Министерство науки и высшего образования РФ

ФГАОУ ВО Пермский национальный исследовательский

политехнический университет

Кафедра «Вычислительная математика, механика и биомеханика»

Отчет по лабораторной работе № 3

тема «Разработка синтаксически управляемого транслятора с использованием flex и bison»

по дисциплине «Языки программирования и методы трансляции»

Выполнил: студент группы ИСТ-22-1б Богданов Д.О.

Проверил: Батин Сергей Евгеньевич

Пермь, 2024

**Содержание**

[Задание 1 3](#_Toc157104548)

[1.1. Постановка задачи 3](#_Toc157104549)

[1.2. Решение задачи, код программы 3](#_Toc157104550)

[1.3. Решение задачи, код программы 3](#_Toc157104551)

[1.4. Тестирование работы программы 4](#_Toc157104552)

[Задание 2 5](#_Toc157104553)

[2.1. Постановка задачи 5](#_Toc157104554)

[2.2. Решение задачи, код программы 5](#_Toc157104555)

[2.3. Решение задачи, код программы 6](#_Toc157104556)

[2.4. Тестирование работы программы 7](#_Toc157104557)

[Задание 3 8](#_Toc157104558)

[3.1. Постановка задачи 8](#_Toc157104559)

[3.2. Решение задачи, код программы 8](#_Toc157104560)

[3.3. Тестирование работы программы 8](#_Toc157104561)

[Задание 4 9](#_Toc157104562)

[4.1. Постановка задачи 9](#_Toc157104563)

[4.2. Решение задачи, код программы 9](#_Toc157104564)

[4.3. Тестирование работы программы 9](#_Toc157104565)

[Задание 5 11](#_Toc157104566)

[5.1. Постановка задачи 11](#_Toc157104567)

[5.2. Решение задачи, код программы 11](#_Toc157104568)

[5.3. Тестирование работы программы 12](#_Toc157104569)

# Задание 1

## 1.1. Постановка задачи

Написать программу для генератора лексических анализаторов, выводящую в консоль переданный токен.

## 1.2. Решение задачи, код программы

%{

#include <stdio.h>

%}

**%%**

[0-9]+ {printf("NUMBER %s\n", yytext);}

[0-9]+(\.[0-9]+)? {printf("FLT NUMBER %s\n", yytext);}

[a-zA-Z] {printf("ALPHABET SYM %s\n", yytext);}

[a-zA-Z]+ {printf("WORD %s\n", yytext);}

"+" {printf("PLUS\n");}

"-" {printf("MINUS\n");}

"\*" {printf("MULT\n");}

"=" {printf("EQUAL\n");}

"/" {printf("DIVIDE\n");}

[ \t] {}

\n {printf("NEW LINE\n");}

. {printf("UNKNWN SYMB %s\n", yytext);}

**%%**

int main()

{

yylex();

return 1;

}

## 1.3. Решение задачи, код программы

Код программы анализатора состоит из 3 частей:

1 – объявление, опции, в нашем случае добавление библиотеки для ввода/вывода

2 – описание паттернов и действий, при обнаружении очередного токена

3 – код на языке Си, использующийся для вывода токенов на экран

## 1.4. Тестирование работы программы

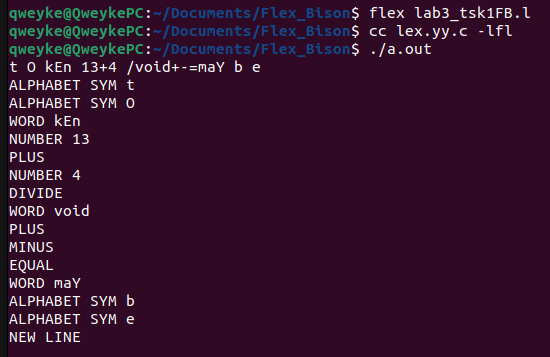


Рис.1 Тестирование программы lab3tsk1FB.l

# Задание 2

## 2.1. Постановка задачи

Описать грамматику, позволяющую вычислять значения арифметических выражений, состоящих из произвольного конечного количества целых чисел и знаков «+» и «-»

## 2.2. Решение задачи, код программы

%{

#include "lab3\_tsk2Bison.tab.h"

%}

**%%**

[0-9] {(yylval = atoi(yytext)); return NUMBER;}

"+" {return ADD;}

"-" {return SUB;}

[ \t] {}

\n {return EOL;}

. {printf("UNKNW SYMB %s\n", yytext);}

**%%**

Приложение 1 Код лексического анализатора Flex

%{

#include <stdio.h>

int yylex(void);

void yyerror();

%}

%token NUMBER

%token ADD SUB

%token EOL

%left ADD SUB

**%%**

calculation:

|calculation exp EOL {printf("=%d\n", $2);}

;

exp: NUMBER

|exp ADD exp { $$ = $1 + $3; }

|exp SUB exp { $$ = $1 - $3; }

;

**%%**

int main()

{

yyparse();

}

void yyerror()

