**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc156055036)

[РАЗДЕЛ 1. АНАЛИЗАТОРЫ 5](#_Toc156055037)

[1.1. Лексический анализатор flex 5](#_Toc156055038)

[1.2. Синтаксический анализатор bison 5](#_Toc156055039)

[РАЗДЕЛ 2. ГРАММАТИКА ЯЗЫКА 8](#_Toc156055040)

[РАЗДЕЛ 3. АНАЛИЗ ВОЗМОЖНОСТЕЙ И ОГРАНИЧЕНИЙ 11](#_Toc156055041)

[ЗАКЛЮЧЕНИЕ 12](#_Toc156055042)

[СПИСОК ЛИТЕРАТУРЫ 13](#_Toc156055043)

[ПРИЛОЖЕНИЯ 14](#_Toc156055044)

# ВВЕДЕНИЕ

Эта курсовая работа посвящена разработке собственного языка программирования с использованием инструментов flex и bison для создания лексического и синтаксического анализаторов. В нее также включено описание грамматики разрабатываемого языка, методы реализации этой грамматики с помощью упомянутых инструментов, а также исходные коды как лексического, так и синтаксического анализаторов. Работа также будет содержать информацию об ограничениях, налагаемых на функционирование создаваемого языка программирования.

Финальным продуктом курсовой станет новый язык программирования, предназначенный для управления роботом-художником, задача которого заключается в закрашивании пола комнаты, ограниченной стенами.

# РАЗДЕЛ 1. АНАЛИЗАТОРЫ

## 1.1. Лексический анализатор flex

Flex является инструментом, используемым для создания программ, выполняющих лексический анализ, таких как компиляторы. Он работает на основе заданных пользователем правил, которые устанавливают соответствие между лексическими единицами, такими как ключевые слова, идентификаторы и операторы, и их шаблонами в коде. Эти правила прописываются в файлах с расширением .lex.l и состоят из пар "шаблон - действие".

После определения правил, Flex производит создание исходного кода для лексического анализатора, который затем может быть скомпилирован и использован в нужном программном проекте. Такой анализатор обрабатывает входящие символьные данные, разделяя их на токены в соответствии с установленными правилами.

Своей гибкостью и производительностью, Flex приобрел репутацию надежного инструмента для разработки лексических анализаторов в области программирования компиляторов и интерпретаторов.

|  |
| --- |
|  |
| Рис. 1. Схема работы flex |

## 1.2. Синтаксический анализатор bison

Bison - это генератор синтаксических анализаторов, который работает на основе грамматики, заданной в форме контекстно-свободной грамматики. Он часто используется для создания синтаксических анализаторов (парсеров) для компиляторов и интерпретаторов[1].

Bison – программный инструментарий, предназначенный для генерации синтаксических анализаторов на основе LALR – грамматик[1].

LALR – грамматики (LA (англ. lookahead) – предпросмотр, L (англ. left) – порядок чтения входной цепочки символов слева направо, R (англ. right) – в результате работы алгоритма получается правосторонний вывод, 1 – количество символов предпросмотра) относятся к классу контекстно-свободных грамматик, к которым применимы восходящие алгоритмы синтаксического анализа. Лексический анализатор, сгенерированный Flex, является сопрограммой для синтаксического анализатора, генерируемого с помощью Bison (рис. 2).

Лексический анализатор возвращает синтаксическому поток токенов. Задача распознавателя выяснить взаимоотношения между этими токенами. Обычно такие взаимоотношения отображаются в виде дерева разбора[1].

Например, для арифметического выражения 1 \* 2 + 3 \* 4 + 5 получится дерево разбора, изображённое на рисунке 3. Операция умножения имеет более высокий приоритет, чем операция сложения, поэтому первыми будут операции 1 \* 2 и 3 \* 4. Затем результаты этих двух операций складываются вместе, и эта сумма складывается с 5. Каждая ветвь дерева показывает взаимоотношение между токенами или поддеревьями.

