C++语言程序设计
第七章 输入输出流

王焦乐

http://faculty.hitsz.edu.cn/jlwang

哈尔滨工业大学（深圳）

机电工程与自动化学院

C++语言程序设计...

群号：599477959

扫一扫二维码，加入群聊。

QQ

本章主要内容

- C++的输入和输出
- 标准输出流
- 标准输入流
- 对数据文件的操作与文件流
- 字符串流

0、C++的输入和输出

- 输入输出的含义

（1）标准设备的输入和输出（**标准I/O**）

标准输入：键盘输入、标准输出：显示器输出

（2）以外存（硬盘、光盘）为对象进行输入和输出（**文件I/O**）

文件输入输出

（3）对内存中制定的空间进行输入和输出（**串I/O**）

字符串输入输出：指定一个字符数组作为储存空间

0、C++的输入和输出

- C++的I/O对C的发展——类型安全和可扩展性

```
printf("%d\n",i); //i为整型变量，正确，输出i的值
printf("%d\n",f); //浮点型变量f在内存中的信息按整数解释并输出
printf("%d\n","C++"); //输出字符串“C++”的起始地址

scanf("%d",&i); //输入一个整数，赋给整型变量i
scanf(“%d”,i); //漏写&，可能产生严重问题
```

- C++为了兼容C保留了printf和scanf，但推荐使用C++的方法。
- C++输入输出进行严格的类型检查，凡是类型不正确的都不可能通过编译，因此是类型安全（type safe）的
- 通过重载方式使得标准类型和用户声明类型都采用相同方法处理，具有可扩展性。

0、C++的输入和输出

- C++的输入输出流 （input output stream）

（1）是指若干字节组成的字节序列按顺序从一个对象传送到另一个对象

（2）缓冲区中（输出缓冲区、输入缓冲区）的数据就是流

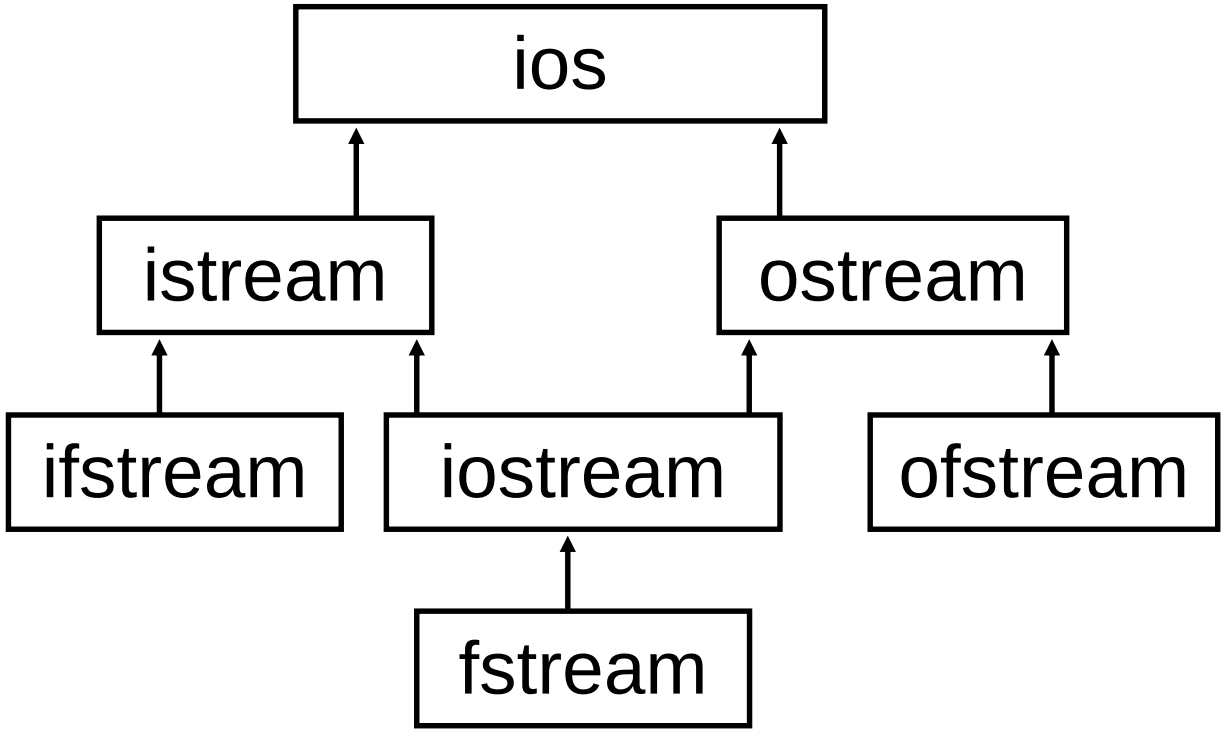
（3）输入输出流被定义为类，C++的I/O库（流类库、流库 ）中的类称为流类（stream class）

