

§ 15. 输入输出流进阶 – Windows与Linux的文件格式差别



要求:

- 1、安装UltraEdit软件，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
- 2、完成本文档中所有的测试程序并填写运行结果，从而体会二进制与十进制文件在不同操作系统下的读写差异，掌握与文件有关的流函数的正确用法
- 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
- 5、无特殊说明，Windows下用VS2022编译，Linux下用C++编译
- 6、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
- 7、**11月3日前**网上提交本次作业（在“文档作业”中提交）

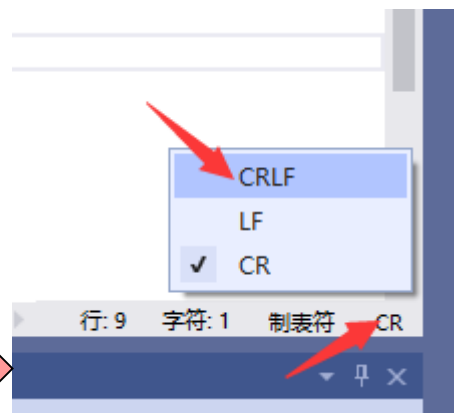
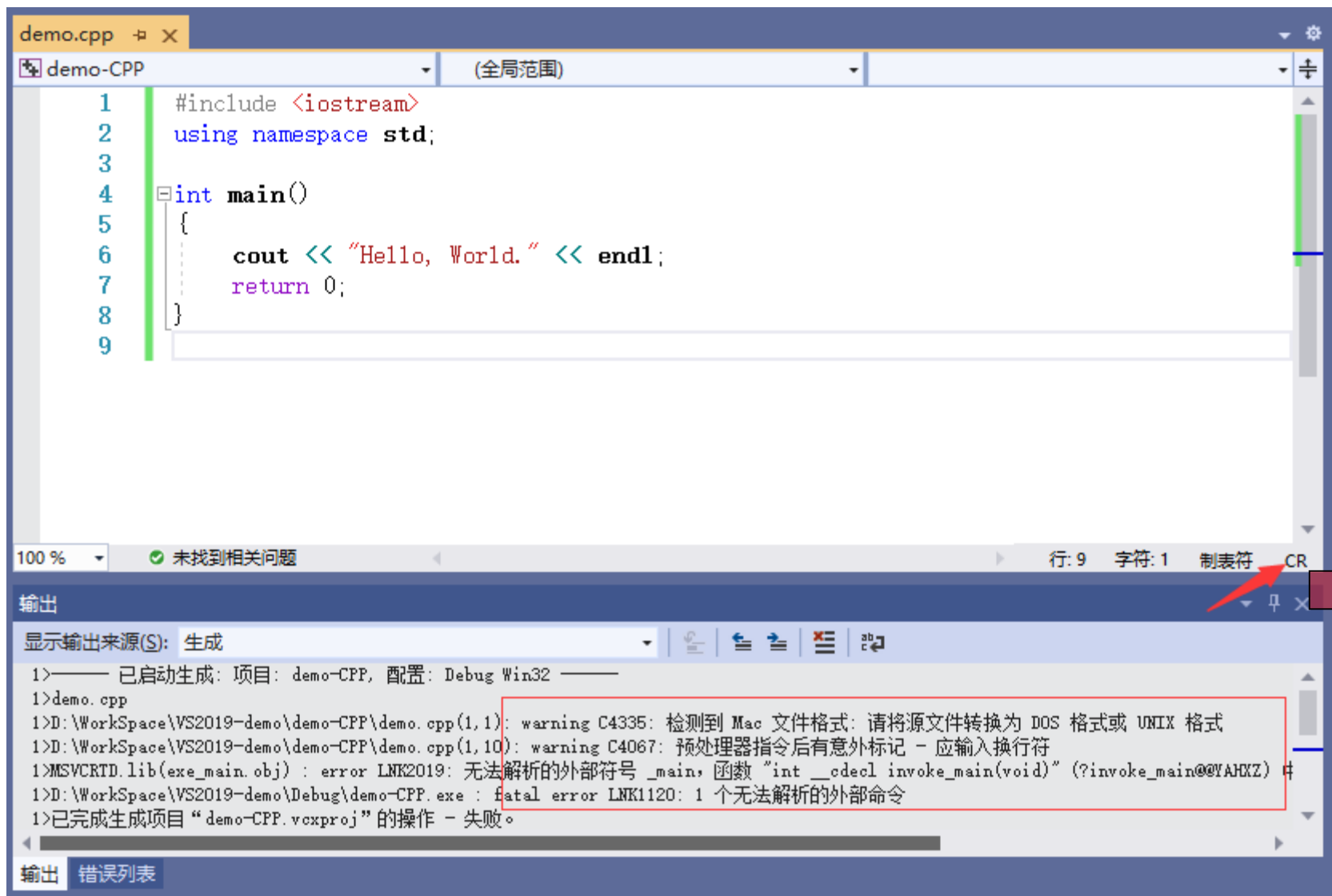
§ 15. 输入输出流进阶 - Windows与Linux的文件格式差别