Любой синтаксический анализатор, сгенерированный с использованием Bison, формирует дерево разбора по мере анализа входных токенов.

|  |
| --- |
|  |
| Рис. 2. Схема совместной работы flex и bison |

|  |
| --- |
|  |
| Рис. 3. Дерево разбора для арифметического выражения 1 \* 2 + 3 \* 4 + 5 |

Bison работает с грамматикой, которая определяется во входном файле, и создаёт синтаксический анализатор, распознающий «предложения», соответствующие этой грамматике. Для грамматики языка программирования такими предложениями будут синтаксически правильные программы на данном языке[1].

Синтаксически правильная программа не всегда является семантически правильной. Например, для языка Си присваивание строкового значения переменной типа int – семантически неверно, но удовлетворяет синтаксическим правилам языка. Bison проверяет только правильность синтаксиса.

Описание грамматики на языке Bison и его соответствие форме Бэкуса-Наура приведено в таблице 1. Вертикальная черта ( | ) показывает, что существует две возможности задания одного и того же нетерминального символа или что несколько правил могут иметь одинаковую левую часть. Символы в левой части правила – нетерминалы. Символы, возвращаемые лексическим анализатором, – терминалы или токены (в таблице 1 таким символом является NAME). Нельзя использовать одинаковые имена для терминальных и нетерминальных символов.

|  |  |  |
| --- | --- | --- |
| |  | | --- | |  | | Рис. 4. Дерево разбора для цепочки символов fred = 12 + 13 | |

В этом примере 12 + 13 соответствует нетерминалу expression, а fred = expression формирует statement. Каждая грамматика содержит начальный символ, который выступает в качестве корня дерева разбора. В данной грамматике statement является таким символом. Правила могут явным или неявным образом ссылаться сами на себя. Такое свойство позволяет разбирать входные предложения любой длины.

# РАЗДЕЛ 2. ГРАММАТИКА ЯЗЫКА

В файле area.txt написаны координаты откуда будет начинать ходить робот. Файл cmd.txt принимает команды, которые мой язык программирования может обрабатывать.

Робот способен перемещаться в четыре стороны, вверх, вправо, влево, вниз и закрашивать те клетки, где стоит для этого есть команда paint.

Приведенный, в приложении 1, код представляет собой спецификацию лексического анализатора (токенизатора) на языке flex.

Этот лексический анализатор будет использоваться для разбора исходного кода, созданного на основе заданной грамматики.

В нем приведены правила для различных лексем, таких как целые числа, ключевые слова (if, else, while), скобки и другие символы.

Данный код описывает лексический анализатор, который будет использоваться в сочетании с синтаксическим анализатором, созданным с помощью bison, для анализа и трансляции входного исходного кода.

В приложении 2 представлен код на языке синтаксического анализатора bison.

Ниже приведен фрагмент кода — это часть синтаксического анализатора, написанного для Bison, который используется для создания компиляторов. Bison преобразует грамматическое описание языка в программу, которая парсит текст в соответствии с этой грамматикой. В фрагменте определяются правила, по которым синтаксический анализатор должен разбирать предложения или конструкции нового языка программирования для управления роботом-маляром.

commands:

| commands command { evaluate($2); treeFree($2); }

;

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

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

| WHILE OB condition CB FOB action FCB { $$ = newFlow('W', $3, $6, NULL); }

| action { $$ = newAst('a', $1, NULL); }

;

condition: AREA IS TRUE move { $$ = newAst('c', $4, NULL); }

;

action: COLOUR OB color CB SEMICOLON { $$ = newAst('o', $3, NULL); }

| move base { $$ = newAst('m', $1, $2); }

;

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

;

color: RED { $$ = newAst('R', NULL, NULL); }

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

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

;

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

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

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

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

;

base: OB operation COMMA lenth CB SEMICOLON { $$ = newAst('b', $2, $4); }

;

operation: NOTHING { $$ = newAst('n', NULL, NULL); }

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

;

lenth: INTEGER { $$ = newNum($1); }

Общая структура описывает, как Bison должен интерпретировать входные данные (написанные на разрабатываемом языке программирования) и преобразовывать их в структурированные данные (AST), которые затем могут быть использованы для генерации исполняемого кода или интерпретации команд роботом.

