第十章 输入/输出

输入/输出（I/O）概述

- 输入/输出（简称I/O）是程序的一个重要组成部分：
  - 程序运行所需要的数据往往要从外设（如：键盘、文件等）得到
  - 程序的运行结果通常也要输出到外设（如：显示器、打印机、文件等）中去。
- 在C++中，输入/输出不是语言定义的成分，而是由具体的实现（编译程序）作为标准库的功能来提供。

C++的I/O流

- 在C++中，输入/输出操作是一种基于字节流的操作：
  - 在进行输入操作时，可把输入的数据看成逐个字节地从外设流入到计算机内部（内存）；
  - 在进行输出操作时，则把输出的数据看成逐个字节地从内存流出到外设。
- 在C++的标准库中，除了提供基于字节的输入/输出操作外，为了方便使用，还提供了基于C++基本数据类型数据的输入/输出操作。
- 在C++程序中也可以对类库中输入/输出类的一些操作进行重载，使其能对自定义类的对象进行输入/输出操作。

I/O的分类

- 基于控制台的I/O：
  - 从标准输入设备（如：键盘）获得数据
  - 把程序结果从标准输出设备（如：显示器）输出
- 基于文件的I/O：
  - 从外存文件获得数据
  - 把程序结果保存到外存文件中
- 基于字符串变量的I/O：
  - 从程序中的字符串变量中获得数据
  - 把程序结果保存到字符串变量中

C++输入输出的实现途径

- 过程式——通过从C语言保留下来的函数库中的输入/输出函数来实现。

- 面向对象——通过C++的I/O类库中的I/O类来实现。

基于函数库的控制台I/O

- 方法：调用C++标准库函数实现输入输出

  ```
  #include <stdio>
  ```

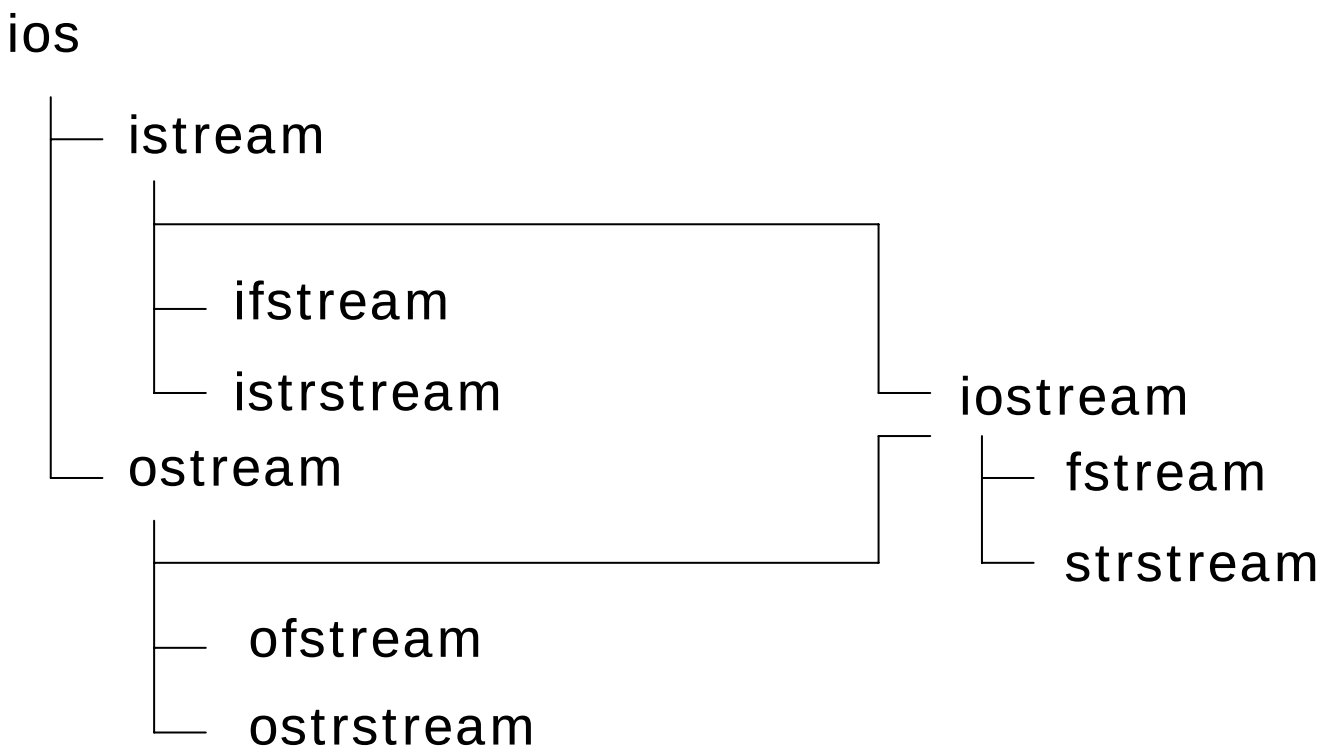
- 常用的I/O函数
  - 输入
    - int getchar(); //从键盘读入一个字符
    - char * gets(char *p)//从键盘读入一个字符串到p
    - int scanf(const char *format[,<参数表>]);//从键盘读入特定类型的数据到参数表
  - 输出
    - int putchar(int ch)；//输出一个字符到标准设备
    - int puts(const char *p);//输出一个字符串
    - int printf(const char * format[,<参数表>]);//将参数表的数据按照指定格式输出
  - 常用格式控制字符
    - %c //char类型
    - %d //int类型
    - %f //double类型
    - %s //字符串类型

