Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**Отчёт по индивидуальной работе №3**

Дисциплина: Транслирующие системы

Тема: Разработка транслятора на языках LEX, YACC

Выполнил студент гр. 5130901/20103 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А. Вагнер

(подпись)

Принял старший преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.Н. Цыган

(подпись)

“ ” 2024 г.

Санкт-Петербург

2024

Текст задания

С использованием утилит LEX и YACC разработать программу транслятора для операторов

IF . . . ELSE языка "С".

Параметры задания:

1) Типы данных – только int (идентификаторы, константы, элементы одномерных массивов).

2) Структура оператора:

if ( Условие )

{

/ / Здесь инструкции, которые будут выполнены,

/ / если значение выражения Условие не равно нулю (т. е. истинно)

}

else

{

/ / Здесь инструкции, которые будут выполнены,

/ / если значение выражения Условие равно нулю

}

3) Операторы "в теле IF" – операторы присваивания вида:

идентификатор = идентификатор ; или

идентификатор = const ; или

идентификатор = бесскобочное логическое выражение ;

Типы операндов – см. пункт 1).

Под бесскобочным логическим выражением здесь понимается логическое выражение с

поразрядными операциями И, ИЛИ, НЕ (синтаксис указанных операторов должен

соответствовать синтаксису языка "С").

Выражение Условие – любое выражение, содержащее два операнда, соединенных операторами

"равно", "не равно", "больше", "больше или равно" и т.п. (синтаксис указанных операторов

должен соответствовать синтаксису языка "С").

4) \* Трансляция операторов объявления переменных по желанию.

5) Напоминаю, что в начале отчета должна быть приведена составленная Вами грамматика (в

БНФ) именно для тех синтаксических конструкций, которые подлежат трансляции в Вашем

задании. Рекомендую вообще начинать выполнение индивидуального задания именно с

составления указанной грамматики (подчеркиваю – для всех подлежащих трансляции

конструкций).

6) Выходной продукт Вашего транслятора - это обобщенные команды в виде строк матрицы синтаксического дерева, то есть абстрактный код.

7) Необходимо предусмотреть не менее двух собственных диагностических сообщений, причем,

очень желательно, чтобы эти сообщения формировались не LEX-программой, а YACCпрограммой.

8)\* Рассмотреть возможность трансляции вложенных операторов IF . . . ELSE .

Модифицировать программу на этот случай или, по крайней мере, указать, какие

программистские проблемы необходимо преодолеть для организации трансляции вложенных

операторов IF . . . ELSE .

Ход работы

В начале разработаем БНФ грамматику будущего транслятора. В нём будут описаны будущие лексемы и правила.

Листинг 1 – БНФ грамматика

<digit>::= 0|1|2|...|9

<letter>::= a|b|...|z|A|...|Z

<identifier>::= <letter>|(<dentifier><digit>|<letter>))

<integer>::= <digit>|(<integer><digit>)

<IF>:: if

<ELSE>:: else

<ASSIGN>::= =

<LBRACE>::= {

<RBRACE>::= }

<LPAREN>::= (

<RPAREN>::= )

<SEMICOLON>::= ;

<CONSTANT\_C>::= '<letter>'

<CONSTANT\_I>::= (<digit>|(<CONSTANT\_I><digit>))

<CONSTANT>:== CONSTANT\_C|CONSTANT\_I

<AROP>::= +|-|/|\*

<INCDEC>::= (++)|(--)

<COMP>::= >|<|(>=)|(<=)|(==)|(!=)

<ARPOINTER>::= <identifier>[<CINSTANT\_I>]

<program>::= <if\_statement>|(<if\_statement><program>)

<if\_statement>::= <IF><condition><body>

<condition>::= <LPAREN><assigned><COMP><assigned><RPAREN>

<body>::= <LBRACE>|(<body><operation>)|(<body><RBRACE>)

<operation>::= <identifier><ASSIGN><arop><SEMICOLON>

<arop>::= (<assigned><AROP><assigned)|(<assigned><INCDEC>)|(<INCDEC><assigned>)

<assigned>::= IDENTIFIER|CONSTANT|ARPOINTER|arop

Далее приведено содержание файла zz.c. Его изменение требуется при необходимости включения режима дебага в программе yacc.

