第4章 结构体与文件读写

In [1]:
```
#include <iostream>
#include <cstdio>
using namespace std;
```

4.1 结构体

结构体的定义和访问

In [2]:
```
struct Student{
    char name[20];
    unsigned ID;
    float math_score;
};
```

标准初始化形式

In [3]:
```
Student s1 = Student({"zhangsan",28,93.5});
cout << s1.name << endl << s1.ID << endl << s1.math_score << endl;
```

```
zhangsan
28
93.5
```

省略类型强制转换函数

In [4]:
```
Student s2 = {"lisi",32,88};
cout << s2.name << endl << s2.ID << endl << s2.math_score << endl;
```

```
lisi
32
88
```

将参数写在变量名后面，用括号括起来

In [5]:
```
Student s3({"wangwu",12,60}) ;
cout << s3.name << endl << s3.ID << endl << s3.math_score << endl;
```

```
wangwu
12
60
```

使用指针访问结构体的成员变量

In [6]:
```
Student *p = &s3;
```

In [7]:
```
cout << p->name;
```

```
wangwu
```

In [8]:
```
cout << (*p).name;
```

```
wangwu
```

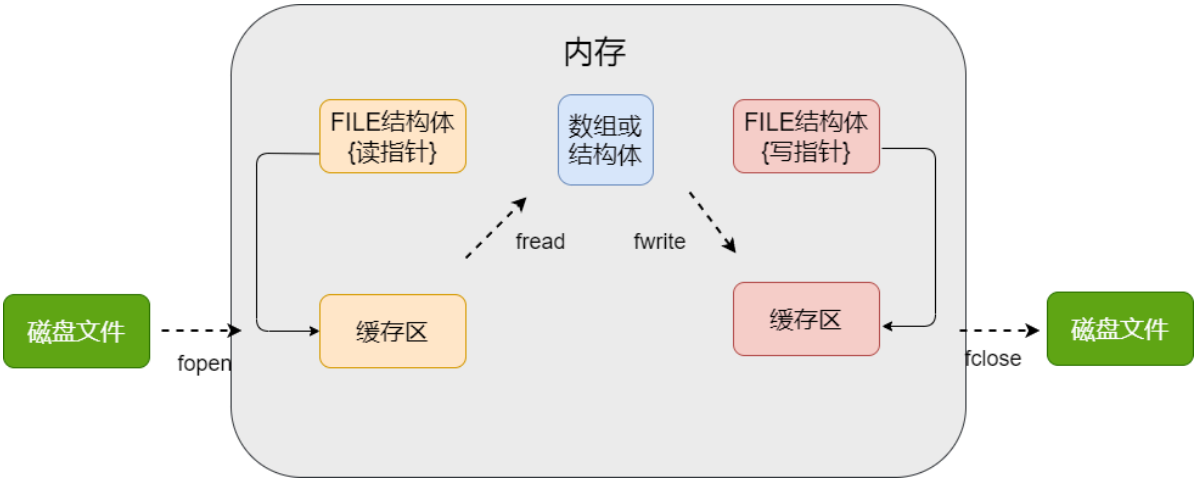
结构体的功能

结构体里面可以存放大量数据用于管理某个数据结构，管理数据结构的方法就是在外面写一堆函数；
由于这些函数包含各种调用，如果程序太大了，就不清楚某个函数到底操作哪个结构体，带来麻烦，
因此，后面引入了对象的概念，这样就把数据和函数封装到一个对象当中，更便于管理。