函数库I/O的例子

```
#include <stdio>

int x;
double y;
scanf(“x=%d,y=%f”,&x,&y);
//输入为 x=10,y=5.0

char password[10];
scanf(“Password=%6s”,password); //把输入的前6个字符放入s中

sprintf(“x=%d,y=%f”,100,20);
//输出为x=100,t=20.0

sprintf(“password=%s”,password); //输出密码字符串
```

基于类库的控制台I/O

- C++的I/O类库提供了以面向对象方式进行输入/输出。以下是I/O类库中基本的I/O类：



- 在进行输入/输出时，首先创建一个I/O类的对象，然后，通过该对象（调用对象类的成员函数）进行输入/输出操作。
- istream类重载了操作符“>>”（抽取），用它可以进行基本类型数据的输入操作。例如：

```
istream in(...);
in >> x; //x是一个变量
in >> y; //y是一个变量
```

或

```
in >> x >> y;
```

- ostream类重载了操作符“<<”（插入），用它可以进行基本数据类型数据的输出操作。例如：

```
ostream out(...);
out << e1; //e1是一个表达式
out << e2; //e2是一个表达式
```

或

```
out << e1 << e2;
```

基于类库的控制台I/O的例子

```
#include <istream>
#include <ostream>

istream in;
ostream out;
int x,y;
in>>x>>y; //从键盘读入两个数到x和y
out<<x<< “+” <<y<< “=” <<x+y;
   //输出x+y的值
```

控制台I/O类库

- 在**I/O**类库中预定义了四个**I/O**对象：**cin**、**cout**、**cerr**以及**clog**，可以直接利用这些对象进行控制台的输入/输出操作：
  - cin属于istream类的对象，它对应着计算机系统的标准输入设备;
  - cout属于ostream类的对象，它对应着计算机系统的用于输出程序正常运行结果的标准输出设备;
  - cerr和clog属于ostream类的对象，它们对应着计算机系统的用于输出程序错误信息的设备，通常情况下它们都对应着显示器;
- 在进行控制台输入/输出时，程序中需要有下面的包含命令：

```
#include <iostream>
```

控制台输出cout例子

```
#include <iostream>
using namespace std;
.......
int x;
float f;
char ch;
.......
cout << x ; //输出x的值。
cout << f; //输出f的值。
cout << ch; //输出ch的值。
cout << "hello"; //输出字符串"hello"。
或
cout << x << f << ch << "hello" << p;
```

输出格式控制

- 为了对输出格式进行进一步的控制，可以通过输出一些操纵符（manipulator）来实现，例如：

```
#include <iostream>
#include <iomanip>  //操纵符声明的头文件。
using namespace std;
.....
int x=10;
cout << hex << x << endl; //以十六进制输出x的值，然后换行。
```

常用输出操纵符

操纵符	含义
endl	输出换行符，并执行flush操作
flush	使输出缓存中的内容立即输出
dec	十进制输出
oct	八进制输出
hex	十六进制输出
setprecision(int n)	设置浮点数有效数字的个数或小数点后数字的位数
setiosflags(long flags)/ resetiosflags(long flags)	设置/取消输出格式，**flags**的取值可以是：**ios::scientific**（以指数形式显示浮点数），**ios::fixed**（以小数形式显示浮点数），等等