（4）对应的实例（instance）就是流对象。如：cout、cin就是iostream类的对象

0、C++的输入和输出

- C++的输入输出流 （input output stream）

（5）C++的流库有两个基类：ios类、streambuf类



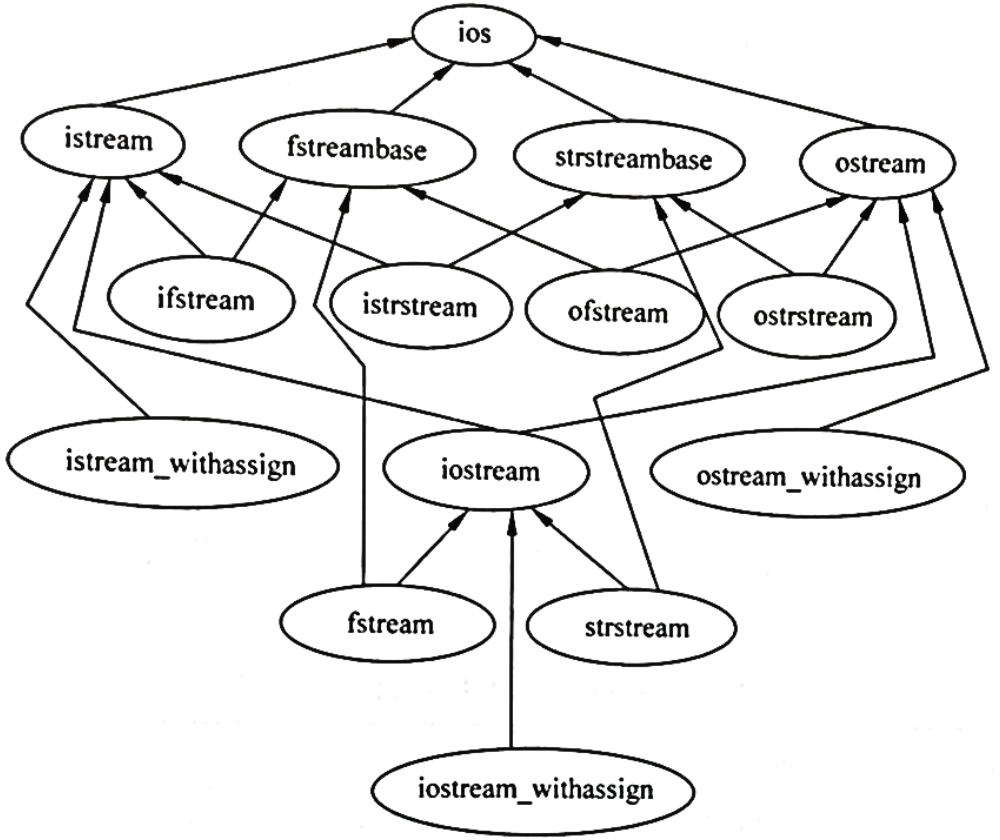


图 7.3

0、C++的输入和输出

- C++的输入输出流 （input output stream）

（6）C++的流库有关的头文件

- iostream　　输入输出流操作
- fstream　　管理文件的I/O操作
- strstream　　字符流的I/O操作
- stdiostream　混合使用C和C++
- iomanip　　使用格式化I/O操作

0、C++的输入和输出

□ C++的输入输出流 （input output stream）

（7） iostream头文件中定义的流对象

对象	含义	设备	C语言中对应的标准文件
cin	标准输入设备	键盘	stdin
cout	标准输出设备	屏幕	stdout
cerr	标准错误输出设备（非缓冲方式）	屏幕	stderr
clog	标准错误输出设备（缓冲方式）	屏幕	stderr

