

2019年春

程序设计实习(I): C++程序设计

第十一讲 输入输出流

刘家瑛

liujiaying@pku.edu.cn



课前多吼歪

- 本周上机采用自愿到机房原则
 - 去年期中考题—模考
 - 没有助教
 - 地点不限
- ACM校内赛对应有加分, 请积极参加
 - 期中考前热身
 - 组队参赛好经历
- 期中考试安排在5月18日



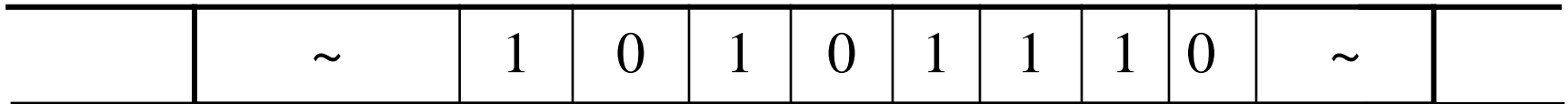
输入输出流

- 流的概念模型
- C++中与流操作相关的类及其继承关系
- 输入输出流对象: `cin` / `cout` / `cerr` / `clog`
- 输出流
- 输入流
- 无格式输入输出
- 流操纵算子
- 流格式状态

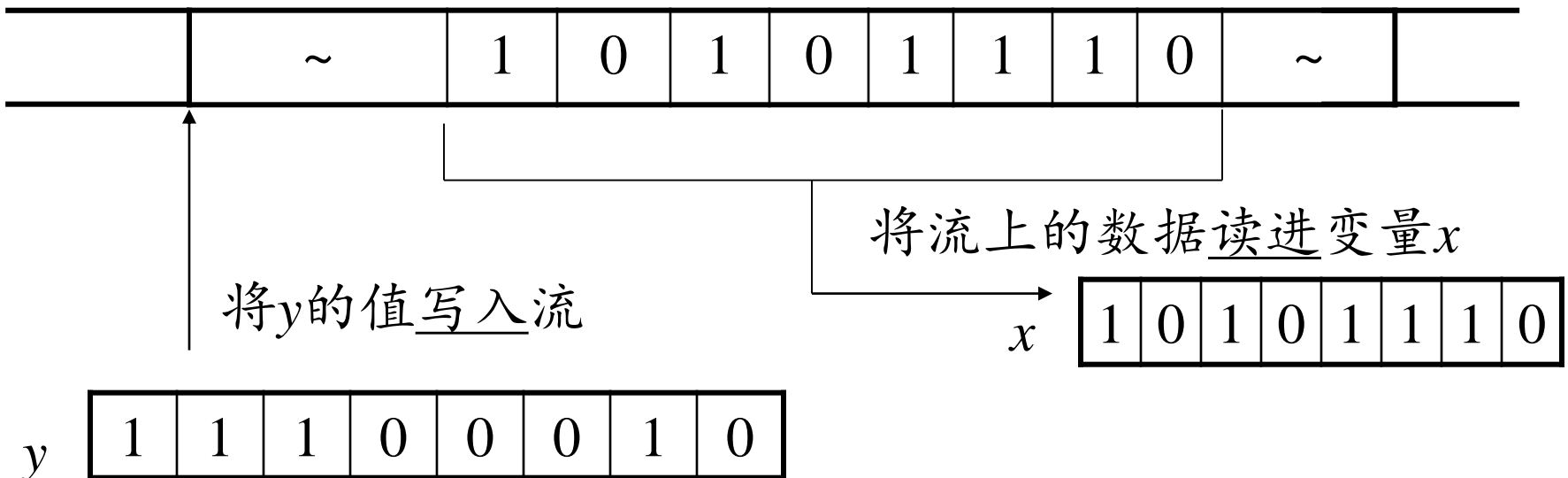


流的概念模型

- **流** — 可以看作一个无限长的二进制数字序列



- 通过**读写指针**进行流的读和写 (以字节为单位)



