C 语言技术栈

其实说到当初为什么要学 C 语言，因为想学计算机底层的东西，然后想到了操作系统，想到了 linux。而 linux 又是用 C 语言写的，所以自然而言的就想去学 C 语言了^< ^

一、 C 语言发展史

1.1 程序语言历史和特点

C 语言是在 1972 年贝尔实验室的丹尼斯 •里奇（Dennis Ritch）和肯 •汤普森（Ken Thompson）在开发 UNIX 操作系统时基于 B 语言设计的。

1) 机器语言
   低级语言，也称为“二进制代码语言”用 0 和 1 组成，特点是计算机可以直接识别，不需要任何的翻译。
2) 汇编语言
3) 高级语言

C 语言同时具备汇编语言和高级语言的优点，不同于 C++，他是一种面向过程的语言，包括 windows, linux 在内很多的操作系统都是用 C 语言写的。
优点：
高效性，具备只有汇编语言才有的微调控制能力，可以根据情况微调程序以获得最大的运行速度和最有效的使用内存；
可移植性：许多计算机系统都可以使用 C 编译器，把 C 代码转换成计算机内部指令的程序；
强大灵活：UNIX 操作系统大部分使用 C 编写的，FORTRAN, Perl, Pascal, Python, Logo, BASIC 等许多编译器和解释器都是用 C 语言编写的，即这些语言最后都是有 C 程序生成最后的可执行程序；
每种 CPU 所能理解的指令有限（指令集）高级语言能让编译器编译成不同的 CPU 都能使用的机器语言。

目前流行的 C 语言有：Microsoft C 或称 MS C；Borland Turbo C 或称 Turbo C；AT&T C

1.2 C 语言标准

- 第一个标准是美国国家标准协会 ANSI 于 1989 年公布，称为 C89，该版本后被国际标准化组织 ISO 接收并批准，所有又可以称为 C90。由于是 ANSI 先公布的标准，所以业内人士一般使用 ANSI C。
- C99 标准，1994 年
- C11 标准，2011 年

使用 gcc 命令时，可以添加参数以使用不同的标准，如：

- gcc -std=c99 inform.c
- gcc -std=c1x inform.c
- gcc -std=c11 inform.c

1.3 HelloWorld

先用 vim 生成一个文件名.c 的文件，然后在里面写

```
#include <stdio.h>

int main(){

    printf("Hello World!");
    return 0;
}
```

然后使用命令：gcc 文件名.c [-o] 编译文件名
如果不加[-o] 编译文件名，则会生成默认的编译文件名 a.out
接着执行该文件，如果当前目录是文件的目录，则需要加“./”
C 语言的源代码文件是一个普通的文件，但是其拓展名必须是.c
函数由函数头和函数体组成;
gcc, Gun Compiler Collection，GNU 编译器套件
-o file 制定生成的文件名为 file；
在 linux 下编译的程序只能在 linux 下运行，不能在 windows 下运行

在 windows 下也可以使用 gcc 命令，前提是你把 gcc 的目录添加到环境变量中去，还要把 include 文件和 lib 文件的目录添加进去，重新打开 cmd，输入 PATH，看是否存在以上三个路径，若存在，恭喜你可以使用 gcc 的命令，亲测有效^< ^。
Windows 切换路径也可以使用 cd，切换盘符可以直接使用英文字母加冒号的方法，比如 d:
cls 是清空命令的意思；
calc 计算器
mspaint 画图板

notepad 记事本

1.4 system 函数的使用

int system(const char *command);　　//在已经运行的程序外部执行另外一个程序。用该函数时需要头文件#include<stdlib.h>，在 linux 平台里面，gcc 编译器中几乎所有的头文件都在/usr 目录下面。

如：

C 文件：

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

int main(){

    printf("Before print\n");
    system("ls");    //查看的是可执行程序.out 所在的目录
    printf("After print\n");
    return 0;
}
```

命令行：

```
[lvhongbin@MiWiFi-R3G-srv cStudy]$ ./02_systemfunctiontest.out
Before print
#01_hellowold.c#  01_helloworld  01_helloworld.c  02_systemfunctiontest.c
02_systemfunctiontest.out  a.out  helloworld
After print
[lvhongbin@MiWiFi-R3G-srv cStudy]$
```

返回值的问题：C 语言所有的库函数调用，只能保证语法一致，不能保证执行的结果一致，比如上面的程序中，被引用的程序返回值为 0，但是利用 system 调用的时候却不能保证接收到的返回值是 0；

1.5 gcc 翻译的过程

C 代码编成可执行程序经过 4 步：

1) 预处理，宏定义展开，头文件展开，条件编译等，同时将代码中的注释删除，这里并不会检查语法，gcc -E hello.c -o hello.i
2) 编译：检查语法，将预处理后的文件编译生成汇编文件，gcc -S hello.i -o hello.s；
3) 汇编：将编译文件生成目标文件（二进制文件），gcc -c hello.s -o hello.o；
   即将源代码转换为机器语言代码，并把结果放在目标代码文件中，此为目标文件，在 windows 系统中一般是.obj 文件
4) 链接：C 语言写的程序需要依赖各种库的，所以编译之后还需要把库链接到

最终的可执行程序中去，gcc hello.o -o hello.elf。
在库中包含有很多类似于 printf()函数的的指令，称为库文件；
启动代码（startup code），充当着程序和操作系统之间的接口，Windows 和 Linux 所需要的启动代码不同。
链接中的链接器就是将目标代码，系统标准启动代码和库代码三部分合并成一个文件，即可执行文件，在 windows 系统中一般是.exe 文件，在 linux 中系统中一般是.out 文件

命令：

1) gcc -E 只进行预处理；
2) gcc -S 只进行预处理和编译；
3) gcc -c 只进行预处理，编译和汇编
4) gcc -o file 指定生成的输出文件名为 file
5) ldd 可执行文件：查看可执行文件需要哪些动态库；

```
[lvhongbin@MiWiFi-R3G-srv cStudy]$ ldd 02_systemfunctiontest.out
linux-vdso.so.1 =>  (0x00007ffc3d977000)
libc.so.6 => /lib64/libc.so.6 (0x00007fa05bb2d000)
/lib64/ld-linux-x86-64.so.2 (0x000055676fdec000)
```

1.6 C 语言程序编写的特点

1) 头文件中包含了 main 函数的声明，所以可以直接使用 main 函数，不需要额外的人为声明；
2) C 语言由函数组成，有且只有一个主函数 main；
3) 程序运行从 main 开始，main 函数由系统自动调用，不需要人为调用；
4) return 0，程序结束；
5) 常用的 IDE 有：Turbo C 2.0，Visual C++ 6.0
6) 字符编码：linux 默认是 utf-8, windows 默认是 ANSI
7)

二、 算法

2.1 基本概念

一个程序是由算法，数据结构，程序设计方法以及语言工具和环境这四个方面组成，其中，算法是核心。
算法是由算法设计和算法分析组成，算法设计就是针对特定问题的解决办法，算

法分析即分析算法步骤的正确性和复杂性

算法的特性

1) 有穷性
2) 确定性
3) 可行性
4) 输入
5) 输出

衡量算法的优劣指标

1) 正确性
2) 可读性
3) 健壮性
4) 时间复杂度和空间的复杂度

三种基本结构

Bohra 和 Jacopini 提出三种基本结构：顺序结构，循环结构，选择结构

三、 数据结构

3.1 编程规范

1) 代码缩进
2) 变量常量命名规范
   常量名统一为大写格式，如果是普通变量，要以数据类型的小写字母开头，然后取实际意义的首字母大写的名称，如果是成员变量，还要加上 m_的开头。指针的话前面要以 p 开头
3) 函数命名规范
   驼峰式命名，首字母大写。
   变量命名使用名词性词组，函数命名使用动词性词组
4) 注释

3.2 关键字标识符数据类型

1) C 语言一共有 32 个关键字
   - 数据类型关键字 12 个 short int long float double char unsigned signed struct union enum void
   - 控制语句关键字 12 个：if else switch while goto return default break do for continue case
   - 存储类关键字 5 个 auto static register extern const
   - 其他关键字 3 个：sizeof（获取定义该数据类型需要分配多大的空间）

typeof volatile
printf("Print sizeof(char)=%u\n",sizeof(char));

2) 标识符其实就是函数，变量，数组等的名字
3) 数据类型：
告诉内存某个变量需要占用多少的内存。
A. 基本类型（short, int, long, char, float, double）
- 整形
  - 短整型 short，一般占 2 个字节，16 位
  - 基本整形 int，一般占 4 个字节，32 位，具体跟计算机的字节数有关
    包括符号整形（取值范围为-32768~+32767）和无符号整形（后面加 U or l，取值范围为 0~65535）
  - 长整形 long32 位（后面加 L or l）
- 字符型 char，一般占 1 个字节，8 位，用单直撇括起的一个字符，不能是字符串
- 实型
  - 单精度型 float，一般占 4 个字节，32 位，7 位有效数字
  - 双精度型 double，一般占 8 个字节，64 位，15-16 位有效数字

B. 构造类型
- 数组类型
- 结构体类型 strut
- 共同体类型 union
- 枚举类型 enum

C. 指针类型
32 位的编译器下所有类型的指针变量，char*也好，int*也好，double*也好，都占 4 个字节；
64 位的编译器下所有类型的指针变量，char*也好，int*也好，double*也好，都占 8 个字节；

D. 空类型 void

4) 进制
A. 八进制
前面添加 0，0 后数字 0~7，不能出现超过 7 的数字
B. 十进制
C. 十六进制
前面添加 0x，0x 后数字 0~9，字母 a~f/A~F
如：printf("Print [8] 17=%o\n",17); //八进制打印
printf("Print [10] 17=%d\n",17); //十进制打印
printf("Print [16] 17=%x\n",17); //十六进制打印，字母以小写输出；
printf("Print [16] 17=%X\n",17); //十六进制打印，字母以大写输出；
结果：Print [8] 17=21

Print [10] 17=17
Print [16] 17=11

5) 原码反码与补码
正数的原码反码和补码相同
为什么不用原码进行储存？
- 0 有两种存储方式，符号位一个为 0，一个为 1；
- 正数和负数相加，结果不正确（计算机只会加不会减）

符号位：最高位；

反码（为了算补码）
- 正数的原码和反码是一样的；
- 求原码；
- 其他位取反（0 为 1，1 为 0）；

为什么不用反码进行储存？
- 0 有两种存储方式，符号位一个为 0，一个为 1；

补码
正数的补码是他的原码，负数的补码是他绝对值的二进制形式按位取反后再加 1，最高位为符号位，0 代表正数，1 代表负数。相同绝对值的正数和负数想加，最高位 1 丢弃，结果为 0000 0000，此时 0 只有一种存储方式；
如果你想从补码得知它的源码的话，最开始的方式是先减 1，然后求反码。
由于正数和负数是可逆的，所以你可以重复求补码的操作，也就是说两次原码求补码的操作得到的是源码；

综上所示，站在计算机的角度，数据储存用的是补码；

6) 实型常量后面可以用 F/f 或者 L/l 来修饰，其中 F/f 表示单精度类型，L/l 表示双精度类型;
7) 字符常量 vs 字符串常量
字符常量只能是一个字符，用单引号作为分界符，字符串可以有多个字符，用双引号作为分界符，末尾系统自动添加\0，作为字符串的结束标志，这也就是“H”的长度为 2 的原因。
8) 转义字符\
- \n 换行回车
- \r 光标位置回到句首
- \t 制表位
- \a 鸣铃

9) 字符型变量
实际上是以无符号整形的 ASCII 码值储存在内存单元中，使用命令 man ascii 可以查看 ACSII 表，小写字母比大写字母大 32，小写转大写，小写字母-32；
10) 长整形 vs 基本整形
长整型是 long 固定 32 位 sizeof(long)==4

基本整形是 int 不固定，根据编译器位数来定
在 TC 这种 16 位编译器 sizeof(int)=2 也就是 16 位，像 GCC，VS 这样的 32 位编译器 sizeof(int)==4 也就是 32 位，跟 long 一样
如果是 64 位编译器，sizeof(int)==16
无论怎么样，int 操作数度是最快的，因为系统相关

11) 无符号 unsigned vs 有符号 signed 的区别
无符号最高位不是符号位，而算作为数的一部分
无符号输出用%u

12) 数值越界
char a=127+2

- 如果是有符号输出，则
  129 转换为二进制码 1000 0001，站在计算机的角度，这是补码
  补码：1000 0001
  反码：1111 1110
  原码：1111 1111 最高位是符号位，得-127
- 如果是无符号输出
  参数入栈时，不足 int 的，扩展为 int 后入栈
  比如 char(129)也即 char(-127)扩展成 int(-127)
  也就是由 10000001 扩展成 11111111 11111111 11111111 10000001
  然后你使用%u，也就是将 11111111 11111111 11111111 10000001 当成 unsigned int 来看，那它当然就是 4294967169 了
  换句话说，要维持值不变，在高位补 1 还是补 0 取决于原来的符号位，这称为符号扩展（Sign Extension）。”

3.3 变量的存储类别

1) auto 关键字
用于定义一个局部变量，意味着每次执行到定义该变量时，都会产生一个新的变量，并且对 其重新初始化；
auto int iInt = 1;
事实上，auto 可以省略不写，因为局部变量的默认存储方式为自动的；

2) static 关键字
静态变量

3) register 变量
寄存器存储变量，对于一个具备可移植性的程序来说，register 的作用并不大，不过不同于放在内存中，放在寄存器中效率更高，但是由于寄存器数量不多；

4) extern 变量
外部存储变量，相当于 public，在另外的 c 文件中可以调用本 c 文件中的变量

5) volatile 变量
防止编译器优化代码，比如跑马灯的时候
int a;

a=1;
a=2;
a=3;
a=4;
跑马灯轮流亮，但是站在编译器的角度，最终结果是 a=4，所以 a=1~3 的步骤会被编译器优化删除，此时跑马灯就跑不起来了，所以为例避免编译器作多余的优化，要加上此关键词。

3.4 类型转换

- 自动类型转换
- 强制类型转换

在变量前使用包含要转换类型的括号，如 int j = (int)i ;

3.5 输入输出屏幕与打印格式

C 语言对字符和字符串的区别非常严格，所以对于单引号和双引号的使用也是非常清晰。

1) 输出：putchar() 输出字符；
   不仅可以输出普通的字符，也可以输出转义字符，如 putchar('\101')等同于 putchar('A')，需要添加#include<stdio.h>
2) 输入：getchar() 输入字符；
   要注意的是如果输出包含有换行\n，getchar 会自动获取该换行符号

3) 输出：puts()，输出字符串，但是不能进行格式化操作；
4) 输入：gets()，输入字符串，但是不能进行格式化操作；

5) 输出：printf，输出字符串；
6) 输入：scanf(格式控制, 地址列表)；
   //阻塞，等待用户输入内容，按回车结束，输入不需要加换行的符号\n
   //编写程序时，在 scanf 函数的地址列表处，一定要使用变量的地址 &标识符，而不是变量的标识符，否则会报错。

- %hd 输入/输出 short 类型；
- %d 输入/输出 int 类型 可以使用%5d 为输出，表示一共 5 个字符，默认是右对齐；%-5d 中减号默认为右对齐；
- %l 输入/输出 long 类型
- %ll 输入/输出 long long 类型(8 位)
- %hu 输入/输出 unsigned short 类型
- %u 输入/输出 unsigned 类型

- %lu 输入/输出 unsigned long 类型
- %llu 输入/输出 unsigned long long 类型
- %c 输入/输出字符类型
- %s 输入/输出字符串类型
- %p 输入/输出指针以 16 进制的形式
- %% 输入/输出一个%

3.6 运算符与表达式

1) 算术运算符

+ - * / % ++ --

两个数相除，如果商想要获得小数，分子和分母必须是一个小数，否则结果只会取整；

2) 赋值运算符

= += -= *= /= %/

3) 比较运算符

== != < > <= >=

4) 逻辑运算符

&& || !

短路规则 ||，左边为真，右边不执行；

&&，左边为假，右边不执行；

优先级比赋值运算符要高；

5) 位运算符

(1)&:按位操作符(“与”)

将两个表达式的值按二进制位展开，对应的位（bit）按值进行”与”运算，结果保留在该位上。

eg1:17&18

对应的二进制就是 00010010

&00010011

00010010

该位只要有一个值是 0 结果就是 0，否则就是 1.

如果两数位数不同，则较短数高位补零，再运算。

不具备短路功能。

(2)按位或运算符(|)

按位或运算将两个运算分量的对应位按位遵照以下规则进行计算：

0 | 0 = 0, 0 | 1 = 1, 1 | 0 = 1, 1 | 1 = 1

即只要有 1 个是 1 的位，结果为 1，否则为 0。

例如，023 | 035 结果为 037。

按位或运算的典型用法是将一个位串信息的某几位置成 1。如将要获得最右 4 为 1，其他位与变量 j 的其他位相同，可用逻辑或运算 017|j。若要把这结果赋给变量 j，可写成：

j = 017|j

(3)按位异或运算符(^)

按位异或运算将两个运算分量的对应位按位遵照以下规则进行计算：

0 ^ 0 = 0, 0 ^ 1 = 1, 1 ^ 0 = 1, 1 ^ 1 = 0

即相应位的值相同的，结果为 0，不相同的结果为 1。

例如，013^035 结果为 026。

异或运算的意思是求两个运算分量相应位值是否相异，相异的为 1，相同的为 0。按位异或运算的典型用法是求一个位串信息的某几位信息的反。如欲求整型变量 j 的最右 4 位信息的反，用逻辑异或运算 017^j，就能求得 j 最右 4 位的信息的反,即原来为 1 的位，结果是 0,原来为 0 的位，结果是 1。

(4)按位取反运算符(~)

按位取反运算是单目运算，用来求一个位串信息按位的反，即哪些为 0 的位，结果是 1，而哪些为 1 的位，结果是 0。例如, ~7 的结果为 0xfff8。

6) 移位运算

移位运算用来将整型或字符型数据作为二进位信息串作整体移动。有两个运算符：

<< (左移) 和 >> (右移)

移位运算是双目运算，有两个运算分量,左分量为移位数据对象，右分量的值为移位位数。移位运算将左运算分量视作由二进位组成的位串信息,对其作向左或向右移位，得到新的位串信息。

移位运算符的优先级低于算术运算符，高于关系运算符，它们的结合方向是自左至右。

(1)左移运算符(<<)

左移运算将一个位串信息向左移指定的位，右端空出的位用 0 补充。例如 014<<2,结果为 060,即 48。

左移时，空出的右端用 0 补充，左端移出的位的信息就被丢弃。在二进制数运算中，在信息没有因移动而丢失的情况下，每左移 1 位相当于乘 2。如 4 << 2，结果为 16。

(2)右移运算符(>>)

右移运算将一个位串信息向右移指定的位，右端移出的位的信息被丢弃。例如 12>>2,结果为 3。与左移相反，对于小整数，每右移 1 位，相当于除以 2。在右移时，需要注意符号位问题。对无符号数据，右移时，左端空出的位用 0 补充。对于带符号的数据，如果移位前符号位为 0(正数)，则左端也是用 0 补充；如果移位前符号位为 1(负数)，则左端用 0 或用 1 补充，取决于计算机系统。对于负数右移，称用 0 补充的系统为“逻辑右移”，用 1 补充的系统为“算术右移”。以下代码能说明读者上机的系统所采用的右移方法：

7) sizeof 运算符

返回的是一个无符号整数类型，实际上该返回类型为 size_t，它使用了 typedef 来定义：

typedef unsigned int size_t;

8) 运算符优先级
优先级 1（从左往右）:【】数组下标 （） 圆括号 .成员变量 ->成员选择
优先级 2（从右往左）:单目运算符 - ～++ -- * &取地址 ！（类型）强制类型转换 sizeof 长度运算符

9) 自增和自减要慎用，不要自作聪明
在同一个表达式或者函数的参数中，对某一个变量执行自加或者自减的操作，很容易出现模凌两可的情况，这是因为编译器的执行顺序的不明确导致的，试看：