1、标准输出流

□ cout，cerr和clog流

（1）cout流在内存中对应开辟了一个缓冲区

（2）向cout流插入一个endl时，立即输出流中的所有数据，然后插入一个换行符并清空缓冲区（\n 不会刷新缓冲区）

（3）cerr标准出错信息流

（4）clog标准出错流

- cerr不经过缓冲区直接向显示器输出出错信息
- clog把出错信息存放在缓冲区，当缓冲区满或遇到endl时向显示器输出出错信息

编写程序，从键盘输入a，b，c的值求解一元二次方程。如果a=0或判别式的值<0，输出出错信息。

```
int main()
{  float  a, b, c, deter;
 cout<<"Please input a,b,c:";
 cin>>a>>b>>c;
 if (a==0)
  cerr<<"Error! a equal to 0"<<endl;
 else
  if ((deter=b*b-4*a*c)<0)
   cerr<<“Error! Determinant b*b-4*a*c<0"<<endl;
  else
   {cout<<"x1="<<(-b+sqrt(deter))/(2*a)<<endl;
    cout<<"x2="<<(-b-sqrt(deter))/(2*a)<<endl;
   }
 return 0;
}
```

1、标准输出流

□ 标准类型数据的格式输出

（1）使用控制符控制输出格式

表 7.3　输入输出流的控制符

控 制 符	作　　用
dec	设置整数的基数为 10
hex	设置整数的基数为 16
oct	设置整数的基数为 8
setbase(n)	设置整数的基数为 n(n 只能是 8,10,16 三者之一)
setfill(c)	设置填充字符 c,c 可以是字符常量或字符变量
setprecision(n)	设置实数的精度为 n 位。在以一般十进制小数形式输出时 n 代表有效数字。在以 fixed(固定小数位数)形式和 scientific(指数)形式输出时 n 为小数位数
setw(n)	设置字段宽度为 n 位
setiosflags(ios :: fixed)	设置浮点数以固定的小数位数显示
setiosflags(ios :: scientific)	设置浮点数以科学记数法(即指数形式)显示
setiosflags(ios :: left)	输出数据左对齐
setiosflags(ios :: right)	输出数据右对齐
setiosflags(ios :: skipws)	忽略前导的空格
setiosflags(ios :: uppercase)	在以科学记数法输出 E 和以十六进制输出字母 X 时以大写表示
setiosflags(ios :: showpos)	输出正数时给出“+”号
resetioflags( )	终止已设置的输出格式状态,在括号中应指定内容

```
#include <iostream>
#include <iomanip>
```

```
int a;
cout << "input a:";
cin >> a;
cout << "dec:" << dec << a << endl;
cout << "hex:" << hex << a << endl;
cout << "oct:" << setbase(8) << a << endl;
char *pt = "China";
cout << setw(10) << pt << endl;
cout << setfill('*') << setw(10) << pt
     << endl;
double pi = 22.0 / 7.0;
cout << setiosflags(ios::scientific)
     << setprecision(8);
cout << "pi=" << pi << endl;
cout << "pi=" << setprecision(4) << pi
     << endl;
cout<<"pi="<<resetiosflags(ios::scientific)
    << setiosflags(ios::fixed) <<pi<< endl;
```

1、标准输出流

□ 标准类型数据的格式输出

（2）用流对象成员函数控制输出格式

表 7.4　用于控制输出格式的流成员函数

流成员函数	与之作用相同的控制符	作　　用
precision(n)	setprecision(n)	设置实数的精度为 n 位
width(n)	setw(n)	设置字段宽度为 n 位
fill(c)	setfill(c)	设置填充字符 c
setf( )	setiosflags( )	设置输出格式状态，括号中应给出格式状态，内容与控制符 setiosflags 括号中的内容相同，如表 7.5 所示
unsetf( )	resetioflags( )	终止已设置的输出格式状态，在括号中应指定内容

