

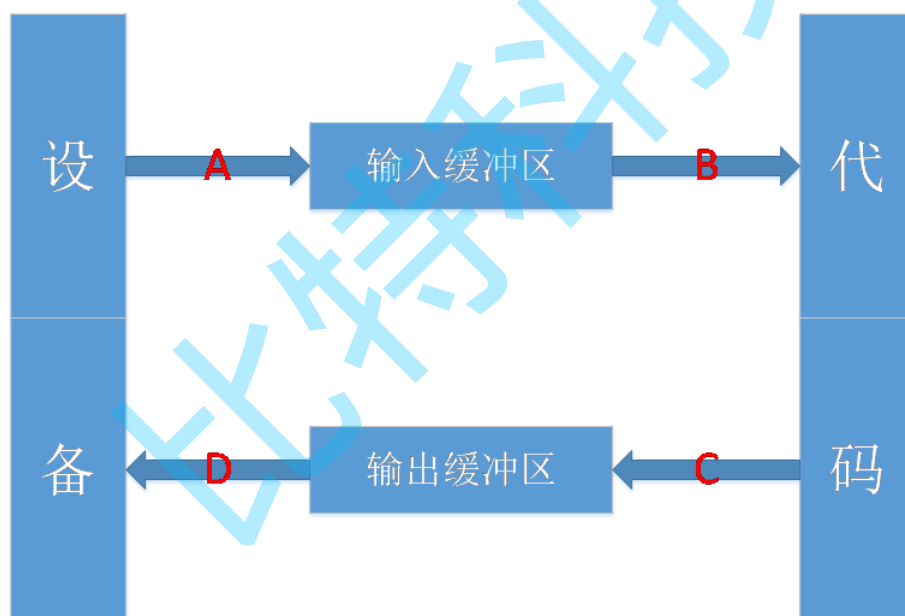
Lesson13---C++的IO流

【本节目标】

- 1. C语言的输入与输出
- 2. 流是什么
- 3. C++IO流
- 4. stringstream的简单介绍

1. C语言的输入与输出

C语言中我们用到的最频繁的输入输出方式就是scanf ()与printf()。scanf(): 从标准输入设备(键盘)读取数据, 并将值存放在变量中。printf(): 将指定的文字/字符串输出到标准输出设备(屏幕)。注意宽度输出和精度输出控制。C语言借助了相应的缓冲区来进行输入与输出。如下图所示:



对输入输出缓冲区的理解:

- 1.可以屏蔽掉低级I/O的实现, 低级I/O的实现依赖操作系统本身内核的实现, 所以如果能够屏蔽这部分的差异, 可以很容易写出可移植的程序。
- 2.可以使用这部分的内容实现“行”读取的行为, 对于计算机而言是没有“行”这个概念, 有了这部分, 就可以定义“行”的概念, 然后解析缓冲区的内容, 返回一个“行”。

