Compiler Construction/Project Milestone 1 Parser for MiniC

Fang Han
fh643@nyu.edu

This document serves as the documentation of ***Pr1FangHan.hx***.

1. General

A. Development

- HACS: `1.1.21`
- Environment: `crunchy5.cims.nyu.edu`

B. Project Partner

- Yongyan Rao

2. Problems resolved

1. **1.1Definition(tokens)** - 1 see 4.3.A below (Integer part)
2. **1.1Definition(tokens)** - 3 see 4.3.C below (String part)
3. **1.1Definition(tokens)** - comments do not nest: my understanding is that if a comment includes /* or */ in it, e.g. `/* hello /*hacs */` , it should be deemed illegal and the compiler would issue an error message. (See 5.5 below)
4. **1.2Definition(expressions)** -
   - precedence and association rules are expressed and commented directly in the `sort Exp` part of **Pr1FangHan.hx**.
   - Function parameter rule: "The grammar uses '. . . ' to indicate that function calls permit any number of ,-separated parameters (including none)" is fulfilled by expanding the form `Expr(Expr, ..., Expr)` recursively with two more production rules:
     - `sort ExpList | ⟦ ⟨Exp⟩ ⟨ExpListTail⟩ ⟧ | ⟦⟧ ;`
     - `sort ExpListTail | ⟦ , ⟨Exp⟩ ⟨ExpListTail⟩ ⟧ | ⟦⟧ ;`
5. **1.3Definition(types)** - similar to the solution described above.
   - Precedence is added onto sort Type production rules.
   - Solution to the parameter type lists is identical to the function parameter rule above:
     - `sort TypeList | ⟦ ⟨Type⟩ ⟨TypeListTail⟩ ⟧ | ⟦⟧;`
     - `sort TypeListTail | ⟦ , ⟨Type⟩ ⟨TypeListTail⟩ ⟧ | ⟦⟧;`
6. **1.4Definition(statements)** - dangling else problem: solved by having another rule to further produce the tail of an if statement
   - `sort IfTail | ⟦ ⟨Stmt⟩ else ⟨Stmt⟩ ⟧ | ⟦ ⟨Stmt⟩ ⟧ ;`
7. **1.4Definition(statements)** - left of the assignment (=) must be an "l-value": resolved by clarify what can appear to the left of (=) in sort stmt:

```
-     |  ⟦ ⟨ID⟩ = ⟨Exp⟩ ; ⟧
-     |  ⟦ ∗ ⟨Exp⟩ = ⟨Exp⟩ ; ⟧
```

8. **1.5Definition(declaration)** - " '. . . ' indicates that there may be zero or more comma-separated formal parameters", This is achieved by having another rule to make Parameterlist and properly incorporate it into both `sort Declaration` and `sort main`. It should be pretty clear in the code.
9. **1.6Definition(program)** - at least one with a function main: See 4.7 below.

3. Discrepancies and future works

1. 1.1 -1 keywords words are not allowed in Identifier (see 4.3.B below) — since we weren't given the set of keywords, this rule isn't fulfilled in the parser.
2. 2.6 - at least one with main or exactly one with main?

4. Work log & Collaboration

1. I first read through the Project Milestone specification file ***pr1.pdf***, made sure I understand what to expect from the project and what the following steps are.
2. I then studied in detail again the ***HACS manual***, especially where it differs from the regex and CFG we learnt from the textbook.
3. I then started writing the ***pr1FangHan.hx*** file. First I worked on the token part.
   A. ***Integer*** is the easiest one. It simply is a sequence of digits which I expressed using RE as

```
token INT | ⟨Digit⟩+ ;
token fragment Digit | [0-9] ;
```

Introducing token fragment instead of directly write it into token INT is for increasing reusability.

