

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування і спеціалізованих комп’ютерних систем

**Розрахунково-графічна робота**

з дисципліни

**«Основи проектування трансляторів»**

Тема: ”Розробка синтаксичного аналізатора”

Виконав: студент ІII курсу

ФПМ групи КВ-93

Мазенко М. О.

Перевірив:

Київ – 2021

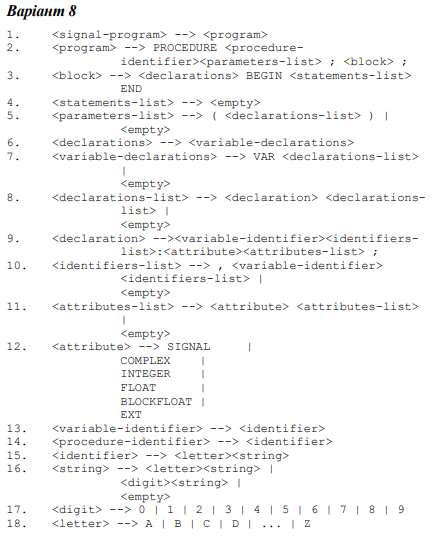
**Мета лабораторної роботи**

Метою розрахунково-графічної роботи «Розробка синтаксисного аналізатора» є засвоєння теоретичного матеріалу та набуття практичного досвіду і практичних навичок розробки синтаксичних аналізаторів (парсерів).

**Постановка задачі**

1. Розробити програму синтаксичного аналізатора (СА) для підмножини мови програмування SIGNAL згідно граматики за варіантом.
2. Програма має забезпечувати наступне:

* читання рядка лексем та таблиць, згенерованих лексичним аналізатором, який було розроблено в лабораторній роботі «Розробка лексичного аналізатора»;
* синтаксичний аналіз (розбір) програми, поданої рядком лексем (алгоритм синтаксичного аналізатора вибирається за варіантом);
* побудову дерева розбору;
* формування таблиць ідентифікаторів та різних констант з повною інформацією, необхідною для генерування коду;
* формування лістингу вхідної програми з повідомленнями про лексичні та синтаксичні помилки



Алгоритм за варіантом: Низхідний розбір за алгоритмом рекурсивного спуску.

**Тести програми**

|  |  |
| --- | --- |
| Максимальний True тест | |
| PROCEDURE TrueTest1  (newVar, ad, ac : FLOAT INTEGER EXT; D4C : SIGNAL;);  VAR adc: FLOAT;  BEGIN  END; | |
| Вхідний рядок лексем | Побудоване дерево розбору |
|  |  |

|  |  |
| --- | --- |
| False тест 1 | |
| PROCEDURE 2TrueTest1  (newVar, ad, ac : FLOAT INTEGER EXT; D4C : SIGNAL;);  VAR adc: FLOAT;  BEGIN  END; | |
| Вхідний рядок лексем | Побудоване дерево розбору |
|  |  |

|  |  |
| --- | --- |
| False тест 2 | |
| PROCEDURE TrueTest1  (newVar, ad, ac FLOAT INTEGER EXT; D4C : SIGNAL;);  VAR adc: FLOAT;  BEGIN  END; | |
| Вхідний рядок лексем | Побудоване дерево розбору |
|  |  |

|  |  |
| --- | --- |
| False тест 3 | |
| PROCEDURE TrueTest1  (newVar, ad, ac : FLOAT INTEGER EXT; D4C : SIGNAL;);  VAR adc: FLOAT;  BuGIN  END; | |
| Вхідний рядок лексем | Побудоване дерево розбору |
|  |  |

|  |  |
| --- | --- |
| False тест 4 | |
| PROCEDURE TrueTest1  (newVar, ad, ac FLOAT INTEGER EXT; D4C : SIGNAL;);  VAR adc: FLdOAT;  BEGIN  END; | |
| Вхідний рядок лексем | Побудоване дерево розбору |
|  |  |

|  |  |
| --- | --- |
| False тест 5 | |
| PROCEDURE TrueTest1  (newVar, ad, ac FLOAT INTEGER EXT; D4C : SIGNAL;;  VAR adc: FLOAT;  BuGIN  END; | |
| Вхідний рядок лексем | Побудоване дерево розбору |
|  |  |

