



§ . 基础知识题 - C++方式输入输出的格式化控制

要求:

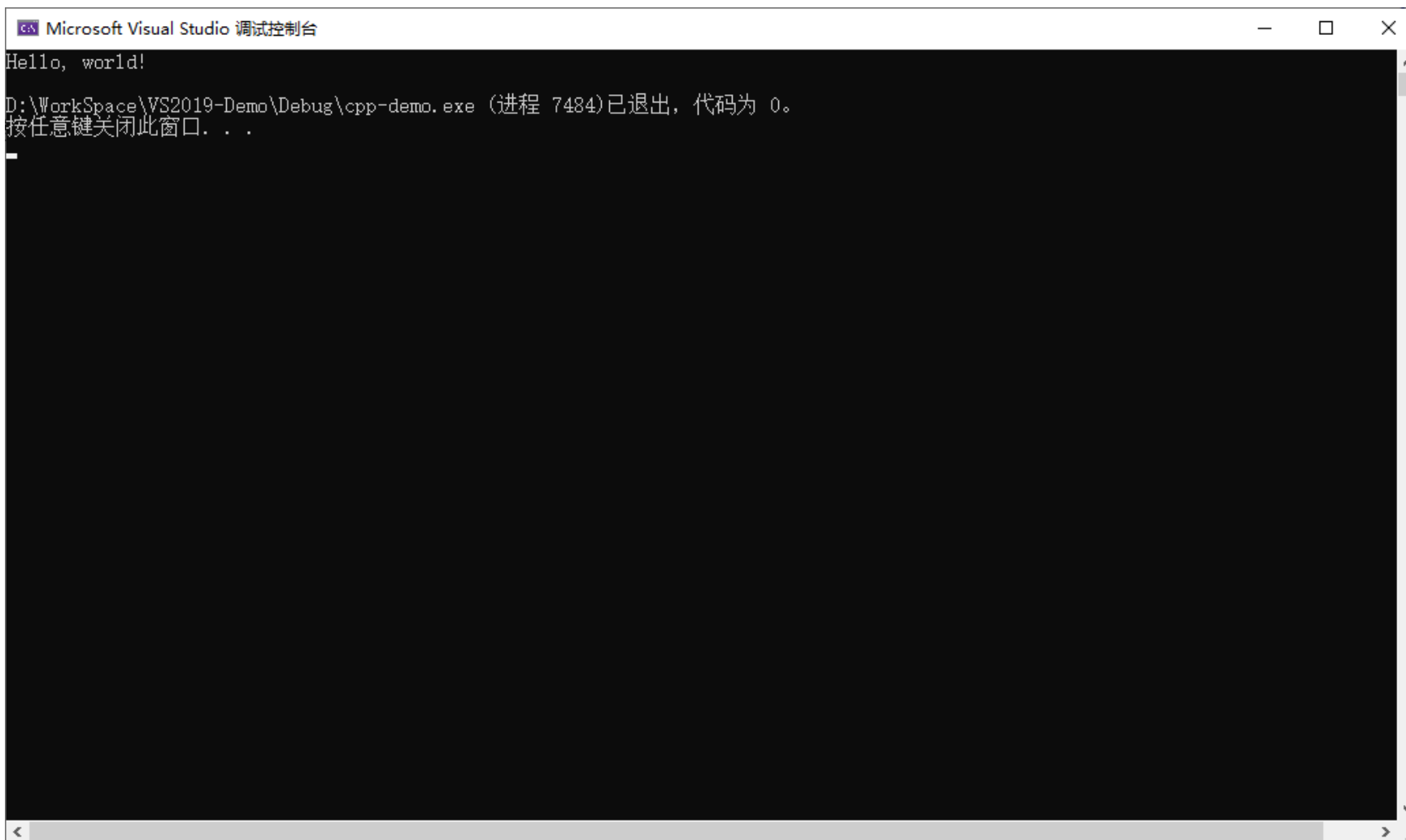
- 1、完成本文档中所有的题目并写出分析、运行结果
- 2、无特殊说明，均使用VS2022编译即可
- 3、直接在本文件上作答，**写出答案/截图（不允许手写、手写拍照截图）**即可；填写答案时，为适应所填内容或贴图，**允许调整**页面的字体大小、颜色、文本框的位置等
 - ★ 贴图要有效部分即可，不需要全部内容
 - ★ 在保证一页一题的前提下，具体页面布局可以自行发挥，简单易读即可
 - ★ **不允许**手写在纸上，再拍照贴图
 - ★ **允许**在各种软件工具上完成（不含手写），再截图贴图
 - ★ 如果某题要求VS+Dev的，则如果两个编译器运行结果一致，贴VS的一张图即可，如果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

