1 结构体对其

1:数据成员对齐规则:

结构(struct)(或联合(union))的数据成员，第一个数据成员放在offset为0的地方，

每个数据成员存储的起始位置要从该成员(每个成员本身)大小的整数倍开始(比如int在32位机为 4 字节,则要从 4 的整数倍地址开始存储)。

例子:

```
struct node
{
     char a;
     int b
};
sizeof(node)=8
```

0是任何数的整倍数，char的大小为1，int大小为4;

a存放在0 offset为位置，b存储的起始位置为1,但不满足对其原则，因为int大小为4,其存储位置应为4的整倍数，因此要在a后补齐，使b存储的起始位置为4

因此node大小为8

2:结构体作为成员:如果一个结构里有某些结构体成员,则结构体成员要从其内部最大元素大小的整数倍地址开始存储.(struct a里存有struct b,b里有char,int ,double等元素,那b应该从8的整数倍开始存储.)

例子:

```
struct pa

{
   char a;
   node b;
}
sizeof(pa)=12

b的起始位置要是4的整倍数，因此要在a后补位，

b占8字节
```

3:收尾工作:结构体的总大小,也就是sizeof的结果,.必须是其内部最大成员的整数倍（结构体成员以最大成员为代表）.不足的要补齐.

例子:

```
struct node

{
    double a;
    char b;
}

sizeof(node)=16

按1,2原则可计算出结果应为9;

但结构体大小应为最大成员的整倍数，因此结果应为16
```

如果编译器中提供了 #pragma pack(n)，上述对其模式就不适用了，例如设定变量以n 字节对齐 方式，则上述成员类型对齐宽度（应当也包括收尾对齐）应该选择成员类型宽度和n中较小者；

2 复杂函数

```
  Int *(*(*pfun)(int *))[10];
```

看到这样的表达式估计让不少人都“不寒而栗”了吧，其实虽然看起来复杂，但是构造这类表达式其实只有一条简单的规则：按照使用的方式来声明。

> 首先先介绍一个著名的解析法则：

右左法则：首先从圆括号起，然后向右看，然后向左看，每当遇到圆括号时，就调转阅读方向，当括号内的内容解析完毕，就跳出这个括号，重复这个过程直到表达式解析完毕。

其实我们发现，所谓复杂指针离不开指针函数，函数指针，指针数组，函数指针这四个概念并且括号，*比较多，其实只要我们仔细分析这些看起来复杂的表达式，其实他的逻辑也是很清晰的。

> 举个例子，使用右左法则解析复杂的表达式：

```
  Int *(*(*pfun)(int *))[10];
```

用右左法则解析这个表达式，

首先要找到未定义的标识符pfun，当往右看的时候遇到括号，于是调转方向，再朝相反的方向看，

1 pfun遇到了 * ，说明pfun是一个指针，

> 再往左看又遇到了括号，因此又要调转方向，

2 遇到的是另外一个括号，因此说明指针所指向的是一个函数，

> 函数的参数是一个整型指针。

3 然后又向相反的方向看，又遇到了一个*，说明该函数的返回值又是一个指针，

> 在往左看又遇到括号，所以再次调转方向，把内侧的括号里的内容看完，出了括号遇到的是数组，

4 说明指针所指向的函数的返回值类型的指针指向的是数组

> 这有点向绕口令，但是还是有逻辑可循的。

但是右左法则确实有点麻烦，我们这样看上面这个表达式：首先fpun是一个指向函数的函数指针，该函数有一个整型指针类型的参数并且返回值也是一个指针，所返回的类型指向的是一个数组，并且这个数组有10个元素，每个元素是整型指针类型。

3 C 文件读写

```
LS06File.exe  ./test/input.mp4 ./test/b.mp4
```

上一章我们讲解了 C 语言处理的标准输入和输出设备。本章我们将介绍 C 程序员如何创建、打开、关闭文本文件或二进制文件。

一个文件，无论它是文本文件还是二进制文件，都是代表了一系列的字节。C 语言不仅提供了访问顶层的函数，也提供了底层（OS）调用来处理存储设备上的文件。本章将讲解文件管理的重要调用。

2.1 打开文件

您可以使用 **fopen( )** 函数来创建一个新的文件或者打开一个已有的文件，这个调用会初始化类型 **FILE** 的一个对象，类型 **FILE** 包含了所有用来控制流的必要的信息。下面是这个函数调用的原型：

```
FILE *fopen( const char * filename, const char * mode );
```

在这里，**filename** 是字符串，用来命名文件，访问模式 **mode** 的值可以是下列值中的一个：

模式	描述
r	打开一个已有的文本文件，允许读取文件。
w	打开一个文本文件，允许写入文件。如果文件不存在，则会创建一个新文件。在这里，您的程序会从文件的开头写入内容。如果文件存在，则该会被截断为零长度，重新写入。
a	打开一个文本文件，以追加模式写入文件。如果文件不存在，则会创建一个新文件。在这里，您的程序会在已有的文件内容中追加内容。
r+	打开一个文本文件，允许读写文件。
w+	打开一个文本文件，允许读写文件。如果文件已存在，则文件会被截断为零长度，如果文件不存在，则会创建一个新文件。
a+	打开一个文本文件，允许读写文件。如果文件不存在，则会创建一个新文件。读取会从文件的开头开始，写入则只能是追加模式。

