Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский политехнический университет»**

Факультет: Прикладной математики и механики

Кафедра: Вычислительной математики, механики и биомеханики

Направление: 09.03.02 «Информационные системы и технологии»

Программа: «Цифровые технологии и интеллектуальные системы управления»

У Т В Е Р Ж Д А Ю

**Зав. кафедрой ВММБ**

\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Ю. Столбов

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ г.

***ЗАДАНИЕ***

***НА КУРСОВУЮ РАБОТУ***

***по дисциплине***

**«Языки программирования и методы трансляции»**

Язов Михаил Максимович ИСТ-22-1б

(фамилия, имя, отчество студента; группа)

1. **Тема курсовой работы**

Язык управления роботом

2. **Срок сдачи студентом отчета:**

ХХ.XX.202X

3. **Содержание отчета:**

В курсовой работе были рассмотрены лексический анализатор flex и синтаксический

анализатор bison , изучена их грамматика, и с помощью данных инструментов был

разработан язык программирования для управления роботом, который способен двигаться

и выполнять 3 функции взаимодействия с окружающей средой, также проведен анализ

ограничений и возможностей созданного ЯП.

С.Е. Батин

Руководитель курсовой работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (расшифровка)

М.М. Язов

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (расшифровка)

Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский**

**политехнический университет»**

Факультет: Прикладной математики и механики

Кафедра: Вычислительной математики, механики и биомеханики

Направление: 09.03.02 «Информационные системы и технологии»

Программа: «Цифровые технологии и интеллектуальные системы управления»

**КУРСОВАЯ РАБОТА**

по дисциплине

**«Языки программирования и методы трансляции»**

Тема: **«Язык управления роботом»**

Выполнил:

ИСТ-22-1б

студент гр.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Язов М.М.

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

(Ф.И.О.)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*(подпись, дата)*

Принял:

Доцент каф. ВММБ, к.т.н. С.Е. Батин

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(должность, ФИО)*

*\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(оценка) (подпись)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(дата)*

**Пермь, 202X**

Содержание

[(подпись) (расшифровка) 1](#_Toc155719198)

[1](#_Toc155719199)

[Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 1](#_Toc155719200)

[Введение 4](#_Toc155719201)

[1. Лексический анализатор 5](#_Toc155719202)

[2. Синтаксический анализатор 7](#_Toc155719203)

[3. Описание языка программирования 9](#_Toc155719204)

[3.1. Грамматика 9](#_Toc155719205)

[3.2. Необходимые дополнения и запуск программы 10](#_Toc155719206)

[3.3. Пример работы ЯП 10](#_Toc155719207)

[4. Анализ возможностей и ограничений 13](#_Toc155719208)

[Заключение 14](#_Toc155719209)

[Библиографический список 15](#_Toc155719210)

[Приложения 16](#_Toc155719211)

[Приложение 1. Код flex 16](#_Toc155719212)

[Приложение 2. Код bison 17](#_Toc155719213)

# Введение

Flex и bison — это средства, которые созданы для разработчиков компиляторов и интерпретаторов, но они также полезны для разных приложений, которые могут быть интересны разработчикам, не занимающимся компиляторами. Любое приложение, которое анализирует свой ввод или использует свой язык ввода или команд, подходит для flex и bison, поэтому эти средства очень удобны для написания курсовой работы. К тому же, они помогают быстро делать прототипы приложений, легко изменять и просто обслуживать программы. Вот несколько примеров, для которых люди применяли flex и bison или их предки lex и yacc:

* Калькулятор
* Препроцессоры набора для математических уравнений и сложных изображений
* Многие другие "языки, специфичные для конкретной области", предназначенные для определенных приложений
* PCC, портативный компилятор языка C, используемый во многих системах Unix
* Сам Flex - транслятор языка баз данных SQL.

Курсовая работа направлена на создание языка программирования с помощью вышеупомянутых инструментов и ЯП C, который будет решать набор простых практических задач

Цель: написать язык управления роботом. Робот может двигаться в произвольном направлении, выполнять заданное количество действий в зависимости от условий внешней среды (не менее 3-х), считать количество выполненных действий. Транслятор должен записать последовательность действий робота при передаче ему программы.

Задачи: изучить грамматику flex и bison, понять, как строится дерево разбора, происходят синтаксический и лексический анализы.

