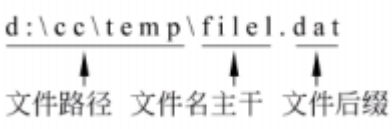
C 文件的有关概念

0. 文件类型：**数据文件**，**程序文件**
1. 程序文件：包括源文件(.c)，目标文件(.obj)，可执行文件(.exe)，这种文件是用来存放程序的，以便实现程序的功能。
2. 数据文件：**供程序运行时读写的数据。**
3. 文件：**存储在外部介质上数据的集合**，如：磁盘，U 盘等。
4. 输入输出流：**表示了信息从源到目的端的流动。**输入操作时，数据从文件流向计算机内存，输出操作时，数据从计算机流向文件(如打印机，磁盘)
5. 流式文件：以字节为单位的文件，由一连串字节组成，中间没有分隔符，对文件的存取是以字节为单位的文件。
6. 数据文件的分类：**ASCII 文件**，也称作文本文件和**二进制文件**
7. 文件标识：**文件路径，文件主干，文件后缀**
    - 

8. 文件缓冲系统：系统自动在内存区为程序中每一个正在使用的文件开辟一个**文件缓冲区。**
9. 文件缓冲区：系统在读写程序时在内存中开辟的数据源与数据目标中间的一个用于保存完整数据内容的缓冲区域。
10. 文件类型指针：**在打开文件时，会在内存建立一个文件信息区，存放文件的有关特征和当前状态。这个信息的数据组织成结构体类型，系统将其命名了 FILE 类型。文件类型指针就是指向文件结构体类型变量的指针，即指向 FILE 文件类型的指针**
11. 通过文件指针访问文件的好处：可以随机访问文件，有效表示数据结构，动态分配内存，方便使用字符串，有效使用数组。
12. 文件信息区：用来存放文件的有关信息(如文件名，文件状态，文件当前位置等)，系统将其命名为 FILE 类型
13. 文件的打开和关闭：所谓的“打开”是指为文件建立相应的信息区和文件缓冲区。“关闭”是指撤销文件信息区和文件缓冲区，使文件指针变量不再指向该文件。

文件缓冲区

1. 定义：C 语言采用"**缓冲文件系统**"处理文件，是指系统自动在内存区为程序中每一个正在使用的文件开辟一个**文件缓冲区**，从内存中向磁盘输出数据必须先送到内存中的缓冲区，装满缓冲区后才送到磁盘，如果从磁盘向内存中读入数据，则一次从磁盘文件将一批数据输入到内存缓冲区(充满缓冲区)，然后再从缓冲区逐个地将数据送到程序数据区(给程序变量)，缓冲区大小由各个具体的 C 编译系统确定。

文件指针类型

1. 定义：缓冲文件系统中，关键的是文件类型指针，简称**文件指针**，每个被使用的文件都在内存中开辟一个相应的**文件信息区**，用来存放文件的有关信息(如文件名，文件状态，文件当前位置等)。这些信息是保存在一个结构体变量中的，该结构体变量由系统声明存放在头文件"stdio.h"中，叫 FILE，在程序中可以直接使用 FILE 类型定义名词。

文件的打开和关闭

1. fopen(文件名，使用文件方式:r/w/a/rb/wb/ab/r+/w+/a+/rb+/wb+/ab+) 失败返回 NULL，成功返回文件首地址。**使用 fopen 时，若以"读"的模式打开，文件必须存在；若以"写"的模式(只要含有 w)打开时，文件可以不存在，会自动创建**
2. fclose(文件指针) 成功返回 0，失败返回 EOF(-1)，**关闭就是撤销文件信息区和文件缓冲区，使文件指针变量不再指向该文件，使文件指针变量与文件脱钩，此后不能通过该指针对原来与其相联系的文件进行读写，，如前所述,在向文件写数据时,是先将数据输出到缓冲区,待缓冲区充满后才正式输出给文件。如果当数据未充满缓冲区而程序结束运行,就会将缓冲区中的数据丢失。用 fclose 函数关闭文件,可以避免这个问题,它先把缓冲区中的数据输出到磁盘文件,然后才释放文件指针变量。**

```
FILE *fp;
fp = fopen("a1", "r");
fclose(fp);
```

文件的顺序读写

向文件读取字符

1. fgetc(fp)： 从 fp 指向的文件读入一个字符，成功返回所读字符，失败返回 EOF(-1)
2. fputc(ch, fp)：把字符 ch 写到文件指针变量 fp 所指向的文件中，成功返回输出字符，失败返回 EOF(-1)，**写入时会清空文件原有的文本**
3. feof(in)：检测文件尾标志 EOF 是否已被读过。被读过表示文件有效字符已全部读完，返回 非 0，否则返回 0
4. 文件读取为什么是按顺序的：访问磁盘文件是逐个字节进行的，为了知道当前访问到第几自己，系统用**文件读写位置标记**来表示**当前所访问的位置**，每访问完一个字节后，当前位置标记就指向下一字节。为了知道对文件的访问是否完成，在一个文件的有效字符的后面一字节中，系统自动设置了一个**文件尾标志**，用 EOF 表示,在 stdio 头文件 EOF 被定义为-1，当读完全部有效字符后，读写位置标记就指向 EOF，再执行一次读操作就读入 EOF,这个时候表示有效字符已经读完。

