C++ 标准库 < iomanip >

`<iomanip>` 是 C++ 标准库中的一个头文件，它提供了对输入/输出流的格式化操作。

`iomanip` 库中的函数允许开发者控制输出格式，如设置小数点后的位数、设置宽度、对齐方式等。

`iomanip` 是 Input/Output Manipulators 的缩写，它提供了一组操作符，用于控制 C++ 标准库中的输入/输出流的格式。

语法

`iomanip` 库中的函数通常与 `<<` 和 `>>` 操作符一起使用，以实现对输出流的控制。以下是一些常用的 `iomanip` 函数：

- `setw(int)`
- `setprecision(int)`
- `fixed`
- `scientific`
- `setiosflags(ios_base::fmtflags)`
- `resetiosflags(ios_base::fmtflags)`
- `setfill(char)`

常见的cout扩展

```
cout<<boolapha
    //把true和false输出为字符串
cout<<left<<right
    //左对齐、右对齐
cout<<internal
    cout<<setw(10)<<internal<<-3.21<<endl;
    //-      3.21
    //数值的符号在域宽内左对齐，数值右对齐，中间由填充字符填充
cout<<showbase
    //强制输出整数的基数，0/0x
cout<<showpoint
    //强制输出浮点数的小数和尾数（5位）
cout<<uppercase
    //科学计数法E、十六进制字母大写
cout<<showpos
    //对正数显示加号
cout<<unitbuf
    //每次输出之后刷新所有的流（但不可以清空缓冲区）
```

有关缓冲区

```
#include <limits> // 用于 std::numeric_limitsint main() {
 int number;

 while (true) {
```

```
 std::cout << "请输入一个整数：";
 std::cin >> number;

 // 检查输入的状态
 if (std::cin.fail()) {
 std::cerr << "输入无效，请输入一个整数。" << std::endl;

 // 清除错误标志
std::cin.clear(); // 清除错误状态 // 清空缓冲区中无效输入
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
 } else {
 std::cout << "你输入的整数是：" << number << std::endl;
 break; // 输入有效，退出循环 }
 }

 return 0;
}
```

`std::cin.ignore()`

- **功能**：`std::cin.ignore()` 是一个用于忽略输入流中的字符（通常是丢弃缓冲区中未处理的输入），直到遇到指定的字符或达到指定的字符数量。
  ###2. `std::numeric_limits<std::streamsize>::max()`
- **功能**：这一部分代码使用 `std::numeric_limits<std::streamsize>::max()` 来确保我们忽略的字符数没有上限。
- **解释**：`std::numeric_limits<std::streamsize>::max()` 返回 `std::streamsize` 类型可以表示的最大值，通常是一个足够大的数字（例如，表示为 `2147483647`）。这样做是为了确保无论输入的长度是多少，我们都将清空所有可能的输入直到换行符。

`'\n'`

- **功能**：这是 `std::cin.ignore()` 函数的第二个参数，表示我们想要忽略输入流中的字符，直到遇到换行符（即用户按下 Enter 键）。
- **解释**：换行符被用作一个标记，代表用户输入结束。在读取输入时，用户通常通过按下 Enter 键来结束输入，所以我们希望丢弃所有在这之前无效的输入。

完整流程综合来看，`std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');` 这一行代码的目的是：

- **清除无效输入**：如果输入流由于读取错误（比如用户输入了字母而不是数字）而被标记为“失败”，这行代码将跳过直到换行符的所有字符，确保输入缓冲区中的所有无效字符都被清除。
- **准备下一次输入**：通过清空缓冲区，可以在下一次循环中安全地读取新的输入，避免再次遇到之前的无效字符。

小结使用这行代码可以有效防止在读取输入时遇到问题，确保输入状态的正常，避免程序产生错误或进入死循环。这是处理用户输入时的常见和重要的做法，特别是在期望用户按格式输入数据的场合。

其他经典的设置

1. 设置宽度

使用 `setw` 可以设置输出的宽度。如果输出内容的字符数少于设置的宽度，剩余部分将用空格填充。

实例

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

int main() {
  std::cout << std::setw(10) << "Hello" << std::endl;
  return 0;
}
```

输出结果:

```
Hello
```

2. 设置精度

使用 `setprecision` 可以设置浮点数的小数点后的位数。

实例

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

int main() {
  double pi = 3.14159265358979323846;
  std::cout << std::setprecision(2) << pi << std::endl;
  return 0;
}
```

输出结果:

```
3.14
```

3. 固定小数点和科学计数法

`fixed` 和 `scientific` 可以控制浮点数的输出格式。

实例

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

int main() {
  double num = 123456789.0;
  std::cout << "Fixed: " << std::fixed << num << std::endl;
  std::cout << "Scientific: " << std::scientific << num << std::endl;
  return 0;
}
```

输出结果:

```
Fixed: 123456789.000000
Scientific: 1.23456789e+08
```

4. 设置填充字符

使用 `setfill` 可以设置填充字符，通常与 `setw` 一起使用。

实例

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

int main() {
  std::cout << std::setfill('*') << std::setw(10) << "World" << std::endl;
  return 0;
}
```

输出结果:

```
*****World
```

5. 设置和重置格式标志

`setiosflags` 和 `resetiosflags` 可以设置或重置流的格式标志。

`sectiosflags` 内部可以使用的成员： `fixed`,`scientfic`...

`sectiosflag(ios::skips)`:忽略前导空格

实例

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

int main() {
  std::cout << std::setiosflags(std::ios::uppercase) << std::hex << 255 <<
std::endl;
  std::cout << std::resetiosflags(std::ios::uppercase) << std::hex << 255 <<
std::endl;
  return 0;
}
```

输出结果:

```
FF
ff
```

6.进制转换

dec 十进制

hex 十六进制

oct 八进制