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](#__RefHeading___Toc446_729700978)

[1.1. Постановка задачи 3](#__RefHeading___Toc448_729700978)

[1.2. Решение задачи, код программы 3](#__RefHeading___Toc450_729700978)

[1.3. Тестирование работы программы 4](#__RefHeading___Toc452_729700978)

[Задание 2 5](#__RefHeading___Toc454_729700978)

[2.1. Постановка задачи 5](#__RefHeading___Toc456_729700978)

[2.2. Решение задачи, код программы 5](#__RefHeading___Toc458_729700978)

[2.3. Тестирование работы программы 7](#__RefHeading___Toc460_729700978)

[Задание 3 8](#__RefHeading___Toc462_729700978)

[3.1. Постановка задачи 8](#__RefHeading___Toc464_729700978)

[3.2. Решение задачи, код программы 8](#__RefHeading___Toc466_729700978)

[3.3. Тестирование работы программы 9](#__RefHeading___Toc468_729700978)

[Задание 4 11](#__RefHeading___Toc470_729700978)

[4.1. Постановка задачи 11](#__RefHeading___Toc472_729700978)

[4.2. Решение задачи, код программы 12](#__RefHeading___Toc474_729700978)

[4.3. Тестирование работы программы 15](#__RefHeading___Toc476_729700978)

# 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>

void exit\_ctrlq();

void Errors();

%}

%%

[0-9]+ {printf("Integer number");}

[a-zA-Z] {printf("Letter");}

[a-zA-Z]+ {printf("Words");}

[0-9]+\.[0-9] {printf("Float/double number");}

"+" {printf("Summ");}

"-" {printf("Subtraction");}

"\*" {printf("Multiplication");}

"/" {printf("Division");}

[ \t]+ ;

. { Errors();}

":exit" {exit\_ctrlq();}

%%

void Errors(){

printf("Something went wrong");

}

void exit\_ctrlq(){

exit(0);

}

## 1.3. Тестирование работы программы

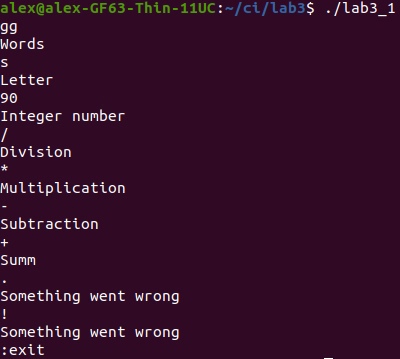


Рис.1.1 Тестирование программы lab3\_1

# 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 lab3\_2.l**

%{

#include <stdio.h>

#include <string.h>

#include "lab3\_2.tab.h"

void exit\_ctrlq();

void Errors();

%}

%%

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

"+" { return SUM; }

"-" { return SUB; }

"\*" { return MUL; }

"/" { return DIV; }

"(" { return OBRACE; }

")" { return EBRACE; }

= { return EOL; }

[ \t]+ ; /\* skip whitespaces \*/

. { Errors();}

":exit" {exit\_ctrlq();}

%%

int yywrap(){

return 1;

}

void Errors(){

printf("Something went wrong");

}

void exit\_ctrlq(){

exit(0);

}

**Файл lab3\_2.y**

%{

#include <stdio.h>

#include <string.h>

#include "lab3\_2.tab.h"

int yylex(void);

void yyerror(const char \*str);

int main(int argc, char \*\*argv);

%}

%token INTEGER

%token SUM SUB MUL DIV

%token OBRACE EBRACE

%token EOL

%token NUMBER

%%

calclist: /\* nothing \*/

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

;

exp: factor

| exp SUM factor { $$ = $1 + $3; }

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

;

factor: term

| factor MUL term { $$ = $1 \* $3; }

| factor DIV term { $$ = $1 / $3; }

;

term: INTEGER

| OBRACE exp EBRACE { $$ = $2; }

;

%%

void yyerror(const char \*str){

fprintf(stderr,"Error: %s\n",str);

}

**Файл lab3\_2.c**

#include <stdio.h>

#include "lab3\_2.tab.h"

int main(int argc, char \*\*argv){

yyparse();

return 0;

