Computacção em Larga Escala

(ano letivo 2024'25)

2 - Word Count

In this exercise, you will develop a program that counts the number of characters, lines, and words in a set of input files. Your program must accept a list of file paths as input and process each file individually while ensuring that all files are correctly interpreted using UTF-8 encoding. For the exercice you will be using C++ and CMake to build the program.

2.1 - Project Structure

```
weather-stations/
├── CMakeLists.txt        # CMake configuration file
├── Makefile              # Make build configuration
├── book                  # directory with all the books
├── src/
│   ├── main.cpp           # Main program file that reads and processes measurements
│   ├── word_count.cpp     # Counts the characters, lines and words.
│   └── utf-8.cpp          # UTF-8 processor
├── build/               # Build output directory (generated)
│   ├── cle-wc           # Main program executable
```

2.2 - Build

Run the following command to build the program:

```
make
```

Run the following command to run the program:

```
./build/cle-wc <input_file>
```

2.3 - Input Data Format

The input file is a list of files to process. Each line in the file follows this format:

```
file_name
```

The input files are plain text file and are encoded in UTF-8.

2.4 - Requirements

1. Modify the `main.cpp`, `word_count.cpp` and `utf-8.cpp` files to:
    - read the input file and get the list of files to process
    - for each file, read the content and process it, i.e. count the number of characters, lines and words
    - accumulate the results for all files
    - implement the utf-8 processor in the `utf-8.cpp` file
    - print the results in the expected format
2. Expected output format for each file:

```
file_name: <number_of_characters> <number_of_lines> <number_of_words>
...

total: <number_of_characters> <number_of_lines> <number_of_words>
```

2.5 - UTF-8 Character Reference Guide

- First byte patterns:

```
1 byte:  0xxxxxxx
2 bytes: 110xxxxx 10xxxxxx
3 bytes: 1110xxxx 10xxxxxx 10xxxxxx
4 bytes: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
```

- Continuation bytes always start with `10` (0x80-0xBF)

See `utf-8.h` for more details.

2.5.1 - Letter Codepoints

- Basic Latin Letters
  - 0x41-0x5A: Uppercase Latin A-Z
  - 0x61-0x7A: Lowercase Latin a-z
- Extended Latin and European Scripts
  - 0xC0-0xFF: Latin-1 Supplement (À-ÿ)
  - 0x100-0x17F: Latin Extended-A
  - 0x180-0x24F: Latin Extended-B
  - 0x250-0x2AF: IPA Extensions
- Other Scripts
  - 0x370-0x3FF: Greek and Coptic
  - 0x400-0x4FF: Cyrillic
  - 0x500-0x52F: Cyrillic Supplement
- Symbols and Special Letters
  - 0x2100-0x214F: Letterlike Symbols
- CJK (Chinese, Japanese, Korean)
  - 0x4E00-0x9FFF: CJK Unified Ideographs

2.5.2 - White Space Codepoints

- ASCII Whitespace
  - 0x0009: Horizontal Tab (\t)
  - 0x000A: Line Feed (\n)
  - 0x000B: Vertical Tab
  - 0x000C: Form Feed
  - 0x000D: Carriage Return (\r)
  - 0x0020: Space
- Unicode Spaces
  - 0x00A0: No-Break Space
  - 0x2007: Figure Space
  - 0x202F: Narrow No-Break Space
  - 0x2060: Word Joiner