МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

О.І. Марченко, О.О. Марченко

ТЕОРІЯ ФОРМАЛЬНИХ МОВ ТА СИНТАКСИЧНОГО АНАЛІЗУ

ЛАБОРАТОРНИЙ ПРАКТИКУМ І РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник
для здобувачів ступеня бакалавра за спеціальністю
123 Комп'ютерна інженерія*

Київ

КПІ ім. Ігоря Сікорського

2023

Рецензент: *Заболотня Т.М.,* канд. техн. наук, доцент, доцент, КПІ ім. Ігоря Сікорського

Відповідальний редактор: *Щербина О.А.,* канд. техн. наук, доцент, доцент

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 6 від 30.03.2023 р.) за поданням Вченої ради Факультету прикладної математики (протокол № 6 від 30.01.2023 р.)

Електронне мережне навчальне видання

Марченко Олександр Іванович, канд. техн. наук, доцент
Марченко Олексій Олександрович, асистент

ТЕОРІЯ ФОРМАЛЬНИХ МОВ ТА СИНТАКСИЧНОГО АНАЛІЗУ

ЛАБОРАТОРНИЙ ПРАКТИКУМ І РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА

Теорія формальних мов та синтаксичного аналізу: Лабораторний практикум і розрахунково-графічна робота з дисципліни «Теорія формальних мов та синтаксичного аналізу» : [Електронний ресурс] : навч. посіб. для студ. спеціальності 123 – «Комп'ютерна інженерія» / О. І. Марченко, О. О. Марченко ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 1,1 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2023. – 122 с.

Навчальний посібник розроблено для ознайомлення студентів із завданнями дисципліни «Теорія формальних мов та синтаксичного аналізу». Навчальний посібник містить інформацію до виконання лабораторних робіт та РГР, яка включає постановку завдання для кожної роботи, необхідні теоретичні відомості, вимоги до проектів програм, вказівки до оформлення звіту та тестування розроблених алгоритмів і відповідних їм програм, варіанти індивідуальних завдань, а також контрольні питання для підготовки до захисту лабораторних робіт. Навчальний посібник призначений для студентів очної форми навчання за спеціальністю 123 – «Комп'ютерна інженерія» факультету прикладної математики КПІ ім. Ігоря Сікорського.

ЗМІСТ

ВСТУП

Вивчення дисципліни «Теорія формальних мов та синтаксичного аналізу» дозволяє сформувати у студентів компетенції, необхідні для розв'язання задач професійної діяльності, пов'язаної із застосуванням на практиці знань з теорії формальних мов та синтаксичного аналізу для проектування реальних трансляторів.

Метою дисципліни є формування у студентів здатностей:

- аналізувати граматики мов програмування з формальної точки зору з метою визначення класу, до якого належить дана граматика;
- обирати найбільш придатні для проектування транслятору даної мови програмування методи лексичного та синтаксичного аналізу;
- обирати класи граматик, які є найбільш придатними для розробки певної нової мови програмування.

Предметом навчальної дисципліни є класи формальних граматик, методи та способи лексичного та синтаксичного аналізу, спрямовані на розвиток теоретичного рівня системних програмістів та глибокого розуміння теоретичного підґрунтя розробки трансляторів..

Цикл семестрових завдань складається з трьох комплексних лабораторних робіт розробницького характеру та розрахунково-графічної роботи і призначений для покриття практичної частини дисципліни «Теорія формальних мов та синтаксичного аналізу».

Роботи дозволяють отримати практичний досвід та навички з розробки лексичних аналізаторів, синтаксичних аналізаторів та генераторів коду.

Інформація до кожної роботи включає:

- постановку завдання та вимоги до алгоритмів та програм, що повинні розробити студенти;
- вказівки до оформлення звіту та тестування розроблених програм;
- методичні вказівки та/або базовий теоретичний матеріал, необхідні для виконання робіт;
- контрольні питання для самоконтролю та підготовки до захисту роботи;
- варіанти індивідуальних завдань.

ЗАГАЛЬНІ ВИМОГИ
ДО ПРОГРАМ ЛАБОРАТОРНИХ РОБІТ

1. **Розробка програм лабораторних робіт повинна бути виконана мовою C|C++**, оскільки транслятори належать до категорії «системне програмне забезпечення», а основною мовою системного програмування де-факто є мова C/C++.

2. Професійний транслятор повинен працювати максимально швидко наскільки це можливо досягнути при розробці. Тому:
   - **заборонено використовувати будь-які бібліотечні засоби роботи з регулярними виразами**;
   - **кожен символ вхідної програми** на мові високого рівня **повинен читатися із файлу і оброблюватися у трансляторі рівно один раз і не більше**: тобто, читання із вхідного файлу лексичним аналізатором (ЛА) повинно виконуватися виняткою посимвольно, відразу повинно прийматися рішення автоматом ЛА, що з цим символом робити, і більше повернення до читання/обробки цього символа не повинно бути;
   - **заборонено використовувати бібліотеки функцій для обробки рядків** (пошук у рядку, виділення підрядків тощо).

3. **Backend лабораторних робіт** (тобто алгоритмічна частина лексичного синтаксичних аналізаторів) **повинен бути строго**

відокремлений від frontend-у (інтерфейсної частини роботи з програмою) **і знаходитись у окремих модулях/класах тощо**. Функції виведення рядка лексем, дерева розбору та інформаційних таблиць на друк є частиною frontend-у, а не backend-у. Вони повинні викликатись один раз після того як структури даних вже остаточно сформовані, а не бути переплетеними з основною логікою в алгоритмічній частині.

4. Кожна лабораторна робота повинна бути виконана у вигляді незалежної програми. Код лабораторних робіт з розробки синтаксичних аналізаторів може містити в собі код лабораторної роботи з розробки лексичного аналізатора, але не навпаки.

5. Повідомлення про помилки повинні бути якнайбільш змістовними, містити в собі всі дані про помилку, які є на момент її виникнення: стадія транслятора (Lexer, Parser), тип помилки (Error, Warning), позиція помилки (Line, Column), текст повідомлення про помилку.

   Наприклад:

Lexer: Error (Line 2, Column 1): Illegal character ‘^’ detected.

Parser: Error (Line 10, Column 15): <identifier> expected but ‘;’ found.

6. На етапі лексичного аналізу (лабораторна робота №1) для кожної лексеми (токена) у рядку лексем (токенів) повинна зберігатися позиція їх знаходження в оригінальному коді програми, що транслюється, тобто номер рядка та колонки першого символу

лексеми (токена), так, як показано на лекційній схемі, що показує загальних принцип роботи лексичного аналізатора.

Позиції лексем нумеруються **від одиниці, а не від нуля**. Синтаксичний аналізатор та генератор коду повинні використовувати ці позиції для виведення повідоьлень про помилки. Зверніть увагу на те, що колонка першого символу лексеми (токена) — це **не** те ж саме, що порядковий номер лексеми (токена) в межах одного рядка.

ЗАГАЛЬНІ ВИМОГИ ДО ТЕСТІВ ТА ТЕСТУВАННЯ

1. Кожен тест містить: 1) код мовою SIGNAL; 2) результат роботи програми, що тестується (лексичний аналізатор, синтаксичний аналізатор), для цього тесту.
2. Тести повинні бути двох категорій:
   - тести, що демонструють правильну роботу програми при правильному вхідному коді тесту (True-тести).
   - тести, що демонструють коректну роботу програми при неправильній вхідній програмі (False-тести), тобто виведення правильного повідомлення про помилку з вказанням правильного місця (рядок, колонка) цієї помилки.
3. True-тести повинні покривати різні варіанти правильного використання конструкцій граматики свого варіанту.
4. False-тести повинні покривати всі можливі типи помилок у конструкціях лексики, синтаксису та семантики граматики свого варіанту.
5. На захист лабораторних робіт та розрахунково-графічної роботи потрібно приходити з вже підготовленими тестами обох категорій, що повністю покривають лексику (ЛР №1), синтаксис (ЛР №2 та №3) свого варіанту.
6. Тести на захист повинні надаватись у різних файлах вже у готовому вигляді, а не коригуватись під час захисту в одному й тому ж файлі.
7. Кількість підготовлених тестів залежить від варіанту граматики та її семантики.

8. У звітах достатньо надрукувати по 3-5 тестів кожної категорії (True-тести, False-тести) для кожної лабораторної роботи та розрахунково-графічної роботи.
9. У звітах не повинно бути скріншотів вікна консолі з чорним фоном. Треба або копіювати текст (бажано), або інвертувати колір скріншоту (https://goo.gl/ki6Amb). В разі використання скріншоту вирізайте область з результатом так, щоб весь результат було видно і його можна було прочитати.

Приклади оформлення тестів лабораторної роботи №1

1. True-тест

Вхідна програма	**Рядок лексем**
PROGRAM SIG01; BEGIN END.	Row Col Code Lexem 1 1 1001 PROGRAM 1 9 2001 SIG01 1 14 501 ; 2 1 1002 BEGIN 3 1 1003 END 3 4 502 .

Плюс роздруківка сформованих інформаційних таблиць.

2. False-тест

Вхідна програма	**Рядок лексем**
PROGRAM SIG01; BEGIN ? END.	Row Col Code Lexem 1 1 1001 PROGRAM 1 9 2001 SIG01 1 14 501 ; 2 1 1002 BEGIN 3 1 1003 END 3 4 502 .

Lexer: Error (line 2, column 7): Illegal symbol ‘?’

Плюс роздруківка сформованих інформаційних таблиць.

Зауваження. Лексичний аналізатор, на відміну від синтаксичного, як правило працює до кінця, а не до першої помилки, і виводить повний рядок лексем, ігноруючи у рядку лексем виведення лексем недопустимих символів. Але повідомлення про недопустимі символи виводиться обов'язково. Хоча варіант реалізації роботи лексичного аналізатора до першої помилки також допускається.

Приклади оформлення тестів для ЛР №2 та №3

1. True-тест

Вхідна програма	**Дерево розбору**
PROGRAM SIG01; BEGIN END.	<signal-program> ..<program> ....1001 PROGRAM ....<procedure-identifier> ......<identifier> ....2001 SIG01 ....501 ; ....<block> ......<declarations> ....<variable-declarations> ....<empty> ......1002 BEGIN ......<statements-list> ....<empty> ......1003 END ....502 .

2. False-тест

Вхідна програма	**Дерево розбору**
PROGRAM SIG01; BEGUN END.	<signal-program> ..<program> ....1001 PROGRAM ....<procedure-identifier> ......<identifier> ....2001 SIG01 ....501 ;

Parser: Error (line 2, column 1): Keyword 'BEGIN' expected

РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА «ГРАМАТИКИ ФОРМАЛЬНИХ МОВ ТА ДЕРЕВА РОЗБОРУ»

Мета лабораторної роботи

Метою розрахунково-графічної роботи «Граматики формальних мов та дерева розбору» є засвоєння теоретичного матеріалу та набуття практичного досвіду і практичних навичок у роботі з граматиками формальних мов, виводом речень, що відповідають граматикам, а також деревами розбору.

Постановка задачі

За граматикою мови згідно варіанту виконати наступні завдання:

1. Написати послідовність дій лівостороннього виводу для програми за граматикою мови згідно варіанту.
2. Написати послідовність дій правостороннього виводу для програми за граматикою мови згідно варіанту.
3. Нарисувати дерево розбору для програми за граматикою мови згідно варіанту.
4. Форма зображення послідовності дій лівостороннього та правостороннього виводів, а також нарисоване дерево повинні мати такий вигляд, як у прикладах дерев розбору у конспекті лекцій. Див. методичні вказівки до завдання.

Зміст звіту

Звіт оформлюється згідно вимог до розрахунково-графічних робіт і має містити наступне:

- титульний аркуш;
- номер варіанту;
- індивідуальне завдання згідно до варіанту;
- послідовність дій лівостороннього виводу для програми за граматикою мови згідно варіанту;
- послідовність дій правостороннього виводу для програми за граматикою мови згідно варіанту;
- дерево розбору для програми за граматикою мови згідно варіанту.

Методичні вказівки

Розглянемо виконання завдання для граматики

1. <рядок> → <двійковий рядок>
2. <двійковий рядок> → <цифра>
3. <двійковий рядок> → <двійковий рядок><цифра>
4. <цифра> → 0
5. <цифра> → 1

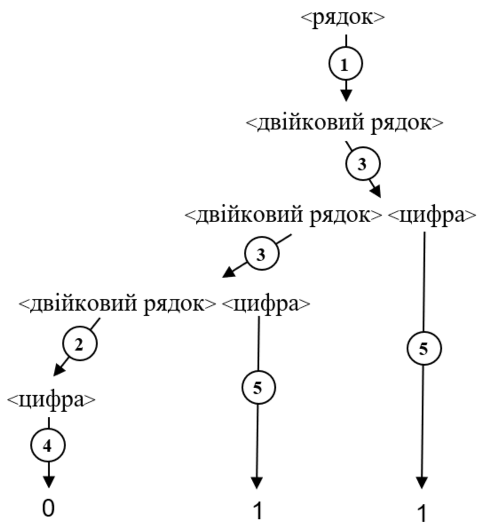
та фрагменту програми за цією граматикою

011

Форма зображення послідовності дій лівостороннього виводу для цієї програми за заданою граматикою має такий вигляд:
<рядок> $\Rightarrow^{1}$ <двійковий рядок> $\Rightarrow^{3}$ <двійковий рядок><цифра> $\Rightarrow^{3}$ <двійковий рядок ><цифра><цифра> $\Rightarrow^{2}$ <цифра><цифра><цифра> $\Rightarrow^{4}$ 0<цифра><цифра>$\Rightarrow^{5}$ 01<цифра> $\Rightarrow^{5}$ 011

Форма зображення послідовності дій правостороннього виводу для цієї програми за заданою граматикою має такий вигляд:
<рядок> $\Rightarrow^{1}$ <двійковий рядок> $\Rightarrow^{3}$ <двійковий рядок><цифра> $\Rightarrow^{5}$ <двійковий рядок> 1 $\Rightarrow^{3}$ <двійковий рядок ><цифра> 1 $\Rightarrow^{5}$ <двійковий рядок > 11 $\Rightarrow^{2}$ <цифра> 11 $\Rightarrow^{4}$ 011

Дерево розбору для цієї програми за заданою граматикою має такий вигляд:



Контрольні питання

1. Визначення транслятора, компілятора, інтерпретатора, асемблера.
2. Структурна схема транслятора.
3. Визачення формальної граматики та її алфавіту. Приклад.
4. Визначення сентенції граматики, сентерціальної форми граматики та мови граматики.
5. Запис граматик метамовою БНФ.
6. Класифікація мов за Хомським.
7. Граматики типу 0 за Хомським.
8. Граматики типу 1 за Хомським.
9. Граматики типу 2 за Хомським.
10. Граматики типу 3 за Хомським.
11. Поняття еквівалентних граматик, однозначних граматик та неоднозначних граматик.
12. Нормальна форма Грейбах і нормально форма Хомського.
13. Лівосторонній вивід.
14. Правосторонній видід.
15. Дерево розбору.