**Лістинг коду програми**

**main.cpp**

#include "view.h"

void LexerTrueTest() {

translator translator\_object = translator();

view::PrintInformTable(translator\_object.GetKeywords());

view::PrintInformTable(translator\_object.GetDelimiters());

view::PrintAttributeTable(translator\_object.GetAttributes());

view::ShowFile("txt/lexer\_true\_test.txt");

translator\_object.Lexer("txt/lexer\_true\_test.txt");

view::PrintLexemTable(translator\_object.GetLexemTable());

//view::PrintInformTable(translator\_object.GetIdentifiers());

//view::PrintInformTable(translator\_object.GetConstants());

//view::PrintInformTable(translator\_object.GetDelimiters());

//view::PrintInformTable(translator\_object.GetKeywords());

view::DrawTree(translator\_object.Parser(), 0);

}

void LexerFalseTest() {

translator translator\_object = translator();

view::ShowFile("txt/lexer\_false\_test.txt");

translator\_object.Lexer("txt/lexer\_false\_test.txt");

view::PrintLexemTable(translator\_object.GetLexemTable());

view::PrintInformTable(translator\_object.GetIdentifiers());

view::PrintInformTable(translator\_object.GetConstants());

view::PrintInformTable(translator\_object.GetDelimiters());

view::PrintInformTable(translator\_object.GetKeywords());

}

int main() {

LexerTrueTest();

//LexerFalseTest();

return 0;

}

**fileReader.h**

#pragma once

#include <fstream>

#include <iostream>

using namespace std;

class fileReader

{

private:

pair<int, int> coordinates;

fstream file;

public:

fileReader() : fileReader("txt/file\_1.txt") {};

fileReader(string path) {

coordinates.first = 0;

coordinates.second = 1;

file = fstream(path, ios\_base::in);

}

~fileReader() {

file.close();

}

bool FileIsEof() {

return file.eof();

}

bool FileIsSuitable() {

return file.is\_open() && !FileIsEof() ? true : false;

}

char ReadSymbol();

pair<int, int> GetCoordinates() { return coordinates; }

};

**fileReader.cpp**

#include "fileReader.h"

char fileReader::ReadSymbol() {

char symbol;

file.get(symbol);

if ((int)symbol == 10) {

coordinates.second++;

coordinates.first = 0;

}

else

coordinates.first++;

return symbol;

}

**translator.h**

#pragma once

#include <string>

#include <vector>

#include <list>

#include "fileReader.h"

#define TableSetBasicInformation(name, position) SetBasicInformation(#name, &name, position)

struct lexem {

pair<int, int> coordinates;

int code;

string word;

};

struct inform\_table {

string word;

int code;

};

struct tree\_node {

string head;

vector<tree\_node> rest;

};

class translator

{

private:

vector<inform\_table> keyword\_table;

vector<inform\_table> identifier\_table;

vector<inform\_table> constant\_table;

vector<inform\_table> delimiter\_table;

vector<lexem> lexem\_table;

vector<inform\_table> basic\_attribures;

vector<int> parser\_identifiers;

static vector<int> attributes;

static vector<int> SetAttributes();

static bool WhitespaceChecker(int);

static bool CommentChecker(int);

static void GetError(string, pair<int, int>, string);

int parser\_position = 0;

bool TreeAddIfNotNull(vector<tree\_node>\*, tree\_node);

bool TreeAddIfSuitable(vector<tree\_node>\*, int);

pair<int, int> TreeGetPosition();

bool TreeFindInTable(vector<tree\_node>\*, vector<inform\_table>);

bool FindAttribute(int, vector<int>);

void SetBasicInformation(const string, vector<inform\_table>\*, int);

char ReadWithAdding(char, vector<int>, list<vector<inform\_table>\*>, fileReader\*);