```
Int num=5;
Int a= num+num*num++;
```

有可能出现 a=5+5*5=30，也有可能出现 a=6+6*5=36 的结果
为了避免这种情况的出现，应该遵循以下两个原则：

- 如果一个变量出现在一个函数的多个参数中，不要对该变量使用递增或者递减运算符；
- 如果一个变量多次出现在一个表达式中，不要对该变量使用递增或者递减运算符；

3.7 副作用和序列点

副作用（side effect）：对数据对象或者文件的修改；
序列点（squence point）：程序执行的点，在该点上，所有的副作用都在进入下一步之前完成，分号是一个序列点，一个完整的表达式也是一个序列点，赋值运算符，递增运算符和递减运算符的副作用必须在一个序列点之前完成，这就说明了在同一个表达式中多次出现的同一个参数，C 编译器只能保证在序列点之前完成相应次数的递增或者递减，但是没办法保证他在子表达式后立即执行，这就是不要自作聪明的原因；

3.8 优先级和结合律

什么是结合律？两个运算符共享一个运算参数。
如：y=6*12+5*20;
上面的例子中两个乘号并不是结合律，优先级并没有规定说先执行那个乘号，C 语言把这个主动权交给了语言的实现者，跟据不同的硬件来决定先计算前者还是后者，但是无论采取那种方案，结果都是一样的。

四、 选择结构程序设计

4.1 选择结构程序设计

1) If

2) 三目运算符
3) switch

switch 语句检验的表达式必须是一个整形表达式，意味着可以使用函数调用和运算符；case 检验的值必须是整形常量；但是其效率没有 if 高，所以当检验的条件较少，如 3~4 个的时候建议用 if，否则用 switch。

```
switch(表达式){
    case  情况 1:
        语句块 1;
    case  情况 2:
        语句块 2;
    case  情况 3:
        语句块 3;
    default:
        语句块 4;
}
```

五、 循环控制

5.1 while(表达式)语句

其实就是相当于 if 语句，当条件满足的时候就进入循环，条件为假的时候就跳出循环。变量的初始化在循环之前

5.2 do…while

不管条件是否为真，循环体语句至少执行一次；

```
do
循环体语句
while(表达式);
```

5.3 for(循环变量赋初值;循环条件;循环变量){}

1) 若省略“循环变量赋初值”，则必须在循环体之前添加初值，且其后的分号不能省略；
2) 若省略“循环条件”，则默认表达式 2 始终为真，则循环会无终止地进行下去；
3) 若省略“循环变量”，则程序员应该设法保证程序能结束；
4) 若三个都省略，for（;;），那就相当于 while(1){循环体语句}程序将无休止地进行下去。

如：打印 ASCII 表

```
printf("Table ASCII\n");
        char cChar;
        for(cChar=65; cChar<91; cChar++){
                printf("%c%c\t", cChar, cChar+32);
                if(4 == cChar % 6){
                        printf("\n");
                }
        }
        printf("\n");
```

注意：

使用 gcc 编译代码是报出

error: 'for' loop initial declarations are only allowed in C99 mode

note: use option -std=c99 or -std=gnu99 to compile your code

错误，这是因为在 gcc 中直接在 for 循环中初始化了增量：

```
for(int i=0; i<len; i++) {
}
```

这语法在 gcc 中是错误的，必须先先定义 i 变量：

```
int i;
for(i=0;i<len;i++){

}
```

这是因为 gcc 基于 c89 标准，换成 C99 标准就可以在 for 循环内定义 i 变量了：

```
gcc src.c -std=c99 -o src
```

5.4 区别

1) for 语句功能更强，凡是用 while 循环能完成的功能，用 for 循环都能实现；
2) while 和 do while，循环变量的初始化需要在循环前完成；
3) c 语言的 break 语句只能跳出离它最近的一层循环，continue 语句结束本次循环；

5.5 goto 语句

goto 标识符；

标识符：

语句

容易出现的错误是错误 4：

label at end of compound statement（标志出现在复合语句的末尾）

GCC4 不允许行标记之后为空，加上空语句";"就可以了，如
goto here;
}
改成：
here: ;
}

5.6 随机数函数

在 C 语言中，生成一个随机数，一般利用两个函数，即 void srand (unsigned int seed) 和 int rand (void)。
srand() 函数用于生成一个随机数种子，种子的值等于参数 seed，这个参数由我们指定一个数、式子或者函数值。而 rand() 函数根据这个随机数种子进行运算生成一个[0 , RAND_MAX (int 或 unsigned int 最大值)]范围内的随机数。
在一个程序中只需运行一次 srand 函数即可，若参数 seed 是一个固定值，则同一程序中多次调用 rand 函数生成的数是随机数，但多次调用这一程序时，由于初始的随机数种子相同，所以生成的随机数列完全相同，因此一般使用系统时间函数 time(NULL)（在 time.h 文件中，返回从 1970 年 1 月 1 日 0 点到现在的秒数的值）作为参数 seed，使每次调用程序时的种子或者随机数列也具有一定的随机性。
例如给 int x 赋值为 [0 , 100) 的随机整数，先调用 srand(time(NULL))，再利用表达式 x = rand()%100 即可。
如果在一个程序在中，循环调用 srand(time(NULL)) 和 rand() 函数，由于现在计算机运行快速，两次循环的时间差不到 1 秒，使每次的随机数种子相同，从而生成的随机数也相同。即使在循环中加延时函数，由于时差不大，生成的随机数也相差不大，而且使程序运行时间大大增加。因此，一定要在循环外使用 srand 函数，一个程序中使用一次即可。
如果对随机性要求较高，要避免采用取模操作%，这是为了避免在某些情况下，某些伪随机数生成器产生的数，低位不够随机的问题，此时应采用 x = (int) (100.0 * rand() / (RAND_MAX + 1.0)) ，生成一个 [0 , 100) 的随机浮点数，再将其转换为 int 型，从而得到一个 0 ~ 99 的随机整数。

六、 数组

6.1 数组的声明和定义

1.1 类型说明符 数组标识符[常量表达式]
1.2 如：int array[5];

1.3 其实也有字符串常量的数组定义：

如：char cChar[6]={“hello”};

至于为什么是 6 而不是 5，是因为字符串“hello”中隐含了一个字符“\0”，这是系统自己自动加的，所以需要特别注意。

1.4 数组使用前一定要初始化，要不然会有很多奇怪的事情发生；

1.5 字符‘\0’跟数字 0 等价；

1.6 典型的错误：

```
Char a[]={‘a’,’b’};
Printf(“%s\n”,a);    //乱码，没有结束符’\0’;

正确的做法 1
Char a[10]={‘a’,’b’};    //初始化时自动补零
Printf(“%s\n”,a);        //正确

正确的做法 2
Char a[]={‘a’,’b’,0};    //有零
Printf(“%s\n”,a);        //正确

正确的做法 3
Char a[]={‘a’,’b’,’\0’}; //有零
Printf(“%s\n”,a);        //正确
```

6.2 选择法排序

先选择第一个数为目标值，然后从后面的数中挑选出最大（小）值，然后跟前一个数（这里先是第一个数）比较，如果挑选出的数更大（小），就进行交换；接着对第二个数为目标值，如此类推，直至完全排序。

```
/* ****************************************************************
 *         Filename: 03_SortSelect.c
 *    Description:
 *          Version: 1.0
 *          Created: 2017/01/12
 *         Revision: none
 *         Compiler: gcc
 *           Author: Lv Hongbin
 *          Company: Shanghai JiaoTong Univerity
 * ****************************************************************/

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

```
int main(){

        printf("\n/* ************************************\n");
        printf("Select Sort\n");
        int N=0;
        printf("Please input N(N<1000) numbers to sort\nN=");
        scanf("%d",&N);
        char cDirection;
        printf("Please select the direction\na:large->small;\tb:small->large;\nDirection=");
        getchar();
        cDirection=getchar();
        int i=0,j=0,b=0,k=0;
        int a[1000];
        for (i=0;i<N;i++){
                printf("No.%d:",i);
                scanf("%d",&a[i]);
        }
        for(j=0;j<N;j++){
                b=0;
                k=j;
                for(i=j+1;i<N;i++){
                        if(b<a[i]){
                                b=a[i];
                                k=i;
                        }
                }
                if(a[j]<b){
                        b=a[j];
                        a[j]=a[k];
                        a[k]=b;
                }
        }
        if(cDirection=='a'){
                printf("\nSort from large one to samll one: \n");
                for(i=0;i<N;i++){
                        printf("%d\t",a[i]);
                }
"03_SortSelect.c" 60L, 1282C
```

6.3 交换法排序

以第一个数为目标数，然后比较剩余的数，遇到最大（小）数就进行交换，然后以第二个数为目标数，如此类推；

```
/* ************************************************************
 *         Filename: 03_SortSwap.c
 *    Description:
 *          Version: 1.0
 *          Created: 2017/01/12
 *         Revision: none
 *         Compiler: gcc
 *           Author: Lv Hongbin
 *          Company: Shanghai JiaoTong Univerity
 * ************************************************************/

#include<stdio.h>
#include<stdlib.h>

int main(){

        printf("\n/* ****************************************\n");
        printf("Select Sort\n");
        int N=0;
        printf("Please input N(N<1000) numbers to sort\nN=");
        scanf("%d",&N);
        char cDirection;
        printf("Please select the direction\na:large->small;\tb:small->large;\nDirection=");
        getchar();
        cDirection=getchar();
        int i=0,j=0,b=0,k=0;
        int a[1000];
        for (i=0;i<N;i++){
                printf("No.%d:",i);
                scanf("%d",&a[i]);
        }
        for(j=0;j<N;j++){
                for(i=j+1;i<N;i++){
                        if(a[j]<a[i]){
                                b=a[i];
                                a[i]=a[j];
                                a[j]=b;
```

```
                        }
                }
        }
        if(cDirection=='a'){
                printf("\nSort from large one to samll one: \n");
                for(i=0;i<N;i++){
                        printf("%d\t",a[i]);
                }
        }else{
                printf("\nSort from samll one to large one: \n");
                for(i=0;i<N;i++){
                        printf("%d\t",a[N-i-1]);
                }
        }
"03_SortSwap.c" 54L, 1233C
```

6.4 冒泡法排序

这个很出名，就不说了。

```
/* ****************************************************************
 *         Filename: 03_SortBubble.c
 *     Description:
 *          Version: 1.0
 *          Created: 2017/01/11
 *         Revision: none
 *         Compiler: gcc
 *           Author: Lv Hongbin
 *          Company: Shanghai JiaoTong Univerity
 * ****************************************************************/

#include<stdio.h>
#include<stdlib.h>

int main(){

        printf("\n/* ****************************************\n");
        printf("Bubble Sort\n");
        int N=0;
        printf("Please input N(N<1000) numbers to sort\nN=");
        scanf("%d",&N);
        char cDirection;
        printf("Please                    select                    the
```

```
direction\na:large->small;\tb:small->large;\nDirection=");
        getchar();
        cDirection=getchar();
        int i=0,j=0,b=0;
        int a[1000];
        for (i=0;i<N;i++){
                printf("No.%d:",i);
                scanf("%d",&a[i]);
        }
        if(cDirection=='a'){
                printf("\nSort from large one to samll one: \n");
                for(j=0;j<N;j++){
                        for(i=0;i<N-1-j;i++){
                                if(a[i]>a[i+1]){
                                        b=a[i+1];
                                        a[i+1]=a[i];
                                        a[i]=b;
                                }
                        }
                        printf("%d\t",a[N-1-j]);
                }
        }else{
                printf("\nSort from samll one to large one: \n");
                for(j=0;j<N;j++){
                        for(i=0;i<N-1-j;i++){
                                if(a[i]<a[i+1]){
                                        b=a[i+1];
                                        a[i+1]=a[i];
                                        a[i]=b;
"03_SortBubble.c" 59L, 1333C
```

6.5 插入法排序

关键把插入的数做好备份，然后简单的做法是写三个循环，其中有一个循环是用来做插入式的序号调整；

```
/* ****************************************************************
 *         Filename: 03_SortInsert.c
 *      Description:
 *          Version: 1.0
 *          Created: 2017/01/11
 *         Revision: none
 *         Compiler: gcc
```

```
 *              Author: Lv Hongbin
 *             Company: Shanghai JiaoTong Univerity
 * ********************************************************************/

#include<stdio.h>
#include<stdlib.h>

int main(){

        printf("\n/* ************************************\n");
        printf("Insert Sort\n");
        int N=0;
        printf("Please input N(N<1000) numbers to sort\nN=");
        scanf("%d",&N);
        char cDirection;
        printf("Please select the direction\na:large->small;\tb:small->large;\nDirection=");
        getchar();
        cDirection=getchar();
        int i=0,j=0,k=0,tem=0;
        int a[1000];
        for (i=0;i<N;i++){
                printf("No.%d:",i);
                scanf("%d",&a[i]);
        }
        for(j=1;j<N;j++){
                tem = a[j];
                for(i=0;i<j;i++){
                        if(a[i]<tem){
                                for(k=j;k>i;k--){
                                        a[k]=a[k-1];
                                }
                                a[i]=tem;
                                break;
                        }
                }
        }
        if(cDirection=='a'){
                printf("\nSort from large one to samll one: \n");
                for(i=0;i<N;i++){
                        printf("%d\t",a[i]);
                }
```

```
        }else{
                printf("\nSort from samll one to large one: \n");
                for(i=0;i<N;i++){
"03_SortInsert.c" 57L, 1280C
```

6.6 折半法排序

这个比较难理解，后面学到数据结构和分析的时候再看把；

6.7 字符串的处理

1) 复制
   Strcpy(目标字符组组名/首地址，源字符组组名/首地址)
   相当于数组的“赋值”功能，我很好奇为什么不能直接使用
   数组 1=数组 2
   的方式，而采用这般猥琐的操作—< —，结果连源字符组的结束标记符“\0”都会复制进去。
   主要不能给野指针拷贝内容。
2) 连接
   strcat(目标字符组组名/首地址，源字符组组名/首地址)
   删去目标字符组原有的结束标记符“\0”，然后在拼接源字符组。
3) 字符串的比较
   strcmp(目标字符组组名/首地址，源字符组组名/首地址)
   按照 ASCII 码表，逐一比较两个字符串每一位的字符，以第一个出现分歧为结果，若
   字符串 1 = 字符串 2，返回值为 0；
   字符串 1 > 字符串 2，返回值为正数；
   字符串 1 < 字符串 2，返回值为负数；
4) 大小写转换
   大写：strupr(字符串/首地址)；
   小写：strlwr(字符串/首地址)；
   只比较字母
5) 获得字符串长度
   strlen(字符串)
6) 查找字符串
   strstr(str1,str2) 函数用于判断字符串 str2 是否是 str1 的子串。如果是，则该函数返回 str2 在 str1 中首次出现的地址；否则，返回 NULL。
7) char *strchr(const char* _Str,char _Val)
   char *strchr(char* _Str,char _Ch)
   头文件：#include <string.h>
   功能：查找字符串_Str 中首次出现字符_Val 的位置

