Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» Факультет прикладної математики Кафедра системного програмування і спеціалізованих комп'ютерних системи

Розрахунково-графічна робота

«РОЗРОБКА СИНТАКСИЧНОГО АНАЛІЗАТОРА»

з дисципліни

«ОСНОВИ ПРОЕКТУВАННЯ ТРАНСЛЯТОРІВ»

Виконав: студент групи КВ-83

Швидкий О.С.

Перевірив: Северін С.

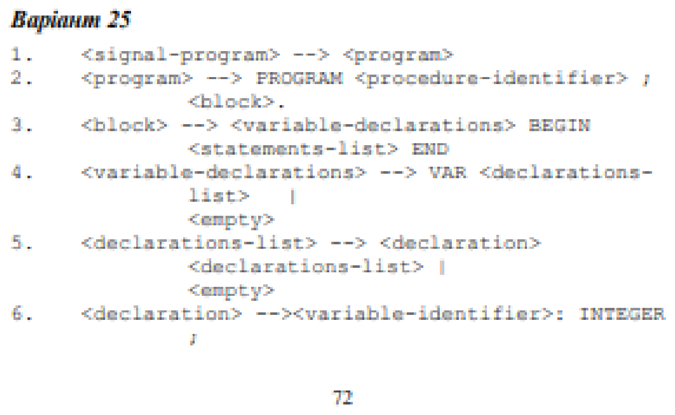
Київ – 2021

Загальне завдання

Розробити програму синтаксичного аналізатора для підмножини мови програмування SIGNAL.

Варіант 25

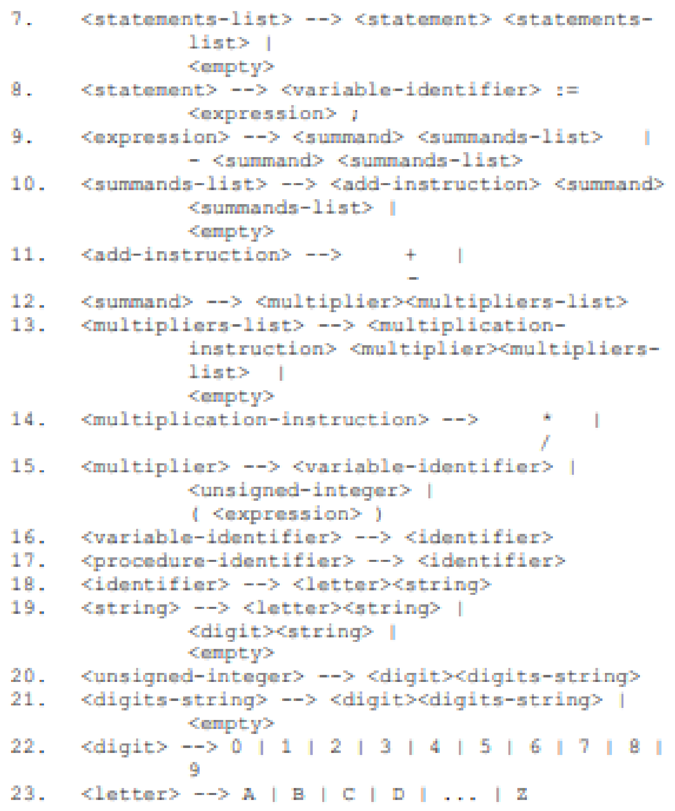
Граматика підмножини мови програмування SIGNAL:



Варіант 25

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
              <block>.
3.    <block> --> <variable-declarations> BEGIN
              <statements-list> END
4.    <variable-declarations> --> VAR <declarations-
              list>   |
              <empty>
5.    <declarations-list> --> <declaration>
              <declarations-list> |
              <empty>
6.    <declaration> --><variable-identifier>: INTEGER
              ;
```

72



```
7.    <statements-list> --> <statement> <statements-
              list> |
              <empty>
8.    <statement> --> <variable-identifier> :=
              <expression> ;
9.    <expression> --> <summand> <summands-list>   |
              - <summand> <summands-list>
10.   <summands-list> --> <add-instruction> <summand>
              <summands-list> |
              <empty>
11.   <add-instruction> -->      +   |
                                 -
12.   <summand> --> <multiplier><multipliers-list>
13.   <multipliers-list> --> <multiplication-
              instruction> <multiplier><multipliers-
              list>  |
              <empty>
14.   <multiplication-instruction> -->      *   |
                                            /
15.   <multiplier> --> <variable-identifier> |
              <unsigned-integer> |
              ( <expression> )
16.   <variable-identifier> --> <identifier>
17.   <procedure-identifier> --> <identifier>
18.   <identifier> --> <letter><string>
19.   <string> --> <letter><string> |
              <digit><string> |
              <empty>
20.   <unsigned-integer> --> <digit><digits-string>
21.   <digits-string> --> <digit><digits-string> |
              <empty>
22.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
              9
23.   <letter> --> A | B | C | D | ... | Z
```

Лістинг програми мовою C++

rgr.cpp