Варіанти індивідуальних завдань

Варіант визначається відповідно порядковому номеру студента в списку журнала групи.

Варіанти граматик складені на основі граматики мови SIGNAL [7] і наведені в додатку 1. В додатку 2 наведена повна граматика мови SIGNAL для загального уявлення про мову та взаємозв'язки її конструкцій.

Варіанти програм для написання лівостороннього та правостороннього виводів, а також для побудови дерева розбору за заданими граматиками наведені нижче.

Варіант 1

```
PROCEDURE  IDN ();
LABEL 12, 53, 7, 89, 34, 567;
BEGIN
END;
```

Варіант 2

```
PROCEDURE  NAME (A, B : SIGNAL FLOAT EXT; C : INTEGER;);
BEGIN
END;
```

Варіант 3

```
PROGRAM  ABC;
CONST
    C1 = 41;
    C2 = –8;
BEGIN
    VAR1 := –765;
END.
```

Варіант 4

```
PROCEDURE  ABC ( V1 : INTEGER; V2 : FLOAT; );
CONST
    TEN = 10; CN = –12;
BEGIN
END;
```

Варіант 5

```
PROGRAM  MAIN;
CONST
    ONE = +1;
    CN = 1#2;
    PI = 314#–2;
BEGIN
END.
```

Варіант 6

```
PROGRAM  PROG;
CONST
      C1 = '';
      C2 = '41,67';
      C3 = '32$EXP(3)';
BEGIN
END.
```

Варіант 7

```
PROGRAM  P7;
VAR
      V1, V2 : BLOCKFLOAT [0..5, 0..4] ;
      V3 : SIGNAL INTEGER EXT;
BEGIN
END.
```

Варіант 8

```
PROCEDURE  PR8 ( P1, P2 : INTEGER; P3 : [0..9] FLOAT; );
VAR
      V1, V2 : INTEGER [0..5, 0..4] EXT;
BEGIN
END;
```

Варіант 9

```
PROGRAM  VAR9;
DEFFUNC
      F1 = 0 \ 0, 360;
      F2 = 789 \ 1, 10;
      F3 = 10 \ 32, 64;
BEGIN
END.
```

Варіант 10

```
PROGRAM  MAIN;
PROCEDURE P1 ();
PROCEDURE P2 (V1 : COMPLEX FLOAT; V2, V3 : INTEGER EXT;);
PROCEDURE P3;
BEGIN
END.
```

Варіант 11

```
PROGRAM IDN;
LABEL 55;
BEGIN
55:  LINK A, 3;   LINK B, 2;
     IN 3;  OUT 2;
     GOTO 55;
END.
```

Варіант 12

```
PROCEDURE  PROC (X, Y, Z);
LABEL 10, 20, 30;
BEGIN
10:
      ($  INSERT1 $)
20:
      ($  INSERT2 $)
      GOTO 10;
30:
      RETURN;
END;
```

Варіант 13

```
PROCEDURE MAIN;
      PROCEDURE PR1 (P1, P2, P3);
      PROCEDURE PR2 (V1, V2);
      PROCEDURE PR3;
BEGIN
      PR1 (3, 5, 2);
      PR2 (120, 64);
      PR3;
      RETURN;
END;
```

Варіант 14

```
PROGRAM PROG;
LABEL 1, 2;
BEGIN
1:   IF A=0 THEN
          IF B=25 THEN GOTO 2;
          ELSE
          ENDIF;
     ELSE GOTO 1;
     ENDIF;
2: ;
END.
```

Варіант 15

```
PROGRAM MAIN;
BEGIN
10:  LINK X, 5;
     IN 5;
     V1 := 40;
     PR1 (R, S, T);
     ($  INSERT1 $)
     GOTO 10;
20:  RETURN;
END.
```

Варіант 16

```
PROGRAM PR16;
BEGIN
    IF A=0 THEN
        IF B>=25 THEN
        ELSE
            WHILE D <= F DO
                IF X <> 0 THEN
                ENDIF;
        ENDIF;
    ENDIF;
END.
```

Варіант 17

```
PROGRAM P17;
BEGIN
    LOOP
        FOR I := –10+J TO M+N–1 DO
            LOOP
            ENDLOOP;
        ENDFOR;
    ENDLOOP;
END.
```

Варіант 18

```
PROGRAM PRG18;
BEGIN
    LOOP
        CASE AB * CD  MOD  10  OF
        X*Y : /  LOOP
                 ENDLOOP;
              \
        A/12: /   \
        ENDCASE;
    ENDLOOP;
END.
```

Варіант 19

```
PROGRAM PROG;
VAR     V1 : INTEGER;
        V2 : FLOAT;
BEGIN
    IF V1=33 THEN
    ELSE
        IF 5=V2 THEN
        ELSE
        ENDIF;
    ENDIF;
END.
```

Варіант 20

```
PROGRAM PROG;
VAR     FL : FLOAT;
        I : INTEGER;
BEGIN
    WHILE FL <= 100 DO
        WHILE I <> 0 DO
        ENDWHILE;
    ENDWHILE;
    WHILE I < FL DO
    ENDWHILE;
END.
```

Варіант 21

```
PROGRAM P21;
VAR     A : INTEGER [0..9];
        B : FLOAT;   I : INTEGER;
BEGIN
    I := 1;
    LOOP
        LOOP
            B := A[I];
        ENDLOOP;
    ENDLOOP;
END.
```

Варіант 22

```
PROGRAM PR22;
VAR     M : SIGNAL INTEGER EXT;
        J : FLOAT EXT;   I : INTEGER;
BEGIN
    FOR I := M MOD 2  TO  M  DO
        FOR J := M/5  TO 100  DO
        ENDFOR;
    ENDFOR;
    FOR I := 0 TO M  DO
    ENDFOR;
END.
```

Варіант 23

```
PROGRAM MAIN;
CONST M = 10;  TWO = 2;
BEGIN
        CASE M+10–TWO  OF
        –12 :     /  CASE  M  OF
                     ENDCASE;
                  \
        M–5:      /  \
        100:      /  \
        ENDCASE;
END.
```

Варіант 24

```
PROGRAM P24;
VAR     A : INTEGER;
        B : INTEGER;
        C : INTEGER;
        D : INTEGER;
BEGIN
    A := [D>=7 AND D<100] AND B<>C;
    B := A<0  OR  A>10 AND A <= 45;
    C := B=A  OR  D>B  OR  A<=200;
END.
```

Варіант 25

```
PROGRAM PR25;
VAR     A : INTEGER;
        B : INTEGER;
        C : INTEGER;
        D : INTEGER;
BEGIN
    D := 100;
    C := (3*D + 50) / 2;
    A := (C * C + D) / (C – 7);
    B := A / 2 – 5*(C + D);
END.
```

Варіант 26

```
PROGRAM  P26;
CONST C1 = –20;  PI = 314#–2;
VAR
      V1 : BLOCKFLOAT [0..5, 0..4] ;
      V2, V3 : SIGNAL INTEGER EXT;
BEGIN
      LINK V1, 0;
      LINK V3, 2;
      WHILE  V2 < PI  DO
            IN 0;
            OUT 2;
      ENDWHILE;
END.
```

Варіант 27

```
PROGRAM PR27;
BEGIN
      [(A+B)*C>=5 OR D<100] AND 2*B<=C+1;
      X>=8 AND X<64 OR X>128;
END.
```

Варіант 28

```
PROCEDURE  MAIN ();
VAR
      V1 : INTEGER;  V2 : FLOAT;
PROCEDURE SUB (P1 : INTEGER; P2 : FLOAT;);
BEGIN
      LINK V1, 0;   LINK V2, 2;
      LOOP
            IN 0;
            SUB (V1, V2);
            OUT 2;
      ENDLOOP;
      RETURN;
END;
```

Варіант 29

```
PROGRAM  P29;
CONST C1= –1#–6; C2=’ 10$EXP(5) ’;
VAR
      A : SIGNAL COMPLEX BLOCKFLOAT [0..3, 0..64] ;
BEGIN
      LINK A, 4;
      IN 4;
      OUT 4;
END.
```

Варіант 30

```
PROGRAM VAR30;
CONST  M = 10;  TWO = 2;
VAR      I : SIGNAL INTEGER EXT;
BEGIN
      FOR  I := 0  TO  TWO*M  DO
            CASE  5*I+3  OF
                  (M–I)/2: /  LOOP
                              ENDLOOP;
                           \
                  –I&5:    /  \
                  64:      /  \
            ENDCASE;
      ENDFOR;
END.
```

ЛАБОРАТОРНА РОБОТА №1
«РОЗРОБКА ЛЕКСИЧНОГО АНАЛІЗАТОРА»

Мета лабораторної роботи

Метою лабораторної роботи «Розробка лексичного аналізатора» є засвоєння теоретичного матеріалу та набуття практичного досвіду і практичних навичок розробки лексичних аналізаторів (сканерів).

Постановка задачі

1. Розробити програму лексичного аналізатора (ЛА) для підмножини мови програмування SIGNAL.
2. Лексичний аналізатор має забезпечувати наступні дії:

- видалення (пропускання) пробільних символів: пробіл (код ASCII 32), повернення каретки (код ASCII 13); перехід на новий рядок (код ASCII 10), горизонтальна та вертикальна табуляція (коди ASCII 9 та 11), перехід на нову сторінку (код ASCII 12);
- згортання ключових слів;
- згортання багато-символьних роздільників (якщо передбачаються граматикою варіанту);
- згортання констант із занесенням до таблиці значення та типу константи (якщо передбачаються граматикою варіанту);
- згортання ідентифікаторів;
- видалення коментарів, заданих у вигляді **(***<текст коментаря>***)**;

- формування рядка лексем зінформацією про позиції лексем;
- заповнення таблиць ідентифікаторів та констант інформацією, отриманою підчас згортки лексем;
- виведення повідомлень про помилки.

Вимоги до програми лексичного аналізатора

1. Входом ЛА має бути наступне:

   • вхідна програма, написана підмножиною мови SIGNAL відповідно до варіанту;

   • масив Attributes (див. лекцію з поясненням принципу роботи ЛА), який містить символи таблиці кодів ASCII з відповідними їм атрибутами для визначення лексем (токенів);

   • таблиця ключових слів;

   • таблиця багато-символьних роздільників (якщо потрібно);

   • таблиця констант, в яку, при необхідності, попередньо можуть бути занесені стандартні константи;

   • таблиця ідентифікаторів, в яку, при необхідності, попередньо занесені наперед визначені ідентифікатори.

2. Виходом ЛА має бути наступне:

   • закодований рядок лексем з інформацією про їх розташування у вихідній програмі (номер рядка, номер колонки);

   • таблиця констант, що сформована для конкретної програми і яка містить значення та тип констант;

   • таблиця ідентифікаторів, що сформована для конкретної програми;

• інформаційні повідомлення про виявлені помилки.

3. Для кодування лексем при їх згортанні необхідно використовувати числові діапазони, вказані в Таблиці 1.

Таблиця 1. Діапазони кодування лексем

Вид лексеми	**Числовий діапазон**
Односимвольні роздільники та знаки операцій (: / ; + тощо)	0 – 255 (тобто коди ASCII)
Багатосимвольні роздільники (:= <= <= тощо)	301 – 400
Ключові слова (BEGIN, END, FOR тощо)	401 – 500
Константи	501 – 1000
Ідентифікатори	1001 – . . .

4. ***Рекурсивні*** **алгоритми реалізації лексичного аналізатора** ***не допускаються*** по причині їх катастрофічної глибини рекурсії, яка має лінійну залежність від кількості лексем програми. Для великих програм це неприпустимо.

5. **Читання із вихідного файлу, як зазначалося у загальних вимогах, відбувається винятково посимвольно**.

6. Позиції повинні коректно визначатися для всіх лексем (токенів); перевіряти можна в текстовому редакторі, наприклад, Notepad. Позиція кожного токена, збережена в рядку токенів, повинна співпадати з фактичною позицією цього токена в текстовому редакторі.

7. При визначенні позиції лексем символ табуляції повинен враховуватися як відповідна кількість пробілів у текстовому редакторі.

Для універсального підлаштування ЛА до конкретних текстових редакторів в коді ЛА можна, наприклад, ввести опцію відповідності кількості пробілів символу табуляції.

8. Позиції лексем (токенів) після багаторядкового коментаря також повинні бути коректними. Наприклад, у наступних двох рядках колонка токена “token” є різною і відрізняється на одну колонку.

```
The last line of comment *)token
The last line of comment *) token
```

9. Різні варіанти коментарів повинні оброблюватись коректно, наприклад

- порожні коментарі (**) (* *)
- (* багаторядковий
  коментар *)
- (*****) (*(()*())*) (*;.:*)
- незакриті коментарі
- коментарі із забороненими символами: всередині коментаря заборонені символи ***допускаються***

10. Коректна обробка заборонених символів: повинна друкуватись помилка щодо знаходження забороненого символа.

11. Коректна обробка суцільних послідовностей роздільників (хоча це й не відповідає більшості граматик), таких як

...;;:(()()());;:=:=:::=;;:.

Наприклад, суцільна послідовність роздільників

:::==

повинна бути розібрана як дві окремих двокрапки, потім одне присвоєння і один знак рівності.

Вимоги до тестів

1. Див. розділ «Загальні вимоги до тестів та тестування».
2. Тести до лексичного аналізатора повинні покривати всі ситуації пунктів 6-11 розділу «Вимоги до програми лексичного аналізатора».

