Assignment 1: Unicode `file(1)`

CompSys 2017
Department of Computer Science
University of Copenhagen

Peter, Tobias & Xueying

17. september 2017

1 Compiling and running the code

1. Unzip `src.zip`.
2. Navigate to the `src` directory.
3. Type the command: `make` in a UNIX-like command line to compile the `file`-program.
4. Type the command: `bash test.sh` in a UNIX-like command line to run the `./file` and `file(1)` comparison-tests.
5. Type the command: `gdb file -x test.gdb` in a UNIX-like command line to run the gdb test script.

2 Implementation

The purpose of `file` is to determine the type(s) of one or more given files.

The program will first find out how many arguments that it took, if none then a usage message will be delivered; if any, it will determine the longest argument, and use it to calculate the number of spaces that the final output uses. It should fulfill its ultimate purpose: determine the type of one or more given files which will be explained in the upcoming sections.

2.1 `type_detector`

The type detector is built to return the corresponding type number to the print function in main with the help of type checkers. While this works we acknowledge that the running time of the program becomes quite high, this is of course the prize of using this kind of method.

Empty type is the first one we check, since if the file is empty then it is unnecessary to go through the rest process. ISO-8859 has a bit larger range than ASCII, hence ASCII checker is placed before ISO checker. We assume that a file

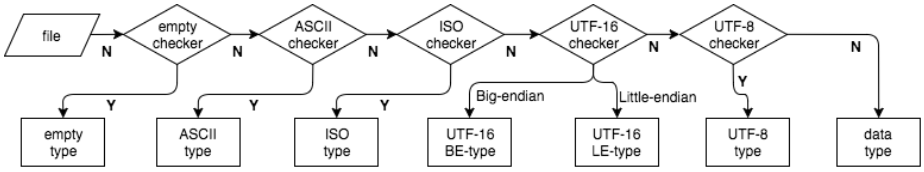

Figure 1: Control flow in `file`.

cannot be encoded in both UTF-8 and UTF-16. As for UTF-16 is less complicated to recognize when we assume such a file always begins with a unique byte-order-mark, we placed the UTF-8 checker at the end of the whole process.

2.2 Empty files

`empty_checker()` uses `fseek()` and `ftell()` to compute the size of the file. If `fseek()` is unable to move the pointer, the length of the file is 0 and the file will be marked as being empty.

2.3 ASCII files

The ASCII checker function works by checking if the file contains any characters that is not considered ASCII characters. These characters lies outside of the range of $\{0x07, 0x08, ...0x0D\} \cup \{0x1B\} \cup \{0x20, 0x21, ...0x7E\}$ If the file consist of non-ASCII characters then the file will be considered non-ASCII, otherwise the file will be marked as ASCII.

2.4 ISO-8859-1 files

The ISO-8859 checker function works in a similar way it just checks within a broader range including the same range as the ASCII checker but also the decimal values of 160-255.

2.5 UTF-8 files

The UTF-8 checker function can determine if a file is encoded in UTF-8 by using the designed bit-sequence of its encoding. As we known that different number of bytes has different unique pattern of byte(s) in UTF-8, according to this characteristic, we have four if-statement to cover all four numbers of bytes.

2.6 UTF-16 files

The UTF-16 checker function is similar to the UTF-8 checker above, it as well uses a characteristic of UTF-16 to determine whether the big-endian or little-endian is used.

3 Theory

Our macro functions make use of *bit masking* to compare a select number of bits to a designated bit-sequence. This is achieved using a *ternary if*-expression,[1] keeping the macros simple.

4 Testing

4.1 `test.sh` - our program in comparison

In addition to the tests described in our previous report, we have written tests that compare the behaviour of `./file` with `file(1)` when processing the new file types / character encoding formats that our program now supports.

Since it is no longer sufficient to just write certain characters to various files, we are using `iconv(1)` to encode the files, thus ensuring that any given file has the required character encoding for the test.

Our way of handling UTF-16 Big Endian and Little Endian character encoding may require further explanation. Considering that we do not know what machine our code will be run on, it is necessary to go through these additional steps: We first encode the test file to UTF-16, leaving the machine to use whatever endian standard is defined. We then examine the specific encoding of the UTF-16-file and lastly use `iconv(1)` to convert from whatever encoding the machine used to the one we require for our test.

4.2 `test.gdb` - testing the debug utility

The script contained in `test.gdb` demonstrates various basic uses of gdb. Initially we print various number representations as other representations. Next we define the macros `UTF8_2B()`, `UTF8_3B()`, `UTF8_4B()` and `UTF8_CONT()` so as to be able to run the supplied tests. It is necessary to define the macros within gdb, as gdb doesn't recognize the ones we have defined in `file.c`. Disregarding their placement in `test.gdb`, the macros are identical to the preprocessor functions defined in `file.c`.

Please note that the gdb command `set pagination off` at line 8 allows the script to run to end without requesting user-input, before continuing.

[1] `boolean expression ? result if true : result if false`.