```cpp
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include "stdlib.h"
#include <string>
#include <iomanip>
#include "lexer.h"
#include "parser.h"
using namespace std;


int main(int argc, char** argv)
{

	for (int i = 1; i < argc; i++) {
		Lexer lexer;

		lexer = Lexer(argv[i]);
		Parser parser = Parser(argv[i], lexer.parse());
		parser.parse();
	}
}
```

Lexer.cpp

```cpp
#include "lexer.h"

int SymbolCategories[255];
int tabulation = 4;
int unsintegers = 501;
int identifiers = 1001;
void fillAscii() {
	for (int i = 0; i < 255; i++) {

		if (i >= 8 && i <= 13 || i == 32) {
			SymbolCategories[i] = 0;
			continue;
		}

		if (i >= 48 && i <= 57) {
			SymbolCategories[i] = 1;
			continue;
		}

		if (i >= 65 && i <= 90) {
			SymbolCategories[i] = 2;
			continue;
		}

		if ((i >= 40 && i <= 47) || (i >= 59 && i <= 62)) {
			if (i == 40) { SymbolCategories[i] = 5; }
			else { SymbolCategories[i] = 3; }
			continue;
		}
		if (i == 58) {
			SymbolCategories[i] = 4;
			continue;
		}

		if ((i >= 0 && i <= 5) || i >= 91 || (i >= 62 && i <= 64) || i == 60 || (i >= 
33 && i <= 39) || i == 44) {
			SymbolCategories[i] = 6;
			continue;
		}

	}
	/*
	cout << "id\t group\t symb" << endl;
	for (int i = 0; i < 255; i++) {
		cout << i << "| \t" << SymbolCategories[i] << "| \t" << (char)i << endl;
	}*/
}

void showLexeme(int row, int column, string lexeme, int ascii, string inputFile) {

	string tmp = "./" + inputFile + "/generated.txt";
	fstream output;

	output.open(tmp, fstream::in | fstream::out | fstream::app);
	output << setw(10) << row << setw(5) << " " << column << setw(5) << " " << ascii << 
setw(5) << " " << lexeme << endl;
	//cout << setw(10) << lexeme << setw(5) << " " << row << setw(5) << " " << column << 
setw(5) << " " << ascii << endl;
	output.close();



}
void showError(string error, string inputFile) {
	string tmp = "./" + inputFile + "/generated.txt";
	fstream output;

	output.open(tmp, fstream::in | fstream::out | fstream::app);
	output << error << endl;
}

Lexer::Lexer(string inputFile) {
	string tmp = "./" + inputFile + "/input.sig";
	strcpy(filePath, tmp.c_str());
	cout << filePath << endl;
	fileName = inputFile;
	input = ifstream(filePath);


}

void clearFile(string inputFile) {
	string tmp = "./" + inputFile + "/generated.txt";
	fstream output;
	output.open(tmp, fstream::in | fstream::out | fstream::trunc);
	output.close();
}
```

```
vector<Lexeme> Lexer::parse() {
	vector<Lexeme> lexemeList;
	clearFile(fileName);
	fillAscii();
	int row = 1;
	int col = 1;
	char buff;

	while (1) {
		input.get(buff);
		if (input.eof()) { break; }
		switch (SymbolCategories[(int)buff])
		{
		case 0://пробелы и символы приравненные к пробелам
		{
			switch ((int)buff)
			{
			case 10: {
				row++;
				col = 1;
				break;
			}
			case 9: {
				col += tabulation;
				break;
			}
			case 32: {
				col++;
				break;
			}
			}
			break;
		}
		case 1: {//числовые костанты
			int position = col;
			string token;
			do {

				if (SymbolCategories[(int)buff] == 2) {
					string error = "Lexer: ERROR! Detected letter after number 
in row:" + to_string(row) + " col:" + to_string(col);
					showError(error, fileName);
				}
				token += buff;
				col++;

			} while (input.get(buff) && SymbolCategories[(int)buff] == 1 || 
SymbolCategories[(int)buff] == 2);

			input.seekg(-1, ios_base::cur);
			auto type = lexemes[token];


			if (!type) {

				lexemes[token] = unsintegers;
				Lexeme newLexeme;
				newLexeme.row = row;
				newLexeme.col = position;
				newLexeme.category = unsintegers++;
				newLexeme.token = token;
```

