C++语言程序设计

第八章 流类库与输入/输出

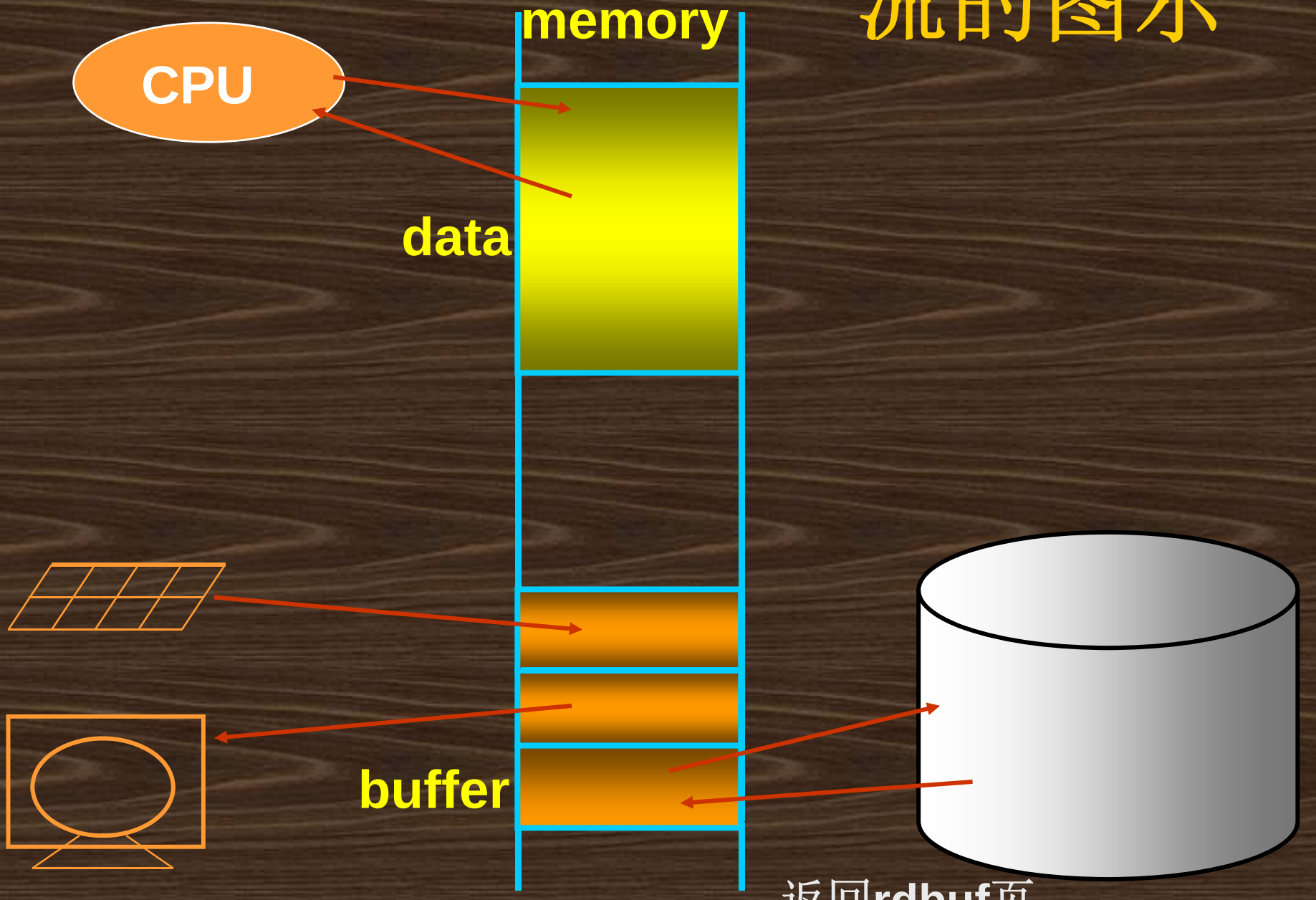
本章主要内容

- I/O流的概念
- 输出流
- 输入流

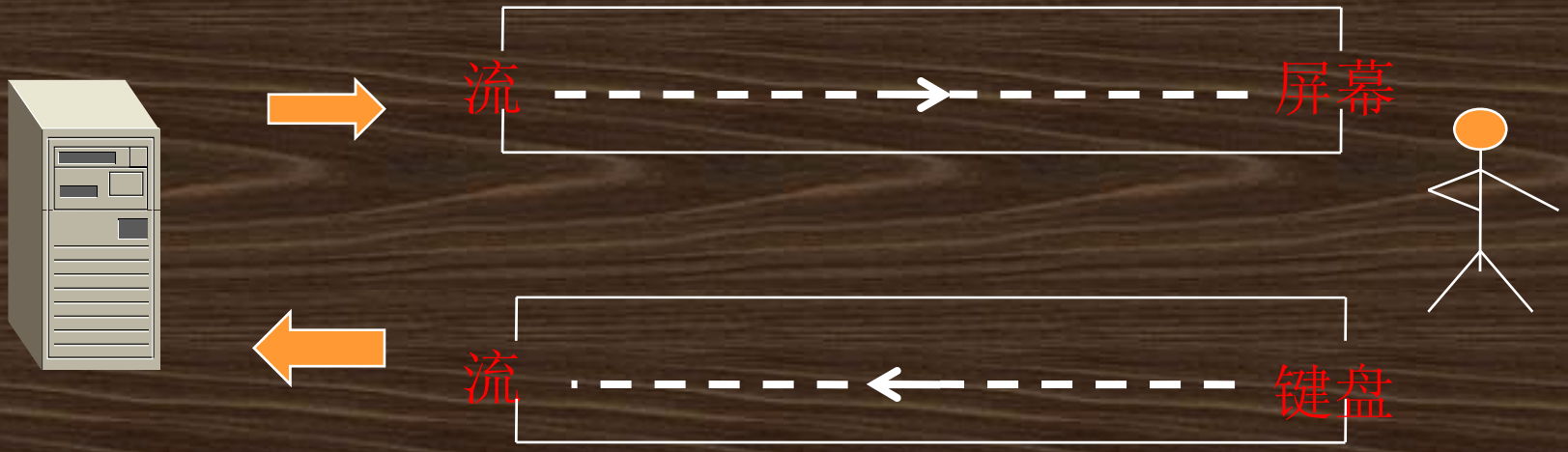
I/O流的概念

- 我要将一封信送给远在北京的朋友，只要将信写好，封上，投入邮筒即可。于是信便开始了它的旅行：进入邮局、分拣、登上火车（或飞机）、再分拣、投递到收信人手中。这段过程我是全然不知的，只是在信发出后几天，朋友就收到了。
- 如果将数据比作信，邮政系统则是“流”，程序对数据的“收发”则是对流的I/O操作。
- 其实“流”是司空见惯的，马路、铁道、航线、传送带、自动扶梯、电梯...都是流，刘就是个通道。

