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 Зверев А.А.

Проверил: Батин Сергей Евгеньевич

Пермь, 2023

**Содержание**

[Задание 1 3](#_Toc154015532)

[1.1. Постановка задачи 3](#_Toc154015533)

[1.2. Решение задачи, код программы 3](#_Toc154015534)

[1.3. Тестирование работы программы 4](#_Toc154015535)

[Задание 2 5](#_Toc154015536)

[2.1. Постановка задачи 5](#_Toc154015537)

[2.2. Решение задачи, код программы 5](#_Toc154015538)

[(int argc, char \*\*argv): это список параметров функции main. argc является целочисленной переменной, которая хранит количество аргументов командной строки, переданных программе. char \*\*argv является указателем на указатель на символьные строки (массив аргументов командной строки). Обычно argv используется для доступа к переданным программе аргументам командной строки. 7](#_Toc154015539)

[2.3. Тестирование работы программы 7](#_Toc154015540)

[Задание 3 7](#_Toc154015542)

[3.1. Постановка задачи 7](#_Toc154015543)

[3.2. Решение задачи, код программы 7](#_Toc154015544)

[3.3. Тестирование работы программы 9](#_Toc154015545)

# Задание 1

## 1.1. Постановка задачи

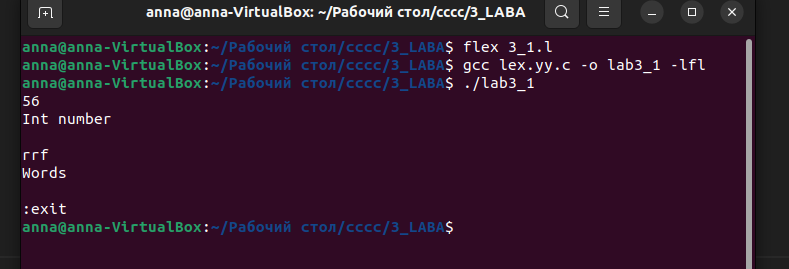
Написать программу для генератора лексических анализаторов, выводящую в консоль переданный токен.

## 1.2. Решение задачи, код программы

1. %{
2. #include <stdio.h>
4. // Объявление функций для обработки ошибки и выхода
5. **void** error();
6. **void** exit();
8. %}
10. // Определение правил распознавания лексем
11. %%
12. [0-9]+       { printf("Integer number\n"); }                // Правило для целых чисел
13. [a-zA-Z]     { printf("Letter\n"); }              // Правило для одной буквы
14. [a-zA-Z]+    { printf("Words\n"); }     // Правило для слов (последовательность букв)
15. [0-9]+\.[0-9]{ printf("Float/double number\n"); }  // Правило для чисел с плавающей точкой или двойной точности
17. "+"         { printf("Plus operator\n"); }                  // Правило для оператора сложения
18. "-"         { printf("Minus operator\n"); }                 // Правило для оператора вычитания
19. .           { printf("Unexpected character: %s\n", yytext); }  // Правило для обработки недопустимых символов
20. "\*"         { printf("Multiply operator\n"); }        // Правило для оператора умножения
21. "/"         { printf("Division operator\n"); }        // Правило для оператора деления
22. [\t]+       ;           // Пропуск символов табуляции
23. .           { error(); }                       // Правило для обработки недопустимых символов
24. "end"       { exit(); }              // Правило для команды выхода "end"
25. %%
27. // Функция для обработки ошибки
28. **void** error(){
29. printf("There is an error\n");
30. }
32. // Функция для выхода из программы при вводе "end"
33. **void** exit(){
34. exit(0);
35. }

