Chat-GPT 3.5

Can you please complete the statements function:

```
;;; ~~~ Statements ~~~

; stmt -> id = expr;
; | if (boolean) stmt;
; | while (boolean) linelist endwhile;
; | read id;
; | write expr;
; | goto id;
; | gosub id;
; | return;
; | break;
; | end;
(define stmt/p
  (or/p (do [identifier <- id/p]
            (string/p "=")
            [e <- expr/p]
            (string/p ";")
            (pure (list 'assign identifier e)))
        (do (string/p "if")
            (string/p "(")
            [cond <- boolean/p]
            (string/p ")")
            [s <- stmt/p]
            (string/p ";")
            (pure (list 'if cond s)))
        (do (string/p "while")
            (string/p "(")
            [cond <- boolean/p]
            (string/p ")")
            [ll <- linelist/p]
            (string/p "endwhile")
            (pure (list 'while cond ll)))
        ; etc....
        ))
```

Help me debug:
```
string:1:4: parse error
  unexpected:
  expected: :, letter, or number

0 | read A ;
```

Input file:
```
read A;
write A;
$$
```

A is a valid letter??

Is this correct?
```
(define id/p
  (do [first-char <- (or/p letter/p (char/p #\_))]
      [rest-chars <- (many/p (or/p letter/p digit/p))]
      (pure (string->symbol (list->string (cons first-char rest-chars)))))
  )
```

I think it's because the read parse doesn't account for a whitespace.
```
(define stmt/p
  (or/p
   ; id = expr;
   (do [identifier <- id/p]
       (string/p "=")
       [e <- expr/p]
       (string/p ";")
       (pure (list 'assign identifier e)))

   ; if (boolean) stmt;
   (do (string/p "if")
       (string/p "(")
       [cond <- boolean/p]
       (string/p ")")
       [s <- stmt/p]
       (string/p ";")
       (pure (list 'if cond s)))

   ; while (boolean) linelist endwhile;
```

```
(do (string/p "while")
    (string/p "(")
    [cond <- boolean/p]
    (string/p ")")
    [ll <- linelist/p]
    (string/p "endwhile")
    (pure (list 'while cond ll)))

; read id;
(do (string/p "read")
    [identifier <- id/p]
    (string/p ";")
    (pure (list 'read identifier)))

; write expr;
(do (string/p "write")
    [e <- expr/p]
    (string/p ";")
    (pure (list 'write e)))

; goto id;
(do (string/p "goto")
    [identifier <- id/p]
    (string/p ";")
    (pure (list 'goto identifier)))

; gosub id;
(do (string/p "gosub")
    [identifier <- id/p]
    (string/p ";")
    (pure (list 'gosub identifier)))

; return;
(do (string/p "return")
    (string/p ";")
    (pure 'return))

; break;
(do (string/p "break")
    (string/p ";")
    (pure 'break))

; end;
(do (string/p "end")
```

```
      (string/p ";")
      (pure 'end))
  ))
```

Why do I get this error
Line: Attempting to parse label: #<void>
Line: Attempting to parse stmt: (read A)
Line: Attempting to parse linetail: #<void>
Linelist: Attempting to parse line: ((#<void>) (read A) #<void>)
Line: Attempting to parse label: #<void>
Expr: Attempting to parse identifier: B
Line: Attempting to parse stmt: (write (B #<void>))
Line: Attempting to parse linetail: #<void>
Linelist: Attempting to parse line: ((#<void>) (write (B #<void>)) #<void>)
Line: Attempting to parse label: goto
string:3:6: parse error
  unexpected: 5
  expected: '_', break, end, gosub, goto, if, letter, read, return, while, whitespace, or write

3 | goto: 5+y;
         ^

I just realized the problem was I was trying to use a reserved word as a label

---

Thank you for helping me. Can you please provide some sample input files

---

Every file needs to end with $$, can you add those in

Can you write a racket function to parse files code001.txt to code011.txt?
(match (parse "input/code001.txt")
  [(success result) (pretty-print result) (printf "~n~nACCEPT :) ~n~n")]
  [(failure err-msg) (displayln err-msg) (printf "~n~nDENY :(~n~n")])

Why don't we do something like this?

```
(define source-files '("input/code001.txt"
                       "input/code002.txt"))

(for-each (lambda (file)
          (printf "~n======================== ~a ========================~n~n" file)
          (match (parse file)
  [(success result) (pretty-print result) (printf "~n~nACCEPT :) ~n~n")]
  [(failure err-msg) (displayln err-msg) (printf "~n~nDENY :( ~n~n")])
        source-files)
```

but instead of write out all of the filenames in source-file, we just take make a list of every file in input?

Why don't we just do:

```
; Get all files from the input directory
(define all-files (directory-list "input"))


(for-each (lambda (file)
          (printf "~n======================== ~a ========================~n~n" file)
          (match (parse file)
            [(success result) (pretty-print result) (printf "~n~nACCEPT :) ~n~n")]
            [(failure err-msg) (displayln err-msg) (printf "~n~nDENY :( ~n~n")]))
        all-files)
```

and string append input/ to each file

Now I want a summary of the input files and whether they were parsed successfully or failed

---

Why is my summary out of order?