# C

## 关键字和一些零散知识点

### 静态库

**静态库的特点**

1、编译阶段加入程序，所以程序可以单独使用，不需要库文件

2、编译阶段加入程序，所以每个程序使用单独的静态库文件，浪费资源

3、 编译阶段加入程序，所以更新静态库需要重新编译程序

**windows下使用vs生成静态库**

1、编写库文件

2、设置项目为静态库项目（找不到入口，是没有选择所以配置）

3、生成静态库

4、找到 .lib文件使用，.h文件可以不使用

**windows下使用静态库**

1、库文件使用 #pragma comment(lib,"./静态库.lib")

2、如果需要使用头文件的话，也需要包含一下头文件

**linux下静态库的生成**

1、编写库文件

2、通过gcc -c 编译源文件获得 .o 二进制文件

3、使用 ar 把 .o 文件打包，生成 .a 库文件

示例：ar rcs libcalc.a add.o div.o mult.o sub.o

ar：命令

rcs：参数

libcalc.a：生成的库文件

**linux下静态库的使用**

1、指定头文件位置，-I (大写i) 指定头文件位置

2、指定库文件位置， -L 指定库文件位置

3、指定库文件名称，-l (小写L)指定库文件名称，不要lib前缀和.a后缀

### 动态库

**动态库的特点**

1、运行阶段加入程序，所以程序运行需要动态库文件

2、多个程序可以使用一份动态库，节省空间

3、运行阶段加入程序，所以更新动态库不需要重新编译程序

**windows下使用vs生成动态库**

1、编写库文件，.h文件里面每个函数的声明前面需要添加\_\_declspec(dllexport)

2、项目设置为动态库

3、生成动态库文件

4、找到.lib文件和.dll文件放到当前目录下使用，.h文件可以不使用

**windows下使用动态库**

1、关联库文件#pragma comment(lib,"./动态库.lib")

2、确保.dll文件在当前目录下，或者在环境变量中

3、如果要使用头文件的话，也包含以下头文件

**linux下动态库的生成**

1、编写库文件

2、使用gcc -c -fpic编译源文件，生成和位置无关 .o文件

示例·：gcc -c sub.c add.c div.c mult.c -fpic

3、使用gcc-shared得到 .so动态库文件

示例：gcc -shared add.o sub.o div.o mult.o -o libcalc.so

**linux下动态库的使用**

1、指定头文件位置，-I (大写i) 指定头文件位置

2、指定库文件位置， -L 指定库文件位置

3、指定库文件名称，-l (小写L)指定库文件名称，不要lib前缀和.so后缀

**linux程序运行时找不到动态库的解决方法**

1、ldd 程序名，查看程序的库连接情况

2、解决方法一，动态库的绝对路径添加到LD\_LIBRARY\_PATH：

i、临时生效执行下面命令

ii、下面命令添加到 .bashrc 文件尾部

iii、下面命令添加到 /etc/下的profile 配置文件中

命令：export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:动态库的绝对路径加名称 ，名称去掉前后缀

2、 解决方法二：在 /etc/ld.so.cache 添加动态库的路径

i、无法直接修改 /etc/ld.so.cache 需要间接修改

ii、在 /etc/ld.so.conf 配置文件中添加动态库所在的示例，不要名称

iii、执行命令使配置生效：sudo ldconfig

3、解决方法三：

i、库文件放入 /use/lib/ 或者 /lib/目录中

### C程序编译步骤

1、预处理 -E

- 生成 .i 文件

- 头文件展开

- 宏替换

- 注释删除

2、编译 -S

生成 .s 汇编代码

3、汇编 -c

生成 .o 二进制文件

4、链接

链接 .o 文件生成可执行程序

### gcc常用参数

-E : 预处理指定的源文件，不进行编译

-S：编译指定源文件，但不进行汇编

-c：编译、汇编指定源文件，但不进行链接

-o：指定连接后，可执行程序的名称

-I（大写i）：指定头文件搜索目录

-g：编译时生成调试信息，可以被调试器调试

-D：编译时指定一个宏

-w：不生成任何警告信息

-Wall：生成所有警告信息

-On：n的取值范围0~3。编译器的优化级别，-O0 没有优化，-O1默认优化级别

-l（小写l）:编译时指定使用的 库名称

-L：编译时指定搜索库的路径

-fpic：生成和位置无关的代码

-shared：生成共享目标文件，生成动态库

-std：指定C 版本，如-std=c99

### printf打印格式

%d 十进制有符号整数(int)