1、标准输出流

□ 标准类型数据的格式输出

（2）用流对象成员函数控制输出格式

表 7.5　设置格式状态的格式标志

格式标志	作　　用
ios :: left	输出数据在本域宽范围内向左对齐
ios :: right	输出数据在本域宽范围内向右对齐
ios :: internal	数值的符号位在域宽内左对齐，数值右对齐，中间由填充字符填充
ios :: dec	设置整数的基数为 10
ios :: oct	设置整数的基数为 8
ios :: hex	设置整数的基数为 16
ios :: showbase	强制输出整数的基数（八进制数以 0 打头，十六进制数以 0x 打头）
ios :: showpoint	强制输出浮点数的小点和尾数 0
ios :: uppercase	在以科学记数法格式 E 和以十六进制输出字母时以大写表示
ios :: showpos	对正数显示“ + ”号
ios :: scientific	浮点数以科学记数法格式输出
ios :: fixed	浮点数以定点格式（小数形式）输出
ios :: unitbuf	每次输出之后刷新所有的流
ios :: stdio	每次输出之后清除 stdout，stderr

```
cout.setf(ios::showbase);
cout << "dec:" << a << endl;
cout.unsetf(ios::dec);
cout.setf(ios::hex);
cout << "hex:" << a << endl;
cout.unsetf(ios::hex);
cout.setf(ios::oct);
cout << "oct:" << a << endl;
char *pt = "China";
cout.width(10);
cout << pt << endl;
cout.width(10);
cout.fill('*');
cout << pt << endl;
double pi = 22.0 / 7.0;
cout.setf(ios::scientific);
cout << "pi=";
cout.width(14);
cout << pi << endl;
cout.unsetf(ios::scientific);
cout.setf(ios::fixed);
cout.width(12);
cout.setf(ios::showpos);
cout.setf(ios::internal);
cout.precision(6);
cout << pi << endl;
```

1、标准输出流

- 用流成员函数put输出字符
- 格式： cout.put(字符/数字)[.put(...) .....]
- 如是字符，直接输出该字符；如是数字，可以用八进制、十进制或十六进制表示整数，用该数字对256取模，输出对应的ASCII码字符。

```
cout.put(256+71).put(79).put(79).put(68).put('\n');
```

```
cout.put(71);  //G
cout.put(79);  //O
cout.put(79);  //O
cout.put(68);  //D
cout.put('\n');
```

```
#include <iostream>
using namespace std;
int main()
{
  char *a = "BASIC";
  for (int i = 4; i >= 0; i--)
    cout.put(*(a + i));
  cout.put('\n');
  return 0;
}
```

```
#include <stdio.h>
int main()
{
  char *a = "BASIC";
  for (int i = 4; i >= 0; i--)
    putchar(*(a + i));
  putchar('\n');
  return 0;
}
```

2、标准输入流

□ cin流

1. istream类的**对象**，从标准输入设备读取数据
2. 流提取运算符>>在流中提取数据时通常跳过流中的空格、tab键、换行符等字符。只有**输入回车键时输入的数据才进入键盘缓冲区**
3. 当遇到**无效字符（与变量数据类型不一致）**或**文件结束符**时，输入流cin就处于出错状态，此时对cin流的所有操作都被终止。
4. 当输入流出错时，**cin的值是false**，所以可以根据cin的值判断流对象是否处于正常状态。

```
while (cin >> grade)
{
  if (grade >= 85)
    cout << grade << " GOOD!" << endl;
  if (grade < 60)
    cout << grade << " Fail!" << endl;
    else
    cout << grade << " Fine!" << endl;
  cout << "Enter grade: ";
}
```

2、标准输入流

- 用于字符输入的流成员函数

（1）get函数读入一个字符

- 格式1：**cin.get()**
- 函数的类型是字符，函数的功能是从输入流中提取一个字符作为函数值返回。如在流中遇到文件结束符EOF时，返回-1。

```
int main()
{
  char c;
  cout << "enter a sentence:" << endl;
  while ((c = cin.get()) != EOF)
    cout.put(c);
  return 0;
}
```

```
int main()
{
  char c;
  cout << "enter a sentence:" << endl;
  while ((c = getchar()) != EOF)
    putchar(c);
  return 0;
}
```

2、标准输入流

- 用于字符输入的流成员函数

（1）get函数读入一个字符

- 格式**2**：**cin.get(字符变量)**
- 从输入流中提取一个字符赋予字符变量。如遇到文件结束符就结束提取。输入回车后再输入文件结束符。

```
int main()
{
  char c;
  cout << "enter a sentence:" << endl;
  while (cin.get(c))
  {
    cout.put(c);
  }
  cout << "end" << endl;
  return 0;
}
```

2、标准输入流

- 用于字符输入的流成员函数

（1）get函数读入一个字符

- 格式3：**cin.get(字符指针，n，终止字符)**
- n 与提取的字符个数相关。函数从键盘缓冲区最多顺序提取n-1个字符，顺序放入字符指针所指的字符数组。如果在提取过程中遇到终止字符，无论是否满足指定的字符个数都要终止提取。

```
void main()
{
  char ch[20];
  cout << "enter a sentence:" << endl;
  cin.get(ch, 10, '\n');
  cout << ch << endl;
}
```