На вход подается псевдокод лексика и грамматика, которого прописана в lexer.l и parser.y соответственно, а также файлы area.txt и cmd.txt. На выход в файл result.txt получаем координаты, на которые робот переместился.

В табл. 1 представлены входные и выходные данные.

|  |  |
| --- | --- |
| Входной файл area.txt | Входной файл cmd.txt |
|  |  |
| Выходной файл result.txt | |
|  | |

# РАЗДЕЛ 3. АНАЛИЗ ВОЗМОЖНОСТЕЙ И ОГРАНИЧЕНИЙ

В данном языке не реализовано создание пользовательский переменных, с помощью которых можно было хранить количество шагов роботу, что было бы удобно для написание больших кодов. Однако в нашем языке целые числа используются только для того, чтобы указывать на сколько шагов робот переместится, то данное ограничение является не значительным.

# ЗАКЛЮЧЕНИЕ

Был создан язык программирования, с помощью которого можно управлять роботом-маляром, который красит пол в комнате. Робот способен взаимодействовать с комнатой.

# СПИСОК ЛИТЕРАТУРЫ

1. Sibsutis.tu: [Электронный ресурс]. URL: <https://csc.sibsutis.ru/sites/csc.sibsutis.ru/files/courses/trans/LabWork4.pdf>. (Дата обращения 13.01.2024)

# ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1

%{

#include <stdlib.h>

#include <stdio.h>

#include "parser.tab.h"

#include <string.h>

%}

%%

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

"(" { return OB; }

")" { return CB; }

"{" { return FOB; }

"}" { return FCB; }

"," { return COMMA; }

";" { return SEMICOLON; }

"if" { return IF; }

"else" { return ELSE; }

"while" { return WHILE; }

"area" { return AREA; }

"->" { return IS; }

"true" { return TRUE; }

"up" { return UP; }

"down" { return DOWN; }

"left" { return LEFT; }

"right" { return RIGHT; }

"red" { return RED; }

"green" { return GREEN; }

"blue" { return BLUE; }

"paint" { return PAINT; }

"colour" { return COLOUR; }

"nothing" { return NOTHING; }

[ \t\n]+ { /\* Ignore whitespace. \*/ }

. { printf("Unrecognized character %c\n", \*yytext); }

%%

ПРИЛОЖЕНИЕ 2

%{

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "parser.tab.h"

int yylex(void);

extern int yylineno;

extern FILE\* yyin;

extern FILE\* yyout;

void yyerror(char \*str);

int yywrap(){

return 1;

}

struct ast{

int nodetype;

struct ast \*l;

struct ast \*r;

};

struct numval{

int nodetype;

int number;

};

struct flow{

int nodetype;

struct ast \*cond;

struct ast \*tl;

struct ast \*el;

};

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

struct ast \*newNum(int integer);

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

int evaluate(struct ast \*);

void treeFree(struct ast \*);

int count = 0;

int robot[2];

int area[2];

void robot\_funcs(int checkOperation, int step);

int area\_check(int step);

void move(int step);

int root;

void rep\_colour(int checkColor);

int temp\_colour = 0;

void paint(int step);

int subPaint(int x1, int y1, int x2, int y2, int x, int y);

int \*paint\_arr = NULL;

int paint\_row;

%}

%union{

struct ast \*a;

int ivalue;

}

%token OB CB FCB FOB COMMA SEMICOLON

%token IF ELSE WHILE

%token AREA IS TRUE

%token UP DOWN LEFT RIGHT

%token PAINT COLOUR NOTHING

%token RED GREEN BLUE

%token <ivalue> INTEGER

%type <a> command condition else action base move color operation lenth

%%

commands:

| commands command { evaluate($2); treeFree($2); }

;

