Міністерство освіти і науки, молоді та спорту України
Національний Технічний Університет України
“Київський Політехнічний Інститут ім.. Сікорського”
Факультет прикладної математики
Кафедра СПіСКС

Лабораторна робота № 1

з дисципліни

“Інженерія програмного забезпечення. Основи проектування трансляторів”

Тема: “Розробка лексичного аналізатора (ЛА)”

Варіант - 11

Виконав:
Студент групи КВ-72
Садовенко Максим

Київ 2020 р.

Постановка задачі

Розробити програму лексичного аналізатора (ЛА) для підмножини мови програмування SIGNAL. Програма має забезпечувати наступне (якщо це передбачається граматикою варіанту):

- згортання ідентифікаторів;
- згортання ключових слів;
- згортання цілих десяткових констант;
- згортання дійсних десяткових констант;
- згортання строкових констант, якщо вони визначені в заданій

мові; Також у всіх варіантах необхідно забезпечити:

- видалення коментарів, заданих у вигляді (*<текст

коментарю>*) Для кодування лексем необхідно використовувати числові діапазони.

Вид лексеми	**Числовий діапазон**
односимвольні роздільники та знаки операцій (: / ; + і т.д.)	0 – 255, Тобто коди ASCII
багатосимвольні роздільники (:= ,<= ,<=, і т.д.)	301 - 400
цілі десяткові константи	401 - 500
символьні константи	501 - 600
рядкові константи	601 - 700
ключові слова (BEGIN, END, FOR) та ідентифікатори	701 – 1000

Входом ЛА має бути наступне:

- вихідна програма, написана підмножиною мови SIGNAL відповідно до варіанту;
- таблиця кодів ASCII з атрибутами для визначення токенів;
- таблиця багато символьних роздільників;
- таблиця ідентифікаторів, в яку попередньо занесені ключові слова з атрибутом ключового слова;
-

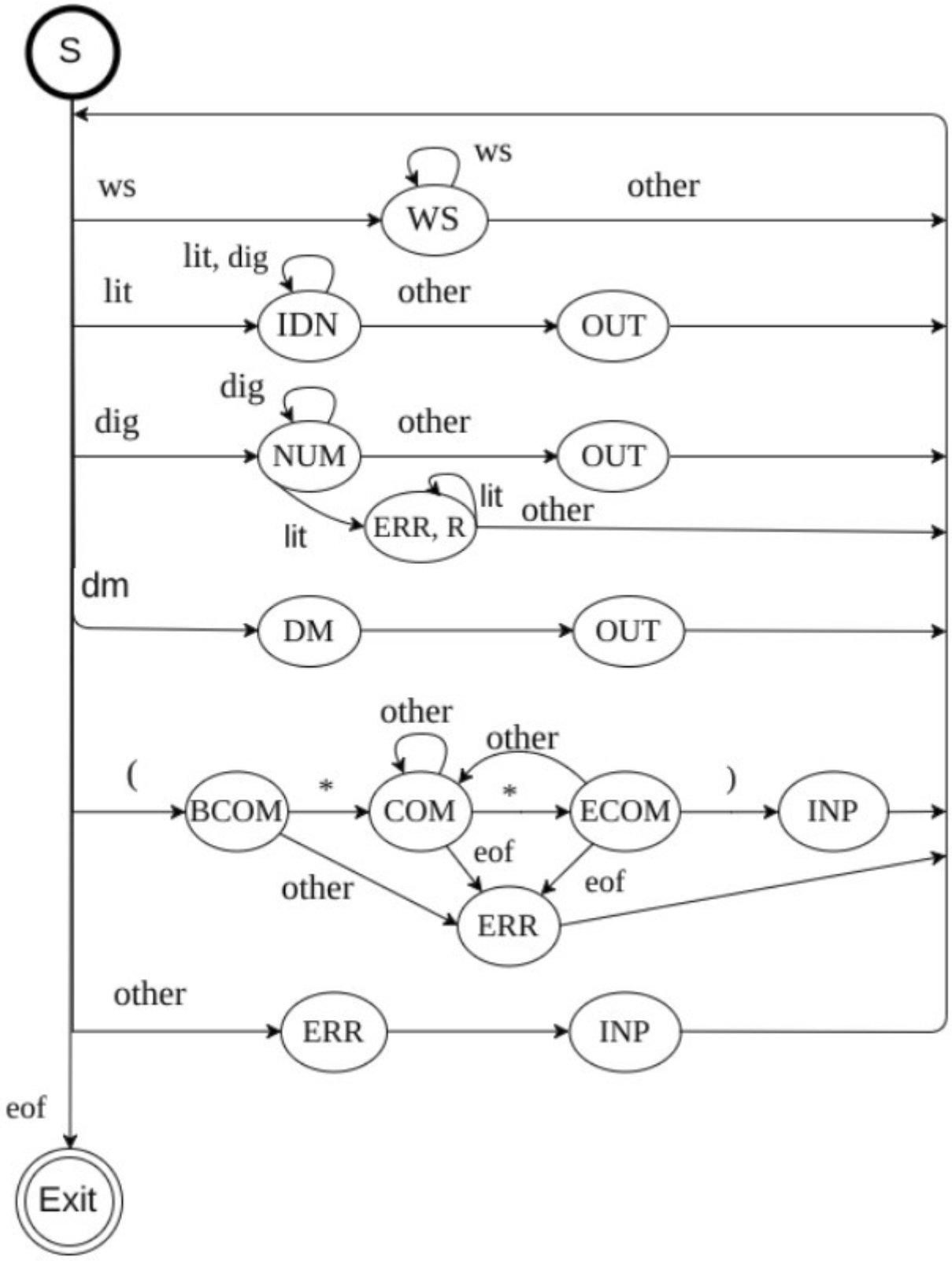
Вихід ЛА має бути наступним:

- закодований рядок лексем;
- таблиці ідентифікаторів, числових, символьних та рядкових констант, сформовані для конкретної програми;

Варіант 11

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
              <block>.
3.    <block> --> <declarations> BEGIN <statements-
              list> END
4.    <declarations> --> <label-declarations>
5.    <label-declarations> --> LABEL <unsigned-
              integer> <labels-list>; |
              <empty>
6.    <labels-list> -->  , <unsigned-integer>
              <labels-list> |
              <empty>
7.    <statements-list> --> <statement> <statements-
              list> |
              <empty>
8.    <statement> --> <unsigned-integer> :
              <statement> |
              GOTO <unsigned-integer> ; |
              LINK <variable-identifier> , <unsigned-
              integer> ; |
              IN <unsigned-integer>;  |
              OUT <unsigned-integer>;
9.    <variable-identifier> --> <identifier>
10.   <procedure-identifier> --> <identifier>
11.   <identifier> --> <letter><string>
12.   <string> --> <letter><string> |
              <digit><string> |
              <empty>
13.   <unsigned-integer> --> <digit><digits-string>
14.   <digits-string> --> <digit><digits-string> |
              <empty>
15.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
              9
16.   <letter> --> A | B | C | D | ... | Z
```

Діаграма переходів ЛА



Код програми

Main.cpp

```
#include "Lexer.hpp"

int main(){
	string file_name;
	cout << "This is lexer of subset of SIGNAL programming language. Made by Maksym Sadovenko." << endl;
	cout << "Write file name:" << endl;
	cin >> file_name;
	Lexer lexer;
	lexer.lexical_analysis(file_name);
	return 0;
}
```

Lexer.hpp

```
#pragma once
#include <iostream>
#include <string>
#include <vector>
#include <iomanip>
#include <fstream>

using namespace std;

struct lexem {
	lexem(int code, int line, int column, string name){
		this->code = code;
		this->line = line;
		this->column = column;
		this->name = name;
	}
	lexem(string name, int code){
		this->name = name;
		this->code = code;
		line = column = 0;
	}
	int code;
	int line;
	int column;
	string name;
};

class Lexer {
public:

	Lexer();
	~Lexer(){};

	bool lexical_analysis(string);
	vector<lexem*> get_lexems() const {return lexem_table;};
	vector<string> get_idetidiers() const {return identifier_talbe;};
	vector<string> get_digits() const {return digit_table;};

private:
	fstream input_file;

	int lex_counter;
	vector <lexem*> lexem_table;
	vector <string> identifier_talbe;
	vector <string> digit_table;
	vector <const lexem*> key_word_table;

	int line;
	int save_line;

	int column;
	int save_column;
```

```
        int lex_error_code;
        int identifier_counter;
        int digit_counter;
        int ASCIIArr[128];
        int pos;
        char sbuff;
        bool was_error;
        void add_lexem(int, int, int, string);
        bool size_out();
        void INP();
        void SPACE();
        void IDN();
        void DG();
        void BCOM(); // begin comment
        void COM(string); // comment
        void ECOM(string); // end comment
        void DM();
        void ERR(string);
        int search_in_digit_table(string);
        int search_in_identifier_table(string);
        int search_in_key_word_table(string);
        void make_listing(string);


        const int MIN_KEY_WORDS_CODE = 401;
        const int MIN_DIGIT_CODE = 501;
        const int MIN_IDENTIFIER_CODE = 1001;
        const int LEX_ERROR_CODE = 2000;

        const lexem* PROGRAM = new lexem("PROGRAM", 401);
        const lexem* BEGIN = new lexem("BEGIN", 402);
        const lexem* END = new lexem("END", 403);
        const lexem* LABEL = new lexem("LABEL", 404);
        const lexem* GOTO = new lexem("GOTO", 405);
        const lexem* LINK = new lexem("LINK", 406);
        const lexem* IN = new lexem("IN", 407);
        const lexem* OUT = new lexem("OUT", 408);



};
```

LexAnaliz.cpp

```
#include "Lexer.hpp"
```

```
Lexer::Lexer() {
	line = 1;
	column = 1;
	digit_counter = MIN_DIGIT_CODE;
	identifier_counter = MIN_IDENTIFIER_CODE;
	was_error = false;
	key_word_table.push_back(PROGRAM);
	key_word_table.push_back(BEGIN);
	key_word_table.push_back(END);
	key_word_table.push_back(LABEL);
	key_word_table.push_back(GOTO);
	key_word_table.push_back(LINK);
	key_word_table.push_back(IN);
	key_word_table.push_back(OUT);

	for (int i = 0; i < 128; i++) {
		if ((i == 9) || (i == 10) || (i == 32) || (i == 12) || (i == 13))
			ASCIIArr[i] = 0;
		else {
			if ((i < 91) && (i > 64))
				ASCIIArr[i] = 1;
			else {
				if ((i < 58) && (i > 47))
					ASCIIArr[i] = 2;
				else {
					if ((i == ')') || (i == '(') || (i == '*') || (i == '.') || (i == ';') || (i == ',') || (i ==
':'))
						ASCIIArr[i] = 3;
					else {
						ASCIIArr[i] = 4;
					}
				}
			}
		}
	}
}

void Lexer::add_lexem(int code, int line, int column, string name) {
	lexem_table.push_back(new lexem(code, line, column, name));
}

bool Lexer::size_out(){
	return !input_file.eof();
}

bool Lexer::lexical_analysis(string filename) {
	input_file.open(filename + ".sig");
	if (!input_file.is_open()) {
		cout << "failed to open " << filename << '\n';
		return false;
	}
```

```
	pos = 0;
	line = 1;
	column = 1;
	sbuff = input_file.get();
	if (size_out())
		INP();
	make_listing(filename);
	return !was_error;
}

void Lexer::INP(){
	while (size_out()){
		switch (ASCIIArr[sbuff]){
		case 0:
			SPACE();
			break;
		case 1:
			IDN();
			break;
		case 2:
			DG();
			break;
		case 3:
			DM();
			break;
		case 4:
			ERR("");
			break;
		}
	}
}

void Lexer::SPACE(){
	if (size_out()){
		while ((size_out()) && (ASCIIArr[sbuff] == 0)){
			if (sbuff == 10){
				pos++;
				column++;
				sbuff = input_file.get();
				line++;
				column = 1;
			}
			else {
				pos++;
				column++;
				sbuff = input_file.get();
			}
		}
		return;
	}
}
```

