**C++ 基本的输入输出**

C++ 标准库提供了一组丰富的输入/输出功能，我们将在后续的章节进行介绍。本章将讨论 C++ 编程中最基本和最常见的 I/O 操作。

C++ 的 I/O 发生在流中，流是字节序列。如果字节流是从设备（如键盘、磁盘驱动器、网络连接等）流向内存，这叫做**输入操作**。如果字节流是从内存流向设备（如显示屏、打印机、磁盘驱动器、网络连接等），这叫做**输出操作**。

**I/O 库头文件**

下列的头文件在 C++ 编程中很重要。

|  |  |
| --- | --- |
| **头文件** | **函数和描述** |
| <iostream> | 该文件定义了 **cin、cout、cerr** 和 **clog** 对象，分别对应于标准输入流、标准输出流、非缓冲标准错误流和缓冲标准错误流。 |
| <iomanip> | 该文件通过所谓的参数化的流操纵器（比如 **setw** 和 **setprecision**），来声明对执行标准化 I/O 有用的服务。 |
| <fstream> | 该文件为用户控制的文件处理声明服务。我们将在文件和流的相关章节讨论它的细节。 |

**标准输出流（cout）**

预定义的对象 **cout** 是 **iostream** 类的一个实例。cout 对象"连接"到标准输出设备，通常是显示屏。**cout** 是与流插入运算符 << 结合使用的

C++ 编译器根据要输出变量的数据类型，选择合适的流插入运算符来显示值。<< 运算符被重载来输出内置类型（整型、浮点型、double 型、字符串和指针）的数据项。

流插入运算符 << 在一个语句中可以多次使用，如上面实例中所示，**endl** 用于在行末添加一个换行符。

## 标准输入流（cin）

预定义的对象 **cin** 是 **iostream** 类的一个实例。cin 对象附属到标准输入设备，通常是键盘。**cin** 是与流提取运算符 >> 结合使用的

C++ 编译器根据要输入值的数据类型，选择合适的流提取运算符来提取值，并把它存储在给定的变量中。

流提取运算符 >> 在一个语句中可以多次使用，如果要求输入多个数据，可以使用如下语句：

cin >> name >> age;

**标准错误流（cerr）**

预定义的对象 **cerr** 是 **iostream** 类的一个实例。cerr 对象附属到标准错误设备，通常也是显示屏，但是 **cerr** 对象是非缓冲的，且每个流插入到 cerr 都会立即输出。

**cerr** 也是与流插入运算符 << 结合使用的，如下所示：

**实例**

#include <iostream>

using namespace std;

int main( )

{

char str[] = "Unable to read....";

cerr << "Error message : " << str << endl;

}

当上面的代码被编译和执行时，它会产生下列结果：

Error message : Unable to read....

**标准日志流（clog）**

预定义的对象 **clog** 是 **iostream** 类的一个实例。clog 对象附属到标准错误设备，通常也是显示屏，但是 **clog** 对象是缓冲的。这意味着每个流插入到 clog 都会先存储在缓冲区，直到缓冲填满或者缓冲区刷新时才会输出。

**clog** 也是与流插入运算符 << 结合使用的，如下所示：

**实例**

#include <iostream>

using namespace std;

int main( )

{ char str[] = "Unable to read....";

clog << "Error message : " << str << endl;

}

当上面的代码被编译和执行时，它会产生下列结果：

Error message : Unable to read....

通过这些小实例，我们无法区分 cout、cerr 和 clog 的差异，但在编写和执行大型程序时，它们之间的差异就变得非常明显。所以良好的编程实践告诉我们，使用 cerr 流来显示错误消息，而其他的日志消息则使用 clog 流来输出。

1. 输入输出流中的函数（模板）：

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

    cout<<setiosflags(ios::left|ios::showpoint); // 设左对齐，以一般实数方式显示

    cout.precision(5); // 设置除小数点外有五位有效数字

    cout<<123.456789<<endl;

    cout.width(10); // 设置显示域宽10

    cout.fill('\*'); // 在显示区域空白处用\*填充

    cout<<resetiosflags(ios::left); // 清除状态左对齐

    cout<<setiosflags(ios::right); // 设置右对齐

    cout<<123.456789<<endl;

    cout<<setiosflags(ios::left|ios::fixed); // 设左对齐，以固定小数位显示

    cout.precision(3); // 设置实数显示三位小数

    cout<<999.123456<<endl;

    cout<<resetiosflags(ios::left|ios::fixed); //清除状态左对齐和定点格式

    cout<<setiosflags(ios::left|ios::scientific); //设置左对齐，以科学技术法显示

    cout.precision(3); //设置保留三位小数

    cout<<123.45678<<endl;

    return 0;

