第七章 文件 与 结构体

文件是根据特定的目的而收集在一起的有关数据的集合。C++把每一个文件都看成是一个有序的字节流，每个文件都以文件结束标志结束，如果要操作某个文件，程序必须首先打开该文件。当一个文件被打开后，该文件就和一个流关联起来，这里的流实际上是一个字节序列。

C++将文件分为文本文件和二进制文件。二进制文件一般含有特殊的格式或计算机代码，如图文件和可执行文件等。文本文件则是可以用任何文字处理程序阅读和编辑的简单ASCII文件。

下面我们学习如何编写C++代码来实现对文本文件的输入和输出。

第一节　文件操作

C++语言提供了一批用于文件操作的标准函数，本节不是介绍文件打开函数fopen，而是介绍另一个函数freopen，它们都包含于标准库cstdio中，文件操作基本步骤如下：

(1)打开文件，将文件指针指向文件，决定打开文件类型；

(2)对文件进行读、写操作；

(3)在使用完文件后，关闭文件。

一、重定向版

【命令格式】

- FILE * freopen (const char * filename, const char * mode, FILE * stream);

【参数说明】

- filename: 要打开的文件名
- mode: 文件打开的模式，和fopen中的模式(r/w)相同
- stream：文件指针，通常使用标准流文件(stdin/stdout/stderr)
- 其中stdin是标准输入流，默认为键盘；stdout是标准输出流，默认为屏幕；
- stderr是标准错误流，一般把屏幕设为默认。通过调用freopen，就可以修改标准流文件的默认值，实现重定向。

【使用方法】

- 因为文件指针使用的是标准流文件，因此我们可以不定义文件指针。接下来我们使用freopen()函数以只读方式r(read)打开输入文件slyar.in。
- 格式：freopen("slyar.in", "r", stdin);
- 然后使用freopen()函数以写入方式w(write)打开输出文件slyar.out。
- 格式：freopen("slyar.out", "w", stdout);
- 接下来的事情就是使用freopen()函数的优点了，我们不再需要修改scanf，printf，cin和cout。而是维持代码的原样就可以了。因为freopen()函数重定向了标准流，使其指向前面指定的文件，省时省力。最后只要使用fclose关闭输入文件和输出文件即可。
- 格式：fclose(stdin);fclose(stdout);
- 若要恢复句柄，可以重新打开标准控制台设备文件，只是这个设备文件的名字是与操作系统相关的。

- 格式：freopen("CON", "r", stdin);

代码模版:

```
#include<cstdio>          //使用freopen语句，须调用cstdio库
    int main()
    {
        freopen("slyar.in", "r", stdin);
        freopen("slyar.out", "w", stdout);

        /* 中间按原样写代码，什么都不用修改 */

        fclose(stdin);fclose(stdout);
        return 0;
    }
```

例7.1　从in.txt文件中读入数据，把它们的和保存out.txt文件中。

```
#include<cstdio>
int main()
{
   freopen("in.txt","r",stdin);              //定义输入文件名
   freopen("out.txt","w",stdout);            //定义输出文件名
   int temp,sum=0;
   while (scanf("%d",&temp)==1)       //(cin>>temp)从输入文件中读入数据
                                          //在C++中非0为真
     {
        sum=sum+temp;
     }
   printf("%d\n",sum);                       // cout<<sum<<endl;
   fclose(stdin);fclose(stdout);             //关闭文件，可省略
   return 0;
}
```

in.txt数据：

```
    1  2  3  4  5
```

out.txt结果：

```
    15
```

说明：　while (fin>>temp)和(scanf("%d",&temp)==1)主要是用于判断数据是否已经读完，以便及时终止循环。还可以用成员函数eof来判断是否达到数据流的末尾。对scanf、printf和cin、cout语句都适用。