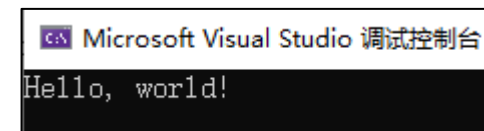


贴图要求：只需要截取输出窗口中的有效部分即可，如果全部截取/截取过大，则视为无效贴图

例：无效贴图

A screenshot of the Microsoft Visual Studio debug console window. The title bar reads 'Microsoft Visual Studio 调试控制台'. The console output shows 'Hello, world!' on the first line, followed by 'D:\Workspace\VS2019-Demo\Debug\cpp-demo.exe (进程 7484)已退出, 代码为 0。' and '按任意键关闭此窗口...' on the next line. The console is mostly empty with a black background and white text.

例：有效贴图

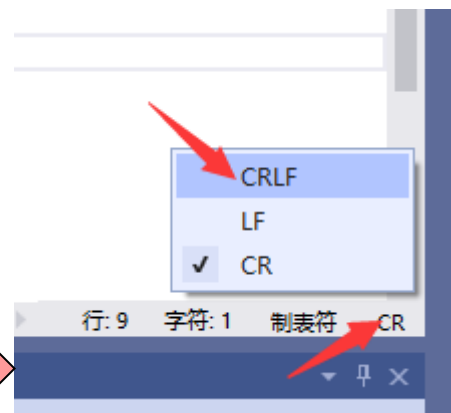
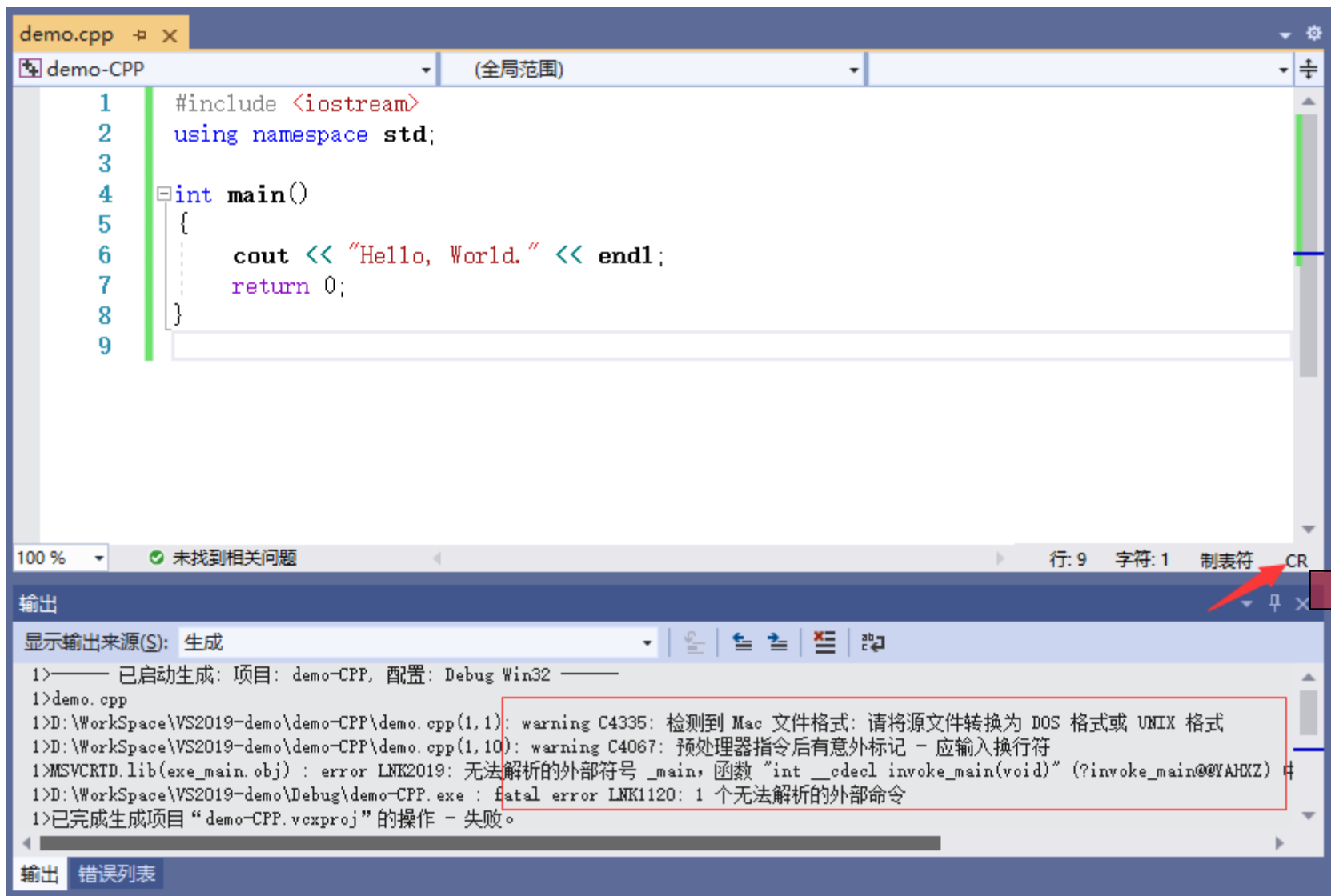
A screenshot of the Microsoft Visual Studio debug console window, showing only the first line of output: 'Hello, world!'. The title bar reads 'Microsoft Visual Studio 调试控制台'.



