Get File Name Function w/ `racket/path`

(ended up using something else)

if this code is defined in project/src/user-interface.rkt but is accessed in project/main.rkt, what should I define the modified-input as if I am accessing files in the project/parser-input directory

```
; in:  no args
; out: returns a valid string file name from user input
(define (get-file-name)
  (display "Enter Parser Input File Name\n")
  (define input (read-line))
  (define modified-input (format ""))
```

racket Copy

```racket
#lang racket

(require racket/path) ; For path manipulation functions

;; in:  no args
;; out: returns a valid string file name from user input
(define (get-file-name)
  (display "Enter Parser Input File Name\n")
  (define input (read-line))

  ;; Construct the full path to the file in the project/parser-input directory
  (define base-dir (current-directory))
  (define parser-input-dir (build-path base-dir "parser-input"))
  (define modified-input (build-path parser-input-dir input))

  ;; Check if the file exists
  (if (file-exists? modified-input)
      modified-input
      (begin
        (printf "Error: File '~a' does not exist in the 'parser-input' directory.\n" input)
        (get-file-name)))) ; Recursively ask for input again
```

PLUS / MINUS / NUMSIGN Token Ambiguity in Lexer

I got this error:

Error: Encountered parsing error near "-" (token 'MINUS) while parsing 'unknown [line=3, column=4, offset=23]

<Insert Useless AI Hallucinations that Didn't Help>

From my understanding, there is ambiguity between the NUMSIGN token and the PLUS and MINUS tokens. How would I fix this without getting rid of either?

```
;; PLUS and MINUS (for arithmetic operations)
[(union "+" "-")
 (let ([next-char (peek-char ip)])
   (if (and next-char (char-numeric? next-char))
       ;; If followed by a digit, treat as NUMSIGN (part of a number)
       (token 'NUMSIGN lexeme)
       ;; Otherwise, treat as PLUS or MINUS (arithmetic operation)
       (token (if (equal? lexeme "+") 'PLUS 'MINUS) lexeme)))]
```

Graceful Error Handling w/ `with-handlers`

If I use Brag's parser generating function "(define my-parser (make-rule-parser program))" where 'program' is the start symbol, is there a way to gracefully handle errors in this case? Maybe with an either monad?

Option 1: Using `with-handlers` for Error Handling

You can use Racket's `with-handlers` to catch exceptions thrown by the parser and report errors gracefully.

Steps:

1. Define your grammar and generate the parser using `make-rule-parser`.
2. Wrap the parser in a function that uses `with-handlers` to catch errors.
3. Report errors with their location and continue processing.

<I didn't end up using any of the code it spat out, but I did use the `with-handlers` function>