Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Univerzitetni študijski program: Računalništvo in informacijske tehnologije

PREVAJANJE PROGRAMSKIH JEZIKOV: ZBIRKA NALOG IN REŠITEV

doc. dr. Tomaž Kos izr. prof. dr. Tomaž Kosar

Maribor, 2023

Naslov: Prevajanje programskih jezikov: zbirka nalog in rešitev

Avtorji: doc. dr. Tomaž Kos
mailto://tomaz.kos@guest.um.si
izr. prof. dr. Tomaž Kosar
mailto://tomaz.kosar@um.si

Ime predmeta: Prevajanje programskih jezikov

Leto izdaje: 2023

izdaja: Prva izdaja

Izdajateljica: Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko (UM FERI)
Koroška cesta 46, 2000, Maribor, Slovenija
https://feri.um.si
mailto://feri@um.si

Recenzent: izr. prof. dr. Matej Črepinšek

Vrsta publikacije: e-publikacija
zbirka vaj in nalog

Dostopno na: https://dk.um.si/Dokument.php?id=175102&lang=slv

Posodobitev: Dokument nazadnje posodobljen 14.6.2023

Zgodovina dokumenta: Prva verzija dokumenta: 9.12.2022

KAZALO

POJMOVNO KAZALO

Povzetek

V zbranem učnem gradivu računalniških vaj pri predmetu Prevajanje programskih jezikov obravnavamo uvod v prevajalnike. Bralca seznanimo z osnovno zgradbo prevajalnikov. Obravnavamo leksikalno analizo, sintaktično analizo in semantično analizo. V gradivu zajamemo tudi generatorje prevajalnikov in nekatere praktične primere uporabe v programskem inženirstvu kot je razpoznavanje dokumentov XML in uporaba regularnih izrazov temelječih na gramatikah.

Abstract

In the collected teaching material of computer exercises on the subject of Compiling programming languages, we discuss the introduction to compilers. We acquaint the reader with the basic structure of compilers. We cover lexical analysis, syntactic analysis, and semantic analysis. The teaching material also covers compiler generators and some practical examples of use in software engineering, such as the parsing XML documents and the use of regular expressions.

Predgovor

Zbirka vaj je namenjena študentom Računalništva in informatike predmeta Prevajanje programskih jezikov.

Naloge pokrivajo snov, ki jo obravnavamo na vajah. Naloge so pripravljene tako, da jih študent lahko reši postopoma. Nekatere naloge se navezujejo ena na drugo kar pomeni, da reševanje mora potekati od prve do zadnje naloge.

Naloge so namenjene utrjevanju snovi, ki jo obravnavamo pri predmetu in služi kot priprava na zagovor.

Želim vam uspešno reševanje nalog!

UVOD

1.1. Osnovni cilji predmeta

Cilj tega predmeta (Vilfan, 1991) je razumeti osnovne pojme iz teorije avtomatov (J. E. Hopcroft, 2001) in programskih jezikov (A. V. Aho M. S., 2007) in njihovo praktično uporabo pri prevajanju programskih jezikov (A. W. Appel, 2002). S tem študentje pridobijo znanje, ki ga je mogoče uporabiti pri praktičnih in industrijskih primerih.

1.2. Vsebina predmeta

- Spoznavanje z regularnimi izrazi in uporaba izrazov znotraj jezika C++.
- Implementacija leksikalnega analizatorja za aritmetične izraze v programskem jeziku C++.
- Implementacija sintaktičnega analizatorja za aritmetične izraze v programskem jeziku C++.
- Implementacija semantičnega analizatorja za aritmetične izraze v programskem jeziku C++.
- Validacija, razpoznavanje in povpraševanje dokumentov XML (dokument XML, sheme XML, XML DOM, XPath).
- Generiranje jezika za aritmetične izraze znotraj orodja ANTLR.

REGULARNI IZRAZI

1.3. Naloga 1

Pojdite na spletno stran RegexOne (https://regexone.com/) in rešite naslednje primere 1-8.

Primer 1: opis elektronske pošte

Primer 2: opis telefonskih števil v obliki XXX-XXX-XXX in XX-XXX-XXX

Primer 3: opis decimalnih števil

Primer 4: opis datumov v obliki DD/MM/YYYY

Primer 5: opis vključenih *.h knjižnic

Primer 6: opis deklaracij C++ struktur

Primer 7: opis deklaracij statičnih metod razreda

1.4. Naloga 2

Napišite program C++, ki prebere tekstovno datoteko (xml_serializer.h) z besedilom (primer na spletni strani http://www.cplusplus.com/doc/tutorial/files/).

Preštejte in izpišite vse znake v datoteki.

S pomočjo C++ knjižnice regex (https://en.cppreference.com/w/cpp/regex) preštejte število besed v datoteki.

Izpišite vse deklaracije statičnih metod razreda (uporabite std::regex_search).

xml_serializer.h

```
/**
 * @file    xml_serializer.h
 * @author  Miha Novak
 * @email   miha.novak@um.si
```

```
 * @tell    041-123-456
 * @date    2/11/2018
 * @version 1.0
 *
 * @brief XmlSerializer class implements functions for parsing xml files using
libxml2 library and for creating new xml files.
 *
 */

#pragma once
#include "dx_node.h"
#include "xml_not.h"

BEGIN_NAMESPACE_SERIALIZATION

struct DxDocument;
using DxDocumentPtr = std::shared_ptr<DxDocument>;

/**
 * @brief DxDocument struct that contains shared pointer to a root node.
 */
struct DxDocument
{
    DxNodePtr startNode = nullptr;
};

/**
 * @brief DxDocumentWriteParameters struct that contains information if nodes
should be indented.
 */
struct DxDocumentWriteParameters
{
    bool indent = true;
};

struct SaxContext;

/**
* @brief XmlSerializer class implements functions for parsing xml files using
libxml2 library and for creating new xml files.
*/
class XmlSerializer
{
public:

    /**
     * @brief Function creates new DxDocument.
     * @return Returns a shared pointer to newly created DxDocument
     */
    static DxDocumentPtr dxDocumentCreate();

    /**
     * @brief Function recursively clears all child nodes and sets document
pointer to nullptr.
     * @param document Shared pointer to a document
     */
    static void dxDocumentDestroy(DxDocumentPtr& document);
```

```
    /**
     * @brief Function returns a shared pointer to a root node (start node).
     * @param document Shared pointer to a document
     * @return A shared pointer to a root node
     */
    static DxNodePtr dxDocumentGetStartNode(const DxDocumentPtr& document);

    /**
     * @brief Function sets a root node (start node) to a given document.
     * @param document Shared pointer to a document
     * @param startNode Shared pointer to a node
     */
    static void dxDocumentSetStartNode(const DxDocumentPtr& document, const
DxNodePtr& startNode);

    /**
     * @brief Function writes a document to a file stream.
     * @param document Shared pointer to a document
     * @param fileName Xml file name
     * @param writeParams A struct that contains information if document nodes
should be indented
     * @param node Shared pointer to a node. In case pointer is a nullptr than
documents root node is assigned to it
     */
    static void dxDocumentWriteToFile(const DxDocumentPtr& document, const
std::string& fileName, const DxDocumentWriteParameters writeParams, DxNodePtr&
node);

    /**
     * @brief Function reads and parses an xml file using libxml2 library and
returns parsed document.
     * @param fileName Xml file name
     * @return A shared pointer to a newly created document. If xml file is not
valid it returns a nullptr
     */
    static DxDocumentPtr dxDocumentParse(const std::string& fileName);

protected:
    void outputValue(std::string value, bool escapeQuote, std::ofstream&
stream);
    virtual void writeNode(const DxNodePtr& node, int indent, std::ofstream&
stream, const DxDocumentWriteParameters writeParams);
    void freeSaxContext(SaxContext* saxContext);
};
END_NAMESPACE_SERIALIZATION
```

