§15.输入输出流 - Windows与Linux的文件格式差别

要求：

1、安装UltraEdit软件（附件已给，版本虽旧，但够用），学会使用16进制方式查看文件，并掌握ASCII及16进制查看间的切换

★ 已安装VSCode的也可通过相关插件进行16进制方式的查看(VSCode某种情况下会自动做格式转换或字符集转换，要注意!!!)

★ 也可以使用其它编辑软件，但不建议NotePad++

2、完成本文档中所有的测试程序并填写运行结果，从而掌握Windows与Linux两个系统下的文本文件的差异

3、题目明确指定编译器外，Windows下用VS2022编译，Linux下用C++编译

★ 如果要换成其他编译器，可能需要自行修改头文件适配

★ 部分代码编译时有warning，不影响概念理解，可以忽略

4、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

★ 如果两个编译器运行结果一致，贴其中一张图即可，如果不一致，则两个图都要贴

5、转换为pdf后提交

6、11月7日前网上提交本次作业（在“文档作业”中提交）

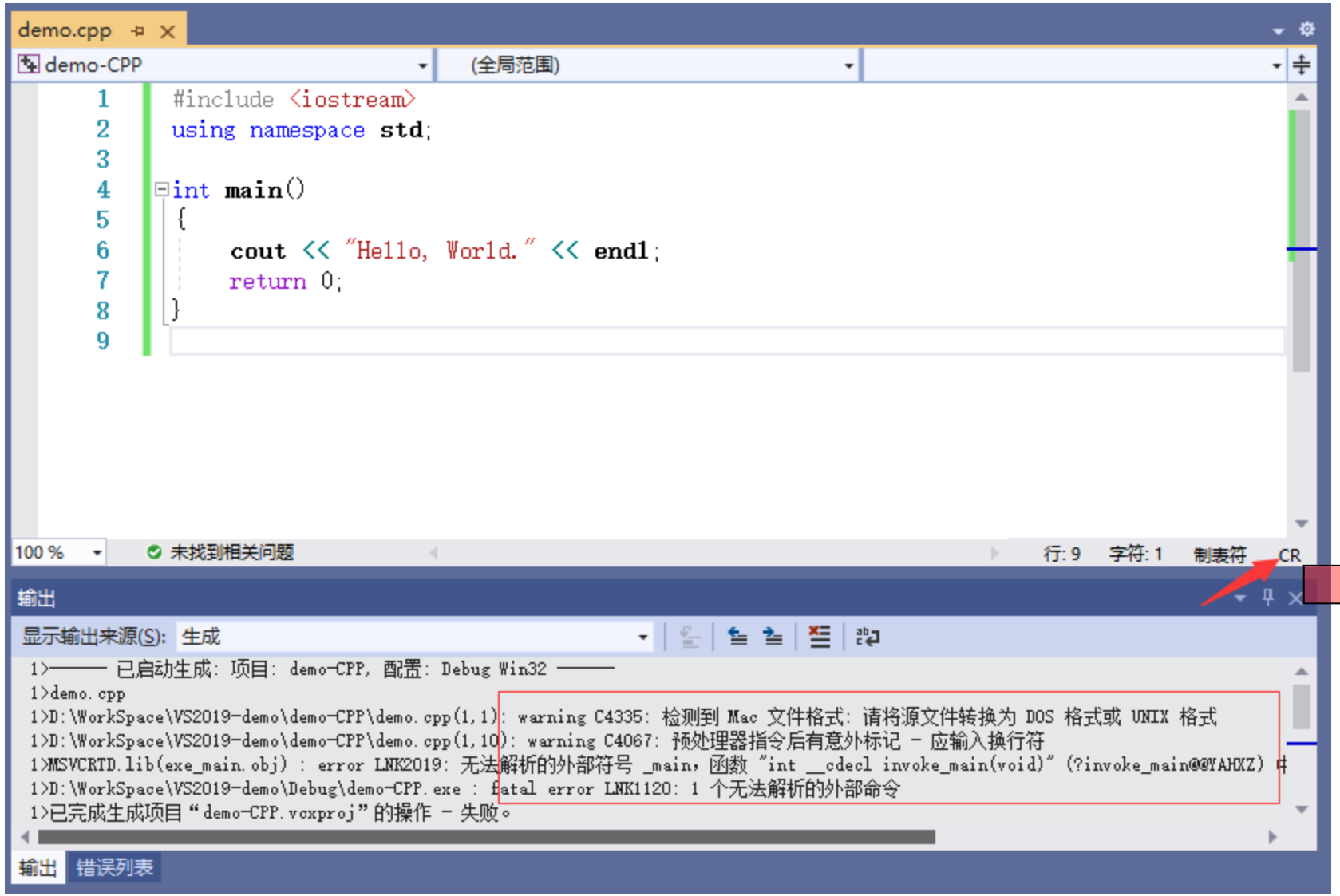
特别说明：

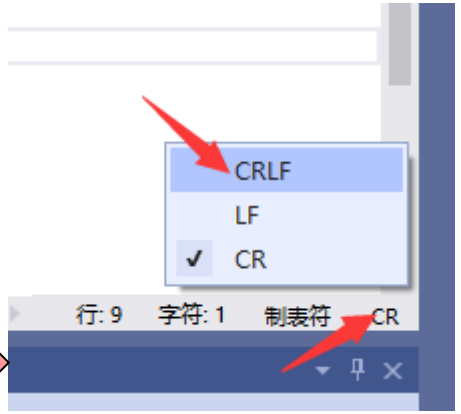
★ 因为篇幅问题，打开文件后均省略了是否打开成功的判断，这在实际应用中是不允许的