Зміст звіту

Звіт оформлюється згідно вимог до лабораторних робіт і має містити наступне:

- титульний аркуш;
- індивідуальне завдання згідно до варіанту:
  - постановка задачі;
  - номер варіанту;
  - граматика за варіантом в гарно читабельному вигляді
- граф автомату, що визначає алгоритм ЛА:
  - граф в точності повинен відповідати граматиці за варіантом, тобто повинен містити стани для розбору всіх можливих за вашим варіантом токенів і **не** містити станів, які не потрібні в даному варіанті.
  - граф повинен містити гілку станів розбору коментаря. Необхідно добре розуміти як ця гілка станів працює,

оскільки одним із завдань може бути рисування гілки станів розбору коментаря і пояснення як вона працює;

- граф ЛА конкретного варіанту – це не скріншот графа із конспекту лекцій чи методички, оскільки для конкретного варіанту граф ЛА мйже завжди буде іншим;
- Рекомендація: для рисування графа можна використати, наприклад, http://graphviz.org/ (використовуйте dot) або https://www.draw.io/ ;

- лістинг коду програми ЛА:
  - звіт повинен містити **весь** код програми;
  - для коду використовувати **моноширинний** шрифт;
  - якщо код не широкий, використовуйте форматування у дві колонки (https://goo.gl/wy9ub6);

- контрольні приклади, необхідні для демонстрації всіх конструкцій заданої граматики, а також всіх можливих помилкових ситуацій (опис кожного контрольного прикладу має містити згенерований рядок лексем та заповнені таблиці).

До звіту в електронному вигляді мають бути додані:

- файл з текстом звіту;
- проект працездатної програми на мові програмування;
- завантажувальний модуль програми з необхідними файлами даних.

Методичні вказівки

Для реалізації лексичного аналізатора спочатку виконується категоризація символів таблиці ASCII, тобто кожен символ цієї таблиці призначається до певної категорії. Категорії символів будуть використовуватись в автоматі лексичного аналізатора в якості вхідних символів для переходу з одного стану в наступний. У випадках обробки багатосимвольних роздільників в якості вхідних символів автомата можуть бути використані також власне самі символи таблиці ASCII замість їх категорій.

Категоризація символів таблиці ASCII в лексичному аналізаторі, як правило, виконується у вигляді одномірного масива (вектора) чисел SymbolCategories, де коди символів таблиці ASCII використовуються в якості індексів елементів цього масива, а значеннями елементів масива є категорії відповідних символів, що призначаються цим символам.

Приклад категоризації символів.

SymbolCategories – масив категорій символів таблиці ASCII

NUL		BEL	BS		CR		Space		(	)		0		9
0		7	8		13		32		40	41		48		57
6	…	6	0	…	0	…	0	…	5	3	…	1	…	1

:	;	<	=	>		A		Z		a		z		DEL
58	59	60	61	62		65		90		97		122		127
41	3	42	3	43	…	2	…	2	…	2	…	2	…	6

A - символ таблиці ASCII (в масиві SymbolCategories не використовується)

65 - код символа таблиці ASCII (індекс масива SymbolCategories)

$\boxed{2}$ - категорія символа таблиці ASCII, призначена символу для реалізації лексичного аналізатора

В даному прикладі використані наступні категорії символів ASCII:

0 – категорія пробільних символів (whitespace): пробіл (space) – код 32 та прирівняні до пробіла керуючі символи (як правило, символи з кодами від 8 до 13);

1 – категорія символів, з яких можуть починатись **числа**;

2 – категорія символів, з яких можуть починатись **ідентифікатори та ключові слова**;

3 – категорія односимвольних роздільників;

4x (41, 42, …) – група категорій символів, з яких починаються **багатосимвольні роздільники** (на кожен такий роздільник – окрема категорія);

5x (51, 52, …) – група категорій символів, з яких починаються **багатосимвольні роздільники коментарів** (в даному прикладі така категорія одна);

6 – категорія недопустимих символів.

Розглянемо приклад побудови лексичного аналізатора (ЛА) для граматики умовного оператора, показаної на рис.1.

```
1.  <statement> → if <expr> then <expr> |
                  if <expr> then <expr>  else <expr> |
                  <empty>
2.  <expr> → <term> <operation> <term>
3.  <operation> → = | < | <=
4.  <term> → <number> | <idn>
5.  <number> → <digit> | <digits>
6.  <digits> → <digit><digits>
7.  <idn> → <let><lets_or_digits>
8.  <lets_or_digits> → <let><lets_or_digits> |
                  <digit><lets_or_digits> |
                  <empty>
9.  <let> → a | b | ...  | z
10. <digit> →  0 | 1 | ... | 9
```

Рис.1 Граматика умовного оператора

В цій граматиці правила 1–4 складають синтаксичну частину граматики і будуть реалізовані у синтаксичному аналізаторі. Правила 5–10 складають лексичну частину граматики, тобто описують **лексеми (токени)** граматики і повинні бути реалізовані у лексичному аналізаторі. Лексична частина граматики, що визначається правилами 5–10, може бути для зручності записана також у вигляді **регулярних визначень:**

number → digit*

identifier → let (let | digit)*

let → [a-z]

digit → [0-9]

Визначимо лексеми (токени), які будуть виділятись лексичним аналізатором. Згідно правил 5–10, для даної граматики такими лексемами будуть «число» (**number**) і «ідентифікатор» (**identifier**) Крім того, до лексем граматики належать також всі її ключові слова (**if**, **then**, **else**) та роздільники (=, <, <=), які присутні безпосередньо в граматиці.

Для розпізнавання цих лексем, виконаємо категоризацію символів ASCII згідно їх використанню в даній граматиці. Основні елементи масива SymbolCategories категорій символів ASCII для даної граматики будуть такими:

NUL		BEL	BS		CR		Space		(	)		0		9
0		7	8		13		32		40	41		48		57
6	…	6	0	…	0	…	0	…	5	6	…	1	…	1

:	;	<	=	>		A		Z		A		z		DEL
58	59	60	61	62		65		90		97		122		127
6	6	4	3	6	…	2	…	2	…	2	…	2	…	6

Призначення кодів категоріям символів виконано згідно вищенаведеного прикладу. Символи, з яких може починатися лексема «число» (**number**), належать до категорії 1 (позначимо її символами **dig**). Символи, з яких можуть починатися лексеми «ідентифікатор» та «ключове слово» (**identifier**), належать до категорії 2 (позначимо її символами **let**).

В даній маленькій граматиці є тільки одна «чиста» лексема типу «односимвольний роздільник» (**delimiter1**) – символ **'='**, тому цей символ належить до категорії 3 (позначимо її символами **dm1**), і тільки одна лексема типу «багатосимвольний роздільник» **'<='** (**delimiter2**), тому символ **'<'**, з якого ця лексема починаєтися, належить до категорії 4 (позначимо її символами **dm2**). Символ **'<'** в якості односимвольного роздільника буде розпізнаватись в рамках обробки категорії 4.

Крім того, коментарі в даному прикладі будуть обмежуватись символами **(*** коментар ***)**. Тому, символу **'('** призначається категорія 5 (позначимо її символами **com**). Інших роздільників в даній граматиці нема, тому вони є недопустимими символами. Недопустимі символи належать до категорії 6 (позначимо її символами **err**). Нагадаємо також, що лексичний аналізатор, крім лексем, що визначаються граматикою мови, повинен ще виділяти і оброблювати пробільні символи (категорія 0, позначимо її символами **ws**).

Закінчення роботи автомата виконується при надходженні символа кіця файла **eof**.

Побудуємо **укрупнений граф лексичного автомата** для даного прикладу (рис.2).

Стани в графі позначаються еліпсами. Початковий стан позначається еліпсом з товстою лінією, а кінцевий «подвійним» еліпсом. Стани з'єднуються між собою ребрами з мітками, що означають умову переходу (вхідний символ автомата). Ребра, що не позначені мітками, означають безумовний перехід до стану.

В укрупненому графі автомата ЛА стартовий стан позначений **S**, кінцевий стан – **EXIT**, стан введення наступного символа – **INP** . Інші стани відповідають розпізнаванню визначених вище лексем (токенів) з іменами, що відповідають іменам лексем (**number, identifier, delimiter1, delimiter2**). Крім того, є також додаткові стани (**comment, whitespace**), які відповідають обробці інших категорій символів (**com, ws**), з яких вихідні лексеми не формуються, але які також потрібно оброблювати. Стан **ERR** введений для обробки знайденої помилки при надходженні символа категорії недопустимих символів **err**. Вважається, що на виході кожного із цих станів, окрім **ERR**, завжди буде вже новий, ще необроблений, символ.

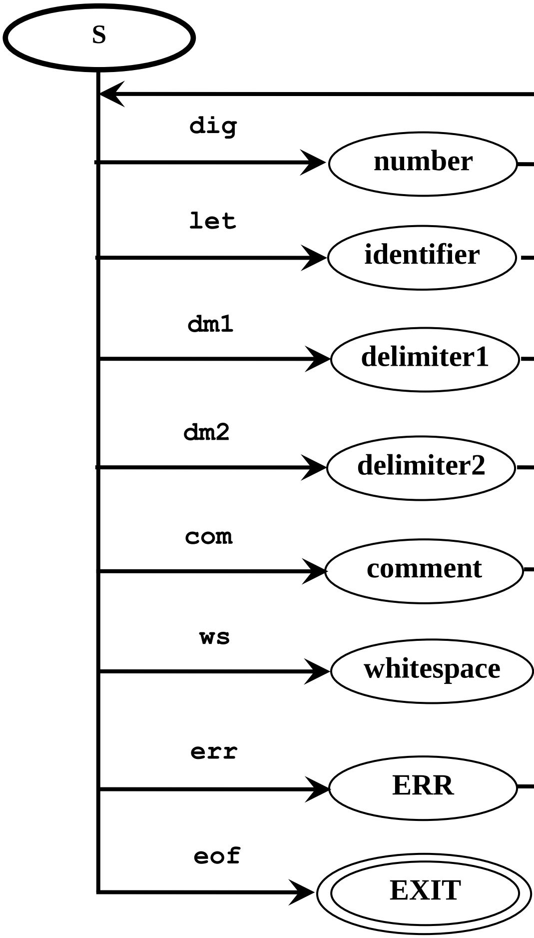


Рис.2 Укрупнений граф автомата ЛА

Значення міток на укрупненому графі автомата лексичного аналізатора (категорій вхідних символів):

- **let** – категорія літер (**a .. z**);
- **dig** – категорія цифр (**0 .. 9**);

- **dm1** – категорія односимвольних роздільників (=);
- **dm2** – категорія багатосимвольних роздільників (<=);
- **com** – категорія початкових символів коментаря (**(**)
- **ws** – категорія пробільних символів;
- **err** – категорія недопустимих символів;
- **eof** – символ кінця файлу.

Значення станів на укрупненому графі автомата лексичного аналізатора:

- **S** – стартовий стан (стан графа автомата в момент початку роботи ЛА);
- **EXIT** – кінцевий стан ЛА;
- **INP** – введення чергового символа;
- **number** – виділення і обробка лексеми (токена) **number**;
- **identifier** – виділення і обробка лексеми (токена) **identifier**;
- **delimiter1** – виділення і обробка лексеми (токена) **delimiter1**;
- **delimiter2** – виділення і обробка лексеми (токена) **delimiter2**;
- **comment** – виділення і обробка коментарів;
- **whitespace** – виділення і обробка пробільних символів;
- **ERR** – видача повідомлення про помилку.

Подальшу розробку ЛА будемо вести методом покрокової деталізації. Деталізуємо роботу ЛА для станів **identifier**, **number**, **delimiter1**, **delimiter2**, **whitespace** та **comment**. Для цього побудуємо часткові підграфи роботи автомата ЛА в цих станах. Необхідно зауважити, що для невеликих граматик можна одразу побудувати всю діаграму переходів ЛА повністю, без покрокової деталізації.

Значення станів на часткових підграфах виділення та обробки вищезазначених лексем:

- INP – введення чергового символа;
- OUT – обробка та виведення лексеми (токена);
- NUM – обробка чергового символа лексеми (токена) **number** і введення наступного символа;
- IDN – обробка чергового символа лексеми (токена) **identifier** і введення наступного символа;
- DM2 – введення наступного символа після символа ‘<’, який є початковим символом можливого двосимвольного роздільника ‘<=’, і прийняття рішення про наступний перехід;
- WS – пропуск чергового пробільного символа і введення наступного символа;
- BCOM – введення наступного символа після першого символа початку коментаря (в даному випадку символа ‘(’) і прийняття рішення про наступний перехід;
- COM – введення наступного символа коментаря і прийняття рішення про наступний перехід;
- ECOM – введення наступного символа після символа ‘*’ і прийняття рішення про закінчення чи продовження коментаря;
- ERR – видача повідомлення про помилку.

У всіх станах, окрім OUT та ERR, виконується введення чергового символу. У стані OUT, окрім виведення сформованої лексеми (код, рядок, колонка) у файл (послідовність лексем), виконуються також всі необхідні дії з відповідними таблицями ідентифікаторів, констант, тощо, а вже введений, але ще не проаналізований, наступний символ передається далі на обробку.

На ребрах часткових підграфів автомата ЛА використовується також мітка '**other**', яка означає, що перехід виконується при надходженні будь-якого іншого символа, крім тих, категорії яких є на інших ребрах, що виходять з цього ж стану.

На рис.3 показаний підграф виділення лексеми (токена) **number**.

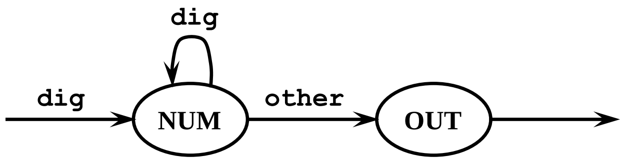


Рис.3 Підграф виділення лексеми (токена) **number**

На рис.4 показаний підграф виділення лексеми (токену) **identifier.**

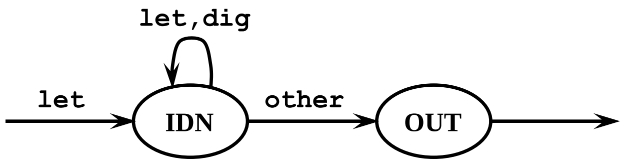


Рис.4 Підграф виділення лексеми (токену) **identifier**

Одним із способів відділення ключових слів від ідентифікаторів є розміщення ключових слів у окремій таблиці при ініціалізації ЛА. Для нашого прикладу граматики ця таблиця має бути проініціалізована ключовими словами **if**, **then**, **else**. Але оскільки ключові слова відповідають правилам запису ідентифікаторів, то при початковому виділенні ЛА не може відрізнити їх від ідентифікаторів. Саме тому обробка лексем (токенів) **if, then, else** буде відбуватись у стані **identifier**. Після виділення лексеми, що може бути або ключовим словом або ідентифікатором, спочатку виконується пошук цієї лексеми у таблиці ключових слів. Якщо ця лексема буде знайдена у цій таблиці, значить було віділене ключлве слово, інакше – звичайний ідентифікатор.

