Software Compiler Assignment on:

Switch Statement and LLVM

The Software Compiler course exam is composed by two parts. One is a written test, the other is an homework, to be terminated before course last class. The written test contributes with the 40% to the whole grade, while the homework contributes with the remaining 60%. To pass the whole exam, you must get a pass grade from both the test and the homework.

During the lab classes, we show you the ACSE compiler and the associated assembler [3]. They must be used as starting point for your homework. During the last class, you must present your work, showing your edits, giving a brief demo, and responding at some questions given by course lecturers.

Sources of ACSE can be found on the course site [4]. It is a tarball of the ACSE mercurial [2] repository. You are required to version your code with mercurial. A copy of your edits must be submitted to course lectures in the form of a patch with respect to the version of ACSE you have used as starting point.

Assignment

You are required to modify both the front-end of the compiler and the back-end.

Front-end

Control statement support in LANCE is poor. For this reason, a simplified version of the `switch` statement is needed. Figure 1 shows an example of the simplified statement.

With respect to the C version of the statement, the following restrictions are applied:

- `break` statement is not supported
- `default` case is not supported
- at the end of each case, control is transferred at `switch` end, thus, ...
- ... no support for `case` fall-through

You are required to modify the ACSE compiler in order to recognize the `switch` statement and to generate code for the ACSE assembler.

```
switch(a) {
case 0:
  ...
case 1:
  ...
}
```

Figure 1: A minimalistic `switch` statement

Back-end

LLVM [1] is a framework for developing compilers. It provides different libraries helping developing a state-of-the-art compiler. In particular, it exposes a rich code generation framework.

Exploiting LLVM in building a compiler is not a difficult task. The framework exposes functions allowing to create modules, functions, basic blocks, ... The language used by LLVM to define such things is called *LLVM bitcode* and is very similar to ACSE assembly.

You are required to write a new back-end for the ACSE compilation system, targeting LLVM bitcode. The backed must be a new ACSE tool that reads an ACSE assembly input file, translates it into LLVM bitcode, and writes the result in a new file.

The `asm` tool can be used as starting point for your backend. You can use LLVM libraries, C or C++ interface, in your work.

Creating and Applying Patches

The ACSE distribution is already provided as a mercurial repository. See [5] for the basic mercurial commands.

Assuming that your work has been committed with revision X, to generate a patch use the following command:

```
hg diff -r 0 -r X
```

Assuming the patch has been saved in the file `patch.diff`, use the following command to apply it to a freshly unpacked ACSE source tree:

```
hg import patch.diff -m "Applied patch"
```

The submitted project must successfully pass the above process, i.e. before submitting your patch check that it can be applied to a freshly unpacked ACSE source tree including compilation and testing process where applicable.

References

[1] LLVM. http://llvm.org, 2011.

[2] Mercurial. http://mercurial.selenic.com, 2011.

[3] A. Di Biagio and G. Agosta. Advanced Compiler System for Education. http://compilergroup.elet.polimi.it, 2008.

[4] Formal Languages and Compilers Group. Software Compilers. http://compilergroup.elet.polimi.it, 2010.

[5] J. Spolsky. Hg Init: a Mercurial Tutorial. http://hginit.com, 2011.