Листинг 2 – Файл zz.c

#include <stdio.h>

**extern** **int** yydebug;

**extern** **int** **yyparse**();

main ()

{

//yydebug = 0;

**return** yyparse();

}

yyerror( **char** \*s )

{

fprintf( stderr, "?-%s**\n**", s );

}

На основе полученной структуры были разработаны файлы на языках lex и yacc.

Листинг 3 – Файл c.l

%{

#include <stdlib.h>

#include "y.tab.h"

%}

digit [**0**-**9**]

letter [a-zA-Z\_]

identifier {letter}({letter}|{digit})\*

integer {digit}+

%%

"if" { **return** IF; }

"else" { **return** ELSE; }

"=" { **return** ASSIGN; }

"{" { **return** LBRACE; }

"}" { **return** RBRACE; }

"(" { **return** LPAREN; }

")" { **return** RPAREN; }

";" { **return** SEMICOLON; }

\'{letter}\' { yylval.str = strdup(yytext); **return** CONSTANT; }

{integer} { yylval.str = strdup(yytext); **return** CONSTANT; }

{identifier} { yylval.str = strdup(yytext); **return** IDENTIFIER; }

(\+|\-|\/|\\*) { yylval.str = strdup(yytext); **return** AROP;}

((\+\+)|(\-\-)) { yylval.str = strdup(yytext); **return** INCDEC;}

(>|<|>=|<=|==|!=) { yylval.str = strdup(yytext); **return** COMP;}

{identifier}\[{digit}+\] { yylval.str = strdup(yytext); **return** ARPOINTER; }

[ \t\n] ; /\* пропускаем пробелы и символы новой строки \*/

. ;

%%

#ifndef yywrap

**int** yywrap () { **return** **1**; }

#endif

Листинг 4 – Файл c.y

%{

#include <stdlib.h>

#include <string.h>

#include "y.tab.h"

#include "malloc.h"

#define TRUE 1

#define FALSE 0

**int** cur = **0**;

**int** blockCount = **0**;

**int** l = **0**;

**int** r = **0**;

**char**\* operator;

**int**\* arr;

**int** len = **0**;

%}

%**union** {

**char**\* str;

**int** num;

};

%token <str> IDENTIFIER AROP INCDEC ARPOINTER COMP CONSTANT

%token IF ELSE ASSIGN LBRACE RBRACE LPAREN RPAREN SEMICOLON

%type <num> arop assigned

%start PROGRAM

%%

**PROGRAM:** if\_statement

|if\_statement PROGRAM

;

**if\_statement:**

IF condition body

;

**condition:**

LPAREN assigned COMP assigned RPAREN {

l = $<str>**2**;

operator = $<str>**3**;

r = $<str>**4**;

}

| LPAREN arop RPAREN {

printf("ERROR: Wrong syntax**\n**");

exit(-**1**);

}

| LPAREN assigned RPAREN {

printf("ERROR: Wrong syntax**\n**");

exit(-**1**);

}

| LPAREN RPAREN {

printf("ERROR: Wrong syntax**\n**");

exit(-**1**);

}

;

**body:**

LBRACE {

printf("**\n**EXECUTE BLOCK %d IF R%d %s R%d == TRUE**\n**", blockCount, l, operator, r);

**if** (len == **0**) {

arr = (**int**\*) malloc(**sizeof**(**int**));

arr[**0**] = blockCount;

}

**if** (blockCount >= len) {

arr = (**int**\*) realloc (arr, (len + **1**) \* **sizeof**(**int**));

}

arr[blockCount + **1**] = cur;

printf("<START OF BLOCK %d>**\n**", blockCount++);

