C++语言程序设计：MOOC版

清华大学出版社（ISBN 978-7-302-42104-7）

第9章 流类库与文件读写

中国農業大學 阚道宏

第9章 流类库与文件读写

- C++语言的输入与输出
  - 将提供输入数据的数据源称作输入数据流
  - 将输出数据时的目的地称作输出数据流
  - 输入数据流和输出数据流统称为输入/输出流

- C语言的输入/输出函数 vs C++语言的输入/输出流类

- 流类库：以ios为基类的类族
  - 通用输入/输出流类：提供通用的输入/输出（简称标准I/O）功能。
  - 文件输入/输出流类：提供文件输入/输出（简称文件I/O）功能。
  - 字符串输入/输出流类：提供字符串输入/输出（简称字符串I/O）功能

第9章 流类库与文件读写

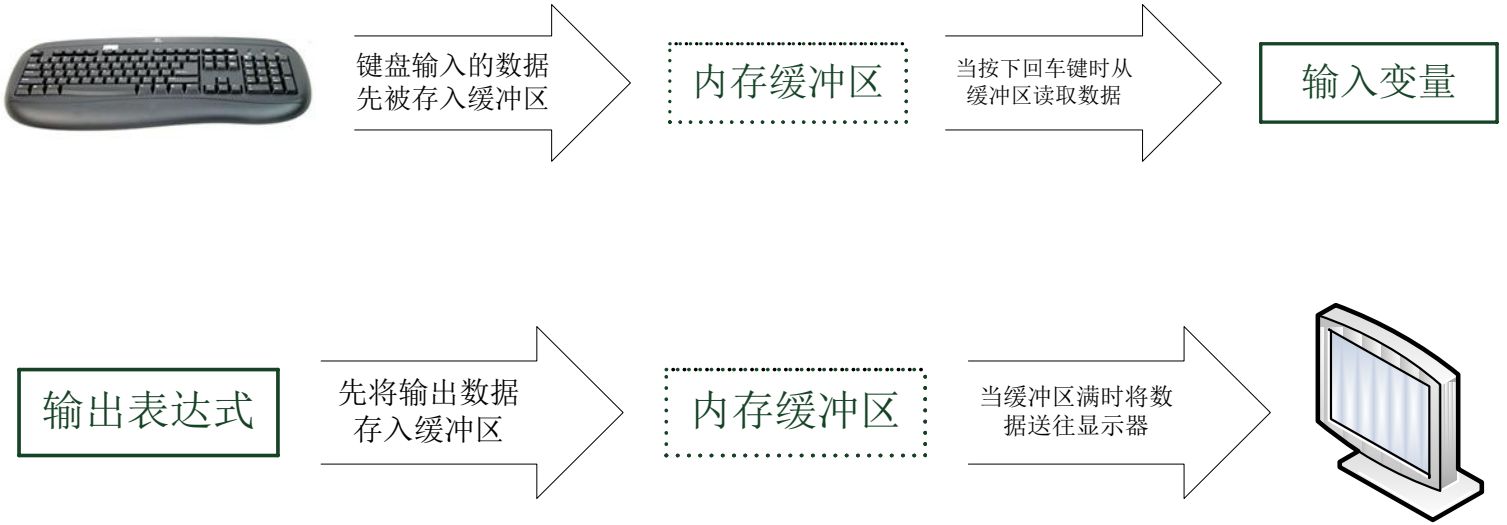
- 本章内容
  - 9.1 流类库
  - 9.2 标准I/O
  - 9.3 文件I/O
  - 9.4 string类及字符串I/O
  - 9.5 基于Unicode编码的流类库

9.1 流类库

- 格式化输入/输出
  - 二进制数值与字符串之间的转换
    - 1个short型整数：20
    - 计算机内部的二进制格式：00000000 00010100
    - 人类阅读：字符串”20”，其中’2’和’0’是单个字符
  - 多种不同的文本格式
    - “ 20” // 位数为4位，不足部分补空格
    - “20 ” // 位数为4位，左对齐
    - “ +20” // 位数为4位，数字前面加正负号
    - “0x16” // 十六进制

