МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Факультет автоматики и вычислительной техники

Кафедра вычислительной техники



**ОТЧЁТ**

**по лабораторной работе №4**

**«Синтаксис языков программирования. Нисходящий синтаксический анализ»**

по дисциплине: «Теория формальных языков и компиляторов»

Вариант № 144211511, файл Lab4

Выполнил:Проверил:

студент гр. АВТ-918 Малявко А.А.

Ванин Константин «\_\_\_» \_\_\_\_\_\_\_\_\_ 2022 г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка, подпись)

Новосибирск

2022

# Цель работы

Изучить основные идеи и понятия нисходящих методов синтаксического анализа, выявления свойств формальных грамматик, необходимых для реализации нисходящего восстановления дерева грамматического разбора, приобрести навыки построения процедурной и различных автоматных реализаций нисходящего анализа, исследовать поведение нисходящих синтаксических акцепторов.

# Ход работы

## Реализация LL(1)-грамматики для учебного языка программирования:

Видимые/редактируемые:

program func +

func ident "=" ( type | "void" ) "(" ( ident ( "," type ident ) \* ) ? ")" "start" operator + "end"

type "int" | "real" | "char"

operator "put" expr "to" ident ";"

operator "if" "(" expr ")" "then" blockoroperator partnot

operator "exec" blockoroperator "with" ident "from" const "to" const ( "step" const ) ?

operator type ( "put" const "to" ) ? ident ( "," ( "put" const "to" ) ? ident ) \* ";"

operator "fin" ";"

operator "choice" expr ("option" ( minus ) ? const ":" blockoroperator) + ( "nooption" blockoroperator ) ? "end"

operator "return" expr ";"

expr ( minus ) ? begexpr ( endexpr ) ?

blockoroperator operator | ( "{" operator + "}" )

partnot "not" blockoroperator

partnot ~ "not"

begexpr ident ( "(" expr ")" ) ?

begexpr const

begexpr "(" expr ")"

const constdec | constreal | constexh | constchar

endexpr sign expr

sign oper | minus

ident [a-z] [0-9] {1,4} [a-zA-Z] +

minus [-]

constdec [0-9] +

constreal [0-9] + [.] [0-9] \*

constexh [0-9] + ( [.] [0-9] \* ) ? [e] [-] ? [0-9] +

constchar ["] [] ["]

oper ([-+/\*])|([!=][=])|([<>][=]?)

space [ \n\r\t] + {ignoreLastWord=true;}

comment [%][]\*[\n\r]{ignoreLastWord=true;}

keyword [a-z]+

switchword [:]

Множества выбора правил разработанной LL(1) грамматики представлены в таблице 1.