char ReadWithoutAdding(char, bool (\*)(int), fileReader\*);

char ReadComment(char, fileReader\*);

void FindInTable(string, list<vector<inform\_table>\*>, pair<int, int>);

tree\_node SignalProgram();

tree\_node Program();

tree\_node Block();

tree\_node ParametersList();

tree\_node Declarations();

tree\_node VariableDeclarations();

tree\_node DeclarationsList();

tree\_node Declaration();

tree\_node IdentifiersList();

tree\_node AttributesList();

tree\_node Attribute();

tree\_node VariableIdentifier();

tree\_node ProcedureIdentifier();

tree\_node Identifier();

public:

translator();

void Lexer(string path);

tree\_node Parser();

vector<inform\_table> GetKeywords() { return keyword\_table; };

vector<inform\_table> GetIdentifiers() { return identifier\_table; };

vector<inform\_table> GetConstants() { return constant\_table; };

vector<inform\_table> GetDelimiters() { return delimiter\_table; };

vector<lexem> GetLexemTable() { return lexem\_table; };

vector<int> GetAttributes() { return attributes; };

};

**translator.cpp**

#include "translator.h"

vector<int> translator::attributes = translator::SetAttributes();

void translator::GetError(string source, pair<int, int> position, string message) {

cout << "[" << position.first << ", " << position.second << "] " << source << ": " << message << endl;

}

vector<int> translator::SetAttributes() {

vector<int> local\_list;

local\_list.assign(127, 6);

string current\_field;

int line\_iteration = 0, current\_number = 0;

fstream file("txt/attributes.txt", ios\_base::in);

if (file.is\_open()) {

while (file >> current\_field) {

if (current\_field == ".")

line\_iteration++;

else if (current\_field == "-") {

file >> current\_field;

for (int iteration = current\_number + 1; iteration <= stoi(current\_field); iteration++)

local\_list[iteration] = line\_iteration;

}

else {

current\_number = stoi(current\_field);

local\_list[current\_number] = line\_iteration;

}

}

}

return local\_list;

}

translator::translator() {

TableSetBasicInformation(keyword\_table, 300);

TableSetBasicInformation(identifier\_table, 400);

TableSetBasicInformation(constant\_table, 500);

TableSetBasicInformation(delimiter\_table, 55);

basic\_attribures = identifier\_table;

}

void translator::SetBasicInformation(const string name, vector<inform\_table>\* lst, int position) {

vector<inform\_table> inform\_table\_list;

inform\_table local\_inform\_table;

fstream file("txt/" + name + ".txt", ios\_base::in);

if (file.is\_open()) {

while (getline(file, local\_inform\_table.word)) {

local\_inform\_table.code = ++position;

inform\_table\_list.push\_back(local\_inform\_table);

}

\*lst = inform\_table\_list;

}

}

void translator::Lexer(string path) {

fileReader file\_object(path);

char current\_symbol;

bool suppress\_output = false;

vector<int> acceptable\_attributes;

list<vector<inform\_table>\*> acceptable\_tables;

int current\_value, current\_attribute;

if (file\_object.FileIsSuitable())

current\_symbol = file\_object.ReadSymbol();

else {

GetError("Lexer", file\_object.GetCoordinates(), "invalid file name or content");

return;

}

do {

current\_value = (int)current\_symbol;

current\_attribute = attributes[current\_value];

switch (current\_attribute) {

case 0:

current\_symbol = ReadWithoutAdding(file\_object.ReadSymbol(), &WhitespaceChecker, &file\_object);

break;

case 1:

current\_symbol = ReadWithAdding(current\_symbol, { 1 }, { &constant\_table }, &file\_object);

break;

case 2:

current\_symbol = ReadWithAdding(current\_symbol, { 1, 2 }, { &keyword\_table, &constant\_table, &identifier\_table }, &file\_object);

break;

case 3:

FindInTable(string(1, current\_symbol), { &delimiter\_table }, file\_object.GetCoordinates());

current\_symbol = file\_object.ReadSymbol();

break;

case 4:

current\_symbol = ReadComment(current\_symbol, &file\_object);

break;

default:

GetError("Lexer", file\_object.GetCoordinates(), "wrong symbol \"" + string(1, current\_symbol) + "\"");

current\_symbol = file\_object.ReadSymbol();

break;

}

} while (!file\_object.FileIsEof());

}

char translator::ReadComment(char start\_symbol, fileReader\* file\_object) {

char current\_symbol;

pair<int, int> start\_coordinates = file\_object->GetCoordinates();

current\_symbol = file\_object->ReadSymbol();

if (attributes[(int)current\_symbol] == 5) {

do {

ReadWithoutAdding(current\_symbol, &CommentChecker, file\_object);

current\_symbol = file\_object->ReadSymbol();

if (file\_object->FileIsEof()) {

GetError("Lexer", file\_object->GetCoordinates(), "end of comment is missing");

return 0;

}

} while (current\_symbol != ')');

current\_symbol = file\_object->ReadSymbol();

}

else {

FindInTable(string(1, start\_symbol), { &delimiter\_table }, start\_coordinates);

}

return current\_symbol;

}

char translator::ReadWithoutAdding(char current\_symbol, bool (\*Condition)(int), fileReader\* file\_object) {

while (!file\_object->FileIsEof() && Condition(attributes[(int)current\_symbol])) {

current\_symbol = file\_object->ReadSymbol();

}

return current\_symbol;

}

void translator::FindInTable(string lexem\_word, list<vector<inform\_table>\*> acceptable\_tables, pair<int, int> coordinates) {

for (vector<inform\_table>\* current\_table\_type : acceptable\_tables) {

for (inform\_table current\_inform\_table : \*current\_table\_type) {

if (current\_inform\_table.word.compare(lexem\_word) == 0) {

lexem new\_lexem = {

coordinates,

current\_inform\_table.code,

lexem\_word };

lexem\_table.push\_back(new\_lexem);

return;

}

}

}

int new\_code = acceptable\_tables.back()->back().code + 1;

acceptable\_tables.back()->push\_back(inform\_table{ lexem\_word, new\_code });

lexem\_table.push\_back(lexem{ coordinates, new\_code, lexem\_word });

}

char translator::ReadWithAdding(char symbol, vector<int> acceptable\_attributes, list<vector<inform\_table>\*> acceptable\_tables, fileReader\* file\_object) {

string current\_lexem;

pair<int, int> start\_coordinates = file\_object->GetCoordinates();

char current\_symbol = symbol;

do {

current\_lexem += current\_symbol;

current\_symbol = file\_object->ReadSymbol();

} while (FindAttribute(attributes[(int)current\_symbol], acceptable\_attributes));

FindInTable(current\_lexem, acceptable\_tables, start\_coordinates);

return current\_symbol;

}

bool translator::WhitespaceChecker(int attribute) {

return attribute == 0 ? true : false;

}

bool translator::CommentChecker(int attribute) {

return attribute != 5 ? true : false;

}

bool translator::FindAttribute(int attribute, vector<int> attribute\_list) {

for (int current\_attribute : attribute\_list) {

if (attribute == current\_attribute) {

return true;

}

}

return false;

}

bool translator::TreeFindInTable(vector<tree\_node>\* node, vector<inform\_table> table) {

for (auto attribute : table)

if (TreeAddIfSuitable(node, attribute.code)) {

return true;

}

return false;

}

bool translator::TreeAddIfNotNull(vector<tree\_node>\* tree, tree\_node current\_node) {

if (!current\_node.rest.empty()) {

tree->push\_back(current\_node);

return true;

}

return false;

}

bool translator::TreeAddIfSuitable(vector<tree\_node>\* tree, int code\_id) {

if (lexem\_table.size() > parser\_position && lexem\_table[parser\_position].code == code\_id) {

tree->push\_back({ lexem\_table[parser\_position].word });

parser\_position++;

return true;

}

return false;

