Compiler Construction

Assigment 1

Kacper Multan

1 Regular languages, NFAs and DFAs

Let the formal language L be all strings over the alphabet {a, b, c}, where there is at least one a, and there are no cs before the first a, nor after the last a.

1.1

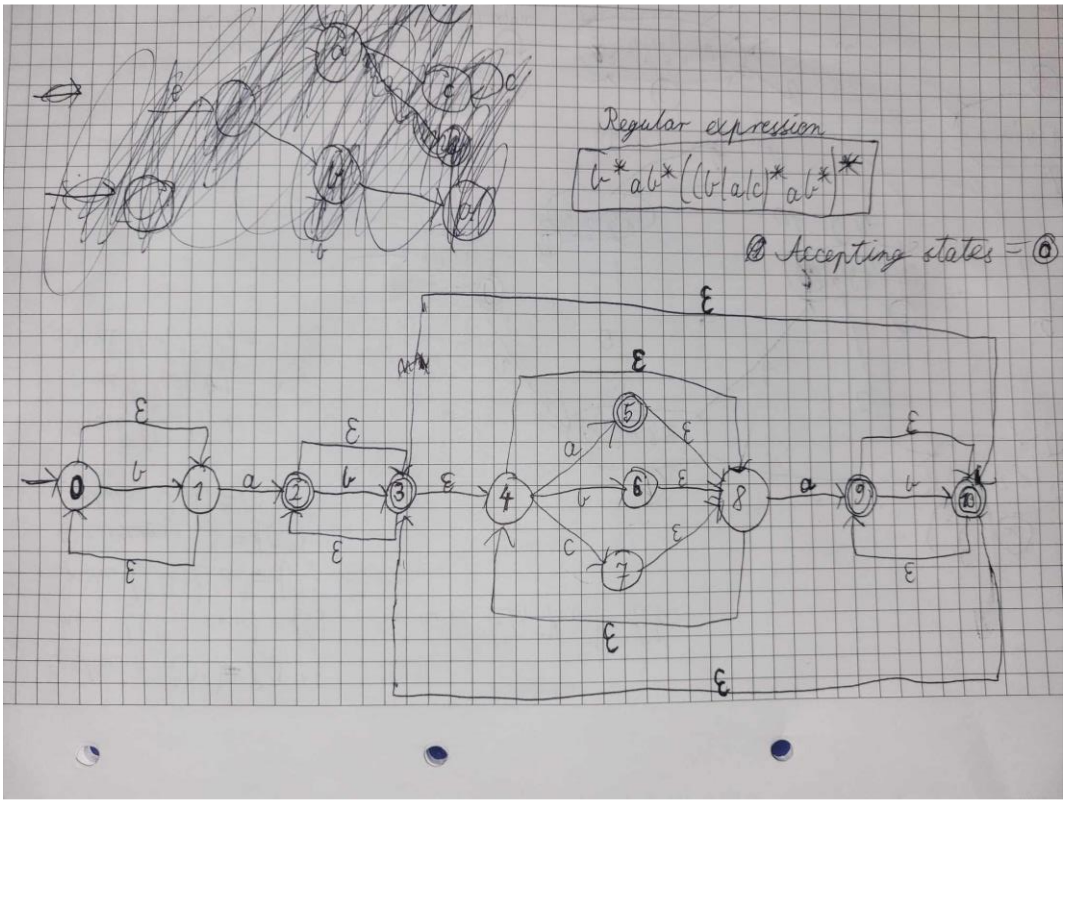
Show that L is a regular language, by writing a regular expression for it. You only need operators described in slideset 03: |, * and grouping with ( ). You may also use X? as a shorthand for (X|).

Regular expression matching language L - b*ab*((b|a|c)*ab*)*

1.2

Convert the regex from 1.1 into a non-deterministic finite automata (NFA) using the McNaughton–Yamada–Thompson algorithm. Remember to number the states, indicate the starting state, and mark states as either accepting or non-accepting.