}

测试输出结果：

123.46

\*\*\*\*123.46

999.123

1.235e+02

其中 cout.setf 跟 setiosflags 一样，cout.precision 跟 setprecision 一样，cout.unsetf 跟 resetiosflags 一样。

setiosflags(ios::fixed) 固定的浮点显示

setiosflags(ios::scientific) 指数表示

setiosflags(ios::left) 左对齐

setiosflags(ios::right) 右对齐

setiosflags(ios::skipws 忽略前导空白

setiosflags(ios::uppercase) 16进制数大写输出

setiosflags(ios::lowercase) 16进制小写输出

setiosflags(ios::showpoint) 强制显示小数点

setiosflags(ios::showpos) 强制显示符号

cout.setf 常见的标志：

| **标志** | **功能** |
| --- | --- |
| boolalpha | 可以使用单词”true”和”false”进行输入/输出的布尔值. |
| oct | 用八进制格式显示数值. |
| dec | 用十进制格式显示数值. |
| hex | 用十六进制格式显示数值. |
| left | 输出调整为左对齐. |
| right | 输出调整为右对齐. |
| scientific | 用科学记数法显示浮点数. |
| fixed | 用正常的记数方法显示浮点数(与科学计数法相对应). |
| showbase | 输出时显示所有数值的基数. |
| showpoint | 显示小数点和额外的零，即使不需要. |
| showpos | 在非负数值前面显示”＋（正号）”. |
| skipws | 当从一个流进行读取时，跳过空白字符(spaces, tabs, newlines). |
| unitbuf | 在每次插入以后，清空缓冲区. |
| internal | 将填充字符回到符号和数值之间. |
| uppercase | 以大写的形式显示科学记数法中的”e”和十六进制格式的”x”. |

iostream 中定义的操作符：

| **操作符** | **描述** | **输入** | **输出** |
| --- | --- | --- | --- |
| boolalpha | 启用boolalpha标志 | √ | √ |
| dec | 启用dec标志 | √ | √ |
| endl | 输出换行标示，并清空缓冲区 |  | √ |
| ends | 输出空字符 |  | √ |
| fixed | 启用fixed标志 |  | √ |
| flush | 清空流 |  | √ |
| hex | 启用 hex 标志 | √ | √ |
| internal | 启用 internal 标志 |  | √ |
| left | 启用 left 标志 |  | √ |
| noboolalpha | 关闭boolalpha 标志 | √ | √ |
| noshowbase | 关闭showbase 标志 |  | √ |
| noshowpoint | 关闭showpoint 标志 |  | √ |
| noshowpos | 关闭showpos 标志 |  | √ |
| noskipws | 关闭skipws 标志 | √ |  |
| nounitbuf | 关闭unitbuf 标志 |  | √ |
| nouppercase | 关闭uppercase 标志 |  | √ |
| oct | 启用 oct 标志 | √ | √ |
| right | 启用 right 标志 |  | √ |
| scientific | 启用 scientific 标志 |  | √ |
| showbase | 启用 showbase 标志 |  | √ |
| showpoint | 启用 showpoint 标志 |  | √ |
| showpos | 启用 showpos 标志 |  | √ |
| skipws | 启用 skipws 标志 | √ |  |
| unitbuf | 启用 unitbuf 标志 |  | √ |
| uppercase | 启用 uppercase 标志 |  | √ |
| ws | 跳过所有前导空白字符 | √ |  |

iomanip 中定义的操作符：

| **操作符** | **描述** | **输入** | **输出** |
| --- | --- | --- | --- |
| resetiosflags(long f) | 关闭被指定为f的标志 | √ | √ |
| setbase(int base) | 设置数值的基本数为base |  | √ |
| setfill(int ch) | 设置填充字符为ch |  | √ |
| setiosflags(long f) | 启用指定为f的标志 | √ | √ |
| setprecision(int p) | 设置数值的精度(四舍五入) |  | √ |
| setw(int w) | 设置域宽度为w |  | √ |

**C++ 输入多个字符，中间用一个字符隔开。**

好像 C++ 没有像 scanf 控制的那么精确，有个 cin.get() 是可以忽略掉一个字符的，但那个字符可以是任何字符，不限定是逗号。

比如:

cin>>a;cin.get();

cin>>b;cin.get();

cin>>c;