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

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

| WHILE OB condition CB FOB action FCB { $$ = newFlow('W', $3, $6, NULL); }

| action { $$ = newAst('a', $1, NULL); }

;

condition: AREA IS TRUE move { $$ = newAst('c', $4, NULL); }

;

action: COLOUR OB color CB SEMICOLON { $$ = newAst('o', $3, NULL); }

| move base { $$ = newAst('m', $1, $2); }

;

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

;

color: RED { $$ = newAst('R', NULL, NULL); }

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

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

;

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

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

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

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

;

base: OB operation COMMA lenth CB SEMICOLON { $$ = newAst('b', $2, $4); }

;

operation: NOTHING { $$ = newAst('n', NULL, NULL); }

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

;

lenth: INTEGER { $$ = newNum($1); }

%%

int main(void){

char \*commandFileName = "cmd.txt";

FILE\* commandFile = fopen(commandFileName, "r");

if (commandFile == NULL){

fprintf(yyout, "Can't open file %s", commandFileName);

exit(1);

}

char \*areaFileName = "area.txt";

FILE\* areaFile = fopen(areaFileName, "r");

if (areaFile == NULL){

fprintf(yyout, "Can't open file %s", areaFileName);

exit(1);

}

fseek(areaFile, 0, SEEK\_SET);

fscanf(areaFile, "%d ", &area[0]);

fscanf(areaFile, "%d", &area[1]);

fscanf(areaFile, "%d ", &robot[0]);

fscanf(areaFile, "%d", &robot[1]);

char \*reportFileName = "result.txt";

FILE\* reportFile = fopen(reportFileName, "w");

yyin = commandFile;

yyout = reportFile;

yyparse();

fclose(yyin);

fclose(areaFile);

fclose(yyout);

free(paint\_arr);

return 0;

}

void yyerror(char \*str){

count++;

fprintf(yyout ,"error: %s in line %d\n", str, yylineno);

exit(1);

}

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

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

if (!a){

yyerror("out of space");

exit(0);

}

a->nodetype = nodetype;

a->l = l;

a->r = r;

return a;

}

struct ast \*newNum(int i){

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

if (!a){

yyerror("out of space");

exit(0);

}

a->nodetype = 'K';

a->number = i;

return (struct ast \*)a;

}

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

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

if(!a) {

yyerror("out of space");

exit(0);

}

a->nodetype = nodetype;

a->cond = cond;

a->tl = tl;

a->el = el;

return (struct ast \*)a;

}

int evaluate(struct ast \*a){

int value;

int checkOperation;

int checkColor;

switch(a->nodetype){

case 'K': value = ((struct numval \*)a)->number; break;

case 'a':

evaluate(a->l);

break;

case 'c':

root = evaluate(a->l);

value = area\_check(1);

break;

case 'e':

evaluate(a->l);

break;

case 'o':

count++;

checkColor = evaluate(a->l);

rep\_colour(checkColor);

temp\_colour = checkColor;

break;

case 'm':

root = evaluate(a->l);

evaluate(a->r);

break;

case 'b':

count++;

checkOperation = evaluate(a->l);

value = evaluate(a->r);

robot\_funcs(checkOperation, value);

break;

case 'n':

value = 'n';

break;

case 's':

value = 's';

break;

case 'l':

value = 'l';

break;

case 'r':

value = 'r';

break;

case 'u':

value = 'u';

break;

case 'd':

value = 'd';

break;

case 'R':

value = 'R';

break;

case 'G':

value = 'G';

break;

case 'B':

value = 'B';

break;

case 'I':

if(evaluate(((struct flow \*)a)->cond) == 't') {

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

evaluate(((struct flow \*)a)->tl);

}

else{

value = 'f';

}

}

else {

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

evaluate(((struct flow \*)a)->el);

}

else {

value = 'f';

}

}

break;

case 'W':

value = 'f';

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

while(evaluate(((struct flow \*)a)->cond) == 't'){

evaluate(((struct flow \*)a)->tl);

}

}

break;

}

return value;

