中山大學
SUN YAT-SEN UNIVERSITY

C

笔记整理

姓名：刘斯宇
学号：17341110

Contents

☑ **第一章--导言**

1 导言

1.1 编译与运行

在 unix 系统中，比如我们有一个 `hello.c` 的 C 文件，我们可以通过 `cc hello.c` 进行编译，然后生成 `a.out` 的可执行文件，输入`./a.out` 就可以运行了。

1.2 变量与算数表达式

在 `printf` 语句的第-一个参数的`%d` 中指明打印宽度，则打印的数字会在打印区域内右对齐。

`printf` 中的转换说明`%3.0f` 表明待打印的浮点数至少占 3 个字符宽，且不带小数点和小数部分; `%6.1f` 表明待打印的数至少占 6 个字符宽，且小点后面有 1 位数字。`%.2f` 按照浮点数打印，小数点后有两位小数.

1.3 字符输入/输出

`getchar` 函数从文本流中读入下一个输入字符，并将其作为结果值返回。

`EOF` 定义在头文件 `<stdio.h>` 中，是个整型数，其具体数值是什么并不重要，只要它与任何 `char` 类型的值都不相同即可。这里使用符号常量，可以确保程序不需要依赖于其对应的任何特定的数值。

2 类型、运算符和表达式

2.1 常量

某些字符可以通过转义字符序列 (例如，换行符 `n`) 表示为字符和字符串常量。转义字符序列看起来像两个字符，但只表示-一个字符。另外，我们可以用'\000' 表示任意的字节大小的位模式。其中，ooo 代表 1 3 个八进制数字 (0...7)。这种位模式还可以用'\xhh' 表示，其中 hh 表示一个或多个十六进制数字。

字符常量'\0' 表示值为 0 的字符，也就是空字符 (null)。我们通常用'\0 ' 的形式代替 0, 以强调某些表达式的字符属性，但其数字值为 0。

常量表达式是仅仅只包含常量的表达式。这种表达式在编译时求值，而不在运行时求值。

标准库函数 strlen(s) 可以返回字符串参数 s 的长度, 但长度不包括末尾的'\0'。

```c
/* strlen: return length of s */
int strlen(char s[])
{
    int i;
    while (s[i] != '\0')
        ++i;
    return i;
}
```

枚举常量是另外一-种类型的常量。枚举是-一个常量整型值的列表,例如 `enum boolean{NO,YES}`. 在没有显式说明的情况下，enum 类型中第-一个枚举名的值为 0，第二个为 1，依此类推。如果只指定了部分枚举名的值，那么未指定值的枚举名的值将依着最后一个指定值向后递增. `enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC };`

枚举和 #define 比较：

- 枚举可以自增 1，这样不用每一个值都定义，而宏必须每个值都定义。
- #define 宏常量是在预编译阶段进行简单替换。枚举常量则是在编译的时候确定其值。
- 一般在编译器里，可以调试枚举常量，但是不能调试宏常量。

2.2 声明

一个声明指定一种变量类型，后面所带的变量表可以包含一个或多个该类型的变量。

初始化：在声明中，如果变量名的后面紧跟一个等号以及一个表达式，该表达式就充当对变量进行初花的初始化表达式。

2.3 算术运算符

2.4 类型转换

转换规则

- 首先，如果任何一个操作数为 long double 类型，则将另一个操作数转换为 long double 类型。
- 否则，如果任何-一个操作数为 double 类型，则将另-一个操作数转换为 double 类型。

- 否则，如果任何一一个操作数为 float 类型，则将另一个操作数转换为 float 类型。
- 否则，同时对两个操作数进行整型提升; 然后，如果任何-一个操作数为 unsigned long int 类型，则将另一个操作数转换为 unsigned long int 类型。
- 否则，如果-一个操作数为 long int 类型且另-一个操作数为 unsigned int 类型，则结果依赖于 long int 类型是否可以表示所有的 unsigned int 类型的值。如果可以，则将 unsigned int 类型的操作数转换为 long int; 如果不可以，则将两个操作数都转换为 unsigned long int 类型。
- 否则，如果一个操作数为 long int 类型，则将另-一个操作数转换为 long int 类型。
- 否则, 如果任何一个操作数为 unsignedint 类型, 则将另-一个操作数转换为 unsigned int 类型。
- 否则，将两个操作数都转换为 int 类型。

2.5 条件表达式

`expr1 ? expr2 : expr3`，如果 expr2 与 expr3 的类型不同，结果的类型由之前讨论的转换规则决定。举个例子，如果 f 为 float 类型，n 为 int 类型，那么表达式 `(n>0)?f:n`，是 float 类型的，与 n 是否为正值无关。

一个比较好的用条件表达式的例子

```
1 printf("You have %d item%s.\n", n, n==1 ? "" : "s");
```

2.6 运算符优先级与求值次序

Operator	Associativity
() [] -> .	left to right
! ++ – + - * (type) sizeof	right to left
* / %	left to right
+ -	left to right
« »	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
= += -= *= /= %= &= ^= |= «= »=	right to left
,	left to right