Таблица 1 – Множества выбора правил разработанной грамматики

|  |  |
| --- | --- |
| Правило | Множество выбора: |
| **program**: program!0 program!1 | ident |
| **program!0**: ident "=" func!0 "(" func!2 ")" "start" func!3 func!4 "end" | ident |
| **program!1**: program!0 program!1 | ident |
| **program!1**: | EOF |
| **func!0**: type | *"char"*  *"int"*  *"real"* |
| **func!0**: "void" | *"void"* |
| **func!2**: ident func!1 | ident |
| **func!2**: | *")"* |
| **func!3**: "put" expr "to" ident ";" | *"put"* |
| **func!3**: "if" "(" expr ")" "then" blockoroperator partnot | *"if"* |
| **func!3**: "exec" blockoroperator "with" ident "from" const "to" const operator!0 | *"exec"* |
| **func!3**: type operator!1 ident operator!3 ";" | *"char"*  *"int"*  *"real"* |
| **func!3**: "fin" ";" | *"fin"* |
| **func!3**: "choice" expr operator!5 operator!7 operator!6 "end" | *"choice"* |
| **func!3**: "return" expr ";" | *"return"* |
| **func!4**: func!3 func!4 | *"char"*  *"choice"*  *"exec"*  *"fin"*  *"if"*  *"int"*  *"put"*  *"real"*  *"return"* |
| **func!4**: | *"end"* |
| **type**: "int" | *"int"* |
| **type**: "real" | *"real"* |
| **type**: "char" | *"char"* |
| **func!1**: "," type ident func!1 | *","* |
| **func!1**: | *")"* |
| **expr**: expr!0 begexpr expr!1 | minus  ident  constdec  constreal  constexh  constchar  *"("* |
| **blockoroperator**: "put" expr "to" ident ";" | *"put"* |
| **blockoroperator**: "if" "(" expr ")" "then" blockoroperator partnot | *"if"* |
| **blockoroperator**: "exec" blockoroperator "with" ident "from" const "to" const operator!0 | *"exec"* |
| **blockoroperator**: type operator!1 ident operator!3 ";" | *"char"*  *"int"*  *"real"* |
| **blockoroperator**: "fin" ";" | *"fin"* |
| **blockoroperator**: "choice" expr operator!5 operator!7 operator!6 "end" | *"choice"* |
| **blockoroperator**: "return" expr ";" | *"return"* |
| **blockoroperator**: "{" blockoroperator!1 blockoroperator!2 "}" | *"{"* |
| **partnot**: "not" blockoroperator | *"not"* |
| **partnot**: "not" | *"char"*  *"choice"*  *"exec"*  *"fin"*  *"if"*  *"int"*  *"put"*  *"real"*  *"return"*  *"option"*  *"nooption"*  *"end"*  *"with"*  *"}"* |
| **const**: constdec | constdec |
| **const**: constreal | constreal |
| **const**: constexh | constexh |
| **const**: constchar | constchar |
| **operator!0**: "step" const | *"step"* |
| **operator!0**: | *"char"*  *"choice"*  *"exec"*  *"fin"*  *"if"*  *"int"*  *"put"*  *"real"*  *"return"*  *"not"*  *"option"*  *"nooption"*  *"end"*  *"with"*  *"}"* |
| **operator!1**: "put" const "to" | *"put"* |
| **operator!1**: | ident |
| **operator!3**: "," operator!2 ident operator!3 | *","* |
| **operator!3**: | *";"* |
| **operator!5**: "option" operator!4 const ":" blockoroperator | *"option"* |
| **operator!7**: operator!5 operator!7 | *"option"* |
| **operator!7**: | *"nooption"*  *"end"* |
| **operator!6**: "nooption" blockoroperator | *"nooption"* |
| **operator!6**: | *"end"* |
| **expr!0**: minus | minus |
| **expr!0**: | ident  constdec  constreal  constexh  constchar  *"("* |
| **begexpr**: ident begexpr!0 | ident |
| **begexpr**: const | constdec  constreal  constexh  constchar |
| **begexpr**: "(" expr ")" | *"("* |
| **expr!1**: endexpr | minus  oper |
| **expr!1**: | *"option"*  *")"*  *";"*  *"to"* |
| **operator!2**: "put" const "to" | *"put"* |
| **operator!2**: | ident |
| **operator!4**: minus | minus |
| **operator!4**: | constdec  constreal  constexh  constchar |
| **begexpr!0**: "(" expr ")" | *"("* |
| **begexpr!0**: | *"option"*  minus  oper  *")"*  *";"*  *"to"* |
| **endexpr**: sign expr | minus  oper |
| **blockoroperator!1**: "put" expr "to" ident ";" | *"put"* |
| **blockoroperator!1**: "if" "(" expr ")" "then" blockoroperator partnot | *"if"* |
| **blockoroperator!1**: "exec" blockoroperator "with" ident "from" const "to" const operator!0 | *"exec"* |
| **blockoroperator!1**: type operator!1 ident operator!3 ";" | *"char"*  *"int"*  *"real"* |
| **blockoroperator!1**: "fin" ";" | *"fin"* |
| **blockoroperator!1**: "choice" expr operator!5 operator!7 operator!6 "end" | *"choice"* |
| **blockoroperator!1**: "return" expr ";" | *"return"* |
| **blockoroperator!2**: blockoroperator!1 blockoroperator!2 | *"char"*  *"choice"*  *"exec"*  *"fin"*  *"if"*  *"int"*  *"put"*  *"real"*  *"return"* |
| **blockoroperator!2**: | *"}"* |
| **sign**: oper | oper |
| **sign**: minus | minus |

