-邱奇提出来的，他将算法表示为一系列的函数嵌套调用，主函数也是函数中的一个，也是唯一一个。

可以作为测试段，但是需求与功能，主要还是靠函数完成。当然，这里区分一下概念啊，我们用的函数式思想去编写 C 程序，但是 C 程序还是命令式编程，不是严格的函数式编程语言//有些糊对吧。严格意义上的函数式编程语言就有 `scala`, `LISP` 等语言。这里我们只是借用其思想而非外衣，正如 C 中也可以看到 C++ 中面向对象的影子，尽管他

不是一门面向对象的语言。

那么，编写C函数段（本质就是一段重复的代码）是以什么格式编程呢？如果你明白了先声明后引用，那么你 duck 以先声明函数，在最后再编写。声明函数，先声明函数返回值的数据类型（若没有返回值则为空），再定义函数名称，最后是它的参数列表。编写时就写他的自己的代码。

当然了，光看上面的理论玩意儿，你会觉得这是一门玄学。没关系（pa jiu pa）这没什么大不了的。你可能不知道什么是函数返回值，可能不知道什么是参数。没关系，我们可以展示一个例子：

```
#include <stdio.h>
void saohua();//声明了一个函数，没有返回值和参数
int attack(int a);//声明了一个整型函数，参数为a,返回值是个整数
int main(){
    int a=100,b;
    saohua();//调用函数，由于是空类型，没有返回值
    b=attack(a);//调用整形函数，参数为a
    printf("%d",b);
    return 0;
}
void saohua(){//空函数定义
    printf("em, I think I'm the most handsome in the world\n");
}
int attack(int a){//整形函数定义
    int b=a-50;//此处b是个局部变量（中间变量）没有什么影响
    return b;
}
```

```
[Running] cd "c:\Users\malia\Desktop\神经网络学习"
em, I think I'm the most handsome in the world
50
[Done] exited with code=0 in 4.871 seconds
```

那么，我们大概已经了解了函数的定义方法了。我们应该如何学习函数编程的思想，并且广泛应用呢？

## 2. 一些计算机的玄学思想



一个非常重要的概念就是自顶而下。自顶而下是一种逐步求精的设计程序的过程和方法。对要完成的任务进行分解，先对最高层次中的问题进行定义、设计、编程和测试，而将其中未解决的问题作为一个子任务放到下一层次中去解决。这样逐层、逐个地进行定义、设计、编程和测试，直到所有层次上的问题均由实用程序来解决，就能设计出具有层次结构的程序。

那么，说白了，这个玩意啊，就是让你分解问题的需求，使之分解为一个个小功能，功能与功能之间允许相互嵌套，只要不是 *dead loop*，就可以按照一定的次序解决程序设计需求。设计程序之前的问题分解，以及我们的函数，变量甚至于面向对象里面的类，对象，接口都需要设计与分解。这一个过程就被称为代码架构。（一个架构师的工资比普通程序员高哦）。让我一下子举一个自顶而下设计的例子我可能一下子举不出来，到了某个题目要用到自顶而下分析的时候我再来具体谈谈吧。

然后，我们接着想要谈的问题叫做递归。什么是递归？就是求解某一函数值的时候，是倒着往回找到初始值，然后再从初值步步推回来的过程。我们可以看一个例子。这个例子叫做汉诺塔问题（气死梵天系列）。说：古印度有一个牢房，犯人们想要从监狱里面出来也很简单。大堂里有三根柱子，有八八六十四个大小各异的盘子，按着从上到下是从小到大的次序在排列。如果你能够把盘子从一根柱子移动到另外一个柱子就算你赢。但是移动的时候，只能是小盘子放在大盘子上。一次移动一个盘子。现在，我问你，如果一秒移动一个盘子，犯人最



少要花多长时间才能够出狱啊？//奸笑

我们这么思考啊，假设移动  $n$  个盘子的方法数为  $A$ ，原来盘子都在金柱子上，现在要把它移动到银柱子上，我们先把上面的盘子移动到铜柱子上，最下面那个最大的盘子再移动到银柱子上，又是 1 步。随后我们再把铜柱子的  $n-1$  个盘子拿到银柱子上，又是  $A$  步。所以，这个递推关系可以写成： $H(n)=2H(n-1)+1$ 。用程序设计思想描述如下：

```
#include <stdio.h>
long long int hanoi(int n){
    if(n==2) return 3;
    else return (2*hanoi(n-1)+1);
}
int main()
{
    printf("%lld",hanoi(64));
    return 0;
}
```

我这里还没有搞 64 块这么多，我就搞了 32 块，结果你看看这个步数要多少好吧：4294967295 秒，得花上 860000 小时。32 到 64，得乘上 32 个 2。想想这种爆炸式的时间增长吧，你就是死你也搞不完的，放弃吧哈哈哈。。。

另外一个问题是斐波那契数列的问题。一对兔子生了一对小兔子，第二年两队兔子又生了两队兔子。。。形成了 1, 1, 2, 3, 5, 8。。。的数列。那么，这个数列我们怎么输出呢？

```
#include <stdio.h>
int fabonacci(int n){//输出斐波那契数列，返回值为整数
    if(n==1||n==2) return 1;//开始两项为一
    else return (fabonacci(n-1)+fabonacci(n-2));//后面就是它的递推关系
}
int main()
{
    for(int i=1;i<=15;i++){
        printf("%d\n",fabonacci(i));//输出前十五项
    }
    return 0;
}
```

然后我们可以看一下它的运行效果

```
[Running] cd "c:\Users\malia\Desktop\神经网络学习\" && gcc helloworld.c -o helloworld && "c:\Users\malia\Desktop\神经网络学习\helloworld
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
[Done] exited with code=0 in 5.811 seconds
```

然鹅，使用递归代码量虽然减少，而且条件的表达更为流畅，回溯的思想很是新奇，有一个最致命的问题。递归如果不经过程序员的加工，天然的递归时间复杂度和空间复杂度都是很高的你知道伐，不信你可以把15改成那个60项，他就会给你卡到翻。当然，我们的降低处理也是有技巧的。这一技巧被称为递归树分析，因为你看， $f(n)=f(n-1)+f(n-2)$ ,  $f(n-1)=f(n-2)+f(n-3)$ , 所以就要算上两次  $f(n-2)$ 。这一个不浪费，10000个呢？10000000个呢？故古人云：积土成山，风雨兴焉；积水成渊，蛟龙生焉。此并非无理。像在他们麻省理工那里，但凡讲解算法课就必然是要谈递归树（recursion tree analysis）的。包括我在墨尔本留学那段时间，接触到他们西方人讲计算机算法的时候，基本都是脱不开递归树，以及用递归树分析算法复杂度的（虽然我在墨尔本大学是主攻人工智能与机器学习的算法问题而且还一堆听不懂//尬笑）

### 3. 函数的实际参数和形式参数

啥子是形式参数？实际参数又为何物？我初接触C语言的时候反正是没搞清楚。可能看官们也这么觉得吧//确信。那么，我们今天谈一

下这两个概念的区别 OK? 先还是看个例子吧

```
#include <stdio.h>
int max(int a,int b){//求解两数较大值的程序
    //此处a,b为形式参数
    return (a>b?a:b);
}
int main(){
    int a=1,b=2;
    printf("%d\n",max(1,2));//实际参数为1, 2
    printf("%d",max(a,b));//实际参数为a,b
    //这里实际参数名字换成x,y或者其他没有影响的
    //它看的是他的值又不是它的名字
    return 0;
}
```

这个例子就展示了一下形参和实参的区别。形参的名字不重要，只是便于使用 and 规定参数类型。实际参数可以是变量也可以是数值，是在调用的时候使用的玩意儿。后面我们再看到函数调用的时候可以再看一下吧。注意啊，计算机函数的参数相当于数学函数的自变量，数学函数中的参数在计算机函数中是不能出现的（最好不要）

