

# § . 基础知识题 - C++方式输入输出的格式化控制



要求:

- 1、完成本文档中所有的题目并写出分析、运行结果
- 2、无特殊说明，均使用VS2022编译即可
- 3、直接在本文件上作答，**写出答案/截图（不允许手写、手写拍照截图）**即可；填写答案时，为适应所填内容或贴图，**允许调整**页面的字体大小、颜色、文本框的位置等
  - ★ 贴图要有效部分即可，不需要全部内容
  - ★ 在保证一页一题的前提下，具体页面布局可以自行发挥，简单易读即可
  - ★ **不允许**手写在纸上，再拍照贴图
  - ★ **允许**在各种软件工具上完成（不含手写），再截图贴图
  - ★ 如果某题要求VS+Dev的，则如果两个编译器运行结果一致，贴VS的一张图即可，如果不一致，则两个图都要贴
- 4、转换为pdf后提交
- 5、**3月21日前**网上提交本次作业（在“文档作业”中提交）



## §. 基础知识题 - C++方式输入输出的格式化控制

贴图要求：只需要截取输出窗口中的有效部分即可，如果全部截取/截取过大，则视为无效贴图

例：无效贴图

```
Microsoft Visual Studio 调试控制台  
Hello, world!  
D:\Workspace\VS2019-Demo\Debug\cpp-demo.exe (进程 7484)已退出，代码为 0。  
按任意键关闭此窗口...
```

例：有效贴图

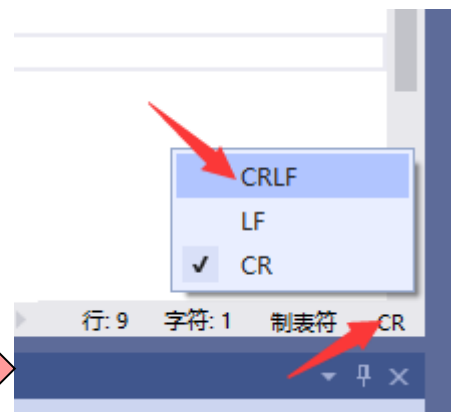
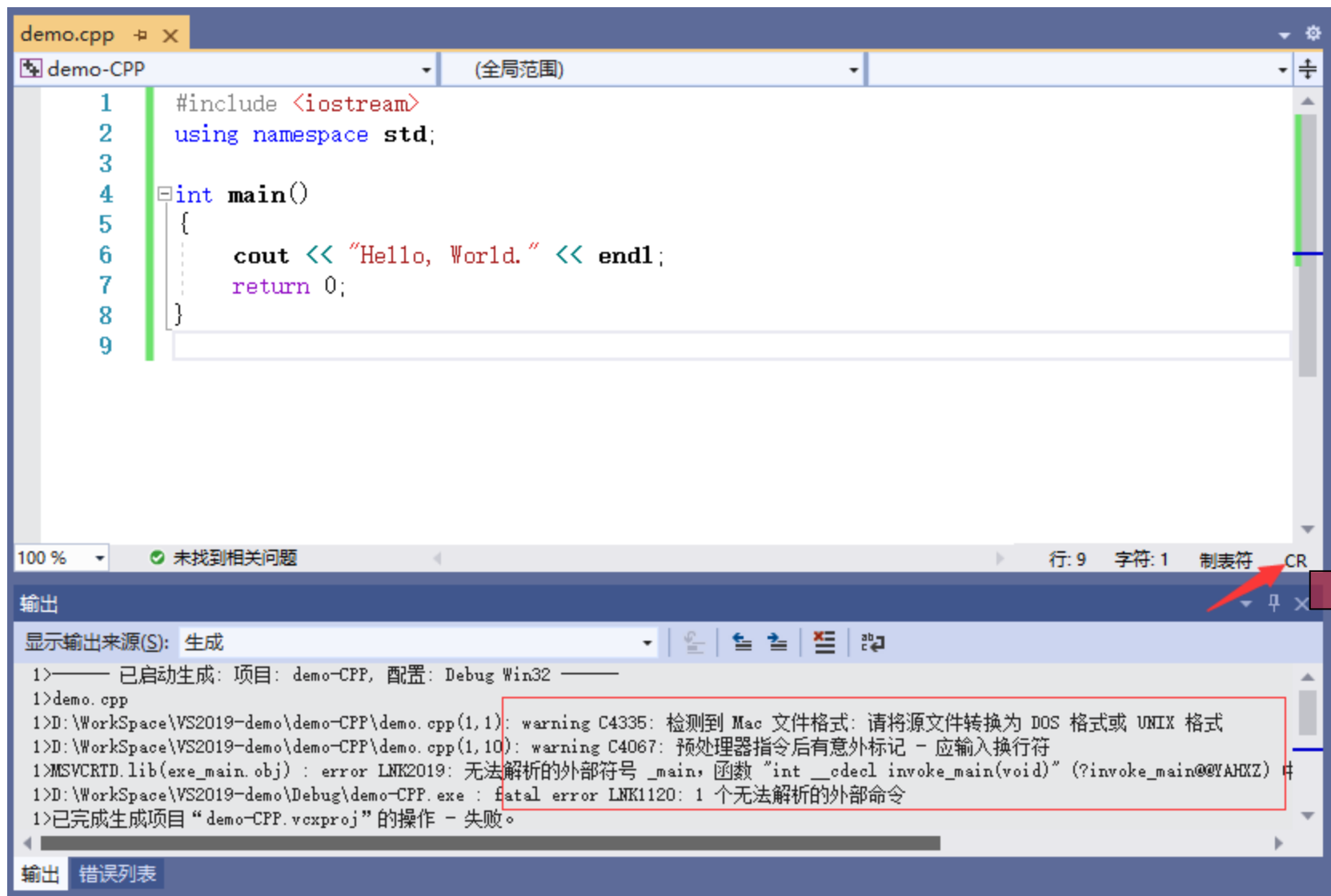
```
Microsoft Visual Studio 调试控制台  
Hello, world!
```



## §. 基础知识题 - C++方式输入输出的格式化控制

附：用WPS等其他第三方软件打开PPT，将代码复制到VS2022中后，如果出现类似下面的**编译报错**，则观察源程序编辑窗

的右下角是否为CR，如果是，单击CR，在弹出中选择CRLF，再次CTRL+F5运行即可



## §. 基础知识题 - C++方式输入输出的格式化控制



特别提示:

- 1、做题过程中，先按要求输入，如果想替换数据，也要先做完指定输入
- 2、如果替换数据后出现某些问题，先记录下来，不要问，等全部完成后，还想不通再问(也许你的问题在后面的题目中有答案)
- 3、不要偷懒、不要自以为是的脑补结论!!!
- 4、先得到题目要求的小结论，再综合考虑上下题目间关系，得到综合结论
- 5、这些结论，是让你记住的，不是让你完成作业后就忘掉了
- 6、换位思考(从老师角度出发)，这些题的目的是希望掌握什么学习方法？