1.5. Naloga 3

Nalogo 2.2 razširite tako, da uporabnik v drugo datoteko napiše besede, ki jih išče v vhodni datoteki.

V datoteki xml_serializer.h zamenjajte vse "@brief" besede z besedo "@opis" (uporabite std::regex_replace).

LEKSIKALNI ANALIZATOR

1.6. Naloga 1

Pregledovalnik (angl. scanner) predstavlja vmesnik med izvornim programom in razpoznavalnikom. Njegova naloga je, da vrne razpoznavalniku terminalne simbole. Njegova naloga je tudi, da izloči prazna mesta, preskoke v novo vrstico ter komentarje.

Pregledovalnik obravnava vhodno datoteko kot niz znakov, ki jih nato sestavlja v terminalne simbole. Za opis terminalnih simbolov uporabljamo končne avtomate, te pa najlažje implementiramo s tabelo.

Osnovni leksikalni simboli:

```
float [0-9]+(.[0-9]+)?
operator + | * | - | / | ^ | %
separator ( | )
variable [a-zA-Z]+[0-9]*
```

Leksikalni analizator implementirajte v programskem jeziku C++!

1.7. Naloga 2

Jezik aritmetičnih izrazov iz naloge 3.1 razširite tako, da bo prepoznal tudi naslednje primere programov:

```
stevilo := 10;
fakulteta:=1;
for i := 1 to stevilo
  fakulteta:= fakulteta * i;
CONSOLE fakulteta;
```

SINTAKTIČNI ANALIZATOR

1.8. Naloga 1

Zapišite rekurzivni navzdolnji razpoznavalnik oz. sintaktični analizator za programski jezik iz naloge 3.1 z naslednjo gramatiko:

```
E ::= T EE;
EE::= + T EE | - T EE | epsilon;
T ::= F TT;
TT::= * F TT | / F TT | ^ F TT | % F TT | epsilon;
F ::= ( E ) | #float | #variable;
```

1.9. Naloga 2

Zapišite sintaktični analizator za jezik podan pri nalogi 3.2.

SEMANTIČNI ANALIZATOR

1.10. Naloga 1

Napišite semantični analizator oz. evaluator izrazov s pomočjo atributne gramatike. Naloga evaluatorja je, da ovrednoti podan aritmetični izraz. Razpoznavalnik iz naloge 4.1. nadgradite s semantičnimi funkcijami tako, da je rezultat ovrednotenja številska vrednost vhodnega izraza.

1.11. Naloga 2

Napišite semantični analizator za razpoznavalnik iz 4.2. naloge tako, da bo rezultat za naslednji primer programa:

```
stevilo := 10;
fakulteta:=1;
for i := 1 to stevilo
     fakulteta:= fakulteta * i;
CONSOLE fakulteta;
```

enak:

```
3628800
```

RAZPOZNAVANJE IN POVPRAŠEVANJE XML DOKUMENTOV

1.12. Naloga 1

Za XML dokument o izdelkih spletne trgovine zapišite strukturo v XML Shemah. Sintaksa XML dokumenta vsebuje naslednje elemente:

```
ID,
ime oz. naziv,
osvezitev,
url,
opis,
slike,
cena, in
kategorija ID.
```

Poleg teh elementov dodajte še vsaj 3 svoje elemente.

Pri definiranju zgoraj naštetih elementov v shemah XML, uporabljajte omejitve (npr. ID izdelka : 1 - 1000) in vzorce (npr. v elementu osvezitev). Podobno naredite tudi za ostale kompleksne/enostavne tipe.

Nekaj koristnih povezav:

- XML jezik (https://en.wikipedia.org/wiki/XML),
- XML tutorial (https://www.w3schools.com/xml/default.asp),
- Online validacija XML dokumentov (http://www.utilities-online.info/xsdvalidation/#.XJvT1phKiUk).

1.13. Naloga 2

V programskem jeziku C# ali Javi uporabite obstoječ XML razpoznavalnik iz knjižnic (uporabite zglede iz spletne strani spodaj):

- Nad XML dokumentom "trgovina.xml" (Glej prejšnjo nalogo) pokažite kako preverimo ali je dokument XML v skladu s strukturo, ki je zapisana v XML shemi.
- Pokažite dodajanje in odstranjevanje elementov iz drevesne strukture.
- Spremenjeno drevesno strukturo dokumenta XML zapišite v datoteko.
- S pomočjo povpraševalnega jezika XPath napišite vsaj tri lastna povpraševanja.

Nekaj koristnih povezav:

- Validacija dokumenta (https://docs.microsoft.com/en-us/dotnet/api/system.xml.xmldocument.validate?view=netframework-4.7.2)
- Dodajanje elementa (https://docs.microsoft.com/en-us/dotnet/api/system.xml.xmldocument.createnode?view=netframework-4.7.2)
- XPath povpraševanje (https://docs.microsoft.com/en-us/dotnet/standard/data/xml/select-nodes-using-xpath-navigation, http://www.java2s.com/Code/CSharp/XML/UseSelectNodestoquerynodesbyXPath.htm)

GENERATORJI PREVAJALNIKOV

1.14. Naloga 1

Z orodjem ANTLR implementirajte evaluator jezika aritmetičnih izrazov za gramatiko, ki je podana spodaj.