§. 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： 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 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。 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
    cout << "dec:" << dec << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;
    cout << "hex:" << hex << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;
    cout << "oct:" << oct << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;
    cout << endl;

    short b1 = -1234, b2 = -0x1234, b3 = -01234, b4 = -0b1101001;
    cout << "dec:" << dec << b1 << ' ' << b2 << ' ' << b3 << ' ' << b4 << endl;
    cout << "hex:" << hex << b1 << ' ' << b2 << ' ' << b3 << ' ' << b4 << endl;
    cout << "oct:" << oct << b1 << ' ' << b2 << ' ' << b3 << ' ' << b4 << endl;
    cout << endl;

    short c1 = 40000, c2 = 0x9876, c3 = 0171234, c4 = 0b11010101001;
    cout << "dec:" << dec << c1 << ' ' << c2 << ' ' << c3 << ' ' << c4 << endl;
    cout << "hex:" << hex << c1 << ' ' << c2 << ' ' << c3 << ' ' << c4 << endl;
    cout << "oct:" << oct << c1 << ' ' << c2 << ' ' << c3 << ' ' << c4 << endl;
    cout << endl;

    return 0;
}
```

```
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____2进制____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__10__进制有负数形式输出；
16进制输出负数时，特征是转换为二进制补码形式，再从补码按无符号形式输出16进制数；
8进制输出负数时，特征是转换为二进制补码形式，再从补码按无符号形式输出 8进制数.



§. 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;
}
```

```
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)

```
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; //常量为各进制表示正数
8     cout << "1进制" << setbase(1) << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;
9     cout << "2进制" << setbase(2) << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;
10    cout << "3进制" << setbase(3) << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;
11    cout << "4进制" << setbase(4) << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;
12    cout << "8进制" << setbase(8) << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;
13    cout << "9进制" << setbase(9) << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;
14    cout << "10进制" << setbase(10) << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;
15    cout << "13进制" << setbase(13) << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;
16    cout << "16进制" << setbase(16) << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;
17
18    int a = 10;
19    cout << setbase(8) << a << ' ' << a + 1 << ' ' << a + 2 << endl;
20    cout << setbase(10) << a << ' ' << a + 1 << ' ' << a + 2 << endl;
21    cout << setbase(16) << a << ' ' << a + 1 << ' ' << a + 2 << endl;
22    return 0;
23 }
```

```
1进制1234 4660 668 105
2进制1234 4660 668 105
3进制1234 4660 668 105
4进制1234 4660 668 105
8进制2322 11064 1234 151
9进制1234 4660 668 105
10进制1234 4660 668 105
13进制1234 4660 668 105
16进制4d2 1234 29c 69
12 13 14
10 11 12
a b c
```