一些比较容易犯的错误：

- 由于位运算的优先级比运算符 ==,!= 的低，因此位测试的时候，必须要用圆括号 `if((x&MASK)==0)`
- 我们可能很疑惑一元的 + 和-是什么意思，这是正负数的表示！千万别忘了！
- `sizeof()` 和 `strlen()` 到底有什么区别?

```
1 char arr[10] = "What?";
2 int len_one = strlen(arr);
3 int len_two = sizeof(arr);
4 cout << len_one << " and " << len_two << endl;
```

输出结果为：5 and 10 点评：sizeof 返回定义 arr 数组时，编译器为其分配的数组空间大小，不关心里面存了多少数据。strlen 只关心存储的数据内容，不关心空间的大小和类型。

-

3 控制流

在不同的系统中，保存在多个源义件中的 C 语言程序的编译与加载机制是不同的。例如，在 UNIX 系统中，可以使用第 1 章中提到过的 cc 命令执行这一任务。假定有 3 个函数分别存放在名为 main.c、getline.c 与 strindex.c 的 3 个文件中，则可以使用命令 `cc main.c getline.c strindex.c` 来编译这 3 个文件，并把生成的目标代码分别存放在文件 main.o、getline.o 与 strindex.o 中，然后再把这 3 个文件-起加载到可执行文件 a.out 中。如果源程序中存在错误 (比如文件 main.c 中存在错误)，则可以通过命令 CC main.C getline.o strindex.o 对 main.c 文件重新编译，并将编译的结果与以前已编译过的目标文件 getline.o 和 strindex.o -起加载到可执行文件中。cc 命令使用 “.c” 与 “.o” 这两种扩" 展名来区分源文件与目标文件。

3.1 外部变量

如果变量定义在任何函数的外部，则是外部变量。

3.2 作用域规则

静态全局变量与动态全局变量的主要不同：动态全局变量可以通过 extern 关键字在外部文件中使用，但静态全局变量不可以在外部文件中使用。静态全局变量相当于限制了动态全局变量的作用域

3.3 初始化

在不进行显式初始化的情况下，外部变量和静态变量都将被初始化为 0，而自动变量和寄存器变量的初值则没有定义 (即初值为无用的信息)。

用字符数组和字符指针变量都可实现字符串的存储和运算，但是两者是有区别的

```
1 char a[]="abc";
2 char b[]={'a','b','c','\0'};
3 char c[]={'a','b','c'};//记住字符数组是不会自动补'\0'的。
4 char *d="abc";
```

上面的 a 和 b 是等价的，但是千万要小心 c 的情况，还有用 sizeof() 函数对上面的 4 个变量运算的时候也是有区别的，因为 d 是一个指针，所以用 sizeof 跟它的内容毫无关系，它表示的是指针的 size!！这个很重要，其他的三个就跟内容有关了。

3.4 宏定义

可以通过 #undef 指令取消名字的宏定义，这样做可以保证后续的调用是函数调用，而不是宏调用。

宏只是简单的替换而已，所以非常容易出错，而且出错的时候也很难找到 bug，比如这样的宏定义 `#define square(x) x * x`，如果我们用 `square (z+1)` 的时候其实结果是这样的 `z+1*z+1` 显然不是正确的答案，所以宏定义如果要用一定加括号。

#if 语句对其中的常量整型表达式 (其中不能包含 sizeof、类型转换运算符或 enum 常量) 进行求值，若该表达式的值不等于 0，则包含其后的各行，直到遇到 #endif、#elif 或 #else 语句为止 (预处理器语句 #elif 类似于 else if)。在 #if 语句中可以使用表达式 defined(名字)，该表达式的值遵循下列规则: 当名字已经定义时，其值为 1; 否则，其值为 0。

C 语言专门定义了两个预处理语句 #ifdef 与 #ifndef, 它们用来测试某个名字是否已经定义。上面有关 #if 的第-一个例子可以改写为下列形式:

```
1  #ifndef HDR
2  #define EDR
3  /* hdr.h 文件的内容放在这里 */
4  #endif
```

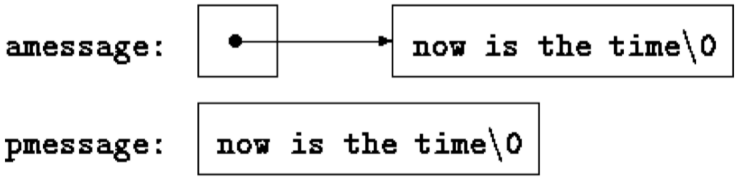
4 Pointers and Arrays

A pointer is a variable that contains the address of a variable.

a pointer is constrained to point to a particular kind of object: every pointer points to a specific data type. (There is one exception: a “pointer to void” is used to hold any type of pointer but cannot be dereferenced itself.)