结构体 struct

1. 定义：由用户自己建立由不同类型数据组成的组合型的数据结构，被称为**结构体**。在其他高级语言中把这种形式的数据结构称为“**记录**”
2. 特点：**结构体变量所占内存长度是各成员占的内存长度之和**，每个成员分别占有自己的内存单元
3. **结构体规则：**
   - 结构体变量中的成员的引用方式为：结构体变量名.成员名
   - 嵌套的结构体，则用类似于 student1.birthday.month
   - 对结构体的变量的成员可以像普通变量一样进行各种运算
   - 同类型的结构体变量可以相互赋值，比如下面例子可以直接 student1 = student2;
   - 可以引用结构体变量成员或结构体变量的地址,

```
scanf("%d", &student2.num);
scanf("%o", &student2);
```

4. **结构体指针：指向结构体数据的指针,一个结构体变量的起始地址就是这个结构体变量的指针。**
5. 定义结构体的方式：**只能对结构体变量赋值，存取或运算，而不能对一个类型赋值，存取或运算。在编译时对类型是不分配空间的，只对变量分配空间**

```
// 定义
struct 结构体名 {  // struct student合起来即为结构体类型名
  成员表列
} 变量名表列
// 例子1：先声明结构体类型，再定义该类型的变量
struct student {
  int num;
  char name[20];
  char sex;
  char addr[20];
}
struct student student1, student2 = { 10101, "shuhongxie", 'M', "guangzhou" };;
// struct student => 结构体类型名  student1,student2 => 结构体变量名
// 例子2：在声明类型的同时定义变量
struct student {
  int num;
  char name[20];
  char sex;
  char addr[20];
} student1, student2 = { 10101, "shuhongxie", 'M', "guangzhou" }; // 直接声明和定义
// 例子3：不指定类型名而直接定义结构体类型变量
struct  {
  int num;
  char name[20];
  char sex;
  char addr[20];
} student1, student2 = { 10101, "shuhongxie", 'M', "guangzhou" }; // 直接声明和定义
```

结构体数组

```
// 定义1
struct 结构体名 {
  成员表列
} 数组名[数组长度];
// 定义2
结构体类型 数组名[数组长度];
struct person leader[3]; // 未初始化
struct person leader[3] = { "Li", 0, "Zhang", 0, "Func", 0 }; // 已初始化

// 示例
struct person {
  char name[20];
  int count;
} leader[3] = {"Li",0,"Zhang",0,"Fun",0};
```

结构体指针

1. 定义：**结构体指针就是指向结构体数据的指针，一个结构体变量的起始地址就是这个结构体变量的指针**
2. 指向运算符：(*p).num 可以变成 p->num;

```
struct student *pt;

struct student {
  int num;
  float score;
  struct student *next;
} std;
pt = &std;
/*
调用函数体的值，可以写成以下三种形式
1. std.num
2. (*p).num
3. p->num
*/
```

用结构体变量和结构体变量的指针做函数参数

1. 结构体变量或结构体成员作为实参进行函数调用时，采取的是“值传递”，等于将结构体变量或结构体成员变量的值传给形参。但如果结构体成员是数组的话除外
2. 结构体指针变量作为实参进行函数调用时，传输的是结构体的地址

结构体和动态链表

1. 使用 malloc 方法**动态分配内存**

共用体类型

1. 特点：**结构体变量所占内存长度是各成员占的内存长度之和，共用体变量所占内存长度等于最长的成员的长度**
2. 由于共同体变量中的各个成员共用同一块存储空间，因此，在任一时刻，只能存放一个成员的值。
3. 共用体变量中起作用的成员值是最后一次被赋值的成员值。即再次赋值会覆盖之前的值。
4. 共用体变量的地址和它成员的地址都是同一地址
5. **可以对结构体进行赋值，但不能对共用体变量赋值也不能企图引用共用体变量来得到成员的值。**

```
union data {
  short int i;
  char ch;
  float f;
} a, b,c;
// 共用体不能用变量进行初始化 比如  a = { 2, 'Z', '3.1' }
// 只能这样
a.i = 20;
a.ch = 'X';
a.f = 3.14142;
// 不能引用共用体变量，只能引用共用体变量中的成员 比如a.i
```

枚举类型(不重要)

```
// 定义
enum weekday { sun, mon, tue, wed, thu, fri, sat }; // sun,mon,tue 被称为枚举元素或枚举常量，他们是
// 使用
// workday,week_end 被称为枚举变量，他们的值只能是sun到sat之一
enum weekday workday,week_end; // 枚举变量
workday = mon;
week_end = sun
// 也可以直接定义枚举变量
enum weekday { sun, mon, tue, wed, thu,fri,sat } workday,week_end;
```

共用体类型

1. 特点：**结构体变量所占内存长度是各成员占的内存长度之和，共用体变量所占内存长度等于最长的成员的长度**