控制台输入cin的例子

```
#include <iostream>
using namespace std;
......
int x;
double y;
char str[10];
cin >> x; cin >> y; cin >> str;
```

或者

```
cin >> x >> y >> str;
```

- 在输入时，各个数据之间用空白符分开。

输入/输出操作符“>>”和“<<”的重载

- 为了能用抽取操作符“>>”和插入操作符“<<”对自定义类的对象进行输入/输出操作，就需要针对自定义的类重载插入操作符“<<”和抽取操作符“>>”。
- 对自定义的类重载插入操作符“<<”和抽取操作符“>>”时，须作为全局函数来重载。

插入操作符“<<”的重载的例子

```
class A
{    int x,y;
   public:
      A(int i,int j) {x=i,y=j};
      ......
   friend ostream& operator << (ostream& out,
                                const A &a);
};
ostream& operator << (ostream& out, const A &a)
{  out << “x=” << a.x << “, y=” << a.y;
   return out;
}
.....
A a(2,3),b(6,7);
cout << a << endl << b; //输出结果是什么?
```

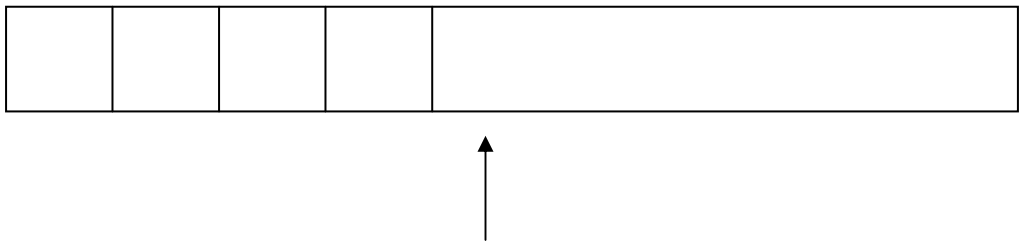
文件I/O

- 程序运行结果有时需要永久性地保存起来，以供其它程序或本程序下一次运行时使用。程序运行所需要的数据也常常要从其它程序或本程序上一次运行所保存的数据中获得。
- 用于永久性保存数据的设备称为外部存储器（简称：外存），如：磁盘、磁带、光盘等。
- 在外存中保存数据的方式通常有两种：文件和数据库。本课程只介绍以文件方式来永久性地保存数据。

文件的基本概念

- 在C++中，把文件看成是由一系列字节所构成的字节串，称为流式文件，对文件中数据的操作（输入/输出）通常是逐个字节顺序进行。
- 在对文件数据进行读写的过程：
  - 打开文件。目的是：在程序内部的一个表示文件的变量/对象与外部的一个具体文件之间建立联系。
  - 文件读/写
  - 关闭文件。目的是把暂存在内存缓冲区中的内容写入到文件中，并归还打开文件时申请的内存资源。

- 每个打开的文件都有一个内部的位置指针，它指出文件的当前读写位置。
- 进行读/写操作时，每读入/写出一个字节，文件位置指针会自动往后移动一个字节的位置。



文件数据的存储方式

- 文本方式（**text**）
  - 只包含可显示字符和有限的几个控制字符（如：‘\r’、‘\n’、‘\t’等）；
  - 一般用于存储具有“行”结构的文本数据；
  - 文本方式存储整数1234567：
    - 依次把1、2、3、4、5、6、7的ASCII码（共7个字节）写入文件。

- 二进制方式（**binary**）
  - 可以包含任意的二进制字节；
  - 一般用于存储无显式结构的数据；
  - 二进制方式存储整数1234567：
    - 把整数1234567的计算机内部表示（假设为32位计算机）：0012D687（十六进制，4个字节：00 12 D6 87）写入文件。

写文件操作

- 在利用I/O类库中的类进行外部文件的输入/输出时，程序中需要下面的包含命令：

```
#include <ifstream>
#include <ofstream>
```

- 写文件步骤：
  - （1）创建一个ofstream类（是ostream类的派生类）的对象。
  - （2）打开文件：建立ofstream类的对象与外部文件之间的联系。
  - （3）调用类库的成员函数来写文件