```
					lexemeList.push_back(newLexeme);
					//showLexeme(row, position, token, unsintegers++, fileName);
				}
				else {
					//showLexeme(row, position, token, type, fileName);
					Lexeme newLexeme;
					newLexeme.row = row;
					newLexeme.col = position;
					newLexeme.category = type;
					newLexeme.token = token;
					lexemeList.push_back(newLexeme);
				}

				break;
			}

			case 2: {
				int position = col;
				string token;
				do {
					token += buff;
					col++;

				} while (input.get(buff) && SymbolCategories[(int)buff] == 1 || 
SymbolCategories[(int)buff] == 2);
				input.seekg(-1, ios_base::cur);
				auto type = lexemes[token];


				if (!type) {//не найдено существующей лексемы в списке, добавляем новую 
в список
					lexemes[token] = identifiers;
					//showLexeme(row, position, token, identifiers++, fileName);
					Lexeme newLexeme;
					newLexeme.row = row;
					newLexeme.col = position;
					newLexeme.category = identifiers++;
					newLexeme.token = token;
					lexemeList.push_back(newLexeme);
				}
				else {

					//showLexeme(row, position, token, type, fileName);
					Lexeme newLexeme;
					newLexeme.row = row;
					newLexeme.col = position;
					newLexeme.category = type;
					newLexeme.token = token;
					lexemeList.push_back(newLexeme);
				}
				break;

			}
			case 3: {
				//showLexeme(row, col, string{ buff }, (int)buff, fileName);
				Lexeme newLexeme;
				newLexeme.row = row;
				newLexeme.col = col;
				newLexeme.category = (int)buff;
				newLexeme.token = string{ buff };
				lexemeList.push_back(newLexeme);
				col++;
				break;
			}
			case 4: {
				string token = string{ buff };
				int position = col;
				col++;
				if (input.get(buff) && buff == '=') {
					token += string{ buff };
					col++;
					//showLexeme(row, position, token, 301, fileName);
					Lexeme newLexeme;
					newLexeme.row = row;
					newLexeme.col = position;
					newLexeme.category = 301;
					newLexeme.token = token;
					lexemeList.push_back(newLexeme);
				}
				else {
					input.seekg(-1, ios_base::cur);

					//showLexeme(row, position, token, (int)token[0], fileName);
					Lexeme newLexeme;
					newLexeme.row = row;
					newLexeme.col = position;
					newLexeme.category = (int)token[0];
					newLexeme.token = token;
					lexemeList.push_back(newLexeme);
				}

				break;
			}
			case 5: {
				col++;
				if (input.get(buff) && buff == '*') {
					int startComRow = row;
					int startComCol = col;
					col++;
					while (1) {
						while (input.get(buff) && buff != '*') //читаем
комментарий пока не найдем звездочку
						{
							if (input.eof()) { showError("Lexer: ERROR! Comment
on row:" + to_string(startComRow) + " col:" + to_string(startComCol) + " - unexpected end of
file in unclosed comment! ", fileName); }

							if ((int)buff == 10) {
								row++;
								col = 1;
							}
							else if ((int)buff == 9) { col += tabulation; }
							else {
								col++;
							}

						}
						if (input.eof()) { showError("Lexer: ERROR! Comment on
row:" + to_string(startComRow) + " col:" + to_string(startComCol) + " - unexpected end of
file in unclosed comment! ", fileName); }

						col++;//нашли звездочку, инкремент по столбику
```

```
								if (input.get(buff) && buff != ')') {

									//если не закрывающая скобка, комментарий не 
закрываем, проверяем на следующую звездочку, если и там не звездочка то продолжаем читать 
комментарий до звездочки
									if (buff != '*') {
										input.seekg(-1, ios_base::cur);
										continue;
									}
									else {
										//если всё таки звездочка, продолжаем 
считывать звездочки

										col++;
										while (input.get(buff) && buff == '*')
										{
											col++;
											if (input.eof()) { showError("Lexer: 
ERROR! Comment on row:" + to_string(startComRow) + " col:" + to_string(startComCol) + " - 
unexpected end of file in unclosed comment! ", fileName); }
										}
										if (input.eof()) { showError("Lexer: ERROR! 
Comment on row:" + to_string(startComRow) + " col:" + to_string(startComCol) + " - 
unexpected end of file in unclosed comment! ", fileName); }

										if (buff == ')') { col++; break; }//после 
ряда звездочек нашли скобку, end comment
										else { input.seekg(-1, ios_base::cur); 
continue; }//не скобка - сдвигаем каретку назад и читаем символы


									}
								}
								else {
									col++;
									//если закрывающая скобка после звездочки, 
закончили комментарий
									break;
								}

							}
						}
						else {
							input.seekg(-1, ios_base::cur);
							//showLexeme(row, col, string{ '(' }, (int)'(', fileName);
							Lexeme newLexeme;
							newLexeme.row = row;
							newLexeme.col = col;
							newLexeme.category = (int)'(';
							newLexeme.token = string{ '(' };
							lexemeList.push_back(newLexeme);