说明：返回首次出现_Val 的位置的指针，返回的地址是被查找字符串指针开始的第一个与 Val 相同字符的指针，如果 Str 中不存在 Val 则返回 NULL。

8) 注意使用 strcpy/malloc/strlen 出现如下警告的时候
warning: incompatible implicit declaration of built-in function ‘strcpy’
可以通过在源文件中加入如下头文件解决

#include <string.h>
http://blog.csdn.net/razorluo/article/details/6301090
例子：

```
/*
*********************************************************************
 *         Filename: 04_StringInverse
 *      Description:
 *          Version: 1.0
 *          Created: 2017/01/12
 *         Revision: none
 *         Compiler: gcc
 *           Author: Lv Hongbin
 *          Company: Shanghai JiaoTong Univerity
 *
*********************************************************************/

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

int main(){
        printf("/* ***********************************\n");
        printf("Please input one String Array(max<1000)\n");
        printf("cChar: ");
        char cChar[1000]={}, cInverse[1000]={};
        scanf("%s",&cChar);
        int iSizeOfString = strlen(cChar);
        int i=0;
        for(i=0;i<iSizeOfString;i++){
                cInverse[i]=cChar[iSizeOfString-1-i];
        }
        printf("Result:\nlength of cChar: %d\n",iSizeOfString);
        printf("cChar: %s\n",cChar);
        printf("cInverse: %s\n",cInverse);
        printf("* ***********************************\n");
        return 0;
```

}

6.8 输出系统的日期和时间

获取系统日期：time;
获取系统时间：localtime
time 函数
返回 1970-1-1, 00:00:00 以来经过的秒数
原型: time_t time(time_t *calptr)
结果可以通过返回值，也可以通过参数得到，见实例
头文件 <time.h>
返回值:
成功：秒数，从 1970-1-1,00:00:00 可以当成整型输出或用于其它函数
失败：-1
例:

```
time_t now;
time(&now);// 等同于 now = time(NULL)
printf("now time is %d\n", now);
```

2. localtime 函数
将时间数值变换成本地时间，考虑到本地时区和夏令时标志;
原型: struct tm *localtime(const time_t * calptr);
头文件 <time.h>
返回值:
成功: struct tm *结构体, 原型如下:

```
struct tm {
  int tm_sec;          /* 秒 – 取值区间为[0,59] */
  int tm_min;          /* 分 - 取值区间为[0,59] */
  int tm_hour;         /* 时 - 取值区间为[0,23] */
  int tm_mday;         /* 一个月中的日期 - 取值区间为[1,31] */
  int tm_mon;          /* 月份（从一月开始，0 代表一月） - 取值区间为[0,11] */
  int tm_year;         /* 年份，其值等于实际年份减去 1900 */
  int tm_wday;         /* 星期 – 取值区间为[0,6]，其中 0 代表星期天，1 代表星期一 */
  int tm_yday;         /* 从每年 1 月 1 日开始的天数– 取值区间[0,365]，其中 0 代表 1 月 1 日 */
  int tm_isdst;        /* 夏令时标识符，夏令时 tm_isdst 为正；不实行夏令时 tm_isdst 为 0 */
};
```

此结构体空间由内核自动分配, 而且不要去释放它.
失败: NULL
例:

```
        time_t now ;
        struct tm *tm_now ;
        time(&now) ;
        tm_now = localtime(&now) ;
        printf("now datetime: %d-%d-%d %d:%d:%d\n",
tm_now->tm_year+1900,  tm_now->tm_mon+1,  tm_now->tm_mday,
tm_now->tm_hour, tm_now->tm_min, tm_now->tm_sec) ;
```

6.9 字符串的加密

```
/* ************************************************************
 *         Filename: 04_StringEncryption
 *      Description:
 *          Version: 1.0
 *          Created: 2017/01/12
 *         Revision: none
 *         Compiler: gcc
 *           Author: Lv Hongbin
 *          Company: Shanghai JiaoTong Univerity
 * ************************************************************/

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

int main(){
        printf("/* ************************************\n");
        printf("Please input one String Array(max<1000) to encrypt\n");
        printf("cChar: ");
        char cChar[1000]={}, cEncryption[1000]={};
        scanf("%s",&cChar);
        int iSizeOfString = strlen(cChar);
        int i=0;
        for(i=0;i<iSizeOfString;i++){
                cEncryption[i]=cChar[iSizeOfString-1-i]+i+5;
        }
        printf("Result:\nlength of cChar: %d\n",iSizeOfString);
        printf("cChar: %s\n",cChar);
        printf("cEncryption: %s\n",cEncryption);
        printf("* ************************************\n");
        return 0;
}
```

七、 函数

7.1 函数概述与定义

构成 C 程序的基本单元是函数，函数中包含程序的可执行代码
在 main 函数中调用其他函数，执行完毕后又返回 main 函数中，这些被调用的函数叫下层函数；
一个源文件由一个或者多个函数组成，一个源程序文件是一个编译单位，即以源程序为单位进行编译，而不是以函数为单位进行编译；

定义的函数包括函数头个函数体两部分：

1) 函数头
   包括返回值类型，函数名和参数表
2) 函数体
   每个 C 程序必须要有一个 main 函数，而且系统自动帮我们声明了该函数。
   程序中从来不会调用 main 函数，只会在开始和结束的时候调用。

7.2 声明与参数

1) 声明
   在程序中编写函数时，需要相对函数进行声明，然后再进行定义，
   声明的格式如下：
   返回值类型 函数名(参数表)

2) 参数
   形式参数 vs 实际参数
   形式参数就是在定义函数是括号中的参数表；
   实际参数就是在实际调用中传递的参数，该参数会被复制给相应的形式参数；
   参数可以时可变长度的数组，如

```
void creatArray(char cString[]){

}
```

7.3 调用方式

调用方式

- 函数语句的调用，直接使用；
- 函数表达式的调用，函数在表达式中出现；

- 函数参数的调用，函数作为参数；
- 嵌套调用；
- 递归调用；
  递归之所以能实现，是因为函数在每个执行过程中在栈中都有自己的形参和局部变量的副本，位于栈中较低的位置。该副本与该函数其他执行过程不发生关系；
  在递归函数中如何结束呢？一般在条件结构中添加 return 即可；

7.4 内部函数与外部函数

内部函数与外部函数
static，内部函数，在 C 语言中添加该关键字表示该函数只能在本源文件中使用，不能被外部源文件所调用。相当于 private。比较奇怪的是在 java 中 static 表示静态的意思，作用在成员变量或者成员函数中，让其在程序一开始就被创建，而且在以后程序运行的过程中，不会再被创建，直至程序结束。
extern，外部函数，跟 static 的作用相反，类似于 public。C 语言默认是 extern。

7.5 局部变量和全局变量

定义
局部变量：在一个函数内部定义的变量称为局部变量，其作用域由一对大括号所限定。局部变量的屏蔽作用 ：内层作用域中的变量将屏蔽外层域中的那个变量；
全局变量：一个变量在所有函数的外部进行声明，这就是全局变量。全局变量不属于某个函数，而属于整个源文件，如果外部文件要使用的话，需要 extern 关键字进行修饰。
全局变量的缺陷：
- 只能定义一次，但是声明可以多次；
- 定义一个全局变量，mri 与赋值（初始化），无法确定是定义，还是声明；
- 如果定义一个全局变量，同时初始化，这个肯定时定义
- 如果声明一个全局变量，建议加 extern，这样会清晰一点，函数和函数里面的变量都需要声明。当然可以自己写.h 头文件。但是为了避免头文件重复使用导致的多重定义，需要在头文件中添加：

```
#ifndef    A_H          // A_H 表示头文件名字 a.h 的大写
#define    A_H

//下面定义全局变量

#endif
```

7.6 库函数

为了方便用户调用，编译系统通常会提供一些库函数，如
需要添加#include<math.h>头文件：

- abs()
- labs() 求长整数的绝对值
- fabs()，求浮点数的绝对值
- sin()
- cos()
- tan()

需要添加#include<ctype.h>头文件：

- isalpha(), 检测字母
- isdigit(), 检测数字
- isalnum(), 检测数字

上述函数如果是真，则返回非零值，若是否，则返回零值。

7.7 main 函数

main 函数其实有两个参数 int main(int argc, char *argv[])，分别是整型的 argc 和不定长度的字符串数组指针的*argv[]，在程序调用的时候在./05_MainFunctionWithParameters.out 后面添加字符串，以空格隔开作为每一个元素，不过零元素默认为是文件名./05_MainFunctionWithParameters.out，其实就相当于 linux 中的命令 ls -al ./Desktop 等形式，第一个是命令，后面的就是参数和内容。

```
    /*
**************************************************************
     *         Filename: 05_MainFunctionWithParameters.c
     *      Description:
     *          Version: 1.0
     *          Created: 2017/01/13
     *         Revision: none
     *         Compiler: gcc
     *           Author: Lv Hongbin
     *          Company: Shanghai JiaoTong Univerity
     *
**************************************************************/

    #include<stdio.h>
    #include<stdlib.h>

    int main(int argc,char *argv[]){
```

```
        printf("\n/* ************************************\n");
        printf("MainFunctionWithParameters\n");
        int i;
        for (i=0;i<argc;i++){
                printf("NO.%d String: %s\t",i,argv[i]);
        }
        printf("\n * ************************************\n");
        return 0;
}
```

结果：

[root@MiWiFi-R3G-srv /home/lvhongbin/Desktop/cStudy]# gcc 05_MainFunctionWithParameters.c -o 05_MainFunctionWithParameters.out && ./05_MainFunctionWithParameters.out lvhongbin lvhongchao

/* ************************************
MainFunctionWithParameters
NO.0 String: ./05_MainFunctionWithParameters.out NO.1 String: lvhongbin NO.2 String: lvhongchao
 * ************************************

八、 指针

8.1 内存

1) 存储地址空间：对存储器编码的范围，我们常说的内存是指这一层含义。
一般分为两个过程：
   - 编码：对每个物理存储单元（1 个字节）分配一个号码；
   - 寻址：可以根据分配的号码找到相应的存储单元，完成数据的读写

2) 内存地址：
将内存抽象成一个很大的一维字符数组
编码就是对内存的每一个字节分配一个 32 位或者 64 位的编号（与 32 位或者 64 位的处理器有关）
这个内存的编号就是我们称之为内存地址。

3) 内存中每一个数据都会分配相应的地址；
   - char：占一个字节分配一个地址；
   - int：占四个字节分配四个地址；
   - float，struct，函数，数组等；

8.2 指针变量

指针即地址，也即内存编号，指针和指针变量是不一样的，指针变量是用来专门用来存放另一个变量的地址
如果在程序中定义了一个变量，在对程序进行编译或者运行的时候，系统会给这个变量分配内存地址。
指针也是一种数据类型，一般在基本数据类型后面加上星号*。

1) 指针变量的一般形式
   类型说明 *变量名
   其中*表示该变量是一个指针变量，变量名即为定义的指针变量名，其实就是在声明的时候有用，告诉别人我*后面的变量是指针变量啦。类型说明表示本指针所指向的变量的数据类型。
2) 指针变量的赋值
   &地址运算符，表示变量的地址
   &变量名
   如：先定义指针变量的同时赋值

```
int a;
int *p=&a;
```

   先定义指针变量之后再赋值

```
int a;
int *p;
p=&a;
```

   注意：不能直接把一个数字赋值给一个指针变量；
3) 指针变量的引用
   *指针变量
   作用是引用指针变量所指向的值。
4) 运算符
   & 取地址运算符 返回操作地址的单目运算符；
   * 指针运算符，作用是返回指定地址内变量的值；
   &和*的优先级是相同的，按自右向左的方向结合；
   如&*p，意思是引用 p 地址所在的变量，然后再取该变量的地址，实际上也还是 p 本身。
5) 指针的自加自减运算
   指针的自加自减表示在原来地址的基础上加上或者减去其基本数据类型的容量，比如 int 类型占 4 个字节，那么指针变量自加就加 4，如果是短整型则加 2。

8.3 数组与指针

当定义一个数组的时候，就会给他分配一块内存，其中
数组的首地址=数组名=数组的第一个元素地址

所以一下语句是等效的
语句 1：int *p, a[10];
p = a;
语句 2：int *p, a[10];
p = &a[0];

1) 如果是一个一维数组
其中*(p+n)等效于 a[n]。

2) 如果数组是一个二维数组
a[0]+n 表示第 0 行第 n 个元素的地址。
&a[n]，表示第 n 行的首地址
3) 数组后面带的维数小于总维数表示表示引用某几维的变量，在此基础上再加上数字，则表示指针。如 int aa[2][3][4]; aa,aa[2],aa[2][3]表示变量，aa+1,aa[2]+2等表示地址，aa[1][2][3]等效为*(*(*(aa+1)+2)+3)
4) 即 aa[n]等效于 aa+n，*(aa[n]) = *(aa+n)
P 等效于 p+0 等效于 a 等效于 a+0 等效于&p[0] 等效于&a[0]
p 等效于(p+0)等效于 p[0]，
但是 p[n]=100，就会出现野指针的问题，或者叫越界，因为数组没有定义到 n，出现野指针。

例子：输入三维数组并输出的某一个 table

```
    /* ****************************************************************
 *         Filename: 04_PointArrayOutput
 *      Description:
 *          Version: 1.0
 *          Created: 2017/01/12
 *         Revision: none
 *         Compiler: gcc
 *           Author: Lv Hongbin
 *          Company: Shanghai JiaoTong Univerity
 * ****************************************************************/

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

int main(){
        printf("/* ************************************\n");
        printf("Please input the dimenson of a[i][j][k](i,j,k<20))\n");
        char cStringArray[20][20][20]={};
        int i,j,k,iDimension[3]={};
```

```
        for(i=0;i<3;i++){
                printf("cStringArray %dD number: ",i+1);
                scanf("%d",iDimension+i);
        }
        printf("Please input one String Array(a[i][j][k])\n");
        for(i=0;i<iDimension[0];i++){
                for(j=0;j<iDimension[1];j++){
                        for(k=0;k<iDimension[2];k++){
                                printf("ijk=%d%d%d: ",i,j,k);
                                scanf("%d",*(*(cStringArray+i)+j)+k);
                        }
                }
        }
        printf("Please select which table to output: ");
        scanf("%d",&i);
        printf("Table%d\n",i);
        for(j=0;j<iDimension[1];j++){
                for(k=0;k<iDimension[2];k++){
                        printf("%d\t",*(*(cStringArray[i-1]+j)+k));
                }
                putchar('\n');
        }
        printf("* ************************************\n");
        return 0;
}

"04_PointArrayOutput.c" 46L, 1230C
```

8.4 字符指针与字符串常量 vs 字符数组

1) 字符数组与字符指针处理字符串的不同

前者字符数组，相当于把字符串中一个一个的字符赋给数组：

```
char cChar[]=”hello”;
int i=0;
for(i=0; i<strlen(cChar);i++){
    printf("%c\n", cChar[i]);
}
```

后者字符指针：char *cString="hello";

```
printf("%s\n",cString);
```

后者为什么给一个字符串的首地址就能打印呢？原因是编译器对%s 做了处理：

```
int i=0;
```

```
while(*(cstring+i)!='\0'){
    printf("%c", (*(cstring+i));
    i++;
}
```

%s：从首元素开始打印，直至结束符为止；

2) 字符串常量

每个字符串常量都是首字母的地址

程序中某处的“hello”和另外一处的“hello”所指向的内存地址是一样的

```
Printf(“%s\n”,”hello”+1);      //将会跳过首字母 h，得到的结果是 ello。
Printf(“%c\n”, *(”hello”));    //将会得到首字母 h；
```

3) 数组和字符串内存空间的差异

内存分了两个储存空间，分别是储存变量的栈区和储存字符串常量的文字常量区。文字常量区只读不能写，所以以下的情况是不允许的：

情况 1：

```
char *p=”hello”;
*p=’a’;  //error，因为字符串常量不能改
```

情况 2：

```
char *p=”hello”;
strcop(p,”hi”); //error，字符串常量不允许修改
```

全局数据区和栈区的字符串（也包括其他数据）有读取和写入的权限，而常量区的字符串（也包括其他数据）只有读取权限，没有写入权限。

内存权限的不同导致的一个明显结果就是，后者字符数组在定义后可以读取和修改每个字符，而对于前者形式的字符串，一旦被定义后就只能读取不能修改，任何对它的赋值都是错误的。

4) p++

p=p+1; //相当于跳过了一个字符

8.5 野指针

定义：这个指针保存了一个没有意义的（非法）地址

如：

```
int * p;
P=0x1234;
printf(“p=%d\n”,p);
*p=100;     //这是不允许的
```

上面的问题是 0x1234 的地址是不合法的，因为只有定义了的变量后自动分配的内存地址才是合法的，要不然谁都可以定义内存地址，内存就会乱了。

不合法的内存不能赋值；

操作野指针变量本身没有任何问题，操作野指针所指向的内存才有问题。

所以*p=a 是没有问题的，但是*p=&a 就会出现野指针的问题。

也就是说野指针变量可以赋值，但是其内容不能赋值；

为了避免指针的随意使用，需要用到空指针的意义

```
int * p=NULL;        //在 C 语言中 NULL 相当于 0，
int a=100;           //这样给指针变量赋值就相当于野地址
p=&a;
if(p!=null){
    printf("%d\n",*p);
}
```

8.6 多级指针 vs 万能指针 vs 常指针 vs 指针步长

1) 多级指针

```
int a= 10;
int *p=&a;
int **q=&p;
int ***t=&q;
int ****m=&t;

printf("*m =q 的地址=%d\n**m =p 的地址=%d\n", **m, )
```

2) 万能指针

```
void *p;          //万能指针
int a =2;
*p=&a;
*((int *)p=4;          //需要强制类型转换(int *)
```