   B. **Identifier** is more complicated. Limited by the stage of compilation we're faced with right now, I wasn't able to exclude the set of keywords from ID, which is addressed as one of the discrepancies in part 3.1 of this file.
   C. **String** is the most complicated.
   - ignore newline is expressed in `[^\"\\\n]`
   - byte encoding for characters is expressed by: `token fragment Octal | [0-7] | [0-7][0-7] | [0-7][0-7][0-7];`
   - Under a similar light, I expressed the rule for tokenizing "hex" by: `token fragment Hex | [0-9A-Fa-f] ;`
   - All the above and the "nt" rule is combined into `token fragment Escape | [\n\\nt"] | "x" ⟨Hex⟩ ⟨Hex⟩ | ⟨Octal⟩;`

4. Then I moved on to writing CFG. Each of the definition groups would correspond to a set of production rules. This part is pretty straight forward. Except that of

course there'll be lots of bugs waiting··· Some problems I managed to resolve are described in section 2 above. More detailed comments on code can be found in the file **Pr1FangHan.hx** itself.

5. After long hours of debugging, my **.hx** file still couldn't go through the HACS compiler. I got together with my project partner, Yongyan, who consequently told me that our TA— Daniel told him the use of kleene stars might be confusing to HACS. The two of us decided to get rid of all kleene stars in the syntax analysis section and see if it solves the problem.
6. It turned out for both of us that making sure kleene star doesn't show up in the syntax analysis section would solve the compilation problem. Ta-da!!

```
[fh643@crunchy5 samples]$ $HOME/.hacs/bin/hacs Pr1FangHan.hx
HACS 1.1.23
rm Pr1FangHan.hxraw
rm /home/fh643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHanParser.jj /home/fh643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHan.hxraw /home
/fh643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHanParser.pg /home/fh643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHanParser.java /home/fh
643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHan.prep
rm /home/fh643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHanHx.pgtemplate /home/fh643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHanHx.jj /h
ome/fh643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHanHx.pg /home/fh643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHan.hxraw /home/fh643/ha
cs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHanHx.java /home/fh643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHan.prep
rm /home/fh643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHanEmbed.jj /home/fh643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHanEmbed.pg /hom
e/fh643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHan.hxraw /home/fh643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHanEmbed.java /home/fh643
/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHan.prep
rm /home/fh643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHan.hxraw /home/fh643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHan.hxprep /home/f
h643/hacs/pr1/samples/build/org/crsx/hacs/samples/Pr1FangHan.env
HACS: Generating user script Pr1FangHan.run (can be moved)...
```

7. After having generated the first working **Pr1FangHan.run** file, I ran it against the sample tests and realized that I didn't successfully code in the last rule which is to have at least one main function. My partner Yongyan recommended that I make another token rule which differentiate `main function`s from other functions. The fix I made with the help of Yongyan does make sure that if a program doesn't contain main, it won't pass the parser. But I'm not sure whether having more than one "main" is supposed to be legal, which is presented as one of the discrepancies I'd like to further work on in section 3.

5. Tests

1. test 1 — strings.MC — passed

```
[fh643@crunchy5 samples]$ ./Pr1FangHan.run --sort=Program strings.MC
   function int main  ( )  {   var  int  dummy ;    dummy =   puts  (   "This string has a \" and \n and \
then it continues with \x68ex and \t\ttabs"    )  ;    return  0  ;     }
```

2. test 2 — strlen.MC — passed

```
[fh643@crunchy5 samples]$ ./Pr1FangHan.run --sort=Program strlen.MC
   function  int  strlen  (  *  char   string   )  {   var  int  length ;    length =  0  ;    while (  *  string   )  {   length =   length  +  1   ;    strin
g =   string  +  1   ;    }    return  length  ;      }    function int main  (  *  char   input   )  {   var  int  dummy ;    dummy =   puts  (   "The len
gth of the string is "    )  ;    dummy =   puti  (   strlen  (   input    )   )  ;    return  0  ;      }
```

3. test 3 — strcpy.MC — passed

```
[fh643@crunchy5 samples]$ ./Pr1FangHan.run --sort=Program strcpy.MC
   function  *  char   strcpy  (  *  char   string   )  {   var  int  length ;    length =   strlen  (   string    )  ;   var  *  char   copy ;    copy =   ma
lloc  (    length  +  1     )  ;   var  *  char   p ;    p =  copy  ;    while (  *  string   )  {   *  p  =  *  string   ;   string =   string  +  1   ;
p =   copy  +  1   ;     }    *  p  =  0  ;    return  copy  ;      }    function int main  (  *  char   input   )  {   var  int  dummy ;    dummy =
 puts  (   "The copy of the string is "    )  ;    dummy =   puts  (   strcpy  (   input    )   )  ;    return  0  ;      }
```