На рис.5 показаний підграф виділення лексеми (токена) **delimiter1**:

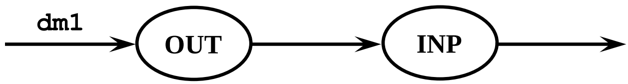


Рис.5 Підграф виділення лексеми (токена) **delimiter1**

На рис.6 показаний підграф виділення лексеми (токена) **delimiter2**:

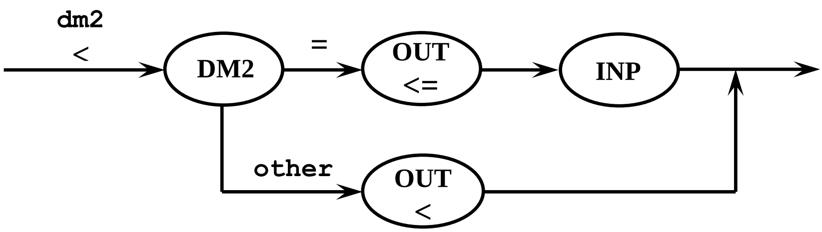


Рис.6 Підграф виділення лексеми (токена) **delimiter2**

На рис.7 показаний підграф виділення та обробки пробільних символів **whitespace**.

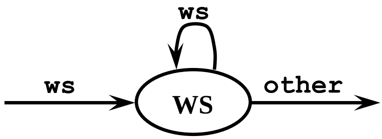


Рис.7 Підграф виділення та обробки **whitespace**

Однією із задач ЛА є видалення коментарів. Хоча коментар не можна назвати лексемою (токеном), його обробку все ж необхідно описати у вигляді окремого підграфа. На рис. 8 показаний підграф обробки коментарів, заданих у вигляді **(***<текст коментаря>***)**.

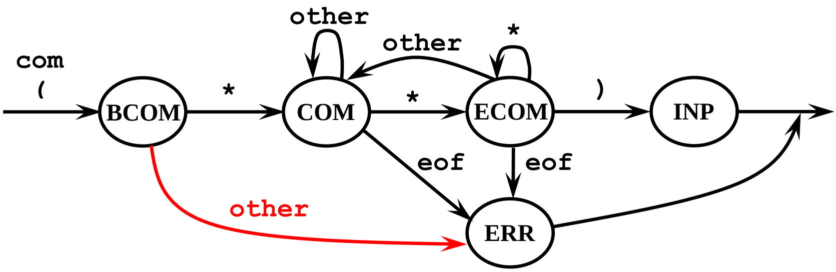


Рис.8 Підграф виділення та обробки коментаря

Важливе зауваження: ***в даному підграфі символ "other" у стані BCOM призводить до помилки (червоне ребро). Однак <u>це вірно лише для граматики даного прикладу</u>, <u>у якій символ "(" використовується тільки як символ початку коментаря і не є односимвольним роздільником</u>. Як виконується обробка цього ребра підграфа у загальному випадку описано в матеріалах лекцій.***

Повний граф автомата ЛА для граматики умовного оператора (рис.1), показаний на рис.9.

Після побудови повного графа автомата можна приступити до реалізації програми ЛА будь-якою мовою програмування так, як це описано в матеріалах лекцій на псевдокоді. Варто зауважити, що при такому підході розмір програми буде пропорційним кількості виділених станів. Кожен стан дає частину коду, що описує поведінку ЛА.

Контрольні питання

1. Визначення транслятора, компілятора, інтерпретатора, асемблера.
2. Визачення формальної граматики та її алфавіту. Приклад.
3. Визначення сентенції граматики, сентерціальної форми граматики та мови граматики.
4. Класифікація мов за Хомським.
5. Поняття еквівалентних граматик, однозначних граматик та неоднозначних граматик.
6. Нормальна форма Грейбах і нормально форма Хомського.
7. Структурна схема та основні функції лексичного аналізатора (сканера)..
8. Поняття лексеми (токена). Види лексем.
9. Типові стани графу автомата лексичного аналізатора (сканера).
10. Спосіб написання програми за графом автомату лексичного аналізатора (сканера).

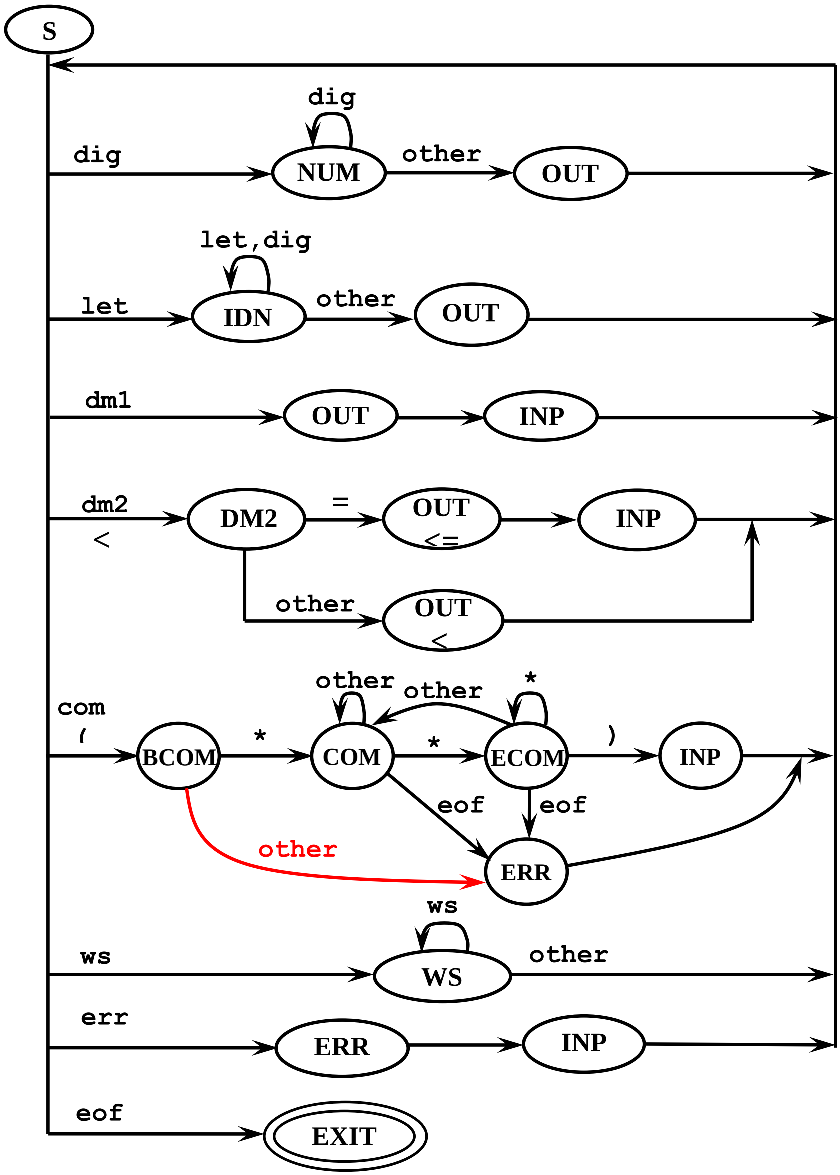


Рис.9. Граф автомата ЛА для граматики умовного оператора

Варіанти індивідуальних завдань

Варіанти граматик складені на основі граматики мови SIGNAL [7] і наведені в додатку 1. В додатку 2 наведена повна граматика мови SIGNAL для загального уявлення про мову та взаємозв'язки її конструкцій.

Варіант визначається відповідно порядковому номеру студента в списку журнала групи.

ЛАБОРАТОРНА РОБОТА №2
«РОЗРОБКА СИНТАКСИЧНОГО АНАЛІЗАТОРА МЕТОДОМ РЕКУРСИВНОГО СПУСКУ»

Мета лабораторної роботи

Метою лабораторної роботи №2 «Розробка синтаксисного аналізатора методом рекурсивного спуску» є засвоєння теоретичного матеріалу та набуття практичного досвіду і практичних навичок розробки синтаксичних аналізаторів (парсерів) методом рекурсивного спуску.

Постановка задачі

1. Розробити програму синтаксичного аналізатора (СА) для підмножини мови програмування SIGNAL згідно граматики за варіантом методом рекурсивного спуску.
2. Програма має забезпечувати наступне:

- читання рядка лексем та таблиць, згенерованих лексичним аналізатором, який було розроблено в лабораторній роботі «Розробка лексичного аналізатора»;
- синтаксичний аналіз (розбір) програми, поданої рядком лексем;
- побудову дерева розбору;
- формування таблиць ідентифікаторів та різних констант з повною інформацією, необхідною для генерування коду;

- формування лістингу вхідної програми з повідомленнями про лексичні та синтаксичні помилки.

Вимоги до програми синтаксичного аналізатора

1. **Заборонено використовувати будь-які бібліотечні засоби роботи з регулярними виразами**;
2. Входом синтаксичного аналізатора має бути наступне:

- закодований рядок лексем;
- таблиці ідентифікаторів, числових, символьних та рядкових констант (якщо це передбачено граматикою варіанту), згенеровані лексичним аналізатором;
- вхідна програма на підмножині мови програмування SIGNAL згідно з варіантом (необхідна для формування лістнигу програми).

3. Виходом синтаксичного аналізатора має бути наступне:

- дерево розбору вхідної програми;
- таблиці ідентифікаторів та різних констант з повною інформацією, необхідною для генерування коду;
- лістинг вхідної програми з повідомленнями про лексичні та синтаксичні помилки.

4. Дерево розбору повинно містити в собі

- всі без винятку нетермінали, по яких розібрався вхідний код;
- тобто, якщо у вас відсутній який-небуть список (тобто, за граматикою може бути <empty>), то необхідно показати в дереві весь шлях до <empty> включно.

5. Правила граматики, які містять в собі альтернативу <empty> , по факту, дозволяють використовувати порожні конструкції.

Вимоги до тестів

1. Див. розділ «Загальні вимоги до тестів та тестування».
2. Тести до синтаксичного аналізатора повинні покривати всі окремі ситуації згідно граматики варіанту.
3. Повинен бути комплексний тест, що містить в собі всі можливі конструкції граматики одночасно (максимальний тест).

Зміст звіту

Звіт оформлюється згідно вимог до лабораторних робіт і має містити наступне:

- титульний аркуш лабораторної роботи;
- індивідуальне завдання згідно до варіанту:
    - постановка задачі;
    - номер варіанту;
    - граматика за варіантом в гарно читабельному вигляді
- перетворену граматику (якщо це необхідно для реалізації заданого алгоритму синтаксичного аналізатора);
- лістинг коду програми СА:
    - звіт повинен містити **весь** код програми;
    - для коду використовувати **моноширинний** шрифт;
    - якщо код не широкий, використовуйте форматування у дві колонки (https://goo.gl/wy9ub6);

- контрольні приклади, необхідні для демонстрації всіх конструкцій заданої граматики, а також всіх можливих помилкових ситуацій;
- опис кожного контрольного прикладу має містити вхідний рядок лексем (результат роботи ЛА, розробленого в лабораторній роботі «Розробка лексичного аналізатора»), згенеровані таблиці, а також зображення побудованого дерева розбору;
- рисунки дерев розбору, побудованих для всіх контрольних прикладів (мінімум 5 прикладів і дерев).

До звіту в електронному вигляді мають бути додані:

- файл з текстом звіту;
- проект працюючої програми на мові програмування;
- завантажувальний модуль програми з необхідними файлами даних.

Методичні вказівки

Загальна схема синтаксичного аналізу на основі недетермінованого МП-автомата працює з поверненнями. Однак, якщо на граматику, яка породжує мову, накласти певні обмеження, а також ефективно використовувати стек автомата, то можна використати певні відомі прийоми, які дозволяють будувати ефективні синтаксичні аналізатори, що працюють без повернень.

На практиці, в СА існуючих мов програмування використовується саме такий підхід. Найефективніші методи синтаксичного аналізу працюють тільки з підкласами граматик. Найбільш швидкими є алгоритми синтаксичного аналізу для LL(1)-граматик, які працюють за низхідною стратегією розбору.

Зауваження: низхідні методи розбору не можуть працювати з граматиками, що мають ліворекурсивні правила. Тому при використанні таких методів необхідно попередньо усунути ліву рекурсію з граматики.

Метод рекурсивного спуску детально пояснюється в конспекті лекцій та у відео-лекції https://youtu.be/oLDmeX-X5rI

Контрольні питання

1. Принцип виконання низсхідного аналізу (розбору).
2. Принцип виконання висхідного аналізу (розбору).
3. Визначення автомату з магазинною (стековою) пам'яттю (МП-автомату).
4. Конфігурація і такт роботи МП-автомата.
5. Детерміновані та не детерміновані МП-автомати.
6. Відповідність між КВ-граматикою та МП-автоматом.
7. Визначення LL(k), LR(k), RL(k) і RR(k) граматик.
8. Основні алгоритми низсхідної стратегії синтаксичного розбору та переваги використання LL(1)-граматик.
9. Принцип роботи синтаксичного аналізу за алгоритмом рекурсивного спуску.
10. Принцип роботи синтаксичного аналізу за алгоритмом граматик передування.

11.Принцип роботи синтаксичного аналізу за алгоритмом таблично-керованого передбачаючого (прогнозуючого) розбіру.

Варіанти індивідуальних завдань

Варіанти граматик складені на основі граматики мови SIGNAL [7] і наведені в додатку 1. В додатку 2 наведена повна граматика мови SIGNAL для загального уявлення про мову та взаємозв'язки її конструкцій.

Варіант визначається відповідно порядковому номеру студента в списку журнала групи.