Expr.g4

```
grammar Expr;

prog : star+ ;
star : expr NEWLINE                 # printExpr
     | VARIABLE '=' expr NEWLINE    # assign
       | NEWLINE                      # blank
       ;
expr : expr op=('*'|'/'|'^') expr   # mulDivPow
       | expr op=('+'|'-') expr       # addSub
       | FLOAT                        # number
       | '-' FLOAT                    # minNumber
       | VARIABLE                     # variable
       | '(' expr ')'                 # parens
     ;

FLOAT : [0-9]+('.'[0-9]+)?;
VARIABLE : [a-zA-Z]+[0-9]* ;
MUL : '*' ;
DIV : '/' ;
POW : '^' ;
ADD : '+' ;
SUB : '-' ;

NEWLINE : '\r'? '\n' ;
IGNORE : [ \t]+ -> skip ; // skip spaces, tabs
```

REŠITVE

V nadaljevanju tega poglavja podajamo rešitve nalog iz prejšnjih poglavij.

1.15. Regularni izrazi - Naloga 1

```
Primer 1: Izraz: ([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})

Primer 2: Izraz: [0-9]{2,3}-[0-9]{3}-[0-9]{3}

Primer 3: Izraz: [0-9]+\.[0-9]?

Primer 4: Izraz: [0-9]{1,2}\/[0-9]{1,2}\/[0-9]{4}

Primer 5: Izraz: #include\s+"[a-zA-z]+.h"

Primer 6: Izraz: struct\s+[a-zA-Z0-9_]+\s*\{[\s\w=;]*};
```

```
Primer 7: Izraz: static\s+[\w\s(\&):,]+;
```

1.16. Regularni izrazi - Naloga 2

naloga2.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <regex>

using namespace std;

int main()
{
	ifstream myfile("c:\\xml_serializer.h");
	if (myfile.is_open())
	{
		std::string inputString;
		char c;
		cout << "Print characters:\n";
		while (myfile.get(c))
		{
			cout << c;
			inputString = inputString + c;
		}
		myfile.close();

		cout << "Number of characters: " << inputString.size() << "\n\n";

		std::regex word_regex("(\\S+)");
		auto   words_begin   =   std::sregex_iterator(inputString.begin(),
inputString.end(), word_regex);
		auto words_end = std::sregex_iterator();
		cout << "Number of words: " << std::distance(words_begin, words_end)
<< '\n';

		cout << "Static method declarations:\n";
		const std::regex staticMethodRegex("static\\s+[\\w\\s(\\&):,]+;");
		std::smatch staticMethod;
		while         (std::regex_search(inputString,         staticMethod,
staticMethodRegex))
		{
			std::cout << staticMethod.str() << '\n';
			inputString = staticMethod.suffix();
		}
	}
	else
		cout << "Unable to open file";

	return 0;
}
```

1.17. Regularni izrazi - Naloga 3

naloga3.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <regex>

using namespace std;

int main()
{
	ifstream myfile("c:\\xml_serializer.h");
	ofstream myOutputfile("c:\\results.txt");
	if (myfile.is_open() && myOutputfile.is_open())
	{
		std::string inputString;
		std::string outputReplaceString;
		char c;
		myOutputfile << "Print characters:\n";
		while (myfile.get(c))
		{
			myOutputfile << c;
			inputString = inputString + c;
		}
		myfile.close();

		outputReplaceString = inputString;

		myOutputfile << "Number of characters: " << inputString.size() <<
"\n\n";

		std::regex word_regex("(\\S+)");
		auto   words_begin   =   std::sregex_iterator(inputString.begin(),
inputString.end(), word_regex);
		auto words_end = std::sregex_iterator();
		myOutputfile << "Number of words: " << std::distance(words_begin,
words_end) << '\n';

		myOutputfile << "Static method declarations:\n";
		const std::regex staticMethodRegex("static\\s+[\\w\\s(\\&):,]+;");
		std::smatch staticMethod;
		while         (std::regex_search(inputString,         staticMethod,
staticMethodRegex))
		{
			myOutputfile << staticMethod.str() << '\n';
			inputString = staticMethod.suffix();
		}
		myOutputfile.close();

		//replace
		std::regex replaceRegex("@brief");
		outputReplaceString    =    std::regex_replace(outputReplaceString,
replaceRegex, "@opis");
		cout << outputReplaceString;
```

```
	}
	else
		cout << "Unable to open file";

	return 0;
}
```

1.18. Leksikalni analizator - Naloga 1

naloga.cpp

```
#include "token.h"
#include "scanner.h"

using namespace std;

int main()
{
	Token token;
	Scanner scanner(std::make_shared<ifstream>("c:\\input1.txt"));

	cout << "Tokens:\n";
	do
	{
		token = scanner.nextToken();
		cout << token << '\n';
	} while (!token.isEof() && (token.getToken() != Scanner::tLexError));
	return 0;
}
```

token.h

```
#pragma once

#include <string>

using namespace std;

//Leksikalni simbol
class Token
{
private:
	string lexem{};//leksikalni element
	int col, row{};
	int token{};//osnovni leksikalni simbol
	bool eof{};
public:
	Token(const string& lexem, int col, int row, int token, bool eof) : lexem{
lexem }, col{ col }, row{ row }, token{ token }, eof{ eof }
	{}
	Token()
	{}

	string getLexem() const
	{
```

```
		return lexem;
	}

	int getCol() const
	{
		return col;
	}

	int getRow() const
	{
		return row;
	}

	int getToken() const
	{
		return token;
	}

	bool isEof() const
	{
		return eof;
	}

	friend ostream& operator<<(ostream& out, const Token& token)
	{
		out << "'" << token.getLexem() << "' " << token.getToken() << " ("
<< token.getRow() <<
			", " << token.getCol() << ") " <<
			(token.isEof() ? "true" : "false");
		return out;
	}
};
```