3) 常指针

- 修饰符 const 修饰*的，写作

  数据类型 const * p = &a; 或者

  const 数据类型 *p = &a

  作用是修改指针变量所指向的内存权限为只读，不能写入，而且不能进行比较，也不能把 const 的地址付给其他普通指针变量让他们去改变，但是可以把 const 地址所指向的内容赋给其他普通变量让他们去改变；

  如 int * a=2;

  Int const *p=&a;

  *p=4; //这是不允许的

  p=NULL;

  但是，这只是说不能通过修饰 const 的变量去改变，但是可以把地址所在的内容付给其他变量，然后通过其他变量去改变，比如：

  char cChar[]="lvhongbin";

  const char *p=cChar;

  p[1]='n'; //错误，不能通过 p 去改变；

但是可以这样子做：

```
char *q=cChar;
const char *p= q;
char *m=*p;
```

m[1]= 'n'; //正确，虽然不能通过 p 去改变，也不能通过 p 传址，但是可以通过*p 传值，而该值恰好是 cChar[]数组的首地址，这样就可以改变数组了。这种情况常应用于具有常形参的函数。

- 修饰符 const 修饰指针变量的，写作

  数据类型 * const p=&a;

  则表示指针变量所指向的地址是只读的，不能覆写；

  也就是说作为函数形参的常指针，你不可以通过复制形参的常指针来改变原来地址所指向的内容，但是你可以通过拷贝原来地址所指向的内容来进行一些操作，这也为了避免你在调用的函数中破坏原来的内容。

4) 指针步长

指针的步长是由其指向的数据类型决定的；

p 指向的是变量的首地址，其终点是由其指向的数据类型决定

如 int *p，p 指向 0x1234，则 p+1 指向 0x1238，p 的终点到 0x1237。

8.7 数组的 sizeof

数组的 sizeof 值等于数组所占用的内存字节数，如：

```
char a1[] = "abc";
int a2[3];
sizeof( a1 ); // 结果为 4，字符 末尾还存在一个 NULL 终止符
sizeof( a2 ); // 结果为 3*4=12（依赖于 int）
```

一些朋友刚开始时把 sizeof 当作了求数组元素的个数，现在，你应该知道这是不对的，

那么应该怎么求数组元素的个数呢 Easy，通常有下面两种写法：

```
int c1 = sizeof( a1 ) / sizeof( char ); // 总长度/单个元素的长度
int c2 = sizeof( a1 ) / sizeof( a1[0] ); // 总长度/第一个元素的长度
```

写到这里，提一问，下面的 c3，c4 值应该是多少呢

```
void foo3(char a3[3]){
    int c3 = sizeof( a3 ); // c3 ==
}
void foo4(char a4[]){
    int c4 = sizeof( a4 ); // c4 ==
}
```

也许当你试图回答 c4 的值时已经意识到 c3 答错了，是的，c3!=3。这里函数参数 a3 已不

再是数组类型，而是蜕变成指针，相当于 char* a3，为什么仔细想想就不难

明白，我
们调用函数 foo1 时，程序会在栈上分配一个大小为 3 的数组吗不会！数组是“传址”的
，调用者只需将实参的地址传递过去，所以 a3 自然为指针类型（char*），c3 的值也就为 4。

8.8 指针数组和二级指针 vs 字符数组指针的区别

```
/* ****************************************************************
 *       Filename: 05_PointArray
 *    Description:
 *        Version: 1.0
 *        Created: 2017/01/13
 *       Revision: none
 *       Compiler: gcc
 *         Author: Lv Hongbin
 *        Company: Shanghai JiaoTong Univerity
 * ****************************************************************/

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

int main(){
        printf("/* ****** Point array practice ********\n");
        printf("Please input one array\n");
        printf("cChar: ");
        char cChar[1000]={}, *p[1000]={};
        scanf("%s",&cChar);
        int iSizeOfString = strlen(cChar);
        int i=0;
        printf("Result:\nlength of cChar: %d\n",iSizeOfString);
        printf("cChar: %s\n",cChar);
        printf("point array address are:\n");
        for(i=0;i<iSizeOfString;i++){
                p[i]=cChar+i;
                printf("%p\t",p[i]);
        }
        putchar('\n');
        printf("Output the point array by *p[i]\n");
        for(i=0;i<iSizeOfString;i++){
                printf("%c\t",*p[i]);
```

```
        }
        putchar('\n');
        printf("Output the point array by *(*(p+i))\n");
        for(i=0;i<iSizeOfString;i++){
                printf("%c\t",*(*(p+i)));
        }
        putchar('\n');
        printf("* ************************************\n");
        return 0;
}
```

二级指针和字符数组指针的区别
Char **cChar 和 char *cChar[]的区别
后者可以使用 char *cChar[]={}；而前者不可以
但是在作为函数形参的时候两者可以互换；

九、 指针与函数

9.1 值传递 vs 指针传递

1) 什么叫值传递？
在函数调用过程中，主调用函数和被调用函数之间有一个数值传递的过程；
但是值传递是单向的调用函数里改变形参的值不会影响到实参，比如新建一个swap()函数用于交换两个数，形参交换了，但是不会影响到实参，因为在函数调用的时候，实参把值赋给形参而已，形参在函数调用后内存便会自动释放。

2) 什么叫指针传递？
指针传递也是值传递的一种，虽然调用函数不能改变实参指针变量的值，但是可以改变实参指针变量所指变量的值。
为了解决调用函数参数无法传递的矛盾，所以用指针传递比较方便，而且能减少值传递带来的开销，因为不用把地址里面的所有内容拷贝一份。

例子：用指针实现冒泡排序

```
/* ****************************************************************
 *         Filename: 05_PointFunctionSortBubble.c
 *      Description:
 *          Version: 1.0
 *          Created: 2017/01/13
 *         Revision: none
 *         Compiler: gcc
 *           Author: Lv Hongbin
```

```
 *            Company: Shanghai JiaoTong Univerity
 * ************************************************************/

#include<stdio.h>
#include<stdlib.h>

void swap(int *pa,int *pb);
void sortBubble(int *p,int N);
int main(){

        printf("\n/* ************************************\n");
        printf("PointFunctionSortBubble\n");
        int N=0;
        printf("Please input N(N<1000) numbers to sort\nN=");
        scanf("%d",&N);
        char cDirection;
        printf("Please select the
direction\na:large->small;\tb:small->large;\nDirection=");
        getchar();
        cDirection=getchar();
        int i=0,j=0,b=0;
        int a[1000];
        for (i=0;i<N;i++){
                printf("No.%d:",i);
                scanf("%d",&a[i]);
        }
        if(cDirection=='a'){
                printf("\nSort from large one to samll one: \n");
                sortBubble(a, N);
                for(i=0;i<N;i++){
                        printf("%d\t",a[N-1-i]);
                }
        }else{
                printf("\nSort from samll one to large one: \n");
                sortBubble(a, N);
                for(i=0;i<N;i++){
                        printf("%d\t",a[i]);
                }
        }
        printf("\n* ************************************/\n");
        return 0;
}
```

```
void sortBubble(int *p,int N){
        int i,j;
        for(j=0;j<N;j++){
                for(i=0;i<N-1-j;i++){
                        if(*(p+i)>*(p+i+1)){
                                swap(p+i,p+i+1);
                        }
                }
        }

}


void swap(int *pa,int *pb){

        int tem;
        tem = *pa;
        *pa=*pb;
        *pb=tem;
}
```

3) 二维数组不是用二级指针的

9.2 指针函数

1) 返回值是指针的函数叫指针函数
2) 返回局部变量指针

如：
```
int * func(){
    int a=0;
    return &a;
}

int main(){
   int *p;
   p=func();
   printf("%d\n",*p)
}
```

这是不允许的，因为函数在调用完毕后内存就会释放，传递出来的内存地址已经没用，内容也很有可能被改掉。

3) 返回全局变量指针

虽然局部变量地址不能传递，但是全局变量地址可以

9.3 函数指针

函数指针常用作另一个函数的参数，告诉该函数要使用那个函数。
函数也有地址，因为函数的机器语言实现由载入内存的代码组成，指向函数的指针中储存这函数代码的起始处地址。
声明一个函数的时候，需要声明一个函数的类型，即包括函数类型和形参类型。如：

```
void ToUpper(char *);
void ToLower(char *);
void (*pf)(char *);    //因为圆括号的优先级比*的优先级要高
char arr[]=”hello”;
pf=ToUpper;
(*pf) (arr);
pf=ToLower;
(*pf) (arr);

作为函数参数使用的时候
Void show(void (*pf)(char *), char *str){

        *pf(str);
}
```

十、 内存分区的介绍

10.1内存分区及介绍

在程序没有执行前，有几个内存分区已经确定，虽然分区确定，但是程序只有运行时才会加载内存。

```
[root@MiWiFi-R3G-srv                    /home/lvhongbin/Desktop/cStudy]#
size ./05_TwoHeadBlockModel.out
   text    data     bss     dec     hex filename
   2015     572       4    2591     a1f
 ./05_TwoHeadBlockModel.out
```

一下分区按照地址从高到低排序：

- 栈区（stack）：向下增长，存放函数的参数值，返回值和的局部变量，自动管理内存，先进后出的特点，一般分配的内存空间比较小，在 linux 中可以使用 ulimit -a 进行查看；
- 堆区（heap）：向上增长，用于动态内存分配，手动申请空间，手动释放，整个程序结束，系统也会自动回收；

- 未初始化数据区（BSS）：用零初始化，位置可以分开，也可以仅靠数据段，存储于数据段的数据（全局未初始化，静态未初始化数据）的生命周期为整个程序运行过程。
- 数据区（Data）：初始化的数据，全局变量，static 变量，文字常量区（只读）
- 代码区：加载的是可执行文件代码段，所有的可执行代码都会加载到代码区，这块内存是不可以在运行期间修改的。

储存类型总结

类型	作用域	生命周期	存储位置
auto 变量	一对{}内	当前函数	栈区
static 局部变量	一对{}内	整个程序运行期	初始化在 data 区，未初始化在 BSS 区
extern 变量	整个程序	整个程序运行期	初始化在 data 区，未初始化在 BSS 区
static 全局变量	当前文件	整个程序运行期	初始化在 data 区，未初始化在 BSS 区
extern 函数	整个程序	整个程序运行期	代码区
static 函数	当前文件	整个程序运行期	代码区
register 变量	一对{}内	当前函数	运行时存储在 CPU 寄存器
字符串常量	当前文件	整个程序运行期	Data 段

10.2栈越界和堆空间溢出说明

1) 栈越界

比如你在函数内部定义了局部变量数组：int a[100000000000]={0};

此时由于栈区空间有限，会报出栈区内存不足的错误；

如：

```
[root@MiWiFi-R3G-srv /home/lvhongbin/Desktop/cStudy]# ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 7168
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time                (seconds, -t) unlimited
max user processes              (-u) 7168
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

2) 堆空间溢出

虽然堆区比栈区大很多，但是还是有可能出现堆空间溢出的情况；

```
int *p = (int *)malloc(10000000000 * sizeof(int));
if(p == Null)
{
    Printf("分配失败\n");
}
```

10.3内存操作函数

头文件：#include<string.h>

- memset(地址, 字符, 内存大小 sizeof()) 主要用于初始化数组或者指针指向的数据的，最终以字符的形式存储。

  void *memset(void *s, int ch, size_t n);

  函数解释：将 s 中当前位置后面的 n 个字节 （typedef unsigned int size_t ）用 ch 替换并返回 s 。

  memset：作用是在一段内存块中填充某个给定的值，它是对较大的结构体或数组进行清零操作的一种最快方法

- memcpy()

  void *memcpy(void*dest, const void *src, size_t n);

  功能

  由 src 指向地址为起始地址的连续 n 个字节的数据复制到以 destin 指向地址为起始地址的空间内。

  返回值

  函数返回一个指向 dest 的指针。

  说明

  1.source 和 destin 所指内存区域不能重叠，函数返回指向 destin 的指针。

  2.与 strcpy 相比，memcpy 并不是遇到'\0'就结束，而是一定会拷贝完 n 个字节。

  注意：

  最好不要出现内存重叠的情况

  如：memcpy(&a[2], a, 5*sizeof(int)) //出现内存重叠的情况；

- memmove()

  如果出现内存重叠的情况，建议用 memmove()；

  但是当目标区域与源区域没有重叠则和 memcpy 函数功能相同。

  如：

```
            /*
************************************************************
     *          Filename: 06_MemoryOperationFunction.c
     *    Description:
     *           Version: 1.0
     *           Created: 2017/01/14
     *          Revision: none
     *          Compiler: gcc
```

```
 *              Author: Lv Hongbin
 *             Company: Shanghai JiaoTong Univerity
 * ********************************************************/

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

int main(){

        printf("\n/* ************************************\n");
        printf("MemoryOperationFunction\n");
        int i=0,M=0,m=0,n=0,iInt[20],iMemsetInt[20];
        char cChar,cMemsetChar[20],*p;
        p=cMemsetChar;
        printf("cMemsetChar=");
        scanf("%s",p);
        getchar();
        printf("Please input byte to memset: cChar=");
        scanf("%c",&cChar);
        getchar();
        printf("How many byte do you want to memset: M=");
        scanf("%d",&M);
        getchar();
        printf("Origin:%s\n",cMemsetChar);
        memset(cMemsetChar,cChar,M*sizeof(char));
        printf("Result:%s\n",cMemsetChar);
        printf("Please input the position of string you want to    memmove : ");
        scanf("%d",&m);
        getchar();
        printf("Please input the position you want to    start memmove : ");
        scanf("%d",&n);
        memmove(cMemsetChar+n-1,cMemsetChar+m-
1,(strlen(cMemsetChar)+1-m)*sizeof(char));
        printf("Result:\n%s",cMemsetChar);
        printf("\n* ************************************/\n");
        return 0;
}
```

结果：

```
        [root@MiWiFi-R3G-srv
/home/lvhongbin/Desktop/cStudy]#./06_MemoryOperationFunction.out
```

```
/* ************************************
MemoryOperationFunction
cMemsetChar=hello
Please input byte to memset: cChar=a
How many byte do you want to memset: M=2
Origin:hello
Result:aallo
Please input the position of string you want to    memmove : 4
Please input the position you want to    start memmove : 1
Result:
lollo
* ************************************/
```

- memcmp()

int memcmp(const void *buf1, const void *buf2, unsigned int count);

功能编辑

比较内存区域 buf1 和 buf2 的前 count 个字节。

所需头文件编辑

#include <string.h>或#include<memory.h>

返回值编辑

当 buf1<buf2 时，返回值小于 0

当 buf1==buf2 时，返回值=0

当 buf1>buf2 时，返回值大于 0

说明编辑

该函数是按字节比较的。

例如：

s1,s2 为字符串时候 memcmp(s1,s2,1)就是比较 s1 和 s2 的第一个字节的 ascII 码值；

memcmp(s1,s2,n)就是比较 s1 和 s2 的前 n 个字节的 ascii 码值；

如:char *s1="abc";

char *s2="acd";

int r=memcmp(s1,s2,3);

就是比较 s1 和 s2 的前 3 个字节，第一个字节相等，第二个字节比较中大小已经确定，不必继续比较第三字节了。所以 r=-1

10.4堆区内存申请函数

(数据类型 *)malloc(sizeof());

- 要使用头文件#include<stdlib.h>
- 参数是指定堆区分配多大的空间；
- 返回值：成功就是堆区首元素地址；
- 失败返回 null;
- 使用时需要强转你需要的类型

- 动态分配的空间，如果程序没有结束，不会自动释放
- 一般使用完，需要人为释放 free(p);
- free(p)不是释放 p 变量，而是释放 p 所指向的内存
- 同一块堆区内存只能释放一次；
- 所谓释放不是指内存消失，指这块内存用户不能再使用，如果再用就是使用野指针了，虽然可能检测运行时没有问题，该地址内的内容并没有被清空，但是也不能这样去做；
- 使用后感觉使用给首地址就好了，~~分配的空间不够他都会自动拓展~~，包括 strcpy()函数也是，也不会出错，这是编译器的问题，gcc 编译器没有检测出来，但是这是错的，不能这样做

如：

```
        /*
************************************************************
**
        *          Filename: 06_HeapOperationFunction.c
        *     Description:
        *           Version: 1.0
        *           Created: 2017/01/14
        *          Revision: none
        *          Compiler: gcc
        *            Author: Lv Hongbin
        *           Company: Shanghai JiaoTong Univerity
        *
************************************************************
*/

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

      int main(){

                printf("\n/*
************************************\n");
                printf("MemoryOperationFunction\n");
                int i=0,M=0,m=0,n=0,N=0;
                char cChar,*p;
                printf("Please input number of char\nN=");
                scanf("%d",&N);
                getchar();
                p=(char *)malloc((N+1)*sizeof(char));
```

```
        printf("cMemsetChar=");
        scanf("%s",p);
        getchar();
        printf("Please input byte to memset: cChar=");
        scanf("%c",&cChar);
        getchar();
        printf("How many byte do you want to memset: M=");
        scanf("%d",&M);
        getchar();
        printf("Origin:%s\n",p);
        memset(p,cChar,M*sizeof(char));
        printf("Result:%s\n",p);
        printf("Please input the position of string you want to
memmove : ");
        scanf("%d",&m);
        getchar();
        printf("Please input the position you want to    start memmove :
");
        scanf("%d",&n);
        memmove(p+n-1,p+m-1,(strlen(p)+1-m)*sizeof(char));
        printf("Result:\n%s",p);
        printf("\n*
**************************************/\n");
        free(p);
        return 0;
}
```

结果：