}

void rep\_colour(int checkColor){

if (checkColor == temp\_colour){

switch(checkColor){

case 'R':

fprintf(yyout, "Red color is already selected\n");

break;

case 'G':

fprintf(yyout, "Green color is already selected\n");

break;

case 'B':

fprintf(yyout, "Blue color is already selected\n");

break;

}

}

else{

switch(checkColor){

case 'R':

fprintf(yyout, "Red color is selected\n");

break;

case 'G':

fprintf(yyout, "Green color is selected\n");

break;

case 'B':

fprintf(yyout, "Blue color is selected\n");

break;

}

}

}

void robot\_funcs(int checkOperation, int step){

switch(checkOperation){

case 'n':

move(step);

fprintf(yyout, "Robot moved to coordinate (%d,%d)\n", robot[0], robot[1]);

break;

case 's':

paint(step);

break;

}

}

int area\_check(int step){

int tempArray[2] = {robot[0], robot[1]};

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

switch(root){

case 'l':

if (tempArray[0] - 1 < 0){

return 'f';

}

break;

case 'r':

if (tempArray[0] + 1 > area[0]){

return 'f';

}

break;

case 'd':

if (tempArray[1] - 1 < 0){

return 'f';

}

break;

case 'u':

if (tempArray[1] + 1 > area[1]){

return 'f';

}

break;

}

switch(root){

case 'l':

tempArray[0] -= 1;

break;

case 'r':

tempArray[0] += 1;

break;

case 'd':

tempArray[1] -= 1;

break;

case 'u':

tempArray[1] += 1;

break;

}

}

return 't';

}

void move(int step){

switch(area\_check(step)){

case 't':

if (root == 'l'){

robot[0] -= step;

}

if (root == 'r'){

robot[0] += step;

}

if (root == 'd'){

robot[1] -= step;

}

if (root == 'u'){

robot[1] += step;

}

break;

case 'f':

if (root == 'l'){

fprintf(yyout, "Error: Robot is trying to move to coordinate (%d,%d) beyond the field boundaries (%d,%d)\n", robot[0] - step, robot[1], area[0], area[1]);

exit(0);

}

if (root == 'r'){

fprintf(yyout, "Error: Robot is trying to move to coordinate (%d,%d) beyond the field boundaries (%d,%d)\n", robot[0] + step, robot[1], area[0], area[1]);

exit(0);

}

if (root == 'd'){

fprintf(yyout, "Error: Robot is trying to move to coordinate (%d,%d) beyond the field boundaries (%d,%d)\n", robot[0], robot[1] - step, area[0], area[1]);

exit(0);

}

if (root == 'u'){

fprintf(yyout, "Error: Robot is trying to move to coordinate (%d,%d) beyond the field boundaries (%d,%d)\n", robot[0], robot[1] + step, area[0], area[1]);

exit(0);

}

break;

}

}

void paint(int step){

if (area\_check(step) == 'f'){

switch(root){

case 'l':

fprintf(yyout, "Error: Robot is trying to paint (%d,%d) (%d,%d) beyond the field boundaries (%d,%d)\n", robot[0] - step, robot[1], robot[0], robot[1], area[0], area[1]);

exit(0);

break;

case 'r':

fprintf(yyout, "Error: Robot is trying to paint (%d,%d) (%d,%d) beyond the field boundaries (%d,%d)\n", robot[0], robot[1], robot[0] + step, robot[1], area[0], area[1]);

exit(0);

break;

case 'd':

fprintf(yyout, "Error: Robot is trying to paint (%d,%d) (%d,%d) beyond the field boundaries (%d,%d)\n", robot[0], robot[1] - step, robot[0], robot[1], area[0], area[1]);

exit(0);

break;

case 'u':

fprintf(yyout, "Error: Robot is trying to paint (%d,%d) (%d,%d) beyond the field boundaries (%d,%d)\n", robot[0], robot[1], robot[0], robot[1] + step, area[0], area[1]);

exit(0);

break;