2. 流是什么

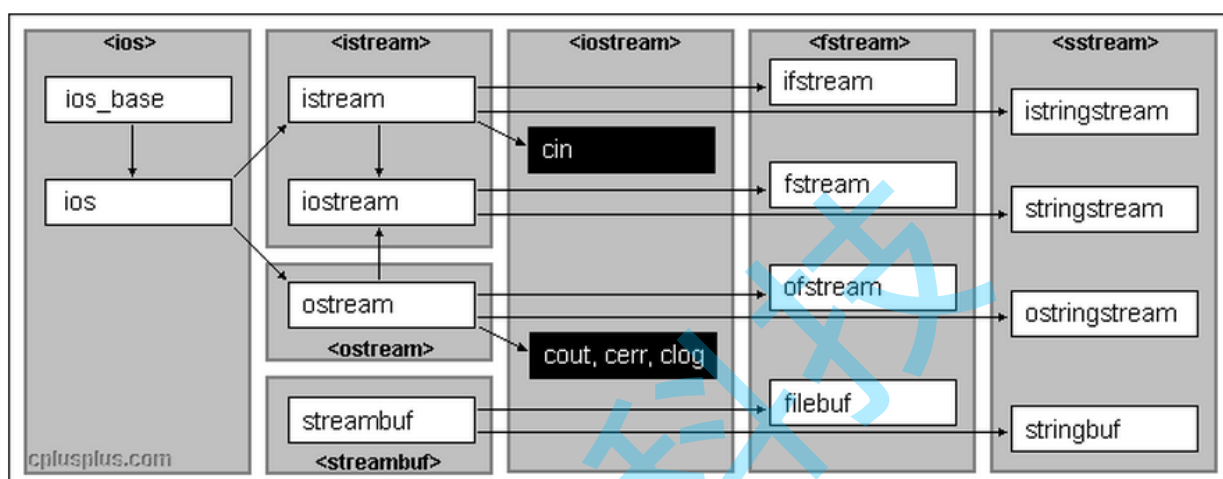
“流”即是流动的意思，是物质从一处向另一处流动的过程，是对一种有序连续且具有方向性的数据（其单位可以是bit, byte, packet）的抽象描述。C++流是指信息从外部输入设备（如键盘）向计算机内部（如内存）输入和从内存向外部输出设备（显示器）输出的过程。这种输入输出的过程被形象的比喻为“流”。

它的特性是：有序连续、具有方向性

为了实现这种流动，C++定义了I/O标准类库，这些每个类都称为流/流类，用以完成某方面的功能

3. C++IO流

C++系统实现了一个庞大的类库，其中ios为基类，其他类都是直接或间接派生自ios类



3.1 C++标准IO流

C++标准库提供了4个全局流对象cin、cout、cerr、clog，使用cout进行标准输出，即数据从内存流向控制台(显示器)。使用cin进行标准输入即数据通过键盘输入到程序中，同时C++标准库还提供了cerr用来进行标准错误的输出，以及clog进行日志的输出，从上图可以看出，cout、cerr、clog是ostream类的三个不同的对象，因此这三个对象现在基本没有区别，只是应用场景不同。

在使用时候必须要包含文件并引入std标准命名空间。

注意：

1. cin为缓冲流。键盘输入的数据保存在缓冲区中，当要提取时，是从缓冲区中拿。如果一次输入过多，会留在那儿慢慢用，如果输入错了，必须在回车之前修改，如果回车键按下就无法挽回了。只有把输入缓冲区中的数据取完后，才要求输入新的数据。
2. 输入的数据类型必须与要提取的数据类型一致，否则出错。出错只是在流的状态字state中对应位置位（置1），程序继续。
3. 空格和回车都可以作为数据之间的分隔符，所以多个数据可以在一行输入，也可以分行输入。但如果是字符型和字符串，则空格（ASCII码为32）无法用cin输入，字符串中也不能有空格。回车符也无法读入。
4. cin和cout可以直接输入和输出内置类型数据，原因：标准库已经将所有内置类型的输入和输出全部重载了：

