程序设计笔记

上官凝

2020 年 8 月 19 日

目录

第 1 章　C 语言进阶、巩固和补充

1.1　输入输出

`scanf()`　第一个参数是格式字符串，后面的参数是变量的地址，函数作用是按照第一个参数指定的格式，将数据读入后面的变量。返回值：

1. `>0`：成功读入的数据项个数
2. `0`：没有项被赋值
3. `EOF`：第一个尝试输入的字符是 EOF（结束）

`printf()`　第一个参数是格式字符串，后面的参数是待输出的变量，函数作用是按照第一个参数指定的格式，将后面的变量在屏幕上输出。返回值：

1. ≥ 0：成功打印的字符数
2. < 0：出错

格式字符串的格式控制符号　如下表所示

表 1.1: 格式控制符号

符号	含义
% d	读入或输出 `int` 变量
% c	读入或输出 `char` 变量
% f	读入或输出 `float` 变量
% s	读入或输出 `char *` 变量
% lf	读入或输出 `double` 变量
% e	以科学计数法格式输出数值
% x	以十六进制读入或输出 `int` 变量
% p	输出指针地址值
% .5lf	输出浮点数，精确到小数点后 5 位

1.2 函数指针

函数指针可以像一般函数一样，用于调用函数、传递参数。在如 C 这样的语言中，通过提供一个简单的选取、执行函数的方法，函数指针可以简化代码。函数指针只能指向具有特定特征的函数。因而所有被同一指针运用的函数必须具有相同的参数和返回类型。与引用数据值相反，函数指针指向内存中的可执行代码。因为指向函数的指针实际上包含了函数的入口点的内存地址，所以赋给指针变量的地址就是函数的入口地址，从而该指针就可以用来代替函数名。这一特性也使得函数指针可以作为参数传递给其他函数。由此可见，可以定义一个指向函数的指针变量，它可以被处理，如传递给函数，或放置在数组中等等。

1.2.1 函数指针的定义

定义一个函数指针，基本格式与定义一个普通指针类似。

普通指针的声明：`int *pa;`　　函数指针的声明：`double (*func1)(double);`

普通指针的赋值：`int *pa = &a;`

函数指针的赋值：`double (*func1)(double) = myfunc;`，在这里，`myfunc`是另外定义的一个函数 `double myfunc(double x);`，需要在函数指针使用之前预先声明或定义。

应注意的是，函数名本身也是一个 const 指针，正如一个数据变量的 & a 一样，所以在使用的时候也可以直接将函数名传递给调用者函数

1.2.2 函数指针的应用

常见的函数指针的应用是`qsort()`（快排函数）排序就是一个不断比较并交换位置的过程。

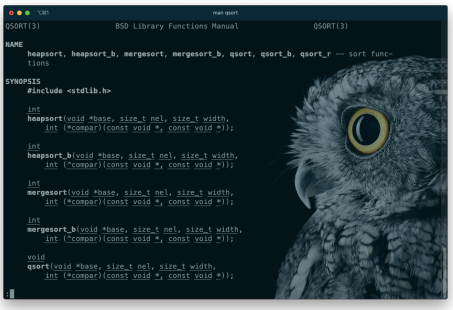

图 1.1: `qsort()` Family

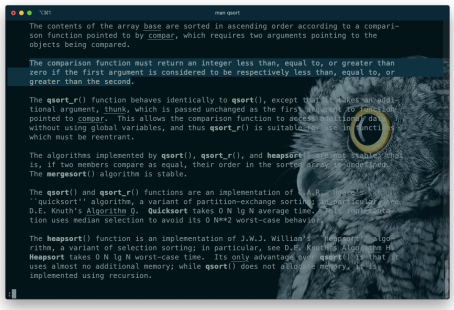

图 1.2: Usage Description

`qsort()`如何在连元素的类型是什么都不知道的情况下，比较两个元素并判断哪个应该在前呢？答案是，`qsort()`函数在执行期间，会通过`compar`指针调用“比较函数”，调用时将要比较的两个元素的地址传给“比较函数”，然后根据“比较函数”返回值判断两个元素哪个更应该排在前面。这个“比较函数”不是 C/C++ 的库函数，而是由使用`qsort()`的程序员编写的。在调用`qsort()`时，将“比较函数”的名字作为实参传递给`compar`。程序员当然清楚该按什么规则决定哪个元素应该在前，哪个元素应该在后，这个规则就体现在“比较函数”中。