ЛАБОРАТОРНА РОБОТА №3
«РОЗРОБКА СИНТАКСИЧНОГО АНАЛІЗАТОРА МЕТОДОМ АНАЛІТИЧНОЇ МАШИНИ КНУТА»

Мета лабораторної роботи

Метою лабораторної роботи №3 «Розробка синтаксисного аналізатора методом аналітичної машини Кнута» є засвоєння теоретичного матеріалу та набуття практичного досвіду і практичних навичок розробки синтаксичних аналізаторів (парсерів) методом аналітичної машини Кнута.

Постановка задачі

1. Розробити програму синтаксичного аналізатора (СА) для підмножини мови програмування SIGNAL згідно граматики за варіантом методом аналітичної машини Кнута.
2. Програма має забезпечувати наступне:

- читання рядка лексем та таблиць, згенерованих лексичним аналізатором, який було розроблено в лабораторній роботі «Розробка лексичного аналізатора»;
- синтаксичний аналіз (розбір) програми, поданої рядком лексем;
- побудову дерева розбору;
- формування таблиць ідентифікаторів та різних констант з повною інформацією, необхідною для генерування коду;

- формування лістингу вхідної програми з повідомленнями про лексичні та синтаксичні помилки.

Вимоги до програми синтаксичного аналізатора

1. **Заборонено використовувати будь-які бібліотечні засоби роботи з регулярними виразами**;
2. Входом синтаксичного аналізатора має бути наступне:

- закодований рядок лексем;
- таблиця перходів аналітичної машини Кнута;
- таблиці ідентифікаторів, числових, символьних та рядкових констант (якщо це передбачено граматикою варіанту), згенеровані лексичним аналізатором;
- вхідна програма на підмножині мови програмування SIGNAL згідно з варіантом (необхідна для формування лістнигу програми).

3. Виходом синтаксичного аналізатора має бути наступне:

- дерево розбору вхідної програми;
- таблиці ідентифікаторів та різних констант з повною інформацією, необхідною для генерування коду;
- лістинг вхідної програми з повідомленнями про лексичні та синтаксичні помилки.

4. Дерево розбору повинно містити в собі:

- всі без винятку нетермінали, по яких розібрався вхідний код;
- тобто, якщо у вас відсутній який-небуть список (тобто, за граматикою може бути <empty>), то необхідно показати в дереві весь шлях до <empty> включно.

5. Правила граматики, які містять в собі альтернативу <empty> , по факту, дозволяють використовувати порожні конструкції.

Вимоги до тестів

1. Див. розділ «Загальні вимоги до тестів та тестування».
2. Тести до синтаксичного аналізатора повинні покривати всі окремі ситуації згідно граматики варіанту.
3. Повинен бути комплексний тест, що містить в собі всі можливі конструкції граматики одночасно (максимальний тест).

Зміст звіту

Звіт оформлюється згідно вимог до лабораторних робіт і має містити наступне:

- титульний аркуш лабораторної роботи;
- індивідуальне завдання згідно до варіанту:
  - постановка задачі;
  - номер варіанту;
  - граматика за варіантом в гарно читабельному вигляді
- перетворену граматику (якщо це необхідно для реалізації заданого алгоритму синтаксичного аналізатора);
- таблицю переходів машини Кнута відповідно до граматики;
- лістинг коду програми СА:
  - звіт повинен містити **весь** код програми;
  - для коду використовувати **моноширинний** шрифт;

  - якщо код не широкий, використовуйте форматування у дві колонки (https://goo.gl/wy9ub6);

- контрольні приклади, необхідні для демонстрації всіх конструкцій заданої граматики, а також всіх можливих помилкових ситуацій;
- опис кожного контрольного прикладу має містити вхідний рядок лексем (результат роботи ЛА, розробленого в лабораторній роботі «Розробка лексичного аналізатора»), згенеровані таблиці, а також зображення побудованого дерева розбору;
- рисунки дерев розбору, побудованих для всіх контрольних прикладів (мінімум 5 прикладів і дерев).

До звіту в електронному вигляді мають бути додані:

- файл з текстом звіту;
- проект працюючої програми на мові програмування;
- завантажувальний модуль програми з необхідними файлами даних.

Методичні вказівки

Загальна схема синтаксичного аналізу на основі недетермінованого МП-автомата працює з поверненнями. Однак, якщо на граматику, яка породжує мову, накласти певні обмеження, а також ефективно використовувати стек автомата, то можна використати певні відомі прийоми, які дозволяють будувати ефективні синтаксичні аналізатори, що працюють без повернень.

На практиці, в СА існуючих мов програмування використовується саме такий підхід. Найефективніші методи синтаксичного аналізу працюють тільки з підкласами граматик. Найбільш швидкими є алгоритми синтаксичного аналізу для LL(1)-граматик, які працюють за низхідною стратегією розбору.

Зауваження: низхідні методи розбору не можуть працювати з граматиками, що мають ліворекурсивні правила. Тому при використанні таких методів необхідно попередньо усунути ліву рекурсію з граматики.

Метод рекурсивного спуску детально пояснюється в конспекті лекцій та у відео-лекції https://youtu.be/oLDmeX-X5rI

Контрольні питання

1. Принцип виконання низсхідного аналізу (розбору).
2. Принцип виконання висхідного аналізу (розбору).
3. Визначення автомату з магазинною (стековою) пам’яттю (МП-автомату).
4. Конфігурація і такт роботи МП-автомата.
5. Детерміновані та не детерміновані МП-автомати.
6. Відповідність між КВ-граматикою та МП-автоматом.
7. Визначення LL(k), LR(k), RL(k) і RR(k) граматик.
8. Основні алгоритми низсхідної стратегії синтаксичного розбору та переваги використання LL(1)-граматик.
9. Принцип роботи синтаксичного аналізу за алгоритмом аналізуючої машини Кнута (АМК).

10.Формування таблиці АМК (програмування АМК).

11.Принцип роботи синтаксичного аналізу за алгоритмом граматик передування.

12.Принцип роботи синтаксичного аналізу за алгоритмом таблично-керованого передбачаючого (прогнозуючого) розбіру.

Варіанти індивідуальних завдань

Варіанти граматик складені на основі граматики мови SIGNAL [7] і наведені в додатку 1. В додатку 2 наведена повна граматика мови SIGNAL для загального уявлення про мову та взаємозв’язки її конструкцій.

Варіант визначається відповідно порядковому номеру студента в списку журнала групи.

ДОДАТОК 1. ВАРІАНТИ ГРАМАТИК ДЛЯ ІНДИВІДУАЛЬНИХ ЗАВДАНЬ.

Варіанти граматик складені на основі граматики мови SIGNAL [7]. В додатку 2 наведена повна граматика мови SIGNAL для загального уявлення про мову та взаємозв’язки її конструкцій.

Варіант визначається відповідно порядковому номеру студента в списку журнала групи.

Варіант 1

```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
                 <block>. |
                 PROCEDURE <procedure-identifier>
                 <parameters-list> ; <block> ;
3.     <block> --> <declarations> BEGIN <statements-list>
                 END
4.     <declarations> --> <label-declarations>
5.     <label-declarations> --> LABEL <unsigned-integer>
                 <labels-list>; |
                 <empty>
6.     <labels-list> --> , <unsigned-integer> <labels-list>
                 |
                 <empty>
7.     <parameters-list> --> ( <declarations-list> ) |
                 <empty>
8.     <declarations-list> --> <empty>
9.     <statements-list> --> <empty>
10.    <procedure-identifier> --> <identifier>
11.    <identifier> --> <letter><string>
12.    <string> --> <letter><string> |
                 <digit><string> |
                 <empty>
13.    <unsigned-integer> --> <digit><digits-string>
14.    <digits-string> --> <digit><digits-string> |
                 <empty>
15.    <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

```
16.   <letter> --> A | B | C | D | ... | Z
```

Варіант 2

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
               <block>. |
               PROCEDURE <procedure-
               identifier><parameters-list> ; <block> ;
3.    <block> --> BEGIN <statements-list> END
4.    <statements-list> --> <empty>
5.    <parameters-list> --> ( <declarations-list> ) |
               <empty>
6.    <declarations-list> --> <declaration> <declarations-
               list> |
               <empty>
7.    <declaration> --><variable-identifier><identifiers-
               list>:<attribute><attributes-list> ;
8.    <identifiers-list> --> , <variable-identifier>
               <identifiers-list> |
               <empty>
9.    <attributes-list> --> <attribute> <attributes-list>
               |
               <empty>
10.   <attribute> --> SIGNAL |
               COMPLEX |
               INTEGER |
               FLOAT |
               BLOCKFLOAT |
               EXT
11.   <procedure-identifier> --> <identifier>
12.   <variable-identifier> --> <identifier>
13.   <identifier> --> <letter><string>
14.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
15.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
16.   <letter> --> A | B | C | D | ... | Z
```

Варіант 3

```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
                 <block>.
3.     <block> --> <declarations> BEGIN <statements-list>
                 END
4.     <declarations> --> <constant-declarations>
5.     <constant-declarations> --> CONST <constant-
                 declarations-list> |
                 <empty>
6.     <constant-declarations-list> --> <constant-
                 declaration> <constant-declarations-list> |
                 <empty>
7.     <constant-declaration> --> <constant-identifier> =
                 <constant>;
8.     <statements-list> --> <statement> <statements-list>
                 |
                 <empty>
9.     <statement> --> <variable-identifier> := <constant>
                 ;
10.    <constant> --> <unsigned-integer>
11.    <constant> --> - <unsigned-integer>
12.    <constant-identifier> --> <identifier>
13.    <variable-identifier> --> <identifier>
14.    <procedure-identifier> --> <identifier>
15.    <identifier> --> <letter><string>
16.    <string> --> <letter><string> |
                 <digit><string> |
                 <empty>
17.    <unsigned-integer> --> <digit><digits-string>
18.    <digits-string> --> <digit><digits-string> |
                 <empty>
19.    <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
20.    <letter> --> A | B | C | D | ... | Z
```

Варіант 4

```
1.    <signal-program> --> <program>
2.    <program> --> PROCEDURE <procedure-
               identifier><parameters-list> ; <block> ;
3.    <block> --> <declarations> BEGIN <statements-list>
               END
4.    <statements-list> --> <empty>
5.    <parameters-list> --> ( <declarations-list> ) |
               <empty>
6.    <declarations-list> --> <declaration> <declarations-
               list> |
               <empty>
7.    <declaration> --> <variable-identifier> :
               <attribute> ;
8.    <attribute> --> INTEGER | FLOAT
9.    <declarations> --> <constant-declarations>
10.   <constant-declarations> --> CONST <constant-
               declarations-list> |
               <empty>
11.   <constant-declarations-list> --> <constant-
               declaration> <constant-declarations-list> |
               <empty>
12.   <constant-declaration> --> <constant-identifier> =
               <constant>;
13.   <constant> --> <unsigned-integer>
14.   <constant> --> - <unsigned-integer>
15.   <constant-identifier> --> <identifier>
16.   <variable-identifier> --> <identifier>
17.   <procedure-identifier> --> <identifier>
18.   <identifier> --> <letter><string>
19.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
20.   <unsigned-integer> --> <digit><digits-string>
21.   <digits-string> --> <digit><digits-string> |
               <empty>
22.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
23.   <letter> --> A | B | C | D | ... | Z
```

Варіант 5

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
               <block>.
3.    <block> --> <declarations> BEGIN <statements-list>
               END
4.    <statements-list> --> <empty>
5.    <declarations> --> <constant-declarations>
6.    <constant-declarations> --> CONST <constant-
               declarations-list> |
               <empty>
7.    <constant-declarations-list> --> <constant-
               declaration> <constant-declarations-list> |
               <empty>
8.    <constant-declaration> --> <constant-identifier> =
               <constant>;
9.    <constant> --> <sign> <unsigned-constant>
10.   <constant-identifier> --> <identifier>
11.   <procedure-identifier> --> <identifier>
12.   <identifier> --> <letter><string>
13.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
14.   <unsigned-constant> --> <integer-part><fractional-
               part>
15.   <integer-part> --> <unsigned-integer>
16.   <fractional-part> --> #<sign><unsigned-integer> |
               <empty>
17.   <unsigned-integer> --> <digit><digits-string>
18.   <digits-string> --> <digit><digits-string> |
               <empty>
19.   <sign> --> + |
               - |
               <empty>
20.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
21.   <letter> --> A | B | C | D | ... | Z
```

Варіант 6

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
              <block>.
3.    <block> --> <declarations> BEGIN <statements-list>
              END
4.    <statements-list> --> <empty>
5.    <declarations> --> <constant-declarations>
6.    <constant-declarations> --> CONST <constant-
              declarations-list> |
              <empty>
7.    <constant-declarations-list> --> <constant-
              declaration> <constant-declarations-list> |
              <empty>
8.    <constant-declaration> --> <constant-identifier> =
              <constant>;
9.    <constant> --> '<complex-number>'
10.   <complex-number> --> <left-part> <right-part>
11.   <left-part> --> <unsigned-integer> |
              <empty>
12.   <right-part> --> ,<unsigned-integer> |
              $EXP( <unsigned-integer> ) |
              <empty>
13.   <constant-identifier> --> <identifier>
14.   <procedure-identifier> --> <identifier>
15.   <identifier> --> <letter><string>
16.   <string> --> <letter><string> |
              <digit><string> |
              <empty>
17.   <unsigned-integer> --> <digit><digits-string>
18.   <digits-string> --> <digit><digits-string> |
              <empty>
19.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
20.   <letter> --> A | B | C | D | ... | Z
```

Варіант 7

```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
                <block>.
3.     <block> --> <declarations> BEGIN <statements-list>
                END
4.     <statements-list> --> <empty>
5.     <declarations> --> <variable-declarations>
6.     <variable-declarations> --> VAR <declarations-list>
                |
                <empty>
7.     <declarations-list> --> <declaration> <declarations-
                list> |
                <empty>
8.     <declaration> --><variable-identifier><identifiers-
                ist>:<attribute><attributes-list> ;
9.     <identifiers-list> --> , <variable-identifier>
                <identifiers-list> |
                <empty>
10.    <attributes-list> --> <attribute> <attributes-list>
                |
                <empty>
11.    <attribute> --> SIGNAL |
                COMPLEX |
                INTEGER |
                FLOAT |
                BLOCKFLOAT |
                EXT |
                [<range><ranges-list>]
12.    <ranges-list> -->  ,<range> <ranges-list> |
                <empty>
13.    <range> --> <unsigned-integer> .. <unsigned-integer>
14.    <variable-identifier> --> <identifier>
15.    <procedure-identifier> --> <identifier>
16.    <identifier> --> <letter><string>
17.    <string> --> <letter><string> |
                <digit><string> |
                <empty>
18.    <unsigned-integer> --> <digit><digits-string>
19.    <digits-string> --> <digit><digits-string> |
                <empty>
20.    <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
21.    <letter> --> A | B | C | D | ... | Z
```

