Remarks on Assignment 3

Assignment 3 has given you the ability to practice-, and has given us the ability to evaluate-, your understanding of the course content; as well as your knowledge, skills, and competences as highlighted in Appendix C (page 9). Highlights in this color cover what was covered by Assignments 1, 2, and 3; while this color covers what was covered by Assignment 3 only.

These remarks constitute our evaluation of your work, and feedback which may come in handy in attaining a good-to-excellent final grade. The remarks do not count towards your final grade. Furthermore, this evaluation has not been internally, or externally examined, and may be biased by the opinions of the teaching assistant (TA) marking your work.

This work is marked by Oleks. In general, the TA here begins with 100 points, and then gradually deduct foul points from that. You may receive positive points for some extraordinary work. However, the working assumption is that you've done excellent work, until proven otherwise. You may end up with more than 100 points. In the end, your final score is projected onto the 6-point marking scheme demanded by the course instructor. Other TAs may be use other schemes.

Appendix B (page 8) lists the files reviewed as the base for these remarks. You can use this list to verify that you are looking at remarks on the *intended revision* of *your* submission.

Appending A (page 3) includes the output from OnlineTA for your submission.

1 Main Task

-2 Define higher-order parser combinators to avoid repetitive parsing tasks.

For instance, you repeat `lexeme (string ...)` quite a lot. Why not declare a function for it?

-2 In your ReadP-based parser you sometimes use the left-biased `<++` instead of the unbiaed `+++` (or `<|>`) without stating why in your report. Deviating from the pure formalism of context-free grammars (in which the order of alternatives is not significant) makes your parser less robust, and harder to argue correct. If you are doing it to improve efficiency, most likely it won't make a visible difference; if it does, you're probably compensating for a serious performance bug elsewhere, and you should try to fix that one instead.

2 Report

-4 Parser combinators allow the implementation to closely resemble a BNF grammar which you can denote in the report. You haven't included a modified grammar in the report, but you have also made substantial changes to it, compared to the version presented in the assignment text.

It can be a good idea to try to write down the modified grammar using BNF.

-2 The report failed mention implementation details, as disucssed above.

-5 You include screenshots of your code in the report. Screenshots seem easy at first; but, they come with a number of issues:

1. Should your code change, you would need to manually re-take your screenshots.
2. The scale of your screenshots will vary, unless you crop your screenshots in a systematic manner.
3. The font of your code editor, may not fit the font of your report, cutting the eye of the reader.
4. You may prefer a dark mode in your code editor, but that does not sit well with a printed document—dark backgrounds tend to be a waste of ink for little benefit to the reader.

It is substantially better to use code listings in your report. For instance, to use the standard `listings` package in LaTeX.

2.1 Efficiency

!! In this assignment, one source of inefficiency is insufficient disambiguation of the grammar. This may lead to more backtracking than necessary during parsing.

