The C Programing Language

# 背景

本工程主要是为了实践《The C Programing Language》的代码。

# Why C?

1. 学习底层知识的基础。我们无论是学习《深入理解计算机技术》，还是学习《The Linux programming interface》,都需要C语言打底；

2. 我们要参与C语言相关的开源项目，比如nginx/curl/redis这些，C语言基础是必须的；

3.

# 版本管理

已经放到github上去了：

|  |
| --- |
| https://github.com/zhoushuo19850509/CProgramingLanguage |

后续提交步骤：

|  |
| --- |
| git add \*  git commit -m “XXX”  git push origin main |

目前是上传到main分支，这是现在github 的默认分支

本地代码目录：

|  |
| --- |
| /Users/zhoushuo/Documents/workspace/testGit/CProgramingLanguage |

# 阅读感受

## 深入浅出

每章末尾一般会有一个规模较大的程序，比如calculator.c啥的，还是需要花费很多精力去理解、去调试的。但是每章开头的基本概念还是好掌握的，这就是所谓的深入浅出吧。

很多经典的书籍，都是类似的套路：掌握基础概念相对方便，但内容也有一定的深度。比如《Thinking in Java》这类的，即便你对java很熟悉了，有十年开发经验，但是书中还是有很多有深度的代码，尤其是Concurrency这章，内容非常丰富。

## 保持状态

和绝大多数编程语言、体育项目一样，C语言要用得熟练，也需要保持状态。尤其是本书的内容，都是环环相扣的：比如chap5/content-sort-v2.c，用来说明pointer to function的意思，引用的头文件如下：

int readlines(char \*lineptr[], int maxlines);

void writelines(char \*lineptr[], int nlines);

char \*alloc(int n);

int getlineA(char line[], int lim);

int numcmp(char \*s, char \*t);

int strcmpv2(char \*s, char \*t);

void swap(void \*v[], int i, int j);

void qsort(void \*v[], int left, int right,

int (\*comp)(void \*, void \*));

double atof(char s[]);

这些头文件是我们之前在各个章节实践过的。如果我们隔了很长时间，去看content-sort-v2.c这个代码文件，即便搞清楚了content-sort-v2.c本身的逻辑，但可能已经忘记了引用的这些函数是怎么实现的了。如果回过头要把这些函数一个个搞明白，那就太费劲了。还有就是这些函数还引用了库函数，比如isdigit()，你如果很久没有实践C语言，可能搞不懂这个函数是在哪个库文件中。

解决方法，一是一气呵成，在一段时间内(比如五一假期)把本书各个例子都敲一遍，形成一个完整的体系、完整的闭环；二是要经常练手，这样才能保持状态、保持熟练；三是找一个能够实践的项目，比如能否参与一个C语言开发的项目；四是找某个研究主题，比如研究linux下epoll的使用方法、实现原理、应用场景，也可以用到C语言。

所以，我们与其说C语言是一种能力，不如说C语言是一种状态、一种Context，一种世界观。

## 精炼的语言本身和庞大的生态体系

我们实践下来，到了chap5快结束了，我们看到C语言本身没有什么神秘的，语法也不多、语法糖也很少，但是如果把C语言和算法结合起来，和Linux/UNIX系统接口开发结合起来，和那些C语言体系的开源软件结合起来(redis/nginx/ffmpeg)，那是非常博大精深的。

简洁的语言本身和庞大的生态体系，这种对比反差，有一种特别的高级感。有点像100米：但凡你有腿，就能跑100米，但要参加比赛，就必须有强健的体魄、科学体系的训练、夜以继日地保持状态；也有点像说相声：但凡有嘴就能说，但你要上台表演，没有数十年的功力，那是没法气定神闲在台上说的。