```
    [root@MiWiFi-R3G-srv
/home/lvhongbin/Desktop/cStudy]#./06_HeapOperationFunction.out

    /* ************************************
    MemoryOperationFunction
    Please input number of char
    N=5
    cMemsetChar=lvhongbin
    Please input byte to memset: cChar=a
    How many byte do you want to memset: M=2
    Origin:lvhongbin
    Result:aahongbin
    Please input the position of string you want to    memmove : 8
    Please input the position you want to    start memmove : 1
```

Result:
inhongbin
* ************************************/

内存泄漏：动态分配内存不释放；
内存污染：非法操作内存；野指针

十一、 字符串和字符串函数

11.1字符串的定义

字符串定义的方法有三种：

- 字符串字面量（string literal）或者叫字符串常量（string constant）
  - 从 ASCI 标准开始，如果字符串字面量之间没有间隔，或者用空白字符分割，C 将其串联起来。
    如：Char geeting[100]=“hello,” “boys” “ how are you?”
    等价于 Char geeting[100]=“hello, boys how are you?”
    ^-^ 你猜猜这条语句输出的是什么：printf(“%c\n”, *”lvhongbin”);
    答案是“l”,因为取字符串首地址然后再取其首地址的内容,那当然是“l”喽^< ^
  - 字符串常量属于静态储存类别（static storage class），只会被储存一次，在程序整个生命周期中存在。
  - 建议在定义字符串字面量的时候加上 const 限定符，避免不小心修改了常量，访问内存错误，导致程序中断。
  - 总之，初始化数组把静态存储区的字符串拷贝到数组中，而初始化指针只把字符串的地址拷贝到指针。
- char 数组
  数组名是地址常量，不能使用自加或者自减。
  字符串储存在静态储存区（static memory），但是程序在开始运行的时候才会为该数组分配内存，此时，才将字符串拷贝到数组中。此时字符串有两个副本，一个是静态内存中的字符串字面量，另一个是储存在 ar1 数组中的字符串。所以使用数组的效率比用指针低得多。
- 指向 char 类型的指针
  数组和指针相同的地方：都可以使用数组表示法和指针加减法操作
  数组和指针不同的地方：只有指针可以使用自加或者自减的操作，因为赋值运算符左侧必须是变量（概括的说应该是可修改的左值）

11.2不幸的 gets()函数

读取正行的输入，直至遇到换行符，然后丢掉换行符，储存其余字符，并在其余字符末尾添加一个空字符使其称为一个 C 字符串。经常和 puts()配合使用。
用法如下：

```
char word[100];
gets(word);
```

但是，该函数只知道数组的开始处，并不知道数组中有多少个字符，如果输入的字符过长，就会导致缓冲区溢出（buffer overflow），如果这些溢出的字符知识占据尚未使用的内存，则没有什么问题，但是一旦擦写掉内存中的其他数据，就会导致程序异常中断。
所以有些不怀好意的人就专门针对这个漏洞写入一些病毒代码，于是 C 编程社区就有很多人建议在编程的时候摒弃 gets()，C99 标准委员会也接受了这个建议，承认 gets()是有问题的，但是由于当时有很多程序依然使用这 gets()函数，所以没有采取强硬的手段。
但是 C11 标准委员会态度就比较强硬了，直接从标准中放弃了 gets()函数，既然标准已经发出，那么编译器就必须根据标准来调整了。
跟 puts()配合使用的时候，其会自动在字符串末尾再添加一个换行符‘\n’

11.3gets()的替代品

fgets()函数
fgets()函数通过第二个参数限制输入的字符数来解决溢出的问题
fget(char *char, n, char *Address)，读入 n-1 个字符，或者遇到第一个换行符结束（字符串+’\n’+’\0’），并把换行符储存，这个跟 gets()去掉换行符不同，第三个参数指明要写入的地址或者文件。通常跟 fput((char *char, char *Address)配合使用，其不会自动在字符串末尾再添加一个换行符‘\n’，要不然用 puts()的话，有换行符的时候会在本身输出换行的基础上，再加上一个换行。
另外没有读完的话他会返回第一个参数的地址，读完文件的话会返回 NULL，读完其他的输入也会返回第一个参数的地址。

gets_s()函数
C11 标准中新增的，跟 fgets()函数类似，只是规定只能从标准输入 stdin 中读取，如果 gets_s 遇到换行符，他会丢掉它

11.4sprintf()函数

将多个字符串合并，用法跟 printf 差不多，只是在前面多添加一个地址参数用于储存。

十二、 字符的输入/输出和输入验证

12.1缓冲区

1) 无缓冲输入：回显用户输入的字符后立即可以使用的字符；
2) 缓冲输入：用户按下【Enter】之前不会立即被使用的字符；
3) 缓冲区：用户输入的字符收集并储存的一个临时储存区；
4) 存在缓冲区的理由：把若干字符作为一个块进行传输比逐个打印更加节省时间，其次，如果用户打错了，可以直接通过键盘进行修改，当最后按下【Enter】时，输入的才是正确的输入。
5) 存在无缓冲输入的理由：比如像打游戏的时候，用户总是希望按下一个快捷键就能立即得到相应的操作的相应，而不是按下一个键之后再去按一个【Enter】键
6) 缓冲也分两种：完全缓冲 I/O 和行缓冲 I/O
   - 完全缓冲 I/O：当用户的输入超过缓存区的内存时才刷新缓冲区（即缓冲区内存中的数据发送至目的地），一般的缓冲区的大小取决于系统，常见的大小是 512 字节和 4096 字节。常出现在文件输入中。
   - 行缓冲 I/O：出现换行符的时候刷新缓冲区，键盘的输入通常是行缓冲输入，所以当我们按下【Enter】键的时候才会刷新缓冲区。
7) 回显输入：用户输入的字符直接显示在屏幕上；
8) 无回显输入：用户输入的字符不显示在屏幕上；
9) ACSI C 采用把缓冲输入作为标准输入作为标准的原因是：一些计算机不允许无缓冲输入

12.2结束键盘输入

在较低的层面上，C 可以使用主机操作系统的一些基本文件工具来操作文件，直接调用操作系统的函数被称为底层 I/O（Low Level I/O）,由于计算机系统的差异性，ACSI C 不可能为所有的操作系统的底层 I/O 都建立一套标准库。为解决这个问题，ACSI C 建立了一套标准 I/O 包（standard I/O Package）来处理文件。上面所说的计算机系统的差异性，指的是不同系统储存文件的方式不同，处理文件结尾的方式不同，就好像有的喜欢用换行符作为每行的结尾，有的喜欢用回车符加换行符作为每行的结尾等。

对于这些差异性，ACSI C 解决的方法是不直接处理文件，而是处理流（stream），拿什么是流呢？流其实是文件输入和输出映射的理想化数据流。这意味这不同属性和不同种类的输入，由属性更加统一的流来表示。

C 的输入函数都内置了文件结尾检测器。

C 把输入和输出设备视为存储设备上的普通文件，尤其是把键盘和显示设备视为每个 C 程序自动打开的文件。

以前的 CP/M，IBM-DOS 和 MS-DOS 的文本文件曾经使用以一些特殊字符作为文件结尾的方式标记文件的结束，现代的操作系统一般用【Ctrl】+z 的方式结尾。
如：hello, lv Hongbin.
实际上显示的字符是：hello, lv Hongbin.\n^z

无论操作系统采用何种方式作为文件的结尾，C 语言的输入函数在读到文件结束的时候都会返回 EOF（end of file），EOF 的定义在 stdio.h 文件中
#define EOF (-1)

12.3重定向和文件

1) 在 UNIX 和 Linux 中<符号是重定向运算符，该运算符可以将文件和 stdin 关联起来，把文件中的内容导入到 10_echo_eof.c 程序中，而 10_echo_eof.c 程序本身并不关心输入的内容是来自于文件或者说是键盘，他只知道读取的是字符流，把字符流打印到屏幕中，直至文件的结尾。
2) 当然了。你还可以使用>符号进行重定向输出
3) 组合重定向；
   ./10_echo_eof.out < 10_echo_eof_test.txt > 10_echo_eof_test2.txt
   或者：
   ./10_echo_eof.out > 10_echo_eof_test2.txt < 10_echo_eof_test2.txt
   以上都是让 10_echo_eof_test.txt 作为输入，10_echo_eof_test2.txt 作为输出
4) 遵循的规则：
   - 重定向运算符连接一个可执行程序（包括标准操作系统命令）和一个数据文件，不能进行命令和命令的连接，或者文件与文件的连接。
   - 重定向运算符不能读取多个文件的输入，也不能输出定向至多个文件；
   - 通常，文件名和运算符之间的空格不是必须；

源文件 10_echo_eof.c：

```
/* ****************************************************************
*         Filename: 10_echo_eof.c
*      Description:
*          Version: 1.0
*          Created: 2017/01/21
*         Revision: none
*         Compiler: gcc
*           Author: Lv Hongbin
*          Company: Shanghai JiaoTong Univerity
* ****************************************************************/
```

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

        int ch;
        while((ch=getchar())!=EOF){
                putchar(ch);
        }
        return 0;
}
```

10_echo_eof_test.txt 文件内容

```
/* ****************************************************************
*    Filename: echo_eof__test.txt
*     Description:
*           Version: 1.0
*     Created: 2017/01/10
*          Revision: none
*          Compiler: gcc
*             Author: Lv Hongbin
*           Company: Shanghai JiaoTong Univerity
* ****************************************************************/

#include<stdio.h>
#include<stdlib.h>

int main(){

      return 0;
}
```

指令和结果：

```
    [root@MiWiFi-R3G-srv      /home/lvhongbin/Desktop/cStudy]#./10_echo_eof.out
< 10_echo_eof_test.txt
/* ****************************************************************
*    Filename: echo_eof__test.txt
*     Description:
*           Version: 1.0
*     Created: 2017/01/10
*          Revision: none
*          Compiler: gcc
*             Author: Lv Hongbin
```

```
*            Company: Shanghai JiaoTong Univerity
* ********************************************************/

#include<stdio.h>
#include<stdlib.h>

int main(){

    return 0;
}
```

重定向输出：如果该文件不存在则会创建，如果存在则会现进行数据清除在进行数据写入。

```
[root@MiWiFi-R3G-srv  /home/lvhongbin/Desktop/cStudy]#./10_echo_eof.out   > 10_echo_eof_test.txt
hello,lvhongbin.
how are you?
[root@MiWiFi-R3G-srv     /home/lvhongbin/Desktop/cStudy]#./10_echo_eof.out <   10_echo_eof_test.txt
hello,lvhongbin.
how are you?

[root@MiWiFi-R3G-srv /home/lvhongbin/Desktop/cStudy]#./10_echo_eof.out   >>   10_echo_eof_test.txt
I'm fine,thanks.
[root@MiWiFi-R3G-srv     /home/lvhongbin/Desktop/cStudy]#./10_echo_eof.out <   10_echo_eof_test.txt
hello,lvhongbin.
how are you?
I'm fine,thanks.
```

12.4创建友好的用户界面

断行吸【Enter】
跳过剩余的输入行，除非按下【Enter】

```
 while(getchar()!='\n'){
    continue;
 }
```

12.5scanf 函数

返回输入的个数；

十三、 结构体 struct

13.1基本使用

1) struct 是一个关键字，用于定义一个新的类型
2) struct 结构体所占内存大小可以简单认为是里面所有变量之和，当然了实际上其内存大小会稍微比里面所有变量之和大一点点，因为里面涉及到变量内存空间对齐的问题。
3) struct student 合起来才是结构体类型
4) 结构体内部定义的变量不能直接赋值
5) 结构体只是一个类型，没有定义变量之前，是没有分配空间的，没有空间，就不能赋值；

```
struct student{
    int age;
    char name[20];
    int score;
};      //有分号

struct student stu={18,”mike”, 88};    //别忘了 struct 关键字
```

也可以单独变量定义，用.点运算符或者->箭头运算符

```
stu.age=18;
```

或者(&stu)->age=18;

或者(*(&stu)).age=18; //这里有个优先级的问题，.点运算符的优先级比*要高，所以要加一个括号；

或者(&stu)[0].age=18; //这里没有问题，因为中括号和.点运算符是同级的关系，结合顺序是从左往右；

6) 如果是指针变量 struct Student *p，指针要有合法的指向，才能操作结构体成员。
7) 当然了，你还可以在定义结构体类型的同时定义结构体类型的变量

```
如：struct student{
        int age;
        char name[20];
        int score;
    } s1, s2;
如：struct student{
        int age;
        char name[20];
        int score;
    } s1{18，”mike”, 59}, s2;
```

8) 另外可以定义匿名的结构体类型的同时定义结构体类型的变量

```
如：struct {
```

```
        int age;
        char name[20];
        int score;
    } s1, s2;
```

13.2结构数组

struct student a[5];
(a+1)->age=18;

13.3结构体的嵌套

.号也可以连续使用；
s.info.age=18;

13.4同类型结构体的赋值

13.5结构体内包含一级指针

如果你用为结构体分配了堆内存，而结构体内有成员指针，那么你还要为成员指针分配内存，否则会出现野指针的错误；
而且在最后释放的时候也要注意要先释放结构体内部的根堆内存，再释放结构体的堆内存，否则万一顺序相反，先释放结构体的堆内存，再释放结构体内部的根堆内存，编译器就会由于找不到结构体内部的根堆内存而释放失败；

十四、 共同体 union

14.1基本使用

1) 共同体的内存大小计算跟 struct 不同，不是里面所有成员变量内存的累加，而是由其内部最大成员变量的内存决定；

   如：

```
union test{
    unsigned char a;
    unsigned short b;
    unsigned int c;
 }obj;
```

2) 共用体内的成员变量公用一块内存，所有成员的地址都一样，不信你可以打印一下他们的地址
Printf(“%p\t, %p\t, %p\t”, &obj.a, &obj.b, &obj.c);

3) 给某个成员赋值会影响到其他成员，比如&obj.c=0x44332211;左边是高位，右边是低位，高位放高地址，低位放低地址；首地址放低位；

如：

```
/* ****************************************************************
 *         Filename: 06_UnionTest.c
 *      Description:
 *          Version: 1.0
 *          Created: 2017/01/14
 *         Revision: none
 *         Compiler: gcc
 *           Author: Lv Hongbin
 *          Company: Shanghai JiaoTong Univerity
 * ****************************************************************/

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

union test{

        int a;
        short b;
        char c;
}obj;


int main(){

        printf("\n/* ***************************************\n");
        printf("UnionTest\n");
        printf("Please input a 0x number to print: obj.c=");
        scanf("%x",&obj.c);
        getchar();
        printf("Result:\n");
        printf("The address of a=%p\tb=%p\tc=%p\n",&obj.a,&obj.b,&obj.c);
        printf("The content of a=%x\tb=%x\tc=%x\n",obj.a,obj.b,obj.c);
        printf("\n* ***************************************/\n");
        return 0;
```

}

结果：

[root@MiWiFi-R3G-srv /home/lvhongbin/Desktop/cStudy]# gcc 06_UnionTest.c -o 06_UnionTest.out && ./06_UnionTest.out

/* ************************************
UnionTest
Please input a 0x number to print: obj.c=0x12345678
Result:
The address of a=0x601050 b=0x601050 c=0x601050
The content of a=12345678 b=5678 c=78

十五、 枚举 enum 和 typedef 的使用

15.1枚举 enum 基本使用

1) 在 main 函数前面添加

```
enum  color{
    black, white, yellow, blue, grey
};
```

这样就相当于：

```
#define black 0；
#define white 1；
#define yellow 2；
#define blue 3；
#define grey 4；
```

2) 当然了，你也可以直接在定义的时候赋值，如：

```
enum  color{
    black, white, yellow=10, blue, grey
};
```

这就相当于 black=0，white=1，yellow=10，blue=11，grey=12；

3) 定义枚举变量的时候直接用

枚举类型 变量名；

如：

```
emun color flag;
```

15.2typedef 基本使用

其实就是把自己命名数据类型，给他起别的名字，特别是应对数据体结构和共同体结构类型的时候，可以自定义自己喜欢的名字；

typedef unsigned int size_t;

或者

Typedef struct Test{

} Test2;　　//相当于给 struct Test 结构体类型自定义名字为 Test2

不同于宏定义的是，其发生在编译阶段

十六、 文件操作

16.1磁盘文件 vs 设备文件

1) 磁盘文件
   指一组相关数据的有序集合，通常存储在外部介质（如磁盘，硬盘）上，使用时才调入内存。而磁盘文件又分为文本文件和二进制文件；
2) 设备文件
   在操作系统中把每一个与主机连接的外部输入和输出设备看作是一个文件，把他们的输入和输出等同于对磁盘文件的读和写。如鼠标和键盘就是标准的输入设备，屏幕就是标准的输出设备。

16.2FILE 结构体和相关函数概述

printf 函数也好，scanf 函数也好，还是 fget()或者 fput()也好，数据的输入和输出到屏幕或者文件并不是一步到位的，而是中间隔了一个缓冲区，避免频繁交互，提高效率。

FILE 所有平台的名字都一样，这是为了方便移植， FILE 是一个结构体类型，里面的成员功能一样，不同平台成员的名字不一样。

```
Typedef struct{
    short     level;    //缓冲区“满”或者“空”的程度；
    unsigned     flags;    //文件状态标志；
    char     fd;  //文件描述符，就是跟洗头膏最多能同时打开 1024 个文件有关；
    unsigned char     hold;     //如无缓冲区不读取字符；
    short     bsize;    //缓冲区的大小；
    unsigned char     *buffer; //数据缓冲区的位置；
    unsigned     ar;  //指针，当前的指向，其实跟鼠标的光标所在的位置有关；
```

```
    unsigned     istemp; //临时文件，指示器；
    short     token;   //用于有效性的检查；
}FILE；
```