}

## 2.3. Тестирование работы программы

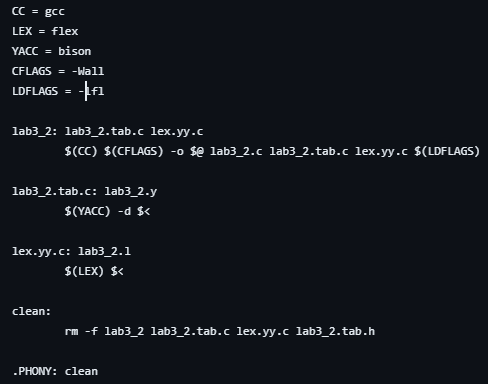


Рис.2.1 Make-file для lab3\_2



Рис. 2.2 Тестирование программы lab3\_2

# 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**

%{

#include <stdio.h>

#include "lab3\_3.tab.h"

void Errors();

%}

%option noyywrap

%%

[0-9]+(\.[0-9]+)? { yylval.number = atof(yytext); return NUMBER; }

[-+\*/(){}] { return yytext[0]; }

\n { return EOL; }

[ \t] { /\* ignore whitespace \*/ }

. { Errors(); }

%%

void Errors(){

printf("Something went wrong");

}

**Файл lab3\_3.y**

%{

#include <stdio.h>

#include <stdlib.h>

int yylex(void);

void yyerror(const char \*s);

%}

%union {

double number;

}

%token <number> NUMBER

%token EOL

%left '+' '-'

%left '\*' '/'

%%

comman: /\* nothing \*/

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

exp: term

| exp '+' term { printf("+ "); }

| exp '-' term { printf("- "); }

term: factor

| term '\*' factor { printf("\* "); }

| term '/' factor { printf("/ "); }

factor: NUMBER { printf("%.2f ", $1); }

| '(' exp ')' {}

%%

int main(void) {

yyparse();

return 0;

}

void yyerror(const char \*s) {

fprintf(stderr, "Error: %s\n", s);

}

## 3.3. Тестирование работы программы

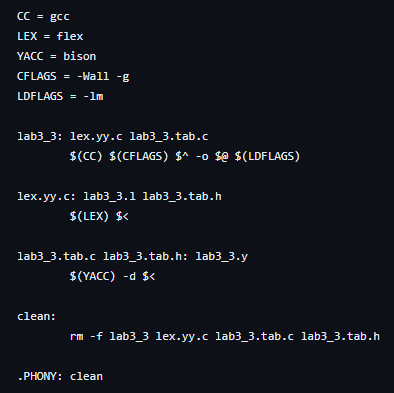


Рис.3.1 Make-file для lab3\_3

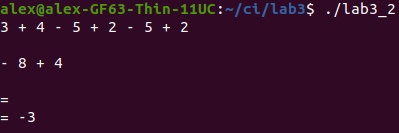


Рис.3.2 Тестирование программы lab3\_3

# Задание 4

## 4.1. Постановка задачи

Организовать вычисление выражений с использованием абстрактного синтаксического дерева. Для этого описать необходимые структуры и функции, осуществляющие построение абстрактного синтаксического дерева при разборе, а также его рекурсивных обход и вычисление арифметического выражения. В качестве основы грамматики использовать синтаксис арифметических выражений из лекции 3 (Слайд 9). При этом типы узлов дерева разбора примут вид:

%union {

struct ast \*a;

double d;

}

Для реализации необходимо написать код для прототипов функций:

/\* build an AST \*/

struct ast \*newast(int nodetype, struct ast \*l, struct ast \*r);

struct ast \*newnum(double d);

/\* evaluate an AST \*/

double eval(struct ast \*);

/\* delete and free an AST \*/

void treefree(struct ast \*);

Где структуры для указателей описаны как:

/\* nodes in the Abstract Syntax Tree \*/

struct ast {

int nodetype;

struct ast \*l;

struct ast \*r;

};

struct numval {

int nodetype; /\* type K \*/

double number;

};