所以，C语言怎么形容呢，就是门槛很低，但同时门槛又很高。

## C语言和底层联系紧密

C语言的特点：和底层联系非常紧密。体现在哪里呢？

1.C语言编程开发的核心：数组、指针，数组、指针对应的就是内存模型；

2.C语言能够无缝调用Linux/UNIX发布的系统接口，比如read/write这些系统接口，能够直接捕获Linux信号量；

# Chap1 Introduction

第一章介绍了C语言的一些基本功能，包括：

## 变量/算数运算

代码示例：

|  |
| --- |
| Temperature.c |

## Char input/output

代码示例：

|  |
| --- |
| Copy.c  char\_count.c  count\_word.c  line\_count.c |

## Function

代码示例：

|  |
| --- |
| function\_power.c |

## Array

代码示例：

|  |
| --- |
| count\_digits.cl |

## Char array

代码示例：

|  |
| --- |
| print\_longest\_line.c |

## Extern variable

代码示例：

|  |
| --- |
| print\_longest\_line\_v2.c |

# chap2 types operations and expressions

## 2.1 variable names

## 2.2 data types and sizes

## 2.3 constants

|  |
| --- |
| strlen.c |

## 2.4 declaratioins

## 2.5 arithmetic operations

## 2.7 type conversions

|  |
| --- |
| // 字符串转int  atoi.c  // 改小写  lower.c  // 生成伪随机数  randdom.c |

## 2.8 increment and decrement operators

|  |
| --- |
| //  squeeze.c  // 字符串拼接  strcat.c |

## 2.9 bitwise operators

## 2.10 assignment operators and expressions

|  |
| --- |
| bitcount.c |

## 2.11 conditional expressions

## 2.12 precedence and order of evaluation

## 2.13 summary

本章内容比较基础，代码也较少

# chap3 control flow

## 3.1 statement and blocks

## 3.2 if-else

## 3.3 else-if

搜索算法

|  |
| --- |
| binsearch.c |

## 3.4 switch

统计数字、空格

|  |
| --- |
| countdigit.c |

## 3.5 loops while and for

字符转数字

|  |
| --- |
| atoi.c |

排序算法

|  |
| --- |
| shellsort.c |

翻转

|  |
| --- |
| reverse.c |

## 3.6 loops do-while

|  |
| --- |
| // 数字转字符  itoa.c |

## 3.7 break and continue

去掉外围的空格

|  |
| --- |
| trim.c |

## 3.8 goto and labels

## summary

这章的代码，相对较少，没有什么大型的程序

# chap4 functions and program structure

## 4.1 basic of function

字符串匹配

|  |
| --- |
| /\*\*  \* 功能： 字符串匹配  \* 目标： 为了说明function的用法  \*/  finds-matching.c |

## 4.2 functions returning non-integer

函数返回值不是int类型

字符串转浮点数

|  |
| --- |
| /\*\*  \* 这个代码的功能： atof() 字符串转float number  \* 为了说明function可以返回非int类型的元素  \*/  double atof(char s[]) |

## 4.3 external variables

实现一个计算器

|  |
| --- |
| /\*\*  \* 功能： 实现一个计算器  \* 目标： 1.为了说明external variables的用法  \* 2.为了说明Header files的用法  \*/  calculator.c |

作为本章最重要的代码，我们要好好总结一下。

这个代码我们已经实现过很多遍了，功能是非常明确的，就是实现计算器的功能。

这个代码想要说明C语言什么功能呢？

1.external variables

2.header files

3.缓存的使用

这个代码案例的收获：

1.通过这个例子，我们学习了如何通过C语言读取、解析一段数字、字符混合的内容，有点编译原理那么点意思了。

2.通过这个例子，我们掌握了stack这个基础的数据结构。有点数据结构和算法的思想萌芽了；

3.通过这个例子，我们清楚地了解header file的作用，就是把代码工程用到的各个方法整合到一个Header file，有点interface的那个感觉了；

4.

## 4.4 scope rules

## 4.5 header files

## 4.6 static variables

## 4.7 register variables

## 4.8 block structure

## 4.9 intializations

## 4.10 recursion

|  |
| --- |
| /\*\*  \* 功能： 把一个数字以char的形式一个个字符打印出来  \* 目标： 为了说明嵌套的用法  \*/  printd()  /\*\*  \* 代码功能： 快排的方式，进行排序  \* 目标：为了说明嵌套的用法  \*/  qsort() |

## 4.11 the c preprocessor

## summary

这章的核心就是那个计算器程序，这个程序规模还挺大的

我们囫囵吞枣把这章的代码都实践了，后续写一下课后作业、把本章内容再仔细看看。

# chap5 pointers and arrays

## 5.1 pointers and address

## 5.2 pointers and function arguments

|  |
| --- |
| /\*\*  \* 功能： 交换两个int类型的变量  目标： 为了说明指针作为函数参数的场景  \*/  swap.c  /\*\*  \* 功能： getint()函数从标准输入读取一个个字符，  如果这些字符组成的是数字，就把数字保存在getint()的参数： \*pn中  \*pn是一个指向int类型的指针  目标： 为了说明指针作为函数参数的场景  \*/  getint.c |

## 5.3 pointers and arrays

|  |
| --- |
| /\*\*  \* 功能：计算string长度  \* 目标：为了说明指针和数组的关系  \*/  strlen.c |

## 5.4 address arithmetic

申请内存、释放内存

|  |
| --- |
| memory-alloc.c  包含两个函数：  alloc()  afree() |

## 5.5 character poniters and functions

|  |
| --- |
| /\*\*  \* 功能： 字符串比较（数组实现）  \* 目标： 为了说明数组方式实现字符串比较  \*/  strcmp-v1.c  /\*\*  \* 功能： 字符串比较（指针实现）  \* 目标： 为了说明指针移动的功能  \*/  strcmp-v2.c |

## 5.6 pointer arrays;pointer to pionter

|  |
| --- |
| /\*\*  \* 功能： 读取输入的一个文本，按照行进行排序  \* 目标： 为了说明pointer array的意思。  pointer array本质上是一个数组，  数组中每个元素是一个char \*(指向char的指针，其实说白了就是string)  所以pointer array本质上就是sting array  \*/  content\_sort.c |