{

printf("Error\n");

}

Приложение 2 Код синтаксического анализатора Bison

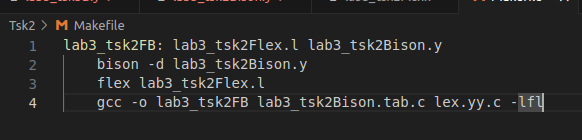


Рис.2 Код Makefile

## 2.3. Решение задачи, код программы

Код программы анализатора состоит из 2 частей:

1 – объявления, опции, в нашем случае добавление библиотеки с определениями токенов бизона

2 – описание паттернов и действий, при обнаружении очередного токена

Код программы парсера состоит из 4 частей:

1 – объявления функции лексера, возвращающей код токена, функции, возвращающей сообщение об ошибке в парсинге, добавление библиотеки с инструкциями ввода/вывода

2 – объявление токенов

3 – определение построения действий, их последовательность выполнения

4 – код Си с функцией запуска парсинга и функцией вывода ошибки в терминал

Далее собирается мэйк файл, производится тестирование.

## 2.4. Тестирование работы программы

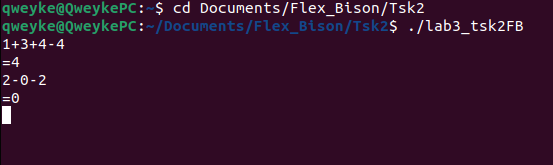


Рис.3 Тестирование программы lab3tsk2FB

# Задание 3

## 3.1. Постановка задачи

Описать грамматику, позволяющую вычислять значения арифметических выражений, состоящих из произвольного конечного количества целых чисел и знаков «+» и «-»

## 3.2. Решение задачи, код программы

%{

#include "lab3\_tsk3B.tab.h"

%}

**%%**

[0-9]+ {yylval = atoi(yytext); return NUM;}

"+" {return ADD;}

"-" {return SUB;}

"/" {return DIV;}

"\*" {return MUL;}

"(" {return \*yytext;}

")" {return \*yytext;}

[ \t] {}

\n {return EOL;}

. {printf("UNKNOWN INPUT %s", yytext);}

**%%**

Приложение 3 Код лексического анализатора Flex

%{

#include <stdio.h>

int yylex(void);

void yyerror();

%}

%token NUM

%token ADD SUB DIV MUL

%token EOL

%left ADD SUB

%left MUL DIV

**%%**

calculation:

|calculation exp EOL {printf("\n");}

;

exp: term

|exp ADD exp { printf("+ "); }

|exp SUB exp { printf("- "); }

|exp MUL exp { printf ("\* ");}

|exp DIV exp { printf ("/ ");}

;

term: NUM {printf("%d ", $1);}

|'(' exp ')' {$$ = $2;}

;

**%%**

int main(void)

{

yyparse();

}

void yyerror()

{

printf("Error");

}

Приложение 4 Код синтаксического анализатора Bison

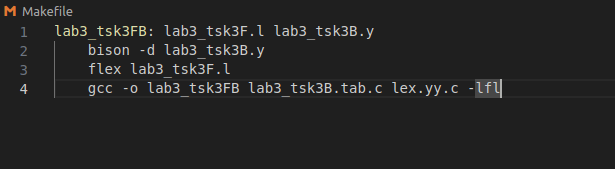


Рис.4 Код Makefile

## 3.3. Решение задачи, код программы

Код программы анализатора состоит из 2 частей:

1 – объявления, опции, в нашем случае добавление библиотеки с определениями токенов бизона

2 – описание паттернов и действий, при обнаружении очередного токена

Код программы парсера состоит из 5 частей:

1 – объявления функции лексера, возвращающей код токена, функции, возвращающей сообщение об ошибке в парсинге, добавление библиотеки с инструкциями ввода/вывода

2 – объявление токенов

3 – определение очередности парсинга токенов, сторону рекурсивности

4 – определение построения действий, их последовательность выполнения

5 – код Си с функцией запуска парсинга и функцией вывода ошибки в терминал

Далее собирается мэйк файл, производится тестирование.

Программа рекурсивно разбирает поступающие лексемы, последовательно разбирая их, строит дерево, последовательно выполняет его, начиная с term.

Если exp состоит из двух полноценных exp со знаком, то выводит операнды первого выражения и его знак, потом второго, а после знак этого составного exp, как и должно быть в постфиксной записи. При определении парсером неполного exp, например с NUM на одном конце – знак будет присвоен той паре чисел или выражений, которая была определена главнее в объявлении %{%}. Скобки имеют такой же приоритет, что и NUM, поэтому exp в них имеет приоритет в присвоении знаков.

## 3.4. Тестирование работы программы

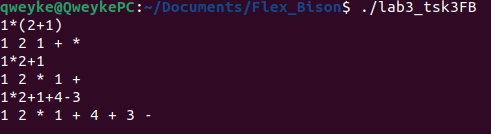


Рис.3 Тестирование программы lab3tsk2FB