二、fopen版

- 重定向用起来很方便，但并不是所有算法竞赛都允许读写文件。甚至有的竞赛允许访问文件，但不允许使用freopen这样的重定向方式读写文件，可以使用fopen版，对scanf和printf语句适用。程序如下：

```
#include<cstdio>
using namespace std;
int main()
{
   FILE *fin,*fout;
   fin = fopen("in.txt","rb");                  //定义输入文件名
   fout = fopen("out.txt","wb");                //定义输出文件名
   int temp,sum=0;
   while (fscanf(fin,"%d",&temp)==1)   //从输入文件中读入数据
    {
       sum=sum+temp;
    }
   fprintf(fout,"%d\n",sum);                    // cout<<sum<<endl;
   fclose(fin);fclose(fout);                   //关闭文件，可省略
   return 0;
}
```

先声明变量**fin**和**fout**（暂且不用管**FILE ***为何物），把**scanf**改成**fscanf**，第一个参数为**fin**；把**printf**改成**fprintf**，第一个参数为**fout**，最后执行**fclose**，关闭两个文件。

重定向和**fopen**两种方法各有优劣。重定向的方法写起来简单、自然，但是不能同时读写文件和标准输入输出；**fopen**的写法稍显繁琐，但是灵活性比较大（例如可以反复打开并读写文件）。顺便说一句，如 果把**fopen**版的程序改成读写标准输入输出，只需赋值**fin=stdin**；**fout=stdout**；即可，不要调用**fopen**和**fclose**。

程序如下：

```
#include<cstdio>
using namespace std;
int main()
{
   FILE *fin,*fout;
   fin=stdin;
   fout=stdout;
   /*   本处语句同上   */
   fprintf(fout,"%d\n",sum);
   return 0;
}
```

三、文件输入输出流

- 在C++中，文件输入流(ifstream)和文件输出流(ofstream)的类，它们的默认输入输出设备都是磁盘文件。C++可以在创建对象时，设定输入或输出到哪个文件。由于这些类的定义是在fstream中进行的，因此，在使用这此类进行输入输出操作时，必须要在程序的首部利用#include指令包进fstream头文件。

- 例如：若想用fin作为输入对象，fout作为输出对象，则可以使用如下定义：
- ifstream fin("输入文件名.扩展名");
- ofstream fout("输出文件名.扩展名");

程序如下：

```
#include<fstream>                                    //使用文件输入输出流，对cin、cout
语句适用
using namespace std;
int main()
{
   ifstream fin("in.txt");                          //定义输入文件名
   ofstream fout("out.txt");                        //定义输出文件名
   int temp,sum=0;
   while (fin>>temp)  sum=sum+temp;   //从输入文件中读入数据
   fout<<sum<<endl;
   fin.close();fout.close();                        //关闭文件，可省略
   return 0;
}
```

如果想再次使用cin和cout，是否要逐个把程序中的所有fin和fout替换为cin和cout？不用这么麻烦，只需要把fin和fout的声明语句去掉，并加上这样两行即可：

```
#define fin cin
#define fout cout
```

用条件编译，还可以让程序在本机上读写标准输入输出，比赛测试时读写文件（请自行实验）。

第二节　结构体

在实际问题中，一组数据往往具有不同的数据类型。例如，人口大普查时，我们需要记录每一位公民的姓名，年龄，性别，住址，身份证号码。这些信息分别要用整型，字符型，字符串型来记录。为了解决问题，C++语言给出了另一种构造数据类型——“结构体”，它在数据存储方面相当于其他高级语言中的记录，但它有着面向对象的优势。

7.2.1 结构体定义和操作

1. 定义结构体及结构体变量

结构体变量的定义有两种方式：

（1）定义结构体的同时定义结构体变量

```
struct 结构体名 {          //其中 struct 是关键字
    成员表                 //可以有多个成员
    成员函数       //可以有多个成员函数，也可以没有
} 结构体变量表;     //可以同时定义多个结构体变量
```

结构体变量名列表的各个变量用“，”隔开。

例如：

```
struct DATA{
    int a[2];                          //成员为一个数组
    int c;
}data_a,data_b;
```