## 5.7 multi-dimensional arrays

|  |
| --- |
| /\*\*  \* 本代码的功能： 提供两个函数  函数1 day\_of\_year() yyyymmdd -> day of year(某年中的一天)  函数2 month\_day() day of year(某年中的一天) -> yyyymmdd  \* 目标：为了说明多维数组的使用方法  \*/  date\_compute.c |

## 5.8 initialization of pointer arrays

## 5.9 pointer vs multi-dimensional arrays

## 5.10 command-line arguments

这个小节为了说明如何处理main函数的参数

|  |
| --- |
| int main(int argc, char \*argv[]){ ... } |

|  |
| --- |
| /\*\*  \* 功能： 实现类似echo的功能  \* 目标： 为了说明main argument的用法(数组法)  \*/  echo-demo-v1.c  /\*\*  \* 功能： 实现类似echo的功能  \* 目标： 为了说明main argument的用法(遍历指针法)  \*/  echo-demo-v2.c  /\*\*  \* 功能： 模拟find命令： find aaa xxx  备注： 功能和chap4/finds-matching.c 功能类似，  就是打印、统计匹配关键字的行  \* 目标： 为了说明main argument的用法  \*/  find-demo.c  /\*\*  \* 功能： 模拟find命令： find -x -n aaa xxx  备注： 功能和find-demo.c类似，就是打印、统计匹配关键字的行  不同的是，支持参数，  -x 是反向匹配的意思 比如原来匹配的就不匹配了，原来不匹配的现在匹配了  -n 意思是打印匹配的行号  \* 目标： 1.为了说明main argument的用法  2.为了介绍linux shell命令的解释过程  \*/  find-demo-with-param.c |

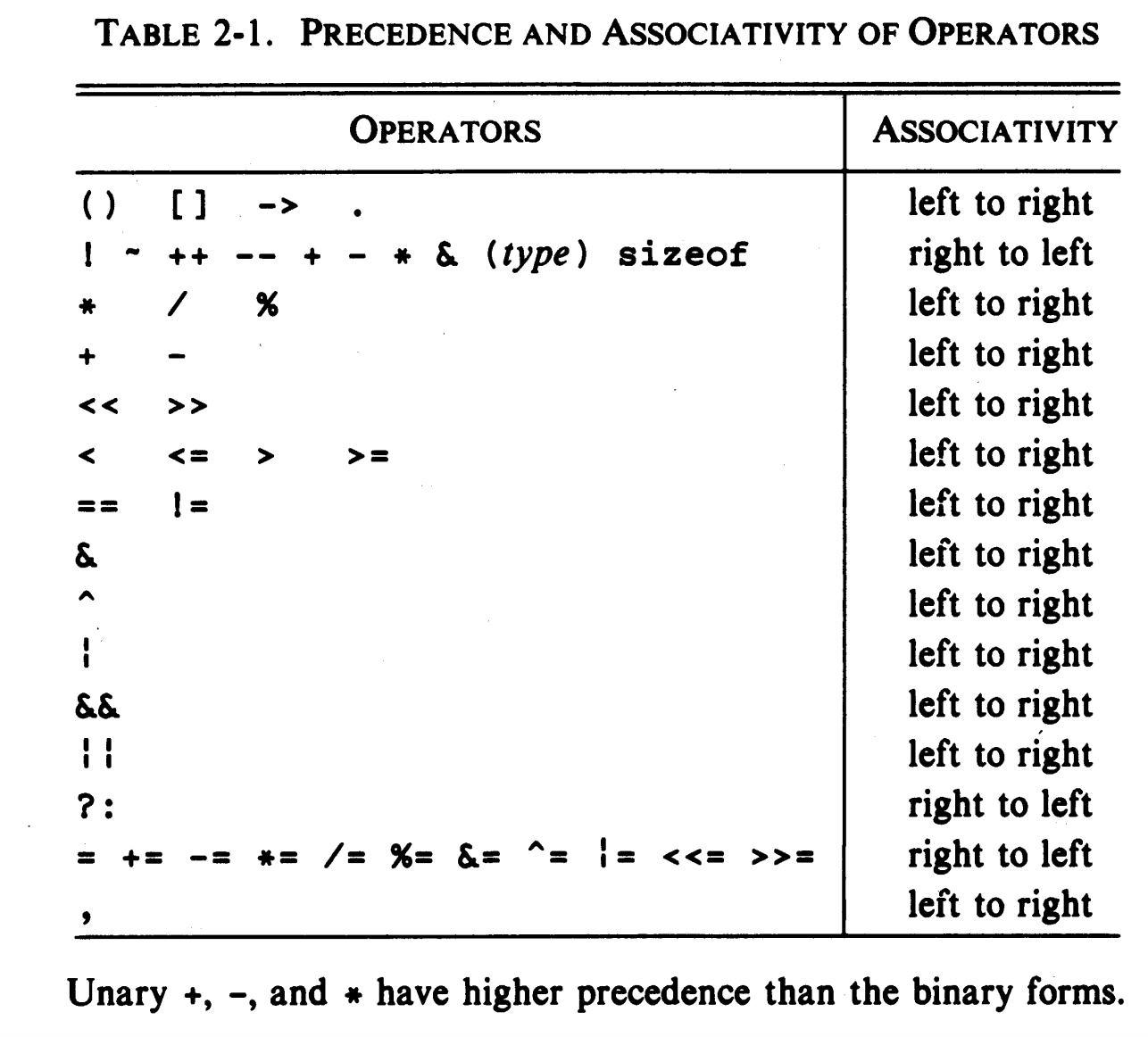
这个小节的核心就是find-demo-with-param.c中两个非常容易混淆的地方：