# § . 基础知识题 - C++方式输入输出的格式化控制

说明：C++中的格式控制很丰富，实现方法也有多种，下表列出的只是常用一部分，用于本次作业

| 控制符                          | 作用  | <div>重要提示：<br/>1、后面作业需要的知识点，除非明确提示自行上网查找，都先在本文档中查找是否有符合要求的设置项<br/>2、不看本页，网上瞎找，然后说作业多的，本课程及本作业不背锅</div> |
|------------------------------|---|--|
| dec                          | 设置整数为10进制   |  |
| hex                          | 设置整数为16进制   |  |
| oct                          | 设置整数为8进制  |  |
| setbase(n)                   | 设置整数为n进制 (n=8, 10, 16)  |  |
| setfill(c)                   | 设置填充字符，c可以是字符常量或字符变量  |  |
| setprecision(n)              | 设置实数的精度为n位。在以一般十进制形式输出时，n代表有效数字。<br>在以fixed(固定小数位)形式和scientific(指数)形式输出时，n为小数位数 |  |
| setw(n)                      | 设置字段宽度为n  |  |
| setiosflags(ios::fixed)      | 设置浮点数以固定的小数位数显示   |  |
| setiosflags(ios::scientific) | 设置浮点数以科学计数法（即指数形式）显示  |  |
| setiosflags(ios::left)       | 输出数据左对齐   |  |
| setiosflags(ios::right)      | 输出数据右对齐   |  |
| setiosflags(ios::skipws)     | 忽略前导的空格   |  |
| setiosflags(ios::uppercase)  | 在以科学计数法输出E和十六进制输出字母X时，以大写表示   |  |
| setiosflags(ios::showpos)    | 输出正数时，给出“+”号  |  |
| resetiosflags(*)             | 终止已设置的输出格式状态，括号内为具体内容(本处用*替代)   |  |



## §. 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

#### A. 进制前导符的使用：回答问题并将程序的运行结果截图贴上(允许多页)

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    short a1 = 1234, a2 = 0x1234, a3 = 01234, a4 = 0b1101001; //常量为各进制
    cout << "dec:" << dec << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << '\n';
    cout << "hex:" << hex << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << '\n';
    cout << "oct:" << oct << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << '\n';
    cout << endl;

    short b1 = -1234, b2 = -0x1234, b3 = -01234, b4 = -0b1101001; //常量为各进制
    cout << "dec:" << dec << b1 << ' ' << b2 << ' ' << b3 << ' ' << b4 << '\n';
    cout << "hex:" << hex << b1 << ' ' << b2 << ' ' << b3 << ' ' << b4 << '\n';
    cout << "oct:" << oct << b1 << ' ' << b2 << ' ' << b3 << ' ' << b4 << '\n';
    cout << endl;

    short c1 = 40000, c2 = 0x9876, c3 = 0171234, c4 = 0b1101010100111100;
    cout << "dec:" << dec << c1 << ' ' << c2 << ' ' << c3 << ' ' << c4 << '\n';
    cout << "hex:" << hex << c1 << ' ' << c2 << ' ' << c3 << ' ' << c4 << '\n';
    cout << "oct:" << oct << c1 << ' ' << c2 << ' ' << c3 << ' ' << c4 << '\n';
    cout << endl;

    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
dec:1234 4660 668 105
hex:4d2 1234 29c 69
oct:2322 11064 1234 151

dec:-1234 -4660 -668 -105
hex:fb2e edcc fd64 ff97
oct:175456 166714 176544 177627

dec:-25536 -26506 -3428 -10948
hex:9c40 9876 f29c d53c
oct:116100 114166 171234 152474
```

```
warning C4309: "初始化": 截断常量值
warning C4309: "初始化": 截断常量值
warning C4309: "初始化": 截断常量值
warning C4309: "初始化": 截断常量值
```

//允许贴图覆盖代码部分



## §. 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

#### A. 总结及结论:

- 1、源程序中的整数，有\_\_4\_\_种不同进制的表示形式
- 2、无论源程序中整型常量表示为何种进制，它的机内存储均为\_\_二进制补码\_\_形式
- 3、如果想使数据输出时使用不同进制，要加\_`dec`, `hex`, `oct`\_等进制前导符
- 4、输出\_\_无\_\_(有/无)二进制前导符
- 5、只有\_\_十\_\_进制有负数形式输出；  
16进制输出负数时，特征是输出该数值二进制补码直接转化为16进制表达；  
8进制输出负数时，特征是输出该数值二进制补码直接转化为8进制表达



## §. 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

B. 进制前导符的连续使用：回答问题并将程序的运行结果截图贴上

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

using namespace std;
int main()
{
    int a = 10;

    cout << a << ' ' << a+1 << ' ' << a+2 << endl;
    cout << hex;
    cout << a << ' ' << a+1 << ' ' << a+2 << endl;
    cout << oct;
    cout << a << ' ' << a+1 << ' ' << a+2 << endl;
    cout << dec;
    cout << a << ' ' << a+1 << ' ' << a+2 << endl;

    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
10 11 12
a b c
12 13 14
10 11 12
```

结论：

dec/hex/oct等进制前导符设置后，对后面的\_\_所有\_\_(仅一个/所有)数据有效，直到用另一个控制符去改变为止





## §. 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

C. setbase的使用：同1. A的形式，按要求自行构造测试程序，回答问题并将程序的运行结果截图贴上(允许多页)

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

using namespace std;
int main()
{
    return 0;
} //允许直接贴构造的程序，不用再输入到这里
```

```
Microsoft Visual Studio 调试控制台
2322 11064 1234 151
4d2 1234 29c 69
1234 4660 668 105

175456 166714 176544 177627
fb2e edcc fd64 ff97
-1234 -4660 -668 -105

1234 8500 668 217
1234 8500 668 217
1234 8500 668 217

64 65 66
144 145 146
```

//构造的程序要求能看出对右侧问题的回答  
//将构造的程序直接贴图上来，左侧不写也可

自行构造若干组测试数据，运行并截图

结论：

- 1、setbase中允许的合法值有\_\_8, 10, 16\_\_
- 2、当setbase中出现非法值时，处理方法是默认按照转换成十进制输出处理
- 3、setbase设置后，对后面的\_\_所有\_\_(仅一个/所有)数据有效，直到用另一个setbase去改变为止

```
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4
5 int main()
6 {
7     short a1 = 1234, a2 = 0x1234, a3 = 01234, a4 = 0b1101001;
8     cout << setbase(8) << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;
9     cout << setbase(16) << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;
10    cout << setbase(10) << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;
11    cout << endl;
12
13    short b1 = -1234, b2 = -0x1234, b3 = -01234, b4 = -0b1101001;
14    cout << setbase(8) << b1 << ' ' << b2 << ' ' << b3 << ' ' << b4 << endl;
15    cout << setbase(16) << b1 << ' ' << b2 << ' ' << b3 << ' ' << b4 << endl;
16    cout << setbase(10) << b1 << ' ' << b2 << ' ' << b3 << ' ' << b4 << endl;
17    cout << endl;
18
19    short c1 = 1234, c2 = 0x2134, c3 = 01234, c4 = 0b11011001;
20    cout << setbase(10) << c1 << ' ' << c2 << ' ' << c3 << ' ' << c4 << endl;
21    cout << setbase(2) << c1 << ' ' << c2 << ' ' << c3 << ' ' << c4 << endl;
22    cout << setbase(6) << c1 << ' ' << c2 << ' ' << c3 << ' ' << c4 << endl;
23    cout << endl;
24
25    int a = 100;
26    cout << setbase(16) << a << ' ' << a + 1 << ' ' << a + 2 << endl;
27    cout << setbase(8) << a << ' ' << a + 1 << ' ' << a + 2 << endl;
28    cout << setbase(10) << a << ' ' << a + 1 << ' ' << a + 2 << endl;
29    return 0;
30 }
31
```



## §. 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

D. ios::uppercase的使用：按要求自行构造测试程序，能对比看出用和不用的差别即可

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

using namespace std;
int main()
{
    return 0;
} //允许直接贴构造的程序，不用再输入到这里
```

Microsoft Visual Studio 调试控制台

```
123456 987654
361100 3611006
1E240 F1206
123456 987654
361100 3611006
1e240 f1206
```

//构造的程序要求能看出对右侧问题的回答  
//将构造的程序直接贴图上来，左侧不写也可

测试程序中的数据类型为int，自行构造若干组测试数据，运行并截图

结论：

- 1、uppercase和\_16\_进制一起使用才能看出效果
- 2、uppercase设置后，对后面的\_\_**所有**\_\_ (仅一个/所有)数据有效
- 3、同一个程序中，设置完uppercase，如果想恢复小写，具体的做法是 利用resetiosflags (ios :: uppercase)终止该格式的输出  
(本小问如果不会，先不要问，先往后做，看后面的题目是否有相似问题可以启发你)