```
void Lexer::DG(){
	save_line = line;
	save_column = column;
	string Buf;
	while ((size_out()) && (ASCIIArr[sbuff] == 2)){
		Buf += sbuff;
		pos++;
		sbuff = input_file.get();
		column++;
	}

	switch (ASCIIArr[sbuff]){
	case 0:
	case 3:
	case 4:
	{
		int n = search_in_digit_table(Buf);
		if (n == -1){
			add_lexem(digit_counter, save_line, save_column, Buf);
			digit_table.push_back(Buf);
			digit_counter++;
		}
		else
		{
			add_lexem(n + MIN_DIGIT_CODE, save_line, save_column, Buf);
		}
		Buf = "";
		return;
	}
	break;
	case 1:
	case 5:
		ERR(Buf);
		break;
	}
}

void Lexer::IDN(){
	int n;
	save_line = line;
	save_column = column;
	string Buf;
	while ((size_out()) && ((ASCIIArr[sbuff] == 2) || (ASCIIArr[sbuff] == 1))){
		Buf += sbuff;
		pos++;
		sbuff = input_file.get();
		column++;
	}

	if (ASCIIArr[sbuff] == 5){
		ERR(Buf);
```

```
	}
	else
	{
		n = search_in_key_word_table(Buf);
		if (n == -1) {
			n = search_in_identifier_table(Buf);
			if (n == -1)
			{
				add_lexem(identifier_counter, save_line, save_column, Buf);
				identifier_talbe.push_back(Buf);
				identifier_counter++;
			}
			else
			{
				add_lexem(n + MIN_IDENTIFIER_CODE, save_line, save_column, Buf);
			}
		}
		else add_lexem(key_word_table[n]->code, save_line, save_column, key_word_table[n]-
>name);
		Buf = "";
		return;
	}
}

void Lexer::BCOM(){
		pos++;
		sbuff = input_file.get();
		column++;
		COM("(");
}

void Lexer::COM(string Buf){

	while ((size_out()) && (sbuff != '*')){
		Buf += sbuff;
		if (sbuff == '\n'){
			line++;
			column = 1;
			pos++;
			sbuff = input_file.get();
		}
		else {
			pos++;
			sbuff = input_file.get();
			column++;
		}
	}
	if(!size_out()) ERR(Buf);
	else ECOM(Buf);
}

void Lexer::ECOM(string Buf){
```

```
	sbuff = input_file.get();
	pos++;
	column++;

	if ((size_out()) && (sbuff == ')')){
		pos++;
		sbuff = input_file.get();
		column++;
		return;
	}
	else if(!size_out()){
		ERR(Buf);
	}else COM(Buf);

}

void Lexer::DM(){
	save_line = line;
	save_column = column;
	string Buf = "";

	if (sbuff == '('){
		Buf += sbuff;
		pos++;
		sbuff = input_file.get();
		column++;
		if (sbuff == '*') {
			BCOM();
		}
		else {
			add_lexem('(', line, column - 1, Buf);
			return;
		}
	}
	else {
		if (sbuff == '*')
		{
			Buf += sbuff;
			ERR(Buf);
		}
		else
		{
			Buf += sbuff;
			add_lexem(sbuff, line, column, Buf);
			pos++;
			sbuff = input_file.get();
			column++;
			return;
		}
	}
}
```

```
void Lexer::ERR(string pt){
	string Buf = pt;
	was_error = true;
	if ((sbuff == '*') || (sbuff == ')')){
		Buf += sbuff;
		add_lexem(LEX_ERROR_CODE, save_line, save_column, Buf);
		pos++;
		sbuff = input_file.get();
		column++;
		Buf = "";
		INP();
	}
	else
	{
		while ((ASCIIArr[sbuff] != 0) && (ASCIIArr[sbuff] != 3) && (size_out()))
		{
			Buf += sbuff;
			pos++;
			sbuff = input_file.get();
			column++;
		}

		add_lexem(LEX_ERROR_CODE, save_line, save_column, Buf);
		Buf = "";
		INP();
	}
}

int Lexer::search_in_identifier_table(string Ident){
	for (int i = 0; i < identifier_talbe.size(); i++){
		if (identifier_talbe[i] == Ident)
			return i;
	}
	return -1;
}

int Lexer::search_in_digit_table(string Digit){
	for (int i = 0; i < digit_table.size(); i++){
		if (digit_table[i] == Digit)
			return i;
	}
	return -1;
}

int Lexer::search_in_key_word_table(string Ident){
	for (int i = 0; i < key_word_table.size(); i++){
		if (key_word_table[i]->name == Ident)
			return i;
	}
	return -1;
}
```