#### 4. 函数的返回值和调用

函数分有返回值和无返回值两种。无返回值就是一段代码来执行一系列操作，常用空型 `void` 表示；而有返回值的则是进行一些计算，计算出最终结果。这个最终结果可以是整数，浮点数，字符，也可以是指针。但不能是数组，或者有多个返回值毕竟中学数学里面函数只有一个值与自变量对应（但是在 `python` 语言中，函数的返回值可以有两个，也可以返回一个列表）

调用函数就是在测试代码段 `main()` 函数或者是其他函数需要进行嵌套的时候啊，对函数进行使用。使用时要注意，参数列表中每个参数的

数据类型要对应。

#### 5. 习题分析：

这节我们来看几个例子。

首先第一个问题，使用函数的方法，实现阶乘运算，计算出  $10!$

这道题大致上有两种解法。第一种是递推的方法（说白了就是循环），

第二种就是递归算法（分暴力递归和递归树快速幂算法，快速幂算法

接下来一个例子我稍微展示一下，想要详细资料可以上 CSDN，也可

以上 B 站，有个小学生讲的挺好的//wtcl）大家先思考一下，稍后我来

分享解法。

```
#include <stdio.h>
int power(int n){//函数名: power;形式参数: n;返回值类型: 整型
    int p=1;//设置返回值初值
    for(int i=1;i<=n;i++){//设置计数变量i, 计数循环
        p*=i;
    }
    return p;//返回
}
int main(){
    printf("%d",power(10));//调用函数, 实际参数为10
    return 0;
}
```

```
#include <stdio.h>
int power(int n){//函数名: power;形式参数: n;返回值类型: 整型
    if(n==1||n==0) return 1;//0! =1, 1! =1
    else return (n*power(n-1));//n!=n*(n-1)!
}
int main(){
    printf("%d",power(10));//调用函数, 实际参数为10
    return 0;
}
```

```
[Running] cd "c:\Users\malia\Desktop\神经网络"
3628800
[Done] exited with code=0 in 10.914 seconds

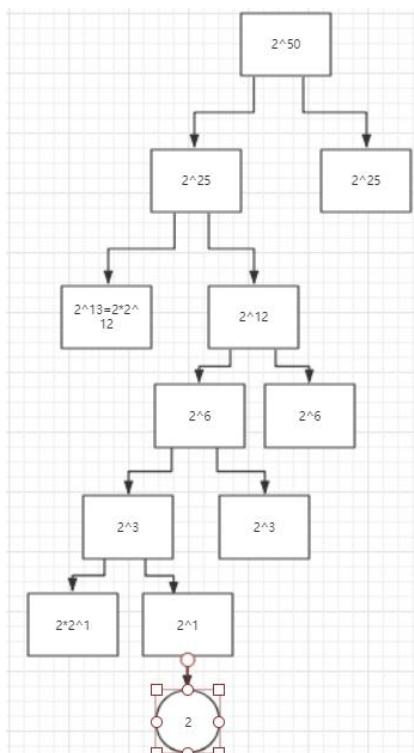
[Running] cd "c:\Users\malia\Desktop\神经网络"
3628800
[Done] exited with code=0 in 18.488 seconds
```

下面一题与阶乘十分类似，求解 2 的 10 次方，要求不准用循环，只能用递归算法。

```
#include <stdio.h>
int power(int n){//函数名: power;形式参数: n;返回值类型: 整型
    if(n==0) return 1;//2^0=1
    else return (2*power(n-1));//2^n=2*2^(n-1)
}
int main(){
    printf("%d",power(10));//调用函数，实际参数为10
    return 0;
}
```

也许只让你算 10 次方你感觉还要，我现在要你算 50 次方，怎么让时间最短内存占用最小呢？这就要用到一个东西叫做快速幂算法。

我们先来说一下快速幂算法的基本思路。按着麻省理工他们的讲法，我也学着画一个递归树：



这个递归关系说明，我们并不用回溯 50 次那么多啊，只要回溯 6 次，充分利用计算重复性，就可以把本来是线性的时间复杂度变成对数模式。

```

#include <stdio.h>
int power(int n){//函数名: power;形式参数: n;返回值类型: 整型
    if(n==1) return 2;//2^1=2
    else{
        if(n%2==0) return (power(n/2)*power(n/2));//分治思想
        else return (2*power((n-1)/2)*power((n-1)/2));//若不能均分
    }
}
int main(){
    printf("%d",power(10));//调用函数, 实际参数为10
    return 0;
}

```

这里

的算法被称为快速幂算法，它的背后是一种计算机科学的重要思想：*divide-and-conquer*，分治算法思想。就类似于高一数学里面的二分法查找零点，那也是分治思想的一个实例。只不过那里是分治查找算法，这里是分治递归算法。

现在第三个问题的话，就比较头疼了啊。完美数的定义：自己百度吧哈哈，这里给个例子， $28=1*28=2*14=4*7$ ， $1+2+4+7+14=28$ 。

现在，用函数式编程，求出1000以内的所有完美数。我这里不给出答案，各位自己思考，实在不行上CSDN上面搜索。这里我提示一下，首先对于这个程序，用自顶而下的观念，把功能分解为：遍历1-1000，1.判断每个数的因数：用小于每个数去除，看余数多少 2.对每个因数求和 3.判断是不是完美数。是就输出。当然了，这是我的一个想法，可能不是最优解。如果有时间复杂度更小的算法，你可以随使用。

第四个问题，函数输出1000以内的所有质数

第五个问题，对于1000以内的所有偶质数，验证哥德巴赫猜想//你还是杀了我吧.jpg



## 第六章. 数组

在 C 语言中一个重要的基本数据结构类型就是数组，它表示一系列相同数据类型的数据的集合。有了数组以后，我们便可以把一些元素放在一起进行操作。

数组是一个重难点，我们将以如下方式展开：

1. 数组的定义与使用
2. 数组元素的操作
3. 习题解析

### 1. 数组的定义与使用

数组是指一些相同数据类型的量组成的集合。有整型数组，浮点数数组，字符串，甚至数组本身也能构成元素形成多维数组。数组元素的顺序和平时我们的习惯不太一样，因为它是从 0 开始算起，最后一项是  $a[n-1]$ 。所以，如果你们看到一个人数数 0, 1, 2, 3... 请不要嘲笑，那个人一定是程序猿/媛。

声明数组，可以像这样：`int a[50]`；表示一个容量为 50 的整型数组。当然，也可以这样：`int a[]={1,2,3,4}`；表示一个数组，项分别为 1, 2, 3, 4。那么数组的输入呢？可以使用之前的循环结构一个个输入。如果是字符串，就不用这么麻烦了。直接 `scanf("%s",string)`；就可以了。注意哦，这里的 `string` 我们没有加 `&`。

### 2. 数组元素的操作

数组元素的操作可以通过下标引用来实现。例如：`a[2]--`；`x=a[3]+2`；等等。基本好讲的内容并不多，我们来看几个例子加强理解。

第一个例子是有关函数 `sizeof` 的。这个函数返回参数的字节长度。我们可以看个例子。

```
#include <stdio.h>
int main(){
    int a[10];
    printf("%d\n",sizeof(a[2]));
    printf("%d",sizeof(a));
    return 0;
}
```

```
4
40
```

第二个例子是将整数的二进制形式打印出来，要求数组实现。这里为了方便，就干脆规定这个数就是 25。用容量为 5 的数组就可以实现了。那如果这个数我自己规定怎么求，你自己想吧哈哈。因为这个例子就是一个简化的问题，程序猿要能够将问题简化。

```
#include <stdio.h>
int main(){
    int x=25;
    int s[5];
    for(int i=0;i<5;i++){
        s[i]=x%2;
        x=x/2;
    }//小学老师教过短除法的
    for(int i=4;i>=0;i--){
        printf("%d",s[i]);
    }//没想到吧，我还可以倒着输出啊
    return 0;
}
```