}

pair<int, int> translator::TreeGetPosition() {

return lexem\_table.size() > parser\_position ?

lexem\_table[parser\_position].coordinates :

lexem\_table[lexem\_table.size() - 1].coordinates;

}

tree\_node translator::Parser() {

return SignalProgram();

}

tree\_node translator::SignalProgram() {

tree\_node node;

parser\_position = 0;

node.head = "<signal-program>";

TreeAddIfNotNull(&node.rest, Program());

return node;

}

tree\_node translator::Program() {

tree\_node node;

node.head = "<program>";

if (!TreeAddIfSuitable(&node.rest, 301))

return node;

if (!TreeAddIfNotNull(&node.rest, ProcedureIdentifier())) {

GetError("Parser", TreeGetPosition(), "identifier expected");

return node;

}

TreeAddIfNotNull(&node.rest, ParametersList());

if (!TreeAddIfSuitable(&node.rest, 56)) {

GetError("Parser", TreeGetPosition(), "\";\" expected");

return node;

}

if(!TreeAddIfNotNull(&node.rest, Block())) {

GetError("Parser", TreeGetPosition(), "<block> expected");

return node;

}

if (!TreeAddIfSuitable(&node.rest, 56)) {

GetError("Parser", TreeGetPosition(), "\";\" expected");

return node;

}

return node;

}

tree\_node translator::Block() {

tree\_node node;

node.head = "<block>";

TreeAddIfNotNull(&node.rest, Declarations());

if (!TreeAddIfSuitable(&node.rest, 302)) {

GetError("Parser", TreeGetPosition(), "\"BEGIN\" expected");

return node;

}

if (!TreeAddIfSuitable(&node.rest, 303)) {

GetError("Parser", TreeGetPosition(), "\"END\" expected");

return node;

}

return node;

}

tree\_node translator::Declarations() {

tree\_node node;

node.head = "<declarations>";

TreeAddIfNotNull(&node.rest, VariableDeclarations());

return node;

}

tree\_node translator::VariableDeclarations() {

tree\_node node;

node.head = "<variable-declarations>";

if (!TreeAddIfSuitable(&node.rest, 304)) {

return node;

}

if(!TreeAddIfNotNull(&node.rest, DeclarationsList())) {

GetError("Parser", TreeGetPosition(), "identifier expected");

return node;

}

return node;

}

tree\_node translator::ProcedureIdentifier() {

tree\_node node;

node.head = "<procedure-identifier>";

TreeAddIfNotNull(&node.rest, Identifier());

return node;

}

tree\_node translator::Identifier() {

tree\_node node;

node.head = "<identifier>";

if (lexem\_table[parser\_position].code > 499) {

GetError("Parser", TreeGetPosition(), "wrong identifier");

return tree\_node{};

}

if (TreeFindInTable(&node.rest, identifier\_table) &&

(lexem\_table.size() > parser\_position - 1)) {

for (auto identifier : parser\_identifiers) {

if (identifier == lexem\_table[parser\_position - 1].code) {

GetError("Parser", TreeGetPosition(), "identifier already declared");

return node;

}

}

parser\_identifiers.push\_back(lexem\_table[parser\_position - 1].code);

}

return node;

}

tree\_node translator::ParametersList() {

tree\_node node;

node.head = "<parameters-list>";

if (!TreeAddIfSuitable(&node.rest, 57))

return node;

TreeAddIfNotNull(&node.rest, DeclarationsList());

if (!TreeAddIfSuitable(&node.rest, 58)) {

GetError("Parser", TreeGetPosition(), "\")\" expected");

return node;

}

return node;

}

tree\_node translator::DeclarationsList() {

tree\_node node;

node.head = "<declarations-list>";

if (TreeAddIfNotNull(&node.rest, Declaration()))

TreeAddIfNotNull(&node.rest, DeclarationsList());

return node;

}

tree\_node translator::Declaration() {

tree\_node node;

if (!TreeAddIfNotNull(&node.rest, VariableIdentifier()))

return tree\_node{};