## Описание учебного языка

1. Идентификаторы: <б><пЦ><пБ> - одна маленькая буква; последовательность цифр длины от 1 до 4, непустая последовательность букв. Примеры: a1928A, b0CD;

2. Константы:

целые по основанию 10 - представляет собой число любой длины, состоящее из цифр 0-9. Примеры: 0, 25;

вещественные - представляет собой число любой длины, состоящее из цифр 0-9, целая и дробная части которой разделены точкой. Примеры: 0., 0.125;

экспоненциальные - представляет собой число любой длины, состоящее из цифр 0-9, экспоненциальная часть которой начинается с буквы e. Отрицательный показатель начинается со знака минус. Примеры: 10e2, 10e-2, 0.1e-2;

символьные – представляет собой любой символ, заключенный в кавычки. Пример: “c”;

3. Знаки операций: -, +, /, \*;

4. Знаки сравнения: !=, ==, >, >=, <, <=;

5. Разделителем является знак «;»;

6. Комментарии начинаются с символа % и могут быть только однострочными;

7. Оператор присваивания задаётся следующим образом: **put** <В> **to**<И>**;**, где < В> – произвольное выражение, <И> – идентификатор, жирным шрифтом помечены ключевые слова;

8. Условный присваивания задаётся следующим образом: **if (**<ЛВ> **)** **then**<ОБ>**[ not** <ОБ> **]** где <ЛВ> – логическое выражение, <ОБ>**–** оператор или блок операторов. В квадратных скобках выделена часть конструкции, которая не является обязательной;

9. Оператор цикла задаётся следующим образом: **exec**<ОБ>**with**<И> **from** <К> **to** <К>**[ step**<K> **]** где <К> – константа;

10. Оператор переключателя задаётся следующим образом: **choice**<В>**option** <К> **:** <ОБ> **[ fin; ]** **[ option … ] … [ nooption** <ОБ> **]** **end**, где троеточие следует за конструкцией, которое можно повториться неограниченное число раз;

11. Объявление функций происходит следующим образом:  **[**<Тип>**] ( [**<АргЛист>**] )**<Блок> где <Тип>**–** ключевое слово типа, <АргЛист> – список формальных аргументов функции, <Блок> – блок операторов.

**Фрагменты историй работы нисходящего синтаксического акцептора**

Приведём фрагменты истории работы для конечного автомата со стековой памятью и несколькими состояниями, для конечного автомата со стековой памятью и одним состоянием и для программной реализации нисходящего рекурсивного спуска.

В качестве правильного тестового примера возьмём следующий код:

a1A = void() start put 0 to a1B; end

В качестве ошибочного:

a1A = void(put 0 to a1C) start put 0 to a1B; end

Истории работы представлены в таблицах 2-7.

Таблица 2. История работы автоматной реализации (конечный автомат с одним состоянием) нисходящего синтаксического акцептора для правильного тестового примера