# 1. Лексический анализатор

Для обработки структурированного текста люди обычно используют два этапа. На первом этапе, лексическом (он же сканирование), ввод разбивается на значимые фрагменты символов. Второй этап, синтаксический разбор, группирует отсканированные фрагменты, следуя потенциально рекурсивным правилам. Однако такой хороший инструмент лексики, как flex, может быть полезен и сам по себе, даже если он не работает в паре с синтаксическим анализатором. [1]

Сканеры обычно работают, отыскивая шаблоны символов во входных данных. Например, в программе на языке Си целочисленная константа — это строка из одной или нескольких цифр, имя переменной — это буква, за которой следует ноль или более букв или цифр, а различные операторы — это отдельные символы или пары символов. Прямым способом описания этих шаблонов являются регулярные выражения (regular expression), которые часто сокращают до regex или regexp. Программа flex в основном состоит из списка regexp и инструкций о том, что делать, когда входные данные соответствуют любому из них, называемых действиями. Сгенерированный flex-сканер считывает входные данные, сопоставляя их со всеми regexps и выполняя соответствующее действие при каждом совпадении. Flex переводит все regexps в эффективную внутреннюю форму, которая позволяет ему сопоставлять входные данные со всеми шаблонами одновременно, поэтому он работает так же быстро для 100 шаблонов, как и для одного. [2]

Flex программа состоит из трех разделов, разделенных %% строками [2]

**%{**

**int chars = 0; // Счетчик символов**

**int words = 0; // Счетчик слов**

**int lines = 0; // Счетчик строк**

**%}**

**%%**

**[a-zA-Z]+ { words++; chars += strlen(yytext); } // При обнаружении слова увеличиваем счетчики слов и символов**

**\n { chars++; lines++; } // При обнаружении новой строки увеличиваем счетчики строк и символов**

**. { chars++; } // При обнаружении любого другого символа увеличиваем счетчик символов**

**%%**

**main()**

**{**

**yylex(); // Запуск лексического анализатора**

**printf("%8d%8d%8d\n", lines, words, chars); // Вывод результатов подсчета строк, слов и символов**

**}**

Первый раздел содержит объявления и настройки опций. Вторая секция — это список шаблонов и действий, а третья - код на языке Си, который копируется в сгенерированный сканер, обычно это небольшие программы, связанные с кодом в действиях.

В секции деклараций код внутри %{ и %} дословно копируется в начало сгенерированного исходного файла на языке Си. В данном случае он просто устанавливает переменные для строк, слов и символов.

Во втором разделе каждый шаблон находится в начале строки, за ним следует код на языке Си, который будет выполняться при совпадении шаблона. Код на языке Си может быть одним оператором или, возможно, многострочным блоком в скобках { }. (Каждый шаблон должен начинаться с начала строки, так как flex считает любую строку, начинающуюся с пробельных символов, кодом для копирования в сгенерированную программу на языке C).

Код на Си в конце (его может даже вовсе не быть) — это основная программа, которая вызывает yylex(), имя, которое flex дает подпрограмме сканера, а затем печатает результаты. В отсутствие каких-либо других настроек сканер считывает данные со стандартного ввода.

Подытоживая, всё, что делает лексический анализатор – это анализирует входной поток и на каждый найденный шаблон выполняет определенные действия (чаще всего – передача токенов).

# 2. Синтаксический анализатор

Задача синтаксического анализатора - выяснить отношения между входными лексемами, т.е. определить, как с ними нужно действовать. [1] Общим способом отображения таких отношений является дерево разбора. Например, в соответствии с обычными правилами арифметики, арифметическое выражение 1 \* 2 + 3 \* 4 + 5 будет иметь дерево разбора в виде (рис.2).

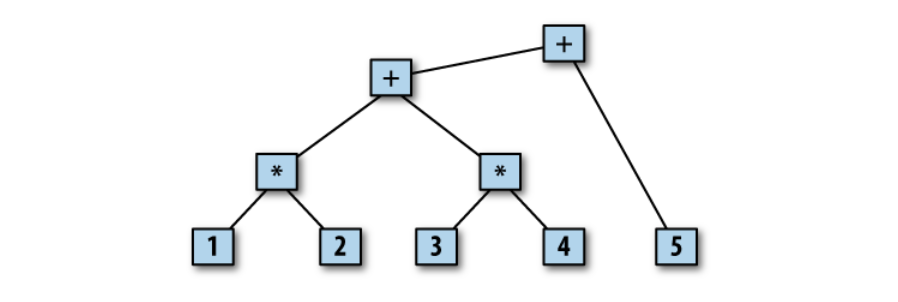


Рисунок 2. Пример дерева разбора

Для того чтобы написать синтаксический анализатор, нам нужно каким-то образом описать правила, которые он использует для преобразования последовательности лексем в дерево разбора. Это делается с помощью варианта Backus Naur Form (BNF). Эта техника, разработанная Джоном Бакусом и Питером Науром, была использована для описания языка Algol 60. Грамматика БНФ может использоваться для выражения контекстно-свободных языков. Большинство конструкций в современных языках программирования могут быть представлены в БНФ. Например, грамматика для выражения, которое умножает и складывает числа:

E -> E + E

E -> E \* E

E -> id

Было определено три продукции. Термы, которые появляются в левой части (lhs) продукции, например, E (выражение), являются нетерминальными. Термы, такие как id (идентификатор), являются терминалами (лексемы, возвращаемые lex) и появляются только в правой части (rhs) продукции. Эта грамматика определяет, что выражение может быть суммой двух выражений, произведением двух выражений, или идентификатором.