比如下面这个简单的例子

```
typedef struct
{
    int number;
    int score;
} stu;

int self_cmp(const void * a, const void * b)
{
    stu * _a = (stu *) a;
    stu * _b = (stu *) b;
    if (*_a->score != *_b->score)
    {
        return *_b->score - *_a->score; // 从大到小排序
    }
    else
        return *_a->number - *_b->number;
}
// 在主函数中调用qsort(array, array_lenth, sizeof(stu), self_cmp);
// 这样就是先按照成绩降序排列，成绩相等时按照编号升序排列
```

1.3 动态内存分配

众所周知，在经典 C 语言中，使用`malloc()`函数来进行堆空间分配。而 C++ 提供了**new运算符**来处理这个事情。new运算符可以分配一个数据，或者一个数组。此表达式也具有返回值，即若返回值为 NULL，则空间分配失败（不是说 lazy 策略，不使用就不实际分配吗...？）。配合`delete()`使用，否则会造成内存空间泄漏

1.4 命令行参数

主函数新写法：

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

可以参考一下这篇文章。

1.5 一些库函数

1.5.1 数学函数

在`math.h`中声明

表 1.2: 常用的数学函数

函数	功能
abs(x)	求整型数 x 的绝对值
cos(x)	求 x(弧度) 的余弦
fabs(x)	求浮点数 x 的绝对值
ceil(x)	求不小于 x 的最小整数
floor(x)	求不大于 x 的最小整数
log(x)	求 x 的自然对数
log10(x)	求 x 的对数 (底为 10)
pow(x,y)	求 x 的 y 次方
sin(x)	求 x(弧度) 的正弦
sqrt(x)	求 x 的平方根

1.5.2 字符处理函数

在`ctype.h`中声明

表 1.3: 字符处理函数

函数	功能
int isdigit(int c)	判断 c 是否是数字字符
int isalpha(int c)	判断 c 是否是一个字母
int isalnum(int c)	判断 c 是否是一个数字或字母
int islower(int c)	判断 c 是否是一个小写字母
int islower(int c)	判断 c 是否是一个小写字母
int isupper(int c)	判断 c 是否是一个大写字母
int toupper(int c)	如果 c 是一个小写字母，则返回其大写字母
int tolower(int c)	如果 c 是一个大写字母，则返回其小写字母

1.5.3 字符串和内存操作函数

在`string.h`中声明

1.5.4 字符串转换函数

在`stdlib.h`中声明

表 1.4: 字符串和内存操作函数

函数	功能
char * strchr(char * s, int c)	如果 s 中包含字符 c，则返回一个指向 s 第一次出现的该字符的指针，否则返回 NULL
char * strstr(char * s1, char * s2)	如果 s2 是 s1 的一个子串，则返回一个指向 s1 中首次出现 s2 的位置的指针，否则返回 NULL
char * strlwr(char * s)	将 s 中的字母都变成小写
char * strupr(char * s)	将 s 中的字母都变成大写
char * strcpy(char * s1, char * s2)	将字符串 s2 的内容拷贝到 s1 中去
char * strncpy(char * s1, char * s2,int n)	将字符串 s2 的内容拷贝到 s1 中去，但是最多拷贝 n 个字节。如果拷贝字节数达到 n，那么就不会往 s1 中写入结尾的 '\0'
char * strcat(char * s1, char * s2)	将字符串 s2 添加到 s1 末尾
int strcmp(char * s1, char * s2)	比较两个字符串，大小写相关。如果返回值小于 0，则说明 s1 按字典顺序在 s2 前面；返回值等于 0，则说明两个字符串一样；返回值大于 0，则说明 s1 按字典顺序在 s2 后面
int stricmp(char * s1, char * s2)	比较两个字符串，大小写无关。其他和 strcmp 同
void * memcpy(void * s1, void * s2, int n)	将内存地址 s2 处的 n 字节内容拷贝到内存地址 s1
void * memset(void * s, int c, int n)	将内存地址 s 开始的 n 个字节全部置为 c

表 1.5: 字符串转换函数

函数	功能
int atoi(char *s)	将字符串 s 里的内容转换成一个整型数返回
double atof(char *s)	将字符串 s 中的内容转换成浮点数
char *itoa(int value, char *string, int radix)	将整型值 value 以 radix 进制表示法写入 string