//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_输出10进制结果

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



§. 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序，能对比看出用和不用的差别即可

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

int main()
{
    int a = 1000;
    cout << a << ' ' << a + 1 << ' ' << a + 2 << endl;
    cout << setiosflags(ios::uppercase) << hex;
    cout << a << ' ' << a + 1 << ' ' << a + 2 << endl;
    cout << setiosflags(ios::uppercase) << oct;
    cout << a << ' ' << a + 1 << ' ' << a + 2 << endl;
    cout << setiosflags(ios::uppercase) << dec;
    cout << a << ' ' << a + 1 << ' ' << a + 2 << endl;
    cout << resetiosflags(ios::uppercase) << hex;
    cout << a << ' ' << a + 1 << ' ' << a + 2 << endl;
    return 0;
}
```

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__cout << resetiosflags(ios::uppercase)__
(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)

```
1000 1001 1002
3E8 3E9 3EA
1750 1751 1752
1000 1001 1002
3e8 3e9 3ea
```

//构造的程序要求能看出
//将构造的程序直接贴图上来，左侧不写也可



§. 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()
{
    int a = 100;
    cout << a << ' ' << a + 1 << ' ' << a + 2 << endl;
    cout << setiosflags(ios::showpos) << hex;
    cout << a << ' ' << a + 1 << ' ' << a + 2 << endl;
    cout << setiosflags(ios::showpos) << oct;
    cout << a << ' ' << a + 1 << ' ' << a + 2 << endl;
    cout << setiosflags(ios::showpos) << dec;
    cout << a << ' ' << a + 1 << ' ' << a + 2 << endl;
    cout << resetiosflags(ios::showpos) << dec;
    cout << a << ' ' << a + 1 << ' ' << a + 2 << endl;
    return 0;
}
```

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____cout << resetiosflags(ios::showpos)____
(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)

//构造的程序要求能看
//将构造的程序直接贴

```
100 101 102
64 65 66
144 145 146
+100 +101 +102
100 101 102
```



§. 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

```
1234.57 8765.43
1e+03 9e+03

1e+03 9e+03
1.2e+03 8.8e+03
1.23e+03 8.77e+03
1235 8765

1234.6 8765.4
1234.57 8765.43
1234.568 8765.432

1234.5677 8765.4316
1234.56775 8765.43164
1234.567749 8765.431641
1234.5677490234375 8765.431640625
```



§. 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; //在表示不可信数字时, 最后
    cout << setprecision(25) << f1 << ' ' << f2 << endl; //一位随机表示为0, 无输出,
    return 0; //故f1少一位
}
```

本例贴图

```
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

```
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）

对于浮点数，n代表有效数字，即若小数点前整数位有数字（不为0）时，n代表有效数字；若小数点前整数位为0，则n代表小数位数。

在截断位数小于或等于float型有效数字时，若截断位数后还有数字，则此数对前一位进行四舍五入；
在截断位数大于float型有效数字时，系统会按指定位数输出，但有效位数之后的数字均为不可信的。

猜测F（2）中f1比f2少一位的原因：在表示不可信数字时，最后一位随机表示为0，无输出，故f1少一位。

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）

double型同样适用



§. 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:

```
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 8765.431640625000000000000000
```




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

贴图：

```
1. 23457e+18 9. 87654e+18
1234567939550609408. 000000 9876543516404875264. 000000

1234567939550609408. 0 9876543516404875264. 0
1234567939550609408. 0000 9876543516404875264. 0000
1234567939550609408. 0000000 9876543516404875264. 0000000
1234567939550609408. 0000000000 9876543516404875264. 0000000000
1234567939550609408. 000000000000000000000000 9876543516404875264. 000000000000000000000000
```



§ . 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:

```
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）

对于浮点数，在setiosflags(ios::fixed)不设precision时，所表示数整数部分位数与原数相同，不论原数小数部分有多少位，setiosflags(ios::fixed)之后，小数部分都将输出至小数点后6位，缺少数字则补0。

在设置precision后，setprecision(n)内n表示的位数即为小数点后的位数，不足则补0。

同样，在截断位数小于或等于float型有效数字时，若截断位数后还有数字，则此数对前一位进行四舍五入；在截断位数大于float型有效数字时，系统会按指定位数输出，但有效位数之后的数字均为不可信的。