FILE *fp
fp 指针，只用调用 fopen()，在堆区分配空间，把地址返回给 fp 指针变量。但是，fp 指针不是指向文件，fp 指针和文件关联，fp 内部成员保存了文件的状态。操作 fp 指针，不能直接操作，必须通过文件库函数来操作 fp 指针，通过库函数操作 fp 指针，对文件的任何操作，fp 里面的成员会相应的变化（系统自动完成）。

16.3读写文件的过程

1) 打开文件 fopen()
2) 读写文件
   - 按字符读写 fgetc(),fputc();
   - 按字符串（行）读取文件 fgets()，fputs();
   - 文件结尾的判断 feof();
3) 关闭文件 fclose()

C 语言中有三个特殊的文件指针由系统默认打开，用户无需定义即可直接使用：

1) stdin：标准输入，默认是当前的终端（键盘），我们使用 scanf，getchar 函数默认从此终端获得数据；
   关掉此指针调用 fclose(stderr)，关掉此指针意味着无法使用 scanf，getchar 函数等函数
2) stdout：标准输出，默认是当前的终端（屏幕），我们使用 printf，putchar 函数默认从此终端输出数据, 关掉此指针调用 fclose(stderr);；
   如
   printf(“1234567”);
   fclose(stdout);
   printf(“890”);   //此句无法打印，原因在于标准输出文件已经被关闭；

3) stderr：标准出错，默认是当前的终端（屏幕），我们使用 perror 函数默认从此终端输出数据；
   关掉此指针调用 fclose(stderr);

close(1);   1 表示标准输出设备，关闭了，1 就是空闲的数字
int fd =open("/home/lvhongbin/Desktop/cStudy/stdio.txt",O_WRONLY,0777);
//打开输出设备，并返回一个最小可用的数字

如：

```
    /* ************************************************************
```

```
 *         Filename: 07_StdioTest.c
 *      Description:
 *           Version: 1.0
 *           Created: 2017/01/15
 *         Revision: none
 *         Compiler: gcc
 *           Author: Lv Hongbin
 *           Company: Shanghai JiaoTong Univerity
 * ================================================================*/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(){

        printf("\n/* ***************************************\n");
        printf("StdioTest\n");
        printf("Please input what you want to write:    ");
        char *pChar;
        pChar=(char *)malloc(20*sizeof(char));
        scanf("%s",pChar);
        printf("Result:%s\n",pChar);
        close(1);
        printf("123456789\n");
        perror("The string \"123456789\" was ignore because of close(1)");
        perror("write \"123456789\" on the
/home/lvhongbin/Desktop/cStudy/stdio.txt");
        perror("\n* ***************************************/");
        free(pChar);
        int fd
=open("/home/lvhongbin/Desktop/cStudy/stdio.txt",O_WRONLY,0777);
        printf("123456789\n");
        close(1);
        return 0;
}
```

结果：

/* **
StdioTest
Please input what you want to write: hello
Result:hello
The string "123456789" was ignore because of close(1): Bad file descriptor
write "123456789" on the /home/lvhongbin/Desktop/cStudy/stdio.txt: Bad file descriptor

* **/: Bad file descriptor

stdio.txt:
123456789

16.4文件的打开

任何文件在打开之前必须打开；
FILE *fopen(const char *filename, const char * mode);
功能：打开文件
参数：

- filename：需要打开的文件名，跟据需要加上路径；
  在 windows 系统中，有两种方式的路径的写法
  D:\\abc\\def\\ghi　　由于“\”是转义字符，所以要多写一次
  或者：
  D:/abc/def/ghi
  在 linux 系统中，只有一下一种写法
  /home/lvhongbin/Desktop

  相对路径：在 Linux 中，相对路径相对于可执行程序
  在 VS 中
  编译的同时运行，相对路径相对于 xxx.vcxproj 所在的路径；
  如果直接运行程序，相对路径相对于可执行程序
  在 Qt 中
  编译的同时运行，相对路径相对于 debug 所在的路径；
  如果直接运行程序，相对路径相对于可执行程序
- mode：打开文件的模式设置；
  r 或者 rb：以只读的方式打开一个文本文件，若文件不存在则报错；
  w 或者 wb：以写的方式打开一个文本文件，再清空所有，最后写入数据，若文件不存在则创建；
  a 或者 ab：以写的方式打开一个文本文件，接着文字的末尾写入数据，若文件不存在则创建；
  r+或者 rb+：以可读可写的方式打开一个文本文件，若文件不存在则报错；

w+或者 wb+：以可读可写的方式打开一个文本文件，再清空所有，最后读写数据，若文件不存在则报错；

a 或者 ab：以可读可写的方式打开一个文本文件，接着文字的末尾写入数据，最后读写数据，若文件不存在则报错；

返回值：

成功：文件指针；

失败：NULL；

16.5读写文件

1) 写文件

Int fputc(int ch, FILE *stream);

功能：将 ch 转换为 unsigned char 后写入 stream 制定的文件中，并且光标后移一位；

参数：

ch：需要写入文件的字符

stream：文件指针

返回值：

成功：成功写入文件的字符

失败：返回-1；

Int fputs(char *ch, int sizeof(ch), FILE *stream);

功能：将字符串写入文件，换行需要额外加换行符 ；

参数：

ch：需要写入文件的字符

stream：文件指针

返回值：

成功：成功写入文件的字符

失败：返回-1；

2) 读文件

Int fgetc(FILE *stream);

功能：从 stream 文件中读取一个字符，并且光标后移一位；

参数：

stream：文件指针

返回值：

成功：返回读取到的字符

失败：返回-1；

注意：如果是文本文件，可以通过-1 或者 EOF 判断文件是否结尾

如果是二进制文件，不可以通过-1 判断文件的结尾，这时候需要用 feof(FILE *stream)函数判断是否为文件结尾，任何文件都可以，特别是文件中穿插着 char -1 字符的时候。如果返回值为真，说明检测到了文件的末尾。

Int fgets(FILE *stream);
功能：从 stream 文件中读取字符串，一次最大读取为 sizeof(buf)-1，如果遇到换行符，文件结尾，出错，结束本次读取，不过换行符和文件结尾都会都进来。
参数：
stream：文件指针
返回值：
成功：返回读取到的字符
失败：返回-1；

结束符 '\0' ascii 是 0
空格‘ ’ ascii 是 32
空格不是'\0'

3) 判断文件末尾
feof(FILE *stream)
判断是否为文件结尾，任何文件都可以，特别是文件中穿插着 char -1 字符的时候。如果返回值为真，说明检测到了文件的末尾。
但是需要注意的是，如果第一次没有进行文件的读操作，直接调用此函数，永远返回假，因为这只是判断上一次读取的结果，而且判断后光标不会移动 ，所以使用此函数时必须先读。

Vim 命令和 cat 命令的练习：

```
 *           Created: 2017/01/15
 *          Revision: none
 *          Compiler: gcc
 *            Author: Lv Hongbin
 *           Company: Shanghai JiaoTong Univerity
 * ======================================================================/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(){

        printf("\n/* ***************************************\n");
        printf("StdioTest2\n");
```

```
        printf("Please input what you want to write/read:    ");
        char *pChar;
        pChar=(char *)malloc(20*sizeof(char));
        scanf("%s",pChar);
        getchar();
        printf("Please input where you want to write/read:    ");
        char *pDir;
        pDir=(char *)malloc(40*sizeof(char));
        scanf("%s",pDir);
        getchar();
        printf("Please input the mode of file:    ");
        char *pMode;
        pMode=(char *)malloc(5*sizeof(char));
        scanf("%s",pMode);
        getchar();
        printf("Result:\nDir:%s\nData:%s\nMode:%s\n",pDir,pChar,pMode);
        FILE *fp=fopen(pDir,pMode);
        char buf[1024];
        int i=0,j=0;
        if(strcmp(pMode,"r")==0){
                while(1){

memset(buf,0,sizeof(buf)/sizeof(buf[0])*sizeof(char));
                        fgets(buf,sizeof(buf),fp);
                        if(feof(fp)){
                                break;
                        }
                        printf("No.%d:%s",++i,buf);
                }
        }else{
                while(1){

memset(buf,0,sizeof(buf)/sizeof(buf[0])*sizeof(char));
                        printf("No.%d: ",++i);
printf("No.%d: ",++i);
                        for(j=0;j<1024;j++){
                                buf[j]=getchar();
                                if(buf[j]=='\n'){
                                        break;
                                }
                        }
                        if(strcmp(buf,":wq\n")==0){
```

```
                                        break;
                                    }
                                    fputs(buf,fp);
                                }
                            }
                            free(pChar);
                            free(pDir);
                            free(pMode);
                            fclose(fp);
                            memset(buf,0,sizeof(buf)/sizeof(buf[0])*sizeof(char));
                            printf("\n* ****************************************/");
                            return 0;
                        }
```

4) 从 buf 中读取和写入数据
   sscanf(char * ch, 正则表达式，&a, &b, &c …) ；
   sprintf(char * ch, 正则表达式，内容) ；
5) fread 和 fwrite
   fread(buf, size, count, fp) //count 代表多少个 size，size 的单位是字节数
   fwrite(buf, size, count, fp)
6) 文件的定位
   Fseek(文件类型指针，位移量，起始点);
   文件首 SEEK-SET 0；
   文件当前位置 SEEK-CUR 1；
   文件末尾 SEEK-END 2
   位移量需要是长整形，以便在文件长度大于 64KB 时不会出错，负数表示向后退多少个光标；

   Int rewind(文件类型指针) //光标重返文件开头

   Long ftell(文件类型指针) //得到当前光标相对于开头的位移量，-1L 时表示错误，可以用来求字符串长度

十七、 位操作

高级语言一般不会提供给位的操作，但是 C 语言在提供高级语言便利的同时，还能在为汇编语言所保留的级别上工作，这使其称为编写设备驱动器和嵌入式代码的首选语言。

17.1按位(bitwise)逻辑运算符

1) 二进制反码/按位取反~
如： ~(10011010)
结果：01100101
改运算符不会改变原来的值，只会创建一个可以使用或者赋值的新值。
2) 按位与&
只有两个都为 1 的时候才为 1
如(10010011)&(00111101)
结果：00010001
可以跟赋值符号结合&=
常用于掩码（mask），所谓掩码指的是一些设置为开（1）或者关（0）的位组合，如 flags=flag&(00000010)，那么只有 1 号位能看，其他位都设置成 0，看不到，所以叫掩码。
清空位的操作
3) 按位或|
一个为 1 则为真
常用于打开位，如 flags=flag | (00000010)，那么只有 1 号位肯定被打开；
4) 按位异或^
按位两个数不同的时候才为真
5) 左移<<
用 0 填充空出的位置
如：（10001010） << 2；
结果：00101000
无符号整数每左移一位一般相当于乘以 2。
6) 右移>>
至于用什么填充空出的位置，取决于机器
如：（10001010） << 2；
结果：00101000

17.2位字段（bit field）：

1) 位字段是一个 signed int 或者 unsigned int 类型变量中的一组相邻的位，通过一个结构生命来建立，为每个字段提供一个标签，并确定该字段的宽度，如：

```
struct{
    unsigned int qw: 1;
    unsigned int er: 1;
    unsigned int ty: 1;
    unsigned int ui: 1;
}prnt;
```

通过该声明可以创建一个包含 4 位的数据类型，由于每个字段都只有一位，所以每位只能赋值 0 或者 1；
prnt.qw=1;

2) 无符号整数每左移一位一般相当于乘以 2；

十八、 C 预处理器和 C 库

18.1翻译程序的第一步

1) 将源代码中出现的字符映射到源字符集中，
2) 定位反斜杆后面跟着的换行符的实例，并删除他们，也就是把两个物理行（physic line）换成为一个逻辑行（logical line）,这是因为预处理表达式必须是一个逻辑行，一个逻辑行可以包含多个物理行。
3) 编译器将文本划分成预处理记号序列，空白序列和注释序列，而且编译器用一个空格字符替换每一条注释和所有空白字符序列（不包括换行符）。
4) 最后，程序已经准备好进入预处理阶段，查找一行中以#号开始的预处理指令。

18.2明示常量#define

指令可以出现在源文件的任何地方，其定义从指令出现的地方到该文件末尾有效。
如：

#define	PX	printf(“x is %d.”, x);
预处理指令	宏	替换体

宏可以代表值，如本例，这些宏被称为类对象宏（object-like marco）,也可以是类函数宏（function-like macro）
一旦预处理器在程序中找到宏的实例后，就用替换体替代该宏，从宏变成最终替换文字的过程称为宏展开（macro expansion）
当然了，替换体中还可以嵌套宏。

1) 记号
   从技术的角度看，可以把宏的替换体看作是记号（token）型字符串，而不是字符型字符串。预处理器记号是宏定义的替换体中单独的“词”。用空白将他们分开，如： #define FOUR 2*2；该宏有 1 个 2*2 序列；
   如： #define FOUR 2 * 3；该宏有 3 个记号：2、*、3。
   解释为字符型字符串则把空格视为替换体的一部分，若是为记号（token）型字符串，则空格视为分割记号的分隔符。
2) 重定义常量
3) 假如首先把 X 定义为 20，再把 X 定义为 24，这个过程就叫做重定义常量，一般来讲，不同的实现采用不同的技术方案
   - 方案一：除非新旧两种定义完全相同，否则视为错误；（ANSI 标准采用

第一种）

- 方案二：另外一些实现允许重定义，但是会给出警告；

  比如： #define FOUR 2 * 3

  #define FOUR 2 * 3

  第一种具有具有一个记号，而第二种有三个几号，按照方案一的定义，这会报错；

4) 类函数宏（function-like macro）

#define	MEAN(X, Y)	(((X)+(Y))/2)
预处理指令	宏	替换体

宏中的参数在圆括号中，但是替换体中为什么要用圆括号呢？那肯定是有原因的^<*

看以下例子：

```
#include<stdio.h>
#define SQUARE(X) X*X
int main(void){
    int x=5;
    printf("The result is %d.", SQUARE(x+2));
}
```

将会输出 The result is 17.

为什么呢？因为预处理中对与记号是不做计算处理的，只会做一些简单的替换工作，所以就变成 x+2*x+2，当 x=5 的时候，结果自然就是 17 喽。你想要正常的结果也可以，那就在替换体中加上圆括号吧#define SQUARE(X) (X)*(X)，结果就对了。

这就明显说明了函数调用和宏调用的重要区别，函数调用是值传递，宏调用是把参数记号传给程序。

18.3在#define 中使用参数

1) 用宏参数创建字符串：#运算符

   #define PSQR(x) printf("the square of "#x" is %d.\n", ((x)*(x)))

   这过程也叫做字符串化（stringizing）

2) 预处理器黏合剂：##运算符

   ##运算符作用：把两个记号合成一个记号

   如#define XNAME(n) x ## n

   然后在主函数中定义：int XNAME(1) = 14 //则展开后就变成 x1=14

3) 变参宏：…和__VA_ARGS__

   接收的参数数量可以改变的

   如#define MS(X, …) printf("Message "#x" :"__VA_ARGS__)

   在函数中使用 MS(2, the result is %d and %d", x, (x)*2);

   则结果为 Message 2 : the result is 2 and 4。

   注意，字符串和字符串之间可以进行串联，省略号只能替代最后的宏参数。

18.4宏和函数的选择

宏的优点：宏生成内联函数，不用担心变量类型，这是因为宏处理的是字符串，不是实际的值，因此，只要能用 int 或者 float 类型都可以使用宏。由于不需要像函数那样进行回调，所以比较节省时间。
宏的缺点：需要在程序中插入很多次，浪费空间；容易出错。

18.5文件包含：#include

当预处理器发现#include 指令时，会查看后面的文件名，并把文件的内容包含到当前的文件中，即替换源文件中的#include 指令。
#include<stdio.h>
#include”list.h”
尖括号表示告诉预处理器在标准系统目录下查找文件，双引号告诉预处理器首先在当前目录下，或者文件名指定的目录下查找该文件，没有的话再去查找标准文件。
使用头文件可以

- 明示常量
- 宏函数
- 函数声明
- 结构模板定义
- 类型定义

18.6其他指令与预定义宏

#undef 宏名称 //表示取消之前已经定义好的宏
在 C99 标准中，添加了一系列预定义宏，如
_ _DATE_ _表示返回日期的字符串，“Mmm dd yyyy”；
_ _FILE_ _表示当前文件名字符串；
_ _LINE_ _表示当前行号的字符串；
_ _TIME_ _表示返回翻译代码的时间的字符串，格式为“hh:mm:ss”；
_ _DATE_ _表示返回日期的字符串；
_ _DATE_ _表示返回日期的字符串；
在用 gcc 编译时，必须设置-std=c99 或者-std=c11

18.7条件编译 conditional compilation

```
#ifdef 某宏的名称
    //动作
```

```
#else
    //动作
#endif

#ifndef 某宏的名称
    #define 某宏的名称
#elif
    //动作
#endif
```

为了避免头文件定义重复，实现的供应商使用这些方法解决这个问题，用文件名作为标识符，使用大写字母，用下划线字符替代文件名中的点符号，用下划线字符作为前缀或后缀，也可以使用两条下划线

如

```
#ifndef _LVHONGBIN_H
        #define _LVHONGBIN_H
    #endif
```

当然了，这只是供应商额做法，为了避免跟标准头文件中的宏发生冲突，一般在自己的代码中不会这样写，而写成：

```
 #ifndef   LVHONGBIN_H_
          #define LVHONGBIN_H_
      #endif
```