}

| body operation

| body RBRACE {

**if** (blockCount == **1**) {

cur = arr[**0**];

} **else** **if** (blockCount != **0**) {

cur = arr[blockCount];

}

printf("<END OF BLOCK %d>**\n\n**", --blockCount);

}

;

**operation:** IDENTIFIER ASSIGN arop SEMICOLON {

printf("SET %s = R%d**\n**", $**1**, $**3**);

}

| CONSTANT ASSIGN arop SEMICOLON {

printf("ERROR: trying to change constant value");

exit(-**1**);

}

| ARPOINTER ASSIGN arop SEMICOLON {

printf("ERROR: can't change array elements");

exit(-**1**);

}

;

**arop:** assigned AROP assigned {

$$ = cur;

printf("R%d = R%d %s R%d**\n**", cur++, $**1**, $**2**, $**3**);

}

| assigned INCDEC {

**char**\* op = $**2**;

$$ = cur;

**switch**(op[**0**]) {

**case** '+':

printf("R%d = R%d**\n**", cur, $**1**);

printf("R%d = R%d + 1**\n**", $**1**, $**1**);

**break**;

**case** '-':

printf("R%d = R%d**\n**", cur, $**1**);

printf("R%d = R%d - 1**\n**", $**1**, $**1**);

**break**;

cur++;

}

}

| INCDEC assigned {

**char**\* op = $<str>**1**;

$$ = cur;

**switch**(op[**0**]) {

**case** '+':

printf("R%d = R%d + 1**\n**", $**2**, $**2**);

printf("R%d = R%d**\n**", cur, $**2**);

**break**;

**case** '-':

printf("R%d = R%d - 1**\n**", $**2**, $**2**);

printf("R%d = R%d**\n**", cur, $**2**);

**break**;

}

cur++;

}

;

**assigned:** IDENTIFIER {

$$ = cur;

printf("R%d = GET %s**\n**", cur++, $**1**);

}

| CONSTANT {

$$ = cur;

printf("R%d = %s**\n**", cur++, $**1**);

}

| ARPOINTER {

$$ = cur;

**char**\* string = $**1**;

**char**\* name = (**char**\*) malloc(**sizeof**(**char**));

**int** i = **0**;

**while** (string[i] != '[') {

name = (**char**\*) realloc(name, (i + **1**) \* **sizeof**(**char**));

name[i] = string[i];

i++;

}

i++;

**int** pointer = **0**;

**char**\* arrSize = (**char**\*) malloc(**sizeof**(**char**));

**while** (string[i] != ']') {

arrSize = (**char**\*) realloc(arrSize, (pointer + **1**) \* **sizeof**(**char**));

arrSize[pointer] = string[i];

i++;

}

printf("R%d = GETFROM %s %s ELEMENT**\n**", cur++, name, arrSize);

}

| arop { $$ = cur - **1**;}

;

%%

**Общий синтаксис и структура**

1. **Получение данных (GET и GETFROM)**:
   * R0 = GET a: Получение значения из какого-то источника (например, переменной a) и сохранение его в регистре R0.
   * R2 = GETFROM r I ELEMENT: Получение значения из структуры данных (например, массива) на основе индекса I и сохранение в регистре R2.
2. **Арифметические операции**:
   * Простые операции сложения и вычитания:
     + R3 = R0 + R1: Сложение значений из регистров R0 и R1, результат записывается в R3.
     + R11 = R11 - 1: Вычитание единицы из значения в регистре R11.
3. **Условные блоки с переходами**:

* Код использует условные конструкции для выполнения блоков кода в зависимости от значений регистров.
  + EXECUTE BLOCK 0 IF R0 > R1 == TRUE
* Это означает, что блок с номером 0 будет выполнен, если значение в R0 больше, чем в R1. Если условие истинно, происходит переход (goto) к началу блока <START OF BLOCK 0>. Если условие ложно, выполняется переход к <END OF BLOCK 0>, пропуская все инструкции внутри блока.