9.1 流类库

- 输入/输出流中的数据缓冲区



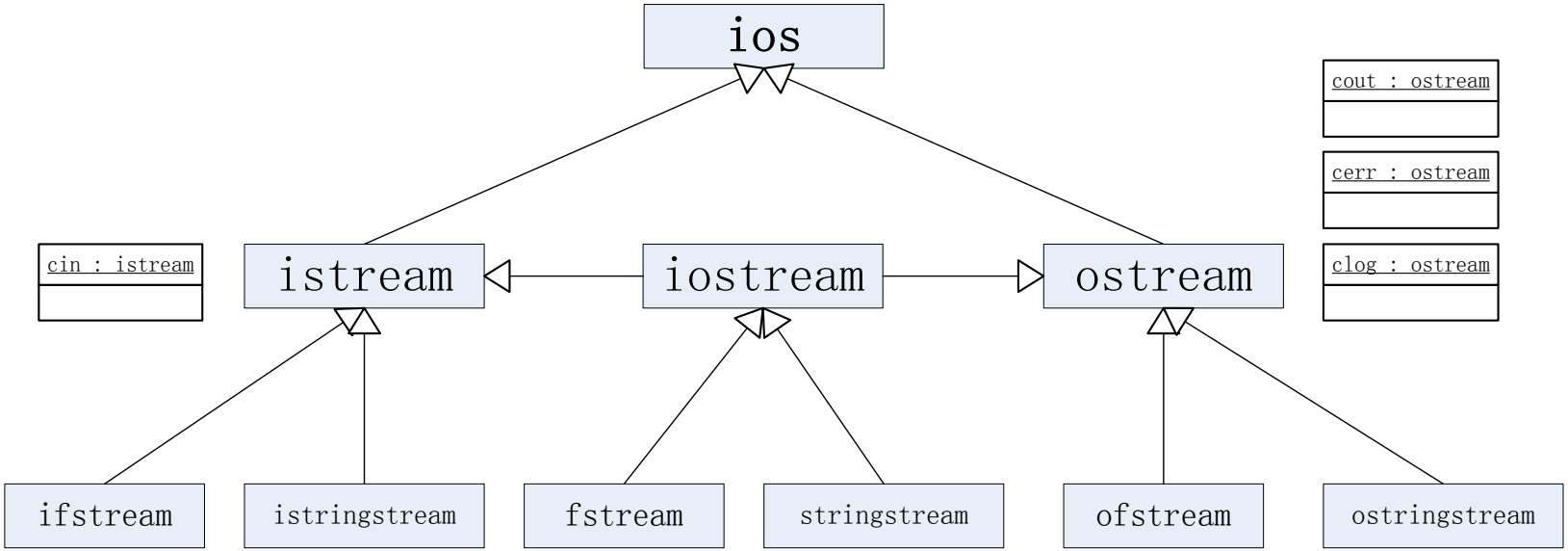
基类ios

例9-1 基类ios的示意代码

```
class ios   // 基类ios的声明部分
{
public:
    enum fmtFlags {   // 枚举类型fmtFlags：定义用作格式标记的枚举常量
      skipws   = 0x0001,   // 输入时跳过空白字符（空格、制表符、回车或换行符）
      left     = 0x0002,   // 输出时左对齐，不足部分在右侧补填充字符
      right    = 0x0004,   // 输出时右对齐，不足部分在左侧补填充字符
      internal  = 0x0008,   // 输出时在正负号或数制后面补填充字符
      dec     = 0x0010,   // 输入/输出时，将整数按十进制进行转换
      oct     = 0x0020,   // 输入/输出时，将整数按八进制进行转换
      hex     = 0x0040,   // 输入/输出时，将整数按十六进制进行转换
      showbase  = 0x0080,   // 输出时包含指示数制的基字符（例如0或0x）
      showpoint = 0x0100,   // 输出浮点数时带小数点和小数0
      uppercase  = 0x0200,   // 输出十六进制数时用大写字母
      showpos  = 0x0400,   // 输出正数时带正号
      scientific  = 0x0800,   // 输出浮点数时采用科学表示法
      fixed    = 0x1000,   // 输出浮点数时不采用科学表示法，即定点格式
    };
    enum open_mode {   // 枚举类型open_mode：定义用作打开模式的枚举常量
      in = 0x01, out = 0x02, ate = 0x04, app = 0x08, trunc = 0x10, binary = 0x80 };
    enum seek_dir {   // 枚举类型seek_dir：定义用作文件指针移动基准的枚举常量
       beg = 0, cur = 1, end = 2 };
    inline long flags(long _l);   // 设置输入/输出时的格式标记，内联函数
    inline long flags() const;   // 返回当前的格式标记，内联函数，常函数成员
    inline int width(int _i);   // 设置下一个数据在输入/输出时的位数，内联函数
    inline int width() const;   // 返回当前的输入/输出位数，内联函数，常函数成员
    inline char fill(char _c);   // 设置输出时的填充字符，内联函数
    inline char fill() const;   // 返回当前的填充字符，内联函数，常函数成员
    inline int precision(int _i);   // 设置浮点数输出时的小数位数，内联函数
    inline int precision() const;   // 返回浮点数输出时的小数位数，内联函数，常函数成员
    inline int good() const;   // 返回输入/输出流状态是否正常，内联函数，常函数成员
    inline int eof() const;   // 返回输入/输出流是否结束，内联函数，常函数成员
    locale imbue(const locale& loc);   // 设置本地语言（例如中文、英文等）
    virtual ~ios();   // 虚析构函数
protected:
    streambuf* bp; // 流缓冲区指针
    ios( );   // 无参构造函数
private:
    ios( const ios& );   // 拷贝构造函数
    ios& operator=( const ios& );   // 重载赋值运算符
    // ..... 以下代码省略
};
```

9.1 流类库

- 流类库的继承关系



9.2 标准I/O

- 输入流对象cin和输出流对象cout

```
namespace  std  // 命名空间std
{
   istream   cin;   // 键盘对象cin
   ostream  cout;  // 显示器对象cout
}

cin >> x;   // 输入数据流：数据从键盘cin流向内存变量x
cout << x;  // 输出数据流：数据从内存变量x流向显示器cout

#include <iostream>
using namespace std;
```

例9-2 通用输入流类istream的示意代码

```
class  istream : virtual public ios  // 公有继承虚基类ios
{
public:
   istream& operator>>(char *);   // 以下代码为重载右移运算符“>>”
   inline istream& operator>>(unsigned char *);
   inline istream& operator>>(signed char *);
   istream& operator>>(char &);
   inline istream& operator>>(unsigned char &);
   inline istream& operator>>(signed char &);
   istream& operator>>(short &);
   istream& operator>>(unsigned short &);
   istream& operator>>(int &);
   istream& operator>>(unsigned int &);
   istream& operator>>(long &);
   istream& operator>>(unsigned long &);
   istream& operator>>(float &);
   istream& operator>>(double &);
   istream& operator>>(long double &);
   int get();   // 以下代码为新增函数成员get
   inline istream& get(unsigned char &);
   inline istream& get(  signed char &);
   inline istream& get(unsigned char *,int,char ='\n');
   inline istream& get(  signed char *,int,char ='\n');
   inline istream& getline(unsigned char *,int,char ='\n');   // 以下代码为新增函数成员getline
   inline istream& getline(  signed char *,int,char ='\n');
   inline istream& read(unsigned char *,int);   // 以下代码为新增函数成员read
   inline istream& read(signed char *,int);
   int gcount() const;   // 新增函数成员gcount
   istream& seekg(long);   // 以下代码为新增函数成员seekg
   istream& seekg(long,ios::seek_dir);
   long tellg();   // 新增函数成员tellg
   // ..... 以下代码省略
};
```