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

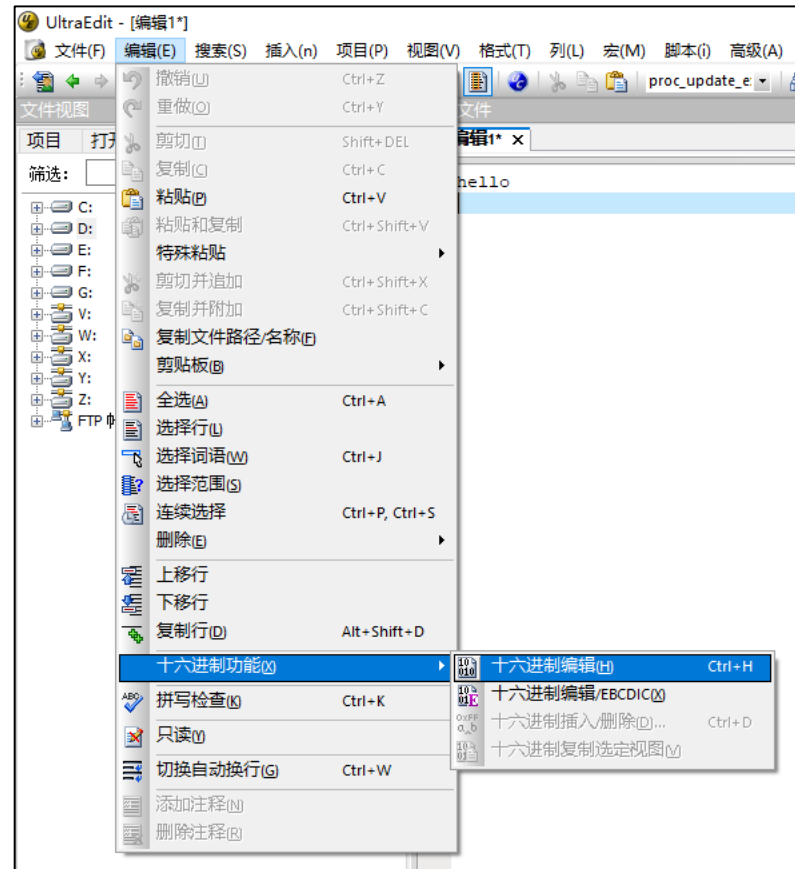
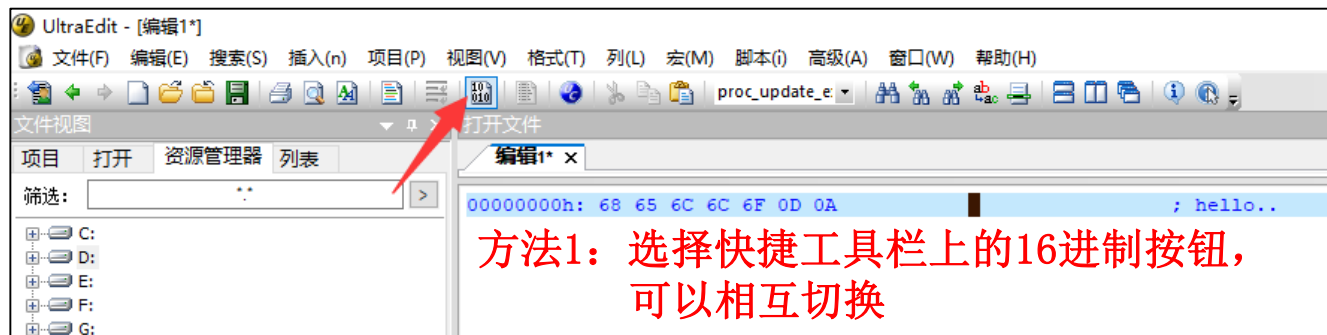
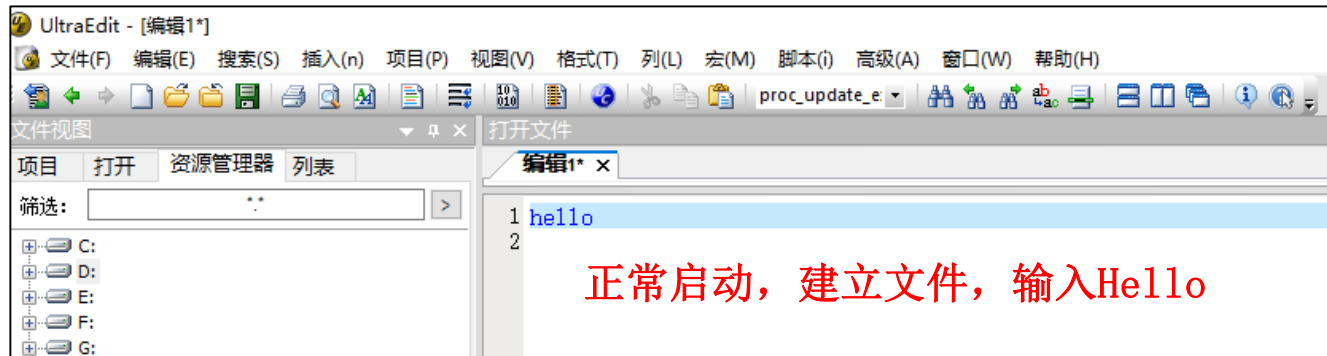