流的图示



C++语言把设备之间的信息交换称作“流”是非常形象的。外部设备到计算机的输入信息和计算机到外部设备的输出信息就像是一条条的水流。



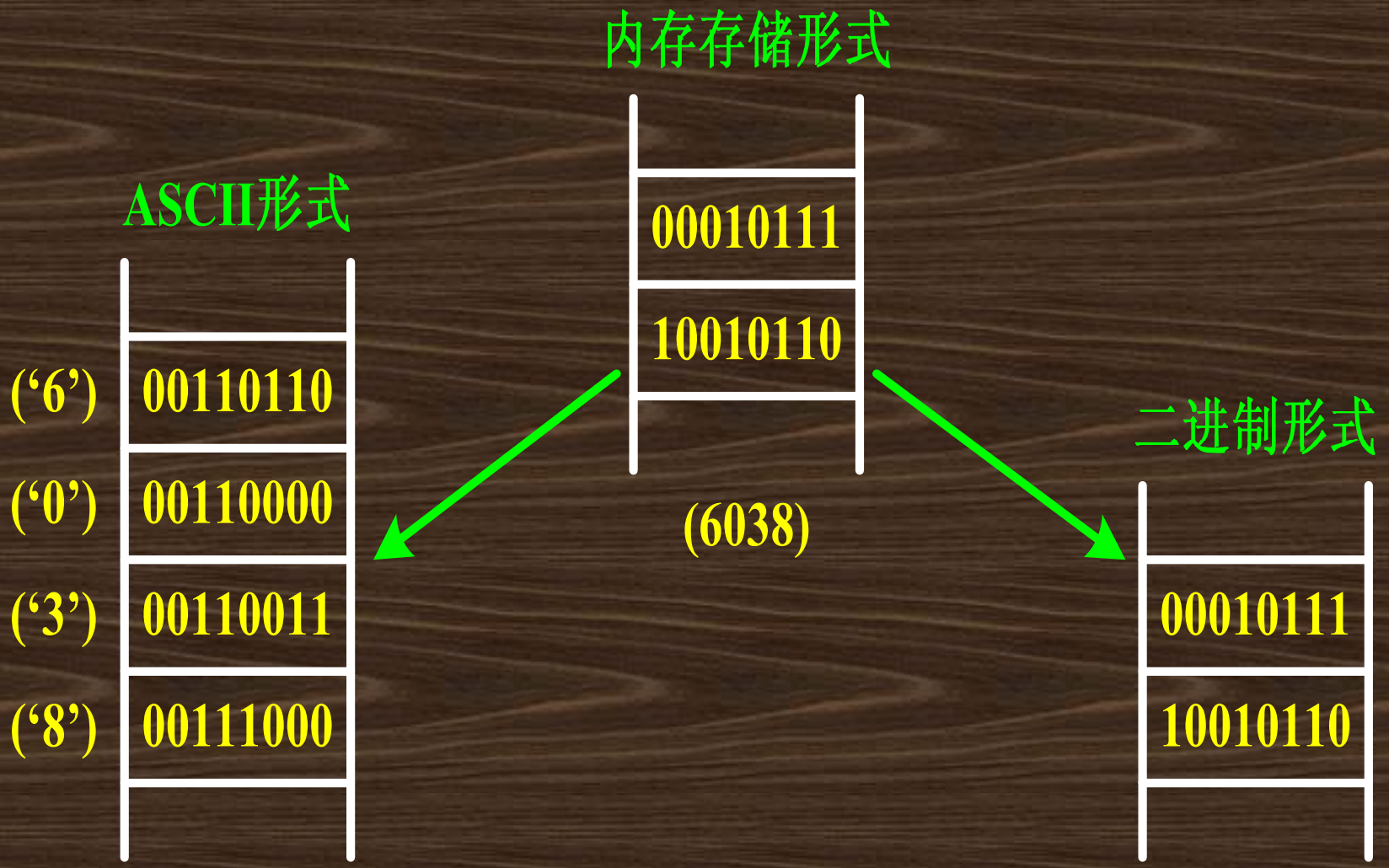
流的工作原理

- “流”是个字节搬运系统，它只管把内存或外设上的一个个字节的东西(无论是字符的文本，还是二进制的数据)搬来搬去，根本不理会搬的是些什么，统统视为川流不息的字节。流就是这个搬运系统的总称。至于搬运的“源”和“目的地”，流也是不加区分的。于是标准操作、文件操作、串操作都统一起来了。
- 至于如何理解这些字节，就是程序的事了。外设(比如硬盘)也不关心放在里面的究竟是些什么，反正要用还得程序取走。
- 程序依靠的是两件法宝：一是数据类型；二是流的工作方式(文本方式或二进制方式)。