scanner.h

```
#pragma once

#include "token.h"
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

class Scanner
{
private:
	std::shared_ptr<ifstream> input;
	Token lastToken;//trenutni osnovni leksikalni simbol
	int row, col;

	//opis tabele prehodov
	const static int maxState = 8;//število stanj avtomata
	const static int startState = 0;//začetno stanje avtomata
	const static int noEdge = -1;//ni prehoda
	int automata[maxState + 1][256];//tabela prehodov
```

```
	int finite[maxState + 1];//tabela končnih stanj

	void initAutomata()
	{
		//tabel prehodov je prazna
		for (int i = 0; i <= maxState; i++)
			for (int j = 0; j < 256; j++)
				automata[i][j] = noEdge;

		//Float
		for (int i = '0'; i <= '9'; i++)
			automata[0][i] = automata[1][i] = 1;
		automata[1]['.'] = 2;
		for (int i = '0'; i <= '9'; i++)
			automata[2][i] = automata[3][i] = 3;

		//Operator
		automata[0]['+']   =   automata[0]['*']   =   automata[0]['-']   =
automata[0]['/'] = automata[0]['^'] = automata[0]['%'] = 4;

		//separator
		automata[0]['('] = automata[0][')'] = 5;

		//variable
		for (int i = 'a'; i <= 'z'; i++)
			automata[0][i] = automata[6][i] = 6;
		for (int i = 'A'; i <= 'Z'; i++)
			automata[0][i] = automata[6][i] = 6;
		for (int i = '0'; i <= '9'; i++)
			automata[6][i] = automata[7][i] = 7;

		//ignore
		automata[0]['\n']   =   automata[0]['\r']   =   automata[0]['\t']   =
automata[0][' '] =
		automata[8]['\n']   =   automata[8]['\r']   =   automata[8]['\t']   =
automata[8][' '] = 8;

		finite[0] = tLexError;
		finite[1] = tFloat;
		finite[2] = tLexError;
		finite[3] = tFloat;
		finite[4] = tOperator;
		finite[5] = tSeparator;
		finite[6] = tVariable;
		finite[7] = tVariable;
		finite[8] = tIgnore;
	}

	int getNextState(int state, int aChar)
	{
		if (aChar == -1)
			return noEdge;
		return automata[state][aChar];
	}

	bool isFiniteState(int state)
	{
```

```
            return finite[state] != tLexError;
      }

      int getFiniteState(int state)
      {
            return finite[state];
      }
      int peek()
      {
            return input->peek();
      }

      int read()
      {
            int temp = input->get();
            col++;
            if (temp == '\n')
            {
                  row++;
                  col = 1;
            }
            return temp;
      }

      bool eof()
      {
            return peek() == -1;
      }

      Token nextTokenImp()
      {
            int currentState = startState;
            string lexem;
            int startCol = col;
            int startRow = row;
            do
            {
                  int tempState = getNextState(currentState, peek());
                  if (tempState != noEdge)
                  {//prehod v novo stanje je možen
                        currentState = tempState;
                        lexem += (char)read();
                  }
                  else
                  {//prehod ni možen, ali je končno stanje
                        if (isFiniteState(currentState))
                        {
                              //stanje je končno in vrnemo osnovni leksikalni
simbol
                              Token    token(lexem,    startCol,    startRow,
getFiniteState(currentState), eof());
                              if (token.getToken() == tIgnore)
                                    return nextToken();
                              else
                                    return token;
                        }
                        else
```

```
                        {//stanje ni končno, vrnemo leksikalno napako
                              return Token("", startCol, startRow, tLexError,
eof());
                        }
                  }
            } while (true);
      }
public:
      const static int tLexError = -1;
      const static int tIgnore = 0;
      const static int tFloat = 1;
      const static int tOperator = 2;
      const static int tSeparator = 3;
      const static int tVariable = 4;

      Scanner(std::shared_ptr<ifstream> input)
      {
            row = 1;
            col = 1;
            initAutomata();
            this->input = input;
      }

      Token nextToken()
      {
            return lastToken = nextTokenImp();
      }

      Token currentToken()
      {
            return lastToken;
      }
};
```

