Университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04

Программная инженерия

Дисциплина «Разработка компиляторов»

Отчет

по зачетному заданию

по курсу «Разработка компиляторов»

Выполнили:

Савельева Диана Александровна

Убоженко Сергей Дмитриевич

P33082

Преподаватель:

[Лаздин Артур Вячеславович](https://my.itmo.ru/persons/102300)

Санкт-Петербург, 2024 г

**1. Задание**

Разработать язык программирования, который должен реализовать следующие компоненты:

* Присваивание (оператор или операция), арифметические и логические операции.
* Ветвление, включая вариант факультативного else.
* Цикл while.
* Поддержка целочисленного и логического типа данных.
* Многострочные комментарии в стиле Си-подобных языков.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* строки комментариев \*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Это минимальные требования (на 4С).  
Что добавит баллы:

Вместо ветвление if [then] else конструируется   
оператор if elif [elif]+ else

Вместо цикла while (или в дополнение к нему) конструируется цикл for.

Реализуется вывод значений.

Одно дополнение позволяет получить 4В, любые два 5А.

Содержание работы:

1. Разработка описания языка в терминах КС грамматики. Определение лексического состава языка.
2. Подготовить файлы \*.l для Flex в соответствии с лексическим составом языка, выполнить синтаксическое описание языка в нотации Bison (файл \*.y). Учесть приоритеты и ассоциативность бинарных операций.
3. Разработать функции, реализующие построение и вывод АСТ (AST Abstract syntax tree). Вывод АСТ осуществить в файл.
4. Получение промежуточного представления программы и генерация кода для него. (ссылка на README в репозитории эмулятора <https://github.com/asurkis/risc-emulator>)
5. Оформление отчета с описанием выполненных действий, демонстрация работы компилятора для корректных и ошибочных программ.

**GitHub с готовым проектом:**

**https://github.com/ITSamantha/chirp-lang**

**2. Лексический состав языка и КС-грамматика**

*Описание языка CHIRP в терминах КС-грамматики:*

chirp -> CHIRP START statements FINISH YYEOF

| error YYEOF

statements -> ε

| expression ';' statements

| statement statements

statement -> if\_statement

| while\_statement

| for\_statement

for\_statement -> FOR '(' expression ';' expression ';' expression ')' START\_STATEMENT statements FINISH\_STATEMENT

while\_statement -> WHILE expression START\_STATEMENT statements FINISH\_STATEMENT

if\_statement -> IF expression START\_STATEMENT statements FINISH\_STATEMENT else\_statement

| IF expression START\_STATEMENT statements FINISH\_STATEMENT elif\_statement

elif\_statement -> ELIF expression START\_STATEMENT statements FINISH\_STATEMENT else\_statement

| ELIF expression START\_STATEMENT statements FINISH\_STATEMENT elif\_statement

else\_statement -> ε

| ELSE START\_STATEMENT statements FINISH\_STATEMENT

expression -> comparison\_operations

| aritmetic\_operations

| logical\_operations

| '(' expression ')'

| INTEGER

| NAME

| NAME '=' expression

logical\_operations -> expression AND expression

| expression OR expression

| NOT expression

aritmetic\_operations -> expression '+' expression

| expression '-' expression

| expression '\*' expression

| expression '/' expression

comparison\_operations -> expression CMP expression

CHIRP -> 'чирп'

START -> 'начпрог'

FINISH -> 'конпрог'

IF -> 'если'

ELSE -> 'иначе'

WHILE -> 'пока'

FOR -> 'для'

ELIF -> 'иначе если'

START\_STATEMENT -> 'нач'

FINISH\_STATEMENT -> 'кон'

AND -> 'и'

OR -> 'или'

NOT -> 'не'

INTEGER -> [0-9]+

NAME -> [a-zA-Z][a-zA-Z0-9]\*

CMP -> '>' | '<' | '!=' | '==' | '>=' | '<='

