Pretty Printing

Download a modified version of the MiniC project from `http://www.lsv.ens-cachan.fr/~schmitz/teach/2012_prog/part1/td4_pp/minic.tar.gz`. Unpack it using “`tar`”. You can start working on the file “`cprint.ml`”.

The purpose of this session is to write an XML pretty-printer for the ASTs of the MiniC project.

1 XML

XML, standing for *eXtensible Markup Language*, is a textual representation for ordered, labeled trees. It is therefore quite well adapted for representing our ASTs.

XML uses *tags* for its syntax. A node of an XML tree is of form `<tag>contents</tag>`. An empty tag `<tag></tag>` is more simply written `<tag/>`. The contents inside an opening tag and the corresponding closing tag can be other XML nodes or plain text. Since we want to describe a tree, any open tag has to be closed, and the elements have to be well-nested: `<h1>Summary of <em>Through the Looking-Glass</h1></em>` is incorrect, and one should write instead `<h1>Summary of <em>Through the Looking-Glass</em></h1>`.

An XML node can have associated *attributs*, written under the form `name="value"`. For instance, a *chapter* node could be refined thanks to an attribute `<chapter class="bibliography"> contents </chapter>`.

There are a few *entities* to make up for reserved characters: `&lt;` for < (*less than*), `&gt;` for > (*greater than*), `&amp;` for & (*ampersand*).

There are many *dialects* of XML, each specialized to a specific task. For instance, XHTML is used for web documents, SVG for vector graphics, OpenDocument for office documents, etc.

Exercise 1. Write a function “pprint_locator” of type $\text{string} * \text{int} * \text{int} * \text{int} * \text{int} \to \text{unit}$ that prints a formatted string for a locator on the standard output. More precisely, we expect an output of form

```
file="tests/cat.c" first-line="26" first-column="8" last-line="26" last-column="9"
```

Exercise 2. Write a function “pprint_vdecls” of type $\text{Cparse.var\_declaration list} \to \text{unit}$ that prints an XML string on the standard output for an input AST.

For instance,

```
int
main (int argc, char **argv)
{
  int a;
  return a + 1;
}
```

can be pretty-printed as (with indentation and newlines manually added)

```
<cfun name="main"
  file="test.c" first-line="2" first-column="0" last-line="2" last-column="4">
  <args>
    <cdecl name="argc"
      file="test.c" first-line="2" first-column="10" last-line="2" last-column="14"/>
    <cdecl name="argv"
      file="test.c" first-line="2" first-column="23" last-line="2" last-column="27"/>
  </args>
  <cblock
    file="test.c" first-line="3" first-column="0" last-line="6" last-column="1">
    <cdecl name="a"
      file="test.c" first-line="4" first-column="6" last-line="4" last-column="7"/>
    <creturn
      file="test.c" first-line="5" first-column="2" last-line="5" last-column="14">
      <add
        file="test.c" first-line="5" first-column="9" last-line="5" last-column="14">
        <var name="a"
          file="test.c" first-line="5" first-column="9" last-line="5" last-column="10"/>
        <cst value="1"
          file="test.c" first-line="5" first-column="13" last-line="5" last-column="14"/>
      </add>
    </creturn>
  </cblock>
</cfun>
```

(You can run your pretty-printer using "`./mcc -A file.c`".)

2 Format

We are going to use the *Format* module of OCaml, for which you will find a tutorial at `http://caml.inria.fr/resources/doc/guides/format.en.html` and documentation using "`man Format`".

Exercise 3. Revise your code using the *Format* module to indent automatically your XML output. Here is the output I get (this time not manually edited):

```
<cfun name="main" file="tests/test.c" first-line="2" first-column="0"
      last-line="2" last-column="4">
  <args>
    <cdecl name="argc" file="tests/test.c" first-line="2" first-column="10"
           last-line="2" last-column="14"/>
    <cdecl name="argv" file="tests/test.c" first-line="2" first-column="23"
           last-line="2" last-column="27"/>  </args>
  <cblock
    file="tests/test.c" first-line="3" first-column="0" last-line="6"
    last-column="1">
    <cdecl name="a" file="tests/test.c" first-line="4" first-column="6"
           last-line="4" last-column="7"/>
    <creturn
      file="tests/test.c" first-line="5" first-column="2" last-line="5"
      last-column="14">
      <add
```

```
            file="tests/test.c" first-line="5" first-column="9" last-line="5"
            last-column="14">
            <var name="a" file="tests/test.c" first-line="5" first-column="9"
                 last-line="5" last-column="10"/>
            <cst value="1" file="tests/test.c" first-line="5" first-column="13"
                 last-line="5" last-column="14"/></add></creturn></cblock>
</cfun>
```