EXP:07

DATE:06/08/2025

PYTHON CODE VALIDATION AND DFA CONSTRUCTION

Q1:

1. **Write a LEX program to validate the given python code snippet**

Note: Use files for giving inputs

Test Case1:

Input:

```
weather = True
if weather:
    print("Time to wear
sunglasses!")
else:
    print("No need for sunglasses.")
for i in range(6):
    print(i)
```

Output: Success

Test Case2:

Input:

```
weather = True
if weather
    print("Time to wear sunglasses!")
else:
    print("No need for sunglasses.")
for i in range(6):
    print(i)
```

Output: Error at line 2

CODE:

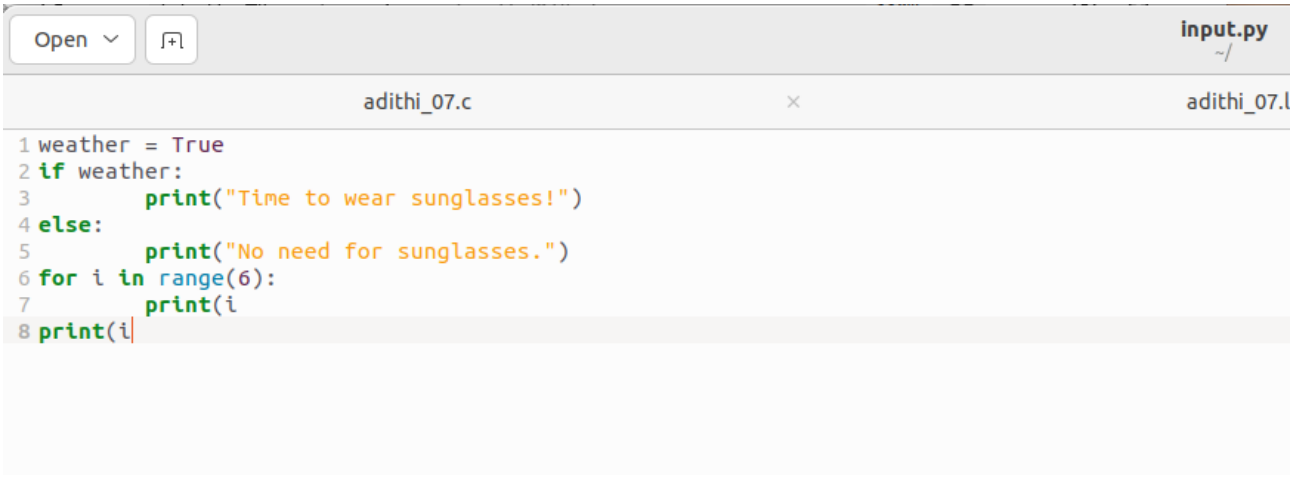
```
%{
#include <stdio.h>
#include <string.h>

int lineno = 1;
int error_lines[1024]; // store up to 1024 error lines
int error_count = 0;

int expect_colon = 0;
int open_string = 0;
int assign_count = 0;
int indent_level = 0;
int prev_indent = 0;
int at_line_start = 1;

int paren_count = 0;
int paren_error_line = -1;
```

```
void add_error_line(int line) {
   for (int i = 0; i < error_count; i++)
      if (error_lines[i] == line)
         return;
   error_lines[error_count++] = line;
}
%}

%option noyywrap

DIGIT       [0-9]
ID          [a-zA-Z_][a-zA-Z0-9_]*
NEWLINE     \n
WHITESPACE  [ \t]+

%%

"if"|"else"|"for"      { expect_colon = 1; at_line_start = 0; }
":"                    { expect_colon = 0; at_line_start = 0; }
"="                    { assign_count++; at_line_start = 0; }

\"([^\\\"]|\\.)*       { open_string = 1; at_line_start = 0; } // unclosed string
\"([^\\\"]|\\.)*\"     { open_string = 0; at_line_start = 0; } // closed string

"(" {
   paren_count++;
   at_line_start = 0;
}

")" {
   paren_count--;
   at_line_start = 0;
   if (paren_count < 0 && paren_error_line != lineno) {
      add_error_line(lineno);
      paren_error_line = lineno;
      paren_count = 0; // reset to prevent cascading errors
   }
}

"["|"]"|"{"|"}"|","|"."    { at_line_start = 0; }
"in"|"range"               { at_line_start = 0; }
"True"|"False"|"print"     { at_line_start = 0; }

{ID}                       { at_line_start = 0; }
{DIGIT}+                   { at_line_start = 0; }
"+"|"-"|"*"|"/"            { at_line_start = 0; }
```

```
{WHITESPACE} {
   if (at_line_start) indent_level = yyleng;
}

{NEWLINE} {
   if (expect_colon) {
      add_error_line(lineno);
      expect_colon = 0;
   }
   if (open_string) {
      add_error_line(lineno);
      open_string = 0;
   }
   if (assign_count > 1) {
      add_error_line(lineno);
   }
   if (indent_level % 4 != 0) {
      add_error_line(lineno);
   }
   if (paren_count != 0) {
      add_error_line(lineno);
      paren_count = 0; // reset each line
   }

   assign_count = 0;
   prev_indent = indent_level;
   indent_level = 0;
   at_line_start = 1;
   lineno++;
}

. {
   add_error_line(lineno);
   at_line_start = 0;
}

%%

int main(int argc, char **argv) {
   if (argc > 1) {
      FILE *file = fopen(argv[1], "r");
      if (!file) {
         perror("File opening failed");
         return 1;
      }
      yyin = file;
```

```
    }

    yylex();

    for (int i = 0; i < error_count; i++) {
        printf("error in line %d\n", error_lines[i]);
    }

    if (!error_count)
        printf("no error, running successfully\n");

    return 0;
}
```