member functions	<pre>ostream& operator<< (bool val); ostream& operator<< (short val); ostream& operator<< (unsigned short val); ostream& operator<< (int val); ostream& operator<< (unsigned int val); ostream& operator<< (long val); ostream& operator<< (unsigned long val); ostream& operator<< (float val); ostream& operator<< (double val); ostream& operator<< (long double val); ostream& operator<< (const void* val); ostream& operator<< (streambuf* sb); ostream& operator<< (ostream& (*pf)(ostream&)); ostream& operator<< (ios& (*pf)(ios&)); ostream& operator<< (ios_base& (*pf)(ios_base&));</pre>
global functions	<pre>ostream& operator<< (ostream& out, char c); ostream& operator<< (ostream& out, signed char c); ostream& operator<< (ostream& out, unsigned char c); ostream& operator<< (ostream& out, const char* s); ostream& operator<< (ostream& out, const signed char* s); ostream& operator<< (ostream& out, const unsigned char* s);</pre>
member functions	<pre>istream& operator>> (bool& val); istream& operator>> (short& val); istream& operator>> (unsigned short& val); istream& operator>> (int& val); istream& operator>> (unsigned int& val); istream& operator>> (long& val); istream& operator>> (unsigned long& val); istream& operator>> (float& val); istream& operator>> (double& val); istream& operator>> (long double& val); istream& operator>> (void*& val); istream& operator>> (streambuf* sb); istream& operator>> (istream& (*pf)(istream&)); istream& operator>> (ios& (*pf)(ios&)); istream& operator>> (ios_base& (*pf)(ios_base&));</pre>
global functions	<pre>istream& operator>> (istream& is, char& ch); istream& operator>> (istream& is, signed char& ch); istream& operator>> (istream& is, unsigned char& ch); istream& operator>> (istream& is, char* str); istream& operator>> (istream& is, signed char* str); istream& operator>> (istream& is, unsigned char* str);</pre>

5. 对于自定义类型，如果要支持cin和cout的标准输入输出，需要对<<和>>进行重载。

6. 在线OJ中的输入和输出：

- 对于IO类型的算法，一般都需要循环输入：

```
1 // 单个元素循环输入
2 while(cin>>a)
3 {
4     // ...
5 }
6
7 // 多个元素循环输入
8 while(c>>a>>b>>c)
9 {
10     // ...
11 }
12
13 // 整行接收
14 while(cin>>str)
15 {
16     // ...
17 }
```

- 输出：严格按照题目的要求进行，多一个少一个空格都不行。

3.2 C++文件IO流

C++根据文件内容的数据格式分为**二进制文件**和**文本文件**。采用文件流对象操作文件的一般步骤：

1. 定义一个文件流对象

- ifstream ifile(只输入用)
- ofstream ofile(只输出用)
- fstream iofile(既输入又输出用)

2. 使用文件流对象的成员函数打开一个磁盘文件，使得文件流对象和磁盘文件之间建立联系

3. 使用提取和插入运算符对文件进行读写操作，或使用成员函数进行读写

4. 关闭文件

```
1 // 使用文件IO流用文本及二进制方式演示读写配置文件
2 struct ServerInfo
3 {
4     char _ip[32];    // ip
5     int _port;       // 端口
6 };
7
8 struct ConfigManager
9 {
10 public:
11     ConfigManager(const char* configfile = "bitserver.config")
12         : _configfile(configfile)
13     {}
14
15     void WriteBin(const ServerInfo& info)
16     {
17         // 这里注意使用二进制方式打开写
18         ofstream ofs(_configfile, ifstream::out | ifstream::binary);
19         ofs.write((const char*)&info, sizeof(ServerInfo));
20         ofs.close();
21     }
22
23     void ReadBin(ServerInfo& info)
24     {
25         // 这里注意使用二进制方式打开读
26         ifstream ifs(_configfile, ifstream::in | ifstream::binary);
27         ifs.read((char*)&info, sizeof(ServerInfo));
28         ifs.close();
29     }
30
31     void WriteText(const ServerInfo& info)
32     {
33         // 这里会发现IO流写整形比C语言那套就简单多了,
34         // C 语言得先把整形itoa再写
35         ofstream ofs(_configfile);
36         ofs << info._ip << endl;
37         ofs << info._port << endl;
38         ofs.close();
39     }
40
41     void ReadText(ServerInfo& info)
42     {
```

```

43         // 这里会发现IO流读整形比C语言那套就简单多了,
44         // C 语言得先读字符串, 再atoi
45         ifstream ifs(_configfile);
46         ifs >> info._ip;
47         ifs >> info._port;
48         ifs.close();
49     }
50
51 private:
52     string _configfile; // 配置文件
53 };
54
55 int main()
56 {
57     ConfigManager cfgMgr;
58
59     ServerInfo wtinfo;
60     ServerInfo rdinfo;
61     strcpy(wtinfo._ip, "127.0.0.1");
62     wtinfo._port = 80;
63
64     // 二进制读写
65     cfgMgr.WriteBin(wtinfo);
66     cfgMgr.ReadBin(rdinfo);
67     cout << rdinfo._ip << endl;
68     cout << rdinfo._port << endl;
69
70     // 文本读写
71     cfgMgr.WriteText(wtinfo);
72     cfgMgr.ReadText(rdinfo);
73     cout << rdinfo._ip << endl;
74     cout << rdinfo._port << endl;
75
76     return 0;
77 }