2、标准输入流

- 用于字符输入的流成员函数

（1）get函数读入一个字符

a. cin.get() 中不带参数和只带一个参数的函数，都是以文件结束符作为终止提取的控制符。如提取一个字符结束会把指针移到下一个字符。

b. cin.get() 中带三个参数的函数，以字符个数或指定终止提取字符为终止提取的控制符。如提取字符结束不会把指针移到下一个字符。

c. cin.get( )函数族不忽略提取的空白字符。

2、标准输入流

- 用于字符输入的流成员函数

（2）getline函数读入一行字符

- 格式：**cin.getline(字符指针，n，终止字符)**
- 函数功能与带三个参数的get函数类似。
- 带三个参数的cin.get和cin.getline相同的是它们都不忽略提取过程中遇到的空白字符，当遇到终止字符时就停止提取。
- 带三个参数的cin.get和cin.getline不同的是停止提取时，**cin.getline会把指针移到终止字符后相邻的字节**，而带三个参数的cin.get函数不会。

```
int main(){
  char ch[20];
  cout << "Enter a sentence: " << endl;
  cin >> ch;
  cout << "The input string is: \""<< ch
       <<"\""<< endl;
  cin.getline(ch,20,'/');
  cout << "The second part is: \""<< ch
       <<"\""<< endl;
  cin.getline(ch,20,'/');
  cout << "The third part is: \""<< ch
       <<"\""<< endl;
  return 0;
}
```

I		I	i	k	e		C	+	+	.	/

I		s	t	u	d	y		C	+	+	.

/	I		a	m		h	a	p	p	y	.

2、标准输入流

- istream类的其他成员函数

（1）eof 函数

当输入缓冲区的指针遇到文件结束符时函数值为真，否则显假。从键盘用**ctrl+z**输入文件结束符。

```
int main()
{
  char c;
  while (!cin.eof())
    if ((c = cin.get()) != ' ')
      cout.put(c);
  return 0;
}
```

（2）peek 函数 c=cin.peek();

返回指针指向的当前字符，只是观测，指针仍停留在当前位置并不后移。如果是EOF，返回-1

2、标准输入流

□ istream类的其他成员函数

（3）putback 函数

将get或getline函数读取的字符ch返回流，插入当前指针位置。

```
int main()
{
  char ch[20];
  int c;
  cout << "Enter a sentence: " << endl;
  cin.getline(ch,15,'/');
  cout << "The first part is: \""<< ch <<"\""<< endl;
  c=cin.peek();
  cout << "The next character (ASCII code) is: "<<c<<endl;
  cin.putback(ch[0]);
  cin.getline(ch,15,'/');
  cout << "The second part is: \""<< ch <<"\""<< endl;
  return 0;
}
```

2、标准输入流

□ istream类的其他成员函数

（4）ignore 函数 cin.ignore(n,终止字符)

跳过n个字符，或在遇到终止字符时提前结束。

无参数默认1个字符

```
int main()
{
  char ch[20];
  cout << "Enter a sentence: " << endl;
  cin.get(ch, 20, '/');
  cout << "The input string is: \"" << ch << "\"" << endl;
  cin.ignore(); // 跳过一个字符
  cin.get(ch, 20, '/');
  cout << "The second part is: \"" << ch << "\"" << endl;
  return 0;
}
```