4.2 文件读写

原理

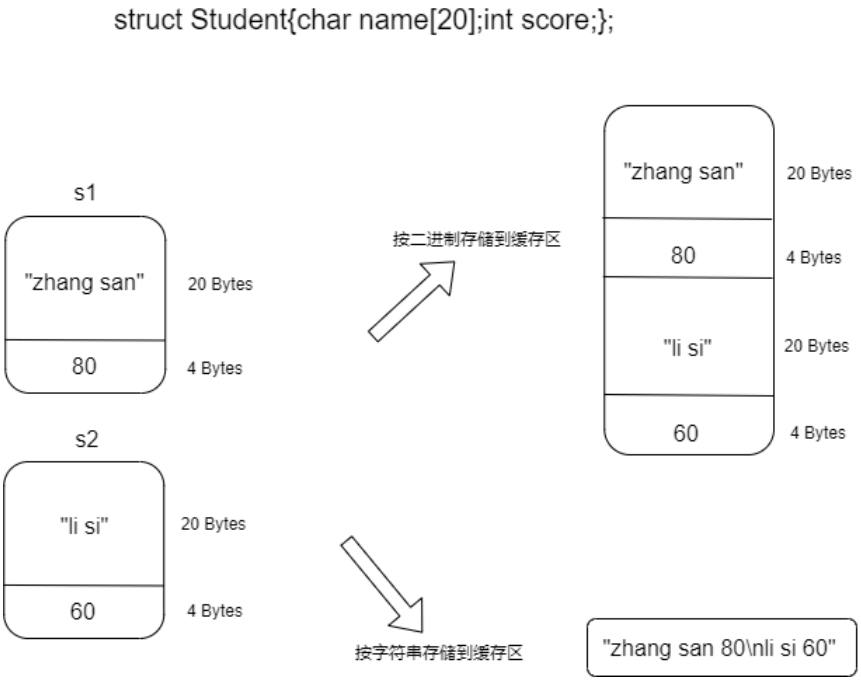
从硬盘move数据到内存上面的指令仅限于内核态所在的内存地址使用（硬件实现），因此操作系统写了一些
底层的读写程序，运行在内核态，经过层层封装，在用户态也提供了相应的读写接口，也就是read和write函数，
然后c语言利用操作系统提供的读写接口，进一步封装成自己的文件打开和读写访问接口，c语言用于管理文件的
数据结构就是一个FILE结构体，其中fopen函数返回的类型就是FILE指针，这个指针可以访问结构体里面的相关属性，
比如这个结构体里面定义了缓存区，读指针，写指针，接着fread和fwrite函数通过FILE指针对缓存区的数据进行读写，
并同步更新结构体里面的属性，实时记录读写指针的位置，最后通过fclose把缓存区数据写回磁盘



补充说明：
c++中用于读写文件的对象实际上就是把上面FILE结构体里面的变量变成对象的属性，
把fopen，fread，fwrite，fclose这些函数封装为对象的方法，这里就不赘述了。

文本文件与二进制文件的存储
比如内存空间一个20字节的字符数组和一个int整数，
按照二进制文件存储，就是把这24个字节原模原样复制到
缓存区，然后保存为文件，原来变量是什么样的就保存为什么样的；
如果按照文本文件存储,那么把上面的字符串和整数存储到文本文件缓存区时，
首先需要把各种数据类型全部转化为一个一个字符串，然后一个一个字符串写入
缓存区，原先如果是字符串"hello"，存进去还是hello，只不过原先放在占用
20个字节的数组中，现在就按"hello"字符串的长度进行存储，因为fputs遇到'\0'
就停止写入，然后对于整数80，需要手动转换成字符串"80"进行存储，因为fputs只能
接收一个字符串变量，不能接收其他类型的变量，这时候作为字符串"80"，只占用2个字节，
也就是不管原来的变量是什么类型，全都变成字符串进行存储。

二进制文件优势：
1.节约空间，比如几十亿的数字，用int保存只需要4个字节，转换成字符串保存就是一长串字符。
2.固定长度存储，变量占多少空间，原模原样按字节拷贝过去，可以任意定位到任何一个变量。



注意，上面按字符串进行存储时，换行符写入到磁盘文件会根据不同操作系统进行改动，
在windows系统中，会变成"\r\n",在mac系统中，会变成"\r",在linux系统中，就是"\n".
同理，从磁盘读取到缓存区时，无论什么形式的换行符，都会变成'\n'.