TreeAddIfNotNull(&node.rest, IdentifiersList());

if (!TreeAddIfSuitable(&node.rest, 60)) {

GetError("Parser", TreeGetPosition(), "\":\" expected");

return tree\_node{};

}

if(!TreeAddIfNotNull(&node.rest, Attribute())) {

GetError("Parser", TreeGetPosition(), "attribute expected");

return tree\_node{};

}

TreeAddIfNotNull(&node.rest, AttributesList());

if (!TreeAddIfSuitable(&node.rest, 56)) {

GetError("Parser", TreeGetPosition(), "\";\" expected");

return tree\_node{};

}

node.head = "<declaration>";

return node;

}

tree\_node translator::VariableIdentifier() {

tree\_node node;

node.head = "<variable-identifier>";

TreeAddIfNotNull(&node.rest, Identifier());

return node;

}

tree\_node translator::IdentifiersList() {

tree\_node node;

node.head = "<identifiers-list>";

if (!TreeAddIfSuitable(&node.rest, 59))

return node;

if(TreeAddIfNotNull(&node.rest, VariableIdentifier()))

TreeAddIfNotNull(&node.rest, IdentifiersList());

return node;

}

tree\_node translator::Attribute() {

tree\_node node;

node.head = "<attribute>";

TreeFindInTable(&node.rest, basic\_attribures);

return node;

}

tree\_node translator::AttributesList() {

tree\_node node;

node.head = "<attributes-list>";

if (TreeAddIfNotNull(&node.rest, Attribute()))

TreeAddIfNotNull(&node.rest, AttributesList());

return node;

}

**view.h**

#pragma once

#include "translator.h"

#include <iomanip>

static class view

{

public:

static void PrintInformTable(vector<inform\_table>);

static void PrintAttributeTable(vector<int>);

static void PrintLexemTable(vector<lexem>);

static void ShowFile(string);

static void DrawTree(tree\_node, int);

};

**view.cpp**

#include "view.h"

#include <time.h>

void view::DrawTree(tree\_node node, int depth) {

for (int iter = 0; iter < depth; iter++)

cout << " ";

cout << node.head << endl;

for (auto current\_node : node.rest)

DrawTree(current\_node, depth + 1);

}

void view::ShowFile(string path) {

fileReader file\_object(path);

char symbol = 10;

cout << "Input file:\n";

cout << "\_\_\_\_\_\_\_\_\_\_\_";

while (!file\_object.FileIsEof()) {

cout << symbol;

symbol = file\_object.ReadSymbol();

}

cout << "\n\_\_\_\_\_\_\_\_\_\_\_\n";

}

void view::PrintLexemTable(vector<lexem> lexem\_table) {

cout << "+=========================================+\n";

cout << "| Column| Row| Code| Lexem|\n";

cout << "|=========================================|\n";

for (lexem current\_lexem : lexem\_table) {

cout

<< "|"

<< setw(7) << current\_lexem.coordinates.first << "|"

<< setw(7) << current\_lexem.coordinates.second << "|"

<< setw(7) << current\_lexem.code << "|"

<< setw(17) << current\_lexem.word << "|\n";

}

cout << "+=========================================+\n\n";

}

void view::PrintInformTable(vector<inform\_table> lst) {

cout << "+==============================+\n";

cout << "|" << setw(20) << "Name|" << setw(10) << "Code" << "|\n";

cout << "|==============================|\n";

for (inform\_table local\_inform\_table : lst) {

cout << "|" << setw(20) << local\_inform\_table.word + "|" << setw(10) << local\_inform\_table.code << "|\n";

}

cout << "|==============================|\n\n";

}

void view::PrintAttributeTable(vector<int> attributes) {

cout << "+=======================================================================+\n";

for (int iteration = 32; iteration < attributes.size(); iteration++) {

cout << "| " << attributes[iteration] << " " << setw(5) << (char)iteration;

if (iteration % 8 == 7) {

cout << "|" << endl;

}

}

cout << "|\t|\n";

cout << "+=======================================================================+\n\n";

}