Assignment 2 COMP2111 18s1: Emirps

Kai Engelhardt

Revision: 1.1 of Date: 2018/04/16 04:33:19

This assignment is worth 15 marks and due before the logical end of week 8, that is, **Wednesday May 2nd, 12:59:59** local time Sydney. Assignments are done in pairs.

Problem Statement

"An *emirp* ('prime' spelled backwards) is a prime whose (base 10) reversal is also prime, but which is not a palindromic prime. The first few are 13, 17, 31, 37, 71, 73, 79, 97, 107, 113, 149, 157, ... (OEIS A006567)." (MathWorld)

In this assignment, we're going to find some more of them.

Tasks

1. Specify a function that upon input of a positive number found in the parameter n returns the n'th emirp.

2. Use the refinement calculus laws to formally derive an implementation of your specification. Exploit theorems from mathematics if you like, but clearly indicate when you do so and give references.

3. Translate your implementation into a C function with a prototype similar to

   ```
   unsigned long emirp(unsigned long n)
   ```

   if you stick to fixed width integers. (If you'd rather write an arguably better program that works even for numbers exceeding the size of an **unsigned long**, feel free to use gmp.) Add a main for reading in **unsigned long** n from stdin and printing the result unadorned to stdout, that is, using a format control string similar to "%lu\n" with printf (unless you use GMP, in which case you should use "%Zd\n" with gmp_printf).

4. Describe your solutions to tasks 1–3 in a LaTeX document that your tutor enjoys reading. In more detail:

   - State clearly what the requirements are.

- Justify that your formal specification captures the requirements.
- Describe how you derived the implementation from the specification. List all arising proof obligations (premises to Laws) and discharge them by proof. Properly reference any outside sources you use.
- Discuss and justify the changes made during the translation to C. Specify the limitations of your implementation should there be any.
- Run some tests of your C code on a CSE lab machine or login server and list runtimes for a representative sample of emirps. Use these tests to justify your choice of representation for the result.

Help

Reversing numbers in decimal representation is trivial to code and tedious to derive. Whence I provide a specification and C implementation of a reversing function (for both, **unsigned long** and mpz_t). The relevant files can be found here. The specification is

$$\textbf{proc}\ \mathit{reversen}(\textbf{value}\ n : \mathbb{N}, \textbf{result}\ r : \mathbb{N}) \cdot$$
$$\textbf{con}\ S : [10]^* \cdot r : \left[n = \sum_{i=0}^{c(n)} (S_i 10^{(c(n)-i)}) \wedge n > 0, r = \sum_{i=0}^{c(n)} (S_i 10^i) \right]$$

where $c(n) = \lfloor \log_{10} n \rfloor$.

Deliverables

`emirp.c` C source. Feel free to **#include** "reverse.h".

`Makefile` with a default target that builds the executable `emirp` from your C sources (and those provided).

`emirp.tex` is a LaTeX document with your names or student numbers mentioned in the `\author` command. It contains your task 4 solution.

Examples

Examples of the interaction with your C source are as follows. (Our shell prompt is `$` and user input is coloured red.)

```
$ make
$ ./emirp
1
13
$ ./emirp
2
```

```
17
$ ./emirp
23
733
$ time ./emirp
123456
12952207

real 6m9.933s
user 6m5.364s
sys 0m0.088s
$ make emirp.pdf
...
```

Submission Instructions

When submissions are enabled, the `give` command to be run is:

```
% 2111
% give cs2111 ass2 emirp.c Makefile emirp.tex
```

Do not submit any of the provided files except for a suitable `Makefile`. The subject-specific LaTeX style files as well as the C files `reverse.h` and `reverse.c` will be copied into the directory where your submission is built.

```
$Log: ass2.tex,v $
Revision 1.1  2018/04/16 04:33:19  kaie
Initial revision
```