(*ip)++ ，*p ++ 改变的是指针，运算符优先级的问题!

```
1  char amessage[] = "now is the time"; /* an array */
2  char *pmessage = "now is the time"; /* a pointer */
```



```c
/* strcpy: copy t to s; pointer version 2 */
void strcpy(char *s, char *t) {
    while ((*s++ = *t++) != '\0')
        ;
    }

/* strcpy: copy t to s; pointer version 3 */
void strcpy(char *s, char *t) {
    while (*s++ = *t++)
        ;
    }
```

```c
1. 不要忘记最后还要加上结束符
void strcat2(char s[], char t[]) {
  int i = strlen(s);
  char c;
  while ((c = *t) != '\0') {
    *(s + i++) = c;
    t++;
  }
  *(s + i) = '\0';
}
```

```c
/*
* Write the function strend(s,t),
* which returns 1 if the string t occurs at the end of the string s,
   and zero otherwise.
*/

#include <string.h>

int strend(char s[], char t[]) {
```

```
9   int lens = strlen(s);
10  int lent = strlen(t);
11
12  if (!lent || lens < lent) return 0;
13
14  while (lent > 1) {
15    if (*(t + lent--) != *(s + lens--)) {
16      return 0;
17    }
18  }
19
20  return 1;
21 }
```

If a two-dimensional array is to be passed to a function, the parameter declaration in the function must include the number of columns; the number of rows is irrelevant, since what is passed is, as before, a pointer to an array of rows, where each row is an array of 13 ints. In this particular case, it is a pointer to objects that are arrays of 13 ints. Thus if the array daytab is to be passed to a function f, the declaration of f would be:

`f(int daytab[2][13]) { ... }`

It could also be

`f(int daytab[][13]) { ... }`

since the number of rows is irrelevant, or it could be

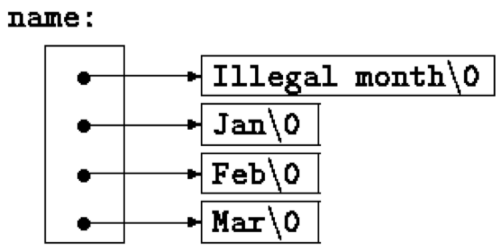
`f(int (*daytab)[13]) { ... }`

`int (*daytab)[13]` is a pointer to an array of 13 integers

`int *daytab[13]` is an array of 13 pointers to integers.

4.1 Pointers vs. Multi-dimensional Arrays

```
char *name[] =  "Illegal month", "Jan", "Feb", "Mar" ;
```




```
char aname[][] =  "Illegal month", "Jan", "Feb", "Mar" ;
```

aname:

Illegal month\0 Jan\0	Feb\0	Mar\0
0 15	30	45

4.2 Command-line Arguments

When main is called, it is called with two arguments. The first (conventionally called argc, for argument count) is the number of command-line arguments the program was invoked with; the second (argv, for argument vector) is a pointer to an array of character strings that contain the arguments, one per string.

By convention, argv[0] is the name by which the program was invoked, so argc is at least 1. If argc is 1, there are no command-line arguments after the program name.

4.3 Pointers to Functions

```
1  char **argv argv: pointer to char
2  int (*daytab)[13] daytab: pointer to array[13] of int
3  int *daytab[13] daytab: array[13] of pointer to int
4  void *comp() comp: function returning pointer to void
5  void (*comp)() comp: pointer to function returning void
6  char (*(*x())[])() x: function returning pointer to array[] of
       pointer to function returning char
7  char (*(*x[3])())[5] x: array[3] of pointer to function returning
       pointer to array[5] of char
```

5 Structure

5.1 Basics of Structures

```
struct point {
    int x; int y;
    };

struct rect {
    struct point pt1;
    struct point pt2;
    };

struct rect screen;//declare
```

5.2 Structures and Functions

```
    struct point {
        int x;
        int y;
    };
    struct point makepoint(int x, int y) {
        struct point temp;
        temp.x = x;
        temp.y = y;
        return temp;
    }
```

++p->len 等价于 ++(p->len)

*p->str fetches whatever str points to; *p->str++ increments str after accessing whatever it points to (just like *s++); (*p->str)++ increments whatever str points to; and *p++->str increments p after accessing whatever str points to.

5.3 Arrays of Structures

```
struct key
{ char *word; int count; }
keytab[] =
{
    "auto", 0,
    "break", 0,
    "case", 0,
    "char", 0,
    "const", 0,
    "continue", 0,
    "default", 0,
    /* ... */
    "unsigned", 0,
    "void", 0,
    "volatile", 0,
    "while", 0
};
```

5.4 Pointers to Structures

5.5 Self-referential Structures

```
struct tnode { /* the tree node: */
    char *word;/* points to the text */
    int count; /* number of occurrences */
    struct tnode *left; /* left child */
    struct tnode *right; /* right child */
    };
```

5.6 Typedef

C provides a facility called typedef for creating new data type names. For example, the declaration

typedef char *String; makes String a synonym for char declarations and casts:* or char

5.7 Unions

Unions provide a way to manipulate different kinds of data in a single area of storage, without embedding any machinedependent information in the program. They are analogous to variant records in pascal.

It is the programmer's responsibility to keep track of which type is currently stored in a union;

6 Input and Output

fp = fopen(name, mode);

The first argument of fopen is a character string containing the name of the file. The second argument is the mode, also a character string, which indicates how one intends to use the file. Allowable modes include read ("r"), write ("w"), and append ("a").