流的概念模型

- 输出流

可以看作一端无限, 另一端通过写指针不停

向后写入新内容的单向流

~	1	1	0	0	0	1	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---



写指针

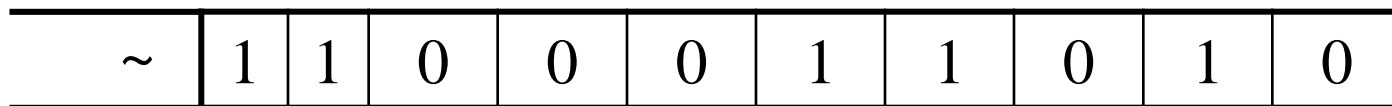


流的概念模型

- 输入流

可以看作一端无限, 另一端通过读指针不停从流中读取新内容的单向流,

读出的内容从流中删去



↓ 读指针



有格式读写和无格式读写

- 有格式读写

- 以某种数据类型为单位读写
- 例: 读一个整数, 写一个浮点数等

- 无格式读写

- 以字节为单位读写, 不区分其中的内容
- 例: 读20个字节, 写50个字节等

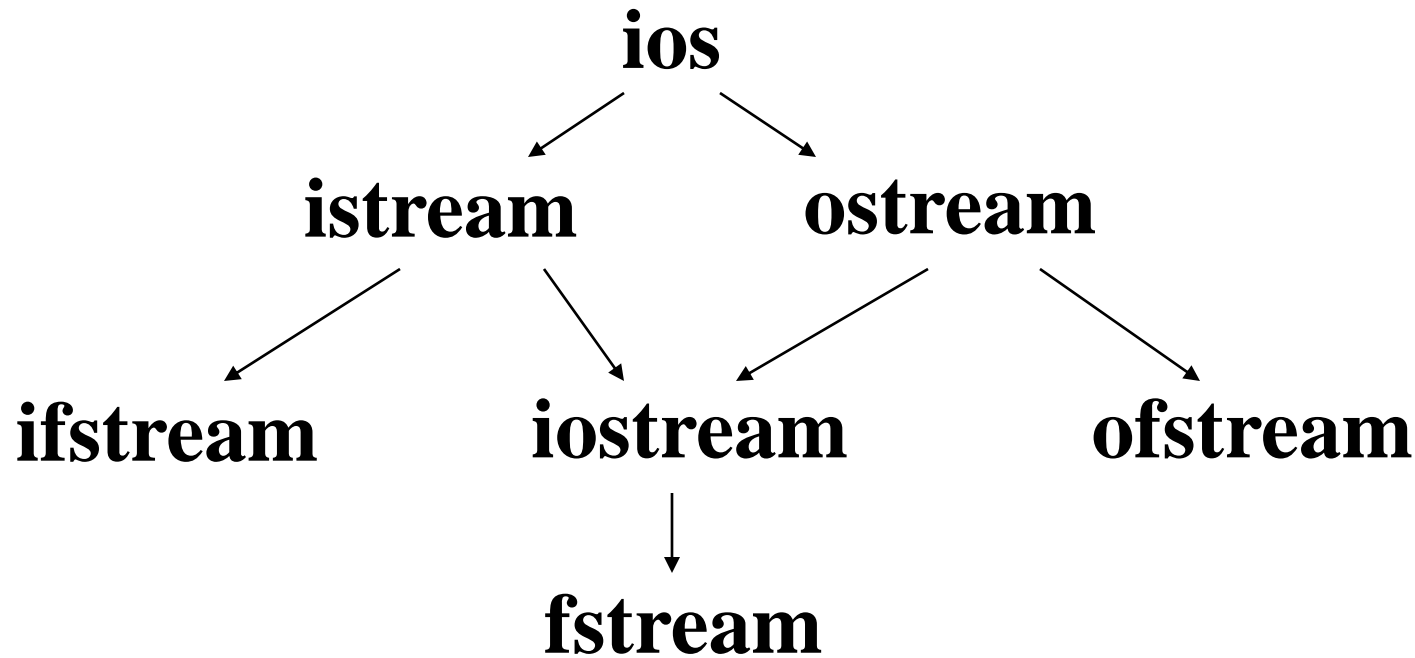


缓冲区刷新

- 向输出流中写数据时,
 - 先向缓冲区中写
 - 当缓冲区写满时,才真正向输出流写
- 可以通过函数在程序中主动将缓冲区内容写入输出流



C++中与流操作相关的类及其继承关系



../vc/crt/src/cout.cpp



标准流对象

- 输入流对象

- **cin** 与标准输入设备相连 (可重定向为从文件中读取数据)

- 输出流对象

- **cout** 与标准输出设备相连 (可以重定向为向文件写入数据)

- **cerr** 与标准错误输出设备相连, 非缓冲输出 (不可以重定向)