```

4 stringstream的简单介绍

在C语言中, 如果想要将一个整型变量的数据转化为字符串格式, 如何去做?

1. 使用itoa()函数
2. 使用sprintf()函数

但是两个函数在转化时, 都得需要先给出保存结果的空间, 那空间要给多大呢, 就不太好界定, 而且转化格式不匹配时, 可能还会得到错误的结果甚至程序崩溃。

```

1  int main()
2  {
3      int n = 123456789;
4      char s1[32];
5      _itoa(n, s1, 10);
6
7      char s2[32];
8      sprintf(s2, "%d", n);
9
10     char s3[32];
11     sprintf(s3, "%f", n);
12     return 0;
13 }

```

在C++中，可以使用stringstream类对象来避开此问题。

在程序中如果想要使用stringstream，必须要包含头文件。在该头文件下，标准库三个类：istringstream、ostringstream 和 stringstream，分别用来进行流的输入、输出和输入输出操作，本文主要介绍stringstream。

stringstream主要可以用来：

1. 将数值类型数据格式化为字符串

```

1  #include<sstream>
2  int main()
3  {
4      int a = 12345678;
5      string sa;
6
7      // 将一个整形变量转化为字符串，存储到string类对象中
8      stringstream s;
9      s << a;
10     s >> sa;
11
12     // clear()
13     // 注意多次转换时，必须使用clear将上次转换状态清空掉
14     // stringstream在转换结尾时(即最后一个转换后)，会将其内部状态设置为badbit
15     // 因此下一次转换是必须调用clear()将状态重置为goodbit才可以转换
16     // 但是clear()不会将stringstreams底层字符串清空掉
17
18     // s.str("");
19     // 将stringstream底层管理string对象设置成"",
20     // 否则多次转换时，会将结果全部累积在底层string对象中
21
22     s.str("");
23     s.clear(); // 清空s，不清空会转化失败
24     double d = 12.34;
25     s << d;
26     s >> sa;
27
28     string sValue;

```

```
29     sValue = s.str();    // str()方法: 返回stringstream中管理的string类型
30     cout << sValue << endl;
31     return 0;
32 }
```

2. 字符串拼接

```
1  int main()
2  {
3      stringstream sstream;
4
5      // 将多个字符串放入 sstream 中
6      sstream << "first" << " " << "string,";
7      sstream << " second string";
8      cout << "strResult is: " << sstream.str() << endl;
9
10     // 清空 sstream
11     sstream.str("");
12     sstream << "third string";
13     cout << "After clear, strResult is: " << sstream.str() << endl;
14
15     return 0;
16 }
```

注意:

1. `stringstream`实际是在其底层维护了一个`string`类型的对象用来保存结果。
2. 多次数据类型转化时，一定要用`clear()`来清空，才能正确转化，但`clear()`不会将`stringstream`底层的`string`对象清空。
3. 可以使用`s.str("")`方法将底层`string`对象设置为""空字符串。
4. 可以使用`s.str()`将让`stringstream`返回其底层的`string`对象。
5. `stringstream`使用`string`类对象代替字符数组，可以避免缓冲区溢出的危险，而且其会对参数类型进行推演，不需要格式化控制，也不会出现格式化失败的风险，因此使用更方便，更安全。