}

}

int resultFirstSubPaint;

int resultSecondSubPaint;

int x1paint\_arr;

int y1paint\_arr;

int x2paint\_arr;

int y2paint\_arr;

for (int i = 0; i < paint\_row; i++){

x1paint\_arr = \*(paint\_arr + i\*4 + 0);

y1paint\_arr = \*(paint\_arr + i\*4 + 1);

x2paint\_arr = \*(paint\_arr + i\*4 + 2);

y2paint\_arr = \*(paint\_arr + i\*4 + 3);

switch(root){

case 'l':

resultFirstSubPaint = subPaint(robot[0] - step, robot[1], robot[0], robot[1], x1paint\_arr, y1paint\_arr);

resultSecondSubPaint = subPaint(robot[0] - step, robot[1], robot[0], robot[1], x2paint\_arr, y2paint\_arr);

if (robot[1] == y1paint\_arr & robot[1] == y2paint\_arr & (resultFirstSubPaint == 't' | resultSecondSubPaint == 't')){

fprintf(yyout, "Error: Robot cannot paint (%d,%d) (%d,%d) because it is already painted (%d,%d) (%d,%d)", robot[0] - step, robot[1], robot[0], robot[1], x1paint\_arr, y1paint\_arr, x2paint\_arr, y2paint\_arr);

exit(0);

}

break;

case 'r':

resultFirstSubPaint = subPaint(robot[0], robot[1], robot[0] + step, robot[1], x1paint\_arr, y1paint\_arr);

resultSecondSubPaint = subPaint(robot[0], robot[1], robot[0] + step, robot[1], x2paint\_arr, y2paint\_arr);

if (robot[1] == y1paint\_arr & robot[1] == y2paint\_arr & (resultFirstSubPaint == 't' | resultSecondSubPaint == 't')){

fprintf(yyout, "Error: Robot cannot paint (%d,%d) (%d,%d) because it is already painted (%d,%d) (%d,%d)", robot[0], robot[1], robot[0] + step, robot[1], x1paint\_arr, y1paint\_arr, x2paint\_arr, y2paint\_arr);

exit(0);

}

break;

case 'd':

resultFirstSubPaint = subPaint(robot[0], robot[1] - step, robot[0], robot[1], x1paint\_arr, y1paint\_arr);

resultSecondSubPaint = subPaint(robot[0], robot[1] - step, robot[0], robot[1], x2paint\_arr, y2paint\_arr);

if (robot[0] == x1paint\_arr & robot[0] == x2paint\_arr & (resultFirstSubPaint == 't' | resultSecondSubPaint == 't')){

fprintf(yyout, "Error: Robot cannot paint (%d,%d) (%d,%d) because it is already painted (%d,%d) (%d,%d)", robot[0], robot[1] - step, robot[0], robot[1], x1paint\_arr, y1paint\_arr, x2paint\_arr, y2paint\_arr);

exit(0);

}

break;

case 'u':

resultFirstSubPaint = subPaint(robot[0], robot[1], robot[0], robot[1] + step, x1paint\_arr, y1paint\_arr);

resultSecondSubPaint = subPaint(robot[0], robot[1], robot[0], robot[1] + step, x2paint\_arr, y2paint\_arr);

if (robot[0] == x1paint\_arr & robot[0] == x2paint\_arr & (resultFirstSubPaint == 't' | resultSecondSubPaint == 't')){

fprintf(yyout, "Error: Robot cannot paint (%d,%d) (%d,%d) because it is already painted (%d,%d) (%d,%d)", robot[0], robot[1], robot[0], robot[1] + step, x1paint\_arr, y1paint\_arr, x2paint\_arr, y2paint\_arr);

exit(0);

}

break;

}

}

paint\_row++;

int paint\_rowArray = paint\_row - 1;

paint\_arr = (int\*) realloc(paint\_arr, (paint\_row + 1) \* 4 \* sizeof(int));

switch(root){

case 'l':

\*(paint\_arr + paint\_rowArray \* 4 + 0) = robot[0] - step;