```
void Lexer::make_listing(string filename) {
	ofstream file(filename + "_generated.txt");
	file << setw(10) << "code" << setw(10) << "line" << setw(10) << "column" << setw(20) << "name"
<< endl << endl;
	for (int i = 0; i < lexem_table.size(); i++){
		file << setw(10) << lexem_table[i]->code << setw(10) << lexem_table[i]->line << setw(10) <<
lexem_table[i]->column << setw(20) << lexem_table[i]->name << endl;
	}
	file << endl << endl;
	int p = 0;
	for (int i = 0; i < lexem_table.size(); i++){
		if (lexem_table[i]->code == LEX_ERROR_CODE) {
			file << "Lexical error on line " << lexem_table[i]->line << " column " <<
lexem_table[i]->column << ": Impossible combination of characters: " << lexem_table[i]->name << endl;
			p++;
		}
	}
	file.close();
	if (p == 0) {
		cout << "lexical analysis completed successfully" << endl;
	}
	else {
		for (unsigned int i = 0; i < lexem_table.size(); i++)
		{
			if (lexem_table[i]->code == LEX_ERROR_CODE) {
				cout << "Lexical error on line " << lexem_table[i]->line << " column " <<
lexem_table[i]->column << ": Impossible combination of characters: " << lexem_table[i]->name << endl;
			}
		}
	}
}
```

Тести

True-тест:

singTrue.sig