§15. 输入输出流 - Windows与Linux的文件格式差别

注意：

附1：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

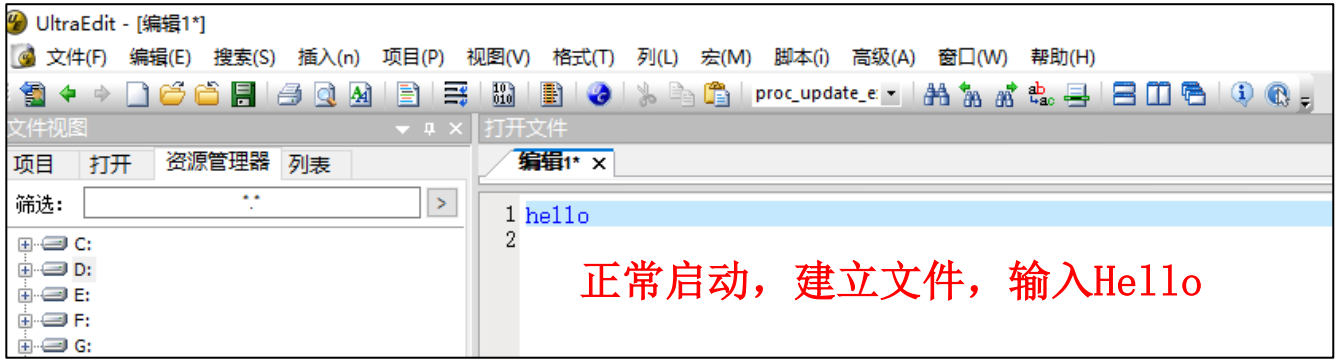




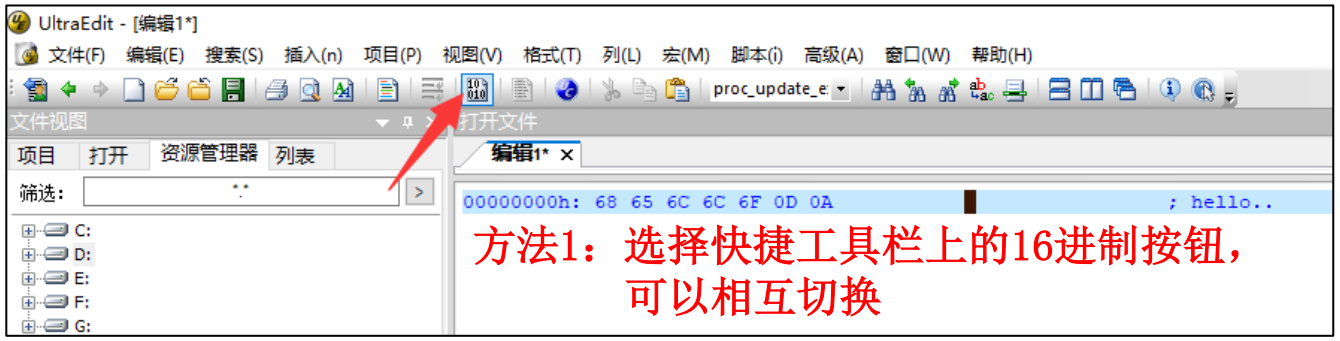
§15. 输入输出流 - Windows与Linux的文件格式差别

注意：

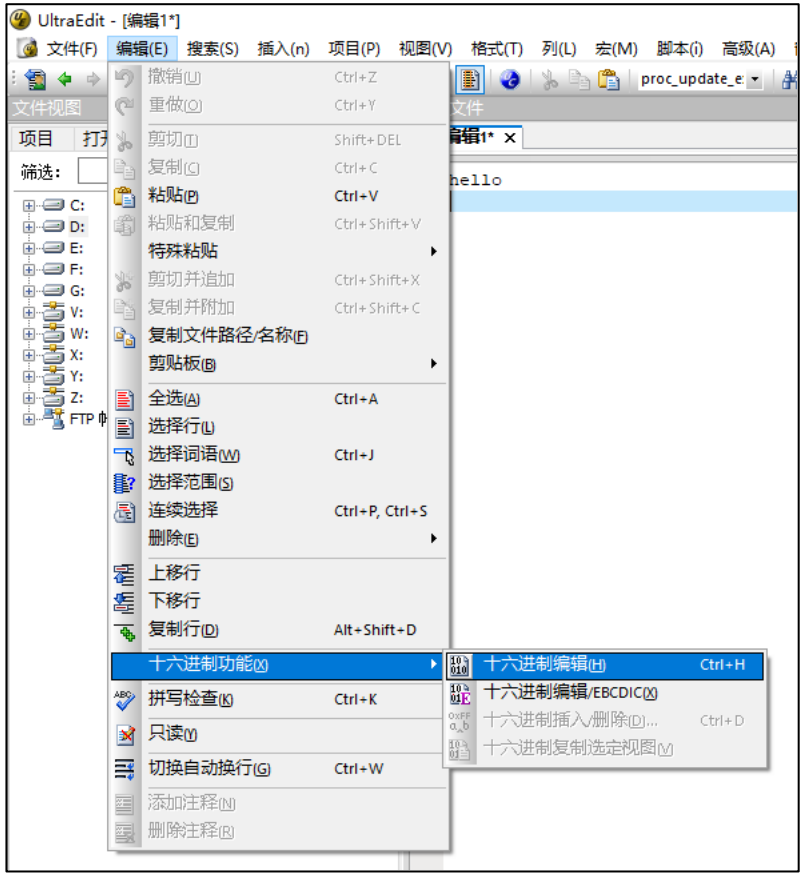
附2：附件给出的UltraEdit查看文件的16进制形式的方法（三种）



正常启动，建立文件，输入Hello



方法1：选择快捷工具栏上的16进制按钮，
可以相互切换



方法2：”编辑”-”十六进制功能”菜单，
可以相互切换

方法3：Ctrl + H 快捷键可以相互切换

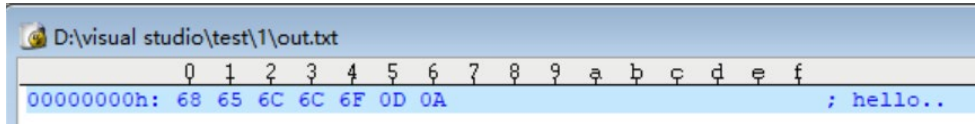
§15.输入输出流 - Windows与Linux的

本页需填写答案

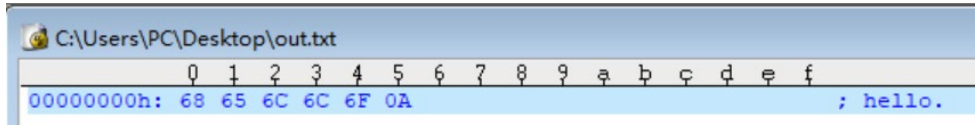
例1：十进制方式写，在Windows/Linux下的差别

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

int main(int argc, char *argv[])
{
    ofstream out("out.txt", ios::out);