- **clog** 与标准错误输出设备相连, 缓冲输出 (不可以重定向)

- 缺省情况下

```
cerr << "Hello, world" << endl;
```

```
clog << "Hello, world" << endl;
```

和 `cout << "Hello, world" << endl;` 一样

* 重定向: 将输入的源或输出的目的地改变



输出重定向

```
#include <iostream>
using namespace std;
int main() {
    int x, y;
    cin >> x >> y;
    freopen("test.txt", "w", stdout); //将标准输出重定
                                        //向到test.txt文件
    if(y == 0) //除数为0则输出错误信息
        cerr << "error." << endl;
    else
        cout << x / y ;
    return 0;
}
```



输出流

- 流插入运算符 <<

`cout << "Good morning!\n";` 不刷新缓冲区

`cout << "Good";` 不刷新缓冲区

`cout << "morning!";` 不刷新缓冲区

`cout << endl;` 刷新缓冲区

`cout << flush;` 刷新缓冲区



输出流

- 用成员函数 **put** 输出字符

```
cout.put('A');
```

- put的连续使用

```
cout.put('A').put('a');
```



输入流

- 可以用如下方法判输入流结束:

```
int x;
```

```
while ( cin>>x ) {
```

```
    ...
```

```
}
```

```
return 0;
```

- 如果从键盘输入, 则输入Ctrl+Z代表输入流结束
- 如果从文件输入, 例如前面有

```
freopen("some.txt", "r", stdin); //将一个指定的文件打开一个  
//预定义的流: 标准输入
```

读到文件尾部, 输入流就算结束



输入流

- 读取运算的返回值

重载 >> 运算符的定义:

```
istream &operator >> (istream &input, A & a){  
    ...  
    return input;  
}
```

可以用如下方法判输入结束:

```
int x;  
while ( cin>>x ) { ... } // 类型不匹配, 为什么可以?
```

在istream或其基类里重载了 **operator bool**



```
#include <iostream>
using namespace std;
class MyCin
{
};
```

```
int main()
{
    MyCin m;    // m $\leftrightarrow$ cin
    int n;
    while( m >> n )
        cout << "number:" << n << endl;
    return 0;
}
```

补齐MyCin类, 要求输入100, 则程序结束




```
#include <iostream>
using namespace std;
class MyCin
{
    bool bStop;
public:
    MyCin():bStop(false) { }
    operator bool( ) { //重载类型强制转换运算符 bool
        return !bStop;
    }
    MyCin & operator >> (int n){
        cin >> n;
        if(n == 100) bStop = true;
        return * this;
    }
};
```



输入流

- **istream**类的成员函数

int get() -- 读入一个字符, 返回该字符的ASCII码;
读到输入末尾, 则返回 EOF

```
#include <iostream>
using namespace std;
int main(){
    int c;
    while( ( c = cin.get()) != EOF)
        cout.put(c);
    return 0;
}
```

get函数不会跳过空格, 制表符, 回车等特殊字符,
所有字符都能被读入



输入流

- **istream类的成员函数**

istream & getline(char * buf, int bufSize);

istream & getline(char * buf, int bufSize, char delim);

- **第一个:**

从输入流中读取bufSize-1个字符到缓冲区buf, 或读到 '\n' 为止
函数会自动在buf中读入数据的结尾添加 '\0'

- **第二个: 读到delim为止**

'\n' 或 delim 都不会被读入buf, 但会被从输入流中删掉
如果输入流 '\n' 或delim 之前的字符个数超过了bufSize个,
就导致读入出错

→ 其结果就是: 本次读入已经完成, 但之后的读入就都会失败

□ 可以用 **if(!cin.getline (...))** 判断输入是否结束



输入流

- **cin.peek()**: 返回下一个字符, 但不从流中去掉
- **cin.putback(char ch)**: 将字符ch放回输入流
- **cin.gcount()**: 返回上次读入的字节数
- **cin.ignore(int *nCount* = 1, int *delim* = EOF)**:
从流中删掉最多nCount个字符, 遇到EOF时结束



无格式输入输出

- 用read/write进行指定字节数的输入输出

```
const int SIZE = 80;
```

```
char buffer[SIZE];
```

```
cin.read(buffer, 20);
```

```
    //cin.get(buffer, 20);
```

```
cout.write(buffer, cin.gcount());
```

```
    //gcount返回上次读入的字节数
```

```
cout << endl;
```