(\*++argv)[0]

\*++argv[0]

这两个是非常容易混淆的我们结合操作符优先级详细说明一下：

操作符优先级参考2.12 precedence and order of evaluation



比如

(\*++argv)[0]

\*++argv[0]

这两个是非常容易混淆的

如果没有括号， \*++argv[0]

那么优先级是[] > ++ > \*

所以\*++argv[0]实际上就是 ：

\*++(argv[0])

举例来说，我们的main函数argv为： find -xn

(1)第一个while循环中，(\*++argv)[0]的意思是argv指针先往前进一个，然后取第一个字符，也就是

-x的第一个字符： '-'

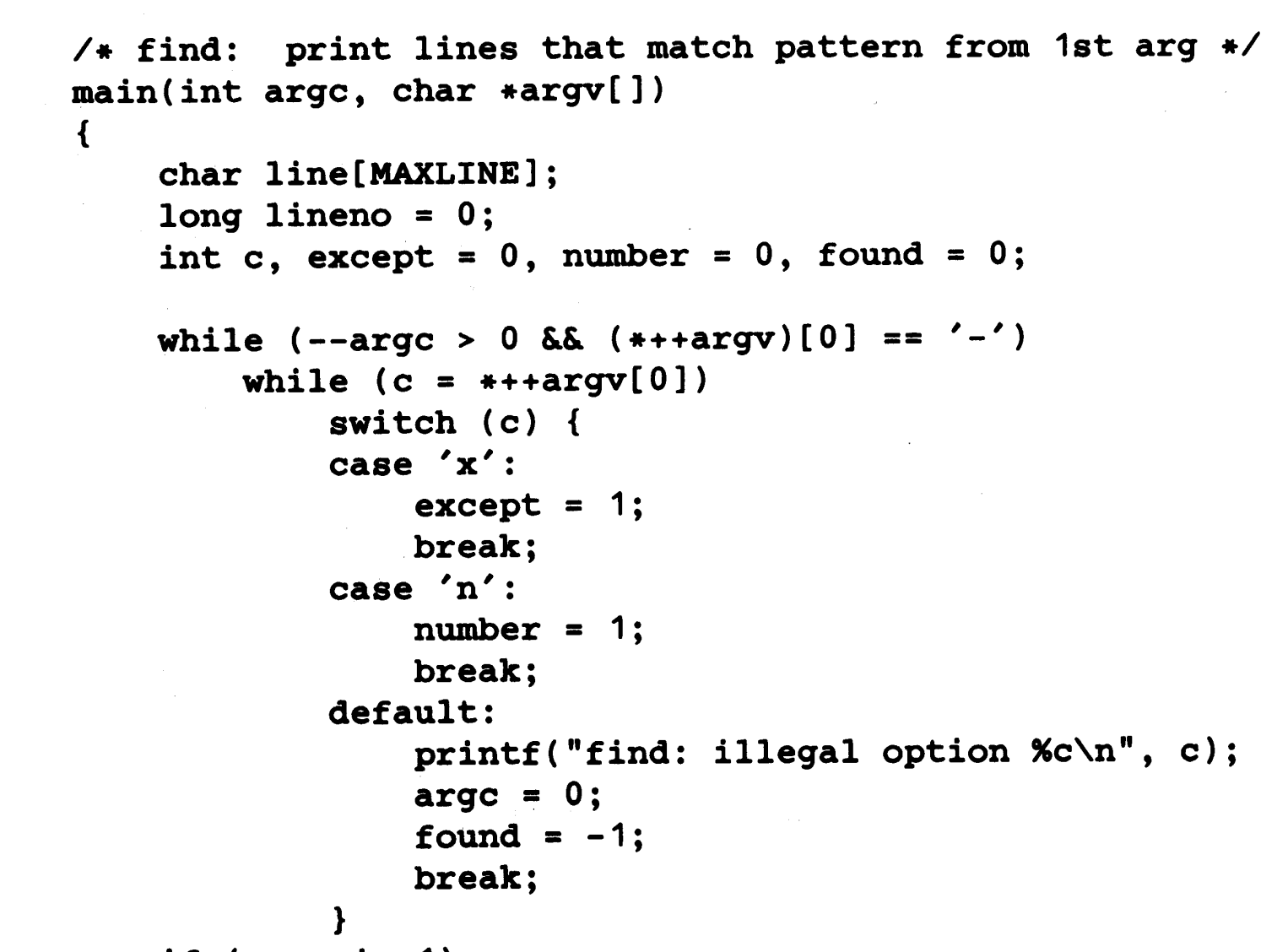
(2)第二个while循环中\*++argv[0]的意思是argv[0]也就是-x，然后执行++，就是前进一个位置，也就是-x的第二个字符： 'x'