1.6 代码风格

使用比如匈牙利命名法

1. 标识符号应能提供足够信息，最好是可以发音的。
2. 为全局变量取长的，描述信息多的名字，为局部变量取短名字
3. 名字太长时可以适当采用单词的缩写。但要注意，缩写方式要一致。要缩写就全都缩写。比如单词 Number, 如果在某个变量里缩写成了：int nDoorNum；那么最好包含 Number 单词的变量都缩写成 Num。
4. 注意使用单词的复数形式。如 `int nTotalStudents, nStudents` 容易让人理解成代表学生数目，而 nStudent 含义就不十分明显
5. 对于返回值为真或假的函数，加“Is”前缀如：`int IsCanceled();` `int isalpha();`（C 语言标准库函数）`BOOL IsButtonPushed();`
6. 对于获取某个数值的函数，加“Get”前缀 `char * GetFileName();`
7. 对于设置某个数值的函数，加“Set”前缀 `void SetMaxVolume();`
8. 一般变量和结构名用名词，函数名用动词或动宾词组

1.7 字符串

字符串是一个 const 数组，它存储的是一个字符串序列，其中还包括转义字符序列。在读入字符串时，单个字符串中不能有空格。常用的字符串处理函数包括（均需要头文件 `string.h`：

1. 将格式化数据写入字符串：sprintf
2. 字符串长度查询函数：strlen
3. 字符串复制函数：strcpy、strncpy
4. 字符串连接函数：strcat
5. 字符串比较函数：strcmp、strncmp、stricmp、strnicmp
6. 字符串搜索函数：strcspn、strspn、strstr、strtok、strchr
7. 字符串大小写转换函数：strlwr、strupr

在使用这些函数时，有一些比较需要注意的问题：

1. 不带着 n 的函数在执行过程中有可能引发越界，导致 segmentation fault
2. `strlen`是一个时间复杂度为 O(n) 的函数，故应避免将其放置在循环控制语句中，如 `for(int i = 0; i < strlen(s) ; i++)` 这么写就不好
3. `stricmp` 不区分大小写，且不同编译器的实现有所不同，故结果也有可能不一样（甚至会报错）

1.8 高精度数

用字符型或者整形数组来存放大整数，加减法即按照竖式直接计算。乘法开一个很大很大的数组，不用着急进位，第 i 位和第 j 位相乘第结果放在结果数组的第 i + j 位上。除法是做大整数减法。

1.9 枚举

枚举的思想：猜测和剪枝。举个例子，一个很大的整数 $(m_1m_2\dots m_n)$，乘以小于等于 n 的数，是否是它自己的循环。转化成判断乘积是否是 $(m_1m_2\dots m_nm_1m_2\dots m_n)$ 的字串，就是逐个枚举。有的时候需要用树结构枚举可能出现的情况，并采取响应的剪枝，比如八皇后的某一种解决办法。

1.9.1 枚举类型

枚举类型有点像一个储存 parameter 的数组，枚举数据类型描述的是一组整型值的集合（这句话其实不太妥当），枚举型是预处理指令 #define 的替代，枚举和宏其实非常类似，宏在预处理阶段将名字替换成对应的值，枚举在编译阶段将名字替换成对应的值

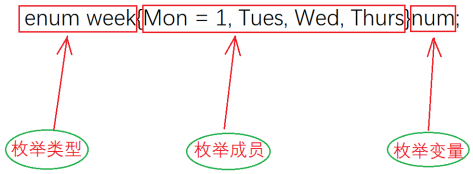

图 1.3: 枚举类型声明的结构体

当枚举类型和枚举变量放在一起定义时，枚举类型的名字（就是 enum week 中的 week）可以省略不写，枚举类型变量的赋值只能用自身的枚举成员来赋值，以上面的例子来说，num 的赋值就只能用枚举成员 Mon、Tues、Wed、Thurs，或者强制类型转换 (enum week) 10，而不能用其他枚举类型的枚举成员来赋值。在显式赋值时，未指定的枚举名的值将依着最后一个指定值向后依次递增（注意是最后一个指定值），如下代码：

```
enum week{Mon = 1, Tues, Wed, Thurs, Fri = 10, Sat, Sun} num;
```

定义了一个枚举类型变量 num，其实际绑定结果为：1, 2, 3, 4, 10, 11, 12

一个整数不能直接赋值给一个枚举变量，必须用该枚举变量所属的枚举类型进行类型强制转换后才能赋值，如

```
num = (enum week) 10
```

总结一下：

1. 在没有显示说明的情况下，枚举常量（也就是花括号中的常量名）默认第一个枚举常量的值为 0，往后每个枚举常量依次递增 1

2. 在部分显示说明的情况下，未指定的枚举名的值将依着之前最有一个指定值向后依次递增
3. 一个整数不能直接赋值给一个枚举变量，必须用该枚举变量所属的枚举类型进行类型强制转换后才能赋值
4. 同一枚举类型中不同的枚举成员可以具有相同的值
5. 同一个程序中不能定义同名的枚举类型，不同的枚举类型中也不能存在同名的枚举成员（枚举常量）