Программы Bison имеют ту же структуру, что и программы flex: декларации, правила и код на Си [2]

**%{**

# include <stdio.h>

**%}**

**%**token NUMBER

**%**token ADD SUB MUL DIV ABS

**%**token OP CP

**%**token EOL

**%%**

calclist**:**

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

**|** calclist EOL **{** printf**(**"> "**);** **}**

**;**

exp**:** factor

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

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

**|** exp ABS factor **{** $$ **=** $1 **|** $3**;** **}**

**;**

factor**:** term

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

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

**;**

term**:** NUMBER

**|** ABS term **{** $$ **=** $2 **>=** 0**?** $2 **:** **-** $2**;** **}**

**|** OP exp CP **{** $$ **=** $2**;** **}**

**;**

**%%**

main**()**

**{**

printf**(**"> "**);**

yyparse**();**

**}**

yyerror**(**char **\***s**)**

**{**

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

**}**

Декларации здесь включают Си-код, который нужно скопировать в начало генерируемого синтаксического анализатора, опять же заключенный в %{ и %}. За ними следуют декларации токенов %token, указывающие bison имена символов в синтаксическом анализаторе, которые являются токенами. По соглашению, токены имеют имена в верхнем регистре, хотя bison не требует этого. Любые символы, не объявленные как лексемы, должны появляться слева хотя бы от одного правила в программе. (Если символ не является лексемой и не появляется в левой части правила, он подобен переменной без ссылки в программе на языке Си. Это ничему не вредит, но, вероятно, означает, что программист допустил ошибку.)Второй раздел содержит правила в упрощенной БНФ. Поскольку границы строк не имеют значения, точка с запятой отмечает конец правила. Опять же, как и в flex, код действия на языке C заключен в скобки в конце каждого правила.[2]

Если говорить коротко: в синтаксическом анализаторе происходит разбор токенов набором синтаксических правил.

# 3. Описание языка программирования

## 3.1. Грамматика

1. Передвижение робота на 4 стороны (вверх, вниз, влево, вправо):

direction N times;

* direction: up/down/left/right
* N – любое натуральное число

2. Действия (разнести, раскрасить, дропнуть (кинуть, положить)):

action rock direction;

* action: destroy/makeup/drop
* direction: up/down/left/right

**Обязательно после каждого движения/действия ставить точку с запятой.**

Обозначим передвижение и действие, как statement.

3. Ветвление if/else:

if (condition){

statement;

}

else{

statement;

}

stateman;

* condition: direction free
* direction: up/down/left/right

## 3.2. Необходимые дополнения и запуск программы

Для функционирования языка нужно создать файлы:

* programm.txt – здесь записывается код
* enviroment.txt – отсюда считывается окружение, нужно записывать так:

robot

1,2

rock

1,3

1,6

3,4

4,6

* res.txt – сюда записываются результаты:

1. Робот разнес камень в точке (1,3)

2. Робот переехал в точку (1,5)

3. Робот переехал в точку (3,5)

4. Робот дропнул камень в точке (2,5)

5. Робот разукрасил камень (сделал его крутым) в точке(2,5)

6. Робот переехал в точку (4,5)

7. Робот разнес камень в точке (4,6)

## 3.3. Пример работы ЯП

if (up free){

up 1 times;

}

else{

destroy rock up;

}

up 3 times;

right 2 times;

drop rock left;

makeup rock left;

right 1 times;

destroy rock up;

if (left free){

left 1 times;

}

else{

destroy rock left;

}

makeup rock down;

destroy rock left;

down 1 times;

Задано окружение в environment.txt:

robot

1,2

rock

1,3

1,6

3,4

4,6

И, наконец, результаты работы программы в res.txt:

1. Робот разнес камень в точке (1,3)

2. Робот переехал в точку (1,5)

3. Робот переехал в точку (3,5)

4. Робот дропнул камень в точке (2,5)

5. Робот разукрасил камень (сделал его крутым) в точке(2,5)

6. Робот переехал в точку (4,5)

7. Робот разнес камень в точке (4,6)

8. Робот переехал в точку (3,5)

9. Робот разукрасил камень (сделал его крутым) в точке(3,4)

10. Робот разнес камень в точке (2,5)

11. Ошибка: робот не может переехать в точку (3,4) из-за камня в точке (3,4)

# 4. Анализ возможностей и ограничений

Все что не реализовано в данном языке программирования – это возможность определять свои переменные, в которых можно сохранять значения целого типа, чтобы не писать одно и то же число несколько раз. Но это не сильно влияет на удобство использования, так как целые значения нужны только для указания количества шагов, которые должен выполнить робот. А также здесь не реализованы циклы, к сожалению у меня не получилось их нормально реализовать. Есть что еще развивать, и улучшать, и над чем работать.

