Gabriel Coyote

March 25, 2021

3: The Task

Introduction

I've completely reinvented my program. Instead of *goto*(s) and *case/if* statements, I went with a table driven scanner. This change in idea help me shorten my program by 150~ lines of code. As a result, my methods/functions need to be restated.

Data Structure

To store the token type we set an one-dimensional integer array to hold these values; The integar array, *tokens_stored,* is of size 200. All values are set to zero before the *scan* function runs. The index 0 of *tokens_stored* represents the first token encounter. Index 1, the second token encountered and so on. The *tokens_stored[0] := 16,* for example reads as the first token encounter is of state 16 (from DFA its token ID). New state, that acts as exception for the ID token are *read* and *write,* state 21 and 22 respectfully.

We have token array, *token_tab,* which is of size 19 to specify the tokens in Fig 2.12 in the textbook. Looking at the Fig 2.12 we see token *div* has a state number of 2, and so in the token array at index 2 we have the value 2. Token *plus* has a state number of 8, and so at *token_tab[8] := 8,* and so on for the rest of the tokens. For indices in the *token_tab* array without tokens, their values are set to 0.

For the scanner table in the Fig 2.12 we need a two dimensional array, named *scan_tab*. The first dimension is index from 0 to 18 (The number of states; index 0 will not be used) and the second dimension is index from 0 to 13 with 0 reprsenting white spaces (space, tab), 1 (newline), 2 (/), 3 (*), and so on. For any integer *i* and *j, scan_tab[i][j]* is a record with field names *action* and *newState.action* can take values of *move*, *recognize*, *error,* and if *action = move* means that the automata should move to the next state (the next state value is equal to the one stored at *scan_tab[i][j]).* If *action = recognize* means that *i* is a final state and the automata can not move to any other state *i* with the input character corresponding to the number *j*. We recognize a token! If *move = error* means that the automata can not get to any state from state *i* with a character corresponding to the number *j*.

Algorithms

Algorithm : print_tokens

Input:

tokens_stored: an integer array to store tokens encountered (their State #)

Output:

N/A

Data:

i: number used for accessing *tokens_stored* indices

Side Effects:

If token is not *white_space* (space,tab) or *comment* then print token

Print the tokens formatted as *(token1, token2, …)*

Plan:

i = 0;

print *(*

while *tokens_stored[i]* is not empty //While there are tokens to print

if *tokens_stored[i]* is *white_space* or *comment*

// Don't print token

else if *tokens_stored[i]* is *div's* state number (2)

print *div*

else if *tokens_stored[i]* is *lparen's* state number (6)

print *lparen*

else if *tokens_stored[i]* is *rparen's* state number (7)

print *rparen*

. . . //Repeat for all tokens' state

else if *tokens_stored[i]* is *read's* state number (21)

print *read*

else if *tokens_stored[i]* is *write's* state number (22)

print *write*

else print error

increment *i* by one

if *tokens_stored[i]* is not empty

if tokens are *white_space* or *comment* Don't print

else print ,

else if *tokens_stored[i]* is empty print *).*

else

End of Algorithm

Algorithm : scan

Input:

File_PTR: The current pointer of the input file
cur_char: current character
cur_state: holds the current state #
remembered_state: holds state #
image: list of characters, used to hold encountered token's string

Output:

token: holds encountered token's state #

Side Effects:

Prints "error." if encountered token is invalid, then terminates the program

Plan:

```
while File_PTR is not EOF
    read cur_char

    case scan_tab[cur_state][ cur_char ].action
        move:
            if token_tab[cur_state]  is not empty (!= 0)
                remembered_state := cur_state
            cur_state := scan_tab[cur_state][cur_char].nextState
        recognize:
            token := token_tab[cur_state]
            unread cur_char
            return token
        error:
            print "error." , then terminate program

    append cur_char to image
```

End of Algorithm

Algorithm : Int_cur_char

Input:

ch: a character

Output:

returns a number, for *scan_tab[][i]* array

Side Effects:

N/A

Plan:

If *ch* is a space or tab
    return 0
else if *ch* is a newline character
    return 1
else if *ch* is a "/"
    return 2
else if *ch* is a "*"
    return 3
    . . . //Repeat for all cur character in the Fig 2.12 table
else if *ch* is a digit
    return 11
else if *ch* is a letter
    return 12
else return 13

End of Algorithm

Algorithm : Driver

Input:

File_PTR: The current pointer of the input file
tokens_stored: array to hold tokens encounterd (Their State #)

Output:

tokens_stored elements' values are set to the token encountered (Their State #)

Data:

i: number used for accessing *tokens_stored* indices
tok: number used to hold encountered token's state #
cur_char: current character
cur_state: holds the current state #
remembered_state: holds state #
image: list of characters, used to hold encountered token's string

Side Effects:

N/A

Plan:

While *File_PTR* is not EOF
    cur_state := *start_state* (1)
    remembered_state := 0 //None
    image := null

    //*tok* is the output of *scan* [Algorithm]
    tok := scan(*File_PTR, cur_char, cur_state, remembered_state, image)*

    if *image* is equal to "read"
        tok := *read's* state number (21)
    else if *image* is equal to "write"
        tok := *write's* state number (22)
    else
        //Leave *tok* as is

    Tokens_stored[i] := *tok*
    increment *i* by one

End of Algorithm

Main Algorithm

Input:

Filename: text file name from the command line

Output:

N/A

Data:

inputFile: the file pointer

tokens_stored: array to hold tokens encounterd (their State #)

Side Effects:

Prints to console error. if there is any non-valid token in the input file; otherwise Prints the list of tokens in the input file

Plan:

inputFile := open *Filename*

Driver(*inputFile*, *tokens_stored*); //Algorithm

Print_tokens(*tokens_stored*); //Algorithm

Close *Filename*

End of Algorithm

Test Cases

1.) The test case will be the text file *foo.txt* that reads as :

```
read
/* foo
   bar */
*
five 5
```

I chose this one as I know what to expect as an output, which should be (read, times, id, number)

2.) The other test case will be the text file *test.txt* that reads as :

```
:= (name) 24
var1 25.56 var3
.25
```

I chose this one as I know it will test my program on the '.' Character and (i|r)paren as well as id that have numbers in them.

Aknoweledgement

Thank you Edward Wertz, Xiaozhen Xue andSamiraTaleb for helping make this project specification. Also big thanks to the book, *Programming Language Pragmatics 4th edition (Micharl L. Scott),* for helping me understand the scanner's purpose.