9.2 标准I/O

- 通用输入流类istream及其对象cin
  - 使用提取运算符“>>”

```
#include <iostream>
using namespace std;
int main( )
{
   int  x = 0;    double  y = 0;
   cin >> x >> y;
   cout << “x=” << x << “, y=” << y << endl;
   return 0;
}
```

19 15.234<回车键>
x=19, y=15.234

15.234 19<回车键>
x=15, y=0.234

abcd<回车键>
x=0, y=0

9.2 标准I/O

- 通用输入流类istream及其对象cin

```
#include <iostream>
using namespace std;
int main( )
{
   char str1[5], str2[8];
   cin >> str1 >> str2;
   cout << "str1=" << str1 << ", str2=" << str2 << endl;
   return 0;
}
```

abcdef 123456789<回车键>

```
cin.width( 5 ); // 设置下一输入项的最大字符个数为5（含字符串结束符）
cin >> str1; // 使用键盘对象cin的提取运算符输入str1，最多提取4个字符
cin.width( 8 ); // 设置下一个输入项的最大字符个数为8（含字符串结束符）
cin >> str2;
```

abcdef 123456789<回车键>
str1=abcd, str2=ef

9.2 标准I/O

- 通用输入流类istream及其对象cin
  - 使用函数成员get

```
#include <iostream>
using namespace std;
int main( )
{
    char ch;
    while (true)
    {
       ch = cin.get( );  // 每次从流缓冲区中读出1个字符
       if (ch == '\n')  break;  // 遇到回车键字符’\n’结束，跳出循环
       if (ch >= 'a' && ch <= 'z')  ch -= 32;  // 如果是小写字母，则转成大写字母
       if (ch == ‘ ‘)  ch = ‘*’;  // 如果是空格，则转成星号’*’
       cout << ch;  // 输出该字符
    }
     cout << endl;
     return 0;
}
```

abc 1230<回车键>
则cout语句的显示结果为：**ABC*1230**

例9-3 通用输出流类ostream的示意代码

```
class  ostream : virtual public ios  // 公有继承虚基类ios
{
public:
      // 以下代码为重载左移运算符“<<”
      inline  ostream& operator<<(ostream& (__cdecl * _f)(ostream&));
      inline  ostream& operator<<(ios& (__cdecl * _f)(ios&));
       ostream& operator<<(const char *);
      inline  ostream& operator<<(const unsigned char *);
      inline  ostream& operator<<(const signed char *);
      inline  ostream& operator<<(char);
       ostream& operator<<(unsigned char);
      inline  ostream& operator<<(signed char);
       ostream& operator<<(short);
       ostream& operator<<(unsigned short);
      ostream& operator<<(int);
       ostream& operator<<(unsigned int);
       ostream& operator<<(long);
       ostream& operator<<(unsigned long);
      inline  ostream& operator<<(float);
       ostream& operator<<(double);
       ostream& operator<<(long double);
       ostream& operator<<(const void *);
       ostream& operator<<(streambuf*);
      inline  ostream& put(char);   // 以下代码为新增函数成员put
       ostream& put(unsigned char);
      inline  ostream& put(signed char);
       ostream& write(const char *,int);   // 以下代码为新增函数成员write
      inline  ostream& write(const unsigned char *,int);
      inline  ostream& write(const signed char *,int);
       ostream& seekp(long);   // 以下代码为新增函数成员seekp
       ostream& seekp(long,ios::seek_dir);
       long tellp();   // 新增函数成员tellp
       ostream& flush();   // 新增函数成员flush：立即输出，然后清空缓冲区
      // ..... 以下代码省略
};
```

9.2 标准I/O

- 通用输出流类ostream及其对象cout
  - 使用插入运算符“<<”
  - 格式标记（Flag）是基类ios中定义的一组枚举常量，用于表示不同的输出格式（参见例9-1中代码第4~18行）。设置输出格式，某些格式需使用函数成员flags，而另外一些格式需通过专门的函数成员。例如：

```
cout.flags( ios::hex ); // 使用函数flags设置格式：以十六进制输出整数
cout.width( 5 ); // 使用专门的函数设置格式：将下一输出项的输出位数设定为5位
cout << 20; // 显示结果：␣␣␣14，其中“␣”表示空格
```

  - 格式操纵符（Manipulator）是流类库中定义的一组函数。这些函数被分散定义在不同的头文件中，其示意代码如下：

```
// 头文件：<iomanip>
inline long  setiosflags(long _l); // 设置某个格式标记，例如：setiosflags( ios::hex )
inline long  resetiosflags(long _l); // 将某个格式标记恢复到其默认值
inline int   setw(int _w); // 设置输出位数，不足部分补空格。仅对下一个输出项有效
inline int   setfill(int _m); // 设置填充字符。输出位数不足部分补填充字符
inline int   setprecision(int _p); // 设置浮点数的输出精度（即小数位数）

cout << hex << setw(5) << 20;  // 显示结果：␣␣␣14，其中“␣”表示空格
```