*Лексический состав языка CHIRP:*

V = { chirp, statements, statement, for\_statement, while\_statement, if\_statement, elif\_statement, else\_statement, expression, logical\_operations, aritmetic\_operations, comparison\_operations, CHIRP, START, FINISH, IF, ELSE, WHILE, FOR, ELIF, START\_STATEMENT, FINISH\_STATEMENT, AND, OR, NOT, CMP, INTEGER, NAME, '+', '-', '\*', '/', '=', ',', ';', '(', ')', '{', '}', YYEOF, a-z, A-Z, 0-9 }

**3. Лексер и парсер для языка CHIRP**

*Файл Flex (лексер) для языка CHIRP:*

%{

#include "ast.h"

#include <stdio.h>

#include "chirp\_parser.tab.h"

%}

%option noyywrap

%x COMMENT

%%

[0-9][a-zA-Z][a-zA-Z0-9]\* { yyerror("Variable name cannot start with a digit!");}

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

"+" |

"-" |

"\*" |

"/" |

"=" |

"," |

";" |

"(" |

")" { return yytext[0]; }

">" { yylval.typeToken = AST\_GT; return CMP; }

"<" { yylval.typeToken = AST\_LT; return CMP; }

"!=" { yylval.typeToken = AST\_NEQ; return CMP; }

"==" { yylval.typeToken = AST\_EQ; return CMP; }

">=" { yylval.typeToken = AST\_GTE; return CMP; }

"<=" { yylval.typeToken = AST\_LTE; return CMP; }

"если" { return IF; }

"иначе" { return ELSE; }

"пока" { return WHILE; }

"для" { return FOR; }

"иначе если" { return ELIF; }

"начпрог" { return START; }

"конпрог" { return FINISH; }

"нач" { return START\_STATEMENT; }

"кон" { return FINISH\_STATEMENT; }

"чирп" { return CHIRP; }

"и" { return AND; }

"или" { return OR; }

"не" { return NOT; }

"правда" { yylval.intVal = 1; return INTEGER; }

"ложь" { yylval.intVal = 0; return INTEGER; }

"/\*" { BEGIN(COMMENT); }

<COMMENT>"\*/" { BEGIN(INITIAL); }

<COMMENT>. { }

<COMMENT>\n { }

<COMMENT><<EOF>> { yyerror("ERROR: Unterminated comment!"); }

"//".\*

[a-zA-Z][a-zA-Z0-9]\* { yylval.name = strdup(yytext); return NAME; }

[ \t]+ { }

\\\n {printf("c> "); }

\n { }

. { printf("Unexpected character: %c\n", yytext[0]); }

%%

*Файл Bison (парсер) для языка CHIRP:*

%{

#include "ast.h"

#include <stdio.h>

#include <stdlib.h>

#include <stdarg.h>

void yyerror(char \*s, ...);

int yylex(void);

%}

%union {

struct ast \*nested;

int intVal;

char \*name;

ntype\_t typeToken;

}

%token <intVal> INTEGER

%token <name> NAME

%token EOL YYEOF

%token IF ELSE WHILE FOR ELIF

%token START FINISH START\_STATEMENT FINISH\_STATEMENT CHIRP

%right '='

%left OR

%left AND

%left NOT

%left <typeToken> CMP

%left '+' '-'

%left '\*' '/'

%type <nested> expression statements statement if\_statement elif\_statement else\_statement for\_statement while\_statement logical\_operations aritmetic\_operations comparison\_operations

%start chirp

%%

chirp: CHIRP START statements FINISH YYEOF {

if($3 != NULL) {

print\_ast(stdout, $3, 0);

FILE\* file1 = fopen("chirp\_out.S", "w");

FILE\* file2 = fopen("chirp\_tree", "w");

if(file1 == NULL) {

printf("ERROR: Can not open file!\n");

exit(1);

}

if(file2 == NULL) {

printf("ERROR: Can not open file!\n");

exit(1);

}

print\_asm(file1, $3);

print\_ast(file2, $3, 0);

fclose(file1);

treefree($3);

}

printf(">>> ");