(3)如果出现命令行参数是-xn 那么第二个while循环会读取到'n'

所以，整体的效果是：

一、(\*++argv)[0]遍历各个参数第一个字符，一般是'-'

二、\*++argv[0]检测参数'-'后面的内容，比如find -xn 中的xn，都能够检测到



总结一下 char \*argv[]这个argv就很神奇：

1.argv本质上是一个指针，这个指针指向一个array，这个array就是pointer array。

2.这个array中的每个元素，都是char \* 也就是指向char的指针，本质上就是string了。

3.我们可以通过argv++，遍历这个pointer array

## 5.11 pointers to functions

|  |
| --- |
| /\*\*  \* 功能： 读取输入的一个文本，每行按照数字的形式进行排序  \* 目标： 为了说明pointer to function的意思。  啥意思呢？funciton name本身就是指向这个funciton的指针  我们可以把function name当做一个指针，作为某个函数的参数。  为啥要把function作为函数的参数呢？这个代码就告诉你为什么。  因为根据不同情况，要调用不同的函数。  这有点java中的interface-implement的味道了  所谓的interface，在这里就是函数的定义，比如numcmp函数和strcmp  除了函数名称不同，其他的参数、返回值都完全一样：  int numcmp(char \*s, char \*t);  int strcmpv2(char \*s, char \*t);  所谓的implement,就是numcmp/strcmpv2这两个函数各自的实现逻辑  \*/  content-sort-v2.c |

具体效果是什么样的呢？

比如一下的内容：

|  |
| --- |
| 123.45  99  8  12000  16  7 |

如果按照string sorting的方式，排序结果是这样的：

./a.out < a.txt

|  |
| --- |
| 12000  123.45  16  7  8  99 |

而如果按照number sorting的方式，排序结果是这样的：

./a.out -n < a.txt

|  |
| --- |
| 7  8  16  99  123.45  12000 |

这个意思就很清楚了吧

## 5.12 complicated declarations

|  |
| --- |
| declaration\_parser.c |

## 5.13 Summary

本章内容很重要，代码比较长的是最后一节declaration\_parser.c

# chap6 structures

## 6.1 basics of structure

## 6.2 structure and function

通过rectangle操作，展示structure的基本用法

|  |
| --- |
| rect-demo.c |

## 6.3 array of structure

统计代码关键字的数量，用来说明structure array的用法

|  |
| --- |
| keyword-count.c |

## 6.4 pointer ot structure

统计代码关键字的数量，用来说明structure point的用法

|  |
| --- |
| keyword-count-v2.c |

## 6.5 self-referential structure

统计各个单词出现的次数，通过一个binary tree的数据结构进行处理

用来说明self-referential structure的用法

|  |
| --- |
| word-frequency-count.c |

## 6.6 table lookup

通过一个structure list这个数据结构，实现hash table

|  |
| --- |
| hash-table.c |

## 6.7 typedef

## 6.8 union

## 6.9 bit-field

## 6.10 summary

本章介绍了structure结构体，综合了之前的pointer/array，通过structure介绍了算法和数据结构的基础，是本书中最重要的一章。

# chap7 intput and output

## 7.1 standard input and output

把输入的内容转化为小写

|  |
| --- |
| printlower.c |

## 7.2 formatted output -printf

## 7.3 variable -length argument list

模拟printf的功能，说明variable -length argument list的用法

|  |
| --- |
| minprintf.c |

## 7.4 formatted input - scanf

## 7.5 file-access

代码功能：

模拟linux cat命令的用法

cat file1 file2 file3 ...

会把所有的文件内容都打印出来

|  |
| --- |
| concatenate-files.c |

用来说明文件访问函数fopen的用法

## 7.6 error handling - stderr and exit

|  |
| --- |
| concatenate-files-v2.c |

用来说明异常处理

## 7.7 line input and output

fgetc()函数的getline()方法中的作用，意思是如何逐行读取文件内容

|  |
| --- |
| getline.c |

## 7.8 miscellaneous functions

这个小节介绍了各种C语言库函数的用法，

比如string /math/ malloc/ random啥的，都是必须要掌握的

## 7.9 summary

这章介绍IO处理的基础概念，表扩基本的输入输出函数scanf/printf,包括文件的读取、写入、异常处理。但是本章没有什么规模很大的代码

# chap8 unix interface

这章主要是介绍unix/linux提供的系统调用函数，比如read/write

备注：在mac下直接使用read/write这些系统函数是有问题的，我们通过引入如下的头文件实现了库文件的引用，具体参考

gitee/The Linux programming interface

#include "tlpi\_hdr.h"

但是，这个头文件也不是万能的，也只能说是引入了部分的unix接口。所以，要实践本书内容，最好还是在linux 中实践，比如ubuntu/oracle linux啥的。