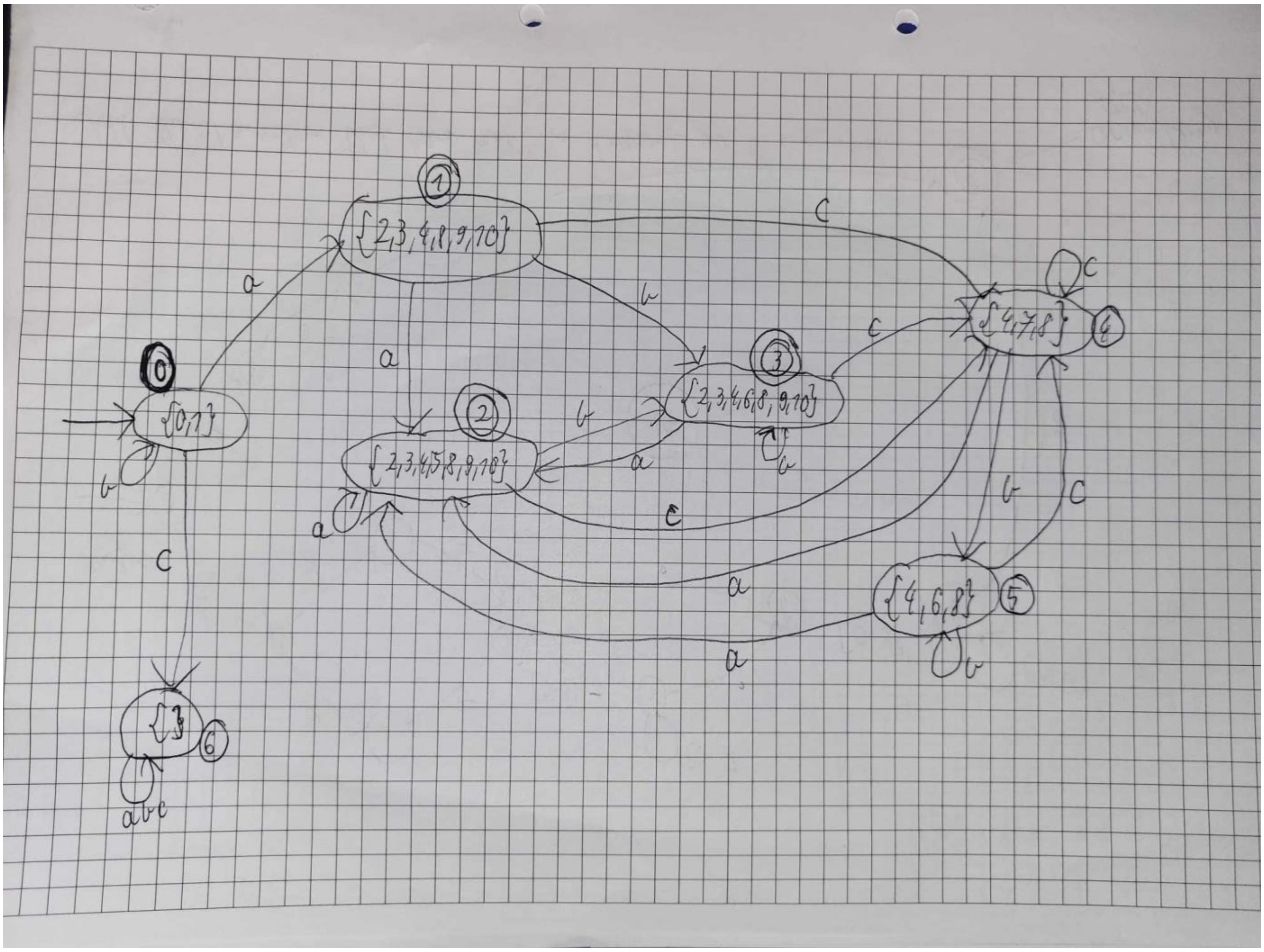
NFA



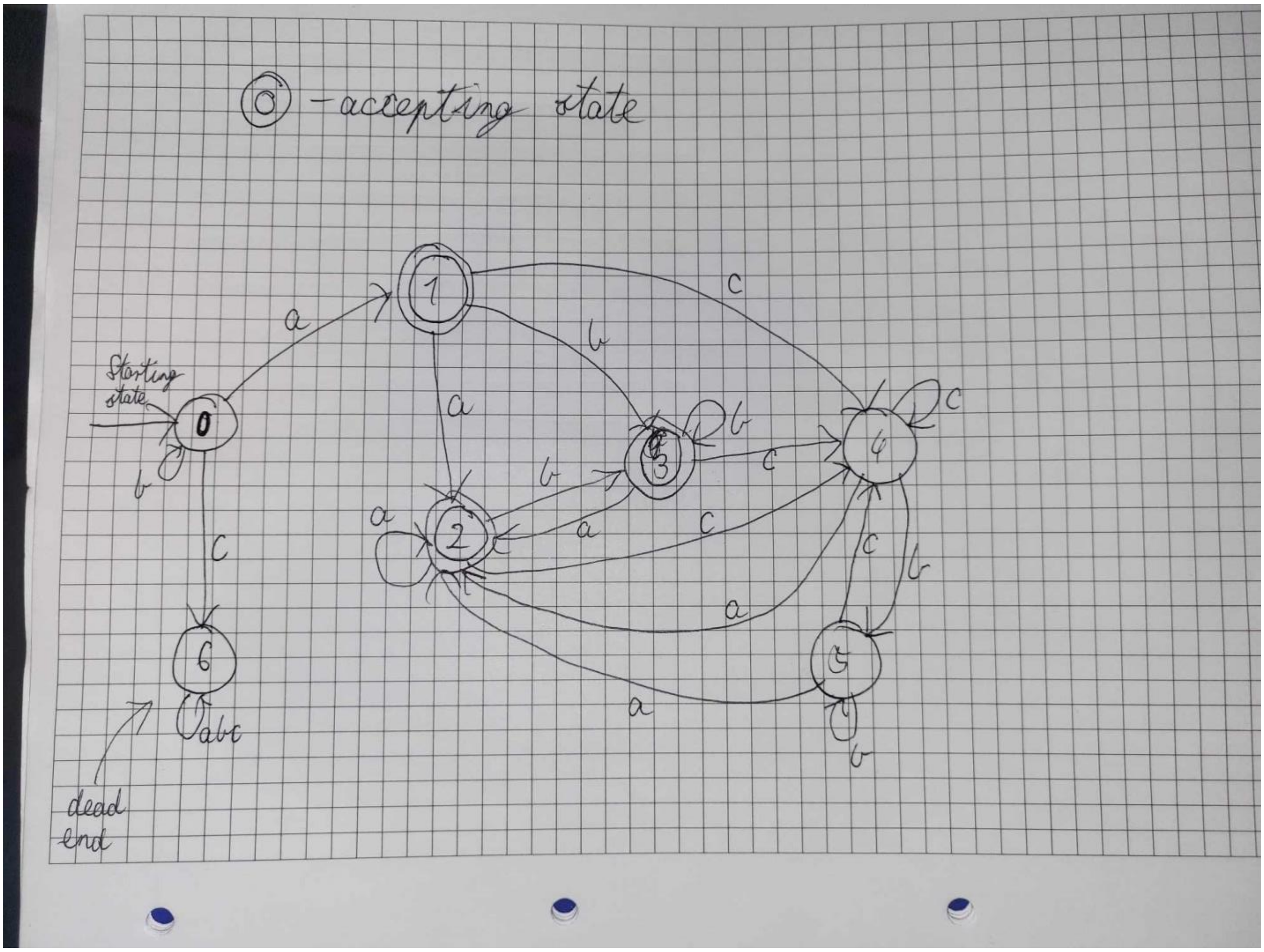
1.3

Convert the NFA from 1.2 into a deterministic finite automata (DFA), using the subset construction method described in slideset 04. The Recitation Lecture shows a more complete example of NFA-to-DFA conversion.