建立ofstream类的对象与外部文件联系的方式

- 直接方式：在创建**ofstream**类的对象时指出外部文件名和打开方式。例如：

```
ofstream out_file(<文件名>,<打开方式>);
```

- 间接方式是在创建了**ofstream**类的对象之后，调用**ofstream**的一个成员函数**open**来指出与外部文件的联系。例如：

```
ofstream out_file;
  out_file.open(<文件名>,<打开方式>);
```

文件输出的打开方式

- 打开方式：
  - ios::out，含义是打开一个外部文件用于写操作，如果外部文件已存在，则首先把它的内容清除；否则，先创建该外部文件。
  - ios::app，含义是打开一个外部文件用于添加（从文件末尾）操作。如果外部文件不存在，则先创建该外部文件。
  - 默认以文本方式打开文件，也可以指定以二进制方式打开

- 打开文件时，必须要对文件打开操作的成功与否进行判断。判断文件是否成功打开可以采用以下方式

```
if (!out_file) //或：out_file.fail()
                    //或：!out_file.is_open()
{ ..... //失败处理
}
```

- 文件成功打开后，可以使用插入操作符“<<”或ofstream类的一些成员函数来进行文件输出操作，包括：
  - fputc(),fputs,fprintf()
  - <<符 //插入符
  - fwrite() //按字节写数据，二进制方式

- 文件输出操作结束时，要使用ofstream的一个成员函数close关闭文件：

```
out_file.close();
```

写文件的例子

```
#include <ofstream>

ofstream out_file("d:\\myfile.dat",ios::out);
if (!out_file) exit(-1);
int x;
double y;
......
out_file << x << ',' << y << endl;
out_file.put('A');
out_file.write("ABCDEFG",7);
out_file.close();
```

文件输入

- 首先创建一个**ifstream**类（**istream**类的派生类）的对象，并与外部文件建立联系。例如：

  ifstream in_file(<文件名>,<打开方式>);

  或

  ifstream in_file;

  in_file.open(<文件名>,<打开方式>);

- 打开方式：
  - ios::in，它的含义与fopen的打开方式“r”相同
  - 也可以把它与ios::binary通过按位或（|）操作实现二进制打开方式。默认为文本方式。
  - 对以文本方式打开的文件，当文件中的字符为连续的'\r'和'\n'时，在某些平台上（如：DOS和Windows平台）将自动转换成一个字符'\n'输入。

- 判断打开成功的方式与输出文件对象相同。
- 文件成功打开后，可以使用抽取操作符“>>”或ifstream类的一些成员函数来进行文件输出操作，例如：

```
ifstream in_file("d:\\myfile.dat",ios::in);
if (!in_file) exit(-1);
char ch,buf[11];
int x;
double y;
......
in_file >> x >> ch >> y;
in_file.get(ch);
in_file.read(buf,7);
buf[7] = '\0';
```

- 判断文件是否结束可以调用ios类的成员函数eof来实现：

  ```
  int ios::eof();
  ```

  - 该函数返回0表示文件未结束；返回非0表示文件结束。
- 文件输出操作结束时，要使用ifstream的一个成员函数close关闭文件：

  ```
  in_file.close();
  ```
- 注意：从文件输入必须要知道文件中数据的存储格式！

文件输入/输出

- 如果需要打开一个既能读入数据、也能输出数据的文件，则需要创建一个**fstream**类的对象（类**fstream**是类**iostream**的派生类）。

- 在创建**fstream**类的对象并建立与外部文件的联系时，文件打开方式应为：**ios::in|ios::out**（可在文件任意位置写）或**ios::in|ios::app**（只能在文件末尾写）

字符串I/O

- 程序中的有些数据并不直接输出到标准输出设备或文件，而是需要保存在程序中的某个字符串变量中。

- 首先需要创建类istrstream、ostrstream或strstream的一个对象
  - 对于ostrstream类：

```
char buf[100];
ostrstream str_buf;
```

或

```
ostrstream str_buf(buf,100);
```

  - 对于istrstream类

```
char buf[100];
..... //通过某种途径在buf中存放了一些字符。
istrstream str_buf(buf);
```

或

```
istrstream str_buf(buf,100);
```

- 然后可以用与控制台和文件输入/输出类似的操作进行基于字符串变量的输入/输出。

小结

- 输入/输出（I/O）概述
- 控制台I/O
- 文件I/O
- 字符串I/O