最后结果 11001 挺好的哈哈

### 3. 习题解析

本次我们说几个有代表性的问题吧。

第一个叫做数组的排序。我们的排序算法有很多种，什么冒泡排序，鸡尾酒排序，快速排序，基数排序，桶排序，堆排序，选择排序。。。实在太多啦。我们今天就来谈三种排序算法（如果还没学数据结构我



们先就学这几种吧)

我们假设待排序的数组是 35142786 这八个数好吧

首先是冒泡排序法。

然后是选择排序法

再来是快速排序法，也就是分治排序算法。

```
#include <stdio.h>
int main(){
    int a[8]={3,5,1,4,2,7,8,6};
    //设定数组，这次用冒泡法
    int t;
    for(int i=0;i<8;i++){//外循环
        for(int j=0;j<8-i;j++){//内循环
            if(a[j]>a[j+1]){//交换
                t=a[j];
                a[j]=a[j+1];
                a[j+1]=t;
            }
        }
    }
    for(int i=0;i<8;i++){//输出
        printf("%d",a[i]);
    }
    return 0;
}
```

```
#include <stdio.h>
#define n 8
int main(){
    int a[8]={3,5,1,4,2,7,8,6};
    //设定数组，这次用选择法
    int t;
    for(int i=0;i<n-1;i++){
        int min=a[i]; //最小值变量
        int p=1; //真假变量
        for(int j=i+1;j<n;j++){
            if(min>a[j]){
                p=0;
                t=min;
                min=a[j];
                a[j]=t;
            } //若发生交换就让逻辑变量为假
        }
        if(p==0){
            t=a[i];
            a[i]=min;
            min=t;
        } //交换
    }
    for(int i=0;i<8;i++){//输出
        printf("%d",a[i]);
    }
    return 0;
}
```

```

#include <stdio.h>
void quicksort(int arr[],int low,int high){
    int first = low;
    int last  = high;
    int key = arr[first];
    if(low >= high)
        return;
    while(first < last)
    {
        while(first < last && arr[last] > key)
        {
            last--;
        }
        arr[first] = arr[last];

        while(first < last && arr[first] < key)
        {
            first++;
        }
        arr[last] = arr[first];
    }
    arr[first] = key;

    quicksort(arr, low, first-1);
    quicksort(arr, first+1, high);
}

int main(){
    int a[8]={3,5,1,4,2,7,8,6};
    //设定数组，这次用快排
    quicksort(a,0,7);
    for(int i=0;i<8;i++){//输出
        printf("%d",a[i]);
    }
    return 0;
}

```

告诉你们一个秘密，其实快排有自己的指令代码可以直接用哦

不过说到快排，我倒是想起来另外一个问题：二分查找。为了降低问题难度，我们不妨就让这个查找数组是从小到大的排列。1, 3, 4, 5, 7, 13, 15, 20 八个数好吧。

最简单的方法当然是线性遍历整个数组，比较八次。现在我们使用

divide-and-conquer, 例如我们要查找 4, 首先比较 4 与  $(5+7)/2=6$ 。很显然,  $4<6$ 。所以我们在 1, 3, 4, 5 里面查找。 $(3+4)/2=3.5<4$ , 我们查找 4, 5。 $(4+5)/2=4.5>4$ , 我们把目标锁定为 4。查找到目标, 过程结束。

```
#include <stdio.h>
int binary_search(int key,int a[],int n) //自定义函数binary_search()
{
    int low,high,mid,count=0,count1=0;
    low=0;
    high=n-1;
    while(low<high)    //查找范围不为0时执行循环体语句
    {
        count++;    //count记录查找次数
        mid=(low+high)/2;    //求中间位置
        if(key<a[mid])    //key小于中间值时
            high=mid-1;    //确定左子表范围
        else if(key>a[mid])    //key 大于中间值时
            low=mid+1;    //确定右子表范围
        else if(key==a[mid])    //当key等于中间值时, 证明查找成功
        {
            printf("查找成功!\n 查找  %d 次!a[%d]=%d",count,mid,key);    //输出查找次数及所查找元素在数组中的位置
            count1++;    //count1记录查找成功次数
            break;
        }
    }
    if(count1==0)    //判断是否查找失败
        printf("查找失败!");    //查找失败输出no found
    return 0;
}
```

```
int main()
{
    int i,key,a[100],n;
    printf("请输入数组的长度: \n");
    scanf("%d",&n);    //输入数组元素个数
    printf("请输入数组元素: \n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);    //输入有序数列到数组a中
    printf("请输入你想查找的元素: \n");
    scanf("%d",&key);    //输入要查找的关键词
    binary_search(key,a,n);    //调用自定义函数
    printf("\n");
    return 0;
}
```

最后的一个问题留给各位自己思考。编写一个程序, 输入一个培训班里面 10 个同学的 C 语言成绩和姓名并排序。这道题预习了结构体的同学可以用结构体, 没有预习的也可以做就是麻烦点。

## 第七章。指针

指针指针,指向的是一个地址。那么,什么是地址?什么是存储?引入指针又有何作用?指针会带来怎样的便利?可以说,指针是C/C++的特有现象,也就是从这一章开始,程序设计的难点开始高密度出现。

从何说起呢?我们应该想想上一章函数式编程的内容,就从那里说起吧。 . . . . .

问题: 交换两个数的值。

```
#include <stdio.h>
void swap(int a,int b){
    int t;
    t=a;
    a=b;
    b=t;
}
int main(){
    int a=1,b=2;
    swap(a,b);
    printf("%d %d",a,b);
    return 0;
}
```

你以为这个程序没有问题,但你以为终究是你以为。如果你运行这个程序,你会惊奇的发现它什么都没有实现,并且你是找不到BUG的。では、一体なぜですか? (so,why?)

我们首先要了解程序设计过程中的存储模型。C语言中,计算机内存大致分为4个部分(也有人说成五个):代码区,静态内存,栈区和堆区。(静态内存又有人分为初始化变量内存和未初始化变量内存)那么,宏定义量,全局变量等都被分配在静态内存里面。静态区下面的代码区是一堆文本,占用不得多少内存。一般而言,静态内存

的内存占用是最大的。我们自定义的函数里面都是局部变量，连同他的参数一起，被存储在了这个栈区里面。像这个函数的参数 a,b 跟主函数的 a,b 压根就不是一个概念。swap 函数的参数是另外分配的内存。所以啊，我们操作的只是个假的参数。真正的 a,b 并没有被我们操纵掉。因为栈区的代码段在函数经过调用后会被自动释放掉，相当于这个 swap 啥都没有干过（一些喜欢喝心灵鸡汤的别跟我杠说啥子过程更重要啊过程美啥的，lz 不吃这一套。个程序调用就 TM 调用了是执行了就执行了哪儿来那么多废话）

那怎么办呢？在 C 语言中，我们就规定了一样语法，可以对地址进行直接操作，从函数所在的栈区牵根线到静态内存去跨区更改，就可以了//诶，你他娘的还真是个天才.jpg

之前讲 I/O 的 scanf 和 printf 的时候我们发现了个鬼现象，就是 scanf 要带个 & 但是 printf 莫得。这里的 & 运算符就是取变量的地址。当然，他也有一个对应运算。那就是\*。\* 做乘号讲是个双目运算，但作为解码符号讲是个单目运算。例如说 \*p, \*p 这个整体就是一个数值，但 p 自己是一个地址。