4. test 4 — comments.MC — passed

```
[fh643@crunchy5 samples]$ ./Pr1FangHan.run --sort=Program comments.MC
   function int main  ( )  {   return  0  ;    }
```

5. test 5 — comments.badMC — not pass

```
[fh643@crunchy5 samples]$ ./Pr1FangHan.run --sort=Program comments.badMC
Exception in thread "main" java.lang.RuntimeException: net.sf.crsx.CRSException: Encountered " "*" "* "" at line 3, column 1.
Was expecting one of:
    "function int main" ...
    "function" ...

        at net.sf.crsx.generic.GenericEvaluator.compute(Unknown Source)
        at net.sf.crsx.generic.GenericEvaluator.compute(Unknown Source)
        at net.sf.crsx.generic.GenericCRS.evaluateArguments(Unknown Source)
        at net.sf.crsx.generic.GenericCRS.loadDirective(Unknown Source)
        at net.sf.crsx.generic.GenericCRS.loadDirective(Unknown Source)
        at net.sf.crsx.generic.GenericCRS.load(Unknown Source)
        at net.sf.crsx.run.Crsx.realMain(Unknown Source)
        at net.sf.crsx.run.Crsx.realMain(Unknown Source)
        at net.sf.crsx.run.Crsx.main(Unknown Source)
Caused by: net.sf.crsx.CRSException: Encountered " "*" "* "" at line 3, column 1.
Was expecting one of:
    "function int main" ...
    "function" ...

        at org.crsx.hacs.samples.Pr1FangHanParser.parse(Pr1FangHanParser.java:115)
        at net.sf.crsx.generic.GenericFactory.parse(Unknown Source)
        ... 9 more
```

6. test 6 — nonlvalue.badMC — not pass

```
[fh643@crunchy5 samples]$ ./Pr1FangHan.run --sort=Program nonlvalue.badMC
Exception in thread "main" java.lang.RuntimeException: net.sf.crsx.CRSException: Encountered " <T_INT> "2 "" at line 2, column 3.
Was expecting one of:
    "*" ...
    "if" ...
    "return" ...
    "var" ...
    "while" ...
    "{" ...
    "}" ...
    <T_ID> ...

        at net.sf.crsx.generic.GenericEvaluator.compute(Unknown Source)
        at net.sf.crsx.generic.GenericEvaluator.compute(Unknown Source)
        at net.sf.crsx.generic.GenericCRS.evaluateArguments(Unknown Source)
        at net.sf.crsx.generic.GenericCRS.loadDirective(Unknown Source)
        at net.sf.crsx.generic.GenericCRS.loadDirective(Unknown Source)
        at net.sf.crsx.generic.GenericCRS.load(Unknown Source)
        at net.sf.crsx.run.Crsx.realMain(Unknown Source)
        at net.sf.crsx.run.Crsx.realMain(Unknown Source)
        at net.sf.crsx.run.Crsx.main(Unknown Source)
```

7. test 7 — nomain.badMC — not pass

```
[fh643@crunchy5 samples]$ ./Pr1FangHan.run --sort=Program nomain.badMC
Exception in thread "main" java.lang.RuntimeException: net.sf.crsx.CRSException: Encountered "<EOF>" at line 3, column 2.
Was expecting one of:
    "function int main" ...
    "function" ...

        at net.sf.crsx.generic.GenericEvaluator.compute(Unknown Source)
        at net.sf.crsx.generic.GenericEvaluator.compute(Unknown Source)
        at net.sf.crsx.generic.GenericCRS.evaluateArguments(Unknown Source)
        at net.sf.crsx.generic.GenericCRS.loadDirective(Unknown Source)
        at net.sf.crsx.generic.GenericCRS.loadDirective(Unknown Source)
        at net.sf.crsx.generic.GenericCRS.load(Unknown Source)
        at net.sf.crsx.run.Crsx.realMain(Unknown Source)
        at net.sf.crsx.run.Crsx.realMain(Unknown Source)
        at net.sf.crsx.run.Crsx.main(Unknown Source)
```

8. test 8 — doubleelse.badMC — not pass