Министерство образования РФ

Государственное образовательное учреждение

высшего профессионального образования

“Тверской государственный университет”

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Факультет прикладной математики и кибернетики

**ОТЧЕТ ПО УЧЕБНО-ВЫЧИСЛИТЕЛЬНОЙ ПРАКТИКЕ**

Направление: «*Прикладная математика и информатика»*

Специализация: *Бакалавриат*

**Выполнил:**

студент 24 группы

Доронин Виталий Евгеньевич

**Проверил:**

Доцент кафедры информатики

Дадеркин Дмитрий Ольгердович

*Тверь – 2016*

# 

# **Оглавление**

[**Оглавление** 2](#_Toc515558604)

[**Задание по практике** 2](#_Toc515558605)

[**Введение** 2](#_Toc515558606)

[**Задание №1** 3](#_Toc515558607)

[Алгоритм и описание работы 3](#_Toc515558608)

[**Задание №2** 5](#_Toc515558609)

[Алгоритм и описание работы 5](#_Toc515558610)

[**Вывод** 5](#_Toc515558611)

[**Приложения** 5](#_Toc515558612)

[Исходные тексты программ 5](#_Toc515558613)

[Программа №1: 5](#_Toc515558614)

[Программа №2: 10](#_Toc515558615)

[**Список использованной литературы** 13](#_Toc515558616)

# **Задание по практике**

**1**. Создать и описать принцип работы синтаксического анализатора языка MINI-BASIC.

**2**. Создать и описать принцип работы генератора кода на базе синтаксического анализатора языка MINI-BASIC.

# **Введение**

В данной работе будет рассмотрено устройство и принципы работы синтаксического анализатора языка MINI-BASIC, а также генератора кода на его базе.

# **Задание №1**

## Алгоритм и описание работы

В основе **синтаксического анализатора** лежит автомат с магазинной памятью, использующий следующую LL(2)-грамматику:

<S> -> <A> <B> END

<A> -> <L> <A>

<A> -> e

<L> -> <B> <C>

<B> -> const

<C> -> LET Var = <F>

<C> -> GOTO <B>

<C> -> IF <J> <B>

<C> -> FOR Var = <F> TO <F> STEP <F> <A> <B> NEXT

<C> -> GOSUB <B>

<C> -> RETURN

<C> -> INPUT

<C> -> PRINT

<J> -> <F> <J'>

<J'> -> =<F> <J'>

<J'> -> < <F>

<J'> -> > <F>

<F> -> <M> <N> <F'>

<F'> -> +<M> <N> <F'>

<F'> -> -<M> <N> <F'>

<F'> -> e

<N> -> e

<N> -> \*<M> <N>

<N> -> /<M> <N>

<M> -> (<M> <N> <F'>)

<M> -> c

<M> -> Var

1. При запуске синтаксического анализатора происходит сортировка меток по возрастанию.
2. Начинается работа с эмулятором конечного автомата. В случае, если на вершине стека находится терминальный символ, совпадающий с таковым на ленте, анализатор «съедает» его. Иначе выводится сообщение об ошибке.  
     
   Если на вершине стека находится нетерминал, то в силу использования LL(2)-грамматики по первым двум символам на ленте выбирается подходящее правило, правая часть которого переходит на стек в обратном порядке.
3. На следующем этапе происходит добавление унарного минуса всем константам.
4. Каждой переменной выражения присваивается свой адрес в памяти.
5. Происходит удаление меток с их заменой на команды.
6. На выходе имеем список атомов (команд).

# **Задание №2**

## Алгоритм и описание работы

**Генератор кода** начинает свою работу опираясь на выходные данные синтаксического анализатора. Список команд, полученный на предыдущем этапе представляет собой обратную польскую запись, которая преобразуется в программный код на языке MINI-BASIC, тем самым образуя изначальные входные данные.

# **Вывод**

Таким образом, в ходе данной работы было рассмотрено устройство и принцип работы синтаксического анализатора языка MINI-BASIC с подробным описанием основных шагов алгоритма, по которому выполняется синтаксический анализ.

# **Приложения**

## Исходный текст программы

#include "stdafx.h"

#include "iostream"

#include "fstream"

#include "fautomata.h"

#include "sautomata.h"

#include "algorithm"

**using** **namespace** std**;**

bool **operator** **<** **(**const vector**<**Lexeme**>&** v**,** const vector**<**Lexeme**>&** v1**)**

**{**

**if** **(**v**.**size**()** **==** 0 **||** v1**.**size**()** **==** 0**)**

**throw** "Unexpected input"**;**

**if** **((\***v**.**begin**()).**getType**()** **!=** NUMBER **||** **(\***v1**.**begin**()).**getType**()** **!=** NUMBER**)**