那么，这个程序怎么修改呢？

```
#include <stdio.h>
void swap(int *a,int *b){
    int t;
    t=*a;
    *a=*b;
    *b=t;
}
int main(){
    int a=1,b=2;
    swap(&a,&b);
    printf("%d %d",a,b);
    return 0;
}
```

有很多朋友对这两个运算符不太理解，这里我借用别人的话来解释一下：1、& 可以作为取地址操作符，获取对象或者叫变量在内存中的地址（取地址操作符只能用于左值）；2、& 也可以作为引用操作符，网上很多人也称间接引用，在实际中具体就是作为另外一个变量或者对象的别名：& 的这种作为间接引用的使用，在 C++ 中多数用于函数的参数中。

注意：& 作为引用操作符的时候呢有如下几点必须注意的：

- 1、& 在此不是求地址运算，而是起标识作用；
- 2、类型标识符是指目标变量的类型。
- 3、声明引用时，必须同时对其进行初始化。
- 4、引用声明完毕后，相当于目标变量名有两个名称，即该目标原名称和引用名，且不能再把该引用名作为其他变量名的别名。
- 5、声明一个引用，不是新定义了一个变量，它只表示该引用名是目标变量名的一个别名，它本身不是一种数据类型，因此引用本身不占存储单元，系统也不给引用分配存储单元。故：对引用求地址，就是对目标变量求地址。
- 6、不能建立数组的引用。因为数组是一个由若干个元素所组成的集合，所以无法建立一个数组的别名。

1、\* 作为指针使用： 2、\* 作为解引用操作符使用。于其定义书上并没有给出明确的定义，不过其意思是对其得到操作数（必须为指针类型）所指变量的引用。：

- 1、指针本质上讲是变量，和整型变量、字符变量等没有本质的

区别。不同之处在于指针的值表示的是内存单元的地址，变量也就是引用内存单元地址的别名。所以定义一个指针变量就是给一个内存地址起了个别名而已。

2. C/C++中的指针其实就是一个变量，和其他类型的变量是一个样子的，它是一个占用四字节的变量（32位机上），它与其他变量的不同之处就在于它的变量值是一个内存地址，指向内存的另外一个地方。reference我的理解就是一个别名，它和linux操作系统上的alias是一个样子的。再者，一个 pointer 变量可以指向 NULL，表示它不指向任何变量地址，但是 reference 必须在声明的时候就得和一个已经存在的变量相绑定，而且这种绑定不可改变。

( 原文链接

https://blog.csdn.net/ichsonx/article/details/7851477?ops\_request\_misc=  
%257B%2522request%255Fid%2522%253A%2522159067205619725247610218%2522%  
252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&re  
quest\_id=159067205619725247610218&biz\_id=0&utm\_medium=distribute.pc\_sear  
ch\_result.none-task-blog-2~all~first\_rank\_v2~rank\_v28-3-7851477.pc\_insert\_v  
2&utm\_term=%26%E6%93%8D%E4%BD%9C%E7%AC%A6%E5%92%8C\*%E7%9  
A%84%E5%8C%BA%E5%88%AB)

那么，与各种数据类型相对应的，就存在浮点数指针，整数指针，字符指针等。这几个点我们不在叙述。我们主要叙述的将会是数组的指针和函数的指针。

首先，相比于函数指针而言，数组指针更容易理解。对于一个数



组  $a[n]$ , 可以这样定义  $*p=a$ ; 这个  $a$  代表数组名。  $*p$  所指就指向了数组的第一个元素  $a[0]$ 。 那么, 如果我们要一个指针指向  $a[1]$ , 是不是要另设呢? 其实不用。 因为数组元素的存储顺序是自底而上, 逐个递增 1 的。 也就是说如果  $a[0]:0\times0000001, \rightarrow a[1]:0\times0000002$ . 以此类推。 用语句表示就是  $*(p++)$ 。 注意, 这里有个括号, 不然  $*p++$  表示  $a[0]$  的值 +1 而非地址加一指向下一个元素。

```
#include <stdio.h>
int main(){
    int a[5]={1,2,3,4,5};
    int *p=a;
    printf("%d\n",a); //数组的起始地址
    printf("%d\n",*p); //指向第一个元素
    printf("%d\n",p); //指针的地址
    printf("%d\n",*(++p)); //下一个元素
    printf("%d\n",*p++); //第二个元素加一
    printf("%d\n",*p); //由于a++的延迟, 现在才看到
}
```

说完了数组指针, 咱们再来谈谈指针数组。 指针数组的作用就相当于一个二维数组。 因为一个指针可以指向一个数组。 那么由指针组成的数组就可以构成由数组组成的数组, 也就是二维数组。

详解有篇博客比我讲的要强, 这里附上一个网址

[https://blog.csdn.net/qq\\_28114615/article/details/86434837?ops\\_request\\_misc=%26request\\_id=%26biz\\_id=102&utm\\_term=%E6%8C%87%E9%92%88%E6%95%B0%E7%BB%84&utm\\_medium=distribute.pc\\_search\\_result.none-task-blog-2~all~sobaiduweb~default-0-86434837](https://blog.csdn.net/qq_28114615/article/details/86434837?ops_request_misc=%26request_id=%26biz_id=102&utm_term=%E6%8C%87%E9%92%88%E6%95%B0%E7%BB%84&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-86434837)

然后相对而言不是那么好理解的就是函数指针与指针函数。 首先讲讲指针函数。 这篇博客可以参考一下。



[https://blog.csdn.net/u010280075/article/details/88914424?ops\\_request\\_misc=%E6%8C%87%E9%92%88%E5%87%BD%E6%95%B0&utm\\_medium=distribute.pc\\_search\\_result.none-task-blog-2~all~sobaiduweb~default-0-88914424](https://blog.csdn.net/u010280075/article/details/88914424?ops_request_misc=%E6%8C%87%E9%92%88%E5%87%BD%E6%95%B0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-88914424)

指针函数，本质上还是一个函数。但是函数的返回值是指针。

具体用法我们可以参考下面这个例子。

```
# include <stdio.h>
# include <stdlib.h>
int * func_sum(int n)
{
    if (n < 0)
    {
        printf("error:n must be > 0\n");
        exit(-1);
    }
    static int sum = 0;
    int *p = &sum;
    for (int i = 0; i < n; i++)
    {
        sum += i;
    }
    return p;
}
int main(void)
{
    int num = 0;
    printf("please input one number:");
    scanf("%d", &num);
    int *p = func_sum(num);
    printf("sum:%d\n", *p);
    return 0;
}
```

函数的指针是一个能够指向栈区函数的指针。指向代码段函数入

□ 地 址 . 用 下 例 来 说 明 问 题 :

```

#include <stdio.h>
int max(int a, int b)
{
    return a > b ? a : b;
}
int main(void)
{
    int (*p)(int, int); //函数指针的定义
    //int (*p)();        //函数指针的另一种定义方式，不过不建议使用
    //int (*p)(int a, int b); //也可以使用这种方式定义函数指针
    p = max; //函数指针初始化
    int ret = p(10, 15); //函数指针的调用
    //int ret = (*max)(10,15);
    //int ret = (*p)(10,15);
    //以上两种写法与第一种写法是等价的，不过建议使用第一种方式
    printf("max = %d \n", ret);
    return 0;
}

```

那么，为何要引入函数指针这么一个复杂概念呢？我觉得主要是在处理复杂代码问题上它可以体现优势。举个例子，如果我们要实现数组的排序，我们知道，常用的数组排序方法有很多种，比如快排，插入排序，冒泡排序，选择排序等，如果不管内部实现，你会发现，除了函数名不一样之外，返回值，包括函数入参都是相同的，这时候如果要调用不同的排序方法，就可以使用指针函数来实现，我们只需要修改函数指针初始化的地方，而不需要去修改每个调用的地方（特别是当调用特别频繁的时候）。