Transforming NFA to DFA



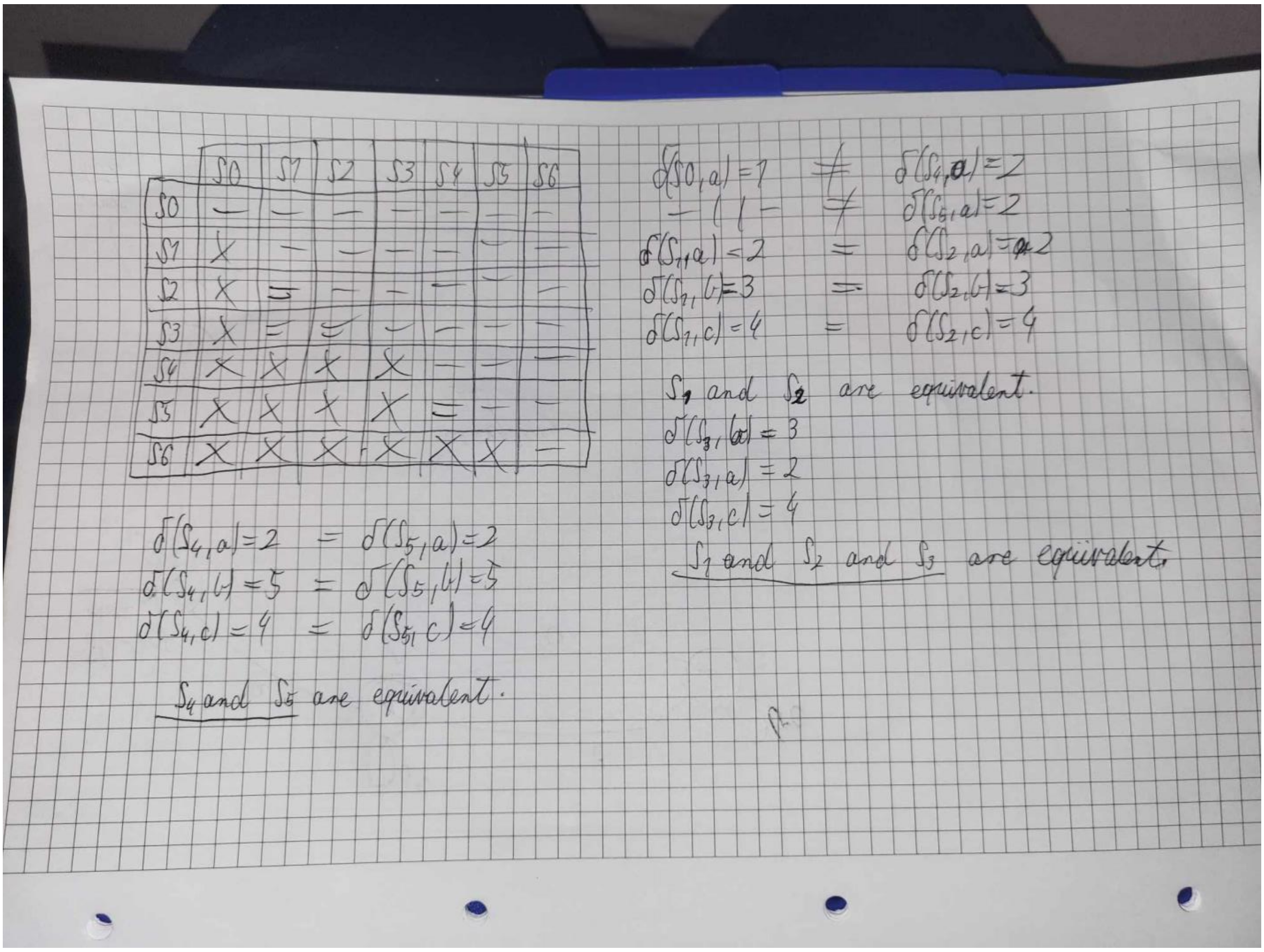
DFA


◎ -accepting state
Starting state
0
a
1
c
b
a
3
b
c
4
c
2
a
b
a
c
c
b
a
5
a
b
c
6
abc
b
dead end

1.4

Minimizing a DFA means creating a new DFA with the minimum number of states that still matches the exact same language. Use the Myhill-Nerode (a.k.a Table-filling) algorithm shown in Slideset 04 to minimize the DFA you created in 1.3.

Table-filling (minimizing)



	S0	S1	S2	S3	S4	S5	S6
S0	—	—	—	—	—	—	—
S1	X	—	—	—	—	—	—
S2	X	=	—	—	—	—	—
S3	X	=	=	—	—	—	—
S4	X	X	X	X	—	—	—
S5	X	X	X	X	=	—	—
S6	X	X	X	X	X	X	—

$\delta(S_4,a)=2 \quad = \quad \delta(S_5,a)=2$

$\delta(S_4,b)=5 \quad = \quad \delta(S_5,b)=5$

$\delta(S_4,c)=4 \quad = \quad \delta(S_5,c)=4$

S4 and S5 are equivalent.

$\delta(S0,a)=1 \quad \neq \quad \delta(S_4,a)=2$

$- \, " \, - \quad \neq \quad \delta(S_5,a)=2$

$\delta(S_1,a)=2 \quad = \quad \delta(S_2,a)=2$

$\delta(S_1,b)=3 \quad = \quad \delta(S_2,b)=3$

$\delta(S_1,c)=4 \quad = \quad \delta(S_2,c)=4$

S1 and S2 are equivalent.

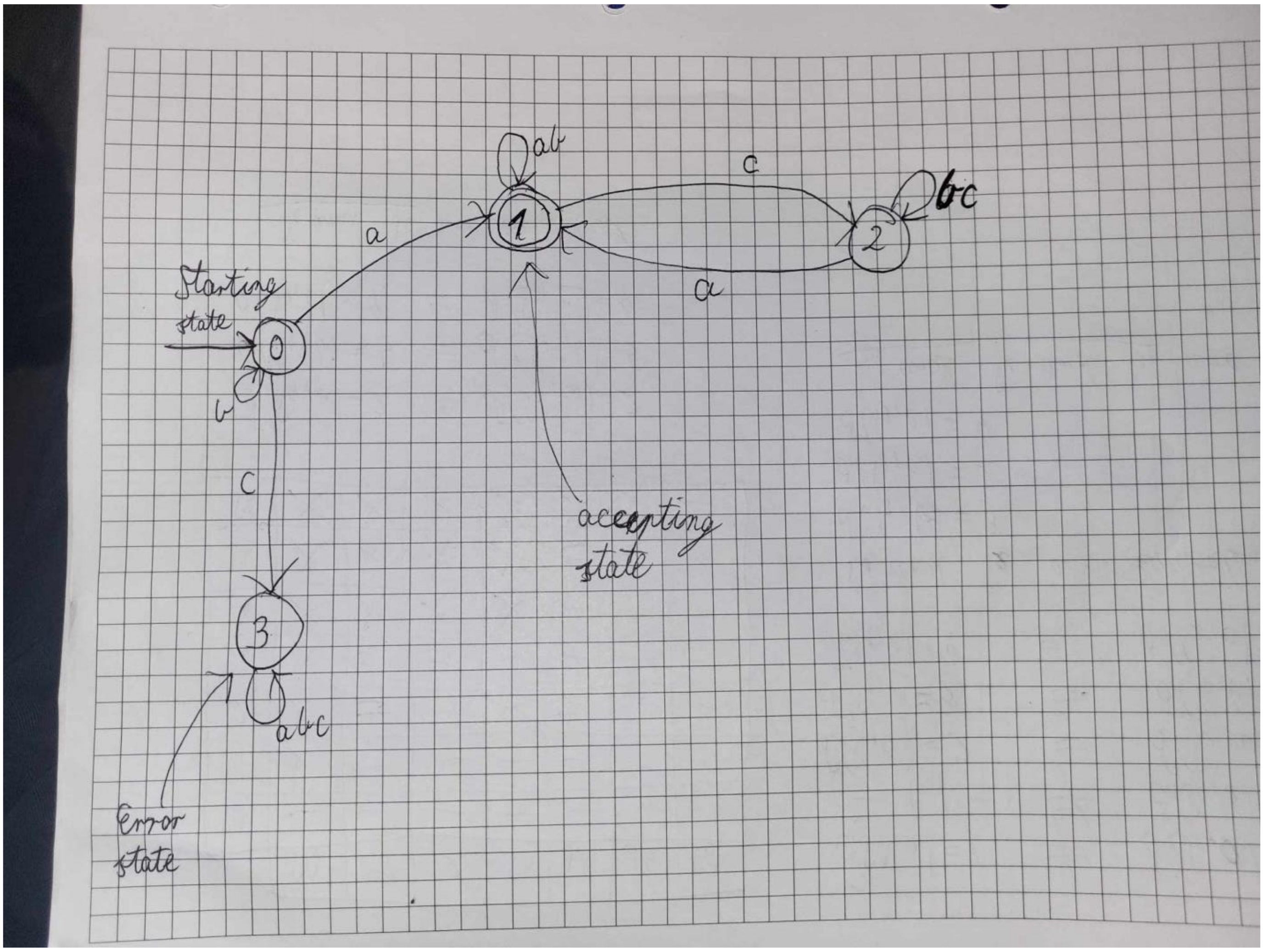
$\delta(S_3,b)=3$

$\delta(S_3,a)=2$

$\delta(S_3,c)=4$

S1 and S2 and S3 are equivalent.