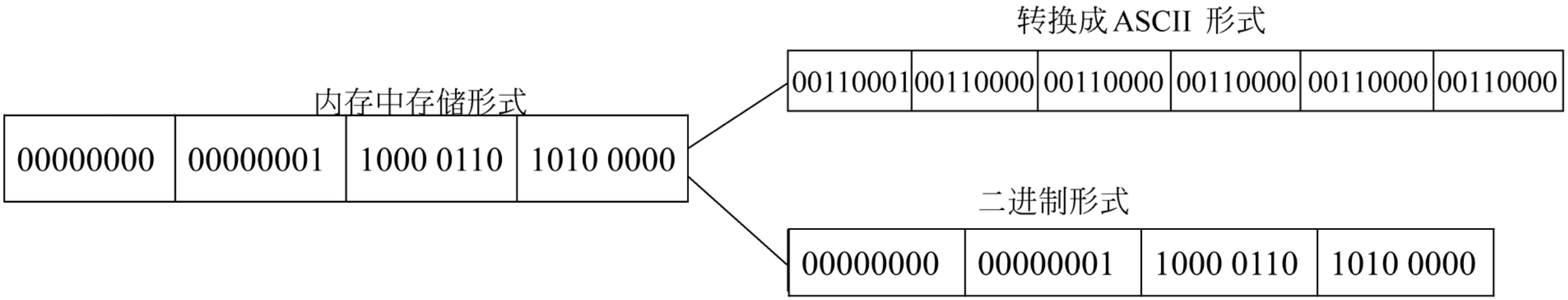
3、对数据文件的操作与文件流

□ 文件的概念

1. 文件是指存储在存储介质上的数据集合。
2. 操作系统把存储介质上的**相关数据抽象为文件**，用标识符为其取名并由文件系统管理文件。
3. 只要用户指出**文件名**，操作系统就可以按名存取文件信息。
4. 根据文件中数据的表示形式，文件分为**ASCII文件**和**二进制文件**
5. ASCII文件就是文本文件，**每个字节表示一个字符**。
6. 二进制文件是把内存中的数据、指令按其在**内存的格式**存放在磁盘上。
7. 字符信息在内存也是以ASCII码形式存放，所以**字符在ASCII码文件和在二进制文件中形式是一样的**。对于数值数据，两者是不一样的。

3、对数据文件的操作与文件流

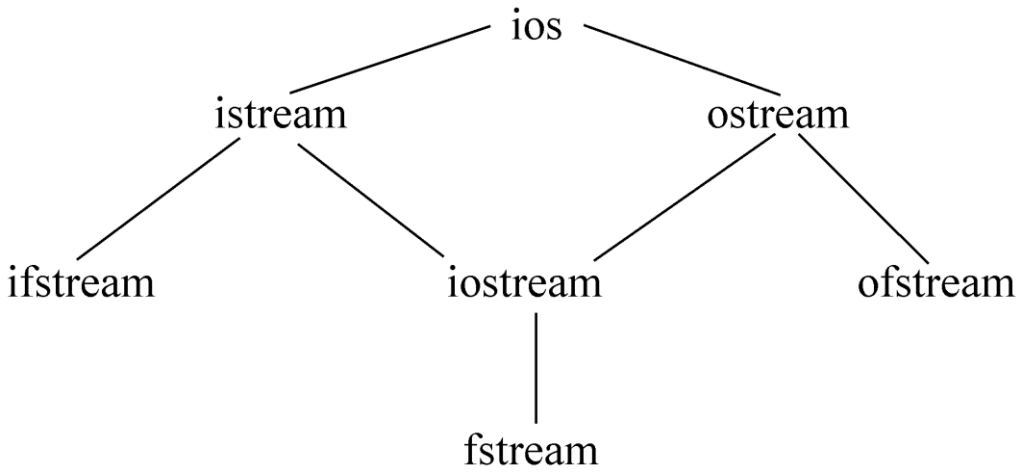
- 文件的概念
- 例如，一个十进制整数100000，用二进制表示时用四个字节而用ASCII码表示时用六个字节。



3、对数据文件的操作与文件流

□ 文件流类与文件流对象

1. 文件流是以外存文件为输入输出对象的数据流。**输出文件流**是从**内存流向**外存文件的数据流，**输入文件流**是从外存文件**流向内存**的数据流。为了弥补访问内存和访问外存的速度差，每个文件流都有一个内存缓冲区
2. ifstream类，支持从磁盘文件输入
3. ofstream类，支持向磁盘文件输出
4. fstream类，支持对磁盘文件输出和输出
5. 要对文件进行输入输出，必须定义一个文件流类对象，用对象调用类的成员函数对文件操作




```
ifstream infile;
ofstream outfile;
fstream iofile;
```

3、对数据文件的操作与文件流

□ 文件的打开与关闭

（1）在文件流对象和磁盘文件之间建立关联（路径及文件名）

（2）指定文件的格式（ASCII还是Binary）

（3）指定文件的工作方式（输入还是输出）

（4）打开文件的方式：open函数、或定义流对象时指定参数

```
ofstream outfile;
outfile.open("f1.txt",ios::out);
```

```
ofstream outfile("f1.txt",ios::out);
```