# Заключение

В этой курсовой работе создан язык программирования робота, который может решать простые практические задачи взаимодействия с окружающим миром. Описаны синтаксис языка, средства его разработки, а также показаны исходные коды лексического и синтаксического анализатора. Сделан анализ возможностей и ограничений, связанных с применением языка. Рассмотрено формирование дерева разбора. Следовательно, задачи решены, цель достигнута.

# Библиографический список

1. Свердлов З.С. Языки программирования и методы трансляции: учебное пособие. – 2-ое изд.: Издательство «Лань», 2019. – 564 с.

2. John R. Levine. flex & bison. – Sebastobol: O’Reilly Media, Inc., 2009. – 263 p.

# Приложения

## Приложение 1. Код flex

%option yylineno

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "RobotControl.tab.h"

%}

%%

[ \t\n]+ ; // Пропуск пробелов и символов новой строки

"if" { return IF; } // Обнаружено ключевое слово "if"

"else" { return ELSE; } // Обнаружено ключевое слово "else"

"destroy" { return DES; } // Обнаружена команда "destroy"

"makeup" { return M; } // Обнаружена команда "makeup"

"drop" { return D; } // Обнаружена команда "drop"

"(" { return OB; } // Обнаружен символ "("

")" { return CB; } // Обнаружен символ ")"

"{" { return FOB; } // Обнаружен символ "{"

"}" { return FCB; } // Обнаружен символ "}"

";" { return SEMICOLON; } // Обнаружена точка с запятой

[0-9]+ { yylval.i = atoi(yytext); return STEPS; } // Обнаружено целое число (шаги)

"left" { return LEFT; } // Обнаружено ключевое слово "left"

"right" { return RIGHT; } // Обнаружено ключевое слово "right"

"up" { return UP; } // Обнаружено ключевое слово "up"

"down" { return DOWN; } // Обнаружено ключевое слово "down"

"rock" { return ROCK; } // Обнаружено ключевое слово "rock"

"times" { return TIMES; } // Обнаружено ключевое слово "times"

"free" { return FREE; } // Обнаружено ключевое слово "free"

. { fprintf(stderr, "Ошибка: неожиданный символ %s в строке %d\n", yytext, yylineno); exit(1); } // Обнаружен неожиданный символ

%%

## Приложение 2. Код bison

%{

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "RobotControl.tab.h"

#define n 2 // количество столбцов в двумерном массиве и ячеек в одномерном массиве

// Объявление функции лексического анализатора

int yylex();

// Функция yywrap - оборачивание лексера, возвращает 1 для указания конца ввода

char yywrap() {

return 1;

}

// Внешняя переменная для отслеживания номера текущей строки в исходном коде

extern int yylineno;

// Функция обработки ошибок, принимает строку с сообщением об ошибке

void yyerror(char \*str);

// Внешние переменные для управления входным и выходным потоками лексера/парсера

extern FILE\* yyin;

extern FILE\* yyout;

// счетчик всех действий и перемещений робота

int counter;

// координаты робота

int robot[n];

// координаты камней

int numberOfRowsFir;

int \*rock;

// флаг крутых камней

int \*makeUpedRock;

// чтение файла окружения

void readEnvironmentFile(FILE\* enviromentFile);

// Структура для узлов в абстрактном синтаксическом дереве (AST)

struct ast {

int nodetype; // Тип узла в дереве

struct ast \*l; // Левое поддерево

struct ast \*r; // Правое поддерево

};

// Структура для узлов, представляющих числовые значения в AST

struct numval {

int nodetype; // Тип K (число)

int number; // Значение числа

};

// Структура для узлов, представляющих управляющие конструкции (if или while) в AST

struct flow {

int nodetype; // Тип I (if)

struct ast \*cond; // Условие

struct ast \*tl; // Тогда или список действий

struct ast \*el; // Необязательный список иначе (else)

};

// построение AST

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

struct ast \*newNum(int i);

struct ast \*newFlow(int nodetype, struct ast \*cond, struct ast \*tl, struct ast \*el);

// оценка AST

int eval(struct ast \*);

// оценка перемещений

void evaluateMovements(int value, int checkDirection);

// оценка действий

void evalActions(int checkAction, int checkDirection);