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4  int main()
5  {
6      int a1 = 123456, a2 = 987654;
7      cout << setiosflags(ios::uppercase) << dec << a1 << " " << a2 << endl;
8      cout << setiosflags(ios::uppercase) << oct << a1 << " " << a2 << endl;
9      cout << setiosflags(ios::uppercase) << hex << a1 << " " << a2 << endl;
10     cout << resetiosflags(ios::uppercase) << dec << a1 << " " << a2 << endl;
11     cout << resetiosflags(ios::uppercase) << oct << a1 << " " << a2 << endl;
12     cout << resetiosflags(ios::uppercase) << hex << a1 << " " << a2 << endl;
13     cout << endl;
14     return 0;
15 }
```



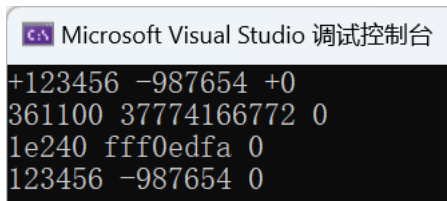
## §. 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

E. ios::showpos的使用：按要求自行构造测试程序，能对比看出用和不用的差别即可

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

using namespace std;
int main()
{
    return 0;
} //允许直接贴构造的程序，不用再输入到这里
```



//构造的程序要求能看出对右侧问题的回答  
//将构造的程序直接贴图上来，左侧不写也可

测试程序中的数据类型为int，自行构造若干组测试数据，运行并截图

结论：

- 1、showpos和\_10\_进制一起使用才能看出效果
- 2、showpos设置后，对后面的\_\_所有\_\_(仅一个/所有)数据有效
- 3、同一个程序中，设置完showpos，如果想取消，具体的做法是利用resetiosflags (ios::showpos)终止该格式的输出 (本小问如果不会，先不要问，先往后做，看后面的题目是否有相似问题可以启发你)

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4  int main()
5  {
6      int a1 = 123456, a2 = -987654, a3 = 0;
7      cout << setiosflags(ios::showpos) << dec << a1 << " " << a2 << " " << a3 << endl;
8      cout << setiosflags(ios::showpos) << oct << a1 << " " << a2 << " " << a3 << endl;
9      cout << setiosflags(ios::showpos) << hex << a1 << " " << a2 << " " << a3 << endl;
10     cout << resetiosflags(ios::showpos) << dec << a1 << " " << a2 << " " << a3 << endl;
11     cout << endl;
12     return 0;
13 }
```



# § . 基础知识题 - C++方式输入输出的格式化控制

## 1、在cout中使用格式化控制符

### F.setprecision的使用 - 单独使用 - (1)

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    float f1 = 1234.5678F;
    float f2 = 8765.4321F;

    /* 第1组: 不设或非法 */
    cout << f1 << ' ' << f2 << endl;
    cout << setprecision(0) << f1 << ' ' << f2 << endl;

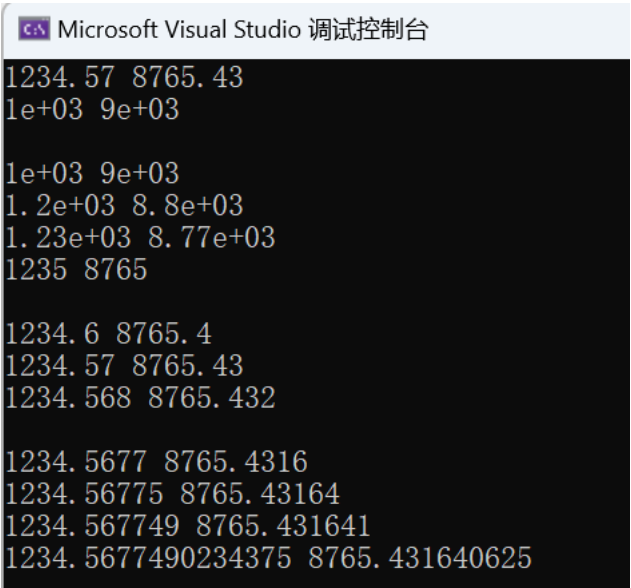
    /* 第2组: 小于等于整数位数 */
    cout << endl;
    cout << setprecision(1) << f1 << ' ' << f2 << endl;
    cout << setprecision(2) << f1 << ' ' << f2 << endl;
    cout << setprecision(3) << f1 << ' ' << f2 << endl;
    cout << setprecision(4) << f1 << ' ' << f2 << endl;

    /* 第3组: 大于整数位数, 但小与等于float型有效数字 */
    cout << endl;
    cout << setprecision(5) << f1 << ' ' << f2 << endl;
    cout << setprecision(6) << f1 << ' ' << f2 << endl;
    cout << setprecision(7) << f1 << ' ' << f2 << endl;

    /* 第4组: 大于float型有效数字 */
    cout << endl;
    cout << setprecision(8) << f1 << ' ' << f2 << endl;
    cout << setprecision(9) << f1 << ' ' << f2 << endl;
    cout << setprecision(10) << f1 << ' ' << f2 << endl;
    cout << setprecision(25) << f1 << ' ' << f2 << endl;

    return 0;
}
```

本例贴图





# §. 基础知识题 - C++方式输入输出的格式化控制

## 1、在cout中使用格式化控制符

### F. setprecision的使用 - 单独使用 - (2)

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    float f1 = 1234567890123456789.0F;
    float f2 = 9876543210987654321.0F;

    /* 第1组: 不设或非法 */
    cout << f1 << ' ' << f2 << endl;
    cout << setprecision(0) << f1 << ' ' << f2 << endl;

    /* 第2组: 小于等于整数位数 并且 小与等于float型有效数字 */
    cout << endl;
    cout << setprecision(1) << f1 << ' ' << f2 << endl;
    cout << setprecision(2) << f1 << ' ' << f2 << endl;
    cout << setprecision(3) << f1 << ' ' << f2 << endl;
    cout << setprecision(4) << f1 << ' ' << f2 << endl;
    cout << setprecision(5) << f1 << ' ' << f2 << endl;
    cout << setprecision(6) << f1 << ' ' << f2 << endl;
    cout << setprecision(7) << f1 << ' ' << f2 << endl;

    /* 第3组: 大于float型有效数字 */
    cout << endl;
    cout << setprecision(8) << f1 << ' ' << f2 << endl;
    cout << setprecision(9) << f1 << ' ' << f2 << endl;
    cout << setprecision(10) << f1 << ' ' << f2 << endl; //为什么f1比f2少一位?
    cout << setprecision(11) << f1 << ' ' << f2 << endl;
    cout << setprecision(25) << f1 << ' ' << f2 << endl;

    return 0;
}
```

因为f1最后一位是0,在该输出格式下不显示,所以比f2少一位。

### 本例贴图

```
Microsoft Visual Studio 调试控制台
1. 23457e+18 9. 87654e+18
1e+18 1e+19

1e+18 1e+19
1. 2e+18 9. 9e+18
1. 23e+18 9. 88e+18
1. 235e+18 9. 877e+18
1. 2346e+18 9. 8765e+18
1. 23457e+18 9. 87654e+18
1. 234568e+18 9. 876544e+18