Final Score

Points: **85**
Score: **5**
Verdict: **Passed**

A OnlineTA Output

```
Checking files...

You have a code/part1/src/WarmupReadP.hs. Good.
You have a code/part1/src/WarmupParsec.hs. Good.
You have a code/part2/src/BoaAST.hs. Good.
You have a code/part2/src/BoaInterp.hs. Good.
You have a code/part2/src/BoaParser.hs. Good.

You have a code/timesheet.txt. Good.
  Timesheet looks OK



Building part 1
OK


Building part 2
OK

Running hlint on part 1 (ReadP)
Running hlint...
No hints

Running hlint on part 1 (Parsec)
Running hlint...
No hints

Running hlint on part 2
Running hlint...
No hints



Running some dynamic tests for Part 1...
Warmup tests
  ReadP variant
    numeral 0:         OK
    numeral 123:       OK
    negation:          OK
    addition:          OK
    subtraction:       OK
    parens:            OK
    spaced string:     OK
    *trailing garbage: OK
    *letters:          OK
    *double negation:  OK
    *mixed ops:        OK
    *prefix plus:      OK
  Parsec variant
    numeral 0:         FAIL
      Exception: Prelude.undefined
      CallStack (from HasCallStack):
        error, called at libraries/base/GHC/Err.hs:75:14 in base:GHC.Err
        undefined, called at src/WarmupParsec.hs:20:15 in main:WarmupParsec
    numeral 123:       FAIL
      Exception: Prelude.undefined
      CallStack (from HasCallStack):
        error, called at libraries/base/GHC/Err.hs:75:14 in base:GHC.Err
        undefined, called at src/WarmupParsec.hs:20:15 in main:WarmupParsec
    negation:          FAIL
      Exception: Prelude.undefined
      CallStack (from HasCallStack):
```

```
        error, called at libraries/base/GHC/Err.hs:75:14 in base:GHC.Err
        undefined, called at src/WarmupParsec.hs:20:15 in main:WarmupParsec
    addition:          FAIL
      Exception: Prelude.undefined
      CallStack (from HasCallStack):
        error, called at libraries/base/GHC/Err.hs:75:14 in base:GHC.Err
        undefined, called at src/WarmupParsec.hs:20:15 in main:WarmupParsec
    subtraction:       FAIL
      Exception: Prelude.undefined
      CallStack (from HasCallStack):
        error, called at libraries/base/GHC/Err.hs:75:14 in base:GHC.Err
        undefined, called at src/WarmupParsec.hs:20:15 in main:WarmupParsec
    parens:            FAIL
      Exception: Prelude.undefined
      CallStack (from HasCallStack):
        error, called at libraries/base/GHC/Err.hs:75:14 in base:GHC.Err
        undefined, called at src/WarmupParsec.hs:20:15 in main:WarmupParsec
    spaced string:     FAIL
      Exception: Prelude.undefined
      CallStack (from HasCallStack):
        error, called at libraries/base/GHC/Err.hs:75:14 in base:GHC.Err
        undefined, called at src/WarmupParsec.hs:20:15 in main:WarmupParsec
    *trailing garbage: FAIL
      Exception: Prelude.undefined
      CallStack (from HasCallStack):
        error, called at libraries/base/GHC/Err.hs:75:14 in base:GHC.Err
        undefined, called at src/WarmupParsec.hs:20:15 in main:WarmupParsec
    *letters:          FAIL
      Exception: Prelude.undefined
      CallStack (from HasCallStack):
        error, called at libraries/base/GHC/Err.hs:75:14 in base:GHC.Err
        undefined, called at src/WarmupParsec.hs:20:15 in main:WarmupParsec
    *double negation:  FAIL
      Exception: Prelude.undefined
      CallStack (from HasCallStack):
        error, called at libraries/base/GHC/Err.hs:75:14 in base:GHC.Err
        undefined, called at src/WarmupParsec.hs:20:15 in main:WarmupParsec
    *mixed ops:        FAIL
      Exception: Prelude.undefined
      CallStack (from HasCallStack):
        error, called at libraries/base/GHC/Err.hs:75:14 in base:GHC.Err
        undefined, called at src/WarmupParsec.hs:20:15 in main:WarmupParsec
    *prefix plus:      FAIL
      Exception: Prelude.undefined
      CallStack (from HasCallStack):
        error, called at libraries/base/GHC/Err.hs:75:14 in base:GHC.Err