// проверка наличия объектов вокруг

int defineEnvironment(int helperArray[], int checkDirection, int flag);

// перезапись массива

void overwriteArray(int helperArray[], int \*array, int sizeArray, int flag);

// удаление и освобождение AST

void freeAstTree(struct ast \*);

%}

%union{

struct ast \*a;

int i;

}

%token <i> STEPS

%token LEFT RIGHT UP DOWN ROCK TIMES FREE DES M D

%token IF ELSE OB CB FOB FCB SEMICOLON

%type <a> body condition elsee statement direction action steps

%%

commands:

| commands body { eval($2); freeAstTree($2); }

;

body: IF OB condition CB FOB body FCB elsee { $$ = newFlow('I', $3, $6, $8); }

| IF OB condition CB FOB body FCB { $$ = newFlow('I', $3, $6, NULL); }

| statement SEMICOLON { $$ = newAst('s', $1, NULL); }

;

elsee: ELSE FOB body FCB { $$ = newAst('e', $3, NULL); }

;

condition: direction FREE { $$ = newAst('F', $1, NULL); }

;

statement: direction steps TIMES { $$ = newAst('T', $1, $2); }

| action ROCK direction { $$ = newAst('a', $1, $3); }

;

direction: UP { $$ = newAst('u', NULL, NULL); }

| DOWN { $$ = newAst('d', NULL, NULL); }

| LEFT { $$ = newAst('l', NULL, NULL); }

| RIGHT { $$ = newAst('r', NULL, NULL); }

;

action: DES { $$ = newAst('DES', NULL, NULL); }

| M { $$ = newAst('M', NULL, NULL); }

| D { $$ = newAst('D', NULL, NULL); }

;

steps: STEPS { $$ = newNum($1); }

;

%%

void yyerror(char \*str){

fprintf(yyout ,"%d. Ошибка: %s в строке %d\n", counter, str, yylineno);

exit(1);

}

int main() {

char \*programmFileName = "programm.txt";

FILE\* programmFile = fopen(programmFileName, "r");

// Проверка успешного открытия файла программы

if (programmFile == NULL) {

fprintf(yyout, "%d. Невозможно открыть файл %s", counter, programmFileName);

exit(1);

}

char \*enviromentFileName = "enviroment.txt";

FILE\* enviromentFile = fopen(enviromentFileName, "r");

// Проверка успешного открытия файла окружения

if (enviromentFile == NULL) {

fprintf(yyout, "%d. Невозможно открыть файл %s", counter, enviromentFileName);

exit(1);

}

char \*logFileName = "res.txt";

FILE\* logFile = fopen(logFileName, "w");

// Настройка ввода и вывода для анализатора

yyin = programmFile;

yyout = logFile;

// Чтение файла окружения

readEnvironmentFile(enviromentFile);

// Инициализация массива makeUpedRock

// 0 - не Крутой камень, 1 - Крутой камень

makeUpedRock = (int\*) malloc(numberOfRowsFir \* sizeof(int));

for (int i = 0; i < numberOfRowsFir; i++) {

\*(makeUpedRock + i) = 0;

}

// Начало синтаксического анализа

yyparse();

// Закрытие файлов

fclose(yyin);

fclose(enviromentFile);

fclose(yyout);

// Освобождение выделенной памяти

free(rock);

free(makeUpedRock);

return 0;

}

// Функция чтения данных из файла окружения

void readEnvironmentFile(FILE\* enviromentFile){

int numberOfRows = 0; // Переменная для подсчета строк в файле

fseek(enviromentFile, 0, SEEK\_SET); // Устанавливаем указатель файла в начало

while (!feof(enviromentFile)){

if (fgetc(enviromentFile) == '\n'){

numberOfRows++; // Подсчитываем строки в файле

}

}

numberOfRowsFir = numberOfRows - 2; // Количество строк для массива rock (за исключением строки с "robot" и строки с размерами)

rock = (int\*) malloc(numberOfRowsFir \* n \* sizeof(int)); // Выделяем память под массив rock

char \*bufferName; // Буфер для имени объекта (robot, rock)

fseek(enviromentFile, 0, SEEK\_SET); // Устанавливаем указатель файла в начало

while (!feof(enviromentFile)){

fscanf(enviromentFile, "%s", bufferName); // Считываем имя объекта из файла

char \*robotName = "robot";

if (strcmp(bufferName, robotName) == 0){ // Если объект - робот

for (int i = 0; i < n; i++){

fscanf(enviromentFile, "%d,", &robot[i]); // Считываем координаты робота

}

}

else{

for (int i = 0; i < numberOfRowsFir; i++){

for (int j = 0; j < n; j++){

fscanf(enviromentFile, "%d,", (rock + i \* n + j)); // Считываем данные о rock

}

}

}

}

}

// Создание нового узла AST с указанным типом и дочерними узлами

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

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

if (!a){

yyerror("Оut of space, robot go away");

exit(0);