1.19. Leksikalni analizator - Naloga 2

scanner.h

```
#pragma once

#include "token.h"
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

class Scanner
{
private:
      std::shared_ptr<ifstream> input;
      Token lastToken;//trenutni osnovni leksikalni simbol
      int row, col;

      //opis tabele prehodov
```

```
	const static int maxState = 23;//število stanj avtomata
	const static int startState = 0;//začetno stanje avtomata
	const static int noEdge = -1;//ni prehoda
	int automata[maxState + 1][256];//tabela prehodov
	int finite[maxState + 1];//tabela končnih stanj

	void initVariable(int state)
	{
		for (int i = 'a'; i <= 'z'; i++)
			automata[state][i] = 9;
		for (int i = 'A'; i <= 'Z'; i++)
			automata[state][i] = 9;
		for (int i = '0'; i <= '9'; i++)
			automata[state][i] = 10;
	}
	void initAutomata()
	{
		//tabel prehodov je prazna
		for (int i = 0; i <= maxState; i++)
			for (int j = 0; j < 256; j++)
				automata[i][j] = noEdge;

		//Float
		for (int i = '0'; i <= '9'; i++)
			automata[0][i] = automata[1][i] = 1;
		automata[1]['.'] = 2;
		for (int i = '0'; i <= '9'; i++)
			automata[2][i] = automata[3][i] = 3;

		//Operator
		automata[0]['+']   =   automata[0]['*']   =   automata[0]['-']   =
automata[0]['/'] = automata[0]['^'] = automata[0]['%'] = 4;

		//separator
		automata[0]['('] = automata[0][')'] = 5;

		//sentence
		automata[0][';'] = 6;

		//assinment
		automata[0][':'] = 7;
		automata[7]['='] = 8;

		//variable
		for (int i = 'a'; i <= 'z'; i++)
			automata[0][i] = automata[9][i] = 9;
		for (int i = 'A'; i <= 'Z'; i++)
			automata[0][i] = automata[9][i] = 9;
		for (int i = '0'; i <= '9'; i++)
			automata[9][i] = automata[10][i] = 10;

		//Func - CONSOLE
		automata[0]['C'] = 11;
		initVariable(11);
		automata[11]['O'] = 12;
		initVariable(12);
		automata[12]['N'] = 13;
```

```
            initVariable(13);
            automata[13]['S'] = 14;
            initVariable(14);
            automata[14]['O'] = 15;
            initVariable(15);
            automata[15]['L'] = 16;
            initVariable(16);
            automata[16]['E'] = 17;
            initVariable(17);

            //Spec. - to
            automata[0]['t'] = 18;
            initVariable(18);
            automata[18]['o'] = 19;
            initVariable(19);
            //Spec. - for
            automata[0]['f'] = 20;
            initVariable(20);
            automata[20]['o'] = 21;
            initVariable(21);
            automata[21]['r'] = 22;
            initVariable(22);

            //ignore
            automata[0]['\n']  =  automata[0]['\r']  =  automata[0]['\t']  =
automata[0][' '] = 23;
            automata[23]['\n']  =  automata[23]['\r']  =  automata[23]['\t']  =
automata[23][' '] = 23;

            finite[0] = tLexError;
            finite[1] = tFloat;
            finite[2] = tLexError;
            finite[3] = tFloat;
            finite[4] = tOperator;
            finite[5] = tSeparator;
            finite[6] = tSentence;
            finite[7] = tLexError;
            finite[8] = tAssinment;
            finite[9] = tVariable;
            finite[10] = tVariable;
            finite[11] = tVariable;
            finite[12] = tVariable;
            finite[13] = tVariable;
            finite[14] = tVariable;
            finite[15] = tVariable;
            finite[16] = tVariable;
            finite[17] = tFunc;
            finite[18] = tVariable;
            finite[19] = tSpec;
            finite[20] = tVariable;
            finite[21] = tVariable;
            finite[22] = tSpec;
            finite[23] = tIgnore;
      }

      int getNextState(int state, int aChar)
      {
```

```
		if (aChar == -1)
			return noEdge;
		return automata[state][aChar];
	}

	bool isFiniteState(int state)
	{
		return finite[state] != tLexError;
	}

	int getFiniteState(int state)
	{
		return finite[state];
	}
	int peek()
	{
		return input->peek();
	}

	int read()
	{
		int temp = input->get();
		col++;
		if (temp == '\n')
		{
			row++;
			col = 1;
		}
		return temp;
	}

	bool eof()
	{
		return peek() == -1;
	}

	Token nextTokenImp()
	{
		int currentState = startState;
		string lexem;
		int startCol = col;
		int startRow = row;
		do
		{
			int tempState = getNextState(currentState, peek());
			if (tempState != noEdge)
			{//prehod v novo stanje je možen
				currentState = tempState;
				lexem += (char)read();
			}
			else
			{//prehod ni možen, ali je končno stanje
				if (isFiniteState(currentState))
				{
					//stanje je končno in vrnemo osnovni leksikalni
simbol
```

```
                                Token    token(lexem,    startCol,    startRow,
getFiniteState(currentState), eof());
                                if (token.getToken() == tIgnore)
                                      return nextToken();
                                else
                                      return token;
                          }
                          else
                          {//stanje ni končno, vrnemo leksikalno napako
                                return Token("", startCol, startRow, tLexError,
eof());
                          }
                    }
             } while (true);
      }
public:
      const static int tLexError = -1;
      const static int tIgnore = 0;
      const static int tFloat = 1;
      const static int tOperator = 2;
      const static int tSeparator = 3;
      const static int tVariable = 4;
      const static int tSentence = 5;
      const static int tAssinment = 6;
      const static int tFunc = 7;
      const static int tSpec = 8;

      Scanner(std::shared_ptr<ifstream> input)
      {
            row = 1;
            col = 1;
            initAutomata();
            this->input = input;
      }

      Token nextToken()
      {
            return lastToken = nextTokenImp();
      }

      Token currentToken()
      {
            return lastToken;
      }
};
```

1.20. Sintaktični analizator - Naloga 1

naloga.cpp

```
#include "parser.h"

using namespace std;

int main()
```

```
{
	Scanner scanner{ std::make_shared<ifstream>("c:\\input1.txt") };
	Parser parser(scanner);
	cout << std::boolalpha << parser.parse();
	return 0;
}
```

parser.h

```
#pragma once

#include "token.h"
#include "scanner.h"

class Parser
{
private:
	Scanner& scanner;

	bool F()
	{
		if (scanner.currentToken().getLexem().compare("(") == 0)
		{
			scanner.nextToken();
			if (E() && (scanner.currentToken().getLexem().compare(")") ==
0))
			{
				scanner.nextToken();
				return true;
			}
		}
		else if (scanner.currentToken().getToken() == Scanner::tFloat)
		{
			scanner.nextToken();
			return true;
		}
		else if (scanner.currentToken().getToken() == Scanner::tVariable)
		{
			scanner.nextToken();
			return true;
		}
		return false;
	}

	bool TT()
	{
		if ((scanner.currentToken().getLexem().compare("*") == 0)
			|| (scanner.currentToken().getLexem().compare("/") == 0)
			|| (scanner.currentToken().getLexem().compare("^") == 0)
			|| (scanner.currentToken().getLexem().compare("%") == 0))
		{
			scanner.nextToken();
			return F() && TT();
		}
		return true;
	}
```

```
	bool T()
	{
		return F() && TT();
	}

	bool EE()
	{
		if ((scanner.currentToken().getLexem().compare("+") == 0)
			|| (scanner.currentToken().getLexem().compare("-") == 0))
		{
			scanner.nextToken();
			return T() && EE();
		}
		return true;
	}

	bool E()
	{
		return T() && EE();
	}

public:
	Parser(Scanner& scanner) : scanner{scanner}
	{
		scanner.nextToken();
	}

	bool parse()
	{
		return E() && scanner.currentToken().isEof();
	}
};
```