						}
						break;
				}
				case 6: {
						showError("Lexer: ERROR! Detected illegal symbol " + string{ buff } + " 
at col:" + to_string(col) + " row: " + to_string(row), fileName);
				}

				}
```

```
	}

	for (int i = 0; i < lexemeList.size(); i++) {
		cout << setw(10) << lexemeList[i].row << setw(5) << " " << lexemeList[i].col
<< setw(5) << " " << lexemeList[i].category << setw(5) << " " << lexemeList[i].token <<
endl;
	}

	return lexemeList;


}
```

Parser.cpp

```
#include "parser.h"


void generateData(string data, string fileName) {
	string tmp = "./" + fileName + "/generated.txt";
	fstream output;


	output.open(tmp, fstream::in | fstream::out | fstream::app);
	output << data << endl;
}


void generateError(string error, string fileName) {
	string tmp = "./" + fileName + "/generated.txt";
	fstream output;


	output.open(tmp, fstream::in | fstream::out | fstream::app);
	output << error << endl;
```

```
}



Parser::Parser(string file, vector<Lexeme> input) {
	this->fileName = file;
	this->lexemes = input;
}

void Parser::parse() {
	TreeNode* node = new TreeNode;
	string depth = "";
	node->keyword = "<signal-program>";
	generateData(depth + node->keyword, this->fileName);
	program(node, depth + "..");
}

void Parser::program(TreeNode* parent, string depth) {
	TreeNode* node = new TreeNode;
	node->keyword = "<program>";
	generateData(depth + node->keyword, this->fileName);
	if (this->lexemes.size() <= this->i) {
		generateError("Syntax-analyzer! ERROR! Unexpected end of
file", this->fileName);
```

```
            parent->next.push_back(node);
            return;
        }

        if (this->lexemes[this->i].category != 401) {
            generateError("Syntax-analyzer! ERROR! 'PROGRAM' identifier
not found at row:" + to_string(this->lexemes[this->i].row) + "col:" +
to_string(this->lexemes[this->i].col), this->fileName);
            parent->next.push_back(node);
            return;
        }

        node->lexemes.push_back(&this->lexemes[this->i]);
        generateData(depth + ".." + node->lexemes[0]->token + "  " +
to_string(node->lexemes[0]->category), this->fileName);
        this->i++;

        procedure_identifier(node, depth + "..");

        if (this->lexemes.size() <= this->i) {
            generateError("Syntax-analyzer! ERROR! Unexpected end of
file", this->fileName);
            parent->next.push_back(node);
            return;
        }


        if (this->lexemes[this->i].category != 59) {
                generateError("Syntax-analyzer! ERROR! semicolon(;) not found at row:" + to_string(this->lexemes[this->i].row) + "col:" + to_string(this->lexemes[this->i].col), this->fileName);
                parent->next.push_back(node);


                return;
        }


        node->lexemes.push_back(&this->lexemes[this->i]);
        generateData(depth + ".." + node->lexemes[1]->token + "  " + to_string(node->lexemes[0]->category), this->fileName);
        this->i++;


        block(node, depth + "..");


        if (this->lexemes.size() <= this->i) {
                generateError("Syntax-analyzer! ERROR! Unexpected end of file", this->fileName);
                parent->next.push_back(node);
                return;
        }
        if (this->lexemes[this->i].category != 46) {
                generateError("Syntax-analyzer! ERROR! '.' not found at row:"
+ to_string(this->lexemes[this->i].row) + "col:" + to_string(this-
>lexemes[this->i].col), this->fileName);
                parent->next.push_back(node);

                return;
        }

        node->lexemes.push_back(&this->lexemes[this->i]);
        generateData(depth + ".." + node->lexemes[2]->token + "  " +
to_string(node->lexemes[0]->category), this->fileName);
        this->i++;

        parent->next.push_back(node);
}
void Parser::procedure_identifier(TreeNode* parent, string depth) {
        TreeNode* node = new TreeNode;
        node->keyword = "<procedure-identifier>";
        generateData(depth + node->keyword, this->fileName);
        identifier(node, depth + "..");
        parent->next.push_back(node);
```

```
}


void Parser::identifier(TreeNode* parent, string depth) {
	TreeNode* node = new TreeNode;
	node->keyword = "<identifier>";
	generateData(depth + node->keyword, this->fileName);
	if (this->lexemes[this->i].category >= 1001) {
		node->lexemes.push_back(&this->lexemes[this->i]);
		generateData(depth + ".." + node->lexemes[0]->token + "  " + 
to_string(node->lexemes[0]->category), this->fileName);
	}
	else {
		generateError("Syntax-analyzer! ERROR! identifier not found at 
row:" + to_string(this->lexemes[this->i].row) + "col:" + to_string(this-
>lexemes[this->i].col), this->fileName);
	}
	this->i++;


	parent->next.push_back(node);