十九、 高级数据表示

通常，程序开发最重要的部分是找到程序中表示数据的好方法

19.1从数组到链表

如果你想要为链表的结构体动态分配内存的话，有两种办法：

- 一次动态创建 n 个结构体；

  ```
  p=( struct name *)malloc(n* sizeof(struct name));
  ```

- 使用循环，循环 n 次创建；

  ```
  for(i=0;i<n;i++){
      p=( struct name *)malloc(sizeof(struct name));
  }
  ```

这有什么不一样呢？前者他们的堆的分配地址是连续的，使用指针的话用一个就够了，但是堆内存的创建一次性的，也就是说一次性要分配确定数字的内存，可能会造成浪费，不能做到动态创建任意数量的内存空间。而后者可以解决动态创建的问题，但是地址则不一定连续，需要 n 个指针进行储存。

那怎么解决呢？

我们选择用后者的方案，然后相出了一招很好的办法解决内存分配地址不连续的问题，就是在每一个的结构体中定义该指向该结构体的指针，用于储存下一个结构体的地址（虽然结构不能含有与本身类型相同的结构，但是可以含有与本身类型相同的指针）。

19.2 使用链表

19.3 抽象数据类型 ADT

什么是类型？类型指的是两类信息：属性和操作；
定义新类型的方法：

1) 提供类型属性和相关操作的抽象描述，这种正式的抽象描述称为 ADT
2) 开发一个实现 ADT 的编程接口，也就是说，指明如何储存数据和执行所需要操作的函数，这类函数就相当于 C 基本类型的内置运算符。说白了，就是用一个头文件新的数据类型和声明操作函数。
3) 编写代码实现接口，其实就是编写代码实现这些操作函数，放在头文件中。

19.4一个成熟的链表程序

*头文件 **list.h***

```
/* ************************************************************
 *       Filename: list.h
 *       Description:
 *       Version: 1.0
 *       Created: 2017/01/19
 *       Revision: none
 *       Compiler: gcc
 *       Author: Lv Hongbin
 *       Company: Shanghai JiaoTong Univerity
 * ************************************************************/

#ifndef LIST_H_
#define LIST_H_
#include<stdbool.h>
/* ****************************************
 *  Type Name:       List
 *  Type Attributes:  Item item;
 *                          struct node *next;
 *
```

```
 *   Type Operation:        void InitializeList(List *plist);
 *                  boolean ListIsEmpty(const List *plist);
 *                  boolean ListIsFull(const List *plist);
 *                  unsigned int ListItemCount(const List *plist);
 *                  boolean AddItem(Item item, List *plist);
 *                  void Traverse(const List *plist, void(*pfun)(Item item));
 *                  void EmptyTheList(List *plist);
 * ****************************************/

#define TSIZE 45

    struct film{

        char title[TSIZE];
        int rating;
    };
    typedef struct film Item;

    typedef struct node{

        Item item;
        struct node *next;
    }Node;

    typedef Node * List;

/*   function prototype     */


/*   Function:     Initialize a list              */
/*   Precondition:      plist point at a list              */
/*   Postcondition:     make a list NULL              */
void InitializeList(List *plist);

/*   Function:     Makesure a list empty              */
/*   Precondition:      plist point at an initialized list */
/*   Postcondition:     Empty?true:false          */
bool ListIsEmpty(const List *plist);

/*   Function:     Makesure a list   full          */
/*   Precondition:      plist point at an initialized list */
/*   Postcondition:     Full?true:false              */
```

```
bool ListIsFull(const List *plist);

/*    Function:     Initialize a list                */
/*    Precondition:     plist point at an initialized list */
/*    Postcondition:    make a list NULL                 */
unsigned int ListItemCount(const List *plist);

/*    Function:     Initialize a list                */
/*    Precondition:     plist point at an initialized list */
/*    Postcondition:    make a list NULL                 */
bool AddItem(Item item, List *plist);

/*    Function:     Apply func on every item         */
/*    Precondition:     plist point at an initialized list */
/*                pfun point at a func without return */
/*    Postcondition:    Apply func on every item         */
void Traverse(const List *plist, void(*pfun)(Item item));

/*    Function:     Delete    a list                 */
/*    Precondition:     plist point at a list            */
/*    Postcondition:    Make a list empty                */
void EmptyTheList(List *plist);

#endif
```

具体实现文件 list.c

```
/* ************************************************************
 *       Filename: list.c
 *       Description:
 *       Version: 1.0
 *       Created: 2017/01/21
 *       Revision: none
 *       Compiler: gcc
 *       Author: Lv Hongbin
 *       Company: Shanghai JiaoTong Univerity
 * ************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include "list.h"

void InitializeList(List *plist){
```

```
    *plist=NULL;
}

bool    ListIsEmpty(const List *plist){

    return *plist==NULL;
}

bool ListIsFull(const List *plist){

    Node *pt;
    pt=(Node *)malloc(sizeof(Node));
    if (pt==NULL){
        return true;
    }else{
        free(pt);
        return false;
    }
}

unsigned int ListItemCount(const List *plist){

    unsigned int count=0;
    Node *temp=*plist;
    while(temp!=NULL){
        count++;
        temp=temp->next;
    }
    return count;
}

bool AddItem(Item item, List *plist){

    Node *pnew;
    Node *temp=*plist;
    pnew=(Node *)malloc(sizeof(Node));
    if(pnew==NULL){
        return false;
    }
    pnew->item=item;
    pnew->next=NULL;
```

```
    if(temp==NULL){
        *plist=pnew;
    }else{
        while(temp->next!=NULL){
            temp=temp->next;
        }
        temp->next=pnew;
    }
    return true;

}

void Traverse(const List *plist, void(*pfun)(Item item)){

    Node *temp=*plist;
    while(temp!=NULL){
        pfun(temp->item);
        temp=temp->next;
    }
}

void EmptyTheList(List *plist){

    Node *tem;
    while(*plist!=NULL){
                    tem=(*plist)->next;
                    free(*plist);
        *plist=tem;
         }
}
```

主程序：film.c

```
/* ****************************************************************
 *       Filename: film.c
 *       Description:
 *       Version: 1.0
 *       Created: 2017/01/21
 *       Revision: none
 *       Compiler: gcc
 *       Author: Lv Hongbin
 *       Company: Shanghai JiaoTong Univerity
```

```
 * ************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include "list.h"

char * s_gets(char *ctitle, int N);
void showmovies(Item item);
int main(){

    List movies;
    Item temp;

    InitializeList(&movies);
    if(ListIsFull(&movies)){
        fprintf(stderr,"No memory avaliable! Bye!\n");
        exit(1);
    }

    puts("Enter the first movie title:");
    while(s_gets(temp.title,TSIZE)!=NULL &&    temp.title[0]!='\0' ){
        puts("Enter your rating<0-10>: ");
        scanf("%d",&(temp.rating));
        while(getchar()!='\n'){
            continue;
        }
        if(AddItem(temp, &movies)==false){
            fprintf(stderr,"Problem allocating memory!\n");
            break;
        }
        if(ListIsFull(&movies)){
                    puts("The lists is full.");
                    break;
            }
        puts("Enter the next movie title(empty line to stop):");
    }

    if(ListIsEmpty(&movies)){
        printf("No data entered.");
    }else{
        printf("Here is the movie list:\n");
        Traverse(&movies,showmovies);
```

```
    }
    printf("You enter %d movies.\n", ListItemCount(&movies));

    EmptyTheList(&movies);
    printf("Bye!\n");
    return 0;
}

void showmovies(Item item){

    printf("Movie:%s Rating:%d\n",item.title,item.rating);
}

char *s_gets(char *ctitle, int N){

    char *newcTitle;
    int i=0;
    newcTitle=fgets(ctitle,N,stdin);
    if(newcTitle){
        while(ctitle[i]!='\n' && ctitle[i]!='\0'){
            i++;
        }
        if(ctitle[i]=='\n'){
            ctitle[i]='\0';
        }else{
            while(getchar()!='\n'){
                continue;
            }
        }
    }
    return newcTitle;
}
```

结果：

```
[root@MiWiFi-R3G-srv /home/lvhongbin/Desktop/cStudy]#gcc list.c film.c -o film.out
[root@MiWiFi-R3G-srv /home/lvhongbin/Desktop/cStudy]#./film.out
Enter the first movie title:
Hello,Lv Hongbin
Enter your rating<0-10>:
10
Enter the next movie title(empty line to stop):
```

Goodbye, Lv Hongbin
Enter your rating<0-10>:
9
Enter the next movie title(empty line to stop):

Here is the movie list:
Movie:Hello,Lv Hongbin Rating:10
Movie:Goodbye, Lv Hongbin Rating:9
You enter 2 movies.
Bye!

提示：考虑函数的形参限定修饰符为 const 的作用与如何在常指针的条件下进行传值。

19.5队列_C primer plus 观点

1) 队列（queue）是拥有两个特殊属性的链表（linked list）：
   - 新项只能添加到链表的结尾；
   - 只能从链表的开头移除项；
2) 队列是一种“先进先出”（first in, first out FIFO）的数据形式

3) 同样遵循设计思路是：
   - 定义队列的抽象数据类型
   - 定义一个接口
     包括定义数据类型和函数原型
   - 实现接口数据表示
     这里的关键问题是如何处理删除项和插入项的问题。这里面有几种方案，大体有两大类，一种是数组，另外一种是链表；而数组的又分为整体移动数组方案，移动数组首项方案和环形数组方案，但是以上的数组方案发现都有问题，或者说都不是特别好的方案。

19.6一个成熟的队列程序

头文件：queue.h

```
/* ************************************************************
 *         Filename: queue.h
 *      Description:
 *          Version: 1.0
 *          Created: 2017/01/21
 *         Revision: none
 *         Compiler: gcc
```

```
 *            Author: Lv Hongbin
 *          Company: Shanghai JiaoTong Univerity
 * ********************************************************/

#ifndef QUEUE_H_
#define QUEUE_H_
#include<stdbool.h>
/* ****************************************
 *        Type Name:                  Queue
 *        Type Attributes:          Item item;
 *                                      struct node *next;
 *
 *        Type Operation:           void InitializeQueue(Queue *pqueue);
 *                                      boolean QueueIsEmpty(const Queue *pqueue);
 *                                      boolean QueueIsFull(const Queue *pqueue);
 *                                        unsigned int QueueItemCount(const Queue
*pqueue);
 *                                      boolean AddItem(Item item, Queue *pqueue);
 *                                              void Traverse(const Queue *pqueue,
void(*pfun)(Item item));
 *                                      void EmptyTheQueue(Queue *pqueue);
 * ****************************************/

#define MAXQUEUE 10

        typedef int Item;

        typedef struct node{

                Item item;
                struct node *next;
        }Node;

        typedef struct queue{
                Node * front;
                Node * rear;
                int items;
        }Queue;

/*      function prototype      */
```

```
/*        Function:          Initialize a queue                          */
/*        Precondition:      pqueue point at a queue                     */
/*        Postcondition:     make a queue NULL                           */
void InitializeQueue(Queue *pqueue);

/*        Function:          Makesure a queue empty                      */
/*        Precondition:      pqueue point at an initialized queue        */
/*        Postcondition:     Empty?true:false                            */
bool QueueIsEmpty(const Queue *pqueue);

/*        Function:          Makesure a queue full                       */
/*        Precondition:      pqueue point at an initialized queue        */
/*        Postcondition:     Full?true:false                             */
bool QueueIsFull(const Queue *pqueue);

/*        Function:          Initialize a queue                          */
/*        Precondition:      pqueue point at an initialized queue        */
/*        Postcondition:     make a queue NULL                           */
unsigned int QueueItemCount(const Queue *pqueue);

/*        Function:          Initialize a queue                          */
/*        Precondition:      pqueue point at an initialized queue        */
/*        Postcondition:     make a queue NULL                           */
bool AddItem(Item item, Queue *pqueue);

/*        Function:          Delete the head item                        */
/*        Precondition:      pqueue point at an initialized queue        */
/*                           pfun point at a func without return         */
/*        Postcondition:     Copy the head item to *pitem                */
bool DeleteItem(Item *pitem, Queue *pqueue);

/*        Function:          Delete  a queue                             */
/*        Precondition:      pqueue point at a queue                     */
/*        Postcondition:     Make a queue empty                          */
void EmptyTheQueue(Queue *pqueue);

#endif
```

接口实现文件：queue.c

```
/* ************************************************************
 *        Filename: queue.c
```

```
 *      Description:
 *           Version: 1.0
 *           Created: 2017/01/22
 *          Revision: none
 *          Compiler: gcc
 *            Author: Lv Hongbin
 *           Company: Shanghai JiaoTong Univerity
 * ======================================================================*/

#include<stdio.h>
#include<stdlib.h>
#include "queue.h"

void InitializeQueue(Queue *pqueue){

        pqueue->front=pqueue->rear=NULL;
        pqueue->items=0;
}

bool    QueueIsEmpty(const Queue *pqueue){

        return pqueue->items==0;
}

bool QueueIsFull(const Queue *pqueue){

        return pqueue->items==MAXQUEUE;
}

unsigned int QueueItemCount(const Queue *pqueue){

        return pqueue->items;
}

bool AddItem(Item item, Queue *pqueue){

        if(QueueIsFull(pqueue)){
                puts("The queue is full");
                return false;
        }else{
                Node *ptem=(Node *)malloc(sizeof(Node));
                if(ptem==NULL){
```

```
                        fprintf(stderr,"Unable to allocate memory!");
                        return false;
                }else{
                        ptem->item=item;
                        ptem->next=NULL;
                        if(QueueIsEmpty(pqueue)){
                                pqueue->front=ptem;
printf("Add the number to the head successfully: %d\n",item);
                         }else{
                                printf("Add the number successfully: %d and
total: %d\n",item,pqueue->items+1);
                                pqueue->rear->next=ptem;
                        }
                                pqueue->rear=ptem;
                                (pqueue->items)++;
                }
        }

        return true;
}

bool DeleteItem(Item *pitem, Queue *pqueue){

        if(QueueIsEmpty(pqueue)){
                puts("The queue is empty");
                return false;
        }else{
                *pitem=pqueue->front->item;
                Node *ptem=pqueue->front->next;
                free(pqueue->front);
                pqueue->front=ptem;
                pqueue->items--;
                if(pqueue->items==1){
                        pqueue->rear=NULL;
                }
                printf("Delete the number successfully: %d and
total:%d\n",*pitem,pqueue->items);
                return true;
        }
}

void EmptyTheQueue(Queue *pqueue){
```

```
        Item tem;
        while(!QueueIsEmpty(pqueue)){
                DeleteItem(&tem,pqueue);
        }
}
```

测试程序：user_queue.c

```
/* ************************************************************
 *         Filename: user_queue.c
 *      Description:
 *          Version: 1.0
 *          Created: 2017/01/21
 *         Revision: none
 *         Compiler: gcc
 *           Author: Lv Hongbin
 *          Company: Shanghai JiaoTong Univerity
 * ************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include "queue.h"

int main(){

        Queue line;
        int num;
        Item item;
        char a;
        InitializeQueue(&line);
        puts("\n/* ***************************************");
        puts("Now test the Queue interface");
        while(1){
                puts("\n\t--- [a] Add a value\t[d] delete a value\t[q] quit ---");
                a=getchar();
                while(getchar()!='\n'){
                        continue;
                 }
                if(a=='q'){
                        break;
                }
                if(a=='a'){
```

```
                puts("Please input one num:");
                scanf("%d",&num);
                while(getchar()!='\n'){
                        continue;
                }
                item=num;
                AddItem(item,&line);
        }
        if(a=='d'){
                DeleteItem(&item,&line);
        }
        if(a='q' && a!='a' && a!='d'){
                puts("Please input 'a' or 'd' or 'q'");
                continue;
        }

    }
   EmptyTheQueue(&line);
    puts("Bye!");
    return 0;
}
```

结果：

```
[root@MiWiFi-R3G-srv /home/lvhongbin/Desktop/cStudy]#gcc queue.c user_queue.c
-o user_queue.out
[root@MiWiFi-R3G-srv /home/lvhongbin/Desktop/cStudy]# ./user_queue.out

/*                      ****************************************
Now test the Queue interface

    --- [a] Add a value     [d] delete a value[q] quit ---
a
Please input one num:
12
Add the number to the head successfully: 12

    --- [a] Add a value     [d] delete a value[q] quit ---
a
Please input one num:
24
Add the number successfully: 24 and total: 2

    --- [a] Add a value     [d] delete a value[q] quit ---
```

a
Please input one num:
36
Add the number successfully: 36 and total: 3

--- [a] Add a value [d] delete a value [q] quit ---
d
Delete the number successfully: 12 and total:2

--- [a] Add a value [d] delete a value [q] quit ---
d
Delete the number successfully: 24 and total:1

--- [a] Add a value [d] delete a value [q] quit ---
q
Delete the number successfully: 36 and total:0
Bye!