}

a->nodetype = nodetype;

a->l = l;

a->r = r;

return a;

}

// Создание нового узла AST с числовым значением

struct ast \*newNum(int i){

struct numval \*a = malloc(sizeof(struct numval));

if (!a){

yyerror("Оut of space, robot go away");

exit(0);

}

a->nodetype = 'K';

a->number = i;

return (struct ast \*)a;

}

// Создание нового узла AST для оператора IF с условием, true-веткой и false-веткой

struct ast \*newFlow(int nodetype, struct ast \*cond, struct ast \*tl, struct ast \*el){

struct flow \*a = malloc(sizeof(struct flow));

if(!a) {

yyerror("Оut of space, robot go away");

exit(0);

}

a->nodetype = nodetype;

a->cond = cond;

a->tl = tl;

a->el = el;

return (struct ast \*)a;

}

/\* Типы узлов:

\* s statement (оператор)

\* e else (иначе)

\* F direction free (свободное направление)

\* T direction step TIMES (направление и шаг во времени)

\* a action ROCK (действие - Залить краской камень, сделать его крутым)

\* u up (вверх)

\* d down (вниз)

\* l left (влево)

\* r right (вправо)

\* DES destroy (разнести)

\* M makeup(сделать крутым)

\* D drop rock (дропнуть камень)

\* I IF statement (условный оператор)

\*/

int eval(struct ast \*a){

// Значение, возвращаемое функцией

int value;

// Переменные для проверки направления и действия

int checkDirection;

int checkAction;

// Флаг для оценки окружения (для использования в функции defineEnvironment)

int flag = -2;

// Вспомогательный массив для оценки окружения

int helperArray[n];

// Определяем тип узла AST и выполняем соответствующие действия

switch(a->nodetype){

case 'K':

value = ((struct numval \*)a)->number; // просто число

break;

case 's':

eval(a->l); // оператор (statement) - выполняем его

break;

case 'e':

eval(a->l); // иначе (else) - выполняем его

break;

case 'T':

counter++;

value = eval(a->r); // значение шага по времени

checkDirection = eval(a->l); // оценка направления

evaluateMovements(value, checkDirection); // выполнение движения в заданном направлении

break;

case 'a':

counter++;

checkAction = eval(a->l); // оценка действия

checkDirection = eval(a->r); // оценка направления

evalActions(checkAction, checkDirection); // выполнение действия в заданном направлении

break;

case 'DES':

value = 'DES'; // снести

break;

case 'M':

value = 'M'; // сделать крутым

break;

case 'D':

value = 'D'; // дропнуть камень

break;

case 'F':

checkDirection = eval(a->l); // оценка направления

value = defineEnvironment(helperArray, checkDirection, flag); // оценка окружения в заданном направлении

break;

case 'u':

value = 0; // вверх

break;

case 'd':

value = 1; // вниз

break;

case 'r':

value = 2; // вправо

break;

case 'l':

value = 3; // влево

break;

case 'I':

if(eval(((struct flow \*)a)->cond) == 0) { // проверка условия - ветка true

if(((struct flow \*)a)->tl) {

eval(((struct flow \*)a)->tl); // выполнение true-ветки

}

else{

value = -1; // значение по умолчанию

}

}

else { // ветка false

if(((struct flow \*)a)->el) {

eval(((struct flow \*)a)->el); // выполнение false-ветки

}

else {

value = -1; // значение по умолчанию

}

}

break;

}

return value; // возвращаем значение

}

// Функция evaluateMovements оценивает перемещения робота в заданном направлении и на определенное расстояние.

void evaluateMovements(int value, int checkDirection){

// Для корректного указания позиции при отображении ошибки, создаем временный массив для хранения текущих координат робота.

int tempRobot[2];

for(int i = 0; i < n; i++){

tempRobot[i] = robot[i];

}

// Создаем вспомогательный массив для оценки окружения робота.

int helperArray[n];

// Флаг -1 используется для оценки, есть ли камень вокруг робота.

int flag = -1;

// Перемещаем робота на заданное расстояние в выбранном направлении.

for(int i = 0; i < value; i++){

switch(defineEnvironment(helperArray, checkDirection, flag)){

// Если вокруг робота есть камень, генерируем ошибку и завершаем программу.

case 1:

if(checkDirection == 0){ // вверх

fprintf(yyout, "%d. Ошибка: робот не может переехать в точку (%d,%d) из-за камня в точке (%d,%d)\n", counter, tempRobot[0], tempRobot[1] + value, helperArray[0], helperArray[1]);

exit(1);