        undefined, called at src/WarmupParsec.hs:20:15 in main:WarmupParsec

12 out of 24 tests failed (0.00s)
Warning:
  Some tests for Part 1 failed, please comment on this in your report


Running some dynamic tests for Part 2...
Boa Parser tests
  Lexical issues
    Complex terminals
      identifiers
        simple:                     OK
        complex:                    OK
        underscore:                 OK
        function:                   OK
        non-Boa keyword:            OK
        miscapitalized keyword:     OK
        leading keyword:            OK
        trailing keyword:           OK
        *keyword RHS:               OK
        *keyword LHS:               OK
        *keyword call:              OK
      numbers
```

```
    simple:                     OK
    negative:                   OK
    zero:                       OK
    negative zero:              OK
    *plus-signed:               OK
    *negative spaced:           OK
    *leading zero:              OK
    *hex:                       OK
    *spaced:                    OK
  strings
    simple:                     OK
    empty:                      OK
    special chars:              FAIL
      onlinetatests/Tests.hs:22:
      Parsing: "'\"#~ '"
      expected: Right [SExp (Const (StringVal "\"#~ "))]
       but got: Right [SExp (Const (StringVal "\"#~"))]
    simple escapes:             OK
    escaped newlines:           FAIL
      onlinetatests/Tests.hs:22:
      Parsing: "'a\\\n b\\n\\\nc\\\n\\nd'"
      expected: Right [SExp (Const (StringVal "a b\nc\nd"))]
       but got: Left "cannot parse"
    *newline:                   FAIL
      onlinetatests/Tests.hs:30:
      Parsing: "'\n'"
      expected: Left "<message>"
       but got: Right [SExp (Const (StringVal ""))]
    *tab:                       FAIL
      onlinetatests/Tests.hs:30:
      Parsing: "'\t'"
      expected: Left "<message>"
       but got: Right [SExp (Const (StringVal ""))]
    *DEL:                       OK
    *NUL:                       OK
    *quote:                     OK
    *quotes:                    OK
    *unfinished:                OK
    *bad escape (space):        OK
    *bad escape (tab):          OK
    *bad escape (dquote):       OK
    *bad escape (t):            OK
    *bad escape (N):            OK
    *bad escape (at end):       OK
  Whitespace & comments
    around ident:               OK
    around num:                 OK
    around string:              OK
    around keyword:             OK
    around symbols:             OK
    tabs and newlines:          OK
    interspersed:               FAIL
      onlinetatests/Tests.hs:22:
      Parsing: " x = ( 2 ) ; print ( None == [ y , z ] , not u ) "
      expected: Right [SDef "x" (Const (IntVal 2)),SExp (Call "print" [Oper Eq (Const NoneVal) (List
 [Var "y",Var "z"]),Not (Var "u")])]
       but got: Left "cannot parse"
    not needed:                 FAIL (0.01s)
      onlinetatests/Tests.hs:22:
      Parsing: "[(x)not\tin(not(y)),[(x)for\ty\tin[z]if(u)]]"
      expected: Right [SExp (List [Not (Oper In (Var "x") (Not (Var "y"))),Compr (Var "x") [CCFor "y"
 (List [Var "z"]),CCIf (Var "u")]])]
       but got: Left "cannot parse"
    comments:                   OK
    empty comments:             OK
    comment at eof:             OK
    comment as separator:       OK
    large whitespace:           FAIL
      onlinetatests/Tests.hs:22:
      Parsing: "                              not                             1
```

```
          "
       expected: Right [SExp (Not (Const (IntVal 1)))]
        but got: Left "cannot parse"
     many comments:                 OK
     *whitespace in symbol:         OK
     *missing space before in:      OK
     *missing space after in:       OK
     *missing space between not in: OK
     *missing space compr body:     OK
     *missing space compr for:      FAIL
       onlinetatests/Tests.hs:30:
       Parsing: "[x fory in z]"
       expected: Left "<message>"
        but got: Right [SExp (Compr (Var "x") [CCFor "y" (Var "z")])]
     *missing space compr in:       OK
     *missing space compr if:       OK
     *comment in string:            FAIL
       onlinetatests/Tests.hs:30:
       Parsing: "'#foo\n'"
       expected: Left "<message>"
        but got: Right [SExp (Const (StringVal "#foo"))]
  Syntactic issues
    General grammar
      Expressions
       misc:                        OK (0.46s)
       all arith ops:               OK (0.03s)
       all plain relational ops:    OK
       all negated relational ops:  OK
       call 0 args:                 OK
       call 1 arg:                  OK
       nested calls:                OK
       strange ranges:              OK (0.09s)
       comprehension:               FAIL
         onlinetatests/Tests.hs:22:
         Parsing: "[x for y in z if u for a in []]"
         expected: Right [SExp (Compr (Var "x") [CCFor "y" (Var "z"),CCIf (Var "u"),CCFor "a" (List
    [])])]
          but got: Left "cannot parse"
       comprehension for ynot in:   OK
       comprehension for in in:     OK
       deep parens:                 TIMEOUT (1.04s)
         Timed out after 1s
       deep brackets:               TIMEOUT (1.06s)
         Timed out after 1s
       *empty parens:               OK
       *deep parens ):              TIMEOUT (1.04s)
         Timed out after 1s
       *( deep parens:              TIMEOUT (1.08s)
         Timed out after 1s
       *deep brackets ]:            TIMEOUT (1.06s)
         Timed out after 1s
       *[ deep brackets:            TIMEOUT (1.06s)
         Timed out after 1s
       *comma:                      OK
       *slash:                      OK
       *diamond:                    OK
       *parens comma:               OK
       *call no parens:             OK
       *call double parens:         OK
       *comprehension if:           OK
       *comprehension if-for:       OK
       *comprehension bad for LHS:  OK
       *comprehension no body:      OK
       *comprehension for not in:   OK
       *comma comprehension:        OK
       *comprehension comma:        OK
      Statements/programs
       def:                         OK
       seq:                         OK
       def minus:                   OK
```

```
        def eq:                      OK
        equality test:               OK
        *empty:                      OK
        *empty stmt L:               OK
        *empty stmt R:               OK
        *empty stmt M:               OK
        *bad def LHS:                OK
        *parenthesized def:          OK
        *nested def:                 OK
    Disambiguation
      Precedence
        increasing:                  FAIL
          onlinetatests/Tests.hs:22:
          Parsing: "not x<y+z*u"
          expected: Right [SExp (Not (Oper Less (Var "x") (Oper Plus (Var "y") (Oper Times (Var "z") (Var
     "u")))))]
           but got: Left "cannot parse"
        falling:                     FAIL
          onlinetatests/Tests.hs:22:
          Parsing: "x%y-z>(not u)"
          expected: Right [SExp (Oper Greater (Oper Minus (Oper Mod (Var "x") (Var "y")) (Var "z")) (Not
     (Var "u")))]
           but got: Left "cannot parse"
        *not as arg:                 OK
      Associativity
        negation:                    FAIL
          onlinetatests/Tests.hs:22:
          Parsing: "not not x"
          expected: Right [SExp (Not (Not (Var "x")))]
           but got: Left "cannot parse"
        additive:                    OK
        multiplicative:              OK
        *relational:                 OK

19 out of 111 tests failed (7.11s)
Warning:
  Some tests for Part 2 failed, please comment on this in your report
dynamic/runtests.sh FAILED
---
I am not fully satisfied.
---
Found a bug? Are the messages too cryptic?
Let us know on Absalon or Discord.
```