接着，我们介绍一下这个最常见的一种指针（严格来说是数组）——字符串。字符串就是一个字符数组。字符串的常用操作函数列出如下：

#### 1.strcpy 函数

原型：strcpy(str1, str2);

功能：将字符串 `str2` 复制到字符串 `str1` 中，并覆盖 `str1` 原始字符串，可以用来为字符串变量赋值

返回：`str1`

注意：1) 字符串 `str2` 会覆盖 `str1` 中的全部字符，2) 字符串 `str2` 的长度不能超过 `str1`

## 2.strncpy 函数

原型：`strncpy(str1, str2, n);`

功能：将字符串 `str2` 中的前 `n` 个字符复制到字符串 `str1` 的前 `n` 个字符中

返回：`str1`

注意：1) 不会清除 `str1` 中全部字符串，只会改变前 `n` 个字符串，2) `n` 不能大于字符串 `str1`、`str2` 的长度 3) 但是如果使用 `strncpy_s` 便会清除 `str1` 中的全部字符串

## 3.strcat 函数

原型：`strcat(str1, str2);`

功能：将字符串 `str2` 添加到字符串 `str1` 的尾部，也就是拼接两个字符串

原型 2：`strncat(str1, str2, n);`

功能 2：将字符串 `str2` 的前 `n` 个字符添加到字符串 `str1` 的尾部

返回：`str1`

注意：拼接之后的长度不能超过字符串数组 `str1` 的长度

## 4.strlen 函数

原型: `strlen(str1);`

功能: 计算字符串 `str1` 的长度

返回: 一个 `int` 值

注意: 字符串的长度不包括字符 `'\0'`

#### 5.strcmp 函数

原型: `strcmp(str1,str2);`

功能: 比较两个字符串, 如果两个字符串相等, 则返回 0; 若 `str1` 大于 `str2` (对于大于的理解, 是指从两个字符串的第一个字符开始比较, 若两个字符相同, 则继续比较, 若发现两个字符不相等, 且 `str1` 中该字符的 ASCII 码大于 `str2` 中的, 则表示 `str1` 大于 `str2`), 返回一个正数 (这个正数不一定是 1); 若 `str1` 小于 `str2`, 返回一个负数 (不一定是 -1); 若字符串 `str1` 的长度大于 `str2`, 且 `str2` 的字符与 `str1` 前面的字符相同, 则也相对于 `str1` 大于 `str2` 处理

原型 2: `strncmp(str1,str2,n);`

功能 2: 比较两个字符串的前 `n` 个字符

原型 3: `stricmp(str1,str2);` (在 Windows 中使用 `stricmp`, 在 Linux 中使用 `strcasecmp`)

功能 3: 忽略两个字符串中的大小写比较字符串, 也就是对大小写不敏感

#### 6.strchr 函数

原型: `strchr(str,c);`

功能: 在 `str` 字符串中查找首次出现字符 `c` 的位置 (从字符串的首地

址开始查找)

原型 2: `strrchr(str,c);`

功能 2: 在字符串 `str` 中从后向前开始查找字符 `c` 首次出现的位置

原型 3: `strstr(str1,str2);`

功能 3: 在字符串 `str1` 中查找字符串 `str2` 的位置, 若找到, 则返回 `str2`

第一个字符在 `str1` 中的位置的指针, 若没找到, 返回 `NULL`

返回: 字符 `c` 的位置的指针, 若没有查找到字符 `c`, 则返回空指针 `NULL`

习题例析:

我们这里就举几个指针方面的例子吧。

这里的例子我不敢放答案我怕做错。题目在网上应该都有解答, 可以上网搜一搜。另外, 有一本书专门讲 C 指针的可以找来看看。

1. 编写一个函数, 对于一个数组, 返回数组中不相同元素的个数
2. 编写一个函数, 从  $n$  个字符串中找出最长的串, 若有多个返回任意一个。返回值为最长串的长度。
3. 用指针试着编写一个矩阵的转置程序
4. 消除游戏是一种益智游戏。彼此相邻的相同元素配对消除。例如,

给定棋盘

变为

4 4 3 1 4

4 4 3 0 4

3 1 1 1 1

3 0 0 0 0

4 3 4 1 2

4 3 4 0 2

4 4 2 2 2

4 4 0 0 0

## 第八章. 结构体

之前我们引入了数组。我们都知道，数组是一个相同数据类型的元素们组成的集合。那么，有没有一种结构可以将不同类型的元素结合到一起呢？在C语言里，结构体就可以实现这样一个功能。

在C++里面虽然仍然保留了结构体的语法，相对结构体而言，C++更喜欢用带有函数（方法）的“结构体”——类与对象。可以说，C的结构体思想是面向对象的萌芽，是面向对象的祖先。

本章我们着重讲解以下内容：

1. 结构体的定义
2. 结构体的使用
3. 结构体表达式的求值问题

我们还会简单的介绍线性链表的创建和动态内存分配问题。

### 1. 结构体的定义

结构体的定义如下图所示，括号内的东西代表这个结构体的内部属性。包括年，月，日。

```
struct{  
    int year;  
    int month;  
    int day;  
}first_day_2000={2000,1,1};
```

### 2. 结构体的使用

结构体的引用：如果是一个结构体对象，可以用点对内部成员进行访问，就像C++的对象一样。如下：

```

struct{
    int year;
    int month;
    int day;
}first_day_2000={2000,1,1};
#include<stdio.h>
int main(){
    printf("%d\n",first_day_2000.year);
    printf("%d\n",first_day_2000.month);
    printf("%d\n",first_day_2000.day);
    return 0;
}

```

在我的记忆中还有另外一种方式

```

struct day0{
    int year;
    int month;
    int day;
};
#include<stdio.h>
int main(){
    struct day0 day1 = {2000,1,1};
    printf("%d\n",day1.year);
    printf("%d\n",day1.month);
    printf("%d\n",day1.day);
    return 0;
}

```

如果是 个 指针，那么引用就要变成箭头而非点运算。

```

struct day0{
    int year;
    int month;
    int day;
}day1={2000,1,1};
#include<stdio.h>
int main(){
    struct day0 *p;
    p=&day1;
    printf("%d\n", (p->year));
    printf("%d\n", p->month);
    printf("%d\n", p->day);
    return 0;
}

```

### 3. 习题

这次就出一个题目。计算下列表达式的值（那天被老师问这题我他妈慌得亚批嘞）

```
char u[]="UVWXYZ",v[]="xyz";
struct T{
    int x;
    char c;
    char *t;
}a[]={11,'A',u},{100,'B',v}},*p=a;
```

序号	表达式	计算值	验证值
1	(++p)->x	100	100
2	P++,p->c	B	B
3	*p++->t,*p->t	x	x
4	*(++p)->t	x	x
5	*++p->t	V	V
6	++*p->t	V	V

有关动态内存分配的我放一个链接上来，基本用法叫做

```
D=(struct a*)malloc(sizeof(struct a));
```

它使用的是我们前面没有使用过的堆内存，还记得我们在指针那一节提到了栈内存和堆内存吗，我们说栈内存计算机会自己释放，但是堆内存他并不会自己释放，这时需要程序员手动释放。malloc叫动态内存分配，可以在有限的内存内存储更多的东西。

[https://blog.csdn.net/qq\\_41071068/article/details/90741413?ops\\_request\\_misc=%E5%8A%A8%E6%80%81%E5%86%85%E5%AD%98%E5%88%86%E9%85%8D&utm\\_medium=distribute.pc\\_search\\_result.none-task-blog-2~all~sobaiduweb~default-0-90741413](https://blog.csdn.net/qq_41071068/article/details/90741413?ops_request_misc=%E5%8A%A8%E6%80%81%E5%86%85%E5%AD%98%E5%88%86%E9%85%8D&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-90741413)