Minimized DFA



1.5

It is not perfect regular expression for “opposite” language but works for most of the situations

b*(c*b*)*|(c+b*a+b*)*|(b*a+b*c+)*|(c+b*a+b*c+)*

Task 2

2.1

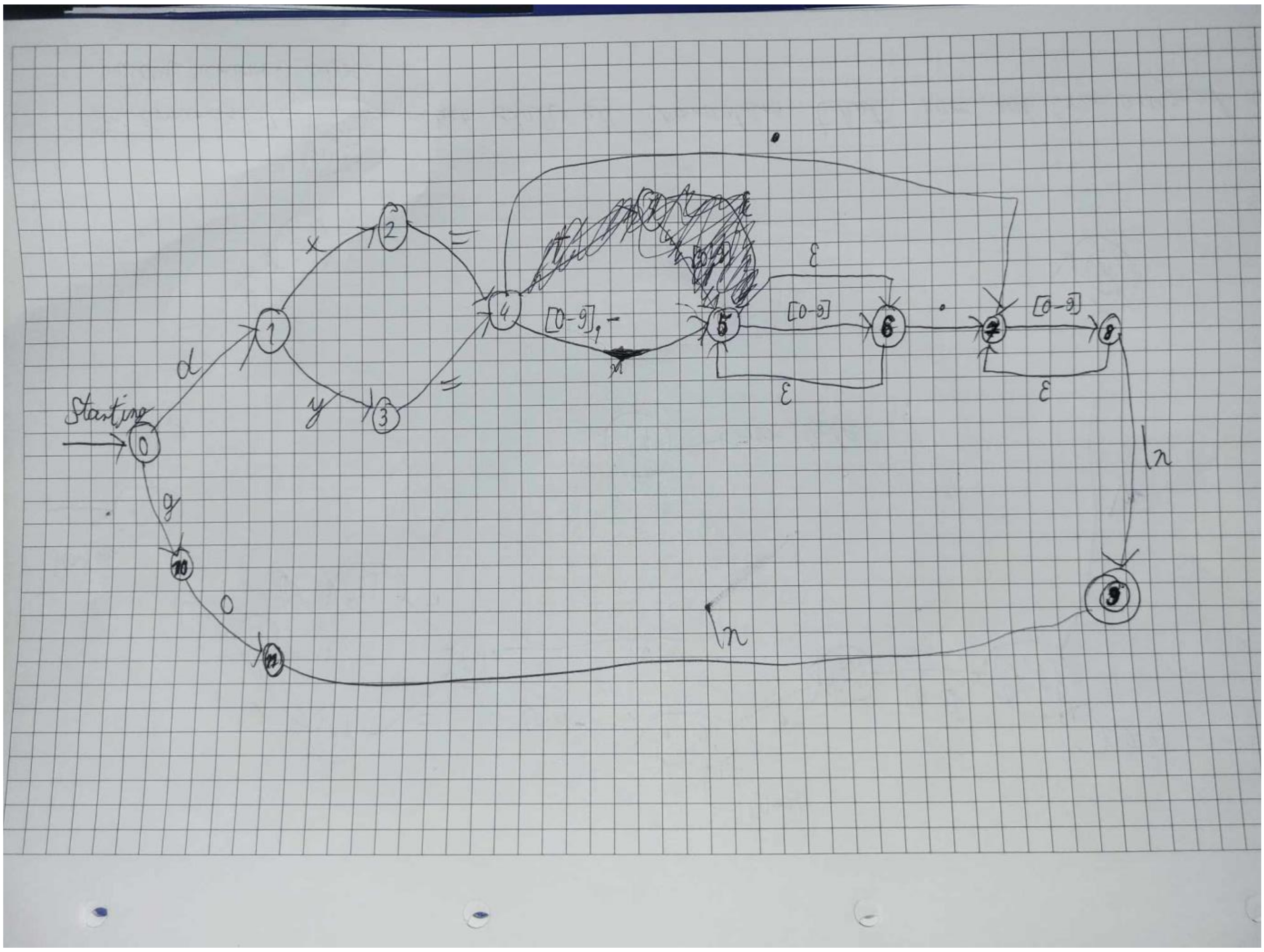
Write a regular expression matching exactly one statement, including the newline character ('\n') at the end. You can use the shorthand [0-9] to mean "any digit", and the operator X + to mean "one or more repretitions of X ".

Regular expression:

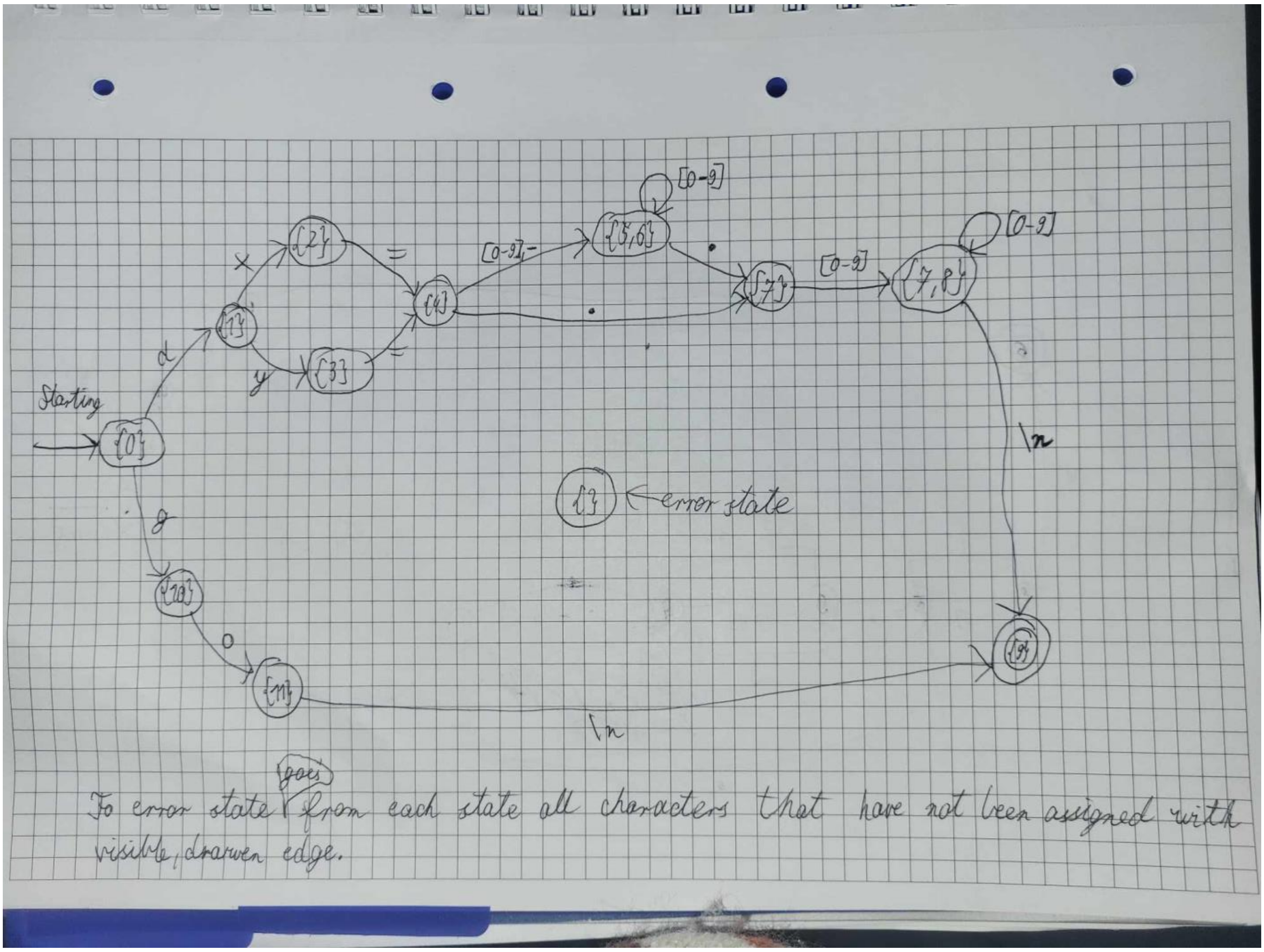
go|((d(x|y)=)(-?)[0-9]*.[0-9]+)\n

2.2

NFA:

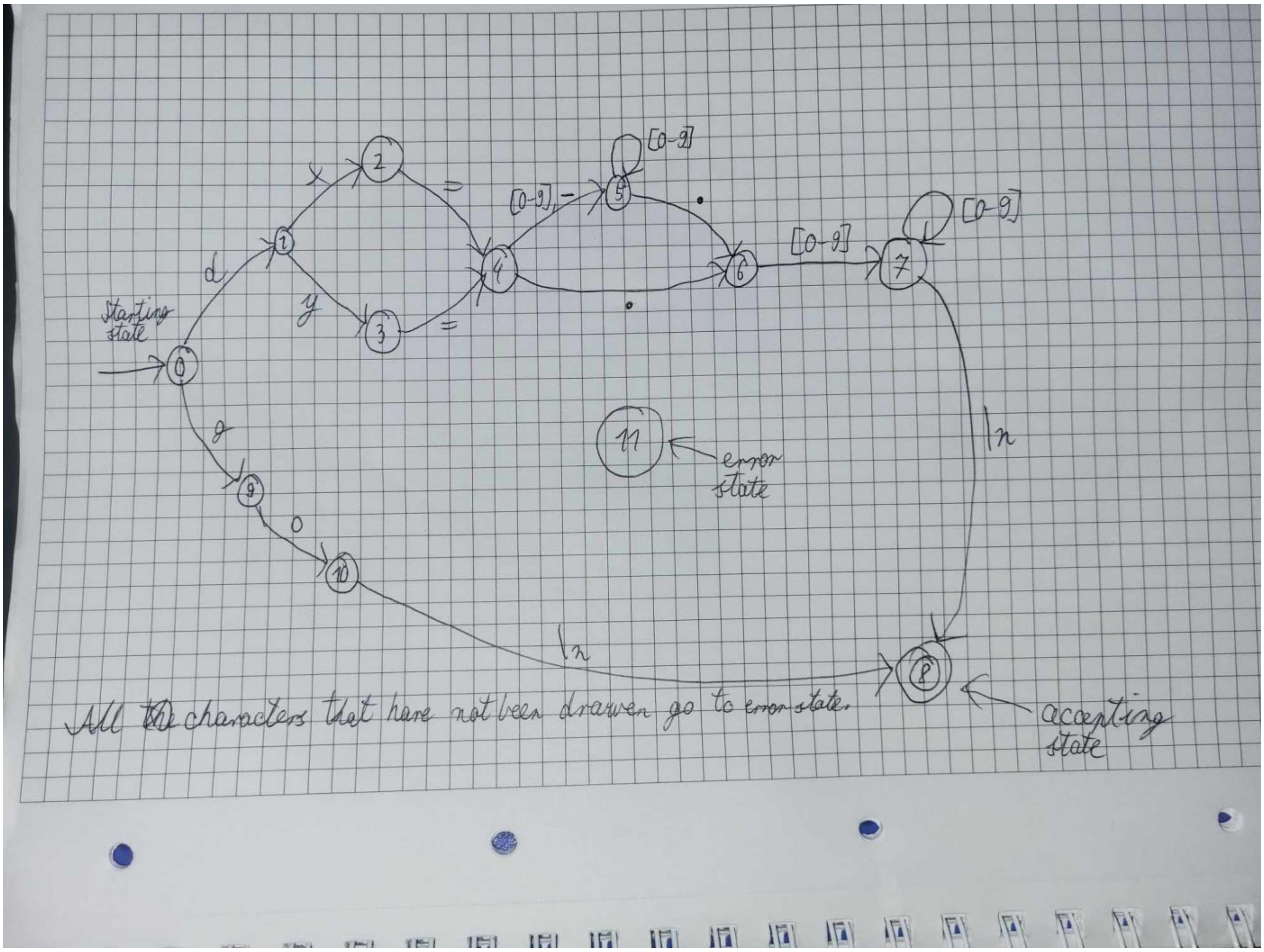


Transforming NFA to DFA



To error state goes from each state all characters that have not been assigned with visible, drawn edge.

DFA:



All the characters that have not been drawen go to error state.

2.3

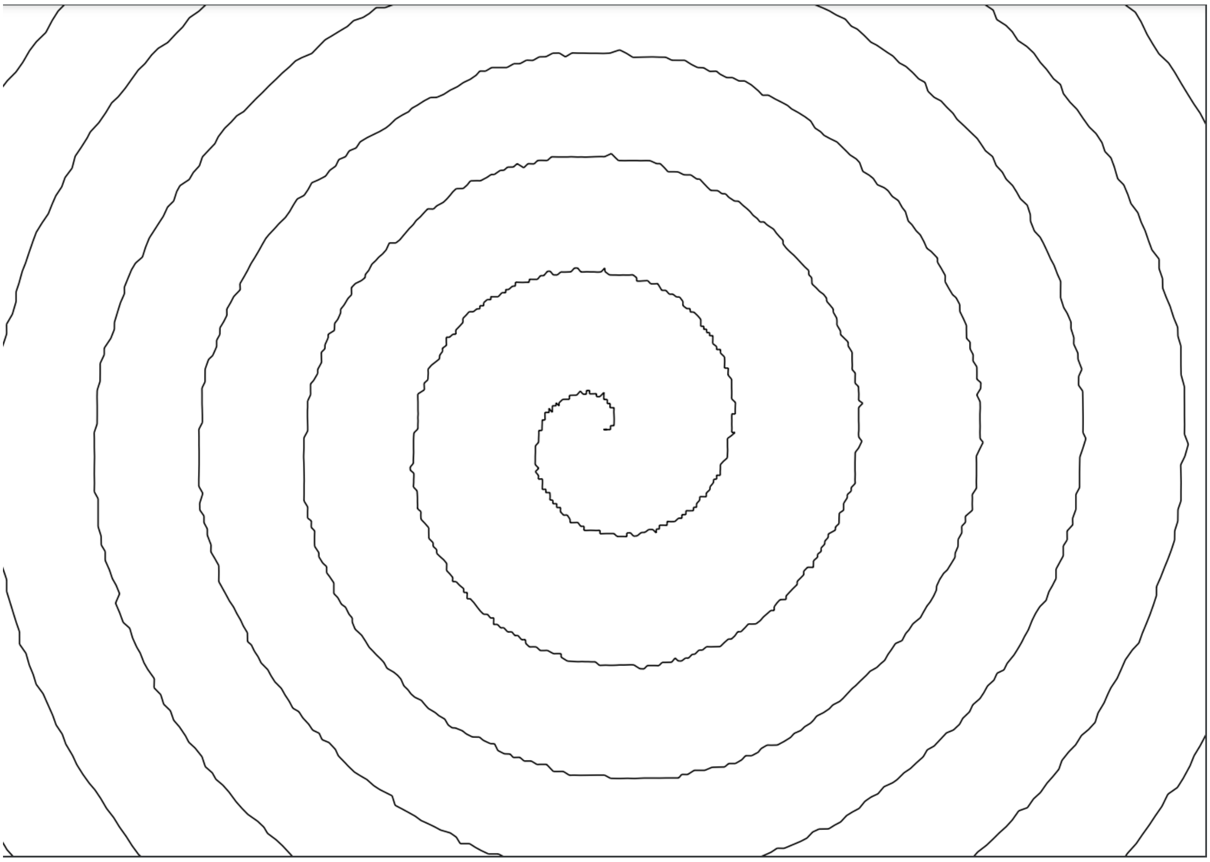
error: 6038: unrecognized statement: dy=-2.55.