（2）先定义结构体再定义结构体变量

```
struct 结构体名 {
    成员表
    成员函数
} ;
```

结构体名 结构体变量表；//同样可以同时定义多个结构体变量

例如：

```
struct DATA{
    int a[2];
    int c;
};
```

DATA data_a,data_b;//这种定义方式与上一种方式的效果是相同的

在定义结构体变量时注意，结构体变量名和结构体名不能相同。在定义结构体时，系统对之不分配实际内存。只有定义结构体变量时，系统才为其分配内存。

2.成员调用

结构体变量与各个成员之间引用的一般形式为：

结构体变量名. 成员名

对于上面定义的结构体变量，我们可以这样操作：

```
cin>>data_a.a[0]>>data_a.a[1];  //一般情况下不能写 cin>>data_a;
int a=data_a.a[0]+data_a.a[1];   //就像用整形变量一样用a[0]、a[1]
data_b=data_a;                   //结构体之间的相互赋值是合法的
data_a.c=0;                      //就如同给整形变量赋值
```

实际上结构体成员的操作与该成员类型所具有的操作是一致的。

成员运算符“.”在存取成员数值时使用，其优先级最高，并具有左结合性。在处理包含结构体的结构体时，可记作：

strua. strub. membb

这说名结构体变量 strua 有结构体成员 strub；结构体变量 strub 有成员 membb。

3.成员函数调用

- 结构体成员函数调用的一般形式为：
- 结构体变量名. 成员函数
-
- 结构体成员函数默认将结构体变量作为引用参数。

7.2.2 结构体操作实例

现在，我们先定义一个简单的结构体，这个结构体将用来记录一个学生的大致情况，所以它的成员应该有学号、姓名、性别、年龄、成绩、家庭住址等。

```
#include<iostream>
using namespace std;
struct student{
        int num;                //学号
        char name[21];          //姓名
        char sex;               //性别
        int age;                //年龄
        float score;            //成绩
        char address[51];       //家庭住址
};  //此处不可忽略分号
struct student a,b;
int main()
{
 cin>>a.num>>a.name>>a.sex>>a.age>>a.score>>a.address;
 cin>>b.num>>b.name>>b.sex>>b.age>>b.score>>b.address;
 cout<<a.num<<' '<<a.name<<' '<<a.sex<<' '<<a.age<<' '<<a.score<<' '<<a.address<<endl;
 cout<<b.num<<' '<<b.name<<' '<<b.sex<<' '<<b.age<<' '<<b.score<<' '<<b.address<<endl;
 return 0;
}
```

这里再举出一个的例子，希望竞赛学子能够举一反三，从中受益。

```
#include<iostream>
using namespace std;
struct DATA{
    int a[2];                    //成员为一个数组
    int c;                  //用来计算总和
    int max()                    //定义成员函数
    {
        return a[0]>a[1]?a[0]:a[1];  //默认该结构体变量的成员作为引用参数
    }
    }data_a[5];              //我们可以定义结构体数组
   /*结构体的初始化，按成员定义的顺序赋值，每个成员用“,”隔开*/
    DATA  data_b={{15,20},35};
  /*每个成员初始化和同类型变量初始化方式相同*/
```

```
int main() {
    cout<<data_b.max()<<endl;
    for (int i=0;i<5;++i) {
        cin>>data_a[i].a[0]>>data_a[i].a[1];
        data_a[i].c=data_a[i].a[0]+data_a[i].a[1];
        }
        for (int i=0;i<5;++i)
            cout<<data_a[i].max()<<' '<<data_a[i].c<<endl;
        return 0;
    }
```

程序会先输出：

20

然后等待我们输入10个数，假设输入的数字为：

19 63 25 36 10 12 25 96 36 12

我们将得到如下输出：

63 82

36 61

12 22

96 121

36 48

从上述例子可以看出，结构体支持初始化，定义为数组，成员运算等多种操作。不仅如此，我们还可以通过重载运算符等方法使结构体的操作像int,double一样方便简洁。但，由于那些内容过多的涉及面向对象编程，所以在此省去。