Министерство образования и науки Российской Федерации
Федеральное государственное образовательное учреждение
высшего образования
«Национальный исследовательский университет «МЭИ»
Институт ИВТ Кафедра ПМИИ

Отчет по лабораторной работе номер 4
Восходящий синтаксический анализ

Подгтовил: Желтиков Александр Алексеевич
Дата: 2 сентября 2022 г.

Содержание

1 Постановка задачи и ее условие.

Задание. С использованием синтаксического анализатора Yacc разработать и реализовать программу восходящего синтаксического анализа.

Номер варианта по журналу → 10

Во всех вариантах символ S является начальным символом грамматики; S, F, T и E обозначают нетерминальные символы. Терминальные символы выделены жирным шрифтом. Вместо терминального символа **a** должны подставляться лексемы - идентификаторы, римские числа.

S → F;
F → for T do F | a := a
T → (F; E; F) | (; E; F) | (F; E;) | (; E;)
E → **a**<**a** | **a**>**a** | **a**=**a**

2 Часть задания номер 1.

2.1 Правила языка в форме БНФ.

<S> ::= <F>
<F> ::= **for** <T> **do** <F> | <**id**> := <**exp**>
<T> ::= (<F>; <E>; <F>) | (; <E>; <F>) | (<F>; <E>;) | (; <E>;)
<E> ::= <**exp**> < <**exp**> | <**exp**> > <**exp**> |<**exp**> = <**exp**>
<id>::= <letter> | <letter><number><id> | <letter><id>
<exp>::= <value> | <id>
<value>::=<number>
<letter> ::= a | b | c | ... | z | A | B | C | ... | Z
<roman> ::= I | X | V
<number> ::= <roman> | <roman><number>

2.2 Описание грамматики языка.

G = (T, N, P, Z)

Множество терминальных символов T:

{ a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, :=, =, <, >, “;” , “(“ , “)” }

Множество не терминальных символов N:

{ <S>, <F>, <T>, <E>, <id>, <exp>, <value>, <letter>, <roman>, <number>, }

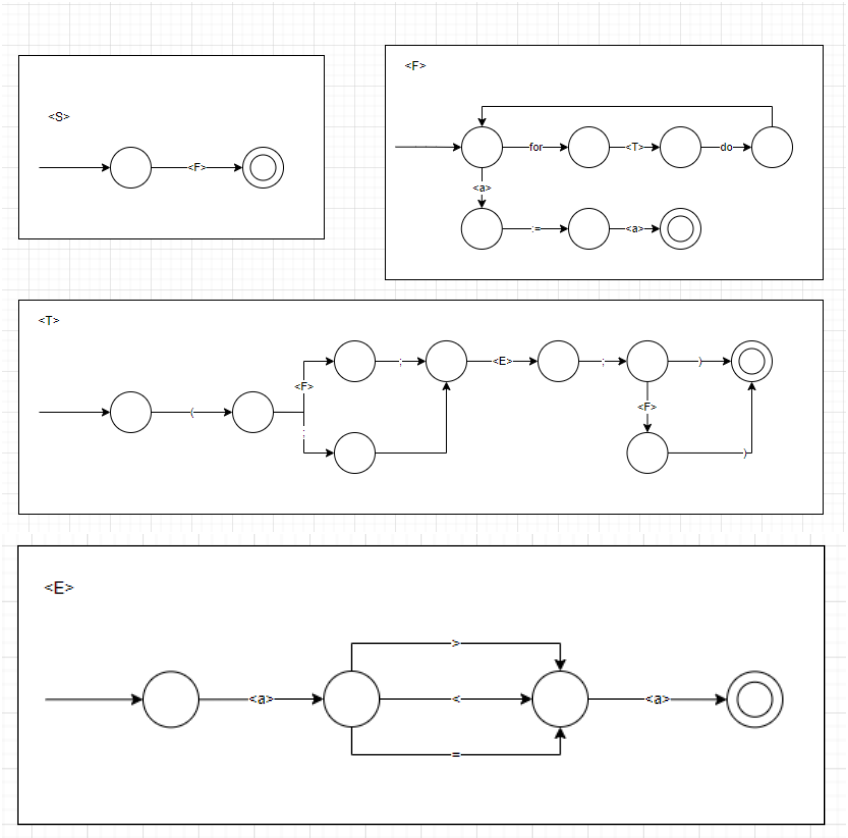
Правила грамматики P:

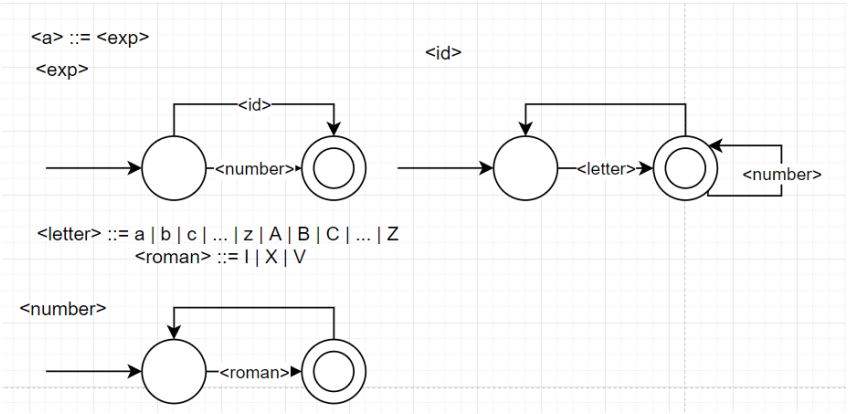
<S> → <F>
<F> → **for** <T> **do** <F> | <**id**> := <**exp**>
<T> → (<F>; <E>; <F>) | (; <E>; <F>) | (<F>; <E>;) | (; <E>;)
<E> → <**exp**> < <**exp**> | <**exp**> > <**exp**> |<**exp**> = <**exp**>
<id>→ <letter> | <letter><number><id> | <letter><id>
<exp>→ <number> | <id>
<letter> → a | b | c | ... | z | A | B | C | ... | Z
<roman> → I | X | V
<number> → <roman> | <roman><number>

Аксиома Z:

Z = S

2.3 ДКА





3 Средства для синтаксического анализа в среде уасс.

уасс — компьютерная программа, служащая стандартным генератором синтаксических анализаторов (парсеров) в Unix-системах. Название является акронимом «Yet Another Compiler Compiler» («ещё один компилятор компиляторов»). Yacc генерирует парсер на основе аналитической грамматики, описанной в нотации BNF (форма Бэкуса-Наура) или контекстно-свободной грамматики. На выходе уасс выдаётся код парсера на языке программирования Си.

Поскольку парсер, генерируемый с помощью уасс, требует использования лексического анализатора, то часто он используется совместно с генератором лексических анализаторов, в большинстве случаев это lex либо flex.

Для того что бы написать алгоритм, мы можем использовать специальное ПО (Flex Windows), которое за нас может скомпелировать файлы, или сделать все через текстовый редактор создав файлы с расширением .y и .l

Парсер, созданный Yaccом, состоит из машины конечных состояний со стеком. Парсер также способен читать и запоминать следующий входной токен (LT - lookahead token). Текущее состояние - всегда на вершине стека. Состояниям машины конечных состояний присваиваются небольшие целые метки; изначально машина находится в состоянии 0, и не прочитано ни одного LT.

Машине доступно только четыре действия, называемые сдвиг, понижение, принятие и ошибка. Каждый шаг парсера происходит следующим образом:

1. Основываясь на текущем состоянии парсер определяет, нужен ли ему LT для решения, какое действие нужно произвести; если ему требуется LT и он его не имеет, то вызывает yylex для получения следующего токена.
2. Используя текущее состояние и, если необходимо, LT парсер принимает решение о следующем действии и производит его. В результате состояния могут быть записаны в стек или прочитаны из него или LT обработан или оставлен.