1. 2345679e+18 9. 8765435e+18
1. 23456794e+18 9. 87654352e+18
1. 23456794e+18 9. 876543516e+18
1. 2345679396e+18 9. 8765435164e+18
1234567939550609408 9876543516404875264
```



## §. 基础知识题 – C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

#### F. setprecision的使用 – 单独使用 – (3)

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    float f1 = 0.12345678F;
    float f2 = 0.87654321F;

    /* 第1组：不设或非法 */
    cout << f1 << ' ' << f2 << endl;
    cout << setprecision(0) << f1 << ' ' << f2 << endl;

    /* 第2组：小与等于float型有效数字 */
    cout << endl;
    cout << setprecision(1) << f1 << ' ' << f2 << endl;
    cout << setprecision(2) << f1 << ' ' << f2 << endl;
    cout << setprecision(3) << f1 << ' ' << f2 << endl;
    cout << setprecision(4) << f1 << ' ' << f2 << endl;
    cout << setprecision(5) << f1 << ' ' << f2 << endl;
    cout << setprecision(6) << f1 << ' ' << f2 << endl;
    cout << setprecision(7) << f1 << ' ' << f2 << endl;

    /* 第3组：大于float型有效数字 */
    cout << endl;
    cout << setprecision(8) << f1 << ' ' << f2 << endl;
    cout << setprecision(9) << f1 << ' ' << f2 << endl;
    cout << setprecision(10) << f1 << ' ' << f2 << endl;
    cout << setprecision(25) << f1 << ' ' << f2 << endl;

    return 0;
}
```

#### 本例贴图

```
Microsoft Visual Studio 调试控制台
0.123457 0.876543
0.1 0.9
0.1 0.9
0.12 0.88
0.123 0.877
0.1235 0.8765
0.12346 0.87654
0.123457 0.876543
0.1234568 0.8765432
0.12345678 0.87654322
0.123456784 0.876543224
0.1234567836 0.8765432239
0.1234567835927009582519531 0.876543223857879638671875
```





## §. 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

#### F. setprecision的使用 - 单独使用 - 总结

**重要结论：** setprecision指定输出位数后，系统会按指定位数输出，即使指定位数超过数据的有效位数（即：输出数据的某位开始是不可信的，但依然会输出）

1、给出setprecision单独使用时的显示规律总结（如果数据不够，可以再自己构造测试数据）

- ①当setprecision (n) 设置非法时按照setprecision (1) 格式处理，n为大于等于1的整数合法。
- ②当n小于数据整数位数时候，按照科学计数法输出，数据有效数位等于n。
- ③当n等于或者大于数据整数位数了，数据保留n位，四舍五入进位。
- ④当n 大于float型7位有效数字时候，只有前7位可信。
- ⑤当数据为小于一的小数时候，输出n位有效数字，只有小数点后7位可信；当数据整数数位超过7位时候，只有高位的7位数字可信。

2、将1. F-(1)~(3)中的数据类型换为double型（有效位数为15位），自行构造测试数据，验证总结出的float型数据的显示规律是否同样适用于double型（如果适用，不用贴图，如果不适用，贴对应代码及运行截图）

同样适用于double型（有效位数为15位）。



## § . 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

#### G. setprecision的使用 - 和ios::fixed一起 - (1)

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

using namespace std;
int main()
{
    float f1 = 1234.5678F;
    float f2 = 8765.4321F;

    /* 第1组: 不设precision */
    cout << f1 << ' ' << f2 << endl;
    cout << setiosflags(ios::fixed) << f1 << ' ' << f2 << endl;

    /* 第2组: 设置precision */
    cout << endl;
    cout << setprecision(1) << f1 << ' ' << f2 << endl;
    cout << setprecision(4) << f1 << ' ' << f2 << endl;
    cout << setprecision(7) << f1 << ' ' << f2 << endl;
    cout << setprecision(10) << f1 << ' ' << f2 << endl;
    cout << setprecision(25) << f1 << ' ' << f2 << endl;

    return 0;
}
```

贴图:

Microsoft Visual Studio 调试控制台

```
1234.57 8765.43
1234.567749 8765.431641

1234.6 8765.4
1234.5677 8765.4316
1234.5677490 8765.4316406
1234.5677490234 8765.4316406250
1234.56774902343750000000000000 8765.43164062500000000000000000
```





## §. 基础知识题 - C++方式输入输出的格式化控制

## 1、在cout中使用格式化控制符

## G. setprecision的使用 - 和ios::fixed一起 - (2)

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

using namespace std;

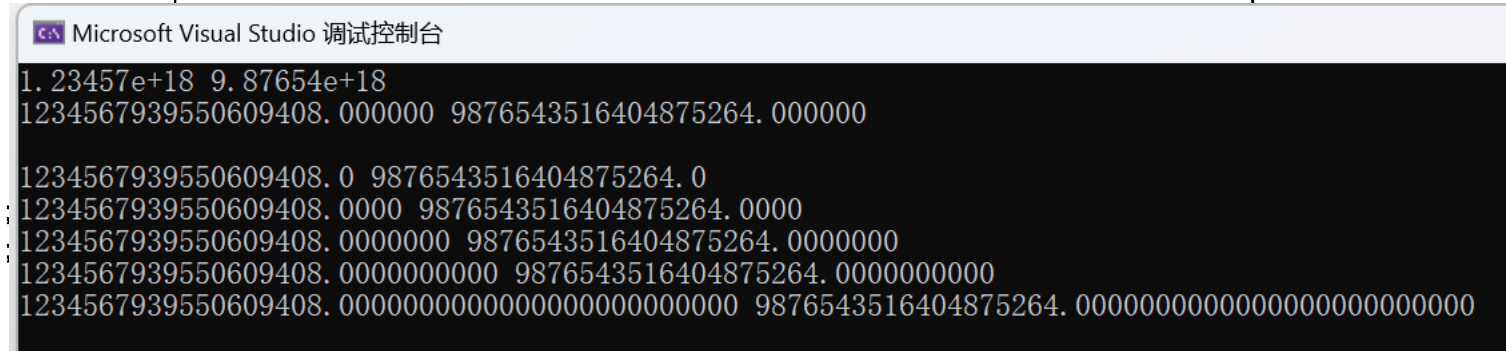
int main()
{
    float f1 = 1234567890123456789.0F;
    float f2 = 9876543210987654321.0F;

    /* 第1组: 不设precision */
    cout << f1 << ' ' << f2 << endl;
    cout << setiosflags(ios::fixed) << f1 << ' ' << f2 << endl;

    /* 第2组: 设置precision */
    cout << endl;
    cout << setprecision(1) << f1 << ' ' << f2 << endl;
    cout << setprecision(4) << f1 << ' ' << f2 << endl;
    cout << setprecision(7) << f1 << ' ' << f2 << endl;
    cout << setprecision(10) << f1 << ' ' << f2 << endl;
    cout << setprecision(25) << f1 << ' ' << f2 << endl;

    return 0;
}
```

贴图:





## § . 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

#### G.setprecision的使用 - 和ios::fixed一起 - (3)

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

using namespace std;
int main()
{
    float f1 = 0.12345678F;
    float f2 = 0.87654321F;

    /* 第1组: 不设precision */
    cout << f1 << ' ' << f2 << endl;
    cout << setiosflags(ios::fixed) << f1 << ' ' << f2 << endl;

