```
begin
    using PlutoUI
    using DataStructures
end
```

```
parse_file (generic function with 1 method)
```

```
function parse_file(io::IO)
    return [collect(line) for line in eachline(io)]
end
```

```
is_starting_char (generic function with 1 method)
```

```
is_starting_char(c) = c == '{' || c == '(' || c == '<' || c == '['
```

```
is_ending_char (generic function with 1 method)
```

```
is_ending_char(c) = c == '}' || c == ')' || c == '>' || c == ']'
```

```
is_corresponding_ending_char (generic function with 1 method)
```

```
function is_corresponding_ending_char(open, close)
    return ((open == '{' && close == '}')
        || (open == '(' && close == ')')
        || (open == '<' && close == '>')
        || (open == '[' && close == ']'))
end
```

```
get_first_illegal_char (generic function with 1 method)
```

```
function get_first_illegal_char(chars)
    s = Stack{Char}()
    for c in chars
        if is_starting_char(c)
            push!(s, c)
        elseif length(s) == 0
            return c
        else
            expected_starting_char = pop!(s)
            if !is_corresponding_ending_char(expected_starting_char, c)
                return c
            end
        end
    end
    return nothing
end
```

Problem 1

```
prob1_score_map =   Dict('}' ⇒ 1197, ']' ⇒ 57, ')' ⇒ 3, '>' ⇒ 25137)
```

```
prob1_score_map = Dict(')' => 3, ']' => 57, '}' => 1197, '>' => 25137)
```

```
get_illegal_chars (generic function with 1 method)
```

```
function get_illegal_chars(lines)
    return get_first_illegal_char.(lines) |> l -> filter(cs -> cs != nothing, l)
end
```

```
243939
```

```
  0.000134 seconds (318 allocations: 434.016 KiB)
```

```
with_terminal() do
    open("./Day10/prob_input.txt") do io
        lines = parse_file(io)
        @time illegal_chars = get_illegal_chars(lines)
        map(c -> prob1_score_map[c], illegal_chars) |> sum
    end
end
```

Problem 2

```
prob2_score_map =   Dict('}' ⇒ 3, ']' ⇒ 2, ')' ⇒ 1, '>' ⇒ 4)
```

```
prob2_score_map = Dict(')' => 1, ']' => 2, '}' => 3, '>' => 4)
```

```
get_closing_char (generic function with 1 method)
```

```
function get_closing_char(c)
    if c == '{'
        return '}'
    elseif c == '('
        return ')'
    elseif c == '<'
        return '>'
    elseif c == '['
        return ']'
    end

    error("Invalid char $(c)")
end
```

```
get_legal_lines (generic function with 1 method)
```

```
function get_legal_lines(lines)
    illegal_lines = get_first_illegal_char.(lines)
    legal_lines_index = findall(c -> c == nothing, illegal_lines)
    lines[legal_lines_index]
end
```

```
get_closing_chars_for_legal_line (generic function with 1 method)
```

```
function get_closing_chars_for_legal_line(chars)
    s = Stack{Char}()
    for c in chars
        if is_starting_char(c)
            push!(s, c)
        else
            # As we will only get legal lines so making that assumption
            pop!(s)
        end
    end

    return [get_closing_char(c) for c in s]
end
```

```
get_score_for_line (generic function with 1 method)
```

```
function get_score_for_line(closing_chars)
    scores = map(c -> prob2_score_map[c], closing_chars)

    score = 0
    for s in scores
        score = (score * 5) + s
    end

    return score
end
```

```
get_median_score_for_completion (generic function with 1 method)
```

```
function get_median_score_for_completion(lines)
    closing_chars_per_line = get_closing_chars_for_legal_line.(lines)
    scores_per_line = get_score_for_line.(closing_chars_per_line)
    sort!(scores_per_line)

    mid_index = Int(floor(length(scores_per_line)/2)) + 1
    return scores_per_line[mid_index]
end

```

```
2421222841
```

```
  0.000301 seconds (1.38 k allocations: 250.922 KiB)
```

```
with_terminal() do
    open("./Day10/prob_input.txt") do io
        lines = parse_file(io)
        legal_lines = get_legal_lines(lines)
        @time get_median_score_for_completion(legal_lines)
    end
end
```