Haskell Live

[11] Parsec

Bong Min Kim
e0327177@student.tuwien.ac.at

Christoph Spörk
christoph.spoerk@inode.at

Florian Hassanen
florian.hassanen@googlemail.com

Bernhard Urban
lewurm@gmail.com

17. Dezember 2010

Parsec Beispiel

Beispiel um den Code auszuführen:

```
$ cat foo.sf

func fa(a,b,c,) {
        var d := (b - 1) + 3;
        var x := 0;
        if ( (b - 1) = (c+a)) {
                print 1;
                ret 1;
                x := 5;
                ret x;
        } else {
                print 0;
        };
        print (a+b)+c;
        ret 5;
}

func fb(eins,) {
        ret not eins;
```

```
}

func main() {
        var bla := fa(2,3,4,);
        var blub := fa(0,2,1,);
        var bar := fb(5,);
        print bla;
        print blub;
        print bar;
}
$ ghc --make 11hl.lhs
[1 of 1] Compiling Main             ( 11hl.lhs, 11hl.o )
Linking 11hl ...
$ ./11hl foo.sf > foo.c
$ gcc -o foo foo.c
[...] Warnings
$ ./foo
0: 0
(a + b) + c: 9
1: 1
bla: 5
blub: 1
bar: 0
```

```
import Text.ParserCombinators.Parsec
import Text.ParserCombinators.Parsec.Expr
import System.Environment
import System.Exit
ident :: Parser String
ident = do
        x ← letter
        do {xs ← ident; return (x : xs)} < | > return [x]
ident' :: Parser String
ident' = do
        xs ← many1 letter
        return xs
ident'' :: Parser String
ident'' = do many1 letter
ident''' :: Parser String
ident''' = many1 letter
s01 = run ident''' "asdfg"
s02 = run ident''' "sdf134"
```

```
s03 = run ident''' "134"
run p input = case (parse p "" input) of
      Left err -> do {putStr "parse error at "; print err}
      Right x  -> do {putStr x}
separators :: Parser String
separators = do
      many separator
      <?> "separator"
separators1 :: Parser String
separators1 = do
      many1 separator
      <?> "separator"
separator :: Parser Char
separator = oneOf " \t\n"
s04 = run separators ""
s05 = run separators "     \t     \n    \t\t\t\n"
s06 = run separators1 ""
  -- "Entryparser"
program :: Parser String
program = functions
addSeps :: String -> String -> String
addSeps = flip (++)
functions :: Parser String
functions = do
      xs <- many1 function
      return $ concat $ map (addSeps "\n\n") xs
function :: Parser String
function = do
      fhead <- funchead
      separators; char '{'; separators
      fbody <- many1 stmt
      separators; char '}'; separators
      return $ fhead ++ "\n{\n\t" ++ (concat $ map (addSeps "\n\t") fbody) ++ "\r}"
      <?> "function"
funchead :: Parser String
funchead = do
      string "func"; separators1
      name <- ident; separators
      char '('
      p <- params
      char ')'; separators
```

```
    return $ "int " ++ name ++ "(" ++ (genParams "int " p) ++ ")"
params :: Parser [String]
params = many $ do {id <- ident; char ','; separators; return id}
genParams :: String -> [String] -> String
genParams _ [] = ""
genParams pre [x] = pre ++ x
genParams pre (x:xs) = pre ++ x ++ ", " ++ genParams pre xs
stmt :: Parser String
stmt = do
    separators;
    x <- do {
      try (assign) <|>
      try (vardef) <|>
      try (ifthenelse) <|>
      try (printVal) <|>
      try (retVal)
    }; char ';'; separators
    return $ x ++ ";"
vardef :: Parser String
vardef = do
    string "var"; separators1
    x <- assign
    return $ "int " ++ x
assign :: Parser String
assign = do
    x <- ident
    separators; string ":="; separators
    e <- expr
    return $ x ++ " = " ++ e
printVal :: Parser String
printVal = do
    string "print"; separators1
    x <- expr
    return $ "printf(\"%s: %d\\n\", \"" ++ x ++ "\"," ++ x ++ ")"