```
PROGRAM TEST1;
LABEL 123, 321, 132;
BEGIN
    123: GOTO 321;
    LINK VAR1, 132;
    IN 321;
    ON 132;
    (*comment*)
END.
```

singTrue_generated.txt

code	line	column	name
401	1	1	PROGRAM
1001	1	9	TEST1
59	1	14	;
404	2	1	LABEL
501	2	7	123
44	2	10	,
502	2	12	321
44	2	15	,
503	2	17	132
59	2	20	;
402	3	1	BEGIN
501	4	2	123
58	4	5	:
405	4	7	GOTO
502	4	12	321
59	4	15	;
406	5	2	LINK
1002	5	7	VAR1
44	5	11	,
503	5	13	132
59	5	16	;
407	6	2	IN
502	6	5	321
59	6	8	;
1003	7	2	ON
503	7	5	132
59	7	8	;
403	9	1	END
46	9	4	.

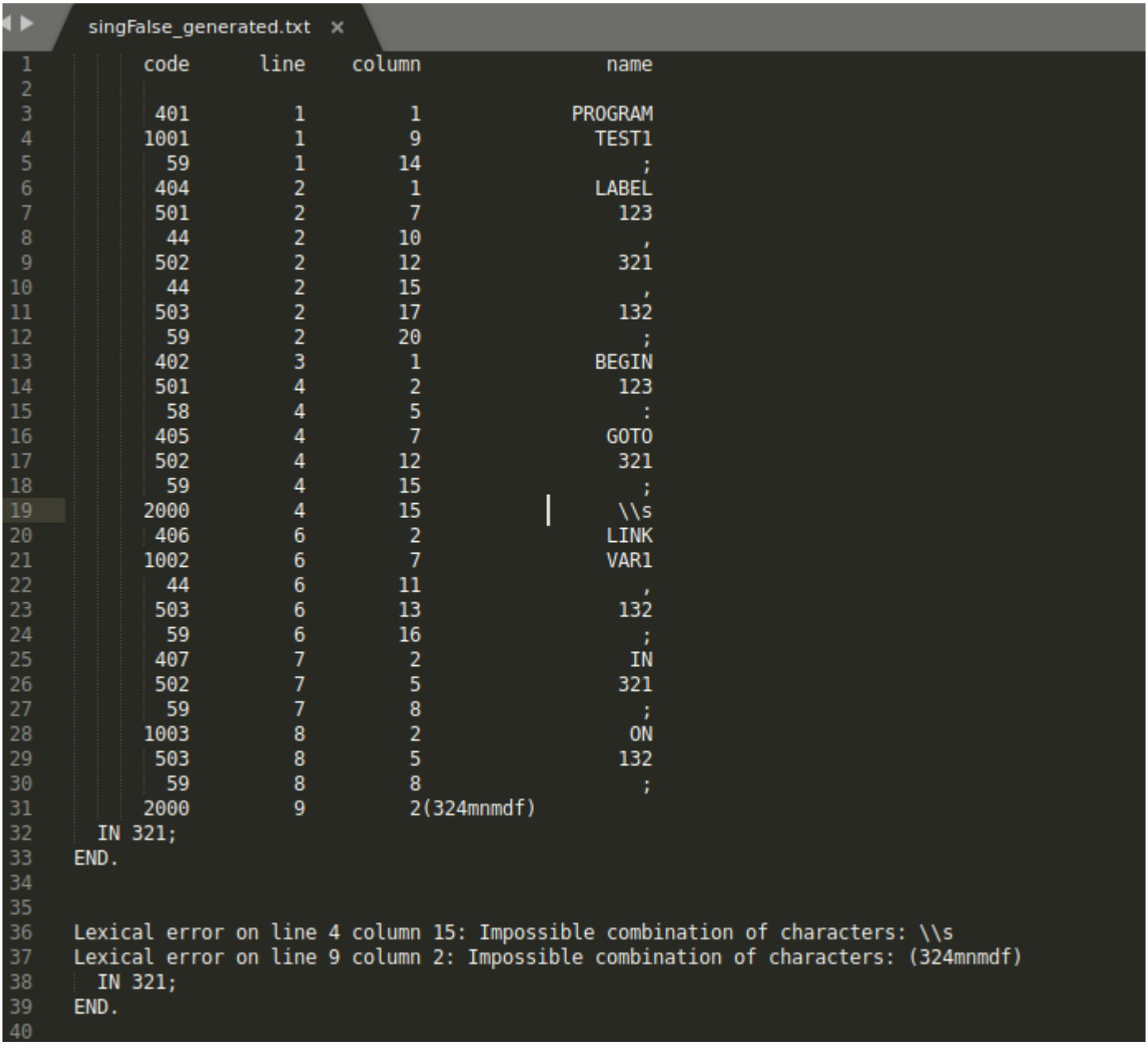
False-тест:

Вхідний файл:

singFalse.sig

```
PROGRAM TEST1;
LABEL 123, 321, 132;
BEGIN
    123: GOTO 321;
    \\s
    LINK VAR1, 132;
    IN 321; (*234*32*)
    ON 132;
    (*324mnmdf)
    IN 321;
END.
```

Вихідний файл:



singFalse_generated.txt

```
    code      line    column                name

     401         1         1             PROGRAM
    1001         1         9               TEST1
      59         1        14                   ;
     404         2         1               LABEL
     501         2         7                 123
      44         2        10                   ,
     502         2        12                 321
      44         2        15                   ,
     503         2        17                 132
      59         2        20                   ;
     402         3         1               BEGIN
     501         4         2                 123
      58         4         5                   :
     405         4         7                GOTO
     502         4        12                 321
      59         4        15                   ;
    2000         4        15                  \\s
     406         6         2                LINK
    1002         6         7                VAR1
      44         6        11                   ,
     503         6        13                 132
      59         6        16                   ;
     407         7         2                  IN
     502         7         5                 321
      59         7         8                   ;
    1003         8         2                  ON
     503         8         5                 132
      59         8         8                   ;
    2000         9         2(324mnmdf)
  IN 321;
END.


Lexical error on line 4 column 15: Impossible combination of characters: \\s
Lexical error on line 9 column 2: Impossible combination of characters: (324mnmdf)
  IN 321;
END.
```