

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  
Р33082  
Преподаватель:  
Лаздин Артур Вячеславович

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
          | arithmetic_operations
          | logical_operations
          | '(' expression ')'
          | INTEGER
          | NAME
          | NAME '=' expression

logical_operations -> expression AND expression
          | expression OR expression
```

	NOT expression
arithmetic_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:*

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

### 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 arithmetic_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

    | arithmetic_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); }

    ;

```

```

    arithmetic_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:

Reload

Step once

Run

Reload & Run

Stop

```
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
```

Program input

Address	Hex	Decimal	Command	Explanation
0000	0000 20EF	8431	jal x1, 2	x1 := pc; pc := pc + 2
0001	0000 000D	13		
0002	0000 000D	13		
0003	00D0 0093	13631635	addi x1, x0, 13	x1 := 0 + 13
0004	0010 20A3	1056931	sw x0, x1, 1	[1] := x1
0005	0010 2083	1056899	lw x1, x0, 1	x1 := [1]
0006	0010 2123	1057059	sw x0, x1, 2	[2] := x1
0007	0010 0073	1048691	ebreak	HALT
0008	0000 0000	0		
0009	0000 0000	0		
000A	0000 0000	0		
000B	0000 0000	0		
000C	0000 0000	0		
000D	0000 0000	0		
000E	0000 0000	0		
000F	0000 0000	0		

Address

Hex

Decimal

Command

Explanation

<<

<

>

>>

pc	0008						
x0	0	x8	0	x16	0	x24	0
x1	13	x9	0	x17	0	x25	0
x2	0	x10	0	x18	0	x26	0

## 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:

Reload

Step once

Run

Reload & Run

Stop

```
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
```

Program input

Clear output

Program output will be here

Address	Hex	Decimal	Command	Explanation
0000	0000 30EF	12527	jal x1, 3	x1 := pc; pc := pc + 3
0001	0000 0008	8		
0002	0000 001E	30		
0003	0000 0036	54		
0004	0010 0093	1048723	addi x1, x0, 1	x1 := 0 + 1
0005	0010 2123	1057059	sw x0, x1, 2	[2] := x1
0006	0020 0093	2097299	addi x1, x0, 2	x1 := 0 + 2
0007	0010 21A3	1057187	sw x0, x1, 3	[3] := x1
0008	0020 2103	2105603	lw x2, x0, 2	x2 := [2]
0009	0010 0193	1048979	addi x3, x0, 1	x3 := 0 + 1
000A	0231 2133	36774195	seq x2, x2, x3	x2 := x2 == x3
000B	0001 0563	66915	beq x2, x0, 5	if x2 == x0 then pc := pc + 5
000C	0030 2083	3154051	lw x1, x0, 3	x1 := [3]
000D	0010 2123	1057059	sw x0, x1, 2	[2] := x1
000E	0360 0093	56623251	addi x1, x0, 54	x1 := 0 + 54
000F	0010 21A3	1057187	sw x0, x1, 3	[3] := x1

<<

<

>

>>

Address	Hex	Decimal	Command	Explanation
pc	0023			
x0	0 x8	0	x16	0 x24
x1	34 x9	0	x17	0 x25
x2	0 x10	0	x18	0 x26
x3	7 x11	0	x19	0 x27
x4	0 x12	0	x20	0 x28
x5	0 x13	0	x21	0 x29
x6	0 x14	0	x22	0 x30
x7	0 x15	0	x23	0 x31

## 5.3 Обработка ошибок

```
samantha@samantha-VirtualBox:~/Desktop/compilers/main_project/chirp-lang$ ./chirp
Interactive mode ON
To execute code press Ctrl+D
>>> !
Unexpected character: !
```

```
samantha@samantha-VirtualBox:~/Desktop/compilers/main_project/chirp-lang$ ./chirp
Interactive mode ON
To execute code press Ctrl+D
>>> чирп
начпрог
x = diana;
коннпрог1: error: NameError: name 'diana' is not defined
```

```
samantha@samantha-VirtualBox:~/Desktop/compilers/main_project/chirp-lang$ ./chirp
Interactive mode ON
To execute code press Ctrl+D
>>> чирп
начпрог
1x = diana;
коннпрог1: error: Variable name cannot start with a digit!
1: error: syntax error
```

## 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 инструмента Flex.