Далее при написании кода в уасс нам нужно знать, что функция, произведенная Yacc-ом, называется yyparse; это функция типа int. Когда она вызывается, то в свою очередь постоянно вызывает yylex, лексический анализатор, предоставляемый пользователем для получения входных токенов. В конце концов либо обнаруживается ошибка - в этом случае yyparse возвращает значение 1, или лексический анализатор возвращает токен конца ввода и парсер совершает действие "принять". В этом случае yyparse возвращает значение 0.

Пользователь должен обеспечить определеную среду для парсера, чтобы получить работающую программу. Например, в каждой программе на C должна быть определена функция main, которая в конечном счете вызывает yyparse. Вдобавок, функция yyerror печатает сообщение при обнаружении синтаксической ошибки.

4 Тестирование алгоритма.

Тест без ошибок:
test.txt: *for (i := I ; i < X ; i := I) do roman := II*
Console output: WELL DONE

Тест ошибкой в for:
test.txt: *fok (i := I ; i < X ; i := I) do roman := II*
Console output: syntax error in line: 1

Тест ошибкой в do:
test.txt: *for (i := I ; i < X ; i := I) da roman := II*
Console output: syntax error in line: 1

Тест с ошибкой скобочной последовательности:
test.txt: *for)i := I ; i < X ; i := I(do roman := II*
Console output: syntax error in line: 1

Тест с ошибкой внутри for:
test.txt: *for (i > I ; i < X ; i := I) do roman := II*
Console output: syntax error in line: 1

Тест с ошибкой после for:
test.txt: *for (i := I ; i < X ; i := I) do roman = II*
Console output: syntax error in line: 1

5 Алгоритм синтаксического анализа.

yacc

```
%{
        #include <stdio.h>
        extern int yylex();
        extern FILE* yyin;
        extern int yylineno;
        extern char* yytext;
        void yyerror(const char* msg);
%}
// Запишем стартовый нетерминал
%start S

// Запишем токены которые мы используем
%token FORBRA FORKET LETTER NUMBER MOREID ID OPERATOR ASSIGN BRA KET SEPARATOR
```

```
// Опишем наши правила
%%
S : F;

F : FORBRA T FORKET F
        | ID ASSIGN ID
        | ID ASSIGN NUMBER;

T : BRA F SEPARATOR E SEPARATOR F KET
        | BRA SEPARATOR E SEPARATOR F KET
        | BRA F SEPARATOR E SEPARATOR KET
        | BRA SEPARATOR E SEPARATOR KET;

E : ID OPERATOR ID
        | NUMBER OPERATOR ID
        | NUMBER OPERATOR NUMBER
        | ID OPERATOR NUMBER;
%%
// Функция которая покажет в какой строке ошибка
void yyerror(const char* msg) {
        printf("%s in line: %d \n", msg, yylineno);
}
int main()
{
   yyin = fopen("test.txt", "r");
   // Проверка входного файла на корректность
   if (yyin == NULL){
        printf("\nWE CAN'T OPEN TEST FILE!.\n");
        return -1;
   }
   // Сравнение с 0 дает нам положительный результат проверки анализатора.
   if (yyparse() == 0)
        printf("WELL DONE");
   fclose(yyin);
   return 0;
}
```

lex

```
%option noyywrap
%option yylineno
%{
        #include<stdio.h>
        #include "y.tab.h" // Библиотека генерируемая при компиляции yacc
        extern void yyerror(const char* msg);
%}
// Описание токенов
FORBRA "for"
FORKET "do"
LETTER [_a-zA-Z]
NUMBER [IVX]+
MOREID {LETTER}|{NUMBER}
ID {LETTER}{MOREID}*
OPERATOR "<"|">"|"="
ASSIGN ":="
BRA "("
KET ")"
SEPARATOR ";"
%%
```

```
[ \t\n]
{FORBRA}        {return (FORBRA); }
{FORKET}        {return (FORKET); }
{NUMBER}        {return (NUMBER); }
{ID}               {return (ID); }
{OPERATOR}       {return (OPERATOR); }
{ASSIGN}        {return (ASSIGN); }
{BRA}              {return (BRA); }
{KET}              {return (KET); }
{SEPARATOR}      {return (SEPARATOR); }
.                  {printf("This is indefinite token! \n\t\"%s\"\n", yytext ,yylineno );}
%%
```