**throw** "Unexpected input"**;**

**return** **(\***v**.**begin**()).**getValue**()<(\***v1**.**begin**()).**getValue**();**

**}**

string getVarName**(**string name**)**

**{**

name **=** name**.**substr**(**1**,** name**.**length**()** **-** 2**);**

**if** **(**name**.**at**(**0**)** **==** '>'**)**

**{**

name **=** name**.**substr**(**1**,** name**.**length**()** **-** 1**);**

**}**

**return** name**;**

**}**

string getLabel**(**string label**)**

**{**

**if** **(**label**.**at**(**1**)** **==** 'l'**)**

**{**

**return** label**.**substr**(**3**,** label**.**length**()** **-** 4**);**

**}**

**if** **(**label**.**at**(**1**)** **==** 'j'**)**

**{**

**return** label**.**substr**(**7**,** label**.**length**()** **-** 8**);**

**}**

**if** **(**label**.**at**(**1**)** **==** 'i'**)**

**{**

**return** label**.**substr**(**6**,** label**.**length**()** **-** 7**);**

**}**

**return** label**;**

**}**

int main**()**

**{**

ifstream in**(**"d:/input.txt"**,** std**::**ifstream**::**in**);**

FAutomata a**(**in**);**

**try**

**{**

a**.**start**();**

vector**<**Lexeme**>** res **=** a**.**getResult**();**

vector**<**Lexeme**>** copy**;**

**for** **(**vector**<**Lexeme**>::**iterator i **=** res**.**begin**();** i **!=** res**.**end**();** **++**i**)**

**{**

**if** **((\***i**).**getType**()** **==** OP\_MINUS**)**

**{**

**if** **((**i **==** res**.**begin**())** **||**

**((\*(**i **-** 1**)).**getType**()** **!=** R\_BRACKET **&&** **(\*(**i **-** 1**)).**getType**()** **!=** NUMBER**))**

**{**

**if** **((**i **+** 1**)** **==** res**.**end**())**

**{**

copy**.**push\_back**(\***i**);**

**}**

**else** **if** **((\*(**i **+** 1**)).**getType**()** **==** NUMBER**)**

**{**

**(\*(**i **+** 1**)).**setValue**(-(\*(**i **+** 1**)).**getValue**());**

**}**

**else**

copy**.**push\_back**(\***i**);**

**}**

**else**

copy**.**push\_back**(\***i**);**

**}**

**else**

copy**.**push\_back**(\***i**);**

**}**

vector**<**vector**<**Lexeme**>>** lines**;**

vector**<**Lexeme**>** line**;**

**for** **(**vector**<**Lexeme**>::**iterator i **=** copy**.**begin**();** i **!=** copy**.**end**();** **++**i**)**

**{**

**if** **((\***i**).**getType**()** **==** EOL**)**

**{**

lines**.**push\_back**(**line**);**

line **=** vector**<**Lexeme**>();**

**}**

**else**

**{**

line**.**push\_back**((\***i**));**

**}**

**}**

**if** **(**line**.**size**()** **!=** 0**)**

**{**

lines**.**push\_back**(**line**);**

**}**

sort**(**lines**.**begin**(),** lines**.**end**());**

res **=** vector**<**Lexeme**>();**

**for** **(**vector**<**vector**<**Lexeme**>>::**iterator i **=** lines**.**begin**();** i **!=** lines**.**end**();** i**++)**