接下来我展示一个创建链表的程序，具体可以在数据结构里面看。



```

#include <stdio.h>
struct list
{
    int num;
    struct list *next;
};
typedef struct list list_single;
list_single *creat_list_tail(int n)//尾插法创建一个链表，并返回一个头指针
{
    int i=0;
    list_single *head,*rear,*node;
    head=(list_single *)malloc(sizeof(list_single));
    rear=head;
    for(;i<n;i++)
    {
        node=(list_single *)malloc(sizeof(list_single));
        node->num=i+1;
        rear->next=node;//新节点插在尾巴
        rear=node;//这点需要着重理解
    }
    rear->next=NULL;
    head->num=n;
    return head;
}

```

这里我也推荐一篇博客，写的也很不错。

[https://blog.csdn.net/BShanJ/article/details/89461525?ops\\_request\\_misc=%26request\\_id=%26biz\\_id=102&utm\\_term=%E5%88%9B%E5%BB%BA%E9%93%BE%E8%A1%A8&utm\\_medium=distribute.pc\\_search\\_result.none-task-blog-2~all~sobaiduweb~default-1-89461525](https://blog.csdn.net/BShanJ/article/details/89461525?ops_request_misc=%26request_id=%26biz_id=102&utm_term=%E5%88%9B%E5%BB%BA%E9%93%BE%E8%A1%A8&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-1-89461525)

## 第九章。文件

这次我们简单介绍一下文件操作，包括这个文件读取，文件重写以及各种 I/O 函数的汇总。

### 一. 文件的各种操作

#### 文件的打开

##### 1. 函数原型

`FILE *fopen(char *pname, char *mode)`

##### 2. 功能说明

按照 `mode` 规定的方式，打开由 `pname` 指定的文件。若找不到由 `pname` 指定的相应文件，就按以下方式之一处理：

- (1) 此时如 `mode` 规定按写方式打开文件，就按由 `pname` 指定的名字建立一个新文件；
- (2) 此时如 `mode` 规定按读方式打开文件，就会产生一个错误。

打开文件的作用是：

- (1) 分配给打开文件一个 `FILE` 类型的文件结构体变量，并将有关信息填入文件结构体变量；
- (2) 开辟一个缓冲区；
- (3) 调用操作系统提供的打开文件或建立新文件功能，打开或建立指定文件；

`FILE *`：指出 `fopen` 是一个返回文件类型的指针函数；

##### 3. 参数说明

`pname`：是一个字符指针，它将指向要打开或建立的文件的文件名字符

串。

`mode`: 是一个指向文件处理方式字符串的字符指针。所有可能的文件处理方式见表 8.1

#### 4. 返回值

正常返回: 被打开文件的文件指针。

异常返回: `NULL`, 表示打开操作不成功。

例如:

//定义一个名叫 `fp` 文件指针

```
FILE *fp;
```

//判断按读方式打开一个名叫 `test` 的文件是否失败

```
if((fp=fopen("test", "r")) == NULL)//打开操作不成功
```

```
{
```

```
printf("The file can not be opened.\n");
```

```
exit(1);//结束程序的执行
```

```
}
```

要说明的是: C 语言将计算机的输入输出设备都看作是文件。例如, 键盘文件、屏幕文件等。ANSI C 标准规定, 在执行程序时系统先自动打开键盘、屏幕、错误三个文件。这三个文件的文件指针分别是: 标准输入 `stdin`、标准输出 `stdout` 和标准出错 `stderr`。

#### \* 文件的关闭

##### 1. 函数原型

```
int fclose(FILE *fp);
```

## 2. 功能说明

关闭由 `fp` 指出的文件。此时调用操作系统提供的文件关闭功能，关闭由 `fp->fd` 指出的文件；释放由 `fp` 指出的文件类型结构体变量；返回操作结果，即 0 或 `EOF`。

## 3. 参数说明

`fp`: 一个已打开文件的文件指针。

## 4. 返回值

正常返回：0。

异常返回：`EOF`，表示文件在关闭时发生错误。

例如：

```
int n=fclose(fp);
```

## \*文件的读写操作

### A. 从文件中读取一个字符

#### 1. 函数原型

```
int fgetc(FILE *fp);
```

#### 2. 功能说明

从 `fp` 所指文件中读取一个字符。

#### 3. 参数说明

`fp`: 这是个文件指针，它指出要从中读取字符的文件。

#### 4. 返回值

正常返回：返回读取字符的代码。

非正常返回：返回 `EOF`。例如，要从“写打开”文件中读取

一个字符时，会发生错误而返回一个 EOF。

## B. 写一个字符到文件中去

### 1. 函数原型

```
int fputc(int ch, FILE *fp)
```

### 2. 功能说明

把 `ch` 中的字符写入由 `fp` 指出的文件中去。

### 3. 参数说明

`ch`: 是一个整型变量，内存要写到文件中的字符（C 语言中整型和字符型可以通用）。

`fp`: 这是个文件指针，指出要在其中写入字符的文件。

### 4. 返回值

正常返回：要写入字符的代码。

非正常返回：返回 EOF。例如，要往“读打开”文件中写一个字符时，会发生错误而返回一个 EOF。

## C. 从文件中读取一个字符串

### 1. 函数原型

```
char *fgets(char *str, int n, FILE *fp)
```

### 2. 功能说明

从由 `fp` 指出的文件中读取 `n-1` 个字符，并把它们存放到由 `str` 指出的字符数组中去，最后加上一个字符串结束符 `'\0'`。

### 3. 参数说明

`str`: 接收字符串的内存地址，可以是数组名，也可以是指针。

`n`: 指出要读取字符的个数。

`fp`: 这是个文件指针, 指出要从中读取字符的文件。

#### 4. 返回值

正常返回: 返回字符串的内存首地址, 即 `str` 的值。

非正常返回: 返回一个 `NULL` 值, 此时应当用 `feof()` 或 `ferror()` 函数来判别是读取到了文件尾, 还是发生了错误。例如, 要从“写打开”文件中读取字符串, 将

发生错误而返回一个 `NULL` 值。

#### D. 写一个字符串到文件中

##### 1. 函数原型

```
int fputs(char *str, FILE *fp)
```

##### 2. 功能说明

把由 `str` 指出的字符串写入到 `fp` 所指的文件中去。

##### 3. 参数说明

`str`: 指出要写到文件中去的字符串。

`fp`: 这是个文件指针, 指出字符串要写入其中的文件。

#### 4. 返回值

正常返回: 写入文件的字符个数, 即字符串的长度。

非正常返回: 返回一个 `NULL` 值, 此时应当用 `feof()` 或 `ferror()` 函数来判别是读取到了文件尾, 还是发生了错误。例如, 要往一个“读打开”文件中写字符串时,

会发生错误而返回一个 `NULL` 值。

## E. 往文件中写格式化数据

### 1. 函数原型

```
int fprintf(FILE *fp, char *format, arg_list)
```

### 2. 功能说明

将变量表列 (arg\_list) 中的数据, 按照 format 指出的格式, 写入由 fp 指定的文件。fprintf() 函数与 printf() 函数的功能相同, 只是 printf() 函数是将数据写入屏幕文件 (stdout)。