    /* 第2组: 设置precision */
    cout << endl;
    cout << setprecision(1) << f1 << ' ' << f2 << endl;
    cout << setprecision(4) << f1 << ' ' << f2 << endl;
    cout << setprecision(7) << f1 << ' ' << f2 << endl;
    cout << setprecision(10) << f1 << ' ' << f2 << endl;
    cout << setprecision(25) << f1 << ' ' << f2 << endl;

    return 0;
}
```

贴图:

选择 Microsoft Visual Studio 调试控制台

```
0.123457 0.876543
0.123457 0.876543

0.1 0.9
0.1235 0.8765
0.1234568 0.8765432
0.1234567836 0.8765432239
0.1234567835927009582519531 0.8765432238578796386718750
```



## § . 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

#### G.setprecision的使用 - 和ios::fixed一起 - 总结

1、给出setprecision+ios::fixed使用时的显示规律总结（如果数据不够，可以再自己构造测试数据）

①ios::fixed之后再setprecision (n)，其中n为小数位数，不是有效数字位数，若之后没有setprecision，默认小数位数为6位，若二者均不加，默认整数和小数共保留6位。

②数据小数部分为0的时候，使用该格式输出，在小数部分补n个0。

③数据小数部分不为0的时候，如果小数部分位数小于n，则补位数到n位（其中整数和小数整体的前7位为可信的，后面的位数不可信）；如果小数位数大于n，则小数部分四舍五入保留n位。

④当数据整数部分数位大于7位，只有高位前7位可信，后面的整数部分为不可信位数。

⑤当数据小数部分不为0，且整数和小数相加多于7位，如果小数部分位数小于n，则只有从高到低前7位可信，后边补不可信位数到n位为止；如果小数部分位数大于n，则也只有从高到低前7位可信，后面跟不可信位数到n位。

2、将1. G-(1)~(3)中的数据类型换为double型（有效位数为15位），自行构造测试数据，验证总结出的float型数据的显示规律是否同样适用于double型（如果适用，不用贴图，如果不适用，贴对应代码及运行截图）

适用于double型（有效位数为15位）。



## §. 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

#### H. setprecision的使用 - 和ios::scientific一起 - (1)

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

using namespace std;
int main()
{
    float f1 = 1234.5678F;
    float f2 = 8765.4321F;

    /* 第1组: 不设precision */
    cout << f1 << ' ' << f2 << endl;
    cout << setiosflags(ios::scientific) << f1 << ' ' << f2 << endl;

    /* 第2组: 设置precision */
    cout << endl;
    cout << setprecision(1) << f1 << ' ' << f2 << endl;
    cout << setprecision(4) << f1 << ' ' << f2 << endl;
    cout << setprecision(7) << f1 << ' ' << f2 << endl;
    cout << setprecision(10) << f1 << ' ' << f2 << endl;
    cout << setprecision(25) << f1 << ' ' << f2 << endl;

    return 0;
}
```

贴图:

```
Microsoft Visual Studio 调试控制台
1234.57 8765.43
1.234568e+03 8.765432e+03

1.2e+03 8.8e+03
1.2346e+03 8.7654e+03
1.2345677e+03 8.7654316e+03
1.2345677490e+03 8.7654316406e+03
1.23456774902343750000000000e+03 8.765431640625000000000000e+03
```



## §. 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

#### H. setprecision的使用 - 和ios::scientific一起 - (2)

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

using namespace std;
int main()
{
    float f1 = 1234567890123456789.0F;
    float f2 = 9876543210987654321.0F;

    /* 第1组: 不设precision */
    cout << f1 << ' ' << f2 << endl;
    cout << setiosflags(ios::scientific) << f1 << ' ' << f2 << endl;

    /* 第2组: 设置precision */
    cout << endl;
    cout << setprecision(1) << f1 << ' ' << f2 << endl;
    cout << setprecision(4) << f1 << ' ' << f2 << endl;
    cout << setprecision(7) << f1 << ' ' << f2 << endl;
    cout << setprecision(10) << f1 << ' ' << f2 << endl;
    cout << setprecision(25) << f1 << ' ' << f2 << endl;

    return 0;
}
```

贴图:

Microsoft Visual Studio 调试控制台

```
1. 23457e+18 9. 87654e+18
1. 234568e+18 9. 876544e+18

1. 2e+18 9. 9e+18
1. 2346e+18 9. 8765e+18
1. 2345679e+18 9. 8765435e+18
1. 2345679396e+18 9. 8765435164e+18
1. 2345679395506094080000000e+18 9. 8765435164048752640000000e+18
```



## §. 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

#### H. setprecision的使用 - 和ios::scientific一起 - (3)

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

using namespace std;
int main()
{
    float f1 = 0.12345678F;
    float f2 = 0.87654321F;

    /* 第1组: 不设precision */
    cout << f1 << ' ' << f2 << endl;
    cout << setiosflags(ios::scientific) << f1 << ' ' << f2 << endl;

    /* 第2组: 设置precision */
    cout << endl;
    cout << setprecision(1) << f1 << ' ' << f2 << endl;
    cout << setprecision(4) << f1 << ' ' << f2 << endl;
    cout << setprecision(7) << f1 << ' ' << f2 << endl;
    cout << setprecision(10) << f1 << ' ' << f2 << endl;
    cout << setprecision(25) << f1 << ' ' << f2 << endl;

    return 0;
}
```

贴图:

Microsoft Visual Studio 调试控制台

```
0.123457 0.876543
1.234568e-01 8.765432e-01

1.2e-01 8.8e-01
1.2346e-01 8.7654e-01
1.2345678e-01 8.7654322e-01
1.2345678359e-01 8.7654322386e-01
1.2345678359270095825195312e-01 8.7654322385787963867187500e-01
```



## §. 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

#### H. setprecision的使用 - 和ios::scientific一起 - 总结

1、给出setprecision+ios::scientific使用时的显示规律总结（如果数据不够，可以再自己构造测试数据）

① ios::scientific之后再setprecision (n)，其中n为科学计数法小数位数，不是有效数字位数，若之后没有setprecision，默认科学计数法尾数小数位数为6位，若二者均不加，默认整数和小数共保留6位。

②数据小数部分为0的时候，使用该格式输出，在小数部分补n个0。

③数据小数部分不为0的时候，如果小数部分位数小于n，则补位数到n位（其中整数和小数整体的前7位为可信的，后面的位数不可信）；如果小数位数大于n，则小数部分四舍五入保留n位。

④当数据整数部分数位大于7位，只有高位前7位可信，后面的位数为不可信位数。

⑤当数据小数部分不为0，且整数和小数相加多于7位，如果小数部分位数小于n，则只有从高到低前7位可信，后边补不可信位数到n位为止；如果小数部分位数大于n，则也只有从高到低前7位可信，后面跟不可信位数到n位。

2、将1. H-(1)~(3)中的数据类型换为double型（有效位数为15位），自行构造测试数据，验证总结出的float型数据的显示规律是否同样适用于double型（如果适用，不用贴图，如果不适用，贴对应代码及运行截图）

适用于double型（有效位数为15位）。

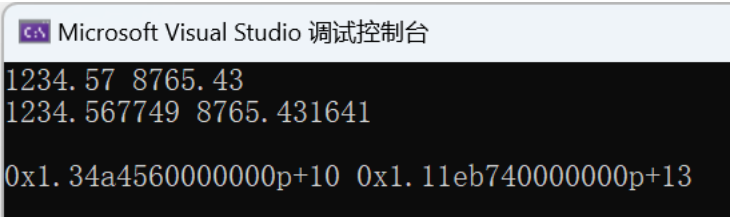
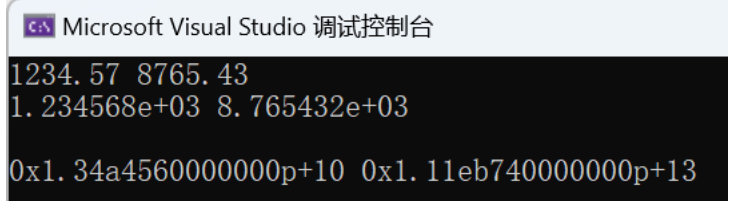