retVal :: Parser String
retVal = do
    string "ret"; separators1
    x <- expr
    return $ "return " ++ x
ifthenelse :: Parser String
ifthenelse = do {
    ifthen <- parseIf;
```

```
        try (do { separators; else1 ← parseElse; return (ifthen ++ else1); }) <|>
          return (ifthen);
}
parseElse :: Parser String
parseElse = do
        string "else"; separators;
        string "{"; separators;
        stmtsElse ← many stmt; separators;
        string "}";
        return $ "else {\n\t\t" ++ (concat $ map (addSeps "\n\t\t") stmtsElse) ++ "\r\t}"
parseIf :: Parser String
parseIf = do
        string "if"; separators;
        string "("; separators;
        condition ← expr; separators;
        string ")"; separators;
        string "{"; separators;
        stmtsIf ← many stmt; separators;
        string "}";
        return $ "if (" ++ condition ++ ") {\n\t\t" ++ (concat $ map (addSeps "\n\t\t") stmtsI
expr :: Parser String
expr = do
        x ← do {
          try (do { string "not"; separators1;
            t ← expr;
            return $ "!(" ++ t ++ ")"}) <|>
                try (specialOp '+') <|>
                try (operator '+') <|>
                try (specialOp '-') <|>
                try (operator '-') <|>
                try (specialOp '=') <|>
                try (operator '=') <|>
                term
        }; separators
        return x
operator :: Char → Parser String
operator op = do
        t ← term; separators
        char op; separators
        t2 ← term
        let retop = case op of
          '+' → "+"
          '-' → "-"
```

```
        '=' -> "=="
    return $ t ++ "  " ++ retop ++ "  " ++ t2
specialOp :: Char -> Parser String
specialOp op = do
    n1 <- number; separators
    char op; separators
    n2 <- number
    let ret = case op of
        '+' -> (str2Int n1) + (str2Int n2)
        '-' -> (str2Int n1) - (str2Int n2)
        '=' -> if (str2Int n1) == (str2Int n2) then 1 else 0
    return $ show ret
    where
        str2Int :: String -> Int
        str2Int = read
term :: Parser String
term = (do {
            char '('; separators;
            x <- expr; char ')';
            return $ "(" ++ x ++ ")"}) <|>
        number <|>
        try (fcall) <|>
        ident
fcall :: Parser String
fcall = do
    name <- ident; separators
    char '('; separators
    p <- many $ do {e <- expr; char ','; separators; return e}
    char ')'; separators
    return $ name ++ "(" ++ (genParams "" p) ++ ")"
number :: Parser String
number = many1 digit;
  -- tests
test = "func lol(b,c,) {var a := \t 3;\nret 4; }"
t1 = run function test
t2 = run program test
t3 = run program (test ++ "\nfunc lala() {xyz := 12; ret 0;}")
t4 = run program $ test ++ "\n "
"func foo(x,) {var a := 4; a := 5; ret a+(not ((x+x)-x)); ret 0; }\n"
"func main(){\n"
"var a:= 5; \n"
"if (a = foo(a,)) {print 400; a := 0;} else {print 200;};"
```

```
"print foo(a,);\n"
"ret 0;\n"
"}"
```

```
t5 = run ifthenelse "if (1 = 2) { \n lol := 3; } else { \n lala := 4; }"
t6 = run stmt "foo(a,b,)"
t7 = run expr "a + 5"
```

```
  -- main
main :: IO ()
main = getArgs >>= arg

arg ["-h"] = usage >> exit
arg ["-v"] = version >> exit
arg [] = t4
arg fs
      | (length fs) == 1 = output
      | otherwise = usage >> exit
      where output = (readAndRun (fs !! 0))

readAndRun :: FilePath -> IO ()
readAndRun fileName = do
  text <- readFile fileName
  run program text

usage = putStrLn "Usage: 11hl [-vh] [file]"
version = putStrLn "Parsec Example HaskellLive 0.1"
exit    = exitWith ExitSuccess
die     = exitWith (ExitFailure 1)
```