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）

double型同样适用



§. 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:

```
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:

```
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:

```
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）

对于浮点数来说，在setiosflags(ios::scientific)不设precision时，不论原数位数，设置后整数位输出统一为1位，小数位输出统一为6位。

在设置precision之后，setprecision(n)内表示的位数为小数点后的位数，不足则补0。

同样，在截断位数小于或等于float型有效数字时，若截断位数后还有数字，则此数对前一位进行四舍五入；在截断位数大于float型有效数字时，系统会按指定位数输出，但有效位数之后的数字均为不可信的。

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）

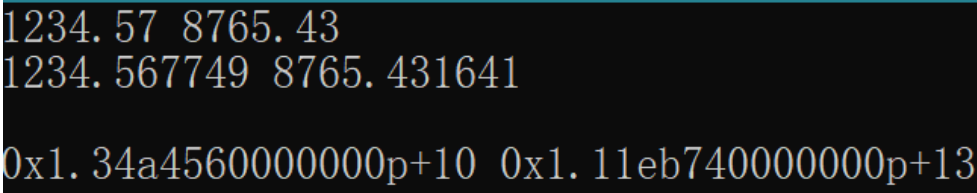
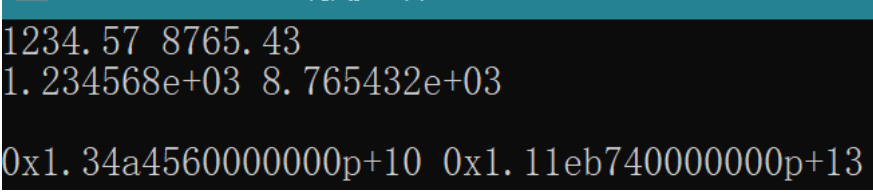
double型同样适用



§ . 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

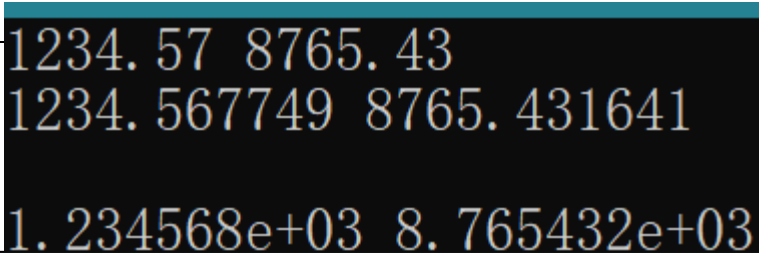
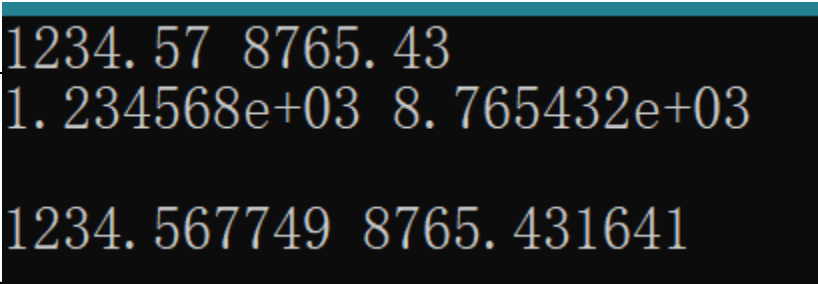
<pre>#include <iostream> #include <iomanip> using namespace std; int main() { float f1 = 1234.5678F, f2 = 8765.4321F; /* 第1组 */ cout << f1 << ' ' << f2 << endl; cout << setiosflags(ios::fixed) << f1 << ' ' << f2 << endl; /* 第2组 */ cout << endl; cout << setiosflags(ios::scientific) << f1 << ' ' << f2 << endl; return 0; }</pre>	<pre>#include <iostream> #include <iomanip> using namespace std; int main() { float f1 = 1234.5678F, f2 = 8765.4321F; /* 第1组 */ cout << f1 << ' ' << f2 << endl; cout << setiosflags(ios::scientific) << f1 << ' ' << f2 << endl; /* 第2组 */ cout << endl; cout << setiosflags(ios::fixed) << f1 << ' ' << f2 << 