1.10 递归

递归就是…套娃，还有一种 DFS 的味道

第 2 章　一些可能会用到的方法

2.1　填充数组

当然，数组可以是动态分配也可以是直接分配。写题（尤其是答卷子）一般不需要考虑能开多大的数组

1. `sprintf()` 函数，可以将输出的数据存进一个数组 buffer 里面
2. `memset()` 可以初始化一片连续的逻辑空间为 0
3. 有的时候可以采用 `enum` 来代替数组，枚举类型类似于 verilog 的`locoparam`，赋值可以`int a = Mon`，但实际上 a 的值是 1（如果周日是枚举表的第一个的话）

2.2　读入多组数据

可用 `while( cin.getline(szLine,210) )` 的方式判断数据是否读完。`cin.getline` 读取一行，第一个参数是缓冲区地址；第二个参数是缓冲区大小，为了防止越界用的。缓冲区不够大，就自动截断。它会自动往缓冲区末尾添加 '`\0`'。

2.2.1　scanf 与 while 的使用

第一种用法　`while(scanf("%d",&n),n)`

功能：当 n 为 0 时中止循环

这里要先说一下逗号表达式：逗号表达式的值是逗号后面的那个数。例如 `x = (5,6)`，则 `x == 6`。`while(scanf("%d",&n),n)` 括号里的语句其实就是个逗号表达式，它的返回值是 n 的值，所以这个语句就相当于 `while(n)`，`n == 0` 时跳出循环，写成这样是为了同时输入。如果是 `while( scanf("%d%d", &n, &m,), n, m )`，那么就相当于 `while(m)`。

第二种用法　`while(scanf("%d", &n) != EOF)` 和 `while(~scanf("%d", &n))`

功能：当读到文件结尾时中止循环

`scanf` 语句的返回值为成功赋值的个数，例如 `scanf("%d %d", &a, &b)`，如果 a、b 均赋值成功返回值为 2，只是 a 赋值成功返回 1，a、b 都不成功返回 0，出错的时候返回 EOF。~ 是按位取反，scanf 语句如果没有输入值就是返回-1，按位取反结果为 0。

注意：这两种方法在输入字母的时候会变成死循环，而 `scanf("%d %d",&a,&b) == 2` 不会。windows 下可通过按 Ctrl +Z 并回车、linux 下可通过 Ctrl + D 来达到“输入”文件结束符的效果，结束循环。

第三种用法 `while(scanf("%d",&n) == 1)`

功能：赋值失败，跳出循环

这个应该很好理解了吧，如果是 `scanf("%d%d",&n,&m)` 就是 `while(scanf("%d %d",&a,&b) == 2)`

另外补充一下**取反**的用法：`~` 是按位取反，由于计算机中数据存储为补码，故实际结果为 $-x-1$。若是想要起到判 0 的作用，一个是使用 `bool` 型变量，另一个也可以使用非 `!`，如 `!(2) == 0`

2.2.2 注意

在读取字符串存进数组的时候，可以直接用 `cin` 直接整体读进去，但要注意在开数组的时候，要比字符串的最大长度**多 1**！因为需要用来存 `\0`

2.3 特殊运算

灵活利用位运算和求余运算，可以简化数值处理的过程。字符型数据同样可以按照 ASCII 码来计算（小写字母 = 大写字母 + 32）

2.4 一些语句

对于 `if...else`，如果是多个 if 语句并列出现，则相当于是顺序的判断，当判断为假时不执行操作。而如果改成 if…else if…，那么就具有了优先级的区分，当高优先级执行时就不会执行低优先级的语句了。

2.5 读入 C 风格的字符串

`scanf("%s %s", array_a, array_b)` 注意这里没有 `&`，因为数组名已经是入口地址了。或者用 `cin>>array_a>>array_b` 也可以的

第 3 章　记录一些坑

3.1　ends

`ends` 不是想象中（老师讲的）那个输出一个高逼格的空格，它实际上是在缓冲区写入一个 `'\0'`。对于 Windows 系统下，它会自己在输出的时候在 `'\0'` 的位置加一个空格，但是 Linux 和 macOS 就什么都没有

3.2　数组

数组开大一点、大一点、大一点……比如字符串是两位的就直接开到 `char array[5]`