9.2 标准I/O

- 设定输出位数及填充字符

(a) 插入运算符 + **格式标记**

```
1  #include <iostream>
2  using namespace std;
3
4  int main( )
5  {
6        char *name[ ] = { “手电筒”, “电池” };
7        double price[ ] = { 75.825, 4.1 };
8        cout << “商品名称 单价” << endl;
9        cout.fill( ‘#’ );  // 位数不足部分补’#’
10       for (int n=0; n < 2; n++)
11       {
12          cout.width( 8 );  cout << name[n];
13          cout << “ “;
14          cout.width( 6 );  cout << price[n];
15          cout << endl;
16       }
17       return 0;
18 }
```

商品名称 单价
##手电筒 75.825
####电池 ###4.1

(b) 插入运算符 + **格式操纵符**

```
#include <iostream>
#include <iomanip>
using namespace std;

int main( )
{
      char *name[ ] = { “手电筒”, “电池” };
      double price[ ] = { 75.825, 4.1 };
      cout << “商品名称 单价” << endl << setfill( ‘#’ );
      for (int n=0; n < 2; n++)
      {
         cout << setw( 8 ) << name[n];
         cout << “ “;
         cout << setw( 6 ) << price[n];
         cout << endl;
      }
      return 0
}
```

商品名称 单价
手电筒 75.825
电池 4.1

9.2 标准I/O

- 设定对齐方式

(a) 插入运算符 + **格式标记**

```
1  #include <iostream>
2  using namespace std;
3
4  int main( )
5  {
6        char *name[ ] = { “手电筒”, “电池” };
7        double price[ ] = { 75.825, 4.1 };
8        cout << “商品名称 单价\n”;
9        cout.flags( ios:: left );  // 左对齐
10       for (int n=0; n < 2; n++)
11       {
12          cout.width( 8 );  cout << name[n];
13          cout << “ “;
14          cout.width( 6 );  cout << price[n];
15          cout << endl;
16       }
17       return 0;
18 }
```

商品名称 单价
手电筒 75.825
电池 4.1

(b) 插入运算符 + **格式操纵符**

```
#include <iostream>
#include <iomanip>
using namespace std;

int main( )
{
     char *name[ ] = { “手电筒”, “电池” };
     double price[ ] = { 75.825, 4.1 };
     cout << “商品名称 单价\n” << setiosflags(ios::left);
     for (int n=0; n < 2; n++)
     {
        cout << setw( 8 ) << name[n];
        cout << “ “;
        cout << setw( 6 ) << price[n];
        cout << endl;
     }
     return 0;
}
```

商品名称 单价
手电筒 75.825
电池 4.1

9.2 标准I/O

- 设定浮点数的输出格式

(a) 插入运算符 + **格式标记**

```
#include <iostream>
using namespace std;

int main( )
{
    double x = 12.345678;
    cout << x << endl; // 默认：12.3457

    cout.flags( ios::fixed ); // 定点表示法
    cout.precision( 2 ); // 保留2位小数
    cout << x << endl; // 显示结果：12.35

    cout.flags(ios::scientific); //科学表示法
    cout.precision( 8 ); // 保留8位小数
    cout << x << endl;
    // 显示结果：1.23456780e+001
    return 0;
}
```

(b) 插入运算符 + **格式操纵符**

```
#include <iostream>
#include <iomanip>
using namespace std;

int main( )
{
    double x = 12.345678;
    cout << x << endl; // 默认：12.3457
    cout << setiosflags( ios::fixed ); // 定点表示法
    cout << setprecision( 2 ); // 保留2位小数
    cout << x << endl; // 显示结果：12.35

    cout << resetiosflags(ios::fixed); //取消定点格式
    cout << setiosflags(ios::scientific); //科学表示法
    cout << setprecision( 8 ); // 保留8位小数
    cout << x << endl; //显示结果：1.23456780e+001
    return 0;
}
```

9.2 标准I/O

- 函数成员put

```
#include <iostream>
using namespace std;
int main( )
{
   char ch, str[ ] = “abcd”;
   int n = 0;
   while (str[n] != ‘\0’)  // 循环输出字符数组str中的字符
   {
      ch = str[n];  // 从数组str中取1个字符（小写字母）
      cout << ch;  // 用插入运算符输出字符ch
      cout.put( ch-32 );  // 将字符ch转成大写字母，再用函数成员put输出
      n++;
   }
   cout << endl;
   return 0;
}
```

执行上述程序，其显示结果为：**aAbBcCdD**

9.3 文件I/O

- 程序所处理的数据
  - 标准I/O：键盘输入、显示器输出
  - 文件I/O：文件输入、文件输出

磁盘文件“temperature.txt”