两种工作方式

- 文本方式：“流”在搬运字节给外设时，把内存中的数据按照人的习惯对照ASCII表，一个个的转换成字符，一个字节装一个字符，写到外设上；从外设搬运字节给内存时，再把外设上的一个个的字符还原成数据原本的模样。比如整型的65535，按ASCII码转换就变成了54、53、53、51、53
- 二进制方式：“流”在搬运字节时(不管是输入还是输出)，都会原封不动的搬运，不对字节做任何加工。还是整数65535，用二进制表示占两个字节：
  11111111 11111111

文本文件和二进制文件：


内存存储形式
00010111
10010110
(6038)
ASCII形式
('6') 00110110
('0') 00110000
('3') 00110011
('8') 00111000
二进制形式
00010111
10010110

- 当程序与外界环境进行信息交换时，要同时动用两个对象，一个是程序中的对象，另一个是文件对象。文件是物理概念，流是逻辑概念。
- 流是文件的抽象，表示了数据的流淌。它负责在数据的生产者（源头）和数据的消费者（目的地）之间建立联系，并管理数据的流动。
- 生产者和消费者可以是变量，常量、数组、对象、引用、指针所指、还可以是键盘、显示器、文件等设备。
- 键盘是字符设备，显示器是块设备。

- C++把C的FILE类型的结构体改造成了流对象，把对流设备的操作收编为流类的成员函数，把文件视为流对象所控制把持的另一个对象。则流与内存(缓冲区)、与文件对象就形成了类的聚集关系。IO操作就完全变成了流的自家事了。
- 程序建立流对象，并指定这个流对象与某文件对象建立连接。程序操作流对象，流对象通过文件系统对所连接的文件对象产生作用。流对象还要把持通道资源。

- 读操作在流数据抽象中被称为（从流中）提取，写操作被称为（向流中）插入。
- 广义地，计算机将输入输出设备都视为文件。

- 若将流对象视为旅行社（提供旅游目的地和交通工具），则字节就是游客，文件对象就是出发或目的地，提取和插入就是行驶方向，缓冲区就是候车厅，流就是某某几日游这样一次旅游活动的总称。

流的分类

- 标准流

  是内存与标准外设（键盘、显示器）间的数据操作。
  操作方式：文本字符，顺序
  头文件：**iostream**
  缓冲类：**stdiobuf**

- 串流

  是内存（数据区）与内存（模拟外设）间的数据操作。
  操作方式：文本字符，顺序
  头文件：**sstream**
  缓冲类：**strstreambuf**

- 文件流

  是内存与外存（磁盘）间的数据操作。
  操作方式：文本字符或二进制字节数据，顺序或随机
  头文件：**fstream**
  缓冲类：**filebuf**

输出流

- 三个重要的输出流是:
  - ostream
  - ofstream
  - ostringstream
- 这是三个输出流类。

用流类iostream替代stdio

- 用流类iostream替代stdio的理由：

  stdio的函数类型不安全；

  stdio的函数的不可扩充性；

  （恰是这两点展示了C++的绝活。）
- 次要的缺点是：将输入输出变量与控制输入输出的格式分开书写的规矩。

输出流对象

输出流

- 系统定义的输出流对象:
  - cout 标准输出，有缓冲。
  - cerr 标准错误输出，没有缓冲，发送给它的内容立即被输出，不可重定向到它处。
  - clog 类似于cerr，但是有缓冲，缓冲区满时被输出。

输出流对象

输出流

- ofstream类支持磁盘文件输出
- 如果在构造函数中指定一个文件名，当构造这个文件时该文件是自动打开的
  - ofstream myFile("filename",ios
- 可以在调用默认构造函数之后使用open成员函数打开文件

```
ofstream myFile; //声明一个输出文件流对象
myFile.open("filename",iosmode);
    //打开文件，使流对象与文件建立联系
ofstream* pmyFile = new ofstream;
   //建立一个输出文件流对象
        pmyFile->open("filename",iosmode);
   //打开文件，使流对象与文件建立联系
```