Варіант 8

```
1.    <signal-program> --> <program>
2.    <program> --> PROCEDURE <procedure-
              identifier><parameters-list> ; <block> ;
3.    <block> --> <declarations> BEGIN <statements-list>
              END
4.    <statements-list> --> <empty>
5.    <parameters-list> --> ( <declarations-list> ) |
              <empty>
6.    <declarations> --> <variable-declarations>
7.    <variable-declarations> --> VAR <declarations-list>
              |
              <empty>
8.    <declarations-list> --> <declaration> <declarations-
              list> |
              <empty>
9.    <declaration> --><variable-identifier><identifiers-
              list>:<attribute><attributes-list> ;
10.   <identifiers-list> --> , <variable-identifier>
              <identifiers-list> |
              <empty>
11.   <attributes-list> --> <attribute> <attributes-list>
              |
              <empty>
12.   <attribute> --> SIGNAL     |
              COMPLEX    |
              INTEGER    |
              FLOAT      |
              BLOCKFLOAT |
              EXT
13.   <variable-identifier> --> <identifier>
14.   <procedure-identifier> --> <identifier>
15.   <identifier> --> <letter><string>
16.   <string> --> <letter><string> |
              <digit><string> |
              <empty>
17.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
18.   <letter> --> A | B | C | D | ... | Z
```

Варіант 9

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
               <block>.
3.    <block> --> <declarations> BEGIN <statements-list>
               END
4.    <statements-list> --> <empty>
5.    <declarations> --> <math-function-declarations>
6.    <math-function-declarations> --> DEFFUNC <function-
               list> |
               <empty>
7.    <function-list> --> <function> <function-list> |
               <empty>
8.    <function> --> <function-identifier> =
               <constant><function-characteristic> ;
9.    <function-characteristic> --> \ <unsigned-integer> ,
               <unsigned-integer>
10.   <constant> --> <unsigned-integer>
11.   <procedure-identifier> --> <identifier>
12.   <function-identifier> --> <identifier>
13.   <identifier> --> <letter><string>
14.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
15.   <unsigned-integer> --> <digit><digits-string>
16.   <digits-string> --> <digit><digits-string> |
               <empty>
17.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
18.   <letter> --> A | B | C | D | ... | Z
```

Варіант 10

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
               <block> ;
3.    <block> --> <declarations> BEGIN <statements-list>
               END
4.    <statements-list> --> <empty>
5.    <declarations> --> <procedure-declarations>
6.    <procedure-declarations> --> <procedure> <procedure-
               declarations> |
               <empty>
7.    <procedure> --> PROCEDURE <procedure-
               identifier><parameters-list> ;
8.    <parameters-list> --> ( <declarations-list> ) |
               <empty>
9.    <declarations-list> --> <declaration> <declarations-
               list> |
               <empty>
10.   <declaration> --><variable-identifier><identifiers-
               list>:<attribute><attributes-list> ;
11.   <identifiers-list> --> , <variable-identifier>
               <identifiers-list> |
               <empty>
12.   <attributes-list> --> <attribute> <attributes-list>
               |
               <empty>
13.   <attribute> --> SIGNAL     |
               COMPLEX    |
               INTEGER    |
               FLOAT      |
               BLOCKFLOAT |
               EXT
14.   <variable-identifier> --> <identifier>
15.   <procedure-identifier> --> <identifier>
16.   <identifier> --> <letter><string>
17.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
18.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
19.   <letter> --> A | B | C | D | ... | Z
```

Варіант 11

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
               <block>.
3.    <block> --> <declarations> BEGIN <statements-list>
               END
4.    <declarations> --> <label-declarations>
5.    <label-declarations> --> LABEL <unsigned-integer>
               <labels-list>; |
               <empty>
6.    <labels-list> -->  , <unsigned-integer> <labels-
               list> |
               <empty>
7.    <statements-list> --> <statement> <statements-list>
               |
               <empty>
8.    <statement> --> <unsigned-integer> : <statement> |
               GOTO <unsigned-integer> ; |
               LINK <variable-identifier> , <unsigned-
               integer> ; |
               IN <unsigned-integer>;  |
               OUT <unsigned-integer>;
9.    <variable-identifier> --> <identifier>
10.   <procedure-identifier> --> <identifier>
11.   <identifier> --> <letter><string>
12.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
13.   <unsigned-integer> --> <digit><digits-string>
14.   <digits-string> --> <digit><digits-string> |
               <empty>
15.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
16.   <letter> --> A | B | C | D | ... | Z
```

Варіант 12

```
1.    <signal-program> --> <program>
2.    <program> --> PROCEDURE <procedure-identifier>
               <parameters-list> ; <block> ;
3.    <block> --> <declarations> BEGIN <statements-list>
               END
4.    <declarations> --> <label-declarations>
5.    <label-declarations> --> LABEL <unsigned-integer>
               <labels-list>; |
               <empty>
6.    <labels-list> --> , <unsigned-integer> <labels-list>
               |
               <empty>
7.    <parameters-list> --> ( <variable-identifier>
               <identifiers-list> ) |
               <empty>
8.    <identifiers-list> --> , <variable-identifier>
               <identifiers-list> |
               <empty>
9.    <statements-list> --> <statement> <statements-list>
               |
               <empty>
10.   <statement> --> <unsigned-integer> : <statement> |
               GOTO <unsigned-integer> ; |
               RETURN ; |
               ;  |
               ($ <assembly-insert-file-identifier> $)
11.   <variable-identifier> --> <identifier>
12.   <procedure-identifier> --> <identifier>
13.   <assembly-insert-file-identifier> --> <identifier>
14.   <identifier> --> <letter><string>
15.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
16.   <unsigned-integer> --> <digit><digits-string>
17.   <digits-string> --> <digit><digits-string> |
               <empty>
18.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
19.   <letter> --> A | B | C | D | ... | Z
```

Варіант 13

```
1.    <signal-program> --> <program>
2.    <program> --> PROCEDURE <procedure-
               identifier><parameters-list>; <block> ;
3.    <block> --> <declarations> BEGIN <statements-list>
               END
4.    <declarations> --> <procedure-declarations>
5.    <procedure-declarations> --> <procedure> <procedure-
               declarations> |
               <empty>
6.    <procedure> --> PROCEDURE <procedure-
               identifier><parameters-list> ;
7.    <parameters-list> --> ( <variable-identifier>
               <identifiers-list> ) |
               <empty>
8.    <identifiers-list> --> , <variable-identifier>
               <identifiers-list> |
               <empty>
9.    <statements-list> --> <statement> <statements-list>
               |
               <empty>
10.   <statement> --> <procedure-identifier><actual-
               arguments> ; |
               RETURN ;
11.   <actual-arguments> --> ( <unsigned-integer> <actual-
               arguments-list> )  |
               <empty>
12.   <actual-arguments-list> --> , <unsigned-integer>
               <actual-arguments-list> |
               <empty>
13.   <variable-identifier> --> <identifier>
14.   <procedure-identifier> --> <identifier>
15.   <identifier> --> <letter><string>
16.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
17.   <unsigned-integer> --> <digit><digits-string>
18.   <digits-string> --> <digit><digits-string> |
               <empty>
19.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
20.   <letter> --> A | B | C | D | ... | Z
```

Варіант 14

```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
                 <block>.
3.     <block> --> <declarations> BEGIN <statements-list>
                 END
4.     <declarations> --> <label-declarations>
5.     <label-declarations> --> LABEL <unsigned-integer>
                 <labels-list>; |
                 <empty>
6.     <labels-list> -->  , <unsigned-integer> <labels-
                 list> |
                 <empty>
7.     <statements-list> --> <statement> <statements-list>
                 |
                 <empty>
8.     <statement> --> <unsigned-integer> : <statement> |
                 GOTO <unsigned-integer> ; |
                 <condition-statement> ENDIF ;|
                 ;
9.     <condition-statement> --> <incomplete-condition-
                 statement><alternative-part>
10.    <incomplete-condition-statement> --> IF
                 <conditional-expression> THEN <statements-
                 list>
11.    <conditional-expression> --> <variable-identifier> =
                 <unsigned-integer>
12.    <alternative-part> --> ELSE<statements-list>  |
                 <empty>
13.    <variable-identifier> --> <identifier>
14.    <procedure-identifier> --> <identifier>
15.    <identifier> --> <letter><string>
16.    <string> --> <letter><string> |
                 <digit><string> |
                 <empty>
17.    <unsigned-integer> --> <digit><digits-string>
18.    <digits-string> --> <digit><digits-string> |
                 <empty>
19.    <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
20.    <letter> --> A | B | C | D | ... | Z
```

Варіант 15

```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
                <block> ;
3.     <block> --> BEGIN <statements-list> END
4.     <statements-list> --> <statement> <statements-list>
                |
                <empty>
5.     <statement> --> <unsigned-integer> : <statement> |
                <variable-identifier> := <unsigned-integer>
                ;  |
                <procedure-identifier><actual-arguments> ;
                |
                GOTO <unsigned-integer> ; |
                LINK <variable-identifier> , <unsigned-
                integer> ; |
                IN <unsigned-integer>;  |
                OUT <unsigned-integer>; |
                RETURN ; |
                ;  |
                ($ <assembly-insert-file-identifier> $)
6.     <actual-arguments> --> ( <variable-identifier>
                <actual-arguments-list> )  |
                <empty>
7.     <actual-arguments-list> --> ,<variable-identifier>
                <actual-arguments-list> |
                <empty>
8.     <variable-identifier> --> <identifier>
9.     <procedure-identifier> --> <identifier>
10.    <assembly-insert-file-identifier> --> <identifier>
11.    <identifier> --> <letter><string>
12.    <string> --> <letter><string> |
                <digit><string> |
                <empty>
13.    <unsigned-integer> --> <digit><digits-string>
14.    <digits-string> --> <digit><digits-string> |
                <empty>
15.    <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
16.    <letter> --> A | B | C | D | ... | Z
```

Варіант 16

```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
                <block>.
3.     <block> --> BEGIN <statements-list> END
4.     <statements-list> --> <statement> <statements-list>
                |
                <empty>
5.     <statement> --> <condition-statement> ENDIF ;  |
                WHILE <conditional-expression> DO
                <statements-list> ENDWHILE ;
6.     <condition-statement> --> <incomplete-condition-
                statement><alternative-part>
7.     <incomplete-condition-statement> --> IF
                <conditional-expression> THEN <statements-
                list>
8.     <alternative-part> --> ELSE<statements-list>  |
                <empty>
9.     <conditional-expression> -->
                <expression><comparison-operator>
                <expression>
10.    <comparison-operator> -->  <   |
                <=  |
                =   |
                <>  |
                >=  |
                >
11.    <expression> --> <variable-identifier> |
                <unsigned-integer>
12.    <variable-identifier> --> <identifier>
13.    <procedure-identifier> --> <identifier>
14.    <identifier> --> <letter><string>
15.    <string> --> <letter><string> |
                <digit><string> |
                <empty>
16.    <unsigned-integer> --> <digit><digits-string>
17.    <digits-string> --> <digit><digits-string> |
                <empty>
18.    <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
19.    <letter> --> A | B | C | D | ... | Z
```

Варіант 17

```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
                <block>.
3.     <block> --> BEGIN <statements-list> END
4.     <statements-list> --> <statement> <statements-list>
                |
                <empty>
5.     <statement> --> LOOP <statements-list> ENDLOOP ; |
                FOR <variable-identifier> := <loop-
                declaration> ENDFOR ;
6.     <loop-declaration> --> <expression> TO <expression>
                DO <statements-list>
7.     <expression> --> <summand> <summands-list>   |
                - <summand> <summands-list>
8.     <summands-list> --> <add-instruction> <summand>
                <summands-list> |
                <empty>
9.     <add-instruction> -->     +   |
                                 -
10.    <summand> --> <variable-identifier> |
                <unsigned-integer>
11.    <variable-identifier> --> <identifier>
12.    <procedure-identifier> --> <identifier>
13.    <identifier> --> <letter><string>
14.    <string> --> <letter><string> |
                <digit><string> |
                <empty>
15.    <unsigned-integer> --> <digit><digits-string>
16.    <digits-string> --> <digit><digits-string> |
                <empty>
17.    <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
18.    <letter> --> A | B | C | D | ... | Z
```

Варіант 18

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
               <block>.
3.    <block> --> BEGIN <statements-list> END
4.    <statements-list> --> <statement> <statements-list>
               |
               <empty>
5.    <statement> --> LOOP <statements-list> ENDLOOP ; |
               CASE <expression> OF <alternatives-list>
               ENDCASE ;
6.    <alternatives-list> --> <alternative> <alternatives-
               list> |
               <empty>
7.    <alternative> --> <expression> : / <statements-list>
               \
8.    <expression> --> <multiplier><multipliers-list>
9.    <multipliers-list> --> <multiplication-instruction>
               <multiplier><multipliers-list>  |
               <empty>
10.   <multiplication-instruction> -->     *   |
               /   |
               MOD
11.   <multiplier> --> <variable-identifier> |
               <unsigned-integer>
12.   <variable-identifier> --> <identifier>
13.   <procedure-identifier> --> <identifier>
14.   <identifier> --> <letter><string>
15.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
16.   <unsigned-integer> --> <digit><digits-string>
17.   <digits-string> --> <digit><digits-string> |
               <empty>
18.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
19.   <letter> --> A | B | C | D | ... | Z
```