§ 15. 输入输出流进阶 - Windows与Linux的文件格式差别



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)



方法3: Ctrl + H 快捷键可以相互切换

§ 15. 输入输出流进阶 – Windows与Linux

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;

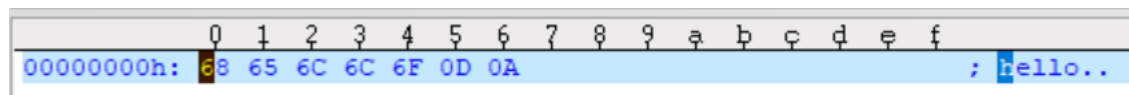
int main(int argc, char *argv[])
{
    ofstream out("out.txt", ios::out);

    out << "hello" << endl;

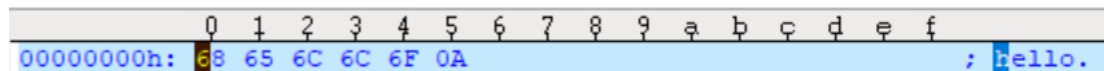
    out.close();

    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. 输入输出流进阶 - Windows与Linux

本页需填写答案



例3: 在Linux读取Windows下写的十进制文件

| | |
|---|---|
| <pre>#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; }</pre> <p>在Linux下运行本程序</p> | <pre>#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; }</pre> <p>在Linux下运行本程序</p> |
| <p>本例说明, 在Linux下读取Windows格式的文件, 要注意0D的处理</p> | |
| <p>Linux下运行, 输出结果是: 说明: 1、in.getline读到__\r__就结束了, __\r__被读掉, 因此in.peek()读到了__-1 (EOF)__. 2、strlen(str)是_6__, 最后一个字符是__\r__</p> | <p>Linux下运行, 输出结果是: 说明: 1、in.getline读到__' o' __就结束了, __' o' __被读掉, 因此in.peek()读到了-1 (EOF)_. 2、strlen(str)是_5__, 最后一个字符是__' o' __</p> |

§ 15. 输入输出流进阶 – Windows与Linux

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?