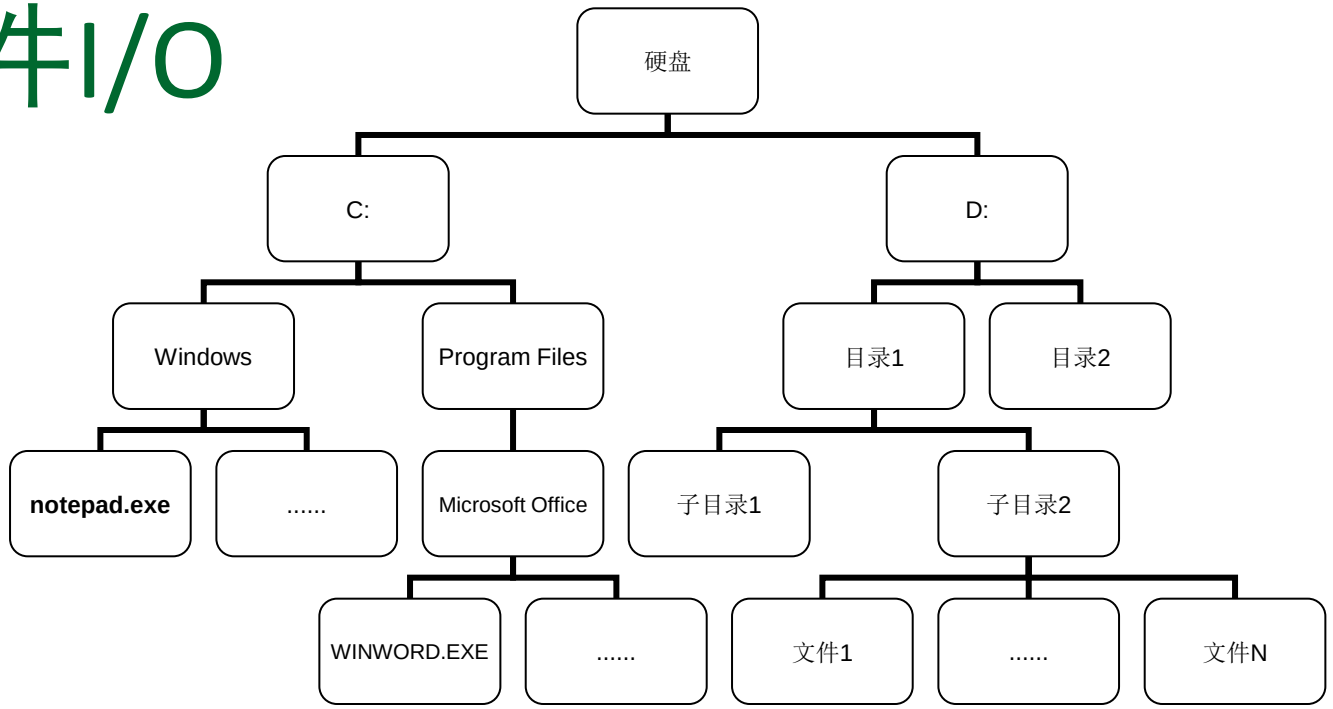
日期 气温
1 30.2
2 28.7
.....
30 25.9

9.3 文件I/O

- 程序文件与数据文件
  - 记事本：notepad.exe，*.txt
  - Word：WINWORD.exe，*.docx
  - 文件I/O：介绍如何从数据文件中输入数据，如何向数据文件中输出数据

- 文件
  - 计算机通过文件（File）来管理存储在外存（硬盘、光盘、U盘等）上的信息
  - 当文件数量很多时，可以为文件建立分类目录（文件夹），将文件分散在不同目录下进行管理
  - 目录下可以再建立子目录
  - 同一子目录下的文件不能重名

9.3 文件I/O



- 一个完整的文件名格式

  盘符:\目录名\子目录名\.....\文件名.扩展名

  C:\Windows\notepad.exe

  “C:\\Windows\\notepad.exe”

9.3 文件I/O

- 文件格式：文本文件、二进制文件

- 文本文件
  - **存储字符编码**。文本文件所存储的内容是一个字符序列。每个字符所存储的是其字符编码，例如英文字母存储的是其ASCII编码（1个字节），中文字符存储的是其机内码（2个字节）
  - **具有换行格式**。文本换行时，存储2个控制字符CR（ASCII编码为13）和LF（ASCII编码为10）
  - **通用性强**。文本文件存储的是纯文本内容，而且使用的是标准编码。文本文件不含任何其它附加信息（例如字体、排版格式等）。阅读文本文件不需要安装特殊的软件，使用类似于“记事本”这样的常规软件就能阅读。文本文件的阅读、修改等不依赖于某个特定的软件。换句话说，文本文件的通用性强
  - **可用于数据交换**。文本文件通用性强，可用于数据交换。例如，一个程序的处理结果可以通过文本文件输出给人来阅读，这是程序-人之间的数据交换；一个程序的处理结果可以通过文本文件输入给另一个程序，这是程序-程序之间的数据交换；一个操作系统中程序的处理结果可以通过文本文件传送给另一个操作系统中的程序，这是操作系统-操作系统之间的数据交换

9.3 文件I/O

- 二进制文件
  - **可保存任意类型的数据**。文本文件只存储字符类型数据，任何其它类型的数据必须转换成字符串才能保存到文本文件中。而二进制文件可保存任意类型的数据
  - **存储效率高**。和文本文件相比，将内存变量中数据保存成二进制文件的效率更高。效率高具体表现在两个方面：一是无需要格式转换，保存速度快；二是保存数值型数据所占用的存储空间少。例如，保存1个short型整数-2100，使用二进制文件需要2个字节。使用文本文件，需将short型整数-2100转换成字符串”-2100”，保存该字符串需要5个字节
  - **通用性差**。二进制文件是程序员自己定义的一种私有格式。不同程序会创建不同的二进制文件。不管什么类型的数据，保存到二进制文件后都变成了二进制的0、1序列。在不了解存储格式的情况下，任何程序都无法正确解释由其它程序创建的二进制文件。二进制文件天生就是一种加密的文件。和文本文件相比，二进制文件的通用性差。对于二进制数据文件，通常是由哪个程序创建，就由哪个程序负责阅读、修改
  - **交换数据需遵循相同的格式标准**。为了能在不同程序间通过二进制文件交换数据，人们需要为二进制文件制定共同的格式标准。例如为了交换图像数据，人们专门制定了一些二进制图像文件的格式标准，常用的有JPEG、BMP、GIF、TIFF等

9.3 文件I/O

- 文件的基本操作
  - **打开文件**

    程序在对文件进行输入/输出操作之前，首先需要打开文件。打开（Open）文件时需指定文件名和打开模式。打开模式指的是程序将对文件进行何种操作，例如读数据（输入）、写数据（输出），或是两者都有（输入+输出）等

  - **读/写数据**

    文件打开以后，程序可以从文件中读数据（输入）、或向文件中写数据（输出），或是两者都有（输入+输出）。向文件读/写数据可以按存储顺序依次读/写，这称为顺序读/写。也可以指定从某个位置开始读/写，这称为随机读/写