Варіант 19

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
               <block>.
3.    <block> --> <variable-declarations> BEGIN
               <statements-list> END
4.    <variable-declarations> --> VAR <declarations-list>
               |
               <empty>
5.    <declarations-list> --> <declaration> <declarations-
               list> |
               <empty>
6.    <declaration> --><variable-identifier>:<attribute> ;
7.    <attribute> --> INTEGER    |
               FLOAT
8.    <statements-list> --> <statement> <statements-list>
               |
               <empty>
9.    <statement> --> <condition-statement> ENDIF ;
10.   <condition-statement> --> <incomplete-condition-
               statement><alternative-part>
11.   <incomplete-condition-statement> --> IF
               <conditional-expression> THEN <statements-
               list>
12.   <alternative-part> --> ELSE <statements-list> |
               <empty>
13.   <conditional-expression> --> <expression> =
               <expression>
14.   <expression> --> <variable-identifier> |
               <unsigned-integer>
15.   <variable-identifier> --> <identifier>
16.   <procedure-identifier> --> <identifier>
17.   <identifier> --> <letter><string>
18.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
19.   <unsigned-integer> --> <digit><digits-string>
20.   <digits-string> --> <digit><digits-string> |
               <empty>
21.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
22.   <letter> --> A | B | C | D | ... | Z
```

Варіант 20

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
               <block>.
3.    <block> --> <variable-declarations> BEGIN
               <statements-list> END
4.    <variable-declarations> --> VAR <declarations-list>
               |
               <empty>
5.    <declarations-list> --> <declaration> <declarations-
               list> |
               <empty>
6.    <declaration> --><variable-identifier>:<attribute> ;
7.    <attribute> --> INTEGER    |
               FLOAT
8.    <statements-list> --> <statement> <statements-list>
               |
               <empty>
9.    <statement> --> WHILE <conditional-expression> DO
               <statements-list> ENDWHILE ;
10.   <conditional-expression> --> <expression>
               <comparison-operator><expression>
11.   <comparison-operator> -->  <   |
               <=  |
               =   |
               <>  |
               >=  |
               >
12.   <expression> --> <variable-identifier> |
               <unsigned-integer>
13.   <variable-identifier> --> <identifier>
14.   <procedure-identifier> --> <identifier>
15.   <identifier> --> <letter><string>
16.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
17.   <unsigned-integer> --> <digit><digits-string>
18.   <digits-string> --> <digit><digits-string> |
               <empty>
19.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
20.   <letter> --> A | B | C | D | ... | Z
```

Варіант 21

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
               <block>.
3.    <block> --> <variable-declarations> BEGIN
               <statements-list> END
4.    <variable-declarations> --> VAR <declarations-list>
               |
               <empty>
5.    <declarations-list> --> <declaration> <declarations-
               list> |
               <empty>
6.    <declaration> --><variable-
               identifier>:<attribute><attributes-list> ;
7.    <attributes-list> --> <attribute> <attributes-list>
               |
               <empty>
8.    <attribute> --> INTEGER    |
               FLOAT      |
               [<range>]
9.    <range> --> <unsigned-integer> .. <unsigned-integer>
10.   <statements-list> --> <statement> <statements-list>
               |
               <empty>
11.   <statement> --> <variable> := <expression> ;  |
               LOOP <statements-list> ENDLOOP ;
12.   <expression> --> <variable> |
               <unsigned-integer>
13.   <variable> --> <variable-identifier><dimension>
14.   <dimension> --> [ <expression> ]   |
               <empty>
15.   <variable-identifier> --> <identifier>
16.   <procedure-identifier> --> <identifier>
17.   <identifier> --> <letter><string>
18.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
19.   <unsigned-integer> --> <digit><digits-string>
20.   <digits-string> --> <digit><digits-string> |
               <empty>
21.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
22.   <letter> --> A | B | C | D | ... | Z
```

Варіант 22

```
1.  <signal-program> --> <program>
2.  <program> --> PROGRAM <procedure-identifier> ; <block>.
3.  <block> --> <variable-declarations> BEGIN <statements-
              list> END
4.  <variable-declarations> --> VAR <declarations-list>   |
              <empty>
5.  <declarations-list> --> <declaration> <declarations-
              list> |
              <empty>
6.  <declaration> --><variable-identifier> :
              <attribute><attributes-list> ;
7.  <attributes-list> --> <attribute> <attributes-list> |
              <empty>
8.  <attribute> --> SIGNAL     |
              INTEGER    |
              FLOAT      |
              EXT
9.  <statements-list> --> <statement> <statements-list> |
              <empty>
10. <statement> --> FOR <variable-identifier> := <loop-
              declaration> ENDFOR ;
11. <loop-declaration> --> <expression> TO <expression> DO
              <statements-list>
12. <expression> --> <multiplier><multipliers-list>
13. <multipliers-list> --> <multiplication-instruction>
              <multiplier><multipliers-list> |
              <empty>
14. <multiplication-instruction> -->     *   |
              /    |
              &    |
              MOD
15. <multiplier> --> <variable-identifier> |
              <unsigned-integer>
16. <variable-identifier> --> <identifier>
17. <procedure-identifier> --> <identifier>
18. <identifier> --> <letter><string>
19. <string> --> <letter><string> |
              <digit><string> |
              <empty>
20. <unsigned-integer> --> <digit><digits-string>
21. <digits-string> --> <digit><digits-string> |
              <empty>
22. <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
23. <letter> --> A | B | C | D | ... | Z
```

Варіант 23

```
1.   <signal-program> --> <program>
2.   <program> --> PROGRAM <procedure-identifier> ;
               <block>.
3.   <block> --> <declarations> BEGIN <statements-list> END
4.   <declarations> --> <constant-declarations>
5.   <constant-declarations> --> CONST <constant-
               declarations-list> |
               <empty>
6.   <constant-declarations-list> --> <constant-
               declaration> <constant-declarations-list> |
               <empty>
7.   <constant-declaration> --> <constant-identifier> =
               <constant>;
8.   <statements-list> --> <statement> <statements-list> |
               <empty>
9.   <statement> --> CASE <expression> OF <alternatives-
               list> ENDCASE ;
10.  <alternatives-list> --> <alternative> <alternatives-
               list> |
               <empty>
11.  <alternative> --> <expression> : /<statements-list>\
12.  <expression> --> <summand> <summands-list>   |
               - <summand> <summands-list>
13.  <summands-list> --> <add-instruction> <summand>
               <summands-list> |
               <empty>
14.  <add-instruction> -->     +   |
                               -
15.  <summand> --> <variable-identifier> |
               <unsigned-integer>
16.  <constant> --> <unsigned-integer>
17.  <variable-identifier> --> <identifier>
18.  <constant-identifier> --> <identifier>
19.  <procedure-identifier> --> <identifier>
20.  <identifier> --> <letter><string>
21.  <string> --> <letter><string> |
               <digit><string> |
               <empty>
22.  <unsigned-integer> --> <digit><digits-string>
23.  <digits-string> --> <digit><digits-string> |
               <empty>
24.  <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
25.  <letter> --> A | B | C | D | ... | Z
```

Варіант 24

```
1.  <signal-program> --> <program>
2.  <program> --> PROGRAM <procedure-identifier> ; <block>.
3.  <block> --> <variable-declarations> BEGIN <statements-
                list> END
4.  <variable-declarations> --> VAR <declarations-list>   |
                <empty>
5.  <declarations-list> --> <declaration> <declarations-
                list> |
                <empty>
6.  <declaration> --><variable-identifier> : INTEGER ;
7.  <statements-list> --> <statement> <statements-list> |
                <empty>
8.  <statement> --> <variable-identifier> := <conditional-
                expression> ;
9.  <conditional-expression> --> <logical-summand>
                <logical>
10. <logical> --> OR <logical-summand> <logical>  |
                <empty>
11. <logical-summand> --> <logical-multiplier> <logical-
                multipliers-list>
12. <logical-multipliers-list> --> AND <logical-
                multiplier><logical-multipliers-list> |
                <empty>
13. <logical-multiplier> --> <expression><comparison-
                operator><expression> |
                [ <conditional-expression> ]  |
                NOT <logical-multiplier>
14. <comparison-operator> -->  <   |
                <=  |
                =   |
                <>  |
                >=  |
                >
15. <expression> --> <variable-identifier> |
                <unsigned-integer>
16. <variable-identifier> --> <identifier>
17. <procedure-identifier> --> <identifier>
18. <identifier> --> <letter><string>
19. <string> --> <letter><string> |
                <digit><string> |
                <empty>
20. <unsigned-integer> --> <digit><digits-string>
21. <digits-string> --> <digit><digits-string> | <empty>
22. <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
23. <letter> --> A | B | C | D | ... | Z
```

Варіант 25

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
               <block>.
3.    <block> --> <variable-declarations> BEGIN
               <statements-list> END
4.    <variable-declarations> --> VAR <declarations-list>
               |
               <empty>
5.    <declarations-list> --> <declaration> <declarations-
               list> |
               <empty>
6.    <declaration> --><variable-identifier>: INTEGER ;
7.    <statements-list> --> <statement> <statements-list>
               |
               <empty>
8.    <statement> --> <variable-identifier> :=
               <expression> ;
9.    <expression> --> <summand> <summands-list>   |
               - <summand> <summands-list>
10.   <summands-list> --> <add-instruction> <summand>
               <summands-list> |
               <empty>
11.   <add-instruction> -->     +   |
                                -
12.   <summand> --> <multiplier><multipliers-list>
13.   <multipliers-list> --> <multiplication-instruction>
               <multiplier><multipliers-list>  |
               <empty>
14.   <multiplication-instruction> -->     *   |
                                           /
15.   <multiplier> --> <variable-identifier> |
               <unsigned-integer> |
               ( <expression> )
16.   <variable-identifier> --> <identifier>
17.   <procedure-identifier> --> <identifier>
18.   <identifier> --> <letter><string>
19.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
20.   <unsigned-integer> --> <digit><digits-string>
21.   <digits-string> --> <digit><digits-string> |
               <empty>
22.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
23.   <letter> --> A | B | C | D | ... | Z
```

Варіант 26

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
               <block>.
3.    <block> --> <declarations> BEGIN <statements-list>
               END
4.    <declarations> --> <constant-declarations><variable-
               declarations>
5.    <constant-declarations> --> CONST <constant-
               declarations-list> |
               <empty>
6.    <constant-declarations-list> --> <constant-
               declaration> <constant-declarations-list> |
               <empty>
7.    <constant-declaration> --> <constant-identifier> =
               <constant>;
8.    <constant> --> <complex-constant> |
               <unsigned-constant>   |
               - <unsigned-constant>
9.    <variable-declarations> --> VAR <declarations-list>
               |
               <empty>
10.   <declarations-list> --> <declaration> <declarations-
               list> |
               <empty>
11.   <declaration> --><variable-identifier> <identifiers-
               list>: <attribute><attributes-list> ;
12.   <identifiers-list> --> , <variable-identifier>
               <identifiers-list> |
               <empty>
13.   <attributes-list> --> <attribute> <attributes-list>
               |
               <empty>
14.   <attribute> --> SIGNAL     |
               COMPLEX    |
               INTEGER    |
               FLOAT      |
               BLOCKFLOAT |
               EXT        |
               [<range><ranges-list>]
15.   <ranges-list> -->  ,<range> <ranges-list> |
               <empty>
16.   <range> --> <unsigned-integer> .. <unsigned-integer>
17.   <statements-list> --> <statement> <statements-list>
               |
               <empty>
```

```
18.   <statement> -->
               <condition-statement> ENDIF ; |
               WHILE <conditional-expression> DO
               <statements-list> ENDWHILE ; |
               LINK <variable-identifier> , <unsigned-
               integer> ; |
               IN <unsigned-integer>;  |
               OUT <unsigned-integer>;
19.   <condition-statement> --> <incomplete-condition-
               statement><alternative-part>
20.   <incomplete-condition-statement> --> IF
               <conditional-expression> THEN <statements-
               list>
21.   <alternative-part> --> ELSE<statements-list>  |
               <empty>
22.   <conditional-expression> --> <logical-summand>
               <logical>
23.   <logical> --> OR <logical-summand> <logical>  |
               <empty>
24.   <logical-summand> --> <logical-multiplier> <logical-
               multipliers-list>
25.   <logical-multipliers-list> --> AND <logical-
               multiplier><logical-multipliers-list> |
               <empty>
26.   <logical-multiplier> --> <expression> <comparison-
               operator><expression> |
               [ <conditional-expression> ]  |
               NOT <logical-multiplier>
27.   <comparison-operator> -->  <   |
               <=  |
               =   |
               <>  |
               >=  |
               >
28.   <expression> --> <variable-identifier> |
               <constant-identifier>
29.   <complex-constant> --> '<complex-number>'
30.   <unsigned-constant> --> <unsigned-number>
31.   <complex-number> --> <left-part> <right-part>
32.   <left-part> --> <expression> |
               <empty>
33.   <right-part> --> ,<expression> |
               $EXP( <expression> )   |
               <empty>
34.   <constant-identifier> --> <identifier>
35.   <variable-identifier> --> <identifier>
36.   <identifier> --> <letter><string>
```

```
37.    <string> --> <letter><string> |
                 <digit><string> |
                 <empty>
38.    <unsigned-number> --> <integer-part><fractional-
                 part>
39.    <integer-part> --> <unsigned-integer>
40.    <fractional-part> --> #<sign><unsigned-integer> |
                 <empty>
41.    <unsigned-integer> --> <digit><digits-string>
42.    <digits-string> --> <digit><digits-string> |
                 <empty>
43.    <sign> --> + |
                 - |
                 <empty>
44.    <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
45.    <letter> --> A | B | C | D | ... | Z
```

Варіант 27

```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
                 <block>.
3.     <block> --> BEGIN <conditional-expression> END
4.     <conditional-expression> --> <logical-summand>
                 <logical>
5.     <logical> --> OR <logical-summand> <logical>  |
                 <empty>
6.     <logical-summand> --> <logical-multiplier> <logical-
                 multipliers-list>
7.     <logical-multipliers-list> --> AND <logical-
                 multiplier><logical-multipliers-list> |
                 <empty>
8.     <logical-multiplier> --> <expression><comparison-
                 operator><expression> |
                 [ <conditional-expression> ]  |
                 NOT <logical-multiplier>
9.     <comparison-operator> -->  <   |
                 <=  |
                 =   |
                 <>  |
                 >=  |
                 >
10.    <expression> --> <summand> <summands-list>   |
                 - <summand> <summands-list>
```