Linux下对换行符处理为0A, 而windows下为0D 0A, 输出到' \0' 结束

§ 15. 输入输出流进阶 – Windows与Linux

本页需填写答案



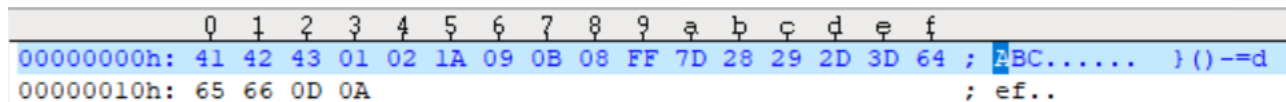
例5：用十进制方式写入含非图形字符(ASCII码32是空格，33-126为图形字符)，但不含\0

```
#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();

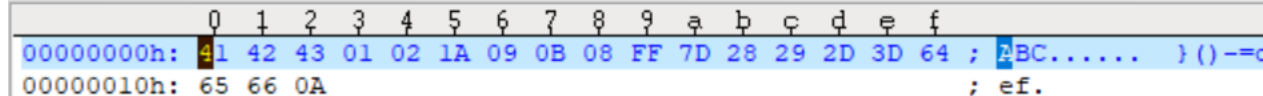
    return 0;
}
```

Windows下运行，out.txt的大小是__20__字节，UltraEdit的16进制显示截图为：



0 1 2 3 4 5 6 7 8 9 a b c d e f
00000000h: 41 42 43 01 02 1A 09 0B 08 FF 7D 28 29 2D 3D 64 ; ABC..... }()-=d
00000010h: 65 66 0D 0A ; ef..

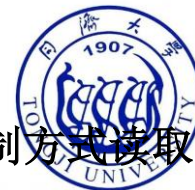
Linux下运行，out.txt的大小是 19 字节，UltraEdit的16进制显示截图为：



0 1 2 3 4 5 6 7 8 9 a b c d e f
00000000h: 41 42 43 01 02 1A 09 0B 08 FF 7D 28 29 2D 3D 64 ; ABC..... }()-=c
00000010h: 65 66 0A ; ef.

§ 15. 输入输出流进阶 - Windows与Linux

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;
}
```

```
#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;
}
```

Windows下运行，文件大小：__20字节__
输出的c是：__6__
Linux下运行，文件大小：__19字节__
输出的c是：__20__

为什么？在Windows十进制下读到0x1A时，视为EOF读取结束，而在Linux中并非如此

Windows下运行，文件大小：__20字节__
输出的c是：__21__
Linux下运行，文件大小：__19字节__
输出的c是：__20__

c的大小比文件大小大，原因是：_feof读到eof时返回值为0，读到下一个字节时返回时才非0，故多读入一个字节__

§ 15. 输入输出流进阶 - Windows与Linux

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;
}
```

```
#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;
}
```

Windows下运行, 文件大小: __19字节__
输出的c是: __5__
Linux下运行, 文件大小: __18字节__
输出的c是: __18__

为什么?在Windows下读到0x1A时, 视为EOF读取结束, 且此次while循环读到EOF时读取便可结束, 而Linux可正常读取

Windows下运行, 文件大小: __19字节__
输出的c是: __5__
Linux下运行, 文件大小: __18字节__
输出的c是: __18__

为什么?此次读取读到EOF时while循环便可正常退出, 且十进制方式读取时0x1A即被视为EOF, 且十进制下换行符被视为一字节处理

§ 15. 输入输出流进阶 - Windows与Linux

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;
}
```

Windows下运行，文件大小：__19字节__
 输出的c是：__18__
 Linux下运行，文件大小：__18字节__
 输出的c是：__18__

为什么?\xff十进制可正常读取，且两种操作系统对换行符的处理不同

