VG101 Lab Manual

Lab 11

Instructor: Dr. Yifei ZHU
TA: Hangrui CAO
TA: Haoxuan SHAN
TA: Qinhang WU
TA: Muchen XU

Table of Contents

I/O Redirection

A program that reads input from the keyboard can also read input from a text file. This is called **input redirection**, and is a feature of the command line interface of most operating systems.

Say that you have a program named `echo.exe` that reads characters from the keyboard and echoes them to directly. Here we create a text file named `input.txt` within the same folder with `echo.exe`. Then we may use the operator `<` in the command line interface to let your program receive the input from the file instead of your keyboard:

```
./echo.exe < ./input.txt
```

Similar feature exists named **output redirection**. The operator `>` in the command line interface allow you to store your output in a local file.

You may take advantage of this feature to test your program more conveniently.

Supplementary on Characters

We summarize a list of special characters here:

ASCII	Definition	Notes
0	NULL	NULL is a special ASCII character rather than nothing.
32	**space**	
10	LF, **Line Feed**	Moves the cursor down to the next line without returning to the beginning of the line.
13	CR, **Carriage Return**	Moves the cursor to the beginning of the line without advancing to the next line.
#undefined	**EOF**	`EOF` is not an ASCII character; indeed it is system-defined, and you should not care what its value is. You need to hold `EOF` with `int` rather than `char`.

NULL (\0)

NULL is an ASCII character, whose decimal value is 0.

You can try this code to check your understanding:

```
printf("%d",0=='\0');
printf("%c %d",'\0','\0');
```

And NULL is always a mark of the end of character array (`char *`). For example, `strlen` will count from the start until it finds a `\0`. Here is a possible realization of `strlen`.

```
int strlen(char *s)
{
    int res = 0;
    while (*s != '\0')
        res++, s++;
    return res;
}
```

Likewise, if you print a character array, it will print the character until a `\0`. And if you read a sequence of character, like

```
char s[20];
scanf("%s",s);//input "Hello, world."
```

The program will automatically **set** a `\0` and the end of the string (right after `.`). Due to the existence of `\0`, you should always reserve extra spaces for your character array, to avoid data overwritten.

```
//This code may behave differently according to the machine.
#include <stdio.h>
int main()
{
    char s[2], S[2];
    S[0] = 'a', S[1] = 'b';
    scanf("%s", s);
    printf("%s", S);
}
//If you enter "a", S[1] will be set by 1
```

Newline (LF,\n)

Newline is an ASCII character, whose decimal value is 10. It moves the cursor to the next line.

You can try

```
printf("%d",10=='\n');
printf("%c %d",'\n','\n');
```

Carriage Return (CR,\r)

CR is an ASCII character, whose decimal value is 13. It moves the cursor to the begining of the line.

You can try

```
printf("%d\n",13=='\r');
printf("123\r23");//This allows you rewrite this line.
```

CRLF (CR-LF)

`CRLF` means a sequence of two characters where `CR` is followed by `LF`. They're used to note the termination of a line. However, they're dealt with differently in today's popular Operating Systems. For example: in Windows both a `CR` and `LF` are required to note the end of a line, whereas in Linux/UNIX/MacOS X a `LF` is only required.

If you are using Windows, you can open `notepad.exe`, just type a return and save the file. You will see the file takes up 2 Bytes disk memory, indicating the file contains `\r\n` (each of which is 1 Bytes).

If interested, you may use software like `Binary Viewer` to check the binary code of a text file.

File input

- When using `'r'` *(text mode)* to read the text from a file, `\r\n` will be automatically converted into `\n` on Windows.
- When using `'rb'` *(binary mode)* to read the text from a file, there is no format transformation. Therefore, you will read two characters `\r` `\n` separately when meeting a newline on Windows.
- Input redirection `<` will read the file with text mode by default.

std::endl

`std::endl` will output a newline with the ASCII character(s) that fit your operating system.

Note: `std::cout << std::endl;` works slightly differently with `std::cout << '\n';`.

EOF

EOF indicates "End of File".

> EOF indicates "end of file". A newline (which is what happens when you press enter) isn't the end of a file, it's the end of a line, so a newline doesn't terminate this loop.
>
> The code isn't wrong*, it just doesn't do what you seem to expect. It reads to the end of the input, but you seem to want to read only to the end of a line.
>
> The value of EOF is -1 because it has to be different from any return value from getchar that is an actual character. So getchar returns any character value as an unsigned char, converted to int, which will therefore be non-negative.

`EOF` is not an ASCII character; indeed it is system-defined, and you should not care what its value is. It can be treated as a status code denoting the condition of input stream (whether it reaches the end of input). For example, `scanf`, `getchar`, or would **return** `EOF` if it encountered the end of input.

So `EOF` shouldn't be compared with a character. It should be compared with the return value of a (well-defined) input function. In a word, You need to hold `EOF` with `int` rather than `char`.

Provoke EOF without Files

If you're typing in the terminal and you want to provoke an end-of-file manually, use CTRL-D (unix-style systems) or CTRL-Z (Windows).

EOF in C

Normally, `EOF` is defined like:

```
#ifndef EOF
# define EOF (-1)
#endif
```

In C, we do comparison with `EOF` explicitly:

```
int num;
while(scanf("%d",&num) != EOF) printf("%d",num);
while(~scanf("%d",&num)) printf("%d",num);// Equivalent, ~(-1)==0
```

```
// EFFECTS: read a character each time until reaching End of File,
// output it whenever the input status is normal
char ch;
while ((ch=getchar()) != EOF) putchar(ch);
while (~(ch=getchar())) putchar(ch);// Equivalent, ~(-1)==0
```

EOF in C++

While in C++, input stream `istream` implicitly returns the state of itself. If it has encountered the end of file, it will implicitly converted to `false`. Or you can explicitly use the method `ios::eof()` to check whether it encounters the end of file.

c++ style:

```
using namespace std;
// ...
int num;
while(!(cin >> num).eof()) cout << num;
while(cin >> num) cout << num;  //Equivalent, EOF state implicitly converted
to false
```

```
// EFFECTS: read a whole line each time until reaching EOF;
// printing it whenever the input status is normal
using namespace std;
// ...
string line;
while(!getline(cin, line).eof) cout << line;
while(getline(cin, line)) cout << line; //Equivalent
```

Grading Rubric

Criteria	Weight	Available Time	Due Time	Entry
Attendance	100%	**4:00pm**, July.24	**11:59am**, July.28	Canvas
Practical Exercise	0%	**4:00pm**, July.24	**11:59pm**, July.28	JOJ

Reference

1. Ascii Table
2. newbie, “End of File (EOF) in C,” Stack Overflow, 05-Dec-2010. [Online]. Available: https://stackoverflow.com/questions/4358728/end-of-file-eof-in-c. [Accessed: 23-Jul-2020].
3. dot, “Input Redirection,” Ccsu.edu, 2020. [Online]. Available: https://chortle.ccsu.edu/java5/Notes/chap22/ch22_2.html. [Accessed: 23-Jul-2020].