## 8.1 file descriptors

## 8.2 low level I/O read and write

把标准输入转到标准输出，为了说明Linux 接口read/write的用法

|  |
| --- |
| readwrite.c |

通过模拟getchar()方法，说明read的用法

|  |
| --- |
| getchar.c |

## 8.3 open , create, close ,unlink

文件拷贝，用于说明open()/create()/read()这些linux接口的用法

|  |
| --- |
| copy.c |

## 8.4 random access - lseek

功能：读取n个字节

目标：为了说明lseek这个Linux接口的用法

|  |
| --- |
| get.c |

## 8.5 exampel - an implementation of Fopen and Getc

模拟fopen()的功能，功能就是打开某个文件

目标：也是为了说明open()/create()/read()这些linux接口的用法

|  |
| --- |
| fopen.c |

功能：申请一段内存，作为file input buffer缓存

目标：也是为了说明malloc()/read()这些linux接口的用法

|  |
| --- |
| fill\_buffer.c |

## 8.6 exampel - listing directories

功能：能够(嵌套)遍历某个目录下所有的文件，打印这些文件的filesize

目标：为了说明open/malloc这些Linux系统接口

|  |
| --- |
| print-file-size.c |

## 8.7 example - a storage allocator

功能：实现malloc/free这些内存分配功能

目标：为了说明sbrk这个Linux系统接口在内存分配中的作用。

|  |
| --- |
| storage-allocator.c |

## 8.8 summary

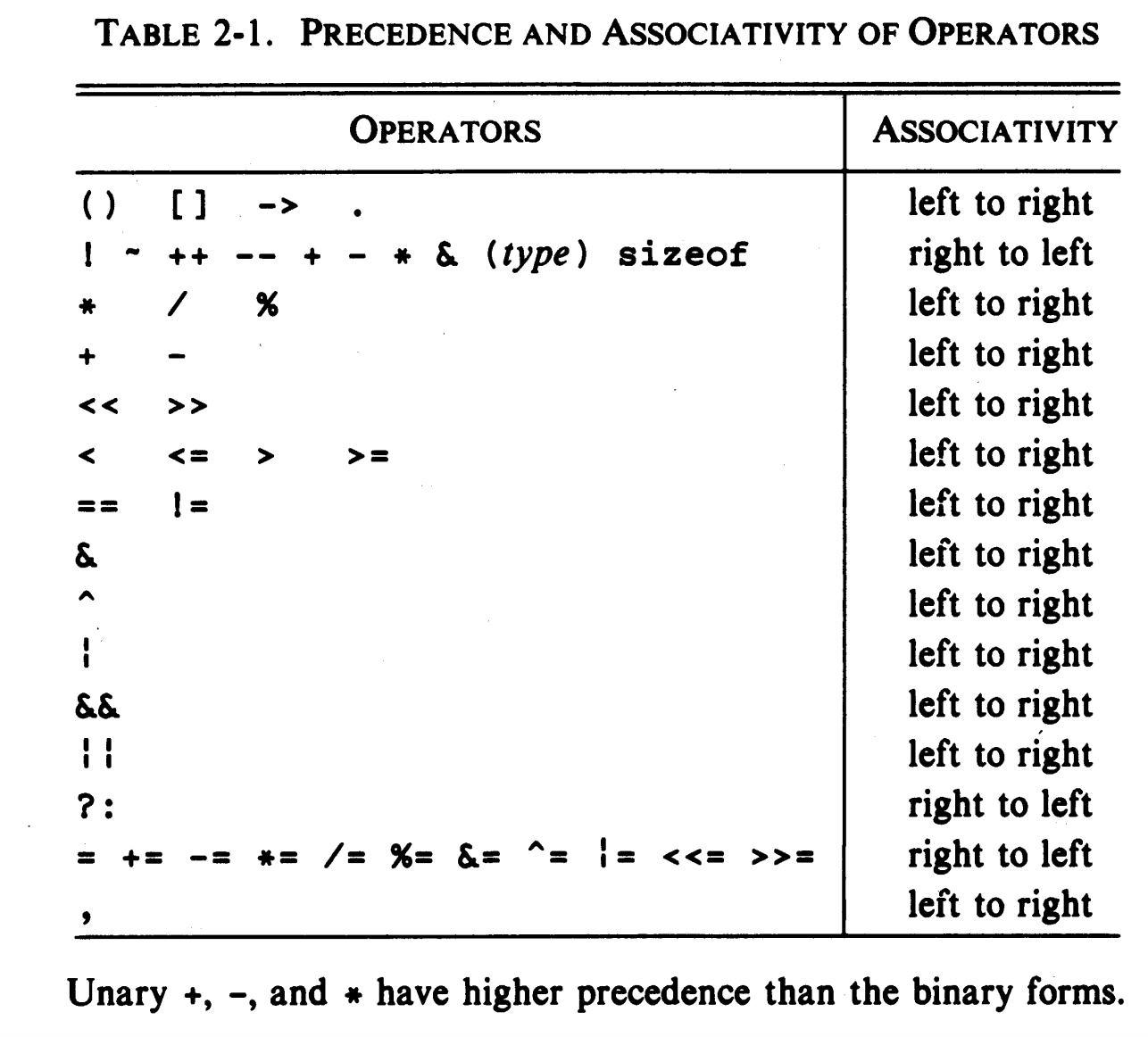
本章介绍了C语言如何利用UNIX/Linux接口read/open/lseek/sbrk这些系统接口，实现一些文件访问、目录遍历、内存申请等功能。代码还是很多很复杂的，需要好好体会。