  - **关闭文件**

    程序在完成对文件的输入/输出操作之后，需要关闭文件。通常情况下，数据文件只能被一个程序打开，这样可以避免读/写冲突。程序在关闭数据文件之后，该文件才可以再被其它程序打开

9.3 文件I/O

- C++流类库中定义了3个不同的文件输入/输出流类
  - 文件输出流类ofstream
  - 文件输入流类ifstream
  - 以及文件输入/输出流类fstream

- 定义文件输入/输出流类的对象，通过对象调用其函数成员就可以实现文件的输入/输出功能。文件输入/输出流类的对象被称为是**文件对象**

- 使用文件输入/输出流类需包含类声明头文件：

  #include <fstream>

9.3 文件I/O

- 文件输入/输出流类
  - 文件输出流类ofstream及文件输出

例9-4 文件输出流类ofstream的示意代码

```
1  class  ofstream : public ostream  // 公有继承通用输出类ostream
2  {
3  public:
4      ofstream();   // 无参构造函数
5      ofstream(const char *, int =ios::out);   // 有参构造函数
6      ~ofstream();   // 析构函数
7
8      void open(const char *, int =ios::out);   // 打开文件
9      bool is_open() const;   // 检查文件是否正确打开
10     void close();   // 关闭文件
11     // ..... 以下代码省略
12 };
```

```
// 输出文本文件

#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;
int main( )
{
   char *name[ ] = { “手电筒”, “电池” };
   double price[ ] = { 75.825, 4.1 };
   int n;
   // 使用显示器对象cout将数据输出到显示器
   cout << “商品名称 单价\n” << setiosflags(ios::left);
   for (n = 0; n < 2; n++)
   {
      cout << setw( 8 ) << name[n];   cout << “ “;
      cout << setw( 6 ) << price[n];   cout << endl;
   }
   // 使用文件输出流类ofstream的文件对象fout将数据输出到文本文件“price.txt”
   ofstream  fout;  // 文件输出流类ofstream对象fout需要程序员自己定义
   fout.open( “price.txt” );  // 打开文件“price.txt”， 如文件不存在则创建新文件
   fout << “商品名称 单价\n” << setiosflags(ios::left);  // 标题行
   for (n = 0; n < 2; n++)
   {
      fout << setw( 8 ) << name[n];   fout << “ “;
      fout << setw( 6 ) << price[n];   fout << endl;
   }
   fout.close( );  // 关闭所打开的文件“price.txt”
   return 0;
}
```

```
商品名称 单价
手电筒   75.825
电池     4.1
```

```
// 输出二进制文件
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string.h>
using namespace std;
int main( )
{
   char *name[ ] = { “手电筒”, “电池” };
   double price[ ] = { 75.825, 4.1 };
   int n;
   // 使用显示器对象cout将数据输出到显示器
   cout << “商品名称 单价\n” << setiosflags(ios::left);
   for (n = 0; n < 2; n++)
   {
      cout << setw( 8 ) << name[n];    cout << “ “;
      cout << setw( 6 ) << price[n];    cout << endl;
   }
   // 使用文件对象fout将数据输出到二进制文件“price.dat”
   ofstream  fout;  // 定义1个文件输出流类ofstream的文件对象fout
   fout.open( “price.dat”, ios::binary );  // 以二进制模式打开文件“price.dat”
   char str[7];
   for (n = 0; n < 2; n++)
   {
      strcpy( str, name[n] );    fout.write( str, sizeof(str) );  // 输出商品名称
      fout.write( (char *)&price[n], sizeof(double) );    // 输出价格
   }
   fout.close( );  // 关闭所打开的文件“price.dat”
   return 0;
}
```

```
商品名称 单价
手电筒   75.825
电池     4.1
```

```
手电筒 吞烫挑R@电池？ffffff@
```

9.3 文件I/O

- 文件输入流类ifstream及文件输入

例9-5 文件输入流类ifstream的示意代码

```
1  class  ifstream : public istream  // 公有继承通用输入类istream
2  {
3  public:
4      ifstream();   // 无参构造函数
5      ifstream(const char *, int =ios::in);   // 有参构造函数
6      ~ifstream();   // 析构函数
7
8      void open(const char *, int =ios::in);   // 打开文件
9      bool is_open() const;   // 检查文件是否正确打开
10     void close();   // 关闭文件
11     // ..... 以下代码省略
12 };
```

```
// 输入文本文件

#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
   char  name[20];
   double price;
   // 使用文件输入流类ifstream的文件对象fin从文本文件“price.txt”中输入数据
   ifstream  fin;  // 文件输入流类ifstream对象fin需要程序员自己定义
   fin.open( “price.txt” );  // 打开文件“price.txt”
   fin.getline( name, 19 );  // 读出标题行
   cout << name << endl;  // 显示所读出的标题行，显示结果：商品名称 单价
   for (int n = 0; n < 2; n++)
   {
      fin >> name >> price;  // 从文件“price.txt”中读取商品名称和单价
      cout << name << “, “ << price << endl;  // 显示商品名称和单价，验证输入结果
   }
   fin.close( );  // 关闭所打开的文件“price.txt”
   return 0;
}
```

```
商品名称 单价
手电筒, 75.825
电池, 4.1
```

```
// 输入二进制文件

#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
   char  name[20];
   double price;
   // 使用文件对象fin从二进制文件“price.dat”中输入数据
   ifstream  fin;  // 定义1个文件输入流类ifstream的文件对象fin
   fin.open( “price.dat”, ios::binary );  // 以二进制模式打开文件“price.dat”
   for (int n = 0; n < 2; n++)
   {
      fin.read( name, 7 );  // 输入商品名称
      fin.read( (char *)&price, 8 );  // 输入单价
      cout << name << “, “ << price << endl;  // 显示商品名称和单价
   }
   fin.close( );  // 关闭所打开的文件“price.dat”
   return 0;
}
```