### 3. 参数说明

fp: 这是个文件指针, 指出要将数据写入的文件。

format: 这是个指向字符串的字符指针, 字符串中含有要写出数据的格式, 所以该字符串成为格式串。格式串描述的规则与 printf() 函数中的格式串相同。

arg\_list: 是要写入文件的变量表列, 各变量之间用逗号分隔。

### 4. 返回值

无。

## G. 以二进制形式读取文件中的数据

### 1. 函数原型

```
int fread(void *buffer, unsigned size, unsigned count, FILE *fp)
```

### 2. 功能说明

从由 fp 指定的文件中, 按二进制形式将 size\*count 个数据读到由 buffer 指出的数据区中。

### 3. 参数说明

`buffer`: 这是一个 `void` 型指针, 指出要将读入数据存放在其中的存储区首地址。

`size`: 指出一个数据块的字节数, 即一个数据块的大小尺寸。

`count`: 指出一次读入多少个数据块 (`size`)。

`fp`: 这是个文件指针, 指出要从其中读出数据的文件。

#### 4. 返回值

正常返回: 实际读取数据块的个数, 即 `count`。

异常返回: 如果文件中剩下的数据块个数少于参数中 `count` 指出的个数, 或者发生了错误, 返回 0 值。此时可以用 `feof()` 和 `ferror()` 来判定到底出现了什么情况。

#### 4. 以二进制形式写数据到文件中去

##### 1. 函数原型

```
int fwrite(void *buffer, unsigned size, unsigned count, FILE *fp)
```

##### 2. 功能说明

按二进制形式, 将由 `buffer` 指定的数据缓冲区内的 `size*count` 个数据写入由 `fp` 指定的文件中。

##### 3. 参数说明

`buffer`: 这是一个 `void` 型指针, 指出要将其中数据输出到文件的缓冲区首地址。

`size`: 指出一个数据块的字节数, 即一个数据块的大小尺寸。

`count`: 一次输出多少个数据块 (`size`)。



fp: 这是个文件指针, 指出要从其中读出数据的文件。

#### 4. 返回值

正常返回: 实际输出数据块的个数, 即 `count`。

异常返回: 返回 0 值, 表示输出结束或发生了错误。

#### 1. 以二进制形式读取一个整数

##### 1. 函数原型

```
int getw(FILE *fp)
```

##### 2. 功能说明

从由 fp 指定的文件中, 以二进制形式读取一个整数。

##### 3. 参数说明

fp: 是文件指针。

##### 4. 返回值

正常返回: 所读取整数的值。

异常返回: 返回 EOF, 即 -1。由于读取的整数值有可能是 -1, 所以必须用 `feof()` 或 `ferror()` 来判断是到了文件结束, 还是出现了一个出错。

#### 文件状态检查

##### A. 文件结束

###### (1) 函数原型

```
int feof(FILE *fp)
```

###### (2) 功能说明

该函数用来判断文件是否结束。

### (3) 参数说明

fp: 文件指针。

### (4) 返回值

0: 假值, 表示文件未结束。

1: 真值, 表示文件结束。

## B. 文件读/写出错

### (1) 函数原型

```
int ferror(FILE *fp)
```

### (2) 功能说明

检查由 fp 指定的文件在读写时是否出错。

### (3) 参数说明

fp: 文件指针。

### (4) 返回值

0: 假值, 表示无错误。

1: 真值, 表示出错。

## C. 清除文件错误标志

### (1) 函数原型

```
void clearerr(FILE *fp)
```

### (2) 功能说明

清除由 fp 指定文件的错误标志。

### (3) 参数说明

fp: 文件指针。

#### (4) 返回值

无。

### D. 了解文件指针的当前位置

#### (1) 函数原型

```
long ftell(FILE *fp)
```

#### (2) 功能说明

取得由 `fp` 指定文件的当前读/写位置，该位置值用相对于文件开头的位移量来表示。

#### (3) 参数说明

`fp`: 文件指针。

#### (4) 返回值

正常返回：位移量（这是个长整数）。

异常返回：-1，表示出错。

### A. 反绕

#### (1) 函数原型

```
void rewind(FILE *fp)
```

#### (2) 功能说明

使由文件指针 `fp` 指定的文件的位置指针重新指向文件的开头位置。

#### (3) 参数说明

`fp`: 文件指针。

#### (4) 返回值

无。

### B. 随机定位

#### (1) 函数原型

```
int fseek(FILE *fp, long offset, int base)
```

#### (2) 功能说明

使文件指针 `fp` 移到基于 `base` 的相对位置 `offset` 处。

#### (3) 参数说明

`fp`: 文件指针。

`offset`: 相对 `base` 的字节位移量。这是个长整数, 用以支持大于 64KB 的文件。

`base`: 文件位置指针移动的基准位置, 是计算文件位置指针移动的基点。ANSI C 定义了 `base` 的可能取值, 以及这些取值的符号常量。

#### (4) 返回值

正常返回: 当前指针位置。

异常返回: -1, 表示定位操作出错。

### 关于 `exit()` 函数

#### 1. 函数原型

```
void exit(int status)
```

#### 2. 功能说明

`exit()` 函数使程序立即终止执行, 同时将缓冲区中剩余的数据输出并关闭所有已经打开的文件。

### 3. 参数说明

status: 为 0 值表示程序正常终止, 为非 0 值表示一个定义错误。

### 4. 返回值

无。

## \* 关于 feof() 函数

### 1. 函数原型

```
int feof(FILE *fp)
```

### 2. 功能说明

在文本文件 (ASCII 文件) 中可以用值为 -1 的符号常量 EOF 来作为文件的结束符。但是在二进制文件中 -1 往往可能是一个有意义的数  
据, 因此不能用它来作为文件的结束标志。为了能有效判别文件是否  
结束, ANSI C 提供了标准函数 feof(), 用来识别文件是否结束。

### 3. 参数说明

fp: 文件指针。

### 4. 返回值

返回为非 0 值: 已到文件尾。

返回为 0 值: 表示还未到文件尾。

## 二. 各种 I/O 语句

C还提供另外一些输入输出函数，总结如下，这里不包括有的与文件操作相关的输入输出函数（后面文章会介绍）。首先是输入函数：

函数	作用
getchar()	不带任何参数，从输入队列中返回下一个字符
gets()	读取整行输入，直至遇到换行符，然后丢弃换行符，存储其它字符，并在这些字符末尾添加一个空字符。但是它无法检查数组是否装得下输入的字符，会造成缓冲区溢出问题，因此一般不建议使用，且C11中已删除
char *fgets(char *restrict, restrict)	第1个参数指定存储的数组，第2个参数指定读入的最大字符数（会读取n-1个字符或遇到第1个换行符），第3个参数指定要读入的文件，如果从键盘读取，则以stdin（标准输入为参数）。fgets()函数会把换行符放在字符串末尾（假设输入行不溢出），正常的话，该函数会返回指向char的指针，但是如果读到文件末尾，会返回空指针。
gets_s()	该函数是C11新增的可选特性。只从标准输入中读取数据，因此不需要第3个参数，也不会存储换行符，但是，如果读到最大字符都没有读到换行符，则需要特殊处理。

比较gets()、fgets()、gets\_s()函数可知，当输入大小没有超过目标存储区大小时，3个函数都没有问题，可是如果输入行太长，fgets()处理最方便。

接下来是输出函数：

函数	作用
getchar()	输出字符，通常与getchar()配合使用，都只处理字符
puts()	输出字符串，直至遇到空字符才会停止输出。而且显示字符串时会自动在末尾添加一个换行符
fputs()	第1个参数指定存储的数组，第2个参数指明要写入数据的文件，如果打印在显示器上，用stdout（标准输出）作为该参数，它在末尾不会添加换行符。通常与fgets()搭配使用

这一节就没有思考题了，大家就自己试一试将自己电脑桌面上的文本文件打开就可以了

课程设计：做完了这个就永远不会挂科了

## 学生选修课程系统设计

假定有  $n$  门课程，每门课程有：课程编号，课程名称，课程性质(公共课、必修课、选修课)，总学时，授课学时，实验或上机学时，学分，开课学期等信息，学生可按要求(如总学分不得少于 60)自由选课。试设计一选修课程系统，使之能提供以下功能：

- 1、系统以菜单方式工作
- 2、课程信息录入功能(课程信息用文件保存)——输入
- 3、课程信息浏览功能——输出
- 4、课程信息查询功能——算法

查询方式

按学分查询

按课程性质查询

- 5、学生选修课程情况(可选项)