另一方面，如果把windows下的文件拷贝到linux系统下，以文本的形式打开，可能换行无法正确显示，
但是由于现在的文本编辑器，比如记事本，具备自动识别的功能，所以还是能正确显示

二进制文件读写
主要使用fread和fwrite两个函数

1.先定义两个结构体，然后把这两个结构体写入文件
fwrite(变量名地址，一个写入单位大小，写入多少个单位，文件指针)

In [9]:
```
struct Student{char name[20];int score;};
Student s1({"zhang san",80});
Student s2({"li si",60});
```

In [10]:
```
FILE *file1 = fopen("students.data","wb");
```

In [11]:
```
fwrite(&s1,sizeof(s1),1,file1);
fwrite(&s2,sizeof(s2),1,file1);
```

In [12]:
```
fclose(file1);
```

2.然后打开上面的文件，存储新定义的结构体变量中，并输出
fread(变量名地址，一个读取单位大小，读取多少个单位，文件指针)
返回值：返回成功读取的字节数量，如果返回值为0，表示读取结束

In [13]:
```
Student s3;
```

In [14]:
```
FILE *file2 = fopen("students.data","rb");
```

In [15]:
```
fread(&s3,sizeof(s1),1,file1);
cout << s3.name << " " << s3.score << endl;
fread(&s3,sizeof(s2),1,file1);
cout << s3.name << " " << s3.score << endl;
```

```
zhang san 80
li si 60
```

In [16]:
```
fread(&s3,sizeof(s2),1,file1)==0  // 由于文件已经读完，读不到字符，所以返回的字符数为0
```

Out[16]:
```
true
```

In [17]:
```
fclose(file2);
```

文本文件读写
主要使用fgets和fputs两个函数

1.先将s1和s2写入文本文件中
fputs(字符串，文件指针)
直接将字符串写入文件，需要什么字符串写什么，遇见字符串结尾，也就是'\0'，停止写入

In [18]:
```
FILE *file3 = fopen("students.txt","w");
```

In [19]:
```
fputs(s1.name,file3);  //写入名字“zhang san”，不包括'\0'，占用9个字节
fputs(" ",file3);      //写入空格，占用1个字节
fputs("80",file3);     //写入字符串“80”，占用2个字节
fputs("\n",file3);     //写入换行符，由于我在linux环境下，所以占用1个字节
```

Out[19]:
```
1
```

In [20]:
```
fputs(s2.name,file3);  //写入名字“li si”，占用5个字节
fputs(" ",file3);      //写入空格，占用1个字节
fputs("60",file3);     //写入字符串“60”，占用2个字节
fputs("\n",file3);     //写入换行符，占用1个字节
```

Out[20]:
```
1
```

In [21]:
```
fclose(file3);         //总计22个字节
```

Out[21]:
```
0
```

2.再读取刚刚的文本文件
fgets(字符串变量,读取的字节数量,文件指针);
返回值：返回读取的字符串变量名，如果没有，返回空指针NULL
下面三种情况会读取结束：
1.已经读取到了指定的字节数量
2.遇到了换行符
3.读到了文件结尾，没东西读了

In [22]:
```
FILE *file4 = fopen("students.txt","r");
```

In [23]:
```
char temp_str[20];
```

In [24]:
```
fgets(temp_str,100,file4);
cout << temp_str;
```

```
zhang san 80
```

In [25]:
```
fgets(temp_str,100,file4);
cout << temp_str;
```

```
li si 60
```

In [26]:
```
fgets(temp_str,100,file4)==NULL
```

Out[26]:
```
true
```

In [27]:
```
fgets(temp_str,100,file4)==0    // 由于NULL就是0.所以和上面二进制文件读取保持一致，这里也用0判断文件读取结束
```

Out[27]:
```
true
```

In [28]:
```
fclose(file4);
```

In [ ]:
```

```