}
void Parser::block(TreeNode* parent, string depth) {
	TreeNode* node = new TreeNode;
```

```
        node->keyword = "<block>";
        generateData(depth + node->keyword, this->fileName);
        variable_declarations(node, depth + "..");
        if (this->lexemes[this->i].category != 402) {
                generateError("Syntax-analyzer! ERROR! 'BEGIN' identifier not
found  at row:" + to_string(this->lexemes[this->i].row) + "col:" +
to_string(this->lexemes[this->i].col), this->fileName);
                parent->next.push_back(node);
                return;
        }
        else {
                node->lexemes.push_back(&this->lexemes[this->i]);
                generateData(depth + ".." + node->lexemes[0]->token + "  " +
to_string(node->lexemes[0]->category), this->fileName);
                this->i++;
                statements_list(node, depth + "..");

                if (this->lexemes.size() <= this->i) { return; }
                if (this->lexemes[this->i].category != 403) {
                        generateError("Syntax-analyzer! ERROR! 'END' identifier
not found  at row:" + to_string(this->lexemes[this->i].row) + "col:" +
to_string(this->lexemes[this->i].col), this->fileName);
                        parent->next.push_back(node);
                        return;
                }
                node->lexemes.push_back(&this->lexemes[this->i]);
                generateData(depth + ".." + node->lexemes[1]->token + "  " +
to_string(node->lexemes[1]->category), this->fileName);
                this->i++;
                parent->next.push_back(node);
        }



}


void Parser::variable_declarations(TreeNode* parent, string depth) {
        TreeNode* node = new TreeNode;
        node->keyword = "<variable-declarations>";
        generateData(depth + node->keyword, this->fileName);
        if (this->lexemes[this->i].category != 404 && this->lexemes[this-
>i].category != 402) {
                generateError("Syntax-analyzer! ERROR! 'VAR' identifier not
found and unexpected token at row:" + to_string(this->lexemes[this-
>i].row) + "col:" + to_string(this->lexemes[this->i].col), this->fileName);


                parent->next.push_back(node);
                return;
        }
        else if (this->lexemes[this->i].category == 402) {
                empty(node, depth + "..");

                parent->next.push_back(node);

                return;

        }

        else {

                node->lexemes.push_back(&this->lexemes[this->i]);

                generateData(depth + ".." + node->lexemes[0]->token + "  " +
to_string(node->lexemes[0]->category), this->fileName);

                this->i++;

                declaration_list(node, depth + "..");

                parent->next.push_back(node);

        }

}


void Parser::empty(TreeNode* parent, string depth) {

        TreeNode* node = new TreeNode;

        node->keyword = "<empty>";

        generateData(depth + node->keyword, this->fileName);

        parent->next.push_back(node);

}



void Parser::declaration_list(TreeNode* parent, string depth) {
	TreeNode* node = new TreeNode;
	node->keyword = "<declaration-list>";
	generateData(depth + node->keyword, this->fileName);



	if (this->lexemes[this->i].category >= 1001) {
		declaration(node, depth + "..");
	}
	parent->next.push_back(node);
}

void Parser::declaration(TreeNode* parent, string depth) {
	TreeNode* node = new TreeNode;
	node->keyword = "<declaration>";
	generateData(depth + node->keyword, this->fileName);
	variable_identifier(node, depth + "..");
	if (this->lexemes[this->i].category != 58) {
		generateError("Syntax-analyzer! ERROR! ':' not found and
unexpected token at row:" + to_string(this->lexemes[this->i].row) + "col:"
+ to_string(this->lexemes[this->i].col), this->fileName);
		parent->next.push_back(node);
		return;
	}
```

```
        node->lexemes.push_back(&this->lexemes[this->i]);
        generateData(depth + ".." + node->lexemes[0]->token + "  " +
to_string(node->lexemes[0]->category), this->fileName);
        this->i++;

        if (this->lexemes[this->i].category != 405) {
                generateError("Syntax-analyzer! ERROR! 'INTEGER' not found
and unexpected token at row:" + to_string(this->lexemes[this->i].row) +
"col:" + to_string(this->lexemes[this->i].col), this->fileName);
                parent->next.push_back(node);
                return;
        }

        node->lexemes.push_back(&this->lexemes[this->i]);
        generateData(depth + ".." + node->lexemes[1]->token + "  " +
to_string(node->lexemes[1]->category), this->fileName);
        this->i++;

        if (this->lexemes[this->i].category != 59) {
                generateError("Syntax-analyzer! ERROR! ';'  not found and
unexpected token at row:" + to_string(this->lexemes[this->i].row) + "col:"
+ to_string(this->lexemes[this->i].col), this->fileName);
                parent->next.push_back(node);
                return;
        }

        node->lexemes.push_back(&this->lexemes[this->i]);
        generateData(depth + ".." + node->lexemes[2]->token + "  " +
    to_string(node->lexemes[1]->category), this->fileName);
        this->i++;




        parent->next.push_back(node);
        if (this->lexemes[this->i].category >= 1001) {
                declaration(parent, depth);
        }

