# EMAT10006 Assignment 2

Feb 2016

## 1 Overview

This assignment tests your ability to make a C program that can do simple file input/output. This is a basic task in computer programming. As before the requirements for this program are precise so please read them carefully.

## 2 Packaging requirements

Your program will be called "hd". You will submit your code to SAFE as a .zip file called "hd-1.0.zip". Inside the zip file is a directory called "hd-1.0" and inside that directory there will be

- Makefile
- README.txt
- hdmain.c
- a folder called tests

The contents of the "tests" folder can be found as "tests.zip" on Blackboard. The program will be compiled by running "make" from inside the "hd-1.0" directory. This will produce an executable called "hd.exe".

## 3 Tests

The "tests.zip" file on Blackboard contains files that you will use as part of this assignment. It contains a number of test input and output files. It also contains a tset script called "runtests.py" which tests your program against each input/output combination. For convenience tests will be executed by running "make test". The "test" command should be added to the "Makefile" and will look like

```
test: hd.exe
	python tests/runtests.py
```

Declaring "hd.exe" as a dependency of tests means that "make test" will recompile the program before running the tests. You should use this command while developing the program. Makefiles are useful: don't just add them at hte end to please me!

# 4 Program

The program "hd" is short for "hex dump". The idea of this program is that it reads a binary (non-text) file and displays its contents in a textual format. This makes it easy to view the content of a binary file in an environment intended for viewing text. Here's an example of how it works. Suppose that we have a C file containing

```
/* hello.c */

#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Hello world!\n");
  return 0;
}
```

Then we can run "hd" to see a representation of the bytes from the file:

```
$ ./hd.exe hello.c
00000000  2f 2a 20 68 65 6c 6c 6f  2e 63 20 2a 2f 0a 0a 23  |/* hello.c */..#|
00000010  69 6e 63 6c 75 64 65 20  3c 73 74 64 69 6f 2e 68  |include <stdio.h|
00000020  3e 0a 0a 69 6e 74 20 6d  61 69 6e 28 69 6e 74 20  |>..int main(int |
00000030  61 72 67 63 2c 20 63 68  61 72 20 2a 61 72 67 76  |argc, char *argv|
00000040  5b 5d 29 0a 7b 0a 20 20  70 75 74 73 28 22 48 65  |[]).{.  puts("He|
00000050  6c 6c 6f 20 77 6f 72 6c  64 21 5c 6e 22 29 3b 0a  |llo world!\n");.|
00000060  20 20 72 65 74 75 72 6e  20 30 3b 0a 7d 0a        |  return 0;.}.|
0000006e
```

Let's consider what that output means:

- Firstly "hd" reads the file specified on the command line. In this case it is reading the file called "hello.c" because that was given as input argument. If a different filename were given on the command line say "foo.txt" then the program should read that file instead.
- Secondly the program reads the file 16 bytes at a time. Each chunk of 16 bytes is displayed on one line of output.
- The first column of output is the number of the first byte in the chunk in 8 digit hexadecimal.
- After the first column there are two spaces.
- Then the 16 bytes are displayed in two groups of 8 with 2 spaces between the groups.

- Each group shows 8 bytes each in 2 digit hexadecimal separated with 1 space between each byte.
- After the 2nd group there are two spaces.
- Then a text representation of the 16 bytes is shown. Bytes whose ASCII character is "printable" are written to the output (the terminal shows the ASCII character). Non-printable bytes such as newline/tab etc are shown as ".".
- Bytes are considered to be printable if their integer value $n$ is in the range $0x20 \leq n < 0x79$.
- The text representation is between two pipe "—" characters.
- The final line just shows the total number of bytes in the file.
- The total number of characters on each line is 78 (not including the newline):

$$8 + 2 + (8 \times 2 + 7) + 2 + (8 \times 2 + 7) + 2 + (1 + 16 + 1) = 78$$

- The second last line of output will be incomplete if the number of bytes in the file is not a multiple of 16.

Your program should also detect if the user doesn't give the right number of arguments or if the file given cannot be opened. In this case it should print an error message to stderr and return exit code 1:

```
$ ./hd.exe
Usage: ./hd.exe FILENAME
$ ./hd.exe too many arguments
Usage: ./hd.exe FILENAME
$ ./hd.exe invalid_filename
Unable to open "invalid_filename"
$ echo $?
1
```

# 5 Tips

You can read the file 16 bytes at a time using the "fread" function. To print a number in hexadecimal with 8 digits and leading zeros you can use

```
printf("Here is is: %08x  \n", number);
```

This will ensure that e.g. the number 1 is printed as 00000001.

Use the test files and test script when making this program!