（5）关闭文件的方式：close函数

```
outfile.close();
```

3、对数据文件的操作与文件流

□ 文件的打开与关闭

表 7.6　文件输入输出方式设置值

方　　式	作　　用
ios :: in	以输入方式打开文件
ios :: out	以输出方式打开文件(这是默认方式),如果已有此名字的文件,则将其原有内容全部清除
ios :: app	以输出方式打开文件,写入的数据添加在文件末尾
ios :: ate	打开一个已有的文件,文件指针指向文件末尾
ios :: trunc	打开一个文件,如果文件已存在,则删除其中全部数据,如文件不存在,则建立新文件。如已指定了 ios :: out 方式,而未指定 ios :: app,ios :: ate,ios :: in,则同时默认此方式
ios :: binary	以二进制方式打开一个文件,如不指定此方式则默认为 ASCII 方式
ios :: nocreate	打开一个已有的文件,如文件不存在,则打开失败。nocreat 的意思是不建立新文件
ios :: noreplace	如果文件不存在则建立新文件,如果文件已存在则操作失败,noreplace 的意思是不更新原有文件
ios :: in | ios :: out	以输入和输出方式打开文件,文件可读可写
ios :: out | ios :: binary	以二进制方式打开一个输出文件
ios :: in | ios :: binar	以二进制方式打开一个输入文件

3、对数据文件的操作与文件流

- 对ASCII文件的操作

1. ASCII码文件也是文本文件，文件中一个字节存放一个字符。对ASCII码文件操作包括向文件写入字符和从文件读取字符。
2. 读写ASCII码文件有用文件流对象**提取（>>）、插入（<<）运算符**和用文件流对象调用类的**成员函数put, get, getline**方法。

默认新建或清除

```
int main()
{
  int a[10];
  ofstream outfile("f1.txt");
  if (!outfile)
  {
    cerr << "File open error!" << endl;
    exit(1);
  }
  cout << "Enter 10 integer numbers:"
       << endl;
  for (int i = 0; i < 10; i++)
  {
    cin >> a[i];
    outfile << a[i] << " ";
  }
  outfile.close();
  return 0;
}
```

3、对数据文件的操作与文件流

- 对ASCII文件的操作
- 例子：输入文件流

```
max = a[0];
order = 0;
for (i = 1; i < 10; i++)
  if (a[i] > max)
  {
    max = a[i];
    order = i;
  }
cout << "max=" << max << endl
     << "order=" << order << endl;
```

```
int main()
{
  int a[10], max, i, order;
  ifstream infile("f1.txt", ios::in);
  if (!infile)
  {
    cerr << "File open error!" << endl;
    exit(1);
  }
  for (i = 0; i < 10; i++)
  {
    infile >> a[i];
    cout << a[i] << " ";
  }
  cout << endl;
  infile.close();
  return 0;
}
```

3、对数据文件的操作与文件流

对ASCII文件的操作

- 例子：输入输出文件流

```
void save_to_file() // 写文件
{
  ofstream outfile("f2.txt");
  if (!outfile)
  {
    cerr << "open f2.txt error!" << endl;
    exit(1);
  }
  char c[80];
  cin.getline(c, 80);
  for (int i = 0; c[i] != 0; i++)
    if (c[i] >= 65 && c[i] <= 90 || c[i] >= 97 && c[i] <= 122)
    {
      outfile << c[i];
      cout << c[i];
    }
  cout << endl;
  outfile.close();
}
```

3、对数据文件的操作与文件流

- 对ASCII文件的操作
  - 例子：输入输出文件流

```
void get_from_file() // 读文件
{
  char ch;
  ifstream infile("f2.txt", ios::in);
  if (!infile)
  {
    cerr << "open f2.txt error!" << endl;
    exit(1);
  }
  ofstream outfile("f3.txt");
  if (!outfile)
  {
    cerr << "open f3.txt error!" << endl;
    exit(1);
  }
  while (infile.get(ch))
  {
    if (ch >= 97 && ch <= 122)
      ch = ch - 32;
    outfile << ch;
    cout << ch;
  }
  cout << endl;
  infile.close();
  outfile.close();}
```

3、对数据文件的操作与文件流

□ 对二进制文件的操作

1. 二进制文件是按内存中的数据存储形式写入磁盘文件，因此又称为**内存数据的映象文件**。
2. 对二进制文件操作与对文本文件操作相似的是先定义文件流对象，然后**打开**文件，使用完要**关闭**文件。
3. 在打开时必须指定文件的**存储形式是二进制形式**，二进制文件即可以作为输入文件也可以作为输出文件，还可以作为既能输入又能输出的文件。这是与ASCII文件不同的地方。