    out << "hello" << endl;

    out.close();

    return 0;
}
```

Windows下运行，out.txt是___7___字节，用UltraEdit的16进制方式打开的贴图



Linux下运行，out.txt是___6___字节，用UltraEdit的16进制方式打开的贴图



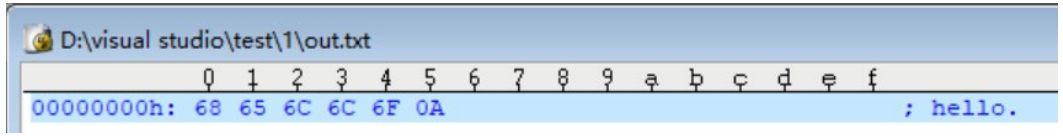
§15. 输入输出流 - Windows与Linux的

本页需填写答案

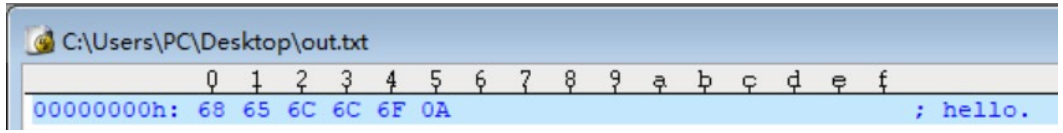
例2：二进制方式写，在Windows/Linux下的差别

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

int main(int argc, char *argv[])
{
    ofstream out("out.txt", ios::out | ios::binary);

    out << "hello" << endl;

    out.close();

    return 0;
}
```

Windows下运行，out.txt是___6___字节，用UltraEdit的16进制方式打开的贴图



Linux下运行，out.txt是___6___字节，用UltraEdit的16进制方式打开的贴图



§15. 输入输出流 - Windows与Linux的文

本页需填写答案

例3：在Linux读取Windows下写的十进制文件

在Linux下运行本程序

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

int main(int argc, char *argv[])
{
    ofstream out("out.txt", ios::out);
    out << "hello\r" << endl; //模拟Windows格式
    out.close();

    char str[80];
    ifstream in("out.txt", ios::in);
    in.getline(str, 80);
    cout << strlen(str) << endl;
    cout << in.peek() << endl;
    in.close();

    return 0;
}
```

在Linux下运行本程序

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

int main(int argc, char *argv[])
{
    ofstream out("out.txt", ios::out);
    out << "hello" << endl;
    out.close();

    char str[80];
    ifstream in("out.txt", ios::in | ios::binary);
    in.getline(str, 80);
    cout << strlen(str) << endl;
    cout << in.peek() << endl;
    in.close();

    return 0;
}
```

本例说明，在Linux下读取Windows格式的文件，要注意0D的处理

Linux下运行，输出结果是：

```
6
-1
```

说明：

1、in.getline读到_换行符_就结束了，_换行符_被读掉，因此in.peek()读到了__EOF__。

2、strlen(str)是__6__，最后一个字符是_换行符_

Linux下运行，输出结果是：

```
5
-1
```

说明：

1、in.getline读到_换行符_就结束了，_换行符_被读掉，因此in.peek()读到了__EOF__。

2、strlen(str)是__5__，最后一个字符是_o_

§15. 输入输出流 - Windows与Linux的文

本页需填写答案

例4：用十进制方式写入含\0的文件，观察文件长度

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

int main(int argc, char *argv[])
{
    ofstream out("out.txt", ios::out);
    out << "ABC\0\x61\x62\x63" << endl;
    out.close();