%u 十进制无符号整数

%o 无符号以八进制表示的整数

%x 无符号以十六进制表示的整数

%f 浮点数

%lf double

%s 字符串

%c 单个字符(char)

%p 指针的值

%e 指数形式的浮点数

%g 自动选择合适的表示法

%ld long

%02X 十六进制打印不足两位的补0

### 运算符优先级

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **优先级** | **运算符** | **名称或含义** | **使用形式** | **结合方向** | **说明** |
| **1** | **[]** | 数组下标 | 数组名[常量表达式] | 左到右 | -- |
| **()** | 圆括号 | (表达式）/函数名(形参表) | -- |
| **.** | 成员选择（对象） | 对象.成员名 | -- |
| **->** | 成员选择（指针） | 对象指针->成员名 | -- |
|  | | | | | |
| **2** | **-** | 负号运算符 | -表达式 | **右到左** | 单目运算符 |
| **~** | 按位取反运算符 | ~表达式 |
| **++** | 自增运算符 | ++变量名/变量名++ |
| **--** | 自减运算符 | --变量名/变量名-- |
| **\*** | 取值运算符 | \*指针变量 |
| **&** | 取地址运算符 | &变量名 |
| **!** | 逻辑非运算符 | !表达式 |
| **(类型)** | 强制类型转换 | (数据类型)表达式 | -- |
| **sizeof** | 长度运算符 | sizeof(表达式) | -- |
|  | | | | | |
| **3** | **/** | 除 | 表达式/表达式 | 左到右 | 双目运算符 |
| **\*** | 乘 | 表达式\*表达式 |
| **%** | 余数（取模） | 整型表达式%整型表达式 |
| **4** | **+** | 加 | 表达式+表达式 | 左到右 | 双目运算符 |
| **-** | 减 | 表达式-表达式 |
| **5** | **<<** | 左移 | 变量<<表达式 | 左到右 | 双目运算符 |
| **>>** | 右移 | 变量>>表达式 |
| **6** | **>** | 大于 | 表达式>表达式 | 左到右 | 双目运算符 |
| **>=** | 大于等于 | 表达式>=表达式 |
| **<** | 小于 | 表达式<表达式 |
| **<=** | 小于等于 | 表达式<=表达式 |
| **7** | **==** | 等于 | 表达式==表达式 | 左到右 | 双目运算符 |
| **!=** | 不等于 | 表达式!= 表达式 |
|  | | | | | |
| **8** | **&** | 按位与 | 表达式&表达式 | 左到右 | 双目运算符 |
| **9** | **^** | 按位异或 | 表达式^表达式 | 左到右 | 双目运算符 |
| **10** | **|** | 按位或 | 表达式|表达式 | 左到右 | 双目运算符 |
| **11** | **&&** | 逻辑与 | 表达式&&表达式 | 左到右 | 双目运算符 |
| **12** | **||** | 逻辑或 | 表达式||表达式 | 左到右 | 双目运算符 |
|  | | | | | |
| **13** | **?:** | 条件运算符 | 表达式1?  表达式2: 表达式3 | **右到左** | 三目运算符 |
| **14** | **=** | 赋值运算符 | 变量=表达式 | **右到左** | -- |
| **/=** | 除后赋值 | 变量/=表达式 | -- |
| **\*=** | 乘后赋值 | 变量\*=表达式 | -- |
| **%=** | 取模后赋值 | 变量%=表达式 | -- |
| **+=** | 加后赋值 | 变量+=表达式 | -- |
| **-=** | 减后赋值 | 变量-=表达式 | -- |
| **<<=** | 左移后赋值 | 变量<<=表达式 | -- |
| **>>=** | 右移后赋值 | 变量>>=表达式 | -- |
| **&=** | 按位与后赋值 | 变量&=表达式 | -- |
| **^=** | 按位异或后赋值 | 变量^=表达式 | -- |
| **|=** | 按位或后赋值 | 变量|=表达式 | -- |
|  | | | | | |
| **15** | **，** | 逗号运算符 | 表达式,表达式,… | 左到右 | -- |