手电筒, 75.825
电池, 4.1

9.3 文件I/O

- 检查输入文件状态
  - eof( )。检查文件是否已结束
  - good( )。检查文件是否已损坏

```
#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
   char  ch;
   ifstream  fin( “price.txt” );  // 定义并初始化输入流类ifstream对象fin
   while (true )
   {
      fin.get( ch );  // 从文件“price.txt”中读取1个字符
      if ( fin.eof( ) == true | |  // eof的返回值：true-文件已结束，false-文件未结束
          fin.good( ) == false )  // good的返回值：true-文件正常，false-文件已损坏
         break;  // 结束文件输入
      cout.put( ch );  // 显示所读出的字符ch
   }
   fin.close( );  // 关闭所打开的文件“price.txt”
   return 0;
}
```

9.3 文件I/O

- 文件输入/输出流类fstream

例9-6 文件输入/输出流类fstream的示意代码

```
1  class  iostream : public istream, public ostream  // 多继承类istream和ostream
2  {
3  public:
4      virtual ~iostream();   // 虚析构函数
5  protected:
6      iostream();   // 无参构造函数
7      iostream(const iostream&);   // 有参构造函数
8      // ..... 以下代码省略
9  };
10
11 class  fstream : public iostream  // 公有继承通用输入/输出类iostream
12 {
13 public:
14     fstream();   // 无参构造函数
15     fstream(const char *, int);   // 有参构造函数
16     ~fstream();   // 析构函数
17
18     void open(const char *, int);   // 打开文件
19     bool is_open() const;   // 检查文件是否正确打开
20     void close();   // 关闭文件
21     // ..... 以下代码省略
22 };
```

9.3 文件I/O

- 文件输入/输出流类fstream
  - **从类istream继承的基类成员**。其中包括提取运算符“>>”，函数成员get、getline、read、seekg和tellg等。这些成员是从类iostream间接继承来的，它们提供了文件输入功能
  - **从类ostream继承的基类成员**。其中包括插入运算符“<<”，函数成员put、write、seekp、tellp和flush等。这些成员也从类iostream间接继承而来，它们提供了文件输出功能
  - **新增函数成员**。其中包括打开文件函数open和关闭文件函数close等
- 打开文件函数open

  void open(const char * filename, int open_mode);

9.3 文件I/O

枚举常量	所表示的打开模式
ios :: in	打开一个输入文件。若文件不存在，则打开失败
ios :: out	打开一个输出文件。若文件不存在则创建文件，若文件存在则清空其内容
ios :: binary	以二进制模式打开文件。后续输入/输出操作应使用read/write函数
ios :: app	以追加（Append）模式打开输出文件。后续输出数据将追加在文件的末尾
ios :: ate	打开文件，并将文件指针移到文件末尾
ios :: trunc	打开文件，并清空其内容。若未指定ios::in、ios::app、ios::ate，则默认此模式

fstream fobj; // 定义1个fstream类的文件对象fobj

- 以不同模式打开文件“aaa.dat”的举例

  fobj.open(“aaa.dat”, ios::in); // 以输入/文本模式打开文件“aaa.dat”，默认为文本模式

  fobj.open(“aaa.dat”, ios::in | ios::binary); // 以输入/二进制模式打开文件“aaa.dat”

  fobj.open(“aaa.dat”, ios::out | ios::binary); // 以输出/二进制模式打开文件“aaa.dat”

  fobj.open(“aaa.dat”, ios::out | ios::app); // 以输出/追加/文本模式打开文件“aaa.dat”

  fobj.open(“aaa.dat”, ios::out | ios::trunc); // 以输出/清空/文本模式打开文件“aaa.dat”

9.3 文件I/O

- 文件的随机读写
  - 文件输入流对象包含一个**读文件指针**
  - 文件输出流对象包含一个**写文件指针**

```
istream& seekg( long bytes, ios::seek_dir origin );  // 移动读文件指针
long tellg( );  // 返回当前读文件指针的位置

ostream& seekp( long bytes, ios::seek_dir origin );  // 移动写文件指针
long tellp();  // 返回当前写文件指针的位置
```

```
#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
   char  name[20];
   double price;
   // 使用文件对象fin从二进制文件“price.dat”中输入数据
   ifstream  fin;  // 定义1个文件输入流类ifstream的文件对象fin
   fin.open( “price.dat”, ios::binary );  // 以二进制模式打开文件“price.dat”
   for (int n = 0; n < 2; n++)
   {
      fin.read( name, 7 );  // 输入商品名称
      fin.read( (char *)&price, 8 );  // 输入单价
      cout << name << “, “ << price << endl;  // 显示商品名称和单价
   }
   fin.seekg( -15, ios::end );  // 从文件尾向前（即往回）移动1行（1行有15个字节）
   fin.read( name, 7 );    fin.read( (char *)&price, 8 );  // 重读最后1行数据
   cout << name << “, “ << price << endl;  // 显示所读出的商品名称和单价
   fin.close( );  // 关闭所打开的文件“price.dat”
   return 0;
}
```

执行上述程序，显示器将显示如下的价格清单：

手电筒, 75.825
电池, 4.1
电池, 4.1

9.4 string类及字符串I/O

- C语言字符串处理
  - 字符数组
  - 字符串处理函数：strlen、strcpy、strcat等

- C++语言的字符串类string
  - 封装了字符数组和字符串处理函数
  - 定义string类对象保存字符串数据，调用其函数成员来处理字符串
  - string类对象可作为输入/输出数据流，即字符串I/O