    return 0;
}
```

Windows下运行，out.txt的大小是__5__字节，Linux下运行，out.txt的大小是__4__字节

为什么？

字符串“ABC\0abc” 被写入，处理字符串时会写入到第一个\0为止。因此输出的结果是“ABC”和endl，Windows下,endl有两字节，最终5字节（A, B, C, \r, \n）。Linux下，endl有一字节，最终4字节（A, B, C, \n）。

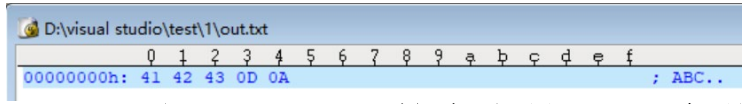
§15.输入输出流 - Windows与Linux的文

本页需填写答案

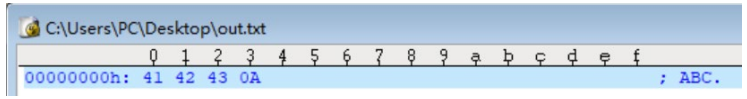
例5：用十进制方式写入含非图形字符(ASCII码32是空格，33-126为图形字符)，但不含\0

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

int main(int argc, char *argv[])
{
    ofstream out("out.txt", ios::out);
    out << "ABC\x1\x2\x1A\t\v\b\xff\175()-=def" << endl;
    out.close();

    return 0;
}
```

Windows下运行，out.txt的大小是__5__字节，UltraEdit的16进制显示截图为：



Linux下运行，out.txt的大小是__4__字节，UltraEdit的16进制显示截图为：



§15. 输入输出流 - Windows与Linux的文

本页需填写答案

例6：用十进制方式写入含\x1A(十进制26=CTRL+Z)和\xFF(十进制255/-1，EOF的定义是-1)的文件，并用十进制/二进制方式读取

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

int main(int argc, char *argv[])
{
    ofstream out("out.txt", ios::out);
    out << "ABC\x1\x2\x1A\t\v\b\xff\175()-=def"<<endl;
    out.close();

    ifstream in("out.txt", ios::in);
    int c=0;
    while(!in.eof()) {
        in.get();
        c++;
        }
    cout << c << endl;
    in.close();

    return 0;
}
```

在Windows系统中，ofstream会在写入文本时自动处理换行符，将\n转换为\r\n（回车和换行），因此文件的大小为20字节。在Linux系统中，文本文件中的换行符保持为\n，不进行转换。所以文件的大小是19字节。在 Windows上，如果你的代码中有\x1A，读取时会在遇到\x1A时停止计数。因此，c的值为6，因为实际读取的字符数（不包括CTRL+Z）是6个。在Linux系统中，读取时每个字符都被单独计数，包括所有的特殊字符和换行符。因此c的值为20。

Windows下运行，文件大小：_____20_____

输出的c是：_____6_____

Linux下运行，文件大小：_____19_____

输出的c是：_____20_____

为什么？

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

int main(int argc, char *argv[])
{
    ofstream out("out.txt", ios::out);
    out << "ABC\x1\x2\x1A\t\v\b\xff\175()-=def"<<endl;
    out.close();

    ifstream in("out.txt", ios::in | ios::binary);
    int c=0;
    while(!in.eof()) {
        in.get();
        c++;
        }
    cout << c << endl;
    in.close();

    return 0;
}
```

in.get()在读取到文件末尾时，eof()标志尚未设置，循环体内的最后一次in.get() 尝试读取超出文件末尾的内容，导致c增加了1。

Windows下运行，文件大小：_____20_____

输出的c是：_____21_____

Linux下运行，文件大小：_____19_____

输出的c是：_____20_____

c的大小比文件大小大_1_，原因是：_____

§15. 输入输出流 - Windows与Linux的文

本页需填写答案

例7：用十进制方式写入含\x1A(十进制26=CTRL+Z)的文件，并用十进制不同方式读取

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

int main(int argc, char *argv[])
{
    ofstream out("out.txt", ios::out);
    out << "ABC\x1\x2\x1A\t\v\b\175()-=def"<<endl;
    out.close();