The problem was with dot at the end, after erasing it, everything works.

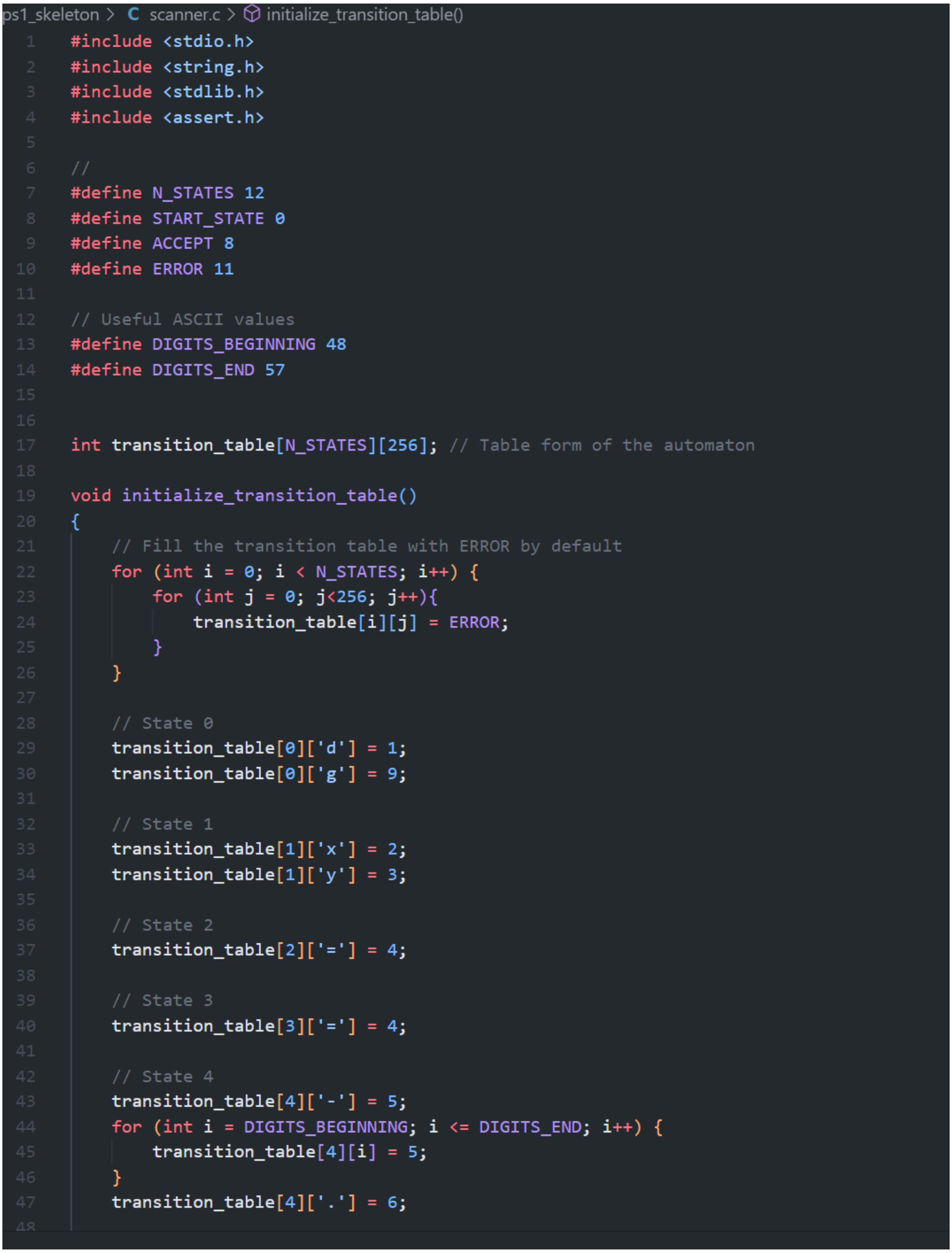
Run of test function:

cat spiral.txt | ./build/scanner | ps2pdf - spiral.pdf

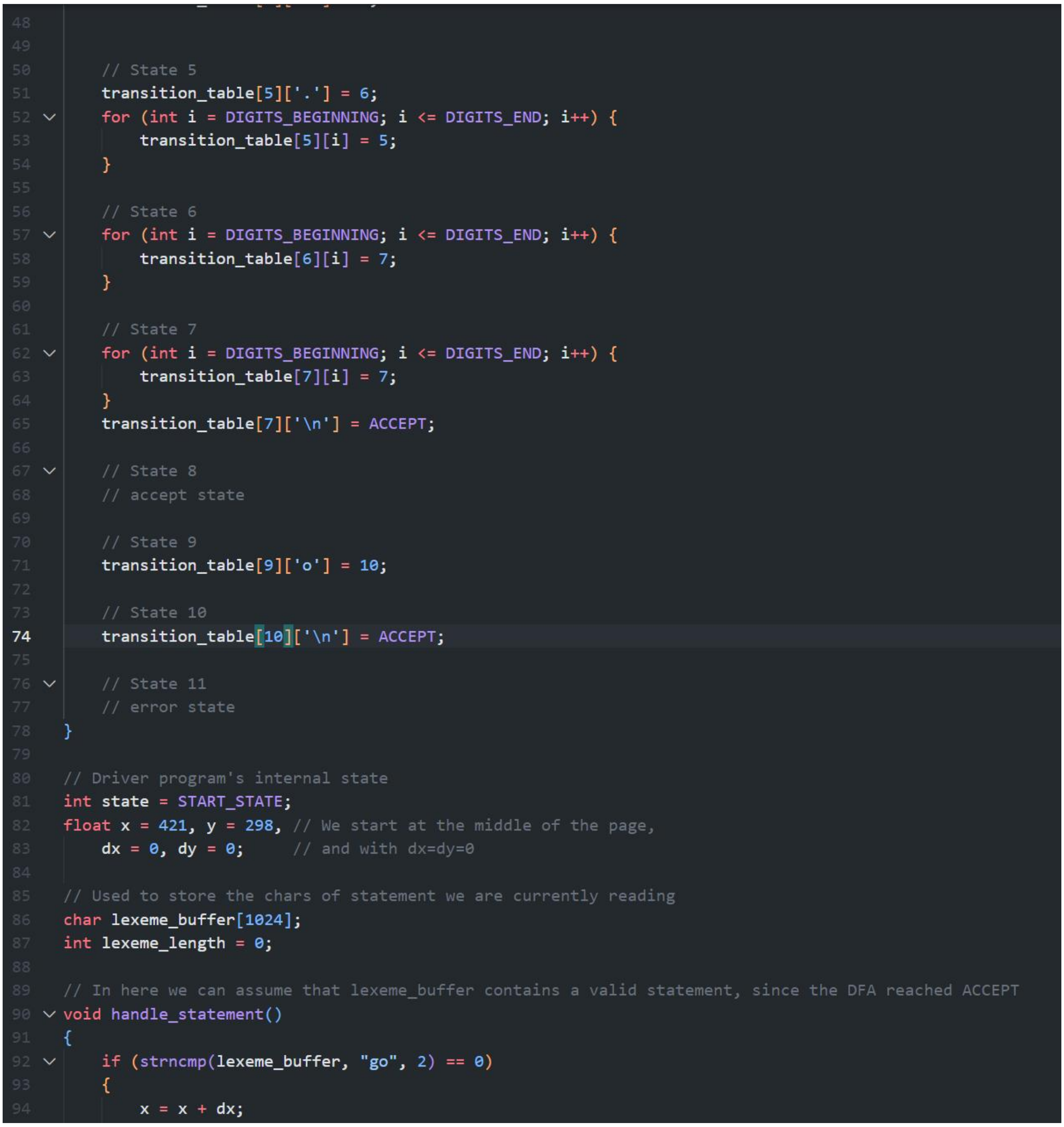
resulted in spiral image below:



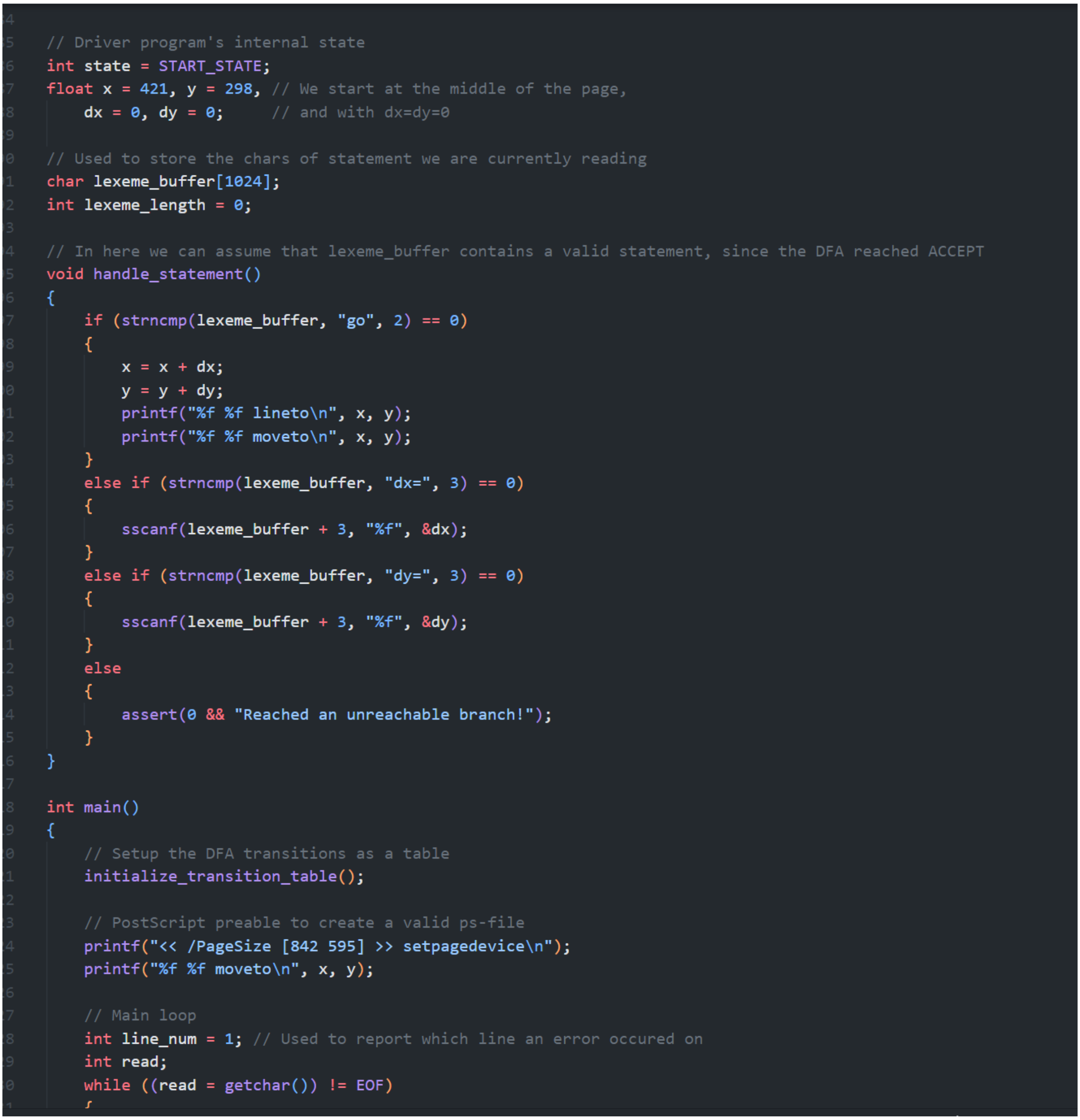
Screens of code from scanner.c file:



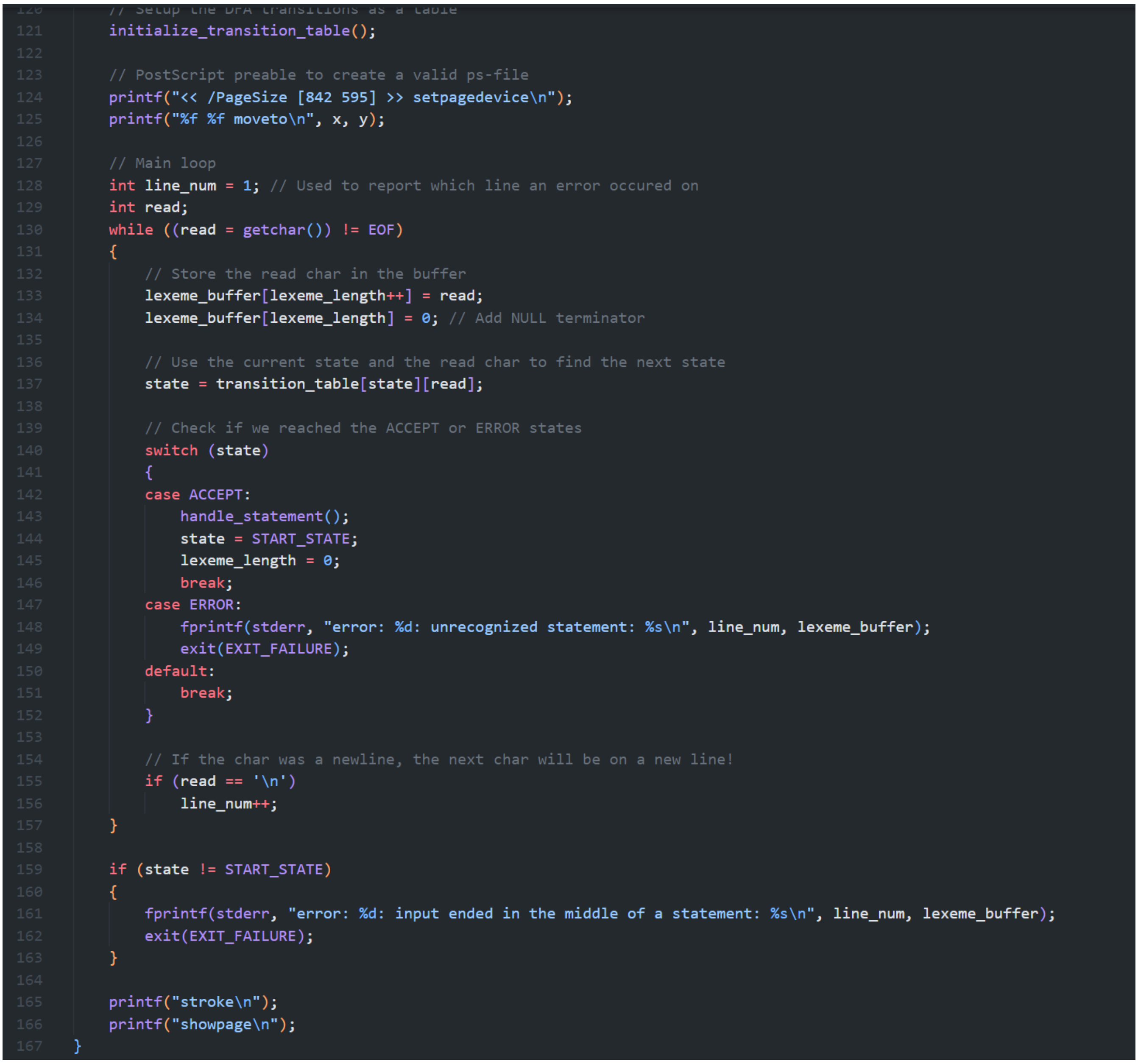
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>

//
#define N_STATES 12
#define START_STATE 0
#define ACCEPT 8
#define ERROR 11

// Useful ASCII values
#define DIGITS_BEGINNING 48
#define DIGITS_END 57


int transition_table[N_STATES][256]; // Table form of the automaton

void initialize_transition_table()
{
    // Fill the transition table with ERROR by default
    for (int i = 0; i < N_STATES; i++) {
        for (int j = 0; j<256; j++){
            transition_table[i][j] = ERROR;
        }
    }

    // State 0
    transition_table[0]['d'] = 1;
    transition_table[0]['g'] = 9;

    // State 1
    transition_table[1]['x'] = 2;
    transition_table[1]['y'] = 3;

    // State 2
    transition_table[2]['='] = 4;

    // State 3
    transition_table[3]['='] = 4;

    // State 4
    transition_table[4]['-'] = 5;
    for (int i = DIGITS_BEGINNING; i <= DIGITS_END; i++) {
        transition_table[4][i] = 5;
    }
    transition_table[4]['.'] = 6;
```



```c


    // State 5
    transition_table[5]['.'] = 6;
    for (int i = DIGITS_BEGINNING; i <= DIGITS_END; i++) {
        transition_table[5][i] = 5;
    }

    // State 6
    for (int i = DIGITS_BEGINNING; i <= DIGITS_END; i++) {
        transition_table[6][i] = 7;
    }

    // State 7
    for (int i = DIGITS_BEGINNING; i <= DIGITS_END; i++) {
        transition_table[7][i] = 7;
    }
    transition_table[7]['\n'] = ACCEPT;

    // State 8
    // accept state

    // State 9
    transition_table[9]['o'] = 10;

    // State 10
    transition_table[10]['\n'] = ACCEPT;

    // State 11
    // error state
}

// Driver program's internal state
int state = START_STATE;
float x = 421, y = 298, // We start at the middle of the page,
    dx = 0, dy = 0;     // and with dx=dy=0

// Used to store the chars of statement we are currently reading
char lexeme_buffer[1024];
int lexeme_length = 0;

// In here we can assume that lexeme_buffer contains a valid statement, since the DFA reached ACCEPT
void handle_statement()
{
    if (strncmp(lexeme_buffer, "go", 2) == 0)
    {
        x = x + dx;
```



```c

// Driver program's internal state
int state = START_STATE;
float x = 421, y = 298, // We start at the middle of the page,
    dx = 0, dy = 0;     // and with dx=dy=0

// Used to store the chars of statement we are currently reading
char lexeme_buffer[1024];
int lexeme_length = 0;

// In here we can assume that lexeme_buffer contains a valid statement, since the DFA reached ACCEPT
void handle_statement()
{
    if (strncmp(lexeme_buffer, "go", 2) == 0)
    {
        x = x + dx;
        y = y + dy;
        printf("%f %f lineto\n", x, y);
        printf("%f %f moveto\n", x, y);
    }
    else if (strncmp(lexeme_buffer, "dx=", 3) == 0)
    {
        sscanf(lexeme_buffer + 3, "%f", &dx);
    }
    else if (strncmp(lexeme_buffer, "dy=", 3) == 0)
    {
        sscanf(lexeme_buffer + 3, "%f", &dy);
    }
    else
    {
        assert(0 && "Reached an unreachable branch!");
    }
}

int main()
{
    // Setup the DFA transitions as a table
    initialize_transition_table();

    // PostScript preable to create a valid ps-file
    printf("<< /PageSize [842 595] >> setpagedevice\n");
    printf("%f %f moveto\n", x, y);

    // Main loop
    int line_num = 1; // Used to report which line an error occured on
    int read;
    while ((read = getchar()) != EOF)
    {
```



```
    // Setup the DFA transitions as a table
    initialize_transition_table();

    // PostScript preable to create a valid ps-file
    printf("<< /PageSize [842 595] >> setpagedevice\n");
    printf("%f %f moveto\n", x, y);

    // Main loop
    int line_num = 1; // Used to report which line an error occured on
    int read;
    while ((read = getchar()) != EOF)
    {
        // Store the read char in the buffer
        lexeme_buffer[lexeme_length++] = read;
        lexeme_buffer[lexeme_length] = 0; // Add NULL terminator

        // Use the current state and the read char to find the next state
        state = transition_table[state][read];

        // Check if we reached the ACCEPT or ERROR states
        switch (state)
        {
        case ACCEPT:
            handle_statement();
            state = START_STATE;
            lexeme_length = 0;
            break;
        case ERROR:
            fprintf(stderr, "error: %d: unrecognized statement: %s\n", line_num, lexeme_buffer);
            exit(EXIT_FAILURE);
        default:
            break;
        }

        // If the char was a newline, the next char will be on a new line!
        if (read == '\n')
            line_num++;
    }

    if (state != START_STATE)
    {
        fprintf(stderr, "error: %d: input ended in the middle of a statement: %s\n", line_num, lexeme_buffer);
        exit(EXIT_FAILURE);
    }

    printf("stroke\n");
    printf("showpage\n");
}
```