1. **Структура блока**:

Каждый блок имеет следующую структуру:

<START OF BLOCK X>

... // инструкции

<END OF BLOCK X>

Например:

assembly

Copy code

<START OF BLOCK 0>

R4 = R2 + R3

SET c = R4

<END OF BLOCK 0>

Внутри блока могут быть присваивания, арифметические операции и использование инструкций SET.

1. **Инструкции SET**:
   * SET используется для сохранения значения регистра в переменную:

SET a = R8

Это означает, что значение из регистра R8 сохраняется в переменную a.

1. **Инициализация регистров**:
   * Перед выполнением операций регистры могут быть инициализированы с помощью прямого присваивания:

R0 = 1

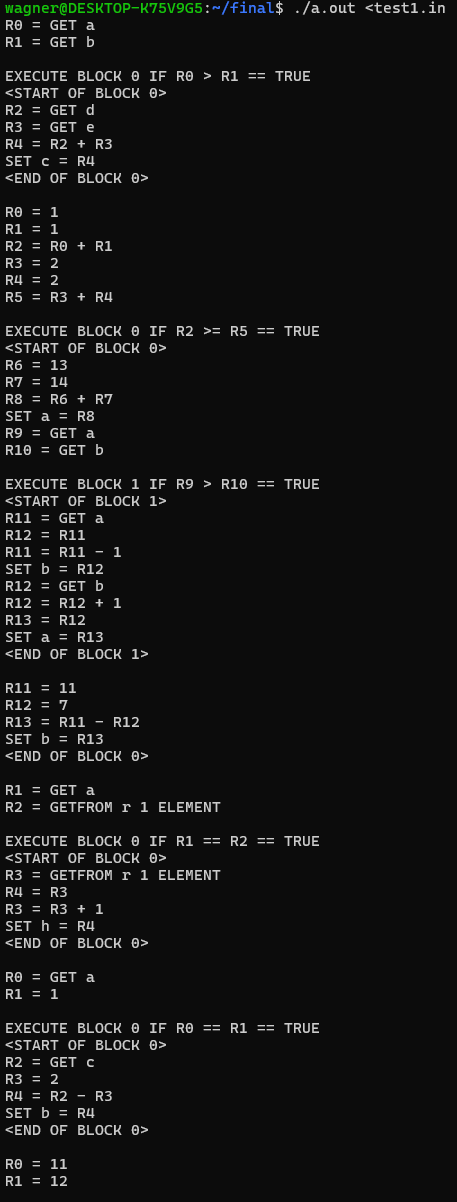
R1 = 1

Далее протестируем работу программы на разных наборах входных данных, эти тесты следует написать так, чтобы были использованы все возможные лексемы и правила.

Листинг 4 – Содержание тестовых данных test1.in

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Листинг 5 – Результат тестирования транслятора Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

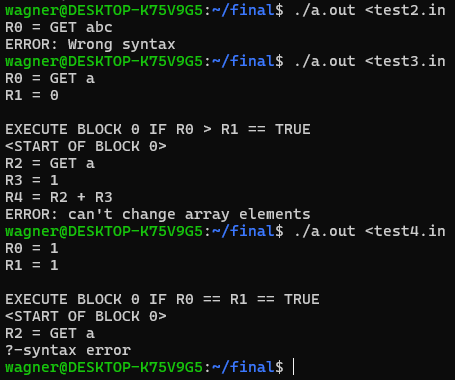
Также следует проверить вывод сообщений об ошибке, напишем ещё несколько тестов с неверными входными данными.

Листинг 6 – Тестирование обнаружения неверного синтаксиса

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Листинг 7 – Результат тестирования



Результат соответствует ожидаемому следовательно программа соответствует спецификации.

Вывод

В ходе выполнения индивидуального задания на языках lex и yacc был написан элементарный транслятор. В соответствии с заданием, транслятор переводит операторы If else (в том числе вложенные) в псевдокод.