File Reading – a caution

chars aren't the same on x86-64
and on ARM !

Code up this test program:

Compile and run it on both the lab computer (or your laptop),

and on a Raspberry Pi

```
//  Test EOF and characters on x86-64, and on Raspberry Pi:
// 2019-10-10
#include <stdio.h>

int main(int argc, char **argv)
{
    if (EOF < 0)
        printf("EOF %02x (%c) is signed\n", EOF, EOF);
    else
        printf("EOF %02x (%c) is unsigned\n", EOF, EOF);

    char c = EOF;
    if (c < 0)
        printf("char c %02x (%c) is signed\n", c, c);
    else
        printf("char c %02x (%c) is unsigned\n", c, c);

    return 0;
}
```

Result on x86-64, and on ARMv8

- x86-64 (Linux):

```
744] ./EOFtest
EOF ffffffff (ÿ) is signed
char c ffffffff (ÿ) is signed
2019-10-10, 03:49 bobmon@whitek
```

- ARMv8 (Raspberry Pi, Linux):

```
261] ./EOFtest
EOF ffffffff (ÿ) is signed
char c ff (ÿ) is unsigned
2019-10-10, 08:53 bobmon@caterp
```

What does this mean for file reading?

- Reading with "fgets()"
  - Returns NULL at End-Of-File
  - No problem (NULL is "zero")
- Reading with "fgetc()" or "fscanf()"
  - Returns EOF at End-Of-File
  - How do you test for this value?

```
char c;
while ( EOF != (c = fgetc(handle)) )
    printf("%#02x %c\n", c, c);
```

Consider this program:

- Features:
  - rewind()
  - fgets() ends w/ NULL
  - fgetc() ends w/ EOF
- Works fine on x86-64

```c
// Test fgetc on x86-64 and on ARMv8.
// 2019-10-10
#include <stdio.h>
#define BUFSIZE 100

int main(int argc, char **argv)
{
   if (argc < 2) {
      fprintf(stderr, "usage: %s <input-file>\n", argv[0]);
      return 1;
   }
   char buffer[BUFSIZE];
   int nlines = 0;
   FILE *handle = fopen(argv[1], "r");
   while ( NULL != fgets(buffer, BUFSIZE, handle) )
      printf("%2d %s", ++nlines, buffer); // No newline in format string!
   printf("\n#--------\n");

   rewind(handle);

   char c;
   int nchars = 0;
   while ( EOF != (c = fgetc(handle)) )
      printf("%2d %#02x %c\n", ++nchars, c, c);

   fclose(handle);

   return 0;
}
```

Program works on x86-64

- This sample file:

```
1st line
last line
```

produces the output shown on the right →

```
746] ./fgetc-fgets sample.txt
 1 1st line
 2 last line

#--------
 1 0x31 1
 2 0x73 s
 3 0x74 t
 4 0x20
 5 0x6c l
 6 0x69 i
 7 0x6e n
 8 0x65 e
 9 0xa

10 0x6c l
11 0x61 a
12 0x73 s
13 0x74 t
14 0x20
15 0x6c l
16 0x69 i
17 0x6e n
18 0x65 e
19 0xa

2019-10-10, 06:01 bobmon@whiteb
747]
```

Copy the program to a Raspberry Pi:

- Copy it over, using "scp":
- **`scp fgets-fgetc.c  compsci@172.16.1.1`**

- Compile and run it there:
- **`gcc -Wall -o fgets-fgetc fgets-fgetc.c`**
- **`./fgets-fgetc  sample.txt`**

- *What happens?*

Oopsy!

- The EOF test to end the "fgetc()" loop fails!
- EOF is "-1" but the Raspberry Pi's character type is a *positive* value!

```
275] ./fgetc-fgets sample.txt
 1 1st line
 2 last line

#--------
 1 0x31 1
 2 0x73 s
 3 0x74 t
 4 0x20
 5 0x6c l
 6 0x69 i
 7 0x6e n
 8 0x65 e
 9 0xa

10 0x6c l
11 0x61 a
12 0x73 s
13 0x74 t
14 0x20
15 0x6c l
16 0x69 i
17 0x6e n
18 0x65 e
19 0xa

20 0xff
21 0xff
22 0xff
23 0xff
24 0xff
25 0xff
26 0xff
```

The portable solution:

- Declare c as an *integer*:

```
int c;
while ( EOF != (c = fgetc(handle)) )
    printf("%#02x %c\n", c, c);
```

- This matches the return type of "fgetc()"
  - "printf()" casts the integer c to a char