1.21. Sintaktični analizator - Naloga 2

scanner.h

```
#pragma once

#include "token.h"
#include "scanner.h"

class Parser
{
private:
	Scanner& scanner;

	bool A()
	{
		if (B() || C() || D())
			return G();
		return true;
	}

	bool B()
```

```
{
	if ((scanner.currentToken().getLexem().compare("CONSOLE") == 0))
	{
		scanner.nextToken();
		return E();
	}
	return false;
}

bool C()
{
	if ((scanner.currentToken().getLexem().compare("for") == 0))
	{
	scanner.nextToken();
	if (D() && (scanner.currentToken().getLexem().compare("to") == 0))
	{
		scanner.nextToken();
		return E() && H();
	}
	}
	return false;
}

bool G()
{
	if ((scanner.currentToken().getLexem().compare(";") == 0))
	{
		scanner.nextToken();
		return A();
	}
	return true;
}

bool H()
{
	return (B() || C() || D());
}

bool D()
{
	if (scanner.currentToken().getToken() == Scanner::tVariable)
	{
		scanner.nextToken();
		if (scanner.currentToken().getLexem().compare(":=") == 0)
		{
			scanner.nextToken();
			return E();
		}
	}
	return false;
}

bool F()
{
	if (scanner.currentToken().getLexem().compare("(") == 0)
	{
		scanner.nextToken();
```

```
                  if (E() && (scanner.currentToken().getLexem().compare(")") ==
0))
                  {
                        scanner.nextToken();
                        return true;
                  }
            }
            else if (scanner.currentToken().getToken() == Scanner::tFloat)
            {
                  scanner.nextToken();
                  return true;
            }
            else if (scanner.currentToken().getToken() == Scanner::tVariable)
            {
                  scanner.nextToken();
                  return true;
            }
            return false;
      }

      bool TT()
      {
            if ((scanner.currentToken().getLexem().compare("*") == 0)
                  || (scanner.currentToken().getLexem().compare("/") == 0)
                  || (scanner.currentToken().getLexem().compare("^") == 0)
                  || (scanner.currentToken().getLexem().compare("%") == 0))
            {
                  scanner.nextToken();
                  return F() && TT();
            }
            return true;
      }

      bool T()
      {
            return F() && TT();
      }

      bool EE()
      {
            if ((scanner.currentToken().getLexem().compare("+") == 0)
                  || (scanner.currentToken().getLexem().compare("-") == 0))
            {
                  scanner.nextToken();
                  return T() && EE();
            }
            return true;
      }

      bool E()
      {
            return T() && EE();
      }

public:
      Parser(Scanner& scanner) : scanner{scanner}
      {
```

```
			scanner.nextToken();
		}

		bool parse()
		{
			return A() && scanner.currentToken().isEof();
		}
};
```

1.22. SEMANTIČNI ANALIZATOR - Naloga 1

naloga.h

```
#include "parser.h"

using namespace std;

int main()
{
	Scanner scanner{ std::make_shared<ifstream>("c:\\input1.txt") };
	Parser parser(scanner);

	//variables
	parser.getVariables()["var1"] = 10;
	parser.getVariables()["var2"] = 1;

	double value;
	if (parser.parse(value))
		cout << "Result: " << value;
	else
		cout << "Error";

	return 0;
}
```

parser.h

```
#pragma once

#include "token.h"
#include "scanner.h"
#include <map>

class Parser
{
private:
	Scanner& scanner;
	std::map<std::string, double> variables;

	bool F(double& outVal)
	{
		if (scanner.currentToken().getLexem().compare("(") == 0)
		{
			scanner.nextToken();
```

```
                  if                        (E(outVal)                        &&
(scanner.currentToken().getLexem().compare(")") == 0))
                  {
                        scanner.nextToken();
                        return true;
                  }
            }
            else if (scanner.currentToken().getToken() == Scanner::tFloat)
            {
                  outVal = std::stod(scanner.currentToken().getLexem());
                  scanner.nextToken();
                  return true;
            }
            else if (scanner.currentToken().getToken() == Scanner::tVariable)
            {
                  if   (variables.find(scanner.currentToken().getLexem())   ==
variables.end() )
                        return false;
                  outVal = variables[scanner.currentToken().getLexem()];
                  scanner.nextToken();
                  return true;
            }
            return false;
      }

      bool TT(double& outVal, double inVal)
      {
            if (scanner.currentToken().getLexem().compare("*") == 0)
            {
                  scanner.nextToken();
                  double tmpVal;
                  return F(tmpVal) && TT(outVal, inVal * tmpVal);
            }
            else if (scanner.currentToken().getLexem().compare("/") == 0)
            {
                  scanner.nextToken();
                  double tmpVal;
                  return F(tmpVal) && TT(outVal, inVal / tmpVal);
            }
            else  if (scanner.currentToken().getLexem().compare("^") == 0)
            {
                  scanner.nextToken();
                  double tmpVal;
                  return F(tmpVal) && TT(outVal, pow(inVal, tmpVal));
            }
            else if (scanner.currentToken().getLexem().compare("%") == 0)
            {
                  scanner.nextToken();
                  double tmpVal;
                  return F(tmpVal) && TT(outVal,  (int)inVal % (int)tmpVal);
            }
            outVal = inVal;
            return true;
      }

      bool T(double& outVal)
      {
```

```
            double inVal;
            return F(inVal) && TT(outVal, inVal);
      }

      bool EE(double& outVal, double inVal)
      {
            if (scanner.currentToken().getLexem().compare("+") == 0)
            {
                  scanner.nextToken();
                  double tmpVal;
                  return T(tmpVal) && EE(outVal, inVal + tmpVal);
            }
            else if (scanner.currentToken().getLexem().compare("-") == 0)
            {
                  scanner.nextToken();
                  double tmpVal;
                  return T(tmpVal) && EE(outVal, inVal - tmpVal);
            }
            outVal = inVal;
            return true;
      }

      bool E(double& outVal)
      {
            double inVal;
            return T(inVal) && EE(outVal, inVal);
      }

public:
      Parser(Scanner& scanner) : scanner{scanner}
      {
            scanner.nextToken();
      }

      bool parse(double& outVal)
      {
            return E(outVal) && scanner.currentToken().isEof();
      }

      std::map<std::string, double>& getVariables()
      {
            return variables;
      }
};
```