}
void Parser::variable_identifier(TreeNode* parent, string depth) {
        TreeNode* node = new TreeNode;
        node->keyword = "<variable_identifier>";
        generateData(depth + node->keyword, this->fileName);
        identifier(node, depth + "..");
        parent->next.push_back(node);
}
```

```
void Parser::statements_list(TreeNode* parent, string depth) {
	TreeNode* node = new TreeNode;
	node->keyword = "<statements_list>";
	generateData(depth + node->keyword, this->fileName);
	if (this->lexemes.size() <= this->i) { return; }
	if (this->lexemes[this->i].category == 403) {
		empty(node, depth + "..");
		return;
	}
	statement(node, depth + "..");
	parent->next.push_back(node);
}

void Parser::statement(TreeNode* parent, string depth) {
	TreeNode* node = new TreeNode;
	node->keyword = "<statement>";
	generateData(depth + node->keyword, this->fileName);
	variable_identifier(node, depth + "..");
	if (this->lexemes.size() <= this->i) { return; }
	if (this->lexemes[this->i].category != 301) {
		generateError("Syntax-analyzer! ERROR! ':=' not found in
statement at row:" + to_string(this->lexemes[this->i].row) + "col:" +
to_string(this->lexemes[this->i].col), this->fileName);
```

```
            parent->next.push_back(node);

            return;

        }

        node->lexemes.push_back(&this->lexemes[this->i]);

        generateData(depth + ".." + node->lexemes[0]->token + "  " +
to_string(node->lexemes[0]->category), this->fileName);

        this->i++;

        expression(node, depth + "..");

        if (this->lexemes.size() <= this->i) { return; }

        if (this->lexemes[this->i].category != 59) {

            generateError("Syntax-analyzer! ERROR! semicolon(;) not
found in end of statement at row:" + to_string(this->lexemes[this->i].row)
+ "col:" + to_string(this->lexemes[this->i].col), this->fileName);

            parent->next.push_back(node);

            return;

        }

        node->lexemes.push_back(&this->lexemes[this->i]);

        generateData(depth + ".." + node->lexemes[1]->token + "  " +
to_string(node->lexemes[1]->category), this->fileName);

        this->i++;


        parent->next.push_back(node);

        if (this->lexemes[this->i].category >= 1001) {

            statement(parent, depth);
```

```
	}
}

void Parser::expression(TreeNode* parent, string depth) {
	TreeNode* node = new TreeNode;
	node->keyword = "<expression>";
	generateData(depth + node->keyword, this->fileName);
	if (this->lexemes.size() <= this->i) { return; }
	if (this->lexemes[this->i].category == 45) {
		node->lexemes.push_back(&this->lexemes[this->i]);
		generateData(depth + ".." + node->lexemes[0]->token + "  " +
to_string(node->lexemes[0]->category), this->fileName);
		this->i++;
	}
	summand(node, depth + "..");
	summands_list(node, depth + "..");

	parent->next.push_back(node);

}

void Parser::summand(TreeNode* parent, string depth) {
	TreeNode* node = new TreeNode;
```

```
        node->keyword = "<summand>";

        generateData(depth + node->keyword, this->fileName);

        multiplier(node, depth + "..");

        multipliers_list(node, depth + "..");

        parent->next.push_back(node);


}

void Parser::summands_list(TreeNode* parent, string depth) {

        TreeNode* node = new TreeNode;

        node->keyword = "<summands-list>";

        generateData(depth + node->keyword, this->fileName);

        if (this->lexemes.size() <= this->i) { return; }

        if (this->lexemes[this->i].category == 45 || this->lexemes[this-
>i].category == 43) {

                node->lexemes.push_back(&this->lexemes[this->i]);

                generateData(depth + ".." + node->lexemes[0]->token + "  " +
to_string(node->lexemes[0]->category), this->fileName);

                this->i++;