3、对数据文件的操作与文件流

- 对二进制文件的操作
- 调用格式：

输入文件流对象.read(内存指针, 长度);
输出文件流对象.write(内存指针, 长度);

```
int main()
{
  Student stud[3] =
{"Lisa", 1001, 18, 'f', "Frank", 1002, 19, 'm',
                  "Wendy", 1004, 17, 'f'};
  ofstream outfile("stud.dat", ios::binary);
  if (!outfile)
  {
    cerr << "open error!" << endl;
    abort();
  }

  outfile.write((char *)&stud, sizeof(stud));
  outfile.close();

  return 0;
}
```

3、对数据文件的操作与文件流

- 对二进制文件的操作
- 调用格式：

输入文件流对象.read(内存指针, 长度);
输出文件流对象.write(内存指针, 长度);

```
int main()
{
  Student stud[3];
  int i;
  ifstream infile("stud.dat", ios::binary);
  if (!infile)
  {
    cerr << "open error!" << endl;
    abort();
  }
  infile.read((char *)stud, sizeof(stud));
  infile.close();
  for (i = 0; i < 3; i++)
  {
    cout << "NO." << i + 1 << endl;
    cout << "Name: " << stud[i].name << endl;
    cout << "Number: " << stud[i].num << endl;
    cout << "Age: " << stud[i].age << endl;
    cout << "Gender: " << stud[i].sex << endl;
  }
  return 0;}
```

3、对数据文件的操作与文件流

□ 对二进制文件的操作

1. 为了随机读取二进制文件中数据，磁盘文件**用一个指针**表示当前要访问的位置。
2. 每次读或写文件后会自动修改指针。使**指针总是指向当前要访问的位置**
3. 对于二进制文件，允许**程序控制指针移动**，实现随机访问文件。文件流类提供了有关文件指针的成员函数
4. ios::beg　以**文件开始**为起点，这是默认值。ios::cur　以指针**当前位置**为起点
   ios::end　以**文件结尾**为起点

表 7.7　文件流与文件指针有关的成员函数

成员函数	作　　用
gcount( )	返回最后一次输入所读入的字节数
tellg( )	返回输入文件指针的当前位置
seekg(文件中的位置)	将输入文件中指针移到指定的位置
seekg(位移量,参照位置)	以参照位置为基础移动若干字节(“参照位置”的用法见说明)
tellp( )	返回输出文件指针当前的位置
seekp(文件中的位置)	将输出文件中指针移到指定的位置
seekp(位移量,参照位置)	以参照位置为基础移动若干字节

3、对数据文件的操作与文件流

□ 对二进制文件的操作

- 随机访问二进制文件

```
fstream iofile("stud.dat", ios::in | ios::out | ios::binary);
if (!iofile)
{
  cerr << "open error!" << endl;
  abort();
}
for (i = 0; i < 5; i++)
  iofile.write((char *)&stud[i], sizeof(stud[i]));
```

```
Student stud1[5];
for (i = 0; i < 5; i = i + 2)
{
  iofile.seekg(i * sizeof(stud[i]), ios::beg);
  iofile.read((char *)&stud1[i / 2], sizeof(stud1[i]));
}

iofile.seekp(2 * sizeof(stud[0]), ios::beg);
iofile.write((char *)&stud[2], sizeof(stud[2]));
iofile.seekg(0, ios::beg);
```

4、字符串流

□ 建立输出字符串流对象

1. ostrstream类的构造函数原型是

ostrstream::ostrstream(char ***bu**, int **n**, int **mode**=ios::out);

bu是指向字符数组首址的指针，n是指定流缓冲区的长度，第三个参数可省略，默认是ios::out。

例：ostrstream strout(ch1,20);

建立字符流对象strout，并与字符数组ch1关联（通过字符串流把数据写入字符数组ch1），流缓冲区长度是20个字节。

4、字符串流

- 建立输入字符串流对象

istrstream::istrstream(char *bu, int n);

istrstream::istrstream(char *bu);

bu是指向字符数组首址的指针，n 是流缓冲区的长度，如没有n，表示缓冲区的长度与字符串数组长度相同。

例：istrstream strin(ch2);

istrstream strin(ch2, 20);

- 建立输入输出字符串流对象

strstream::strstream(char *bu, int n, int mode);

strstream strio(ch3, sizeof(ch3),ios::in|ios::out);

小结

- C++的输入和输出
- 标准输出流
- 标准输入流
- 对数据文件的操作与文件流
- 字符串流