1.23. SEMANTIČNI ANALIZATOR – Naloga 2

parser.h

```
#pragma once
#pragma once

#include "token.h"
#include "scanner.h"
#include <map>
```

```
class Parser
{
private:
	Scanner& scanner;
	std::map<std::string, double> variables;

	bool A()
	{
		std::string varName;
		if (B() || C() || D(varName))
			return G();
		return true;
	}

	bool B()
	{
		if ((scanner.currentToken().getLexem().compare("CONSOLE") == 0))
		{
			scanner.nextToken();
			double tmpVal;
			bool result = E(tmpVal);
			if (result)
				cout << tmpVal << '\n';
			return result;
		}
		return false;
	}

	bool C()
	{
		if ((scanner.currentToken().getLexem().compare("for") == 0))
		{
			scanner.nextToken();
			std::string varName;
			if                        (D(varName)                        &&
(scanner.currentToken().getLexem().compare("to") == 0))
			{
				scanner.nextToken();
				double tmpVal;
				bool result = E(tmpVal);
				if (result)
				{
					size_t filePos = scanner.getPosition();
					Token currToken = scanner.currentToken();

					for (int i = variables[varName]; i <= tmpVal; i++)
					{
						scanner.setPosition(filePos);
						scanner.currentToken() = currToken;

						variables[varName] = i;
						result = result && H();
					}
				}
				return result;
			}
		}
```

```
		return false;
	}

	bool G()
	{
		if ((scanner.currentToken().getLexem().compare(";") == 0))
		{
			scanner.nextToken();
			return A();
		}
		return true;
	}

	bool H()
	{
		std::string tmpVarName;
		return (B() || C() || D(tmpVarName));
	}

	bool D(std::string& outVar)
	{
		if (scanner.currentToken().getToken() == Scanner::tVariable)
		{
			outVar = scanner.currentToken().getLexem();
			scanner.nextToken();
			if (scanner.currentToken().getLexem().compare(":=") == 0)
			{
				scanner.nextToken();
				double tmpVal;
				bool result = E(tmpVal);
				if (result)
					variables[outVar] = tmpVal;
				return result;
			}
		}
		return false;
	}

	bool F(double& outVal)
	{
		if (scanner.currentToken().getLexem().compare("(") == 0)
		{
			scanner.nextToken();
			if                      (E(outVal)                      &&
(scanner.currentToken().getLexem().compare(")") == 0))
			{
				scanner.nextToken();
				return true;
			}
		}
		else if (scanner.currentToken().getToken() == Scanner::tFloat)
		{
			outVal = std::stod(scanner.currentToken().getLexem());
			scanner.nextToken();
			return true;
		}
		else if (scanner.currentToken().getToken() == Scanner::tVariable)
```

```
		{
			if	(variables.find(scanner.currentToken().getLexem())	==
variables.end() )
				 return false;
			outVal = variables[scanner.currentToken().getLexem()];
			scanner.nextToken();
			return true;
		}
		return false;
	}

	bool TT(double& outVal, double inVal)
	{
		if (scanner.currentToken().getLexem().compare("*") == 0)
		{
			scanner.nextToken();
			double tmpVal;
			return F(tmpVal) && TT(outVal, inVal * tmpVal);
		}
		else if (scanner.currentToken().getLexem().compare("/") == 0)
		{
			scanner.nextToken();
			double tmpVal;
			return F(tmpVal) && TT(outVal, inVal / tmpVal);
		}
		else  if (scanner.currentToken().getLexem().compare("^") == 0)
		{
			scanner.nextToken();
			double tmpVal;
			return F(tmpVal) && TT(outVal, pow(inVal, tmpVal));
		}
		else if (scanner.currentToken().getLexem().compare("%") == 0)
		{
			scanner.nextToken();
			double tmpVal;
			return F(tmpVal) && TT(outVal,  (int)inVal % (int)tmpVal);
		}
		outVal = inVal;
		return true;
	}

	bool T(double& outVal)
	{
		double inVal;
		return F(inVal) && TT(outVal, inVal);
	}

	bool EE(double& outVal, double inVal)
	{
		if (scanner.currentToken().getLexem().compare("+") == 0)
		{
			scanner.nextToken();
			double tmpVal;
			return T(tmpVal) && EE(outVal, inVal + tmpVal);
		}
		else if (scanner.currentToken().getLexem().compare("-") == 0)
		{
```

```
                  scanner.nextToken();
                  double tmpVal;
                  return T(tmpVal) && EE(outVal, inVal - tmpVal);
            }
            outVal = inVal;
            return true;
      }

      bool E(double& outVal)
      {
            double inVal;
            return T(inVal) && EE(outVal, inVal);
      }

public:
      Parser(Scanner& scanner) : scanner{scanner}
      {
            scanner.nextToken();
      }

      bool parse()
      {
            return A() && scanner.currentToken().isEof();
      }
};
```

1.24. RAZPOZNAVANJE IN POVPRAŠEVANJE XML DOKUMENTOV - Naloga 1

trgovina.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<trgovina>
      <izdelek>
            <izdelekID>1</izdelekID>
            <izdelekIme>Lenovo Prenosnik IdeaPad G510 2,2 GHz (59-412571) +
Torba za prenosnik aQuip FMB-15CB</izdelekIme>
            <zadnja_osvezitev>7.5.2012 15:33:25</zadnja_osvezitev>
            <url>http://www.mimovrste.com/prenosniki/lenovo-prenosnik-lenovo-
ideapad-g510-22-ghz-59-412571</url>
            <opis>Zaslon: 15.6 cola; Procesor: Intel core i7; Kapaciteta diska:
1TB; RAM: 8GB; Garancij: 24 mesecev</opis>
            <slike>
                  <slika
href="http://i.cdn.nrholding.net/16608815/1000/1000/"/>
                  <slika
href="http://i.cdn.nrholding.net/16608818/1000/1000/"/>
            </slike>
            <cena>649</cena>
            <kategorijaID>639515</kategorijaID>
      </izdelek>
      <izdelek>
            <izdelekID>2</izdelekID>
            <izdelekIme>Lenovo Prenosnik IdeaPad B590 2,4 GHz (59-
362009)</izdelekIme>
```

```
            <zadnja_osvezitev>29.9.2014 15:30:00</zadnja_osvezitev>
            <url>http://www.mimovrste.com/prenosniki/lenovo-prenosnik-lenovo-
ideapad-b590-24-ghz-59-362009</url>
            <opis>Zaslon: 15.6 cola; Procesor: Intel core i3; Kapaciteta diska:
500GB; RAM: 4GB; Garancij: 24 mesecev</opis>
            <slike>
                  <slika
href="http://i.cdn.nrholding.net/16503812/1000/1000"/>
                  <slika
href="http://i.cdn.nrholding.net/16503810/1000/1000"/>
            </slike>
            <cena>399</cena>
            <kategorijaID>619976</kategorijaID>
      </izdelek>
</trgovina>
```

trgovina.xsd

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema                          xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
      <xs:element name="trgovina" type="trgovinaType" />
      <xs:complexType name="trgovinaType">
            <xs:sequence>
                  <xs:element  name="izdelek"  type="izdelekType"  minOccurs="1"
maxOccurs="unbounded" />
            </xs:sequence>
      </xs:complexType>

      <xs:complexType name="izdelekType">
            <xs:sequence>
                  <xs:element name="izdelekID">
                        <xs:simpleType>
                              <xs:restriction base="xs:integer">
                                    <xs:minInclusive value="1" />
                                    <xs:maxInclusive value="1000" />
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>
                  <xs:element name="izdelekIme" type="xs:string" />
                  <xs:element name="zadnja_osvezitev">
                        <xs:simpleType>
                              <xs:restriction base="xs:string">
                                    <xs:pattern          value="[0-9]{1,2}.[0-
9]{1,2}.[0-9]{4} [0-9]{1,2}:[0-9]{1,2}:[0-9]{1,2}" />
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>
                  <xs:element name="url" type="xs:anyURI" />
                  <xs:element name="opis" type="xs:string" />
                  <xs:element name="slike" type="slikeType" />
                  <xs:element name="cena">
                        <xs:simpleType>
                              <xs:restriction base="xs:decimal" >
                                    <xs:minInclusive value="0" />
```

```
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>
                  <xs:element name="kategorijaID">
                        <xs:simpleType>
                              <xs:restriction base="xs:integer">
                                    <xs:minInclusive value="1" />
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>

      <xs:complexType name="slikeType">
            <xs:sequence>
                  <xs:element name="slika" minOccurs="1" maxOccurs="unbounded"
>
                        <xs:complexType>
                              <xs:attribute    name="href"    type="xs:string"
use="required" />
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>

</xs:schema>
```