}

if(checkDirection == 1){ // вниз

fprintf(yyout, "%d. Ошибка: робот не может переехать в точку (%d,%d) из-за камня в точке (%d,%d)\n", counter, tempRobot[0], tempRobot[1] - value, helperArray[0], helperArray[1]);

exit(1);

}

if(checkDirection == 2){ // вправо

fprintf(yyout, "%d. Ошибка: робот не может переехать в точку (%d,%d) из-за камня в точке (%d,%d)\n", counter, tempRobot[0] + value, tempRobot[1], helperArray[0], helperArray[1]);

exit(1);

}

if(checkDirection == 3){ // влево

fprintf(yyout, "%d. Ошибка: робот не может переехать в точку (%d,%d) из-за камня в точке (%d,%d)\n", counter, tempRobot[0] - value, tempRobot[1], helperArray[0], helperArray[1]);

exit(1);

}

break;

// Если вокруг робота нет ели, перемещаем его в выбранном направлении.

case 0:

if(checkDirection == 0){ // вверх

robot[1] += 1;

}

if(checkDirection == 1){ // вниз

robot[1] -= 1;

}

if(checkDirection == 2){ // вправо

robot[0] += 1;

}

if(checkDirection == 3){ // влево

robot[0] -= 1;

}

break;

}

}

// Выводим сообщение о том, что робот переместился в новую позицию.

fprintf(yyout, "%d. Робот переехал в точку (%d,%d)\n", counter, robot[0], robot[1]);

}

void evalActions(int checkAction, int checkDirection){

// строка для удаления или добавления в defineEnvironment

int helperArray[n];

// для указания случая 'DES' или 'M' или 'D'

int flag;

switch(checkAction){

case 'DES': // разнести камень, фактически, строка удаляется из массива

flag = 0;

if(defineEnvironment(helperArray, checkDirection, flag) == 0){

fprintf(yyout, "%d, Ошибкааа: вы пытаетесь разнести камень, которого нет в точке (%d,%d)\n", counter, helperArray[0], helperArray[1]);

exit(1);

}

overwriteArray(helperArray, rock, numberOfRowsFir, flag);

fprintf(yyout, "%d. Робот разнес камень в точке (%d,%d)\n", counter, helperArray[0], helperArray[1]);

break;

case 'M': // разукрасить камень, 0 заменяется на 1 в массиве makeUpedRock

flag = 1;

if(defineEnvironment(helperArray, checkDirection, flag) == 0){

fprintf(yyout, "%d. Ошибкааа: вы пытаетесь сделать камень крутым, которого нет в точке (%d,%d)\n", counter, helperArray[0], helperArray[1]);

exit(1);

}

if(defineEnvironment(helperArray, checkDirection, flag) == 2){

fprintf(yyout, "%d. Камень уже крутой в точке (%d,%d)\n", counter, helperArray[0], helperArray[1]);

}

else{

overwriteArray(helperArray, rock, numberOfRowsFir, flag);

fprintf(yyout, "%d. Робот разукрасил камень (сделал его крутым) в точке(%d,%d)\n", counter, helperArray[0], helperArray[1]);

}

break;

case 'D': // дропнуть камень, новые координаты камня в массиве rock

flag = 2;

if(defineEnvironment(helperArray, checkDirection, flag) == 1){

fprintf(yyout, "%d. Ошибкааа: в точке (%d,%d) уже есть камень (булыжник)\n", counter, helperArray[0], helperArray[1]);

exit(1);

}

overwriteArray(helperArray, rock, numberOfRowsFir, flag);

fprintf(yyout, "%d. Робот дропнул камень в точке (%d,%d)\n", counter, helperArray[0], helperArray[1]);

break;

}

}

// Функция определения окружения по направлению и координатам робота

// Возвращает:

// 0 - если в окружении нет камня

// 1 - если в окружении есть камень

// 2 - если робот столкнулся с с камнем и прекратил выполнение цикла

int defineEnvironment(int helperArray[], int checkDirection, int flag) {

for(int k = 0; k < numberOfRowsFir; k++) {

// Координаты текущего камня

int xFir = \*(rock + k \* n + 0);

int yFir = \*(rock + k \* n + 1);

switch(checkDirection) {

// Проверка направления вверх

case 0:

if (robot[0] == xFir && robot[1] + 1 == yFir) {

helperArray[0] = xFir;

helperArray[1] = yFir;

// Проверка флага и статуса ели

if(flag == 1 && \*(makeUpedRock + k) == 1) {

return 2; // Робот столкнулся с камнем и прекратил выполнение цикла

}

return 1; // В окружении есть камень

} else {

helperArray[0] = robot[0];

helperArray[1] = robot[1] + 1;