为啥说C语言是底层的高级语言呢？原因就在这里：可以直接调用Linux开放的系统接口

# 重点内容汇总

## 1.操作符优先级

操作符优先级参考2.12 precedence and order of evaluation



比如

(\*++argv)[0]

\*++argv[0]

这两个是非常容易混淆的

如果没有括号， \*++argv[0]

那么优先级是[] > ++ > \*

所以\*++argv[0]实际上就是 ：

\*++(argv[0])

举例来说，下面这段代码是5.10 command-line arguments 这个小节

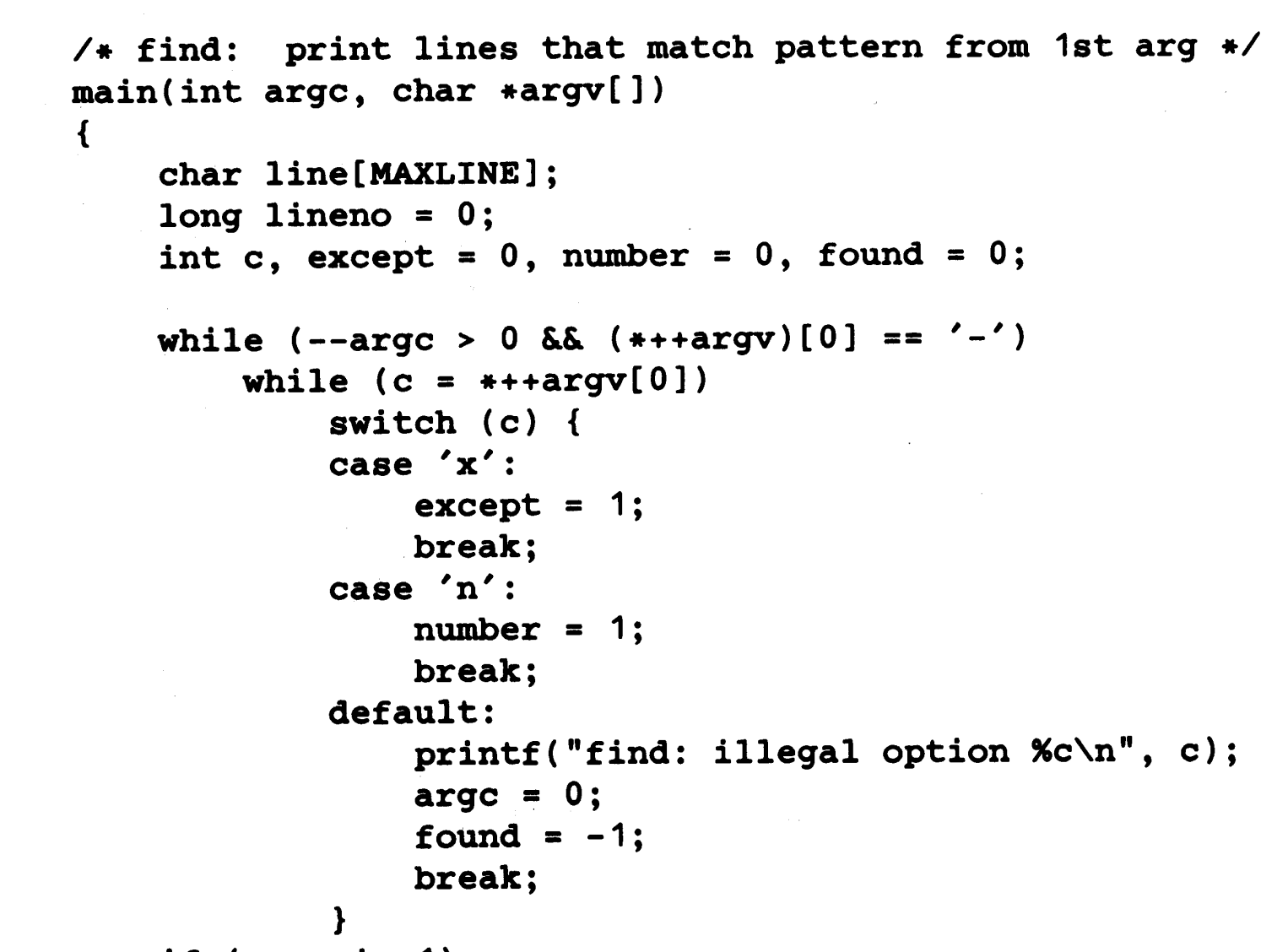
我们的main函数argv为： find -xn

(1)第一个while循环中，(\*++argv)[0]的意思是argv指针先往前进一个，然后取第一个字符，也就是

-x的第一个字符： '-'