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 <iostream> #include <iomanip> using namespace std; int main() { float f1 = 1234.5678F, f2 = 8765.4321F; /* 第1组 */ cout << f1 << ' ' << f2 << endl; cout << setiosflags(ios::fixed) << f1 << ' ' << f2 << endl; cout << resetiosflags(ios::fixed) ; /* 第2组 */ cout << endl; cout << setiosflags(ios::scientific) << f1 << ' ' << f2 << endl; return 0; }</pre>	<pre>#include <iostream> #include <iomanip> using namespace std; int main() { float f1 = 1234.5678F, f2 = 8765.4321F; /* 第1组 */ cout << f1 << ' ' << f2 << endl; cout << setiosflags(ios::scientific) << f1 << ' ' << f2 << endl; cout << resetiosflags(ios::scientific) ; /* 第2组 */ cout << endl; cout << setiosflags(ios::fixed) << f1 << ' ' << f2 << endl; return 0; }</pre>
<p>运行截图:</p> 	<p>运行截图:</p> 
<p>结论: (再强调一遍, 先去读P. 5, 后续不再提示)</p> <p>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: <code>_cout<<resetiosflags(ios::fixed);</code> 者 <code>cout<<resetiosflags(ios::scientific)</code> __</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

J.setw的基本使用 - (1)

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

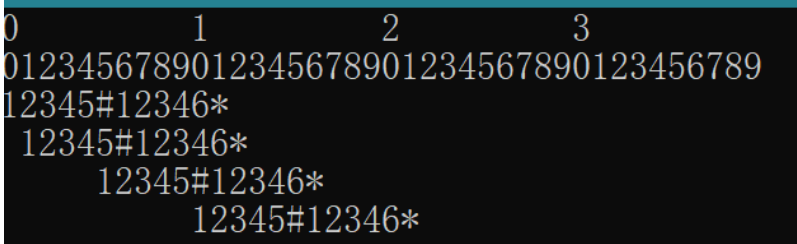
using namespace std;
int main()
{
    int a = 12345;

    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_多余的位置在数据前补充空格；
当总宽度小于数据宽度时，显示规律为_显示数据宽度，无空格补充
- 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？ 方便对齐，便于阅读
- 4、每行输出的最后一个*，目的是什么？ 观察右侧的相差位数，便于比较



§. 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:

```
0          1          2          3
0123456789012345678901234567890123456789
0.123457*
0.123457*
0.123457*
0.123457*
```

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:

```
0          1          2          3
0123456789012345678901234567890123456789
=====12345#=====12346*
-----12345#12346*
```

结论:

1、setfill的作用是_设置填充字符，将多余的空格位填充为设定字符_____

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-

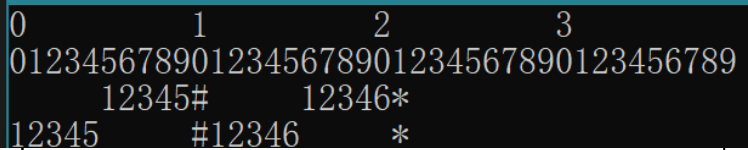
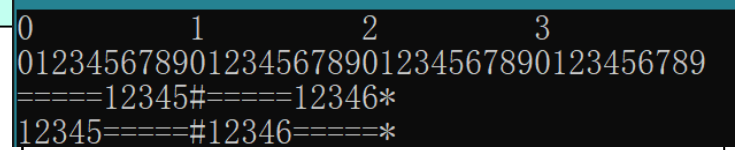
setw只对其后面的一个数据生效，第4行的a+1并没有通过setw设置宽度，所以没有给setfill填充-。



§ . 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)

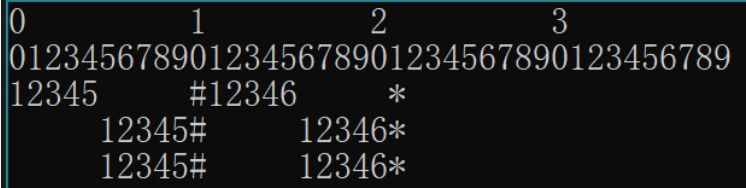
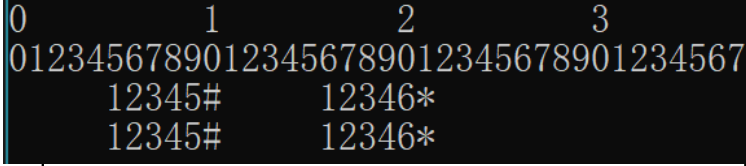
<pre>#include <iostream> #include <iomanip> using namespace std; int main() { int a = 12345; cout << "0 1 2 3" << endl; cout << "0123456789012345678901234567890123456789" << endl; cout << setw(10) << a << '#' << setw(10) << a + 1 << '*' << endl; cout << setiosflags(ios::left); cout << setw(10) << a << '#' << setw(10) << a + 1 << '*' << endl; return 0; }</pre>	<p>运行截图:</p> 
<pre>#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; }</pre>	<p>结论:</p> <p>1、ios::left的作用是__使数据左对齐__</p> <p>2、如果不设置, 缺省是__右对齐__(左/右对齐)</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

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

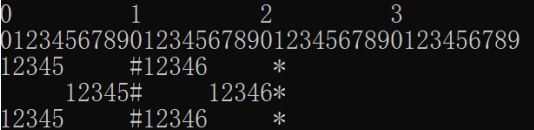
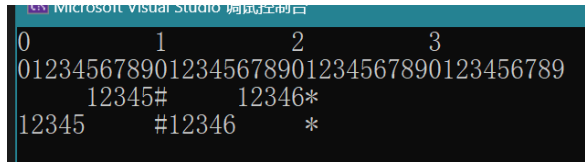
<pre>#include <iostream> #include <iomanip> using namespace std; int main() { int a = 12345; cout << "0 1 2 3" << endl; cout << "0123456789012345678901234567890123456789" << endl; /* 左对齐 */ cout << setiosflags(ios::left) << setw(10) << a << '#' << setw(10) << a + 1 << '*' << endl; /* 右对齐 */ cout << setiosflags(ios::right)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

1a2b
dec:6699
hex:1a2b
oct:15053
- 2、输入：a1b2✓ （超上限但未超同类型的unsigned上限）

a1b2
dec:32767
hex:7fff
oct:77777
- 3、输入：ffffff✓ （超上限且超过同类型的unsigned上限）

ffffff
dec:32767
hex:7fff
oct:77777
- 4、输入：-1a2b✓ （合理负数）

-1a2b
dec:-6699
hex:e5d5
oct:162725
- 5、输入：-ffffff✓ （超下限）

-ffffff
dec:-32768
hex:8000
oct:100000

- 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：__666__ ✓ （合理正数）

666
dec:438
hex:1b6
oct:666

2、输入：_35632624000_ ✓ （超上限但未超同类型的unsigned上限）

35632624000
dec:2147483647
hex:7fffffff
oct:17777777777

3、输入：_45201371000_ ✓ （超上限且超过同类型的unsigned上限）

45201371000
dec:2147483647
hex:7fffffff
oct:17777777777

4、输入：__ -666__ ✓ （合理负数）

-666
dec:-438
hex:fffffe4a
oct:37777777112

5、输入：___ -45201371000_ ✓ （超下限）

-45201371000
dec:-2147483648
hex:80000000
oct:20000000000

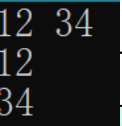
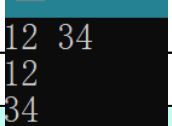
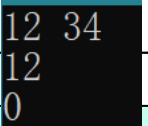
- 1、贴图即可，不需要写分析结果
- 2、暂不考虑输入错误



§ . 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 <iostream> using namespace std; int main() { int a, b; cin >> a >> b; cout << a << endl; cout << b << endl; return 0; }</pre>	<pre>#include <iostream> #include <iomanip> using namespace std; int main() { int a, b; cin >> setiosflags(ios::skipws); cin >> a >> b; cout << a << endl; cout << b << endl; return 0; }</pre>	<pre>#include <iostream> #include <iomanip> using namespace std; int main() { int a, b; cin.unsetf(ios::skipws); cin >> a >> b; cout << a << endl; cout << b << endl; return 0; }</pre>
假设键盘输入为: 12 34✓ 则输出为: 	假设键盘输入为: 12 34✓ 则输出为: 	假设键盘输入为: 12 34✓ 则输出为: 
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_空格符_,而是做为_分隔符_(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