Compilers

Lab 1

Piotr Błaszyński

3rd March 2022

Tasks:

- download all the example files: `http://detox.wi.ps.pl/pb/tk/en/all.zip`
- test compile and run code from directories z1, z2, z3 (instructions at the end),
- run it by passing inX.txt as an argument, the files are in the corresponding directories,
- additionally test the operation of the analyzer in the interactive mode,
- modify the input files in z2 so that they both "compile", or modify the analyzer file (z2.l), this is a task to become familiar with the lexical analyzer framework, check which lexemes are supported by the presented lexical analyzer and remove the unsupported elements from the source file,
- become familiar with the contents of the Makefile in z3,
- download the MIPS processor emulator MARS `http://courses.missouristate.edu/KenVollmar/mars/download.htm` (alternatively QtSpim), run the emulator through a java virtual machine (example at the end),
- run the example program `http://courses.missouristate.edu/KenVollmar/mars/CCSC-CP%20material/row-major.asm`

Homework: Develop a **own** language project. It is easier to make a compiler of a traditional language, you should avoid constructions from esoteric languages. In addition to the constructs themselves, please prepare some program files in your language (testing the constructs below). Required constructions (grade - requirements):

- 3.0 - int and double types - constants (literals) and variables of these types, arithmetic expressions (=, +, -, *, /), simple if (without else and at most one nesting), writing int and double, typing int and double from the console
- 3.5 - for loop (higher grade) or while, string type (only output, no other operations on this type),
- 4.0 - for loop (no need for while), one-dimensional arrays, complex if (lots of nesting) with else,
- 4.5 - multidimensional arrays,
- 5.0 - dynamic allocation of one-dimensional arrays or simple functions (procedures without parameters) (or functions with parameters and return value),

I am checking the assignment in a week, I am accepting electronic version only. You can send email earlier (pblaszynski@zut.edu.pl).

Method of calling compiler in the first two directories:

```
flex zX.l #the file lex.yy.c is created
gcc -c lex.yy.c #the file lex.yy.o is created
gcc lex.yy.o -o my_compiler_name -ll # my_compiler_name
   file is created
```

Running:

```
./name_my_compiler #running in interactive mode
./name_my_compiler < in1.txt #Starting with passing the
   file in1.txt on stdin
./name_my_compiler in1.txt #Starting with feeding in1.
   txt in argv[1] (needs to be handled in main function)
```

We end the interactive mode only by pressing:

```
Ctrl+D
```

We use the Makefile by typing:

```
make
```

Calling bison (we use Makefile, but you need to know what is going on in it):

```
bison -d def.y #def.tab.c and def.tab.h (in C++ version
   def.tab.cc and def.tab.hh) are created
gcc -c def.tab.c #the file def.tab.o is created
gcc lex.yy.o def.tab.o -o name_my_compiler -ll #  file
   name_my_compiler is created
```

Example of running MARS emulator (you can also run in graphical environment, you need to set file permissions for execution beforehand - chmod +x Mars4_5.jar):

```
java -jar Mars4\_5.jar
```