```
#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;
}
```

Windows下运行，文件大小：__19字节__
 输出的c是：__5__
 Linux下运行，文件大小：__18字节__
 输出的c是：__5__

为什么?ch读取\xff后被转换为-1，与EOF值相同，读取停止

综合例6~例8，结论：当文件中含字符__\x1A__时，不能用十进制方式读取，而当文件中含字符__\xff__时，是用二/十进制方式正确读取的



§ 15. 输入输出流进阶 – Windows与Linux的文件格式差别

例9~16: 将例1~8复制为例9~16, 改为C方式的文件读写 (VS下要求.c后缀, Linux下要求用gcc编译), 分别在Windows/Linux下运行(), 要求输出结果与C++一致

本页留空, 下页开始复制

§ 15. 输入输出流进阶 – Windows与Linux

本页需填写答案



例9：十进制方式写，在Windows/Linux下的差别

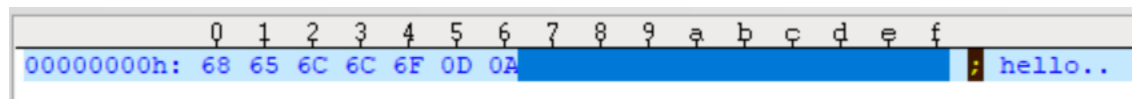
```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main()
{
    FILE *fout;
    fout = fopen("out.txt", "w");
    fprintf(fout, "hello\n");
    fclose(fout);

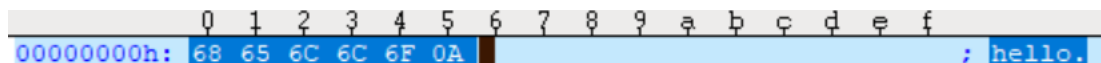
    return 0;
}
```

要求改为C方式读写，
测试程序为.c
endl => \n

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. 输入输出流进阶 - Windows与Linux

本页需填写答案



例10：二进制方式写，在Windows/Linux下的差别

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main()
{
    FILE *fout;
    fout = fopen("out.txt", "wb");
    fprintf(fout, "hello\n");
    fclose(fout);

    return 0;
}
```

要求改为C方式读写，
测试程序为.c
endl => \n

Windows下运行，out.txt是 6 字节，用UltraEdit的16进制方式打开的贴图

[illegible]

Linux下运行，out.txt是 6 字节，用UltraEdit的16进制方式打开的贴图

[illegible]

§ 15. 输入输出流进阶 - Windows与Linux

本页需填写答案



例11: 在Linux读取Windows下写的十进制文件

| | |
|---|---|
| <pre>#define _CRT_SECURE_NO_WARNINGS #include <stdio.h> #include <string.h> int main() { FILE *fout; fout = fopen("out.txt", "w"); fprintf(fout, "hello\r\n"); fclose(fout); char str[80]; fin = fopen("out.txt", "r"); fscanf(fin, "%s", str); printf("%d\n", strlen(str)); printf("%d\n", fgetc(fin)); fclose(fin); return 0; }</pre> <p>在Linux下运行本程序</p> | <pre>#define _CRT_SECURE_NO_WARNINGS #include <stdio.h> #include <string.h> int main() { FILE *fout; fout = fopen("out.txt", "w"); fprintf(fout, "hello\n"); fclose(fout); char str[80]; fin = fopen("out.txt", "rb"); fscanf(fin, "%s", str); printf("%d\n", (int)strlen(str)); printf("%d\n", fgetc(fin)); fclose(fin); return 0; }</pre> <p>在Linux下运行本程序</p> |
| <p>本例说明, 在Linux下读取Windows格式的文件, 要注意0D的处理</p> | |
| <p>Linux下运行, 输出结果是: 说明: 1、fgets读到__ ' \r' __就结束了, __ ' o' __被读掉, 因此fgetc读到了__ ' \r' ____。 2、strlen(str)是_5_, 最后一个字符是__ ' o' __</p> | <p>Linux下运行, 输出结果是: 说明: 1、fgets读到__ ' \n' __就结束了, __ ' o' __被读掉, 因此fgetc读到了__0A(' \n')__。 2、strlen(str)是_5_, 最后一个字符是__ ' o' __</p> |