9.4 string类及字符串I/O

- 字符串类string
  - 字符串对象的定义与初始化

```
#include <iostream>
#include <string>
using namespace std;
int main( )
{
   string  str;  // 定义字符串对象str，未初始化
   string  str1( “Hello, world” );  // 定义字符串对象str1，初始化方法1
   string  str2 = “Hello, world”;  // 定义字符串对象str2，初始化方法2
   string  str3( str2 );  // 定义字符串对象str3，初始化方法3（拷贝构造）

   char  cArray[ ] = “Hello, world”;
   string  str4( cArray );  // 定义字符串对象str4，初始化方法4
   // ..... 以下代码省略
}
```

9.4 string类及字符串I/O

- 字符串类string
  - 字符串对象的输入与输出

```
string  str;
cin >> str;  // 从键盘为字符串对象str输入内容
cout << str << endl;  // 显示字符串对象str中的内容
```

9.4 string类及字符串I/O

- 字符串类string
  - string类的函数成员

```
string  str( “abcdefg” );
cout << str.length( ) << endl;  // 函数length返回字符串长度，显示结果：7
cout << str.find( "cd" ) << endl;  // 函数find查找子串"cd"的位置，显示结果：2
cout << str.substr( 2, 4 ) << endl;  // 函数substr取出1个子串，显示结果：cdef
str.append( "123" );  // 函数append在串尾追加1个字符串
cout << str << endl;  // 显示结果：abcdefg123
```

9.4 string类及字符串I/O

- 字符串类string
  - string类重载的运算符

重载运算符	举例	运算说明
+	cout << str + “123”;	连接字符串。显示结果：abcd123
=	string str1; str1 = str; cout << str1;	对象赋值。显示结果：abcd
+=	str += “123”; cout << str;	追加字符串。显示结果：abcd123
==	str == “abcd”	关系运算：等于。比较2个字符串是否相等
!=	str != “abcd”	关系运算：不等于
<	str < “abce”	关系运算：小于。依次比较字符的ASCII码值
<=	str <= “abce”	关系运算：小于等于
>	str > “abce”	关系运算：大于
>=	str >= “abce”	关系运算：大于等于
[ ]	cout << str[0];	按下标访问串中的字符。显示结果：a

9.4 string类及字符串I/O

- 字符串I/O
  - 可以把string类对象当作数据源，将其中的字符串输入给其它变量，这时string类对象就是一个输入数据流
  - 也可以把string类对象当作输出目的地，将其它变量中的数据输出到string类对象，这时string类对象就是一个输出数据流
  - 在字符串对象和其它变量之间进行数据的输入/输出，这就是字符串I/O

- C++流类库通过字符串输入/输出流类提供了字符串I/O的功能。字符串输入/输出流类共有3个：
  - 字符串输入流类istringstream
  - 字符串输出流类ostringstream
  - 字符串输入/输出流类stringstream
  - #include <sstream>

9.4 string类及字符串I/O

- 字符串I/O
  - 字符串输入流类istringstream

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
int main( )
{
   string  str ( “10 25.76” );  // 定义字符串对象str
   istringstream  strin( str );  // 定义istringstream类对象strin，用str初始化

   int  x;   double  y;
   strin >> x >> y;  // 从串流对象strin中为变量x、y输入数据

   cout << “x=” << x << “, y=” << y << endl;  // 显示结果：x=10, y=25.76
   return 0;
}
```

9.4 string类及字符串I/O

- 字符串I/O
  - 字符串输出流类ostringstream

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
int main( )
{
   int  x = 10;    double  y = 25.76;
   ostringstream  strout;  // 定义ostringstream类对象strout

   strout << "x=" << x << ", y=" << y << endl;  // 向串流对象strout中输出数据
   string  str = strout.str( );  // 将串流对象strout中的字符串赋值给字符串对象str
   cout << str << endl;  // 显示结果：x=10, y=25.76
   return 0;
}
```

9.5 基于Unicode编码的流类库

- 基于Unicode编码的流类库以**w**ios为基类
  - **标准I/O**。包括通用输入流类**w**istream、通用输出流类**w**ostream、通用输入/输出流类**w**iostream
  - **文件I/O**。包括文件输入流类**w**ifstream、文件输出流类**w**ofstream、文件输入/输出流类**w**fstream
  - **字符串I/O**。包括字符串输入流类**w**istringstream、字符串输出流类**w**ostringstream、字符串输入/输出流类**w**stringstream

- 预定义的宽字符流对象

```
namespace  std  // wcin/wcout/wclog/wcerr都被定义在命名空间std中
{
   wistream   wcin;  // 宽字符的键盘对象wcin
   wostream  wcout; // 宽字符的显示器对象wcout
}
```

9.5 基于Unicode编码的流类库

```
#include <iostream>
#include <iomanip>
#include <string>
#include <locale>
using namespace std;

int main( )
{
   wstring  name[ ] = {  L"手电筒", L"电池"  };  // wstring：宽字符串类
   double price[ ] = {  75.825, 4.1  };

   wcout.imbue( locale("chs") );  // 将语言设置为简体中文chs
   wcout << L"商品名称 单价\n";  // 前缀L表示宽字符串常量
   for (int n=0; n < 2; n++)
      wcout << name[n] << L" " << price[n] << endl;
   return 0;
}
```

商品名称 单价
手电筒 75.825
电池 4.1

本章要点

- 应理解之前所用的cin、cout指令实际上分别是通用输入/输出流类的对象
- 通过本章学习，可以从侧面了解全球顶尖的C++程序员是如何来设计和编写类的，这样可以进一步深入体会前面所学习的各种面向对象程序设计知识
- 重点学习如何进行文件读写操作，大部分程序都需要使用文件来保存数据