        }

        else {

                generateError("Syntax-analyzer! ERROR! add instruction not
found at row:" + to_string(this->lexemes[this->i].row) + "col:" +
to_string(this->lexemes[this->i].col), this->fileName);

                parent->next.push_back(node);
                return;
        }
        summand(node, depth + "..");
        if (this->lexemes.size() <= this->i) { return; }

        if (this->lexemes[this->i].category != 59 && this->lexemes[this-
>i].category != 41) { summands_list(node, depth + ".."); }

        parent->next.push_back(node);
}

void Parser::multiplier(TreeNode* parent, string depth) {
        TreeNode* node = new TreeNode;
        node->keyword = "<multiplier>";
        generateData(depth + node->keyword, this->fileName);
        if (this->lexemes[this->i].category >= 1001) {
variable_identifier(node, depth + ".."); }
        else if (this->lexemes[this->i].category >= 501) {
                unsigned_integer(node, depth + "..");
        }
        else if (this->lexemes[this->i].category == 40) {
                node->lexemes.push_back(&this->lexemes[this->i]);
                generateData(depth + ".." + node->lexemes[0]->token + "  " +
to_string(node->lexemes[0]->category), this->fileName);
```

```
                this->i++;


                expression(node, depth + "..");


                node->lexemes.push_back(&this->lexemes[this->i]);
                generateData(depth + ".." + node->lexemes[1]->token + "  " +
to_string(node->lexemes[1]->category), this->fileName);
                this->i++;
        }
        else {
                generateError("Syntax-analyzer! ERROR! multiplier error at
row:" + to_string(this->lexemes[this->i].row) + "col:" + to_string(this-
>lexemes[this->i].col), this->fileName);
        }


        parent->next.push_back(node);
}


void Parser::multipliers_list(TreeNode* parent, string depth) {
        TreeNode* node = new TreeNode;
        node->keyword = "<multipliers-list>";
        generateData(depth + node->keyword, this->fileName);
        if (this->lexemes.size() <= this->i) { return; }
        if (this->lexemes[this->i].category == 42 || this->lexemes[this->i].category == 47) {
                node->lexemes.push_back(&this->lexemes[this->i]);
                generateData(depth + ".." + node->lexemes[0]->token + "  " + to_string(node->lexemes[0]->category), this->fileName);
                this->i++;
        }
        else {
                empty(node, depth + "..");
                parent->next.push_back(node);
                return;
        }

        multiplier(node, depth + "..");
        if (this->lexemes[this->i].category == 42 || this->lexemes[this->i].category == 47) {
                multipliers_list(node, depth + "..");
        }
        parent->next.push_back(node);
}

void Parser::unsigned_integer(TreeNode* parent, string depth) {
        TreeNode* node = new TreeNode;
        node->keyword = "<unsigned-integer>";
```

```
	generateData(depth + node->keyword, this->fileName);

	node->lexemes.push_back(&this->lexemes[this->i]);

	generateData(depth + ".." + node->lexemes[0]->token + "  " +
to_string(node->lexemes[0]->category), this->fileName);

	this->i++;

	parent->next.push_back(node);


}
```

Lexer.h

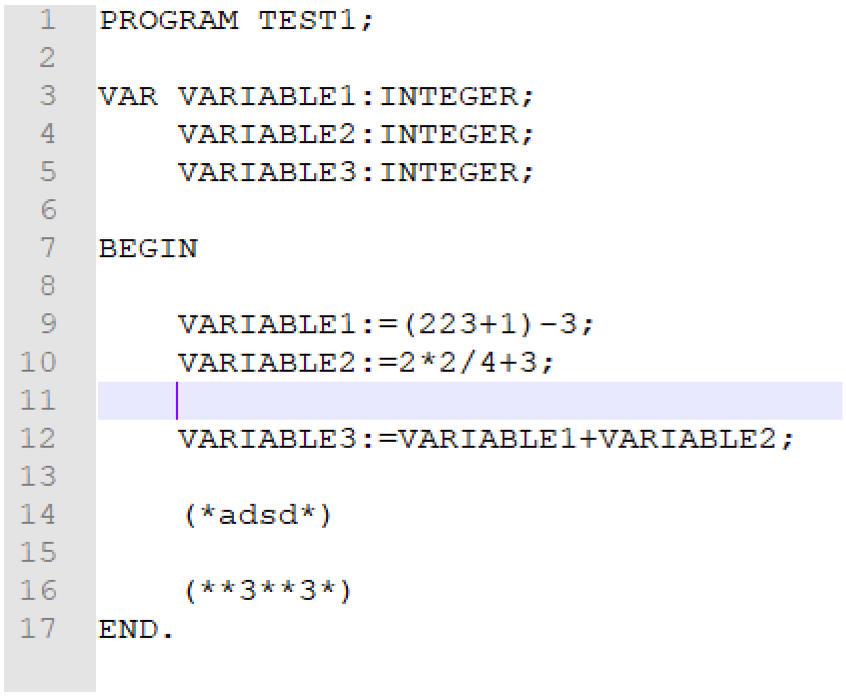
```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <map>
#include <string>
#include <fstream>
#include <iomanip>
#include <vector>
#pragma once
using namespace std;
struct Lexeme
{
	int row;
	int col;
	int category;
	string token;
};


class Lexer {
private:
	char filePath[100];
	ifstream input;
	string fileName;
	int unsintegers = 501;
	int identifiers = 1001;

public:
	map <string, int> lexemes =
	{
		{"PROGRAM", 401},
		{"BEGIN",402},
		{"END",403},
		{"VAR",404},
		{"INTEGER",405},
	};
	Lexer() {
	}
	Lexer(string inputFile);
	vector<Lexeme> parse();
                                        };
```

Parser.h