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | a1A | -1 | 0 |
| 1 | a1A | -2 | -3 | 0 |
| 2 | a1A | 1 | 13 | -4 | 8 | -5 | 9 | 28 | -6 | -7 | 16 | -3 | 0 |
| 3 | = | 13 | -4 | 8 | -5 | 9 | 28 | -6 | -7 | 16 | -3 | 0 |  |
| 4 | void | -4 | 8 | -5 | 9 | 28 | -6 | -7 | 16 | -3 | 0 |  |  |
| 5 | void | 32 | 8 | -5 | 9 | 28 | -6 | -7 | 16 | -3 | 0 |  |  |
| 6 | ( | 8 | -5 | 9 | 28 | -6 | -7 | 16 | -3 | 0 |  |  |  |
| 7 | ) | -5 | 9 | 28 | -6 | -7 | 16 | -3 | 0 |  |  |  |  |
| 8 | ) | 9 | 28 | -6 | -7 | 16 | -3 | 0 |  |  |  |  |  |
| 9 | start | 28 | -6 | -7 | 16 | -3 | 0 |  |  |  |  |  |  |
| 10 | put | -6 | -7 | 16 | -3 | 0 |  |  |  |  |  |  |  |
| 11 | put | 25 | -10 | 31 | 1 | 12 | -7 | 16 | -3 | 0 |  |  |  |
| 12 | 0 | -10 | 31 | 1 | 12 | -7 | 16 | -3 | 0 |  |  |  |  |
| 13 | 0 | -20 | -21 | -22 | 31 | 1 | 12 | -7 | 16 | -3 | 0 |  |  |
| 14 | 0 | -21 | -22 | 31 | 1 | 12 | -7 | 16 | -3 | 0 |  |  |  |
| 15 | 0 | -13 | -22 | 31 | 1 | 12 | -7 | 16 | -3 | 0 |  |  |  |
| 16 | 0 | 2 | -22 | 31 | 1 | 12 | -7 | 16 | -3 | 0 |  |  |  |
| 17 | to | -22 | 31 | 1 | 12 | -7 | 16 | -3 | 0 |  |  |  |  |
| 18 | to | 31 | 1 | 12 | -7 | 16 | -3 | 0 |  |  |  |  |  |
| 19 | a1B | 1 | 12 | -7 | 16 | -3 | 0 |  |  |  |  |  |  |
| 20 | ; | 12 | -7 | 16 | -3 | 0 |  |  |  |  |  |  |  |
| 21 | end | -7 | 16 | -3 | 0 |  |  |  |  |  |  |  |  |
| 22 | end | 16 | -3 | 0 |  |  |  |  |  |  |  |  |  |
| 23 |  | -3 | 0 |  |  |  |  |  |  |  |  |  |  |
| 24 |  | 0 |  |  |  |  |  |  |  |  |  |  |  |

Таблица 3. История работы автоматной реализации (конечный автомат с одним состоянием) нисходящего синтаксического акцептора для ошибочного тестового примера

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | a1A | -1 | 0 |
| 1 | a1A | -2 | -3 | 0 |
| 2 | a1A | 1 | 13 | -4 | 8 | -5 | 9 | 28 | -6 | -7 | 16 | -3 | 0 |
| 3 | = | 13 | -4 | 8 | -5 | 9 | 28 | -6 | -7 | 16 | -3 | 0 |  |
| 4 | void | -4 | 8 | -5 | 9 | 28 | -6 | -7 | 16 | -3 | 0 |  |  |
| 5 | void | 32 | 8 | -5 | 9 | 28 | -6 | -7 | 16 | -3 | 0 |  |  |
| 6 | ( | 8 | -5 | 9 | 28 | -6 | -7 | 16 | -3 | 0 |  |  |  |
| 7 | put | -5 | 9 | 28 | -6 | -7 | 16 | -3 | 0 |  |  |  |  |

Таблица 4. История работы автоматной реализации (конечный автомат с несколькими состояниями) нисходящего синтаксического акцептора для правильного тестового примера