B Reviewed Files

Table 1 (page 8) lists the files reviewed in the course of providing these remarks, with their SHA1 hashsums. If the hashsums do not correspond to the files you have, it *may* be because you are reading someone else's feedback; or you have made changes to the file since. You can obtain the SHA1 hashsum of a file by passing the path to it as an argument to the command `sha1sum` in a Unix-like environment.

The paths beginning with `unpacked/` correspond to what our archive extraction tool[1] extracted from the code archive you submitted.

SHA1 Hashsum	Path
18c437f0...314debc1	Report.pdf
bf5fe0fa...5110e380	code.zip
274bba63...b96a11ee	studenttests-on-own.txt
926bff87...91190471	studenttests-on-refParsec.txt
bae08c2b...d76f544e	studenttests-on-refReadP.txt
e3f9d490...0ccb33c5	unpacked/code/part1/package.yaml
ad568201...b43afa9b	unpacked/code/part1/src/WarmupParsec.hs
4644c5f5...1e2b1918	unpacked/code/part1/src/WarmupReadP.hs
85879dcf...5cae58a7	unpacked/code/part1/stack.yaml
16e4d96b...b6a39443	unpacked/code/part2/app/Main.hs
e79b61cb...bd49c5fc	unpacked/code/part2/examples/crash.ast
e60916ad...e87a1268	unpacked/code/part2/examples/crash.boa
bd8a6e62...64f41a6d	unpacked/code/part2/examples/crash.out
367fe36a...4ef75eb7	unpacked/code/part2/examples/misc.ast
ddbbca1c...c2c34c14	unpacked/code/part2/examples/misc.boa
b72ce17b...a1c2e053	unpacked/code/part2/examples/misc.out
d7ab1846...44e1768e	unpacked/code/part2/package.yaml
868e4b99...f910f67a	unpacked/code/part2/src/BoaAST.hs
2bfd0553...489415d0	unpacked/code/part2/src/BoaInterp.hs
36f2578c...d7a03c89	unpacked/code/part2/src/BoaParser.hs
85879dcf...5cae58a7	unpacked/code/part2/stack.yaml
eac92a17...da6d0ce0	unpacked/code/part2/tests/Test.hs
bfe9f307...c1dbc9f8	unpacked/code/timesheet.txt

Table 1: Reviewed files

[1] This is unintentionally left unspecified.

C Relationship to Course Description

University of Copenhagen - Courses

NDAA09013U Advanced Programming (AP)

EXPAND ALL

Volume 2022/2023

Education

MSc Programme in Computer Science

Content

The purpose of this course is to provide practical experience with sophisticated programming techniques and paradigms from a language-based perspective. The focus is on high-level programming and systematic construction of well-behaved programs.

The selection of the topics covered will be informed by current and emerging trends. The possible topics covered may include, but are not limited to:

- applicative (functional) programming
- concurrent programming
- declarative (logic) programming
- distributed programming
- generic programming
- parallel programming
- reactive programming

Learning Outcome

At course completion, the successful student will have:

Knowledge of

- Higher-level program structuring patterns for separating concerns.
- The basics of message-passing concurrency, and of how concurrent programming can be used for parallel programming.
- Program structuring principles and design patterns for dealing with software errors.
- Property-based testing of functions and stateful APIs

Skills to

- Use program structuring principles and design patterns, such as monads, to structure the code so that there is a clear separation of concerns.
- Use a parser combinator library to write a parser for a medium-sized language with a given grammar, including changing the grammar so that it is on an appropriate form.
- Implement simple concurrent/distributed servers using message passing, with appropriate use of synchronous and asynchronous message passing.
- Use program structuring principles and design patterns for making reliable distributed systems in the presence of software errors.

- Construct systematic test suites for programs and modules, including property-based tests where relevant.

Competences to

- Quickly acquaint themselves with advanced programming techniques, from academic literature and/or technical documentation.
- Use those techniques to solve challenging, realistic problems.
- Write correct, efficient, and maintainable programs with clear separation of concerns.
- Give an assessment of their own code, based on a systematic evaluation of correctness, selection of algorithms and data structures, error scenarios, and elegance.

Literature

See Absalon when the course is set up.

Recommended Academic Qualifications

Programming ability in at least two substantially different languages, and familiarity with basic software-development principles (modularity, abstraction, systematic testing, ...) will be expected.

It is strongly recommended to have some experience with functional programming, corresponding to Chapters 2, 4, 5, and 6 of "Learn You a Haskell for Great Good", for example.

General academic qualifications equivalent to a BSc degree in Computer Science, Software Development, or a closely related subject, are recommended.

Teaching and learning methods

Lectures, mandatory assignments, exercise labs.

Workload

Category	Hours
Lectures	29
Preparation	131
Practical exercises	21
Exam	25
Total	206

Feedback form

Written
Individual
Continuous feedback during the course of the semester

Sign up

Self Service at KUnet