## 1.3. Тестирование работы программы

Рис.1.1 Тестирование программы lab3\_1



# 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

**Файл 3\_2.l**

1. #include <stdio.h>
2. #include <string.h>
3. #include "3\_3.tab.h"
5. // Объявление функций для управления выходом
6. **void** exit\_ctrlq();
7. **void** error();
9. // Опция noyywrap для генератора лексического анализатора
10. %option noyywrap
12. // Объявление разделителей и правил перехода
13. %%
14. [0-9]+(\.[0-9]+)? { yylval.number=atof(yytext); **return** NUM; }// Правило для чисел
15. [-+\*/(){}]        {**return** yytext[0]; }    // Правило для операторов и скобок
16. [\n]              { **return** RAVNO;}   // Правило для символа новой строки
17. [ \t]  /\* skip whitespaces \*/             // Правило для пропуска пробелов и табуляций
18. ":exit"           {exit\_ctrlq();}    // Правило для команды выхода ":exit"
19. .                 {error();}  // Правило для обработки недопустимых символов
20. %%
22. // Функция обработки ошибки
23. **void** error(){
24. printf("Error");
25. }
27. // Функция для выхода из программы при вводе ":exit"
28. **void** exit\_ctrlq(){
29. exit(0);
30. }

**Файл lab3\_2.y**

1. %{
2. #include <stdio.h>
3. #include <string.h>
4. #include "3\_2.tab.h"   // Подключение заголовочного файла "3\_2.tab.h"
6. // Объявление функций и переменных
7. **int** yylex(**void**);
8. **void** yyerror(**const** **char** \*str);
9. **int** main(**int** argc, **char** \*\*argv);
10. %}
12. %token SUM MIN MULT DEL     // Определение токенов
13. %token INTEGER
14. %token prSKOBKA levSKOBKA
15. %token RAVNO
17. %%
19. calclist:
20. | calclist exp RAVNO { printf("= %d\n", $2); }    // Правило для выражения calclist
21. ;                                    // Код действия: вывод результата выражения
23. exp: vtorich                                   // Правило для выражения exp
24. | exp SUM vtorich { $$ = $1 + $3; }       // Правило для суммы двух выражений
25. | exp MIN vtorich { $$ = $1 - $3; }      // Правило для разности двух выражений
26. ;
28. vtorich: pervich                           // Правило для выражения vtorich
29. | vtorich MULT pervich { $$ = $1 \* $3; }  // Правило для умножения двух выражений
30. | vtorich DEL  pervich  { $$ = $1 / $3; }  // Правило для деления двух выражений
31. ;
33. pervich: INTEGER                            // Правило для выражения pervich
34. | prSKOBKA exp levSKOBKA { $$ = $2 ;   // Правило для обработки выражений в скобках
35. ;
37. %%
39. // Функция обработки ошибок
40. **void** yyerror(**const** **char** \*str){
41. fprintf(stderr,"Error: %s\n",str);
42. }
44. **int** main(**int** argc, **char** \*\*argv){
45. yyparse();    // Запуск синтаксического анализатора
46. **return** 0;
47. }

## (int argc, char \*\*argv): это список параметров функции main. argc является целочисленной переменной, которая хранит количество аргументов командной строки, переданных программе. char \*\*argv является указателем на указатель на символьные строки (массив аргументов командной строки). Обычно argv используется для доступа к переданным программе аргументам командной строки.

## 2.3. Тестирование работы программы

## C:\Users\79824\Downloads\image (5).png

## Задание 3

## 3.1. Постановка задачи

Описать грамматику, позволяющую проводить трансляцию арифметических выражений, состоящих из вещественных чисел, знаков «+», «-», «\*», «/» и скобок из инфиксной записи в постфиксную.