(2)第二个while循环中\*++argv[0]的意思是argv[0]也就是-x，然后执行++，就是前进一个位置，也就是-x的第二个字符： 'x'

(3)如果出现命令行参数是-xn 那么第二个while循环会读取到'n'



总结一下 char \*argv[]这个argv就很神奇：

1.argv本质上是一个指针，这个指针指向一个array，这个array就是pointer array。

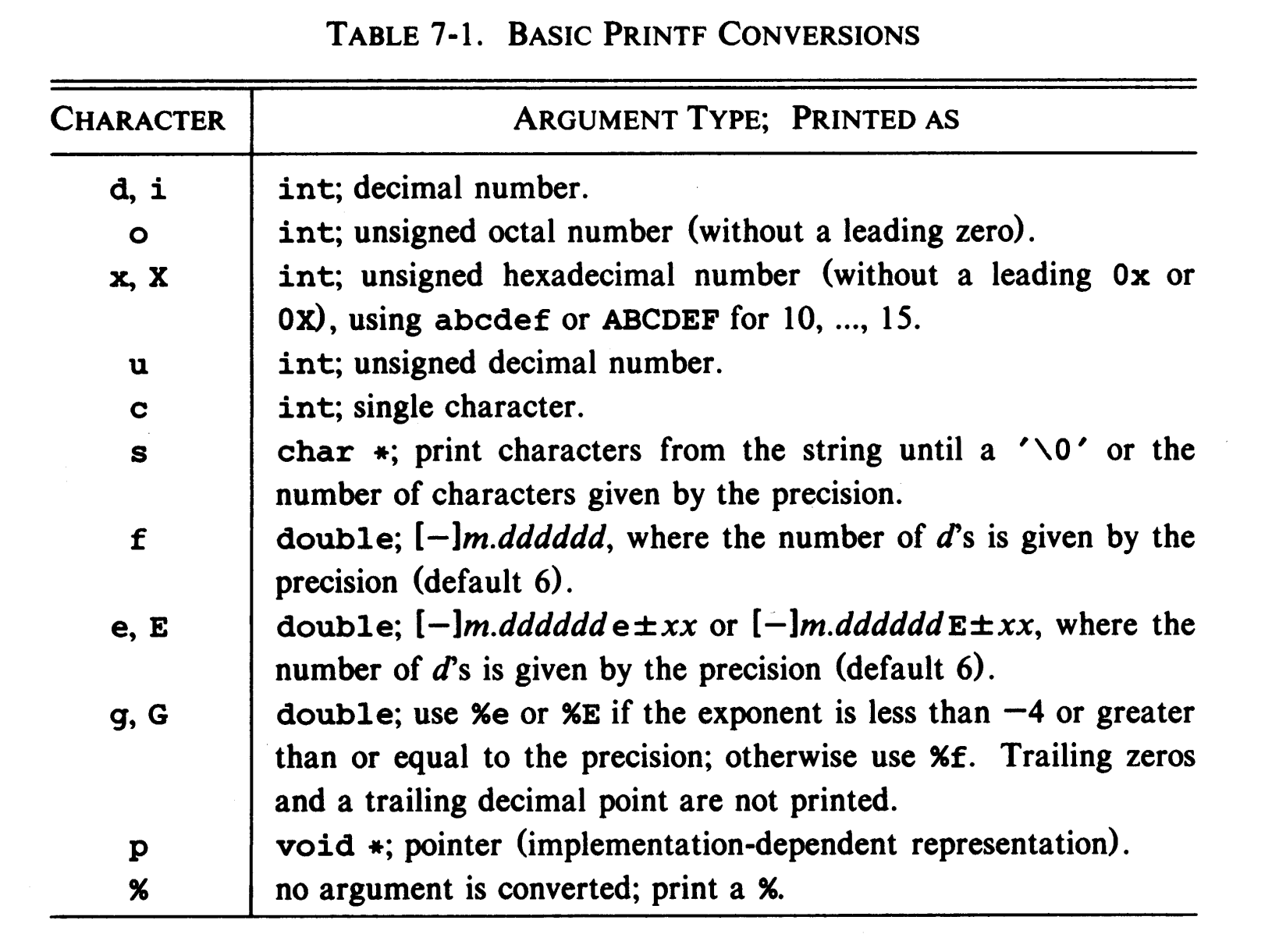
2.这个array中的每个元素，都是char \* 也就是指向char的指针，本质上就是string了。

3.我们可以通过argv++，遍历这个pointer array

## printf打印格式

这个也是老生常谈了，几乎在每个代码中都会用到

具体格式参考 7.2 formatted output - printf



# 待学习

## GDB调试

这个我们已经碰到过很多次了，代码编译没问题，但是执行出错。我们目前使用printf打印各个阶段的数据，但是这个方法也太low了，而且效率很低。后续我们还是要学会GDB调试，提升我们调试效率。

# 总结