### typedef

格式：

typedef 数据类型 需要的别名

优点：

1、简化结构体变量的创建，可以省略struct关键字

2、提高可移植性

### sizeof

1、返回的是unsingned int

2、用于指针时只能获取指针的大小，取\*也只能获取指针的类型大小

### static

1、运行前分配内存

2、程序运行期间一直存活

3、只有第一次初始化有效

4、只能在当前文件使用

### const

1、放在全局的const 变量，无法通过指针修改

### extern

声明某个变量，方法已经存在，让编译器通过编译

### #define 宏

1、预处理阶段展开

2、 宏函数

a、MYADD(x,y) ((x)+(y))

b、多个括号是防止运算优先级出错

3、用的多的东西可以使用宏，方便管理

4、条件编译

### 位运算，左右移

~：按位取反

一个数的取反结果是：0变1，1变0

&：按位与

1、一个数 与 外一个数：同1为1，其他为0

2、将某几位置0

|：按位或

1、一个数 或 另外一个数：同0位0，其他为

^：按位异或

1、一个数 异或 另外一个数：相同为0，不同为1

左移

格式：数据 << N

数据：需要左移的数据

<<：左移运算符

N：表示左移几位

注意：

1、左移后，左侧头部数据被挤出，后面补0

2、位运算符，不会改变数据原有的值

3、多次位移，分开操作

### 内存对齐

内存对齐的大小，看结构体里面的“最大的数据类型”和“对齐模数”，谁小用谁

* 1. #pragma pack(show) #pragma pack(1)，查看默认对齐模数、设置对齐模数
  2. 每一次对齐都是看“最大的数据类型”和“对齐模数”
  3. 即使结构体结束，末尾的地方也要对齐
  4. 数组，结构体，展开单个数据来看，不做整体看

## 内存、字符串操作

头文件#include <string.h>

一般来说字符串操作函数遇到’\0’都会结束

### memset()

void \*memset(void \*s, int c, size\_t n);

参数：

s：需要替换的起始地址

c：需要替换的内容，直接给字符或者ascii都行

n：替换多少个字节

返回值：指向s的指针 (基本不用不太清楚)

注意：

1. 完全使用内存操作系列函数组包时，如果写入txt可能导致乱码，用其他软件打开解决，如notepad++,vs,qt,写字板等等，原因不明

### memcmy()比较

int memcmp(const void \*s1, const void \*s2, size\_t n);

参数：

s1：比较数据1

* s2：比较数据2
* n：比较长度（字节）

返回值：

相同返回：0

不同返回：非0值

注意：

1．不同情况下返回的非0值根据字符串大小决定，字符串大小根据ASCII表决定

### memcpy()拷贝

void \*memcpy(void \*dest, const void \*src, size\_t n);

参数：

dest：拷贝目标地址

src：存有数据的地址

n：拷贝几个字节

返回值：指向dest的指针

注意：src的dest所指的内存区域不能重叠

### 分割字符串strtok()

char \*strtok(char \*str, const char \*delim);

参数：

str：需要分割的字符串

delim：分割符，可以是字符串，只要匹配到其中任何一个字符都会分割

返回值：

还能继续分割返回：分割出来的字符串

分割完成返回：NULL

注意：

1. 函数内部会自动保存剩下需要分割的字符串，所以第一次以后传NULL即可
2. 会改变参数str字符串，字符串中每个找到的分割符，都会被替换成’\0’

示例代码：

void lrStrtok()

{

char p[] = "192.1,6,8.1.1";

char pp[] = ".,";

char \*ppp;

printf("%s\n", strtok(p, pp));

while ((ppp = strtok(NULL, pp))!=NULL)

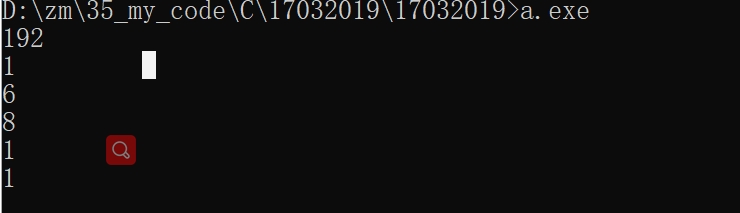
{

printf("%s\n", ppp);

}

}

运行结果：



### 查找指定字符串strstr()

char \*strstr(const char \*haystack, const char \*needle)

