05OGDLP - Algorithms & Programming

Cristiano **Pegoraro Chenet**, Riccardo **Smeriglio**

Department of Control and Computer Engineering (DAUIN),
Politecnico di Torino, Italy

October 10, 2024

Laboratory 3

To have the additional points the exercises of this laboratory must be submitted by Wednesday 16 October 2024 at 23:59 UTC+1.

Learning objectives

- Arrays, multidimensional arrays, strings
- Functions
- Input verification
- File management
- Command line parameters

1 Lab Exercises

1.1 Input verification and conversion

a) Integer conversion with `atoi`: write a program which receives a string from user's keyboard (quantity of tokens) and convert it to an integer.

```
Tokens dispenser
----------------

Please choose the quantity of tokens to deposit: 1234
The tokens value you entered is: 1234

Please choose the quantity of tokens to deposit: 56abc
The tokens value you entered is: 56
```

```
Please choose the quantity of tokens to deposit: 78.9
The tokens value you entered is: 78

Please choose the quantity of tokens to deposit: abc
The tokens value you entered is: 0
```

b) Integer conversion with `atof`: modify to previews program to a deposit operation in ATM machine. The program should receives a string from user's keyboard (quantity to deposit) and convert it to a float with 2 decimal digits.

```
A&P Bank Virtual ATM
--------------------

Enter the amount you wish to deposit: 1234
You deposited: 1234.00

Enter the amount you wish to deposit: 1234.1234
You deposited: 1234.12

Enter the amount you wish to deposit: 56abc
You deposited: 56.00

Enter the amount you wish to deposit: 78.9
You deposited: 78.90

Enter the amount you wish to deposit: abc
You deposited: 0.00
```

c) Verification of the number of passport characters with `strlen`. The italian passport is composed by 9 characters, as follows: CCNNNNNNN, where the first two characters are alphabets followed by a 7-digit numbers. Use the `strlen` function to verify if the number of characters from a user's passport is right.

```
Passport verification
---------------------

Please enter the passport number: AB1234567
The passport you entered is: AB1234567


Please enter the passport number: AB123456
The passport you entered is invalid.
```

```
Please enter the passport number: 123456789
The passport you entered is: 12345678
```

d) Verification of the passport characters according alphanumeric characters with `isalpha` and `isdigit`. Improve the previews program verifying of the passport number is comply with the rule "first two characters are alphabets followed by a 7-digit numbers".

```
Passport verification
_____________________

Please enter the passport number: AB1234567
The passport you entered is: AB1234567

Please enter the passport number: ABC123456
The passport you entered is invalid.

Please enter the passport number: A12375678
The passport you entered is invalid.

Please enter the passport number: 12ABCCDFG
The passport you entered is invalid.

Please enter the passport number: ab1234567
The passport you entered is: ab1234567
```

e) Conversion of input characters to uppercase/lowercase: the program prints two option choices to the user, that should reply with yes (Y) or no (N). Use the function `toupper` or the `tolower` to accept both responses in uppercase and lowercase characters.

```
Digit (Y) to continue and (N) to quit: Y
Continuing the application...
```

```
Digit (Y) to continue and (N) to quit: y
Continuing the application...
```

```
Digit (Y) to continue and (N) to quit: n
Quitting...
```

1.2 Morse Code Translator

Morse code, utilized in telecommunications, encodes text characters into standardized sequences of short and long signals—dots and dashes, respectively. For a deeper understanding, see the detailed description on Wikipedia.

Your task is to write a C program capable of encoding and decoding messages using the International Morse code. This code covers the 26 basic Latin letters ([A-Z]), digits ([0-9]), and a subset of punctuation symbols. The Morse code representations for these characters is provided in the `alphabet.in` file.

Write a C program that:

- Utilizes an efficient data structure, preferably a zero-based array, to map characters to their Morse representations based on the content of `alphabet.in`.
- Presents users with an interactive menu to select: encode, decode, or exit (hint: a switch-case structure can be helpful here).
- Read the text to decode or encode from a specified filename.
- Write the output of the corresponding translation on `stdout` and on a specified file.

```
Morse code translator
---------------------

(E)ncode (D)ecode e(X)it >>> e
Input file name > cleartext.in
Output file name > encoded.out
Encoded message saved to file encoded.out
Encoded message: .... . .-.. .-.. ---  --..--  / .-- --- .-
. .-.. -.. -.-.--

(E)ncode (D)ecode e(X)it >>> d
Input file name > msg.in
Output file name > decoded.out
Decoded message saved to file decoded.out
Decoded message: HELLO, WORLD!

(E)ncode (D)ecode e(X)it >>> x
```

Implementation notes:

- In C, character variables inherently store ASCII values, i.e., integers between 0 to 127. Using this ASCII table can inspire an optimal data structure choice. A zero-based array, with ASCII values serving as indices, can efficiently map characters to Morse codes.

```
fp = fopen("alphabet.in", "r")
while(fscanf(fp, "%c\t%s\n", &letter, code) != EOF){
        strcpy(alphabet[letter - ' '], code);
}
fclose(fp);
```

- Within a word, Morse symbols are separated by spaces. Different words, on the other hand, are demarcated by the `/` character. Additionally, the newline character (`\n`) is represented as `/`.
- Our Morse includes uppercase letters only. However, your solution should equate both upper and lower-case letters in Morse conversion (e.g., `a` and `A` both translate to .-). The functions defined in the ctype.h header can help you achieve this goal.
- Design an intuitive user interface using a `switch-case` structure. It is an effective method to process user selections, guiding them through the encoding and decoding stages.

2 Homework

2.1 Command Line Morse Code translator

Update the previous program to work using only command line parameters. The `alphabet.in` file can remain hard-coded, but the input filename and the output filename to encode/decode must be invoked by the user when launching the program.

```
Morse code translator
---------------------

(E)ncode (D)ecode e(X)it >>> e
Encoded message saved to file encoded.out
Encoded message: .... . .-.. .-.. --- --..-- / .-- --- .-
. .-.. -.. -.-.--
```

```
Morse code translator
---------------------

(E)ncode (D)ecode e(X)it >>> d
Decoded message saved to file decoded.out
Decoded message: HELLO, WORLD!
```

Compile the program:

```
gcc -o morse_command morse_command.c
```

Run the program (example):

```
./morse_command cleartext.in encoded.out
```

Acknowledgment

The teachers would like to thank Leonardo Giannantoni for writing the "1.2 Morse Code Translator".