Exercise sheet 2 – Data representation

Goals:

- Codes: Unicode and UTF
- Number representation and formats

Exercise 2.1: Data representation (recapitulation semester 1–3)

(a) Represent the number *-16* in hex-format for an 32-bit architecture.

Exercise 2.2: Data representation: Unicode and UTF

(a) Find the unicode character U+20214 in the unicode table. Hint: With Linux Mint Mate (the VM), you can use the graphical tool „Character Map“ (Menu -> type: Character Map). Another way is to use `https://unicode-table.com` to perform the search.

(b) On which plane is unicode character U+20214?

(c) Translate (encode) the unicode character U+20214 into UTF-8.

(d) Translate (encode) the unicode character U+20214 into UTF-16.

(e) Translate (encode) the unicode character U+20214 into UTF-32.

Exercise 2.3: Data representation: Unicode and UTF: file handling

(a) What is the meaning of the unicode character U+2696F?

(b) Use Visual Studio Code to create a file `text_utf8.txt` with the content „I like to <unicode character>“. Use U+2696F as the unicode character.

(c) On the shell, use `xxd text_utf8.txt` to visualise the content of the file as hex. Can you find the UTF-8 encoded U+2696F unicode character?

(d) Use `iconv -f UTF-8 -t UTF-16LE text_utf8.txt > text_utf16.txt` to convert the text into UTF-16.

(e) On the shell, use `xxd text_utf16.txt` to visualise the content of the file as hex. Can you find the UTF-16 encoded U+2696F unicode character?

(f) Can you open the `text_utf16.txt` file with Visual Studio Code?

(g) Use again `iconv` to convert the text into UTF-32.

(h) Again, use `xxd` to visualise the content of the file as hex. Can you find the UTF-32 encoded U+2696F unicode character?

(i) Can you open the UTF-32 encoded file with Visual Studio Code?

Exercise 2.4: Number representation (theoretical)

Given are the decimal numbers 50.5 and 0.80.
Hint: Represent the numbers initially as binary fraction. Use appropriate scaling.

(a) State the bit pattern for the **binary fixed point** format. *Hint: You may define the position of the fixed point.*

(b) State the bit pattern for the **decimal fixed point** format and add them. *Hint: You may represent each digit with BCD.*

(c) State the bit pattern for the **binary floating point** format.

(d) List some pros and cons of **fixed point** numbers: **binary fixed point** vs **decimal fixed point**. *Hint: You may think about performance, cost, accuracy, programming language support, ….*

Exercise 2.5: Binary floating point number (coding)

(a) Write a C program with a for loop (from 1 to 500). In every loop it adds 0.8 to a `float` variable. At the end, print the result with a precision of 7 digits.
Use `RA_exercises/sheet_02/binary_floating_point/binary_floating_point.c` as a starting template.

(b) Compile and run the program with:

```
cd RA_exercises/sheet_02/binary_floating_point
make
./program
```

(c) What is the result and what have you expected?

(d) Explain the behaviour.

(e) Use a `double` instead of the `float`. What do you observe?

Exercise 2.6: Binary fixed point number (coding)

We use the *Compositional Numeric Library (CNL)* library from `https://github.com/johnmcfarlane/cnl`. *Hint: The template contains already the library and the build is pre-configured within the `Makefile`.*

(a) Write a C++ program with a for loop (from 1 to 500). In every loop it adds 0.8 to a binary fixed point variable. At the end, print the result.
Use `RA_exercises/sheet_02/binary_fixed_point/binary_fixed_point.cpp` as a starting template.

(b) Follow the TODOs in `binary_fixed_point.c`.

(c) Compile your program using the provided Makefile and run it.

(d) What is the result and what have you expected?

(e) What happens if you change the precision from 7 digits to 14 digits?

Exercise 2.7: Decimal fixed point number (coding)

We use the *Decimal data type for C++* library from `https://github.com/vpiotr/decimal_for_cpp`. *Hint: The template contains already the library and the build is pre-configured within the `Makefile`.*

(a) Write a C++ program with a for loop (from 1 to 500). In every loop it adds 0.8 to a decimal fixed point variable. At the end, print the result.
Use `RA_exercises/sheet_02/decimal_fixed_point/decimal_fixed_point.c` as a starting template.

(b) Follow the TODOs in `decimal_fixed_point_solution.c`.

(c) Compile your program using the provided Makefile and run it.

(d) What is the result and what have you expected?