OUTPUT:



2. **Construct DFA and validate the strings for the language**

$\{b^{m+1}\ a^{2n}, n>=2 \ \&\ m\%2=0, m>=1\}$.

CODE:

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>

#define NUM_STATES 8

// Function to display transition table
void print_transition_table() {
    printf("-------------------------------------------------\n");
    printf("| State |  b  |  0  |  a  |\n");
    printf("-------------------------------------------------\n");
    printf("| q0    | q1  |  -  |  -  |\n");
    printf("| q1    | q2  |  -  |  -  |\n");
    printf("| q2    | q2  |  -  |  -  |\n");
    printf("| q3    |  -  |  -  | q4  |\n");
    printf("| q4    |  -  |  -  | q4  |\n");
    printf("| q5    |  -  |  -  |  -  |\n");
    printf("-------------------------------------------------\n");
    printf("Note: '-' indicates no transition or invalid.\n");
}
```

```
// Function to check if a number is a power of two, >=4
bool is_power_of_two_at_least_4(int n) {
    if (n < 4) return false;
    return (n & (n - 1)) == 0; // power of two check
}

// Function to validate the string based on updated rules
bool is_valid_string(const char *str) {
    int len = strlen(str);
    int i = 0;

    // Count b's
    int b_count = 0;
    while (i < len && str[i] == 'b') {
        b_count++;
        i++;
    }

    int m = b_count - 1;
    // Check if total b's is odd, m ≥ 4 (since m even and ≥4)
    if (b_count < 5 || m % 2 != 0 || m < 4) {
        return false;
    }

    // Count a's
    int a_count = 0;
    while (i < len && str[i] == 'a') {
        a_count++;
        i++;
    }

    // Check if a_count is a power of two >= 4
    if (!is_power_of_two_at_least_4(a_count)) {
        return false;
    }

    // No extra characters
    if (i != len) {
        return false;
    }

    return true;
}

int main() {
    print_transition_table();
```

```
    printf("\nEnter the string to validate: ");
    char input[100];
    scanf("%99s", input);

    if (is_valid_string(input)) {
        printf("String is accepted by the DFA.\n");
    } else {
        printf("String is rejected by the DFA.\n");
    }

    return 0;
}
```

OUTPUT:

```
ubuntu@unix-Veriton-M200-H610:~$ ./adi
-------------------------------------------------
| State |  b  |  0  |  a  |
-------------------------------------------------
| q0    | q1  |  -  |  -  |
| q1    | q2  |  -  |  -  |
| q2    | q2  |  -  |  -  |
| q3    |  -  |  -  | q4  |
| q4    |  -  |  -  | q4  |
| q5    |  -  |  -  |  -  |
-------------------------------------------------
Note: '-' indicates no transition or invalid.

Enter the string to validate: bbbbbaaaa
String is accepted by the DFA.
```

RESULT:

The programs were executed and the outputs have been verified.