输入: Using the read, write and gcount member functions

输出: Using the read, write



流操纵算子

- 整数流的基数: dec, oct, hex, setbase
- 浮点数的精度(precision, setprecision)
- 设置域宽(setw, width)

使用流操纵算子需要 `#include <iomanip>`



流操纵算子

- 整数流的基数: 流操纵算子 dec, oct, hex

```
int n = 10;
```

```
cout << n << endl;
```

```
cout << hex << n << “\n” //十六进制
```

```
    << dec << n << “\n” //十进制
```

```
    << oct << n << endl; //八进制
```

- 输出结果:

10

a

10

12



流操纵算子

- 浮点数的精度 (**precision, setprecision**)

precision是成员函数, 其调用方式为:

cin.precision(5);

setprecision 是流操作算子, 其调用方式为:

cin >> setprecision(5); // 可以连续输出

- 功能相同: 指定输出浮点数的有效位数



控制浮点数精度的流操纵算子

```
#include <iostream>
```

```
#include <iomanip>
```

```
using namespace std;
```

```
int main(){
```

```
    double x = 1234567.89, y = 12.34567;
```

```
    int n = 1234567;
```

```
    int m = 12;
```

```
    cout << setprecision(6) << x << endl
```

```
        << y << endl << n << endl << m;
```

```
}
```

输出：

1.23457e+006

12.3457

1234567

12

浮点数输出最多
6位有效数字



控制浮点数精度的流操纵算子

- 流格式操纵算子 **setiosflags**

- `setiosflags(ios::fixed)` 用定点方式表示实数
- `seiosflags(ios::scientific)` 用指数方式表示实数
- `setiosflags(ios::fixed)` 可以和 `setprecision(n)` 合用

控制小数点右边的数字个数

- `seiosflags(ios::scientific)` 可以和 `setprecision(n)` 合用

控制指数表示法的小数位

- 在用浮点表示的输出中, `setprecision(n)`表示有效位数
- 在用定点/指数表示的输出中, `setprecision(n)`表示小数位数
- 小数位数截短显示时, 进行四舍五入处理



控制浮点数精度的流操纵算子

```
#include <iostream>
```

```
#include <iomanip>
```

```
using namespace std;
```

```
int main(){
```

```
    double x = 1234567.89, y = 12.34567;
```

```
    int n = 1234567;
```

```
    int m = 12;
```

```
    cout << setiosflags(ios::fixed)
```

```
        << setprecision(6) << x << endl
```

```
        << y << endl << n << endl << m;
```

```
}
```

输出：

1234567.890000

12.345670

1234567

12

以小数点位置固定
的方式输出



控制浮点数精度的流操纵算子

```
#include <iostream>
#include <iomanip>
using namespace std;
int main(){
    double x = 1234567.89;
    cout << setiosflags(ios::fixed)
        << setprecision(6) << x << endl
        << resetiosflags(ios::fixed) << x ;
}
```

输出：

1234567.890000

1.23457e+006

取消以小数点位置
固定的方式输出



流操纵算子

- 设置域宽 (**setw, width**)

两者功能相同, 一个是流操作算子, 另一个是成员函数,

调用方式不同:

cin >> setw(5); 或者 cin.width(5);

cout << setw(5); 或者 cout.width(5);

- 不含参数的width函数将输出当前域宽



流操纵算子

- 设置域宽 (setw, width)

例: `int w = 4;` 输入: 1234567890
`char string[10];` 输出: 1234
`cin.width(5);` 5678
`while(cin >> string){` 90
 `cout.width(w++);`
 `cout << string << endl;`
 `cin.width(5);`
}

输入操作提取字符串的最大宽度比定义的域宽小1,
因为在输入的字符串后面必须加上一个空字符



流操纵算子

- 设置域宽 (setw, width)

需要注意的是在每次读入和输出之前都要设置宽度
例如：

```
char str[10];
```

```
cin.width(5);
```

```
cin >> string;
```

```
cout << string << endl;
```

```
cin >> string;
```

```
cout << string << endl;
```

输入：1234567890

输出：1234

567890



流操纵算子

- 设置域宽 (setw, width)

需要注意的是在**每次**读入和输出之前都要设置宽度
例如：

```
char str[10];  
cin.width(5);  
cin >> string;  
cout << string << endl;  
cin.width(5);  
cin >> string;  
cout << string << endl;
```

输入: 1234567890

输出: 1234

5678