要求改为C方式读写,
测试程序为.c
endl => \n
in.getline => fgets
in.peek => fgetc
注: C方式无peek,
可用fgetc/ufgetc模拟

§ 15. 输入输出流进阶 – Windows与Linux

本页需填写答案



例12: 用十进制方式写入含\0的文件, 观察文件长度

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main()
{
    FILE *fout;
    fout = fopen("out.txt", "w");
    fprintf(fout, "ABC\0\x61\x62\x63\n");
    fclose(fout);

    return 0;
}
```

要求改为C方式读写,
测试程序为.c
endl => \n

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?

Linux下对换行符处理为0A, 而windows下为0D 0A, fprintf读到' \0' 结束

§ 15. 输入输出流进阶 - Windows与Linux

本页需填写答案



例13: 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

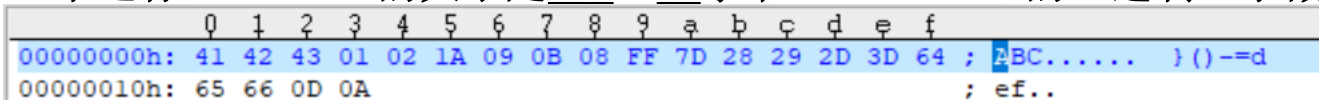
```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main()
{
    FILE *fout;
    fout = fopen("out.txt", "w");
    fprintf(fout, "ABC\x1\x2\x1A\t\v\b\xff\175() -=def\n");
    fclose(fout);

    return 0;
}
```

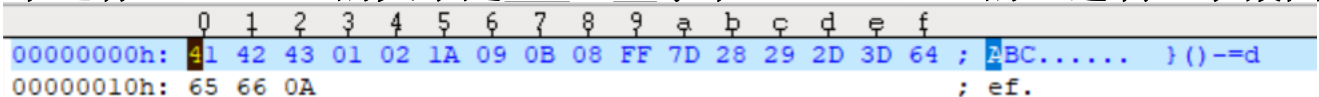
要求改为C方式读写,
测试程序为.c
endl => \n

Windows下运行, out.txt的大小是__20__字节, UltraEdit的16进制显示截图为:



0 1 2 3 4 5 6 7 8 9 a b c d e f
00000000h: 41 42 43 01 02 1A 09 0B 08 FF 7D 28 29 2D 3D 64 ; ABC..... }() -=d
00000010h: 65 66 0D 0A ; ef..

Linux下运行, out.txt的大小是__19__字节, UltraEdit的16进制显示截图为:



0 1 2 3 4 5 6 7 8 9 a b c d e f
00000000h: 41 42 43 01 02 1A 09 0B 08 FF 7D 28 29 2D 3D 64 ; ABC..... }() -=d
00000010h: 65 66 0A ; ef.

§ 15. 输入输出流进阶 - Windows与Linux

本页需填写答案



例14: 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

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
int main()
{
    FILE *fout;
    fout = fopen("out.txt", "w");
    fprintf(fout, "ABC\x1\x2\x1A\t\v\b\xff\175()--def\n");
    fclose(fout);
    fin = fopen("out.txt", "r");
    int c=0;
    while(!feof(fin)) {
        fgetc(fin);
        c++;
    }
    printf("c=%d\n", c);
    fclose(fin);
    return 0;
}
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
int main()
{
    FILE *fout;
    fout = fopen("out.txt", "w");
    fprintf(fout, "ABC\x1\x2\x1A\t\v\b\xff\175()--def\n");
    fclose(fout);
    fin = fopen("out.txt", "rb");
    int c=0;
    while(!feof(fin)) {
        fgetc(fin);
        c++;
    }
    printf("c=%d\n", c);
    fclose(fin);
    return 0;
}
```