**{**

vector**<**Lexeme**>** v **=** **(\***i**);**

**for** **(**vector**<**Lexeme**>::**iterator j **=** v**.**begin**();** j **!=** v**.**end**();** j**++)**

**{**

res**.**push\_back**((\***j**));**

**}**

**}**

sautomata sa**;**

vector**<**Symbol**>** out **=** sa**.**Start**(**res**);**

int currentIndex **=** 0**;**

bool exit **=** **false;**

**while** **(!**exit**)**

**{**

exit **=** **true;**

**for** **(**vector**<**Symbol**>::**iterator i **=** out**.**begin**();** i **!=** out**.**end**();** **++**i**)**

**{**

**if** **((\***i**).**lexeme**.**getType**()** **==** VAR**)**

**{**

exit **=** **false;**

string name **=** **(\***i**).**name**;**

**if** **(**name**.**at**(**1**)** **==** '>'**)**

**{**

**(\***i**).**name **=** "[>X" **+** to\_string**(**currentIndex**)** **+** "]"**;**

**}**

**else**

**{**

**(\***i**).**name **=** "[X" **+** to\_string**(**currentIndex**)** **+** "]"**;**

**}**

**(\***i**).**lexeme**.**setType**(**UNDEFINED**);**

name **=** getVarName**(**name**);**

**for** **(**vector**<**Symbol**>::**iterator j **=** i **+** 1**;** j **!=** out**.**end**();** j**++)**

**{**

**if** **((\***j**).**lexeme**.**getType**()** **==** VAR**)**

**{**

string temp **=** **(\***j**).**name**;**

temp **=** getVarName**(**temp**);**

**if** **(**temp **==** name**)**

**{**

**if** **((\***j**).**name**.**at**(**1**)** **==** '>'**)**

**{**

**(\***j**).**name **=** "[>X" **+** to\_string**(**currentIndex**)** **+** "]"**;**

**}**

**else**

**{**

**(\***j**).**name **=** "[X" **+** to\_string**(**currentIndex**)** **+** "]"**;**

**}**

**(\***j**).**lexeme**.**setType**(**UNDEFINED**);**

**}**

**}**

**}**

currentIndex **=** currentIndex **+** 1**;**

**break;**

**}**

**}**

**}**

exit **=** **false;**

**while** **(!**exit**)**

**{**

exit **=** **true;**

int counter **=** 0**;**

**for** **(**vector**<**Symbol**>::**iterator i **=** out**.**begin**();** i **!=** out**.**end**();** **++**i**)**

**{**

**if** **((\***i**).**lexeme**.**getType**()** **==** LABEL**)**

**{**

exit **=** **false;**

string label **=** **(\***i**).**name**;**

label **=** getLabel**(**label**);**

**for** **(**vector**<**Symbol**>::**iterator j **=** out**.**begin**();** j **!=** out**.**end**();** j**++)**

**{**

**if** **((\***j**).**lexeme**.**getType**()** **==** LABEL\_USE**)**

**{**

string temp **=** **(\***j**).**name**;**

temp **=** getLabel**(**temp**);**

**if** **(**temp **==** label**)**

**{**

**if** **((\***j**).**name**[**1**]** **==** 'j'**)**

**{**

**(\***j**).**name **=** "[jmp " **+** to\_string**(**counter**)** **+** "]"**;**

**(\***j**).**lexeme**.**setType**(**UNDEFINED**);**

**}**

**else** **if** **((\***j**).**name**[**1**]** **==** 'i'**)**

**{**

**(\***j**).**name **=** "[ij " **+** to\_string**(**counter**)** **+** "]"**;**

**(\***j**).**lexeme**.**setType**(**UNDEFINED**);**

**}**

**else**

**{**

**throw** "Unexpected command: " **+** **(\***j**).**name**;**

**}**

**}**

**}**

**}**

**}**

**else**

**{**

counter**++;**

**}**

**}**

**break;**

**}**

vector**<**Symbol**>::**iterator i **=** out**.**begin**();**

**while** **(**i **!=** out**.**end**())**

**{**

**if** **((\***i**).**lexeme**.**getType**()** **==** LABEL**)**

**{**

out**.**erase**(**i**);**

i **=** out**.**begin**();**

**}**

**else**

**{**

i**++;**

**}**

**}**

**for** **(**vector**<**Symbol**>::**iterator i **=** out**.**begin**();** i **!=** out**.**end**();** **++**i**)**

**{**

**if** **((\***i**).**lexeme**.**getType**()** **==** LABEL\_USE**)**

**{**

**if** **((\***i**).**name**[**1**]** **==** 'j'**)**

**{**

**(\***i**).**name **=** "[jmp " **+** to\_string**(**out**.**size**())** **+** "]"**;**

**(\***i**).**lexeme**.**setType**(**UNDEFINED**);**

**}**

**else** **if** **((\***i**).**name**[**1**]** **==** 'i'**)**

**{**

**(\***i**).**name **=** "[ij " **+** to\_string**(**out**.**size**())** **+** "]"**;**

**(\***i**).**lexeme**.**setType**(**UNDEFINED**);**

**}**

**else**

**{**

**throw** "Unexpected command: " **+** **(\***i**).**name**;**

**}**

**}**

**}**

int num **=** 0**;**

**for** **(**vector**<**Symbol**>::**iterator i **=** out**.**begin**();** i **!=** out**.**end**();** **++**i**)**

**{**

cout **<<** num **<<** ":" **<<** **(\***i**).**name **<<** endl**;**

num**++;**

**}**

**}**

**catch** **(**const char**\***e**)**

**{**

cout **<<** "Error:" **<<** e **<<** endl**;**

**}**

**catch** **(**std**::**string str**)**

**{**

cout **<<** str **<<** endl**;**

**}**

**return** 0**;**

**}**

**Список использованной литературы**

1. Н. Вирт, «Построение компиляторов», Москва, 2010, 194 стр.
2. А. Ахо, Дж. Ульман, «Теория синтаксического анализа, перевода и компиляции, Том 1», Москва, 1978, 308
3. А. Ахо, Дж. Ульман, «Теория синтаксического анализа, перевода и компиляции, Том 2», Москва, 1978, 243