    ifstream in("out.txt", ios::in);//不加ios::binary
    int c=0;
    while(in.get()!=EOF) {
        c++;
        }
    cout << c << endl;
    in.close();

    return 0;
}
```

在Windows系统中，ofstream会在写入文本时自动处理换行符，将\n转换为\r\n（回车和换行），因此文件的大小为19字节。在Linux系统中，文本文件中的换行符保持为\n，不进行转换。所以文件的大小是18字节。在 Windows上，如果你的代码中有\x1A，读取时会在遇到\x1A时停止计数。因此，c的值为6，因为实际读取的字符数（不包括CTRL+Z）是6个。在Linux系统中，读取时每个字符都被单独计数，包括所有的特殊字符和换行符。因此c的值为18。

Windows下运行，文件大小：_____19_____

输出的c是：_____5_____

Linux下运行，文件大小：_____18_____

输出的c是：_____18_____

为什么？

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

int main(int argc, char *argv[])
{
    ofstream out("out.txt", ios::out);
    out << "ABC\x1\x2\x1A\t\v\b\175()-=def"<<endl;
    out.close();

    ifstream in("out.txt", ios::in); //不加ios::binary
    int c=0;
    char ch;
    while((ch=in.get())!=EOF) {
        c++;
        }
    cout << c << endl;
    in.close();

    return 0;
}
```

在 Linux 中，文本模式下，读取时不会跳过控制字符。但是，使用 EOF 来判断循环的方式可能导致错误。如果使用 (ch=in.get()) != EOF，实际上 in.get() 读取字符时，如果到达文件末尾，它将返回 EOF (-1)，但是ch是一个char类型。此时将字符值与EOF比较将导致不正确的行为，特别是当文件中包含\x1A等字符时。其余同上

Windows下运行，文件大小：_____19_____

输出的c是：_____5_____

Linux下运行，文件大小：_____18_____

输出的c是：_____空白_____

为什么？

§15. 输入输出流 - Windows与Linux的文

本页需填写答案

例8：用十进制方式写入含\xFF(十进制255/-1，EOF的定义是-1)的文件，并进行正确/错误读取

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

int main(int argc, char *argv[])
{
    ofstream out("out.txt", ios::out);
    out << "ABC\x1\x2\xff\t\v\b\175()-=def"<<endl;
    out.close();

    ifstream in("out.txt", ios::in);//可加ios::binary
    int c=0;
    while(in.get()!=EOF) {
        c++;
        }
    cout << c << endl;
    in.close();

    return 0;
}
```

在Windows系统中，ofstream会在写入文本时自动处理换行符，将\n转换为\r\n（回车和换行），因此文件的大小为19字节。在Linux系统中，文本文件中的换行符保持为\n，不进行转换。所以文件的大小是18字节。

Windows下运行，文件大小：_____19_____
输出的c是：_____18_____
Linux下运行，文件大小：_____18_____
输出的c是：_____18_____
为什么?

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

int main(int argc, char *argv[])
{
    ofstream out("out.txt", ios::out);
    out << "ABC\x1\x2\xff\t\v\b\175()-=def"<<endl;
    out.close();

    ifstream in("out.txt", ios::in); //可加ios::binary
    int c=0;
    char ch;
    while((ch=in.get())!=EOF) {
        c++;
        }
    cout << c << endl;
    in.close();

    return 0;
}
```

在 Linux 中，文本模式下，读取时不会跳过控制字符。但是，使用 EOF 来判断循环的方式可能导致错误。如果使用 (ch=in.get()) != EOF，实际上 in.get() 读取字符时，如果到达文件末尾，它将返回 EOF (-1)，但是ch是一个char类型。此时将字符值与EOF比较将导致不正确的行为，特别是当文件中包含\xff等字符时。其余同上

Windows下运行，文件大小：_____19_____
输出的c是：_____5_____
Linux下运行，文件大小：_____18_____
输出的c是：_____空白_____
为什么?

综合例6~例8，结论：结论：当文件中含字符__控制字符__时，不能用十进制方式读取，而当文件中含字符__普通可见字符__时，是可以用二/十进制方式正确读取的