程序源代码

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

struct lessons

{

    char num[20];

    char name[20];

    char kind[10];

    char total_time[10];

    char teaching_time[10];

    char experimental_time[10];

    char score[5];

    char term[10];

    struct lessons *next;

}lessons[10];

struct students

{

    char name[20];

    char cname[20];

    char num[20];

}stu[10];
```



```

int main()

{

    void menu();

    void input();

    void disply();

    void search();

    void slook();

    FILE *fp1, *fp2;

    int a;           //功能选择需要的号码

    system("cls");

    system("color 8f");

    system("cls");

    struct lessons *head = NULL;

    if ((fp1 = fopen("record.txt", "r")) == NULL)

    {

        fp2 = fopen("record.txt", "w"); //如果不存在 record.txt 就创建一个

        fclose(fp2);

    }

    Loop:menu();

    printf("选择你需要操作的功能号码:\n");

    scanf("%d", &a);

    getch();

```

```

switch (a)

{

case 0:exit(0); break;

case 1:input(); break;

case 2:disply(); break;

case 3:search(); break;

case 4:slook(); break;

default:printf("Enter error!!\n");

}

printf("请按 ENTER 返回功能操作菜单\n");

getchar();

goto Loop;

system("pause");

}

void menu()//菜单函数

{

printf("欢迎使用课程选修系统! \n 将为您提供以下服务:\n");

printf("1.课程信息的录入;\n");

printf("2.课程信息浏览;\n");

printf("3.查询课程信息;\n");

printf("4.学生的选修情况查询;\n");

printf("0.退出系统;\n");

```

```

}

void input()

{

    system("cls");

    int i, j, m, n, k;

    printf("***** 欢迎使用课程信息录入功能 !
    ***** \n");

    FILE *fp1, *fp2;

    struct lessons *head = NULL;

    if ((fp1 = fopen("record.txt", "r")) == NULL)

    {

        fp2 = fopen("record.txt", "w"); //如果不存在 record.txt 就创建一个

        fclose(fp2);

    }

    printf("请问要输入几个课程信息: \n");

    scanf("%d", &m);

    printf("请问要输入几个学生选修信息, 并且每个学生选几门课: \n");

    scanf("%d\t%d", &n, &k);

    if ((fp2 = fopen("record.txt", "w")) != NULL)

    {

        for (i = 0; i < m; i++)

        {

```

```
printf("\n 请输入课程编号:\n");

scanf("%s", lessons[i].num);

printf("请输入课程名称:\n");

scanf("%s", lessons[i].name);

printf("请输入课程性质:\n");

scanf("%s", lessons[i].kind);

printf("请输入总学时:\n");

scanf("%s", lessons[i].total_time);

printf("请输入授课学时:\n");

scanf("%s", lessons[i].teaching_time);

printf("请输入实验或上机学时:\n");

scanf("%s", lessons[i].experimental_time);

printf("请输入学分:\n");

scanf("%s", lessons[i].score);

printf("请输入开课学期:\n");

scanf("%s", lessons[i].term);

printf("\n 请输入学生姓名:\n");

scanf("%s", stu[i].name);

printf("\n 请输入学生学号:\n");

scanf("%s", stu[i].num);

printf("\n 请输入每名学生分别所选的%d门课程:\n", k);

for (j = 0; j < k; j++)
```

```

        scanf("%s", stu[i].cname);

    }

}

printf("录入成功! 祝您使用愉快!! \n");

fclose(fp2);

getchar();

system("pause");

}

void disply()    //课程信息浏览

{

    int i; FILE *fp2;

    printf("***** 欢迎使用课程信息浏览功能!

***** \n");

    if ((fp2 = fopen("record.txt", "r")) == NULL)

        printf("无课程信息\n");

    else

    {

        fprintf(fp2, "课程编号\t课程名称\t课程性质\t总学时\t授课学时

\t实验或上机学时\t学分\t开课学期\n");

        for (i = 0; i < 10; i++)

            printf("\n %s\t%s\t%s\t%s\t%s\t%s\t%s\t%s \n", lessons[i].num,

lessons[i].name, lessons[i].kind, lessons[i].total_time,

```

```

        lessons[i].teaching_time, lessons[i].experimental_time, lessons[i].score,
lessons[i].term);

    }

    getchar();

    fclose(fp2);

    system("pause");

}

void search() //课程信息查询功能
{

    int i = 0, d = 0, e = 0, j = 0, k;

    struct lessons a;

    system("cls");

    printf("***** 欢迎使用课程信息查询功能 !
***** \n");

    printf("请选择查询方式:\n0.退出\n1.按学分查询\n2.按课程性质查询
\n");

    scanf("%d", &k);

    if (k >= 0 && k <= 2)

    {

        if (k == 0)

            return;

        if (k == 1)

```

```

{

printf("请输入学分:\n");

scanf("%s", a.score);

for (i = 0; i < 10; i++)

{

if (strcmp(a.score, lessons[i].score) == 0)

printf("课程编号\t课程名称\t课程性质\t总学时\t授课学时\t\n\n");

printf("实验或上机学时\t学分\t开课学期\n\n");

printf("\n %s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\n", lessons[i].num,
lessons[i].name, lessons[i].kind, lessons[i].total_time,
lessons[i].teaching_time, lessons[i].experimental_time, lessons[i].score,
lessons[i].term);

}

printf("祝您学习愉快!! \n");

}

if (k == 2)

{

printf("请输入课程性质:\n");

scanf("%s", a.kind);

for (i = 0; i < 10; i++)

{

if (strcmp(a.kind, lessons[i].kind) == 0)

```

```
printf("课程编号\t课程名称\t课程性质\t总学时\t授课学时\t  
实验或上机学时\t学分\t开课学期\n");
```

```
printf("\n %s\t%s\t%s\t%s\t%s\t%s\t%s\t%s \n", lessons[i].num,  
lessons[i].name, lessons[i].kind, lessons[i].total_time,
```

```
lessons[i].teaching_time, lessons[i].experimental_time, lessons[i].score,  
lessons[i].term);
```

```
}
```

```
printf("祝您学习愉快!! \n");
```

```
}
```

```
}
```

```
system("pause");
```

```
}
```

```
void slook()//学生选修课程信息查询功能
```

```
{
```

```
int s, i, j, k;
```

```
struct students a;
```

```
system("cls");
```

```
printf("*****欢迎使用学生课程选修情况查询功能!
```

```
***** \n");
```

```
printf("请输入每个学生需要选几门课: \n");
```

```
scanf("%d", &k);
```

```
printf("请选择课程选修查询方法:\n0.退出系统\n1.使用姓名查询\n2.
```



使用学号查询\n~);

```
scanf("%d", &s);
```

```
if (s >= 0 && s <= 2)
```

```
{
```

```
    if (s == 0)
```

```
        return;
```

```
    if (s == 1)
```

```
{
```

```
    printf("请输入学生姓名:\n~);
```

```
    scanf("%s", a.name);
```

```
    for (i = 0; i < 10; i++)
```

```
{
```

```
    if (strcmp(a.name, stu[i].name) == 0)
```

```
{
```

```
    for (j = 0; j < k; j++)
```

```
        printf("%s\t", stu[i].cname);
```

```
    }
```

```
}
```

```
    printf("祝您学习愉快! \n~);
```

```
}
```

```
if (s == 2)
```

```
{
```

```
printf("请输入学生学号:\n");

scanf("%s", a.num);

for (i = 0; i < 10; i++)

{

    if (strcmp(a.num, stu[i].num) == 0)

    {

        for (j = 0; j < k; j++)

            printf("%s\t", stu[i].cname);

    }

}

}

system("pause");

}
```