```
#pragma once
#include <iostream>
#include <string>
#include <fstream>
#include "lexer.h"
using namespace std;

struct TreeNode {
	string keyword;
	vector<Lexeme*> lexemes;
	vector<TreeNode*> next;
};

class Parser {
private:
	string fileName;
	vector<Lexeme> lexemes;
	int i = 0;
public:
	Parser(string file, vector<Lexeme> input);
	void parse();
	void program(TreeNode*, string);
	void procedure_identifier(TreeNode*, string);
	void block(TreeNode*, string);
	void variable_declarations(TreeNode*, string);
	void declaration_list(TreeNode*, string);
	void empty(TreeNode*, string);
	void identifier(TreeNode*, string);
	void statements_list(TreeNode*, string);
	void declaration(TreeNode*, string);
	void variable_identifier(TreeNode*, string);
	void statement(TreeNode*, string);

	void expression(TreeNode*, string);
	void summands_list(TreeNode*, string);
	void summand(TreeNode*, string);
	void multiplier(TreeNode*, string);
	void multipliers_list(TreeNode*, string);
	void unsigned_integer(TreeNode*, string);


};
```

Тести

1.



```
PROGRAM TEST1;

VAR VARIABLE1:INTEGER;
    VARIABLE2:INTEGER;
    VARIABLE3:INTEGER;

BEGIN

    VARIABLE1:=(223+1)-3;
    VARIABLE2:=2*2/4+3;

    VARIABLE3:=VARIABLE1+VARIABLE2;

     (*adsd*)

     (**3**3*)
END.
```

```
<signal-program>
..<program>
....PROGRAM  401
....<procedure-identifier>
......<identifier>
........TEST1  1001
....;  401
....<block>
......<variable-declarations>
........VAR  404
........<declaration-list>
..........<declaration>
............<variable_identifier>
..............<identifier>
................VARIABLE1  1002
............:  58
............INTEGER  405
............;  405
..........<declaration>
............<variable_identifier>
..............<identifier>
................VARIABLE2  1003
............:  58
```

```
............INTEGER  405
............;  405
..........<declaration>
............<variable_identifier>
..............<identifier>
................VARIABLE3  1004
............:  58
............INTEGER  405
............;  405
......BEGIN  402
......<statements_list>
........<statement>
..........<variable_identifier>
............<identifier>
..............VARIABLE1  1002
..........:=  301
..........<expression>
............<summand>
..............<multiplier>
................(  40
................<expression>
..................<summand>
....................<multiplier>
......................<unsigned-integer>
........................223  501
....................<multipliers-list>
......................<empty>
..................<summands-list>
....................+  43
....................<summand>
......................<multiplier>
........................<unsigned-integer>
..........................1  502
......................<multipliers-list>
........................<empty>
................)  41
..............<multipliers-list>
................<empty>
............<summands-list>
..............-  45
..............<summand>
................<multiplier>
..................<unsigned-integer>
....................3  503
................<multipliers-list>
..................<empty>
..........;  59
........<statement>
..........<variable_identifier>
............<identifier>
..............VARIABLE2  1003
..........:=  301
```

```
..........<expression>
............<summand>
..............<multiplier>
................<unsigned-integer>
..................2  504
..............<multipliers-list>
................*  42
................<multiplier>
..................<unsigned-integer>
....................2  504
................<multipliers-list>
................../  47
..................<multiplier>
....................<unsigned-integer>
......................4  505
............<summands-list>
..............+  43
..............<summand>
................<multiplier>
..................<unsigned-integer>
....................3  503
................<multipliers-list>
..................<empty>
..........;  59
........<statement>
..........<variable_identifier>
```

```
............<identifier>
..............VARIABLE3  1004
..........:=  301
..........<expression>
............<summand>
..............<multiplier>
................<variable_identifier>
..................<identifier>
....................VARIABLE1  1002
..............<multipliers-list>
................<empty>
............<summands-list>
..............+  43
..............<summand>
................<multiplier>
..................<variable_identifier>
....................<identifier>
......................VARIABLE2  1003
................<multipliers-list>
..................<empty>
..........;  59
......END  403
.....  401
```