\*(paint\_arr + paint\_rowArray \* 4 + 1) = robot[1];

\*(paint\_arr + paint\_rowArray \* 4 + 2) = robot[0];

\*(paint\_arr + paint\_rowArray \* 4 + 3) = robot[1];

robot[0] -= step;

break;

case 'r':

\*(paint\_arr + paint\_rowArray \* 4 + 0) = robot[0];

\*(paint\_arr + paint\_rowArray \* 4 + 1) = robot[1];

\*(paint\_arr + paint\_rowArray \* 4 + 2) = robot[0] + step;

\*(paint\_arr + paint\_rowArray \* 4 + 3) = robot[1];

robot[0] += step;

break;

case 'd':

\*(paint\_arr + paint\_rowArray \* 4 + 0) = robot[0];

\*(paint\_arr + paint\_rowArray \* 4 + 1) = robot[1] - step;

\*(paint\_arr + paint\_rowArray \* 4 + 2) = robot[0];

\*(paint\_arr + paint\_rowArray \* 4 + 3) = robot[1];

robot[1] -= step;

break;

case 'u':

\*(paint\_arr + paint\_rowArray \* 4 + 0) = robot[0];

\*(paint\_arr + paint\_rowArray \* 4 + 1) = robot[1];

\*(paint\_arr + paint\_rowArray \* 4 + 2) = robot[0];

\*(paint\_arr + paint\_rowArray \* 4 + 3) = robot[1] + step;

robot[0] += step;

break;

}

switch(temp\_colour){

case 'R':

fprintf(yyout, "Robot painted the row with red (%d,%d) (%d,%d)\n",\*(paint\_arr + paint\_rowArray \* 4 + 0), \*(paint\_arr + paint\_rowArray \* 4 + 1), \*(paint\_arr + paint\_rowArray \* 4 + 2), \*(paint\_arr + paint\_rowArray \* 4 + 3));

break;

case 'G':

fprintf(yyout, "Robot painted the row with green (%d,%d) (%d,%d)\n",\*(paint\_arr + paint\_rowArray \* 4 + 0), \*(paint\_arr + paint\_rowArray \* 4 + 1), \*(paint\_arr + paint\_rowArray \* 4 + 2), \*(paint\_arr + paint\_rowArray \* 4 + 3));

break;

case 'B':

fprintf(yyout, "Robot painted the row with blue (%d,%d) (%d,%d)\n",\*(paint\_arr + paint\_rowArray \* 4 + 0), \*(paint\_arr + paint\_rowArray \* 4 + 1), \*(paint\_arr + paint\_rowArray \* 4 + 2), \*(paint\_arr + paint\_rowArray \* 4 + 3));

break;

default:

fprintf(yyout, "Robot painted the row with white (%d,%d) (%d,%d)\n",\*(paint\_arr + paint\_rowArray \* 4 + 0), \*(paint\_arr + paint\_rowArray \* 4 + 1), \*(paint\_arr + paint\_rowArray \* 4 + 2), \*(paint\_arr + paint\_rowArray \* 4 + 3));

break;

}

}

int subPaint(int x1, int y1, int x2, int y2, int x, int y){

int result = (x - x1) \* (y2 - y1) - (x2 - x1) \* (y - y1);

if (result == 0){

return 't';

}

return 'f';

}

void treeFree(struct ast \*a){

switch(a->nodetype){

case 'm':

case 'b':

treeFree(a->r);

case 'a':

case 'o':

case 'e':

treeFree(a->l);

case 'K':

case 'c':

case 'R':

case 'G':

case 'B':

case 'l':

case 'r':

case 'u':

case 'd':

case 'n':

case 's':

case 't':

break;

case 'I':

case 'W':

free( ((struct flow \*)a)->cond);

if( ((struct flow \*)a)->tl) free( ((struct flow \*)a)->tl);

if( ((struct flow \*)a)->el) free( ((struct flow \*)a)->el);

break;

default: fprintf(yyout, "internal error: free bad node %c\n",a->nodetype);

}

}