如果处理的是二进制文件，则需使用下面的访问模式来取代上面的访问模式：

```
"rb", "wb", "ab", "rb+", "r+b", "wb+", "w+b", "ab+", "a+b"
```

2.2 关闭文件

为了关闭文件，请使用 fclose( ) 函数。函数的原型如下：

```
 int fclose( FILE *fp );
```

如果成功关闭文件，**fclose( )** 函数返回零，如果关闭文件时发生错误，函数返回 **EOF**。这个函数实际上，会清空缓冲区中的数据，关闭文件，并释放用于该文件的所有内存。EOF 是一个定义在头文件 **stdio.h** 中的常量。

C 标准库提供了各种函数来按字符或者以固定长度字符串的形式读写文件。

2.3 写入文件

下面是把字符写入到流中的最简单的函数：

```
int fputc( int c, FILE *fp );
```

函数 **fputc()** 把参数 c 的字符值写入到 fp 所指向的输出流中。如果写入成功，它会返回写入的字符，如果发生错误，则会返回 **EOF**。您可以使用下面的函数来把一个以 null 结尾的字符串写入到流中：

```
int fputs( const char *s, FILE *fp );
```

函数 **fputs()** 把字符串 **s** 写入到 fp 所指向的输出流中。如果写入成功，它会返回一个非负值，如果发生错误，则会返回 **EOF**。您也可以使用 **int fprintf(FILE *fp,const char *format, ...)** 函数来写把一个字符串写入到文件中。尝试下面的实例：

> **注意：**请确保您有可用的 **tmp** 目录，如果不存在该目录，则需要在您的计算机上先创建该目录。
>
> **/tmp** 一般是 Linux 系统上的临时目录，如果你在 Windows 系统上运行，则需要修改为本地环境中已存在的目录，
>
> 例如: **C:\tmp**、**D:\tmp**等。

2.4 实例

```
#include <stdio.h>

int main()
{
   FILE *fp = NULL;

   fp = fopen("/tmp/test.txt", "w+");
   fprintf(fp, "This is testing for fprintf...\n");
   fputs("This is testing for fputs...\n", fp);
   fclose(fp);
}
```

当上面的代码被编译和执行时，它会在 /tmp 目录中创建一个新的文件 **test.txt**，并使用两个不同的函数写入两行。接下来让我们来读取这个文件。

2.5 读取文件

下面是从文件读取单个字符的最简单的函数：

```
int fgetc( FILE * fp );
```

fgetc() 函数从 fp 所指向的输入文件中读取一个字符。返回值是读取的字符，如果发生错误则返回 **EOF**。下面的函数允许您从流中读取一个字符串：

```
char *fgets( char *buf, int n, FILE *fp );
```

函数 **fgets()** 从 fp 所指向的输入流中读取 n - 1 个字符。它会把读取的字符串复制到缓冲区 **buf**，并在最后追加一个 **null** 字符来终止字符串。

如果这个函数在读取最后一个字符之前就遇到一个换行符 '\n' 或文件的**末尾 EOF**，则只会返回读取到的字符，包括换行符。

您也可以使用

> **int fscanf(FILE *fp, const char *format, ...)**

函数来从文件中读取字符串，但是在遇到第一个空格和换行符时，它会停止读取。

2.6 实例

```
#include <stdio.h>

int main()
{
   FILE *fp = NULL;
   char buff[255];

   fp = fopen("/tmp/test.txt", "r");
   fscanf(fp, "%s", buff);
   printf("1: %s\n", buff );

   fgets(buff, 255, (FILE*)fp);
   printf("2: %s\n", buff );

   fgets(buff, 255, (FILE*)fp);
   printf("3: %s\n", buff );
   fclose(fp);

}
```

当上面的代码被编译和执行时，它会读取上一部分创建的文件，产生下列结果：

```
1: This
2: is testing for fprintf...

3: This is testing for fputs...
```

首先，**fscanf()** 方法只读取了 **This**，因为它在后边遇到了一个空格。其次，调用 **fgets()** 读取剩余的部分，直到行尾。最后，调用 **fgets()** 完整地读取第二行。

2.7 二进制 I/O 函数

下面两个函数用于二进制输入和输出：

```
size_t fread(void *ptr, size_t size_of_elements,
             size_t number_of_elements, FILE *a_file);

size_t fwrite(const void *ptr, size_t size_of_elements,
             size_t number_of_elements, FILE *a_file);
```

这两个函数都是用于存储块的读写 - 通常是数组或结构体。