}

break;

// Проверка направления вниз

case 1:

if(robot[0] == xFir && robot[1] - 1 == yFir) {

helperArray[0] = xFir;

helperArray[1] = yFir;

if(flag == 1 && \*(makeUpedRock + k) == 1) {

return 2;

}

return 1;

} else {

helperArray[0] = robot[0];

helperArray[1] = robot[1] - 1;

}

break;

// Проверка направления вправо

case 2:

if(robot[0] + 1 == xFir && robot[1] == yFir) {

helperArray[0] = xFir;

helperArray[1] = yFir;

if(flag == 1 && \*(makeUpedRock + k) == 1) {

return 2;

}

return 1;

} else {

helperArray[0] = robot[0] + 1;

helperArray[1] = robot[1];

}

break;

// Проверка направления влево

case 3:

if(robot[0] - 1 == xFir && robot[1] == yFir ) {

helperArray[0] = xFir;

helperArray[1] = yFir;

if(flag == 1 && \*(makeUpedRock + k) == 1) {

return 2;

}

return 1;

} else {

helperArray[0] = robot[0] - 1;

helperArray[1] = robot[1];

}

break;

}

}

return 0; // В окружении нет камня

}

// Функция перезаписи массива с удалением элемента или обновлением флага

void overwriteArray(int helperArray[], int \*array, int sizeArray, int flag) {

int xArray, yArray;

int \*tempArray = NULL;

tempArray = (int\*) realloc(tempArray, sizeArray \* n \* sizeof(int));

int \*tempXMasFir = NULL;

tempXMasFir = (int\*) realloc(tempXMasFir, sizeArray \* sizeof(int));

int j = 0; // Нумерация для новых координат ели

if(flag == 0) { // Удаление элемента из массива

for(int i = 0; i < sizeArray; i++) {

// Координаты камня

xArray = \*(array + i \* n + 0);

yArray = \*(array + i \* n + 1);

if (helperArray[0] != xArray || helperArray[1] != yArray) {

\*(tempArray + j \* n + 0) = xArray;

\*(tempArray + j \* n + 1) = yArray;

j++;

\*(tempXMasFir + j) = \*(makeUpedRock + i);

}

}

free(rock);

numberOfRowsFir -= 1;

rock = tempArray;

free(makeUpedRock);

makeUpedRock = tempXMasFir;

}

if(flag == 1) { // Обновление флага камня

for(int i = 0; i < sizeArray; i++) {

xArray = \*(array + i \* n + 0);

yArray = \*(array + i \* n + 1);

if(helperArray[0] == xArray && helperArray[1] == yArray) {

\*(makeUpedRock + i) = 1;

}

}

}

if(flag == 2) { // Добавление нового элемента в массив

numberOfRowsFir += 1;

rock = (int\*) realloc(rock, numberOfRowsFir \* n \* sizeof(int));

\*(rock + (numberOfRowsFir - 1) \* n + 0) = helperArray[0];

\*(rock + (numberOfRowsFir - 1) \* n + 1) = helperArray[1];

makeUpedRock = (int\*) realloc(makeUpedRock, numberOfRowsFir \* sizeof(int));

\*(makeUpedRock + numberOfRowsFir -1) = 0;

}

}

/\*

\* Типы узлов в AST:

\* 'T': Узел, представляющий оператор с шагами во времени.

\* 'a': Узел, представляющий действие.

\* 's': Узел, представляющий оператор (statement).

\* 'e': Узел, представляющий ветвь "иначе" (else).

\* 'F': Узел, представляющий свободное направление (free direction).

\* 'K': Узел, представляющий числовое значение (число).

\* 'u': Узел, представляющий движение вверх.

\* 'd': Узел, представляющий движение вниз.

\* 'l': Узел, представляющий движение влево.

\* 'r': Узел, представляющий движение вправо.

\* 'D': Узел, представляющий действие.

\* 'M': Узел, представляющий действие.

\* 'DES': Узел, представляющий действие.

\* 'I': Узел, представляющий условный оператор "if".

\*/

// Освобождение памяти занятой AST

void freeAstTree(struct ast \*a) {

switch(a->nodetype) {

// Два поддерева

case 'T':

case 'a':

freeAstTree(a->r);

// Одно поддерево

case 's':

case 'e':

case 'F':

freeAstTree(a->l);

// Нет поддеревьев

case 'K':

case 'u':

case 'd':

case 'l':

case 'r':

case 'D':

case 'M':

case 'DES':

break;

// Условие и цикл

case 'I':

break;

default: fprintf(yyout, "%d. Внутренняя ошибкааа: освобождение некорректного узла %c\n", counter, a->nodetype);

}

}