## 4.2. Решение задачи, код программы

**Файл lab3\_4.l**

%{

#include <stdio.h>

#include <stdlib.h>

#include "lab3\_4.tab.h"

void Errors();

%}

%%

[0-9]+(\.[0-9]+)? { yylval.d = atof(yytext); return NUM; }

[(){}] { return yytext[0]; }

[\*] {return MUL; }

[/] {return DIV; }

[+] {return SUM; }

[-] {return SUB; }

\n {return yywrap(); }

[ \t] {/\* ignore whitespace \*/}

. { Errors(); }

%%

void Errors(){

printf("Something went wrong");

}

int yywrap() {

return 0;

}

**Файл lab3\_4.y**

%{

#include <stdio.h>

#include <stdlib.h>

struct ast {

int nodetype;

struct ast \*l;

struct ast \*r;

};

struct numvalue {

int nodetype;

double num;

};

struct ast \*newast(int nodetype, struct ast \*l, struct ast \*r);

struct ast \*newnum(double d);

double expcheck(struct ast \*);

void freeingmem(struct ast \*);

%}

%union {

struct ast \*a;

double d;

}

%token <d> NUM

%token SUM SUB MUL DIV

%left SUM SUB

%left MUL DIV

%type <a> expr

%%

program: expr { printf("Result: %lf\n", expcheck($1)); freeingmem($1); }

expr: NUM { $$ = newnum($1); }

| expr SUM expr { $$ = newast('+', $1, $3); }

| expr SUB expr { $$ = newast('-', $1, $3); }

| expr MUL expr { $$ = newast('\*', $1, $3); }

| expr DIV expr { $$ = newast('/', $1, $3); }

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

%%

struct ast \*newast(int nodetype, struct ast \*l, struct ast \*r) {

struct ast \*node = (struct ast \*)malloc(sizeof(struct ast));

if (!node) {

fprintf(stderr, "Out of memory\n");

exit(1);

}

node->nodetype = nodetype;

node->l = l;

node->r = r;

return node;

}

struct ast \*newnum(double d) {

struct numvalue \*node = (struct numvalue \*)malloc(sizeof(struct numvalue));

if (!node) {

fprintf(stderr, "Out of memory\n");

exit(1);

}

node->nodetype = 'Cons';

node->num = d;

return (struct ast \*)node;

}

/\* evaluate an AST \*/

double expcheck(struct ast \*node) {

double result;

if (!node) {

fprintf(stderr, "Error in checking expression\n");

exit(1);

}

switch (node->nodetype) {

case 'Cons':

result = ((struct numvalue \*)node)->num;

break;

case '+':

result = expcheck(node->l) + expcheck(node->r);

break;

case '-':

result = expcheck(node->l) - expcheck(node->r);

break;

case '\*':

result = expcheck(node->l) \* expcheck(node->r);

break;

case '/':

result = expcheck(node->l) / expcheck(node->r);

break;

default:

fprintf(stderr, "Unknown node type: %d\n", node->nodetype);

exit(1);

}

return result;

}

void freeingmem(struct ast \*node) {

if (!node) return;

if (node->nodetype == 'Cons') {

free((struct numvalue \*)node);

}

else {

freeingmem(node->l);

freeingmem(node->r);

free(node);

}

}

int main() {

yyparse();

return 0;

}

void yyerror(const char \*s) {

fprintf(stderr, "Error: %s\n", s);

}

## 4.3. Тестирование работы программы

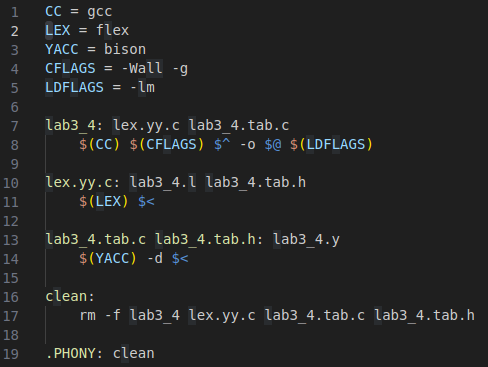


Рис.4.1 Make-file для lab3\_4



Рис.4.2 Тестирование программы lab3\_4