参数：

haystack：被查找字符串(数据)

needle：需要查找的字符（串）（务必使用字符串，不要使用字符，\0关系）

返回值：

找到返回：needle第一次出现的位置

没找到返回：NULL

注意：

1．查找指定字符第一次出现的位置：char \*strchr(const char \*s, int c);

2．查找指定字符最后一次出现的位置：char \*strrchr(const char \*s, int c);

### 拼接字符串strcat()

char \*strcat(char \*dest, const char \*src);

参数：

dest：被追加字符串

src：追加字符串

返回值：dest字符串的首地址

注意：

1. dest需要有足够的空间来存储src的数据
2. char \*strncat(char \*dest, const char \*src, size\_t n);
   1. 可以指定src的长度

### 拷贝字符串strcpy()

char \*strcpy(char \*dest, const char \*src);

参数：

dest：目标字符串

src：有数据的字符串

返回值：dest字符串起始地址

注意：

1. 如果dest所指空间内存不够，会造成缓冲溢出(buffer Overflow)错误。
2. char \*strncpy(char \*dest, const char \*src, size\_t n);
   1. 可以指定src的拷贝长度

### 字符串比较strcmp()

int strcmp(const char \*s1, const char \*s2);

参数:

s1：比较参数1

s2：比较参数2

返回值：

相同返回：0

不同返回：非0值

注意：

1. 不同情况下返回的非0值根据字符串大小决定，字符串大小根据ASCII表决定
2. int strncmp(const char \*s1, const char \*s2, size\_t n);
   1. 可以指定比较前几个字节
3. int strcasecmp(const char \*s1, const char \*s2);
4. int strncasecmp(const char \*s1, const char \*s2, size\_t n);
   1. 3.4条函数比较时可以忽略大小写

## fopen()文件操作

### fopen()

#include <stdio.h>

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，根据需要加上路径

mode：打开文件的模式设置

“r”： 以只读方式打开一个文本文件（不创建文件，若文件不存在则报错）

“w”：以写方式打开文件(如果文件存在则清空文件，文件不存在则创建一个文件)

“a”：以追加方式打开文件，在末尾添加内容，若文件不存在则创建文件

“r+”：以可读、可写的方式打开文件(不创建新文件)

“w+”：以可读、可写的方式打开文件(如果文件存在则清空文件，文件不存在则创建一个文件)

“a+”：以添加方式打开文件，打开文件并在末尾更改文件,若文件不存在则创建文件

返回值：

成功：文件指针

失败：NULL

注意：

1. 第二个参数每种打开模式都可以加b表示二进制方式打开，只在windows下有效
2. windows的磁盘文本文件使用\r\n（占用两个字节）的方式换行，不加b读写时都会自动转换

### fclose()

#include <stdio.h>

int fclose(FILE \* stream);

功能：关闭先前fopen()打开的文件。此动作让缓冲区的数据写入文件中，并释放系统所提供的文件资源。

参数：

stream：文件指针

返回值：

成功：0

失败：-1

### fputc()单个字符写

#include <stdio.h>

int fputc(int ch, FILE \* stream);

功能：将ch转换为unsigned char后写入stream指定的文件中

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

### fgetc()单个字符读

#include <stdio.h>

int fgetc(FILE \* stream);

功能：从stream指定的文件中读取一个字符

参数：

stream：文件指针

返回值：

成功：返回读取到的字符

失败：-1

### fputs()写一个字符串

#include <stdio.h>

int fputs(const char \* str, FILE \* stream);

功能：将str所指定的字符串写入到stream指定的文件中。

参数：

str：需要写的数据

stream：文件指针

返回值：

成功：0

失败：-1

注意：

1. 写字符串，遇\0结束，遇到\n不会结束

### fgets()读一行字符串

#include <stdio.h>

char \* fgets(char \* str, int size, FILE \* stream);

功能：从stream指定的文件内读入字符，保存到str所指定的内存空间，直到出现换行字符、读到文件结尾或是已读了size - 1个字符为止，最后会自动加上字符 '\0' 作为字符串结束。

参数：

str：读取到的数据存放的地方

size：指定最大读取字符串的长度（size - 1）

stream：文件指针

返回值：

成功：成功读取的字符串

读到文件尾或出错： NULL

注意：

1. 遇到\n结束

### fwrite()块写

#include <stdio.h>

size\_t fwrite(const void \*ptr, size\_t size, size\_t nmemb, FILE \*stream);