19.7链表和数组

随机访问（random access）：可以使用数组下标直接访问该数组中的任意元素；
顺序访问（sequential acess）：对链表而言，必须从链表额首字节开始，数个节点移动到要访问的节点。
数组的最大好处是可以进行随机访问，对于顺序排序好的数组使用二分法进行查找，效率最高。
链表的最大好处是可以非常快速的插入某项或者对某项进行删除，但是由于无法使用随机访问，所以不能使用二分法进行查找；

19.8二叉查找树

那存不存在把数组和链表的优点都集合起来的 ADT 呢？答案是有的，那就是二叉查找树，规定二叉树的项不能重复。

```
void InitializeTree(Tree *ptree);
bool TreeIsEmpty(const Tree *ptree);
bool TreeIsFull(const Tree *ptree);
unsigned int TreeItemCount(const Tree *ptree);
bool AddItem(const Item *item, Tree *ptree);
bool DeleteItem(const Item *item, Tree *ptree);
bool FindItem(const Item *item, Tree *ptree);
void Traverse(const Tree *ptree, void (*pfun)(Item *item));
void EmptyTheTree(Tree *ptree);
```

头文件：tree.h

```
/* ************************************************************
 *   Filename: tree.h
 *    Description:
 *         Version: 1.0
 *    Created: 2017/01/22
 *         Revision: none
 *         Compiler: gcc
 *            Author: Lv Hongbin
 *          Company: Shanghai JiaoTong Univerity
 * ************************************************************/

#ifndef TREE_H_
#define TREE_H_
#include<stdbool.h>
/* ****************************************
 *   Type Name:       Tree
 *   Type Attributes:  Item item;
 *                              struct node *next;
 *
 *   Type Operation:void InitializeTree(Tree *ptree);
 *                  bool TreeIsEmpty(const Tree *ptree);
 *                  bool TreeIsFull(const Tree *ptree);
 *                  unsigned int TreeItemCount(const Tree *ptree);
 *                  bool AddItem(const Item *item, Tree *ptree);
 *                  bool DeleteItem(const Item *item, Tree *ptree);
 *                  bool FindItem(const Item *item, Tree *ptree);
 *                  void Traverse(const Tree *ptree, void (*pfun)(Item *item));
 *                  void EmptyTheTree(Tree *ptree);
 *
 * ****************************************/

#define TSIZE    20
#define MAXITEMS 10

    struct item{

        char petname[TSIZE];
        char petkind[TSIZE];
    };
    typedef struct item Item;
```

```
    typedef struct trnode{

        Item item;
        struct trnode * left;
        struct trnode * right;
    }Trnode;

    typedef struct tree{

        Trnode *root;
        int size;
    }Tree;

    typedef struct pair{
        Trnode *parent;
        Trnode *child;
    }Pair;

/*  function prototype  */


/*  Function:   Initialize a tree           */
/*  Precondition:   ptree point at a tree           */
/*  Postcondition:  make a tree NULL            */
void InitializeTree(Tree *ptree);

/*  Function:   Makesure a tree empty           */
/*  Precondition:   ptree point at an initialized tree  */
/*  Postcondition:  Empty?true:false            */
bool TreeIsEmpty(const Tree *ptree);

/*  Function:   Makesure a tree  full           */
/*  Precondition:   ptree point at an initialized tree  */
/*  Postcondition:  Full?true:false             */
bool TreeIsFull(const Tree *ptree);

/*  Function:   Count the item of tree          */
/*  Precondition:   ptree point at an initialized tree  */
/*  Postcondition:  number of item              */
unsigned int TreeItemCount(const Tree *ptree);
```

```
/*    Function:      Add an item into a tree                  */
/*    Precondition:      ptree point at an initialized tree      */
/*    Postcondition:     Add successfully?true:false          */
bool AddItem(const Item *item, Tree *ptree);

/*    Function:      Delete an item into a tree         */
/*    Precondition:      ptree point at an initialized tree      */
/*                 pfun point at a func without return */
/*    Postcondition:     Delete successfully?true:false              */
bool DeleteItem(const Item *item, Tree *ptree);

/*    Function:      Find an item into a tree           */
/*    Precondition:      ptree point at an initialized tree      */
/*                 pfun point at a func without return */
/*    Postcondition:     Apply func on every item         */
bool FindItem(const Item *item, Tree *ptree);

/*    Function:      Apply certain function on every tree     */
/*    Precondition:      ptree point at an initialized tree      */
/*                 pfun point at a func without return */
/*    Postcondition:     Apply certain function on every tree     */
void Traverse(const Tree *ptree, void (*pfun)(Item item));

/*    Function:      Delete    a tree                   */
/*    Precondition:      ptree point at a tree               */
/*    Postcondition:     Make a tree empty                */
void EmptyTheTree(Tree *ptree);

#endif
```

接口实现文件：tree.c

```
/* ************************************************************
 *   Filename: tree.c
 *     Description:
 *          Version: 1.0
 *    Created: 2017/01/10
 *          Revision: none
 *          Compiler: gcc
 *             Author: Lv Hongbin
 *           Company: Shanghai JiaoTong Univerity
 * ************************************************************/
```

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include "tree.h"

static void TraverseTrnode(const Trnode *trnode, void(*pfun)(Item item));
static Pair SeekItem(const Item * item, const Tree *tree);
static Trnode *MakeNode(const Item *item);
static bool ToLeft(const Item *item1, const Item *item2);
static bool ToRight(const Item *item1, const Item *item2);
static void AddTrnode(Trnode * new_trnode, Trnode *root);

void InitializeTree(Tree *ptree){

    ptree->root=NULL;
    ptree->size=0;
}

bool    TreeIsEmpty(const Tree *ptree){

    if(ptree->size==0){
        return true;
    }else{
        return false;
    }
}


bool TreeIsFull(const Tree *ptree){

    Trnode *pt;
    pt=(Trnode *)malloc(sizeof(Trnode));
    if (pt==NULL){
        return true;
    }else{
        free(pt);
        return false;
    }
}

unsigned int TreeItemCount(const Tree *ptree){
```

```
    return ptree->size;
}

bool AddItem(const Item *item, Tree *ptree){


    if(TreeIsFull(ptree)){
        fprintf(stderr,"The tree is full\n");
        return false;
    }
    if(SeekItem(item,ptree).child!=NULL){
        fprintf(stderr,"the trnode with this item does exist\n");
                return false;
    }else{
        Trnode *new_trnode=MakeNode(item);
        if(new_trnode!=NULL){
            (ptree->size)++;
            if(ptree->size==1){
                ptree->root=new_trnode;
            }else{
                AddTrnode(new_trnode,ptree->root);
            }
            return true;
        }else{
            return false;
        }
    }
}

bool DeleteItem(const Item *item, Tree *ptree){

    Pair pair=SeekItem(item,ptree);
    Trnode *tem=pair.child;
    if(tem==NULL){
        fprintf(stderr,"the trnode with this item doesn't exist\n");
        return false;
    }else{
        if(pair.child->left!=NULL && pair.child->right!=NULL){
            if(ToLeft(item,&(pair.parent->item))){
                for(pair.child=pair.child->left;                pair.child->right!=NULL;
pair.child=pair.child->right){
                        continue;
```

```
                }
                pair.child->right=tem->right;
            }else{
                for(pair.child=pair.child->right;          pair.child->left!=NULL;
pair.child=pair.child->left){
                    continue;
                }
                pair.child->left=tem->left;
            }
        }else if(pair.child->right!=NULL){
            pair.child=pair.child->right;
        }else if(pair.child->left!=NULL){
                            pair.child=pair.child->left;
                }
        Trnode *tem1=tem;
        memset(tem1->item.petname,'\0',TSIZE);
        memset(tem1->item.petkind,'\0',TSIZE);
        tem1=NULL;
        free(tem);
        (ptree->size)--;
        return true;
    }
}


bool FindItem(const Item *item, Tree *ptree){

    return SeekItem(item,ptree).child==NULL?false:true;
}

void Traverse(const Tree *ptree, void(*pfun)(Item item)){

    if(ptree!=NULL){
        Trnode *trnode=ptree->root;
        TraverseTrnode(trnode,pfun);
    }
}

static void TraverseTrnode(const Trnode *trnode, void(*pfun)(Item item)){
    if(trnode!=NULL && trnode->item.petname[0]!='\0'){
        (*pfun)(trnode->item);
        TraverseTrnode(trnode->left,pfun);
```

```
        TraverseTrnode(trnode->right,pfun);
    }
}

void EmptyTheTree(Tree *ptree){

    Trnode *tem, *temleft, *temright, *nodeleft, *noderight;
    nodeleft=ptree->root->left;
    noderight=ptree->root->right;
    free(ptree->root);
    while(nodeleft!=NULL || noderight!=NULL){
        temleft=nodeleft;
        temright=noderight;
        free(nodeleft);
        free(noderight);
        nodeleft=temleft->left;
        noderight=temright->right;
    }
}

/*
static Pair SeekItem(const Item * item, const Tree *tree){

    Tree newtree=*tree;
    Pair pair;
    pair.parent=NULL;
    pair.child=tree->root;
    if(pair.child==NULL){
        return pair;
    }else if(pair.child!=NULL){
        if( pair.child->item==*item){
                                    return pair;
        }else if(pair.child->left!=NULL && pair.child->left->item==*item){
            pair.parent=pair.child;
                                    pair.child=pair.parent->left;
                                    return pair;
        }else if(pair.child->right!=NULL && pair.child->right->item==*item){
                                pair.parent=pair.child;
                            pair.child=pair.parent->right;
                                    return pair;
                    }else if(pair.child->left!=NULL && pair.child->left->item!=*item){
            newtree->root=pair.child->left;
```

```
                                    return SeekItem(item,&newtree);
        }else if(pair.child->right!=NULL && pair.child->right->item!=*item){
                                     newtree->root=pair.child->right;
                                    return SeekItem(item,&newtree);
                    }
}
*/


static Pair SeekItem(const Item *item, const Tree *tree){

    Pair pair;
    pair.parent=NULL;
    pair.child=tree->root;
    if(pair.child==NULL){
        return pair;
    }
    while(pair.child!=NULL){
        if(ToLeft(item,&(pair.child->item))){
            pair.parent=pair.child;
            pair.child=pair.child->left;
        }else if(ToRight(item,&(pair.child->item))){
                                    pair.parent=pair.child;
                    pair.child=pair.child->right;
                }else{
            break;
        }
    }
    return pair;
}



static Trnode *MakeNode(const Item *item){

    Trnode *new_trnode;
    new_trnode=(Trnode *)malloc(sizeof(Trnode));
    if(new_trnode==NULL){
        fprintf(stderr,"Fail: Out of memory\n");
        return NULL;
    }else{
        new_trnode->item=*item;
```

```
        new_trnode->left=NULL;
        new_trnode->right=NULL;
    }
    return new_trnode;
}


static bool ToLeft(const Item *item1, const Item *item2){

    int compareresult;
    compareresult=strcmp(item1 -> petname,item2 -> petname);
    if((compareresult=strcmp(item1 -> petname,item2 -> petname))<0){
        return true;
    }else if(compareresult==0 && strcmp(item1 -> petkind,item2 -> petkind)<0){
        return true;
    }else{
        return false;
    }
}

static bool ToRight(const Item *item1, const Item *item2){

        int compareresult;
        if((compareresult=strcmp(item1 -> petname,item2 -> petname))>0){
                return true;
        }else if(compareresult==0 && strcmp(item1 -> petkind, item2 ->
petkind)>0){
                return true;
        }else{
                return false;
        }
}


static void AddTrnode(Trnode * new_trnode, Trnode *root){

    if(ToLeft(&(new_trnode->item),&(root->item))){
        if(root->left==NULL){
            root->left=new_trnode;
        }else{
            AddTrnode(new_trnode,root->left);
        }
```

```
    }else if(ToRight(&(new_trnode->item),&(root->item))){
                    if(root->right==NULL){
                            root->right=new_trnode;
                    }else{
                            AddTrnode(new_trnode,root->right);
                    }

          }else{
        fprintf(stderr,"location error in AddTrnode()\n");
        exit(1);
    }
}
```

主文件：petclub.c

```
/* ************************************************************
 *   Filename: petclub.c
 *    Description:
 *         Version: 1.0
 *    Created: 2017/01/23
 *         Revision: none
 *         Compiler: gcc
 *            Author: Lv Hongbin
 *          Company: Shanghai JiaoTong Univerity
 * ************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include"tree.h"

void ShowListOfPet(Item item);
Item *InputPetInfo(Item *item);
char *s_gets(char *a,int N);

int main(){

    puts("\n/* ************************************");
    puts("\t--- Welcome to Lv Hongbin's pet club ---");
    char cSelect;
    Item pet;
    Tree tree;
    Pair pair;
    InitializeTree(&tree);
```

```
    while(1){
        puts("\nPlease select which command you want to execute:");
                printf("\t[a] Add a pet\t[l] List the pets\t[f] Find a pet\t[s] Show total
number\t[d] Delete a pet\t[q] Quit\n");
        cSelect=getchar();
        while(getchar()!='\n')
            continue;
        if(cSelect=='q'){
                            break;
        }else if(cSelect=='a'){
            InputPetInfo(&pet);
            if(AddItem(&pet,&tree)){
                        printf("Add the pet %s\t%s
succcessfully\n",pet.petname,pet.petkind);
            }else{
                printf("Add Fail!");
            }
        }else if(cSelect=='l'){
            if(!TreeIsEmpty(&tree)){
                Traverse(&tree,ShowListOfPet);
            }else{
                printf("The tree is empty!\n");
            }
        }else if(cSelect=='f'){
            if(!TreeIsEmpty(&tree)){
                InputPetInfo(&pet);
                if(FindItem(&pet,&tree)){
                    printf("The pet does exist\n");
                }else {
                    printf("The pet doesn't exist\n");
                }
            }
        }else if(cSelect=='s'){
            printf("The total number:%d\n",tree.size);
        }else if(cSelect=='d'){
            if(!TreeIsEmpty(&tree)){
                InputPetInfo(&pet);
                DeleteItem(&pet,&tree);
                        printf("Delete the pet %s\t%s
succcessfully\n",pet.petname,pet.petkind);
            }
        }else{
```

```
                printf("Please input the right letter!\n");
            }
        }
        return 0;
}

void ShowListOfPet(Item item){
        printf("The petname:%20s\tpetkind:%20s\n",item.petname,item.petkind);
}

Item *InputPetInfo(Item *item){

        char petname[TSIZE], petkind[TSIZE];
        printf("\nPlease input the pet name:");
            s_gets(item->petname,TSIZE);
             printf("Please input the pet kind:");
             s_gets(item->petkind,TSIZE);
        return item;
}

char *s_gets(char *a,int N){

        int i=0;
        if(fgets(a,N,stdin)!=NULL){
            for(i=0;(a[i]!='\n'&& a[i]!='\0');i++)
            {;}
            if(a[i]=='\n'){
                a[i]='\0';
            }else{
                while(getchar()!='\n')
                    continue;
            }
        }
        return a;
}
```

结果：

```
[lvhongbin@MiWiFi-R3G-srv ~/Desktop/cStudy]$ gcc tree.c petclub.c -o petclub.out
[lvhongbin@MiWiFi-R3G-srv ~/Desktop/cStudy]$ ./petclub.out

/* ************************************
        --- Welcome to Lv Hongbin's pet club ---
```

Please select which command you want to execute:
[a] Add a pet[l] List the pets [f] Find a pet[s] Show total number [d] Delete a pet [q] Quit
a

Please input the pet name:mike
Please input the pet kind:cat
Add the pet mike cat succcessfully

Please select which command you want to execute:
[a] Add a pet[l] List the pets [f] Find a pet[s] Show total number [d] Delete a pet [q] Quit
a

Please input the pet name:Lili
Please input the pet kind:cat
Add the pet Lili cat succcessfully

Please select which command you want to execute:
[a] Add a pet[l] List the pets [f] Find a pet[s] Show total number [d] Delete a pet [q] Quit
a

Please input the pet name:Bob
Please input the pet kind:dog
Add the pet Bob dog succcessfully

Please select which command you want to execute:
[a] Add a pet[l] List the pets [f] Find a pet[s] Show total number [d] Delete a pet [q] Quit
l
The petname: mike petkind: cat
The petname: Lili petkind: cat
The petname: Bob petkind: dog

Please select which command you want to execute:
[a] Add a pet[l] List the pets [f] Find a pet[s] Show total number [d] Delete a pet [q] Quit
s
The total number:3

Please select which command you want to execute:
[a] Add a pet[l] List the pets [f] Find a pet[s] Show total number [d] Delete a pet [q] Quit
d

Please input the pet name:Bob
Please input the pet kind:dog
Delete the pet Bob dog succcessfully

Please select which command you want to execute:
[a] Add a pet[l] List the pets [f] Find a pet[s] Show total number [d] Delete a pet [q] Quit
l
The petname: mike petkind: cat
The petname: Lili petkind: cat

Please select which command you want to execute:
[a] Add a pet[l] List the pets [f] Find a pet[s] Show total number [d] Delete a pet [q] Quit
f

Please input the pet name:mike
Please input the pet kind:cat
The pet does exist

Please select which command you want to execute:
[a] Add a pet[l] List the pets [f] Find a pet[s] Show total number [d] Delete a pet [q] Quit
f

Please input the pet name:Bob
Please input the pet kind:dog
The pet doesn't exist

Please select which command you want to execute:
[a] Add a pet[l] List the pets [f] Find a pet[s] Show total number [d] Delete a pet [q] Quit
q

1.1 我的感悟

其实我做了那么多的数据类型的学习，在设计一个新的 ADT 的时候，一般是先

从最小单位的属性组合开始，组成一个 item 结构体，然后重命名为类型 Item 类型。接着上升一个层次，与向后或者向前或向左右的相同层级联系，组合一个节点，节点是一个循环的最小单位，所以需要与相邻单元的联系，节点结构体 node 定义为 Node。最后再上升一个层级——整体，一般包含着这个 ADT 的基本信息，包括指向下一层级的起始指针，项数和其他主要特征。

二十、 网络套字节编程

19.4计算机网络基础

19.5未完待续，后面跟《计算机网络原理》一起看