1.25. RAZPOZNAVANJE IN POVPRAŠEVANJE XML DOKUMENTOV – Naloga 2

naloga.cs

```
using System;
using System.Xml;
using System.Xml.Schema;
using System.Xml.XPath;

namespace Naloga2
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                // Validate
                XmlReaderSettings settings = new XmlReaderSettings();
                settings.Schemas.Add(null, "c:\\trgovina.xsd");
                settings.ValidationType = ValidationType.Schema;

                XmlReader    reader    =    XmlReader.Create("c:\\trgovina.xml",
settings);
                XmlDocument document = new XmlDocument();
                document.Load(reader);
```

```
                // Create a new element node.
                XmlNode newElem = document.CreateNode("element", "myNewNode",
"");
                newElem.InnerText = "My text";
                document.DocumentElement.AppendChild(newElem);
                Console.WriteLine(document.OuterXml);

                // Delete element
                document.DocumentElement.RemoveChild(newElem);
                Console.WriteLine(document.OuterXml);

                //XPath
                Console.WriteLine("XPath:");
                XmlNodeList                         nodes                         =
document.SelectNodes("/trgovina/izdelek/izdelekIme");
                foreach (XmlNode node in nodes)
                    Console.WriteLine(node.InnerText);

                // Write New XML structure
                document.Save("c:\\new_trgovina.xml");
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
}
```

1.26. GENERATORJI PREVAJALNIKOV - Naloga 1

example.expr

```
a = 6

b = 9

a+b

(-1+2)*4

2^8
```

EvalVisitor.java

```
import java.util.HashMap;
import java.util.Map;
import java.lang.Math;

public class EvalVisitor extends ExprBaseVisitor<Float> {
    Map<String, Float> memory = new HashMap<String, Float>();

      // expr NEWLINE
    @Override public Float visitPrintExpr(ExprParser.PrintExprContext ctx) {
        float value = visit(ctx.expr()); // evaluate the expr child
```

```
        System.out.println(value);         // print the result
        return Float.valueOf(0);                          // return dummy value
    }

    // VARIABLE '=' expr NEWLINE
    @Override public Float visitAssign(ExprParser.AssignContext ctx) {
        String var = ctx.VARIABLE().getText();  // var is left-hand side of '='
        float value = visit(ctx.expr());    // compute value of expression on
right
        memory.put(var, value);           // store it in our memory
        return value;
    }

      // FLOAT
    @Override public Float visitNumber(ExprParser.NumberContext ctx) {
            Float f = Float.valueOf(ctx.FLOAT().getText());
        return f;
    }

      // - FLOAT
    @Override public Float visitMinNumber(ExprParser.MinNumberContext ctx) {
        return -Float.valueOf(ctx.FLOAT().getText());
    }

    // '(' expr ')'
      @Override public Float visitParens(ExprParser.ParensContext ctx) {
        return visit(ctx.expr()); // return child expr's value
    }

    // VARIABLE
    @Override public Float visitVariable(ExprParser.VariableContext ctx) {
        String var = ctx.VARIABLE().getText();
        if (memory.containsKey(var))
                  return memory.get(var);
        return Float.valueOf(0);
    }

      // expr op=('+'|'-') expr
    @Override public Float visitAddSub(ExprParser.AddSubContext ctx) {
            float left = visit(ctx.expr(0));  // get value of left subexpression
        float right = visit(ctx.expr(1)); // get value of right subexpression
        if (ctx.op.getType() == ExprParser.ADD)
                  return left + right;
        return left - right; // must be SUB
    }

      // expr ('*'|'/'|'^') expr
      @Override public Float visitMulDivPow(ExprParser.MulDivPowContext ctx) {
        float left = visit(ctx.expr(0));  // get value of left subexpression
        float right = visit(ctx.expr(1)); // get value of right subexpression
        if (ctx.op.getType() == ExprParser.MUL)
                  return left * right;
            else if (ctx.op.getType() == ExprParser.DIV)
                  return left / right;
            return (float)Math.pow(left, right);
    }
}
```

Calculator.java

```
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.tree.ParseTree;
import java.io.FileInputStream;
import java.io.InputStream;

public class Calculator {
    public static void main(String[] args) throws Exception {
        if (args.length < 1) {
                System.out.println("input file is missing");
                return;
            }
        CharStream codePointCharStream = CharStreams.fromFileName(args[0]);

        ExprLexer lexer = new ExprLexer(codePointCharStream);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        ExprParser parser = new ExprParser(tokens);
        ParseTree tree = parser.prog(); // parse

        EvalVisitor eval = new EvalVisitor();
        eval.visit(tree);
    }
}
```

Viri in literatura

A. V. Aho, M. S. (2007). *Compilers – Principles, Techniques, and Tools, Second Edition.* Reading: Addison-Wesley.

A. W. Appel, J. P. (2002). *Modern compiler implementation in Java.* Cambridge University Press.

J. E. Hopcroft, R. M. (2001). *Introduction to Automata Theory, Languages, and Computation, Second Edition.* Stanford: Addison-Wesley.

Vilfan, B. (1991). *Prevajanje programskih jezikov, I. del.* Ljubljana: Univerza v Ljubljani, Fakulteta za elektrotehniko in računalništvo.