功能：以数据块的方式给文件写入内容

参数：

ptr：准备写入文件数据的地址

size： size\_t 为 unsigned int类型，此参数指定写入文件内容的块数据大小

nmemb：写入文件的块数，写入文件数据总大小为：size \* nmemb

stream：已经打开的文件指针

返回值：

成功：实际成功写入文件数据的块数目，此值和 nmemb 相等

失败：0

### fread()块读

#include <stdio.h>

size\_t fread(void \*ptr, size\_t size, size\_t nmemb, FILE \*stream);

功能：以数据块的方式从文件中读取内容

参数：

ptr：存放读取出来数据的内存空间

size： size\_t 为 unsigned int类型，此参数指定读取文件内容的块数据大小

nmemb：读取文件的块数，读取文件数据总大小为：size \* nmemb

stream：已经打开的文件指针

返回值：

成功：实际成功读取到内容的块数，如果此值比nmemb小，但大于0，说明读到文件的结尾。

失败：0

### fprintf()格式化写

#include <stdio.h>

int fprintf(FILE \* stream, const char \* format, ...);

功能：根据参数format字符串来转换并格式化数据，然后将结果输出到stream指定的文件中，指定出现字符串结束符 '\0' 为止。

参数：

stream：已经打开的文件

format：字符串格式，用法和printf()一样

返回值：

成功：实际写入文件的字符个数

失败：-1

注意：

1. 示例 fprintf(fp, "%d %d %d\n", 1, 2, 3);

### fscanf()格式化读

#include <stdio.h>

int fscanf(FILE \* stream, const char \* format, ...);

功能：从stream指定的文件读取字符串，并根据参数format字符串来转换并格式化数据。

参数：

stream：已经打开的文件

format：字符串格式，用法和scanf()一样

返回值：

成功：参数数目，成功转换的值的个数

失败： - 1

注意：

1. 示例
   1. int a = 0;
   2. int b = 0;
   3. int c = 0;
   4. fscanf(fp, "%d %d %d\n", &a, &b, &c);
   5. printf("a = %d, b = %d, c = %d\n", a, b, c);

### feof()是否读到文件尾

#include <stdio.h>

int feof(FILE \* stream);

功能：检测是否读取到了文件结尾。判断的是最后一次“读操作的内容”，不是当前位置内容(上一个内容)。

参数：

stream：文件指针

返回值：

非0值：已经到文件结尾

0：没有到文件结尾

### stat()获取文件状态

#include <sys/types.h>

#include <sys/stat.h>

int stat(const char \*path, struct stat \*buf);

功能：获取文件状态信息

参数：

path：文件名

buf：传入参数，保存文件信息的结构体

返回值：

成功：0

失败-1

注意：

1. 使用到的结构体

struct stat {

dev\_t st\_dev; //文件的设备编号

ino\_t st\_ino; //节点

mode\_t st\_mode; //文件的类型和存取的权限

nlink\_t st\_nlink; //连到该文件的硬连接数目，刚建立的文件值为1

uid\_t st\_uid; //用户ID

gid\_t st\_gid; //组ID

dev\_t st\_rdev; //(设备类型)若此文件为设备文件，则为其设备编号

off\_t st\_size; //文件字节数(文件大小)

unsigned long st\_blksize; //块大小(文件系统的I/O 缓冲区大小)

unsigned long st\_blocks; //块数

time\_t st\_atime; //最后一次访问时间

time\_t st\_mtime; //最后一次修改时间

time\_t st\_ctime; //最后一次改变时间(指属性)

};

### 删除、重命名文件

#include <stdio.h>

int remove(const char \*pathname);

功能：删除文件

参数：

pathname：文件名

返回值：

成功：0

失败：-1

#include <stdio.h>

int rename(const char \*oldpath, const char \*newpath);

功能：把oldpath的文件名改为newpath

参数：

oldpath：旧文件名

newpath：新文件名

返回值：

成功：0

失败： - 1

### fflush()强制刷新缓冲区

#include <stdio.h>

int fflush(FILE \*stream);

功能：更新缓冲区，让缓冲区的数据立马写到文件中。

参数：

stream：文件指针

返回值：

成功：0

失败：-1

### fopen()缓冲区的刷新机制

1. 满刷新
2. 关闭刷新
3. 强制刷新

注意：

1. 没有行刷新
2. 程序崩溃，没有正常结束不会触发关闭刷新