## 3.2. Решение задачи, код программы

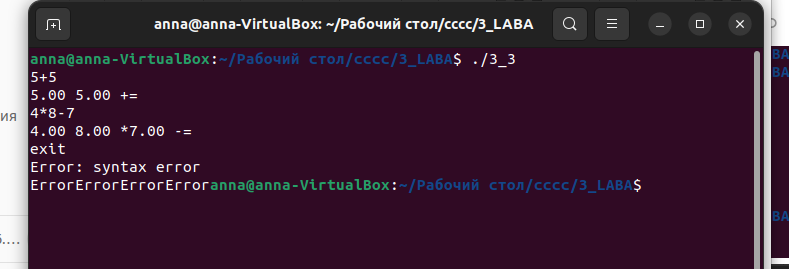
**Файл lab3\_3.l**

1. %{
2. #include <stdio.h>
3. #include <string.h>
4. #include "3\_3.tab.h"      // Подключение заголовочного файла "3\_3.tab.h"
5. **void** exitt ();  // Объявление функций
6. **void** error();
7. %}
9. %option noyywrap   // Опция для отключения обертки yywrap()
11. %%
13. [0-9]+(\.[0-9]+)? { yylval.number=atof(yytext); **return** NUM; } // Правило для чисел с плавающей точкой
14. [-+\*/(){}]        {**return** yytext[0]; }   // Правило для операторов и символов
15. [\n]              { **return** RAVNO;}      // Правило для символа новой строки
16. [ \t]  /\* skip whitespaces \*/                                    // Правило для пробелов и символов табуляции (пропуск)
17. ":exit"           {exitt();}               // Правило для команды ":exit"
18. .                 {error();}     // Правило для всех остальных символов (ошибка)
19. %%
21. **void** error(){
22. printf("Error");
23. }
25. **void** exitt(){
26. exit(0);
27. }

**Файл lab3\_3.y**

1. %{
2. #include <stdio.h>
3. #include <string.h>
4. #include "3\_3.tab.h"
6. **int** yylex(**void**);
7. **void** yyerror(**const** **char** \*str);
8. %}
10. %union {
11. **double** number;
12. }
14. %token <number> NUM   // Токен "<number>" для хранения значения числа
15. %token RAVNO         // Токен "RAVNO" для символа "="
17. %left "+"  "-"       // Левая ассоциативность для операторов "+" и "-"
18. %left "\*"  "/"       // Левая ассоциативность для операторов "\*" и "/"
19. %%
21. postf:
22. | postf exp RAVNO { printf("= \n"); } // Правило для постфиксной записи выражения (пока не определено)
23. ;
25. exp: vtorich
26. | exp '+' vtorich { printf("+"); }  // Правило для сложения
27. | exp '-' vtorich { printf("-"); }   // Правило для вычитания
28. ;
30. vtorich: pervich
31. | vtorich '\*' pervich { printf("\*"); }   // Правило для умножения
32. | vtorich '/'  pervich { printf("/"); }  // Правило для деления
33. ;
35. pervich: NUM { printf("%.2f ", $1); }    // Правило для числа
36. | '(' exp ')' {}                        // Правило для выражений в скобках
37. ;
39. %%
41. **void** yyerror(**const** **char** \*str){
42. fprintf(stderr,"Error: %s\n",str);
43. }
45. **int** main(**int** argc, **char** \*\*argv){
46. yyparse();                          // Запуск парсера
47. **return** 0;
48. }

## 3.3. Тестирование работы программы

****

**Объяснение:**

%union: Определяет объединение (union) для хранения значения числа.

%token <number> NUM: Определение токена <number> для хранения значения числа.

%token RAVNO: Определение токена RAVNO для символа "=".

%left "+" "-", %left "\*" "/": Определение левой ассоциативности для операторов "+" и "-" и операторов "\*" и "/".

postf: Правило для постфиксной записи выражения (пока не определено).

exp: Правило для выражения.

vtorich: Правило для вторичного выражения (операции умножения и деления).

pervich: Правило для первичного выражения (число или выражение в скобках).

void yyerror(const char \*str): Функция обработки ошибок, выводит сообщение об ошибке на стандартный поток ошибок .

int main(int argc, char \*\*argv): Основная функция программы, запускает парсер

Парсер – программный компонент, использующийся в комплиляторах для анализа структуры и синтаксиса текстового ввода с целью преобразования его во внутреннее представление данных.Парсер проверяет его на соответствие определенной грамматике и синтаксиссу.