何谓打开文件？

如何创建的？

如何创建的？

插入运算符（<<）

输出流

- 插入(<<)运算符是输出流对象的成员函数，是针对所有C++基本数据类型预先设计的，它不支持用户自定义类型的对象。
- 用于按字节传送给输出流对象。

重载插入运算符（<<）

插入(<<)运算符是ostream流类的重载了的友元函数:

```
ostream & operator <<(ostream & s , class & r)
  {
        ....
        return s;
  }
```

其中 r 是class类的引用，是被输出的目标;

s 是ostream类的引用,其对象是 cout。

后面的提取运算符(>>)也类似，不另。

使用'提取运算符重载函数'

```
#include <iostream.h>
class coord                          //类的定义
{
private:
    int x,y;                         //私有数据
public:                              //外部接口
    coord(){x=0;y=0;}
    coord(int i,int j){x=i;y=j;}
    friend ostream & operator << (ostream &
    stream, coord obj);              //声明为友元函数'
};
```

```
ostream & operator <<(ostream & stream ,coord obj)
        //友元函数实现
{  cout<<"正在使用'提取运算符重载函数' ...."<<endl;
   stream<<obj.x<<" , "<<obj.y<<endl<<endl;
   return stream;
}
void main()
{
   cout<<"进入主程序 ...."<<endl<<endl;
   coord a(56,78),b(99,88);          //生成类的对象
   cout<<a<<b<<endl;          //使用'提取运算符重载函数'
   cout<<"退出主程序 !"<<endl<<endl;
}
```

控制输出格式

输出流

- 控制输出宽度
  - 为了调整输出，可以通过在流中放入setw操纵符或调用width成员函数为每个项指定输出宽度。
- 例11-1 使用width控制输出宽度

```
#include <iostream>
using namespace std;
void main()
{ double values[] = {1.23,35.36,653.7,4358.24};
  for(int i=0;i<4;i++)
  { cout.width(10);
    cout << values[i] <<'\n';
  }
}
```

调用成员函数

说出此句的语义及完成途径

输出结果:

```
      1.23
     35.36
     653.7
   4358.24
```

例：使用*填充

输出流

```
#include <iostream>
using namespace std;
void main()
{ double values[]={1.23,35.36,653.7,4358.24};
  for(int i=0; i<4; i++)
  {  cout.width(10);
     cout.fill('*');
     cout<<values[i]<<'\n';
  }
}
```

输出结果:

******1.23
*****35.36
*****653.7
***4358.24

例11-2使用setw指定宽度

输出流

```
#include <iostream>
#include <iomanip>
using namespace std;
void main()
{ double values[]={1.23,35.36,653.7,4358.24};
  char *names[]={"Zoot","Jimmy","Al","Stan"};
  for(int i=0;i<4;i++)
   cout<<setw(6)<<names[i]
       <<setw(10)<<values[i]
       <<endl;
}
```

这不是调用成员函数

输出结果:

```
  Zoot      1.23
 Jimmy     35.36
    Al     653.7
  Stan   4358.24
```

关于“操纵符函数”

是定义在头文件iomanip中的用于操控输入输出格式的一些函数。它们和流类的成员函数功能上很类似，但用法上不同。可分为无参和有参两大类:

无参操纵符函数:

lock	对文件句柄加锁，相当于"%d"
unlock	对文件句柄解锁
dec	以十进制格式输入输出数据
hex	以十六进制格式输入输出数据，相当于"%X"
oct	以八进制格式输入输出数据，相当于"%o"
ws	输入时忽略打头的空白
endl	加入一个换行符，并刷新输出流
ends	加入一个空字符(null)，并刷新输出流
flush	强制刷新输出流