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | a1A | 0 |  |  |  |  |
| 1 | a1A | 73 |  |  | s |  |
| 2 | a1A | 1 |  |  |  |  |
| 3 | a1A | 75 | a |  |  |  |
| 4 | = | 76 | a |  |  |  |
| 5 | void | 77 |  |  | s |  |
| 6 | void | 4 |  |  |  | e |
| 7 | void | 5 |  |  |  |  |
| 8 | void | 89 | a | r |  |  |
| 9 | ( | 78 | a |  |  |  |
| 10 | ) | 79 |  |  | s |  |
| 11 | ) | 6 |  |  |  | e |
| 12 | ) | 7 |  |  |  |  |
| 13 | ) | 92 |  | r |  |  |
| 14 | ) | 80 | a |  |  |  |
| 15 | start | 81 | a |  |  |  |
| 16 | put | 82 |  |  | s |  |
| 17 | put | 8 |  |  |  | e |
| 18 | put | 93 | a |  |  |  |
| 19 | 0 | 94 |  |  | s |  |
| 20 | 0 | 22 |  |  |  |  |
| 21 | 0 | 141 |  |  | s |  |
| 22 | 0 | 48 |  |  |  | e |
| 23 | 0 | 49 |  |  |  |  |
| 24 | 0 | 216 |  | r |  |  |
| 25 | 0 | 142 |  |  | s |  |
| 26 | 0 | 50 |  |  |  | e |
| 27 | 0 | 51 |  |  |  | e |
| 28 | 0 | 219 |  |  |  |  |
| 29 | 0 | 33 |  |  |  | e |
| 30 | 0 | 188 | a | r |  |  |
| 31 | to | 143 |  |  |  |  |
| 32 | to | 53 |  |  |  | e |
| 33 | to | 54 |  |  |  |  |
| 34 | to | 224 |  | r |  |  |
| 35 | to | 95 | a |  |  |  |
| 36 | a1B | 96 | a |  |  |  |
| 37 | ; | 97 | a | r |  |  |
| 38 | end | 83 |  |  | s |  |
| 39 | end | 15 |  |  |  | e |
| 40 | end | 16 |  |  |  |  |
| 41 | end | 132 |  | r |  |  |
| 42 | end | 84 | a | r |  |  |
| 43 |  | 74 |  |  |  |  |
| 44 |  | 2 |  |  |  | e |
| 45 |  | 3 |  |  |  |  |
| 46 |  | 87 |  | r |  |  |

Таблица 5. История работы автоматной реализации (конечный автомат с несколькими состояниями) нисходящего синтаксического акцептора для ошибочного тестового примера

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | a1A | 0 |  |  |  |  |
| 1 | a1A | 73 |  |  | s |  |
| 2 | a1A | 1 |  |  |  |  |
| 3 | a1A | 75 | a |  |  |  |
| 4 | = | 76 | a |  |  |  |
| 5 | void | 77 |  |  | s |  |
| 6 | void | 4 |  |  |  | e |
| 7 | void | 5 |  |  |  |  |
| 8 | void | 89 | a | r |  |  |
| 9 | ( | 78 | a |  |  |  |
| 10 | put | 79 |  |  | s |  |
| 11 | put | 6 |  |  |  | e |
| 12 | put | 7 |  |  |  |  |

Таблица 6. История работы процедурной реализации нисходящего синтаксического акцептора для правильного тестового примера

|  |  |  |
| --- | --- | --- |
| 0 | a1A | program |
| 1 | a1A | program\_\_\_0 |
| 2 | void | func\_\_\_0 |
| 3 | ) | func\_\_\_2 |
| 4 | put | func\_\_\_3 |
| 5 | 0 | expr |
| 6 | 0 | expr\_\_\_0 |
| 7 | 0 | begexpr |
| 8 | 0 | const |
| 9 | to | expr\_\_\_1 |
| 10 | end | func\_\_\_4 |
| 11 |  | program\_\_\_1 |

Таблица 7. История работы процедурной реализации нисходящего синтаксического акцептора для ошибочного тестового примера

|  |  |  |
| --- | --- | --- |
| 0 | a1A | program |
| 1 | a1A | program\_\_\_0 |
| 2 | void | func\_\_\_0 |
| 3 | put | func\_\_\_2 |

# Выводы

В данной лабораторной работе были изучены основные идеи и понятия нисходящих методов синтаксического анализа, выявления свойств формальных грамматик, необходимых для реализации нисходящего восстановления дерева грамматического разбора, приобретены навыки построения процедурной и различных автоматных реализаций нисходящего анализа, исследовать поведение нисходящих синтаксических акцепторов.