# § . 基础知识题 - C++方式输入输出的格式化控制

## 1、在cout中使用格式化控制符

### I. ios::fixed和ios::scientific的混合使用 - 错误用法

|   |   |
|---|---|
| <pre>#include &lt;iostream&gt; #include &lt;iomanip&gt;  using namespace std; int main() {     float f1 = 1234.5678F, f2 = 8765.4321F;      /* 第1组 */     cout &lt;&lt; f1 &lt;&lt; ' ' &lt;&lt; f2 &lt;&lt; endl;     cout &lt;&lt; setiosflags(ios::fixed) &lt;&lt; f1 &lt;&lt; ' ' &lt;&lt; f2 &lt;&lt; endl;      /* 第2组 */     cout &lt;&lt; endl;     cout &lt;&lt; setiosflags(ios::scientific) &lt;&lt; f1 &lt;&lt; ' ' &lt;&lt; f2 &lt;&lt; endl;      return 0; }</pre> | <pre>#include &lt;iostream&gt; #include &lt;iomanip&gt;  using namespace std; int main() {     float f1 = 1234.5678F, f2 = 8765.4321F;      /* 第1组 */     cout &lt;&lt; f1 &lt;&lt; ' ' &lt;&lt; f2 &lt;&lt; endl;     cout &lt;&lt; setiosflags(ios::scientific) &lt;&lt; f1 &lt;&lt; ' ' &lt;&lt; f2 &lt;&lt; endl;      /* 第2组 */     cout &lt;&lt; endl;     cout &lt;&lt; setiosflags(ios::fixed) &lt;&lt; f1 &lt;&lt; ' ' &lt;&lt; f2 &lt;&lt; endl;      return 0; }</pre> |
| <p>运行截图:</p>   | <p>运行截图:</p>   |

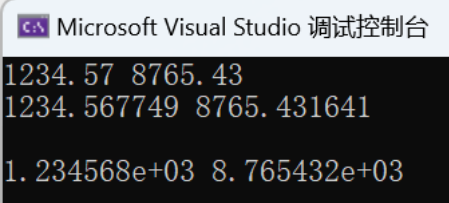
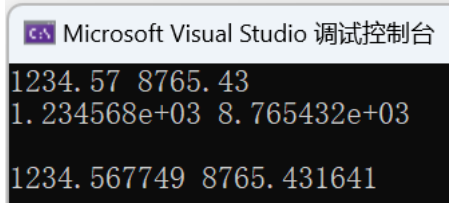




# § . 基础知识题 - C++方式输入输出的格式化控制

## 1、在cout中使用格式化控制符

I. ios::fixed和ios::scientific的混合使用 - 在上一页的基础上将程序改正确，并给出截图

|  |   |
|--|---|
| <pre>#include &lt;iostream&gt; #include &lt;iomanip&gt;  using namespace std; int main() {     float f1 = 1234.5678F, f2 = 8765.4321F;      /* 第1组 */     cout &lt;&lt; f1 &lt;&lt; ' ' &lt;&lt; f2 &lt;&lt; endl;     cout &lt;&lt; setiosflags(ios::fixed) &lt;&lt; f1 &lt;&lt; ' ' &lt;&lt; f2 &lt;&lt; endl;      cout &lt;&lt; resetiosflags(ios::fixed);     /* 第2组 */     cout &lt;&lt; endl;     cout &lt;&lt; setiosflags(ios::scientific) &lt;&lt; f1 &lt;&lt; ' ' &lt;&lt; f2 &lt;&lt; endl;      return 0; }</pre> | <pre>#include &lt;iostream&gt; #include &lt;iomanip&gt;  using namespace std; int main() {     float f1 = 1234.5678F, f2 = 8765.4321F;      /* 第1组 */     cout &lt;&lt; f1 &lt;&lt; ' ' &lt;&lt; f2 &lt;&lt; endl;     cout &lt;&lt; setiosflags(ios::scientific) &lt;&lt; f1 &lt;&lt; ' ' &lt;&lt; f2 &lt;&lt; endl;      cout &lt;&lt; resetiosflags(ios::scientific);     /* 第2组 */     cout &lt;&lt; endl;     cout &lt;&lt; setiosflags(ios::fixed) &lt;&lt; f1 &lt;&lt; ' ' &lt;&lt; f2 &lt;&lt; endl;      return 0; }</pre> |
| <p>运行截图:</p>   | <p>运行截图:</p>    |

结论: (再强调一遍, 先去读P. 5, 后续不再提示)  
如果想要在一个程序中同时显示fixed和scientific形式, 需要在两者之间加入一句:  
左侧程序加cout << resetiosflags(ios::fixed); 右侧程序加cout << resetiosflags(ios::scientific);



# § . 基础知识题 - C++方式输入输出的格式化控制

## 1、在cout中使用格式化控制符

### J. setw的基本使用 - (1)

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

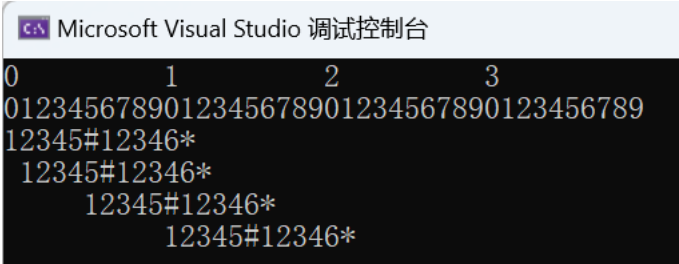
using namespace std;
int main()
{
    int a = 12345;

    cout << "0          1          2          3" << endl;
    cout << "0123456789012345678901234567890123456789" << endl;

    cout << setw(3) << a << '#' << a + 1 << '*' << endl;
    cout << setw(6) << a << '#' << a + 1 << '*' << endl;
    cout << setw(10) << a << '#' << a + 1 << '*' << endl;
    cout << setw(15) << a << '#' << a + 1 << '*' << endl;

    return 0;
}
```

运行截图：



结论：

- 1、setw指定的宽度是总宽度，当总宽度大于数据宽度时，显示规律为 数据前面补空格以达到设置总宽度；  
当总宽度小于数据宽度时，显示规律为 数据前面没有空格
- 2、setw的设置后，对后面的 仅一个 (仅一个/所有) 数据有效
- 3、程序最前面两行的输出，目的是什么？ 显示宽度大小，方便后续比较及规律总结
- 4、每行输出的最后一个\*，目的是什么？ 将上下两行分隔开来，而且可以确定行末没有空格，标志该行输出结束



## §. 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

#### J.setw的基本使用 - (2)

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

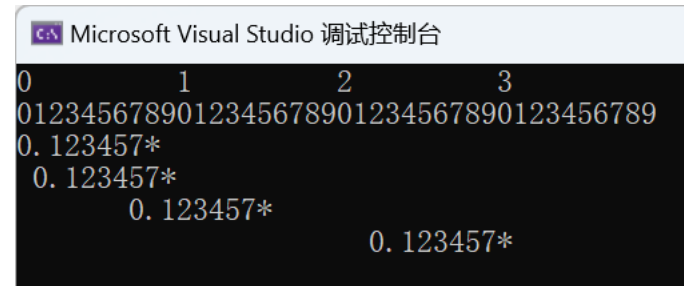
using namespace std;
int main()
{
    double a = 0.123456789012345;

    cout << "0          1          2          3" << endl;
    cout << "0123456789012345678901234567890123456789" << endl;

    cout << setw(6) << a << '*' << endl;
    cout << setw(9) << a << '*' << endl;
    cout << setw(15) << a << '*' << endl;
    cout << setw(30) << a << '*' << endl;

    return 0;
}
```

运行截图:



结论:

1、setw指定的宽度是总宽度，对于实型数据，包含(包含/不包含)小数点



## §. 基础知识题 - C++方式输入输出的格式化控制

### 1、在cout中使用格式化控制符

#### K. setw+setfill的使用

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

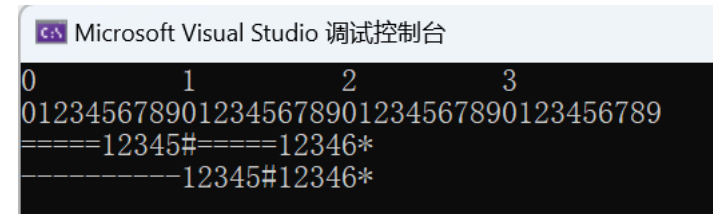
using namespace std;
int main()
{
    int a = 12345;

    cout << "0          1          2          3" << endl;
    cout << "0123456789012345678901234567890123456789" << endl;

    cout << setfill('=') << setw(10) << a << '#' << setw(10) << a + 1 << '*' << endl;
    cout << setw(15) << setfill('-') << a << '#' << a + 1 << '*' << endl;

    return 0;
}
```

运行截图:



结论:

1、setfill的作用是 在总宽度待填充部分填充设定字符

2、setfill的设置后, 对后面的 所有 (仅一个/所有) 数据有效

3、解释为什么第4行的第2个数(12346)前面没有-

setw() 只对一个数据有效, 所以第二个数据不存在待填充区域, 无法填充。



# §. 基础知识题 - C++方式输入输出的格式化控制

## 1、在cout中使用格式化控制符

### L. setw/setfill与ios::left/ios::right的混合使用 - (1)

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

using namespace std;
int main()
{
    int a = 12345;
    cout << "0      1      2      3" << endl;
    cout << "0123456789012345678901234567890123456789" << endl;
    cout << setw(10) << a << '#' << setw(10) << a + 1 << '*' << endl;
    cout << setiosflags(ios::left);
    cout << setw(10) << a << '#' << setw(10) << a + 1 << '*' << endl;
    return 0;
}
```

运行截图:

```
Microsoft Visual Studio 调试控制台
0      1      2      3
0123456789012345678901234567890123456789
12345      #12346      *
```

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

using namespace std;
int main()
{
    int a = 12345;
    cout << "0      1      2      3" << endl;
    cout << "0123456789012345678901234567890123456789" << endl;
    cout << setfill('=') << setw(10) << a << '#' << setw(10) << a + 1 << '*' << endl;
    cout << setiosflags(ios::left);
    cout << setfill('=') << setw(10) << a << '#' << setw(10) << a + 1 << '*' << endl;
    return 0;
}
```

结论:  
1、ios::left的作用是  
    让数据左对齐, 右侧填充到总长度  
2、如果不设置, 缺省是右对齐(左/右对齐)

运行截图:

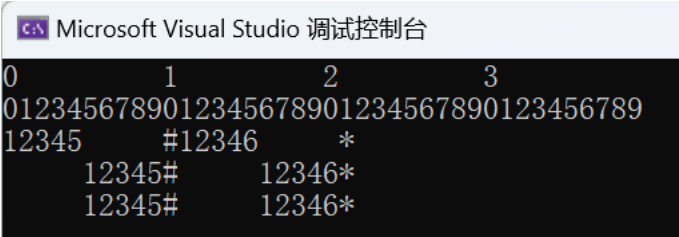
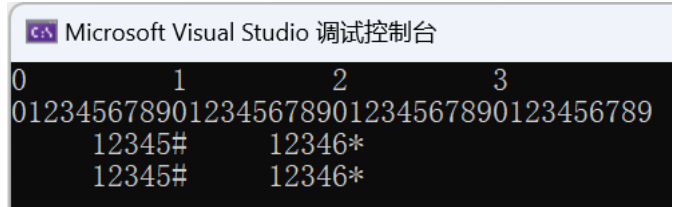
```
Microsoft Visual Studio 调试控制台
0      1      2      3
0123456789012345678901234567890123456789
=====12345#=====12346*
12345=====12346=====*
```



# § . 基础知识题 - C++方式输入输出的格式化控制

## 1、在cout中使用格式化控制符

### L. setw/setfill与ios::left/ios::right的混合使用 - (2) - 同时使用(错误)

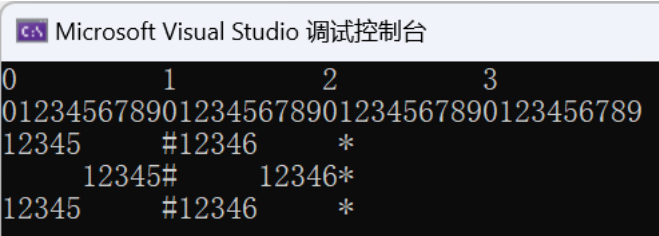
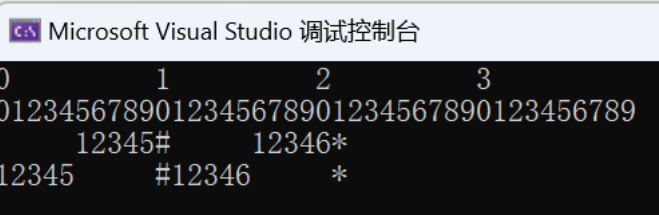
|  |   |
|--|---|
| <pre>#include &lt;iostream&gt; #include &lt;iomanip&gt; using namespace std; int main() {     int a = 12345;     cout &lt;&lt; "0      1      2      3" &lt;&lt; endl;     cout &lt;&lt; "0123456789012345678901234567890123456789" &lt;&lt; endl;     /* 左对齐 */     cout &lt;&lt; setw(10) &lt;&lt; a &lt;&lt; '#' &lt;&lt; setw(10) &lt;&lt; a + 1 &lt;&lt; '*' &lt;&lt; endl;     /* 右对齐 */     cout &lt;&lt; setw(10) &lt;&lt; a &lt;&lt; '#' &lt;&lt; setw(10) &lt;&lt; a + 1 &lt;&lt; '*' &lt;&lt; endl;     /* 左对齐 */     cout &lt;&lt; setw(10) &lt;&lt; a &lt;&lt; '#' &lt;&lt; setw(10) &lt;&lt; a + 1 &lt;&lt; '*' &lt;&lt; endl;     return 0; }</pre> | <p>运行截图:</p>   |
| <pre>#include &lt;iostream&gt; #include &lt;iomanip&gt; using namespace std; int main() {     int a = 12345;     cout &lt;&lt; "0      1      2      3" &lt;&lt; endl;     cout &lt;&lt; "0123456789012345678901234567890123456789" &lt;&lt; endl;     /* 右对齐 */     cout &lt;&lt; setw(10) &lt;&lt; a &lt;&lt; '#' &lt;&lt; setw(10) &lt;&lt; a + 1 &lt;&lt; '*' &lt;&lt; endl;     /* 左对齐 */     cout &lt;&lt; setw(10) &lt;&lt; a &lt;&lt; '#' &lt;&lt; setw(10) &lt;&lt; a + 1 &lt;&lt; '*' &lt;&lt; endl;     return 0; }</pre>   | <p>运行截图:</p>  |



# § . 基础知识题 - C++方式输入输出的格式化控制

## 1、在cout中使用格式化控制符

L. setw/setfill与ios::left/ios::right的混合使用 - 在上一页的基础上将程序改正确，并给出截图

|   |  |
|---|--|
| <pre>#include &lt;iostream&gt; #include &lt;iomanip&gt; using namespace std; int main() {     int a = 12345;     cout &lt;&lt; "0      1      2      3" &lt;&lt; endl;     cout &lt;&lt; "0123456789012345678901234567890123456789" &lt;&lt; endl;     /* 左对齐 */     cout &lt;&lt; setiosflags(ios::left) &lt;&lt; setw(10) &lt;&lt; a &lt;&lt; '#' &lt;&lt; setw(10) &lt;&lt; a + 1 &lt;&lt; '*' &lt;&lt; endl;     cout &lt;&lt; resetiosflags(ios::right);     /* 右对齐 */     cout &lt;&lt; setiosflags(ios::right) &lt;&lt; setw(10) &lt;&lt; a &lt;&lt; '#' &lt;&lt; setw(10) &lt;&lt; a + 1 &lt;&lt; '*' &lt;&lt; endl;     cout &lt;&lt; resetiosflags(ios::right);     /* 左对齐 */     cout &lt;&lt; setiosflags(ios::left) &lt;&lt; setw(10) &lt;&lt; a &lt;&lt; '#' &lt;&lt; setw(10) &lt;&lt; a + 1 &lt;&lt; '*' &lt;&lt; endl;     return 0; }</pre> | <p>运行截图：</p>    |
| <pre>#include &lt;iostream&gt; #include &lt;iomanip&gt; using namespace std; int main() {     int a = 12345;     cout &lt;&lt; "0      1      2      3" &lt;&lt; endl;     cout &lt;&lt; "0123456789012345678901234567890123456789" &lt;&lt; endl;     /* 右对齐 */     cout &lt;&lt; setiosflags(ios::right) &lt;&lt; setw(10) &lt;&lt; a &lt;&lt; '#' &lt;&lt; setw(10) &lt;&lt; a + 1 &lt;&lt; '*' &lt;&lt; endl;     cout &lt;&lt; resetiosflags(ios::right);     /* 左对齐 */     cout &lt;&lt; setiosflags(ios::left) &lt;&lt; setw(10) &lt;&lt; a &lt;&lt; '#' &lt;&lt; setw(10) &lt;&lt; a + 1 &lt;&lt; '*' &lt;&lt; endl;     return 0; }</pre>  | <p>运行截图：</p>  |

结论：  
如果想要right对齐后再left对齐，需要在两者之间加入一句：  
cout << resetiosflags(ios::right);



## §. 基础知识题 - C++方式输入输出的格式化控制

此页不要删除，也没有意义，仅仅为了分隔题目





# § . 基础知识题 - C++方式输入输出的格式化控制

## 2、在cin中使用格式化控制符

### A. 基本要求：从键盘输入16进制数

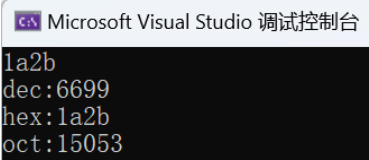
```
#include <iostream>
#include <iomanip>