有参操纵符函数:

setbase(int n) 将数制基数设为n 进制

setw(int n) 设显示域宽度为 n个字符

setfill(char c) 设填充字符为 c

setiosflages(long f) 按 f 参数设置格式

- ios::fixed 固定的浮点显示
- ios::scientific 指数表示
- ios::left 左对齐
- ios::right 右对齐
- ios::skipws 忽略前导空白
- ios::uppercase 16进制数大写输出
- ios::lowercase 16进制小写输出
- ios::showpoint 强制显示小数点
- ios::showpos 强制显示符号

resetiosflages(long f) 解除f 参数设置的格式

setprecision(int n) 将数据精度设为n

格式标志 p375

操纵符函数setiosflages(long f) 所使用的实参:

ios_base:: skipws	0x0001	入
ios_base:: left	0x0002	出
ios_base:: right	0x0004	出
ios_base:: internal	0x0008	出
ios_base:: dec	0x0010	入/出
ios_base:: oct	0x0020	入/出
ios_base:: hex	0x0040	入/出
ios_base:: showbase	0x0080	出
ios_base:: showpoint	0x0100	出
ios_base:: uppercase	0x0200	出
ios_base:: showpos	0x0400	出
ios_base:: scientific	0x0800	出
ios_base:: fixed	0x1000	出
ios_base:: unitbuf	0x2000	出
ios_base:: stdio	0x4000	出

例11-3设置对齐方式

输出流

```
#include <iostream>
#include <iomanip>
using namespace std;
void main()
{ double values[]={1.23,35.36,653.7,4358.24};
  char *names[]={"Zoot","Jimmy","Al","Stan"};
  for(int i=0;i<4;i++)
   cout<<setiosflags(ios::left)
       <<setw(6)<<names[i]
       <<resetiosflags(ios::left)
       <<setw(10)<<values[i]
       <<endl;
}
```

设置为左对齐

取消设置

输出结果:

Zoot	1.23
Jimmy	35.36
Al	653.7
Stan	4358.24

例11-4控制输出精度

输出流

```
#include <iostream>
#include <iomanip>
using namespace std;
void main()
{ double values[]={1.23,35.36,653.7,4358.24};
  char *names[]={"Zoot","Jimmy","Al","Stan"};
  cout<<setiosflags(ios::scientific);
  for(int i=0;i<4;i++)
   cout<<setiosflags(ios::left)
       <<setw(6)<<names[i]
       <<resetiosflags(ios::left)
       <<setw(10)<<setprecision(1)
       << values[i]<<endl;
}
```

输出结果:

```
Zoot        1
Jimmy     4e+001
Al        7e+002
Stan      4e+003
```

进制

输出流

dec、oct和hex操纵符设置数据的输入和输出所使用的进制。

输出文件流成员函数

输出流

- 输出流成员函数有三种：
  - 与操纵符等价的成员函数。
  - 执行非格式化写操作的成员函数。
  - 其它修改流状态且不同于操纵符或插入运算符的成员函数。

输出文件流成员函数

输出流

- open函数
  把流与一个特定的磁盘文件关联起来。
  需要指定打开模式。
- put函数
  把一个字符写到输出流中，只一个字符。
- write函数
  把内存中的一块内容写到输出文件流中
- seekp和tellp函数
  操作文件流的内部指针
- close函数
  关闭与一个输出文件流关联的磁盘文件
- 错误处理函数
  在写一个流时进行错误处理

open函数的函数原型

程序对文件的访问，前提是建立文件与流的关联，即所谓“打开”。文件打开成功，对文件的访问就变成对流的访问。

```
void  open(
  char const *,    //可带盘符路径的文件名（串）
  int  fmode=out/in,     //打开方式
  int access=filebuf::openprot   //保护方式:
  );
```

- 0 普通
- 1 只读
- 2 隐含
- 4 系统
- 8 备份

输出文件流打开方式 p377

0x08	ios_base:: app	追加到文件末尾	出
0x04	ios_base:: ate	打开现存文件，到末尾	出
0x01	ios_base:: in	打开输入文件，不删除	入/出
0x02	ios_base:: out	打开输出文件	出
0x10	ios_base:: trunc	打开输出文件若存在则删除	出
0x20	ios_base:: nocreat		出
0x40	ios_base:: noreplace		出
0x80	ios_base:: binary		入/出