要求改为C方式读写,
测试程序为.c
endl => \n

Windows下运行, 文件大小: __20字节__
输出的c是: __6__
Linux下运行, 文件大小: __19字节__
输出的c是: __20__

为什么?windows下十进制读取时0x1A被视为EOF, 读取结束

Windows下运行, 文件大小: __20字节__
输出的c是: __21__
Linux下运行, 文件大小: __19字节__
输出的c是: __20__

c的大小比文件大小大, 原因是: __feof读到eof时返回值为0, 读到下一个字节时返回时才非0, 故多读入一个字节__

§ 15. 输入输出流进阶 - Windows与Linux

本页需填写答案



例15: 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

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
int main()
{
    FILE *fout;
    fout = fopen("out.txt", "w");
    fprintf(fout, "ABC\x1\x2\x1A\t\v\b\xff\175()--def\n");
    fclose(fout);
    FILE *fin = fopen("out.txt", "r");
    int c=0;

    while(fgetc(fin)!=EOF){
        c++;
    }

    printf("c=%d\n", c);
    fclose(fin);
    return 0;
}
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
int main()
{
    FILE *fout;
    fout = fopen("out.txt", "w");
    fprintf(fout, "ABC\x1\x2\x1A\t\v\b\xff\175()--def\n");
    fclose(fout);
    FILE *fin = fopen("out.txt", "r");
    int c=0;
    char ch;
    while((ch =fgetc(fin))!=EOF){
        c++;
    }

    printf("c=%d\n", c);
    fclose(fin);
    return 0;
}
```

要求改为C方式读写,
测试程序为.c
endl => \n
循环改为feof/fgetc
的相等逻辑即可

Windows下运行, 文件大小: __19字节_____
输出的c是: __5_____
Linux下运行, 文件大小: ____18字节_____
输出的c是: ____18____

为什么? Windows下读到0x1A时, 视为EOF读取结束, 且此次while循环读到EOF时读取便可结束, 而Linux可正常读取

Windows下运行, 文件大小: _19字节_____
输出的c是: __5_____
Linux下运行, 文件大小: ____18字节_____
输出的c是: ____18____

为什么?此次读取读到EOF时while循环便可正常退出, 且十进制方式读取时0x1A即被视为EOF, 且十进制下换行符被视为一字节处理

§ 15. 输入输出流进阶 - Windows与Linux

本页需填写答案



例16: 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

| | |
|--|---|
| <pre>#define _CRT_SECURE_NO_WARNINGS #include <stdio.h> #include <string.h> int main() { FILE *fout; fout = fopen("out.txt", "w"); fprintf(fout, "ABC\x1\x2\xff\t\v\b\xff\175() ==def\n"); fclose(fout); FILE *fin = fopen("out.txt", "r"); int c=0; while(fgetc(fin)!=EOF) { c++; } printf("c=%d\n", c); fclose(fin); return 0; }</pre> | <pre>#define _CRT_SECURE_NO_WARNINGS #include <stdio.h> #include <string.h> int main() { FILE *fout; fout = fopen("out.txt", "w"); fprintf(fout, "ABC\x1\x2\xff\t\v\b\xff\175() ==def\n"); fclose(fout); FILE *fin = fopen("out.txt", "r"); int c=0; char ch; while((ch=fgetc(fin))!=EOF) { c++; } printf("c=%d\n", c); fclose(fin); return 0; }</pre> <div data-bbox="1753 218 2150 468" style="border: 1px solid black; padding: 5px; background-color: #e0ffff;"> <p>要求改为C方式读写, 测试程序为.c endl => \n 循环改为feof/fgetc 的相等逻辑即可</p> </div> |
| <p>Windows下运行, 文件大小: __19字节__ 输出的c是: __18__</p> <p>Linux下运行, 文件大小: __18字节__ 输出的c是: __18__</p> <p>为什么? \xff十进制可正常读取, 且两种操作系统对换行符的处理不同</p> | <p>Windows下运行, 文件大小: __19字节__ 输出的c是: __5__</p> <p>Linux下运行, 文件大小: __18字节__ 输出的c是: __5__</p> <p>为什么? ch读取\xff后被转换为-1, 与EOF值相同, 读取停止</p> |
| <p>综合例14~例16, 结论: 当文件中含字符_\0x1A__时, 不能用十进制方式读取, 而当文件中含字符_\xff__时, 是可以用二/十进制方式正确读取的</p> | |