}

| error YYEOF { yyerrok; printf(">>> ");}

;

statement: if\_statement

| while\_statement

| for\_statement

;

for\_statement : FOR '(' expression ';' expression ';' expression ')' START\_STATEMENT statements FINISH\_STATEMENT { $$ = newfor($3, $5, $7, $10); }

;

while\_statement: WHILE expression START\_STATEMENT statements FINISH\_STATEMENT { $$ = newstatement(AST\_WHILE, $2, $4, NULL); }

;

if\_statement: IF expression START\_STATEMENT statements FINISH\_STATEMENT else\_statement { $$ = newstatement(AST\_IF, $2, $4, $6); }

| IF expression START\_STATEMENT statements FINISH\_STATEMENT elif\_statement { $$ = newstatement(AST\_IF, $2, $4, $6); }

;

elif\_statement: ELIF expression START\_STATEMENT statements FINISH\_STATEMENT else\_statement { $$ = newstatement(AST\_IF, $2, $4, $6); }

| ELIF expression START\_STATEMENT statements FINISH\_STATEMENT elif\_statement { $$ = newstatement(AST\_IF, $2, $4, $6); }

;

else\_statement: { $$ = NULL; }

| ELSE START\_STATEMENT statements FINISH\_STATEMENT { $$ = $3; }

;

statements: { $$ = NULL; }

| expression ';' statements {

if($3 == NULL) $$ = $1;

else $$ = newast(AST\_STATEMENTS, $1, $3);

}

| statement statements {

if($2 == NULL) $$ = $1;

else $$ = newast(AST\_STATEMENTS, $1, $2);

}

;

expression: comparison\_operations

| aritmetic\_operations

| logical\_operations

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

| INTEGER { $$ = newnum($1); }

| NAME { $$ = newref($1); }

| NAME '=' expression { $$ = newasgn($1, $3); }

;

logical\_operations: expression AND expression { $$ = newast(AST\_AND, $1, $3); }

| expression OR expression { $$ = newast(AST\_OR, $1, $3); }

| NOT expression { $$ = newast(AST\_NOT, $2, NULL); }

;

aritmetic\_operations: expression '+' expression { $$ = newast(AST\_ADD, $1, $3); }

| expression '-' expression { $$ = newast(AST\_SUB, $1, $3); }

| expression '\*' expression { $$ = newast(AST\_MUL, $1, $3); }

| expression '/' expression { $$ = newast(AST\_DIV, $1, $3); }

;

comparison\_operations: expression CMP expression { $$ = newcmp($2, $1, $3); }

;

%%

**4. Функции для вывода и построения AST**

Функции для вывода и построения AST для языка CHIRP разработаны в файлах ast.h и ast.c.

<https://github.com/ITSamantha/chirp-lang/blob/main/ast.h>

<https://github.com/ITSamantha/chirp-lang/blob/main/ast.c>

**5. Ход и результаты работы**

**5.1 Присваивание**

*Программа имеет вид:*

чирп

начпрог

x1 = 13;

x2 = x1;

конпрог

*Сформированное AST представление:*

STATEMENTS

ASSIGNMENT x1

CONSTANT 13

ASSIGNMENT x2

NAME x1

*Сформированный chirp\_out.S:*

jal x1, MAIN

x1:

data 0 \* 1

x2:

data 0 \* 1

MAIN:

li x1, 13

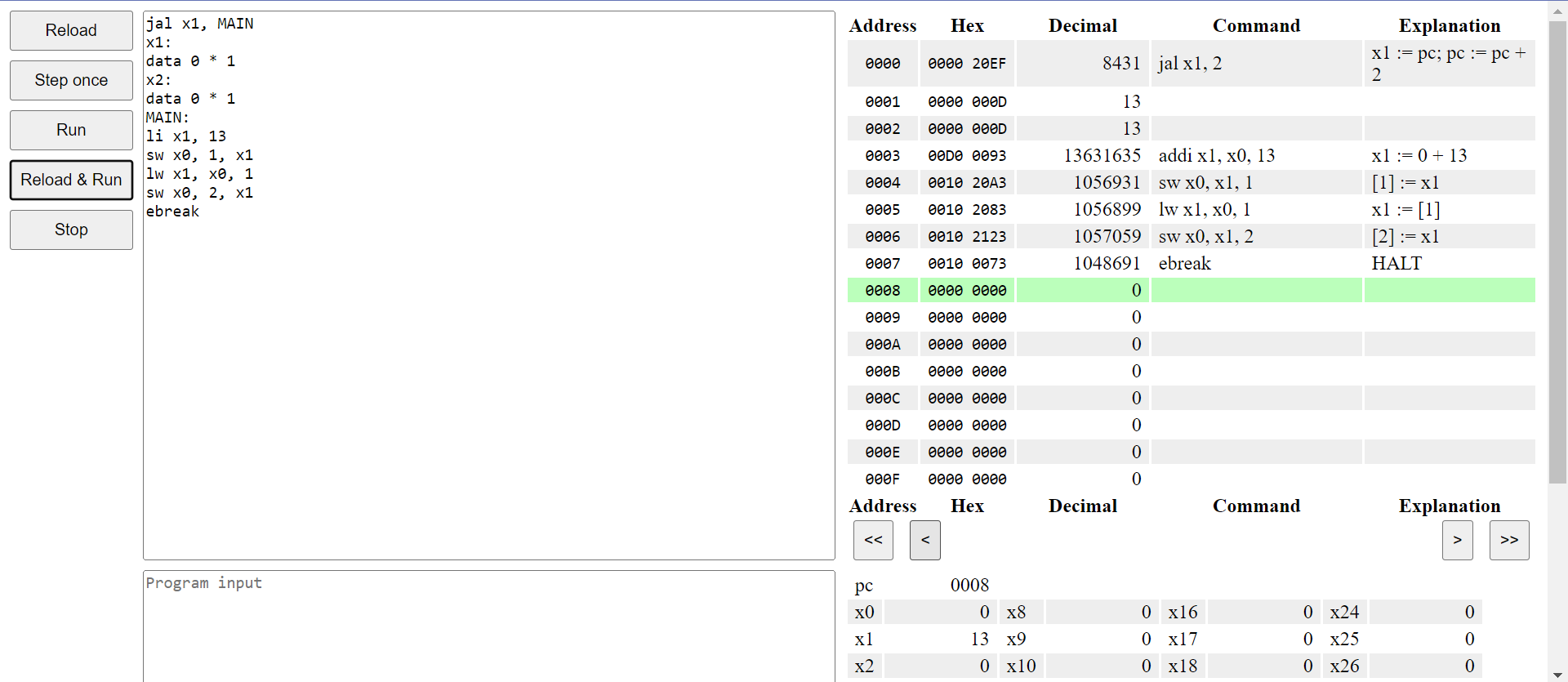
sw x0, 1, x1

lw x1, x0, 1

sw x0, 2, x1

ebreak

*Результат работы для risc-emulator:*

****

**5.2 Ветвление, цикл for**

*Программа имеет вид:*

чирп

начпрог

x=1;

y = 2;

если (x == 1)

нач

x = y;

y = 54;

кон

для (i = 1; i <= 7; i = i + 1)

нач

x = x + i;

кон

конпрог

*Сформированное AST представление:*

STATEMENTS

ASSIGNMENT y

CONSTANT 2

STATEMENTS

IF

COMPARISON ==

NAME x

CONSTANT 1

STATEMENTS

ASSIGNMENT x

NAME y

ASSIGNMENT y

CONSTANT 54

FOR

ASSIGNMENT i

CONSTANT 1

COMPARISON <=

NAME i

CONSTANT 7

ASSIGNMENT i

OPERATOR +

NAME i

CONSTANT 1

ASSIGNMENT x

OPERATOR +

NAME x

NAME i

*Сформированный chirp\_out.S:*

jal x1, MAIN

i:

data 0 \* 1

x:

data 0 \* 1

y:

data 0 \* 1

MAIN:

li x1, 1

sw x0, 2, x1

li x1, 2

sw x0, 3, x1

lw x2, x0, 2

li x3, 1

seq x2, x2, x3

beq x2, x0, ENDIF0

lw x1, x0, 3

sw x0, 2, x1

li x1, 54

sw x0, 3, x1

jal x1, ENDIF0

ENDIF0:

li x1, 1

sw x0, 1, x1

WHILE0:

lw x2, x0, 1

li x3, 7

seq x4, x2, x3

slt x5, x2, x3

or x2, x4, x5

beq x2, x0, ENDWHILE0

lw x1, x0, 2

lw x2, x0, 1

add x1, x1, x2

sw x0, 2, x1

lw x1, x0, 1

li x2, 1

add x1, x1, x2

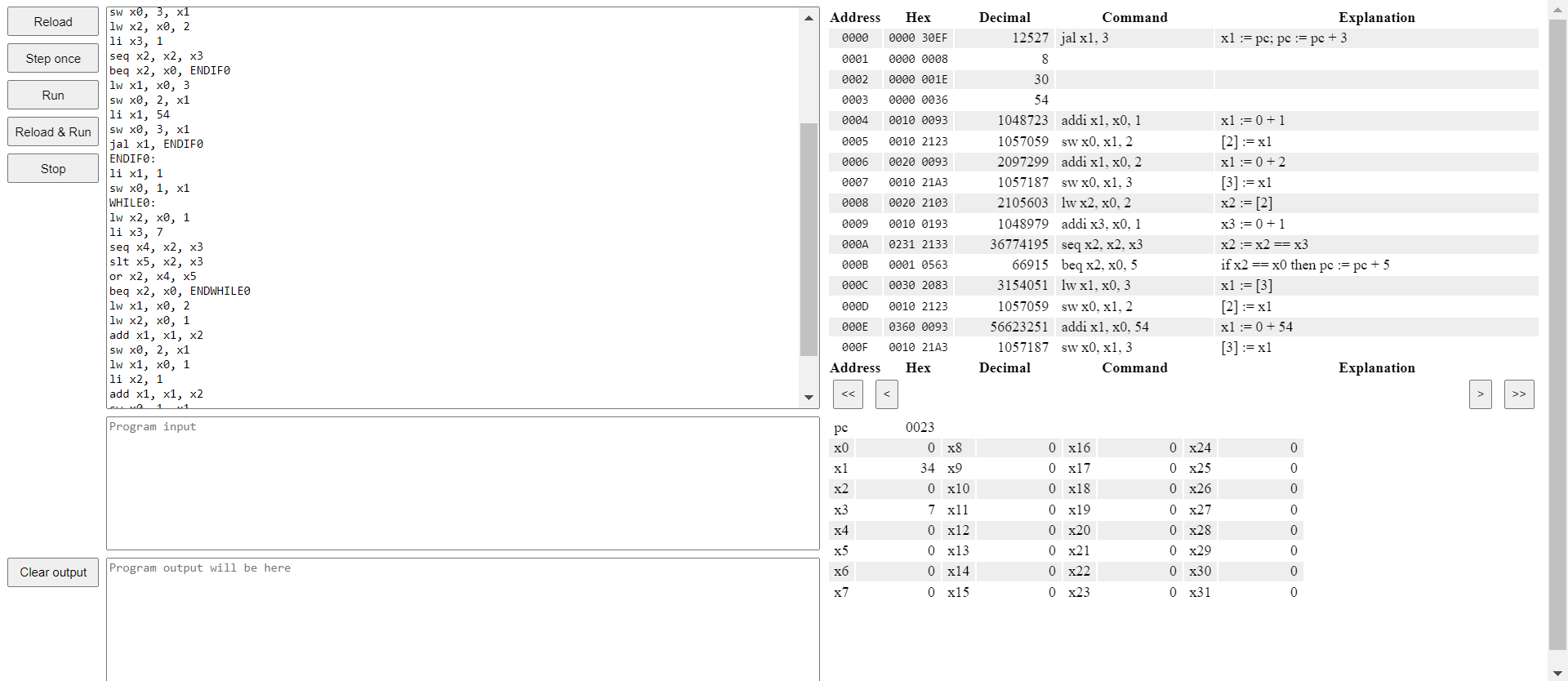
sw x0, 1, x1

jal x1, WHILE0

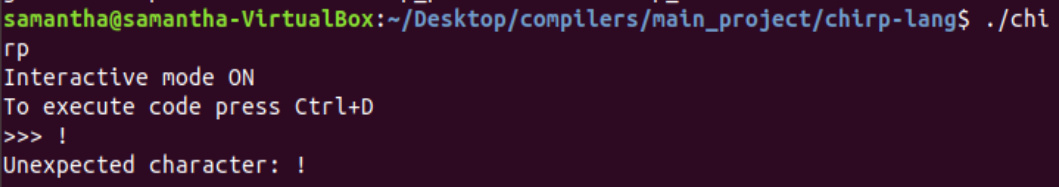
ENDWHILE0:

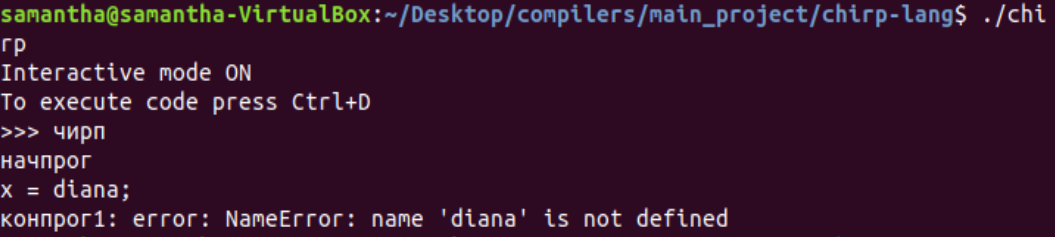
Ebreak

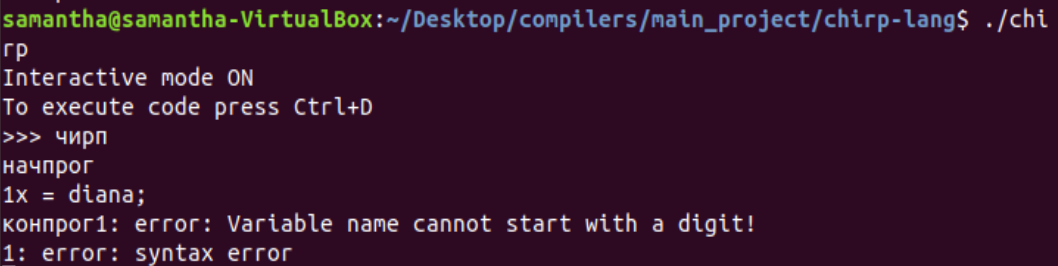
*Результат работы для risc-emulator:*

****

**5.3 Обработка ошибок**

****

****

****

**6. Пример программы**

Язык CHIRP очень уникален. Пример программы, которая включает все возможные инструменты, представлен ниже:

чирп

начпрог

x=1;

y = 2;

если (x == 1)

нач

x = y;

y = 54;

кон

иначе если (x == 4)

нач

y = 87;

кон

иначе

нач

x = 73;

кон

для (i = 1; i <= 7; i = i + 1)

нач

x = x + i;

кон

пока (x >= 19)

нач

y = y - 4;

кон

конпрог

**7. Запуск программы**

Для того, чтобы запустить программу, необходимо выполнить в консоли следующие действия:

git clone <https://github.com/ITSamantha/chirp-lang>

cd chirp-lang

make

./chirp

Программа поддерживает 2 режима:

1. Режим ввода с клавиатуры
2. Режим чтения из файла

Вывод результата работы программы осуществляется в файлы директории:

1. chirp\_out.S – код для risc-emulator
2. chirp\_tree – AST

**8. Выводы**

В ходе работы над итоговым проектом был разработан язык программирования CHIRP. Для него были разработаны парсер с использованием инструмента Bison и лексер с помощью интрумента Flex.