using namespace std;
int main()
{
    short a;
    cin >> hex >> a;

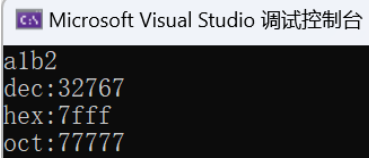
    cout << "dec:" << dec << a << endl;
    cout << "hex:" << hex << a << endl;
    cout << "oct:" << oct << a << endl;

    return 0;
}
```

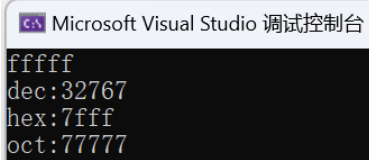
1、输入：1a2b✓（合理正数）



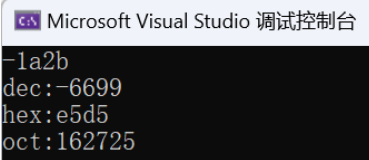
2、输入：a1b2✓（超上限但未超同类型的unsigned上限）



3、输入：ffffff✓（超上限且超过同类型的unsigned上限）



4、输入：-1a2b✓（合理负数）



5、输入：-ffffff✓（超下限）



- 1、贴图即可，不需要写分析结果
- 2、暂不考虑输入错误



# § . 基础知识题 - C++方式输入输出的格式化控制

## 2、在cin中使用格式化控制符

B. 基本要求：从键盘输入8进制数（自行构造测试数据）

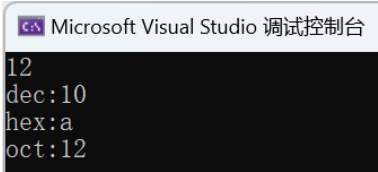
```
#include <iostream>
#include <iomanip>

using namespace std;
int main()
{
    int a;
    cin >> setbase(8) >> a;

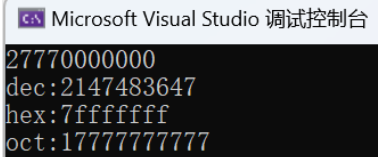
    cout << "dec:" << dec << a << endl;
    cout << "hex:" << hex << a << endl;
    cout << "oct:" << oct << a << endl;

    return 0;
}
```

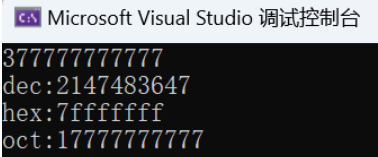
1、输入：\_12\_ ✓ （合理正数）



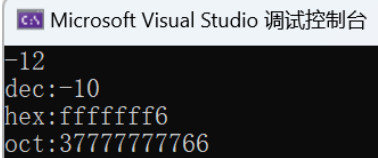
2、输入：\_27770000000\_ ✓ （超上限但未超同类型的unsigned上限）



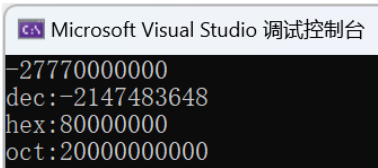
3、输入：\_377777777777\_ ✓ （超上限且超过同类型的unsigned上限）



4、输入：\_-12\_ ✓ （合理负数）



5、输入：\_-27770000000\_ ✓ （超下限）



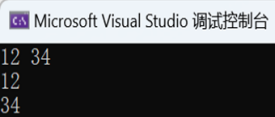
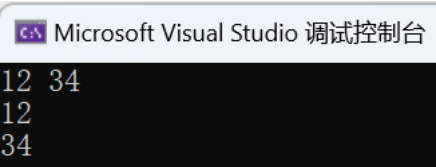
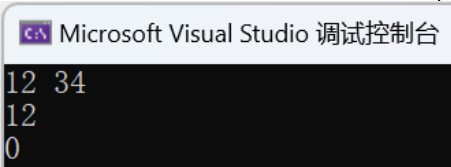
- 1、贴图即可，不需要写分析结果
- 2、暂不考虑输入错误



# § . 基础知识题 - C++方式输入输出的格式化控制

## 2、在cin中使用格式化控制符

### C. 格式控制符setiosflags(ios::skipws)的使用

|  |   |   |
|--|---|---|
| <pre>#include &lt;iostream&gt; using namespace std;  int main() {     int a, b;      cin &gt;&gt; a &gt;&gt; b;      cout &lt;&lt; a &lt;&lt; endl;     cout &lt;&lt; b &lt;&lt; endl;     return 0; }</pre>  | <pre>#include &lt;iostream&gt; #include &lt;iomanip&gt; using namespace std; int main() {     int a, b;     cin &gt;&gt; setiosflags(ios::skipws);     cin &gt;&gt; a &gt;&gt; b;     cout &lt;&lt; a &lt;&lt; endl;     cout &lt;&lt; b &lt;&lt; endl;     return 0; }</pre>  | <pre>#include &lt;iostream&gt; #include &lt;iomanip&gt; using namespace std; int main() {     int a, b;     cin.unsetf(ios::skipws);     cin &gt;&gt; a &gt;&gt; b;     cout &lt;&lt; a &lt;&lt; endl;     cout &lt;&lt; b &lt;&lt; endl;     return 0; }</pre>  |
| 假设键盘输入为: 12 34✓<br>则输出为: 12<br>34  | 假设键盘输入为: 12 34✓<br>则输出为: 12<br>34   | 假设键盘输入为: 12 34✓<br>则输出为: 12<br>0  |

综合以上三个例子可以得到如下结论:

- 1、“忽略前导空格”的意思,是空格不作为\_实际输入\_,而是做为\_输入的分隔\_(因此导致第3个例子b未取得34)
- 2、setiosflags(ios::skipws)在缺省情况下是\_有效\_(有效/无效)的,即不设置也生效
- 3、如果想取消“忽略前导空格”的设置,应使用 `cin.unsetf(ios::skipws);`



## §. 基础知识题 - C++方式输入输出的格式化控制

此页不要删除，也没有意义，仅仅为了分隔题目