```
11.   <summands-list> --> <add-instruction> <summand>
               <summands-list> |
               <empty>
12.   <add-instruction> -->     +   |
               -   |
               !   |
13.   <summand> --> <multiplier><multipliers-list>
14.   <multipliers-list> --> <multiplication-instruction>
               <multiplier><multipliers-list>  |
               <empty>
15.   <multiplication-instruction> -->     *   |
               /   |
               &   |
               MOD
16.   <multiplier> --> <unsigned-constant>  |
               <complex-constant> |
               <variable>  |
               <builtin-function-identifier><actual-
               arguments>  |
               ( <expression> )  |
               - <multiplier>    |
               ^ <multiplier>
17.   <variable> --> <variable-identifier><dimension> |
               <complex-variable>
18.   <complex-variable> --> "<complex-number>"
19.   <dimension> --> [<expression><expressions-list>]   |
               <empty>
20.   <expressions-list> -->  ,<expression><expressions-
               list>  |
               <empty>
21.   <complex-constant> --> '<complex-number>'
22.   <unsigned-constant> --> <unsigned-number>
23.   <complex-number> --> <left-part> <right-part>
24.   <left-part> --> <expression> |
               <empty>
25.   <right-part> --> ,<expression> |
               $EXP( <expression> )   |
               <empty>
26.   <actual-arguments> --> ( <arguments-list> )
27.   <arguments-list> --> <unsigned-integer> <argument-
               list> |
               <empty>
28.   <variable-identifier> --> <identifier>
29.   <procedure-identifier> --> <identifier>
30.   <builtin-function-identifier> --> <identifier>
31.   <identifier> --> <letter><string>
```

```
32.    <string> --> <letter><string> |
                 <digit><string> |
                 <empty>
33.    <unsigned-number> --> <integer-part><fractional-
                 part>
34.    <integer-part> --> <unsigned-integer>
35.    <fractional-part> --> #<sign><unsigned-integer> |
                 <empty>
36.    <unsigned-integer> --> <digit><digits-string>
37.    <digits-string> --> <digit><digits-string> |
                 <empty>
38.    <sign> --> + |
                 - |
                 <empty>
39.    <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
40.    <letter> --> A | B | C | D | ... | Z
```

Варіант 28

```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
                 <block>. |
                 PROCEDURE <procedure-identifier>
                 <parameters-list> ; <block> ;
3.     <block> --> <declarations> BEGIN <statements-list>
                 END
4.     <declarations> --> <label-declarations> <variable-
                 declarations> <math-function-
                 declarations><procedure-declarations>
5.     <label-declarations> --> LABEL <unsigned-integer>
                 <labels-list>; |
                 <empty>
6.     <labels-list> -->  , <unsigned-integer> <labels-
                 list> |
                 <empty>
7.     <variable-declarations> --> VAR <declarations-list>
                 |
                 <empty>
8.     <declarations-list> --> <declaration> <declarations-
                 list> |
                 <empty>
9.     <declaration> --><variable-identifier> <identifiers-
                 list>: <attribute>;
```

```
10.   <identifiers-list> --> , <variable-identifier>
               <identifiers-list> |
               <empty>
11.   <math-function-declarations> --> DEFFUNC <function-
               list> |
               <empty>
12.   <function-list> --> <function><function-list> |
               <empty>
13.   <function> --> <function-identifier> =
               <constant><function-characteristic> ;
14.   <constant> --> <unsigned-integer>
15.   <function-characteristic> --> \ <unsigned-integer> ,
               <unsigned-integer>
16.   <procedure-declarations> --> <procedure> <procedure-
               declarations> |
               <empty>
17.   <procedure> --> PROCEDURE <procedure-
               identifier><parameters-list> ;
18.   <parameters-list> --> ( <attributes-list> ) |
               <empty>
19.   <attributes-list> --> <attribute> , <attributes-
               list> |
               <empty>
20.   <attribute> --> INTEGER    |
               FLOAT      |
               BLOCKFLOAT
21.   <statements-list> --> <statement> <statements-list>
               |
               <empty>
22.   <statement> --> <unsigned-integer> : <statement> |
               <procedure-identifier> <actual-arguments> ;
               |
               LOOP <statements-list> ENDLOOP ; |
               GOTO <unsigned-integer> ; |
               LINK <variable-identifier> , <unsigned-
               integer> ; |
               IN <unsigned-integer>;  |
               OUT <unsigned-integer>; |
               RETURN ; |
               ;  |
               ($ <assembly-insert-file-identifier> $)
23.   <actual-arguments> --> ( <variable-identifier>
               <identifiers-list> )  |
               <empty>
24.   <variable-identifier> --> <identifier>
25.   <procedure-identifier> --> <identifier>
26.   <function-identifier>  --> <identifier>
```

```
27.   <assembly-insert-file-identifier> --> <identifier>
28.   <identifier> --> <letter><string>
29.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
30.   <unsigned-integer> --> <digit><digits-string>
31.   <digits-string> --> <digit><digits-string> |
               <empty>
32.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
33.   <letter> --> A | B | C | D | ... | Z
```

Варіант 29

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
               <block>. |
               PROCEDURE <procedure-identifier>
               <parameters-list> ; <block> ;
3.    <block> --> <declarations> BEGIN <statements-list>
               END
4.    <declarations> --> <constant-declarations>
               <variable-declarations><math-function-
               declarations><procedure-declarations>
5.    <constant-declarations> --> CONST <constant-
               declarations-list> |
               <empty>
6.    <constant-declarations-list> --> <constant-
               declaration> <constant-declarations-list> |
               <empty>
7.    <constant-declaration> --> <constant-identifier> =
               <constant>;
8.    <constant> --> <complex-constant> |
               <unsigned-constant>   |
               - <unsigned-constant>
9.    <variable-declarations> --> VAR <declarations-list>
               |
               <empty>
10.   <declarations-list> --> <declaration> <declarations-
               list> |
               <empty>
11.   <declaration> --><variable-identifier><identifiers-
               list>: <attribute><attributes-list> ;
12.   <identifiers-list> --> , <variable-identifier>
               <identifiers-list> |
               <empty>
```

```
13.   <attributes-list> --> <attribute> <attributes-list>
               |
               <empty>
14.   <attribute> --> SIGNAL     |
               COMPLEX    |
               INTEGER    |
               FLOAT      |
               BLOCKFLOAT |
               EXT        |
               [<range><ranges-list>]
15.   <ranges-list> -->  ,<range> <ranges-list> |
               <empty>
16.   <range> --> <unsigned-integer> .. <unsigned-integer>
17.   <math-function-declarations> --> DEFFUNC <function-
               list> |
               <empty>
18.   <function-list> --> <function> function-list> |
               <empty>
19.   <function> --> <function-identifier> =
               <constant><function-characteristic> ;
20.   <constant> --> <unsigned-integer><function-
               characteristic> --> \ <unsigned-integer> ,
               <unsigned-integer>
21.   <procedure-declarations> --> <procedure> <procedure-
               declarations> |
               <empty>
22.   <procedure> --> PROCEDURE <procedure-identifier>
               <parameters-list> ;
23.   <parameters-list> --> ( <declarations-list> ) |
               <empty>
24.   <statements-list> --> <statement> <statements-list>
               |
               <empty>
25.   <statement> --> LINK <variable-identifier> ,
               <unsigned-integer> ; |
               IN <unsigned-integer>;  |
               OUT <unsigned-integer>;
26.   <complex-constant> --> '<complex-number>'
27.   <unsigned-constant> --> <unsigned-number>
28.   <complex-number> --> <left-part> <right-part>
29.   <left-part> --> <unsigned-integer> |
               <empty>
30.   <right-part> --> ,<unsigned-integer> |
               $EXP( <unsigned-integer> )   |
               <empty>
31.   <constant-identifier> --> <identifier>
32.   <variable-identifier> --> <identifier>
```

```
33.    <procedure-identifier> --> <identifier>
34.    <function-identifier>  --> <identifier>
35.    <identifier> --> <letter><string>
36.    <string> --> <letter><string> |
                <digit><string> |
                <empty>
37.    <unsigned-number> --> <integer-part><fractional-
                part>
38.    <integer-part> --> <unsigned-integer>
39.    <fractional-part> --> #<sign><unsigned-integer> |
                <empty>
40.    <unsigned-integer> --> <digit><digits-string>
41.    <digits-string> --> <digit><digits-string> |
                <empty>
42.    <sign> --> + |
                - |
                <empty>
43.    <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
44.    <letter> --> A | B | C | D | ... | Z
```

Варіант 30

```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
                <block>.
3.     <block> --> <declarations> BEGIN <statements-list>
                END
4.     <declarations> --> <constant-declarations>
                <variable-declarations>
5.     <constant-declarations> --> CONST <constant-
                declarations-list> |
                <empty>
6.     <constant-declarations-list> --> <constant-
                declaration> <constant-declarations-list> |
                <empty>
7.     <constant-declaration> --> <constant-identifier> =
                <constant>;
8.     <constant> -->
                <unsigned-constant>   |
                - <unsigned-constant>
9.     <variable-declarations> --> VAR <declarations-list>
                |
                <empty>
```

```
10.    <declarations-list> --> <declaration> <declarations-
                list> |
                <empty>
11.    <declaration> --><variable-identifier> <identifiers-
                list>: <attribute><attributes-list> ;
12.    <identifiers-list> --> , <variable-identifier>
                <identifiers-list> |
                <empty>
13.    <attributes-list> --> <attribute> <attributes-list>
                |
                <empty>
14.    <attribute> --> SIGNAL     |
                COMPLEX    |
                INTEGER    |
                FLOAT      |
                BLOCKFLOAT |
                EXT        |
                [<range><ranges-list>]
15.    <ranges-list> -->  ,<range> <ranges-list> |
                <empty>
16.    <range> --> <unsigned-integer> .. <unsigned-integer>
17.    <statements-list> --> <statement> <statements-list>
                |
                <empty>
18.    <statement> -->
                LOOP <statements-list> ENDLOOP ; |
                FOR <variable-identifier> := <loop-
                declaration> ENDFOR ; |
                CASE <expression> OF <alternatives-list>
                ENDCASE ;
19.    <loop-declaration> --> <expression> TO <expression>
                DO <statements-list>
20.    <alternatives-list> --> <alternative> <alternatives-
                list> |
                <empty>
21.    <alternative> --> <expression> : / <statements-list>
                \
22.    <expression> --> <summand> <summands-list>   |
                - <summand> <summands-list>
23.    <summands-list> --> <add-instruction> <summand>
                <summands-list> |
                <empty>
24.    <add-instruction> -->     +   |
                -   |
                !
25.    <summand> --> <multiplier><multipliers-list>
```

```
26.   <multipliers-list> --> <multiplication-instruction>
              <multiplier><multipliers-list>  |
              <empty>
27.   <multiplication-instruction> -->     *   |
              /   |
              &   |
              MOD
28.   <multiplier> --> <unsigned-constant>  |
              <constant-identifier>  |
              <variable>  |
              <function-identifier>  |
              <builtin-function-identifier><actual-
              arguments>  |
              ( <expression> )  |
              - <multiplier>    |
              ^ <multiplier>
29.   <variable> --> <variable-identifier><dimension>
30.   <dimension> -->[<expression><expressions-list>] |
              <empty>
31.   <expressions-list> -->  ,<expression><expressions-
              list>  |
              <empty>
32.   <unsigned-constant> --> <unsigned-number>
33.   <constant-identifier> --> <identifier>
34.   <variable-identifier> --> <identifier>
35.   <builtin-function-identifier> --> <identifier>
36.   <identifier> --> <letter><string>
37.   <string> --> <letter><string> |
              <digit><string> |
              <empty>
38.   <unsigned-number> --> <integer-part><fractional-
              part>
39.   <integer-part> --> <unsigned-integer>
40.   <fractional-part> --> #<sign><unsigned-integer> |
              <empty>
41.   <unsigned-integer> --> <digit><digits-string>
42.   <digits-string> --> <digit><digits-string> |
              <empty>
43.   <sign> --> + |
              - |
              <empty>
44.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
45.   <letter> --> A | B | C | D | ... | Z
```

Варіант 31

```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
                 <block>. |
                 PROCEDURE <procedure-identifier>
                 <parameters-list> ; <block> ;
3.     <block> --> <declarations> BEGIN <statements-list>
                 END
4.     <declarations> --> <constant-declarations><variable-
                 declarations><math-function-
                 declarations><procedure-declarations>
5.     <constant-declarations> --> CONST <constant-
                 declarations-list> |
                 <empty>
6.     <constant-declarations-list> --> <constant-
                 declaration> <constant-declarations-list> |
                 <empty>
7.     <constant-declaration> --> <constant-identifier> =
                 <constant>;
8.     <constant> -->
                 <unsigned-constant>   |
                 - <unsigned-constant>
9.     <variable-declarations> --> VAR <declarations-list>
                 |
                 <empty>
10.    <declarations-list> --> <declaration> <declarations-
                 list> |
                 <empty>
11.    <declaration> --><variable-identifier> <identifiers-
                 list>: <attribute><attributes-list> ;
12.    <identifiers-list> --> , <variable-identifier>
                 <identifiers-list> |
                 <empty>
13.    <attributes-list> --> <attribute> <attributes-list>
                 |
                 <empty>
14.    <attribute> --> SIGNAL     |
                 INTEGER    |
                 FLOAT      |
                 BLOCKFLOAT |
                 EXT        |
                 [<range><ranges-list>]
15.    <ranges-list> -->  ,<range> <ranges-list> |
                 <empty>
16.    <range> --> <unsigned-integer> .. <unsigned-integer>
```

```
17.   <math-function-declarations> --> DEFFUNC <function-
                list> |
                <empty>
18.   <function-list> --> <function><function-list> |
                <empty>
19.   <function> --> <function-identifier> =
                <expression><function-characteristic> ;
20.   <function-characteristic> --> \ <unsigned-integer> ,
                <unsigned-integer>
21.   <procedure-declarations> --> <procedure> <procedure-
                declarations> |
                <empty>
22.   <procedure> --> PROCEDURE <procedure-
                identifier><parameters-list> ;
23.   <parameters-list> --> ( <declarations-list> ) |
                <empty>
24.   <statements-list> --> <statement> <statements-list>
                |
                <empty>
25.   <statement> --> <variable> := <expression> ;  |
                <procedure-identifier><actual-arguments> ;
                |
                <condition-statement> ENDIF ; |
                WHILE <conditional-expression> DO
                <statements-list> ENDWHILE ; |
                LOOP <statements-list> ENDLOOP ; |
                FOR <variable-identifier> := <loop-
                declaration> ENDFOR ; |
                CASE <expression> OF <alternatives-list>
                ENDCASE ; |
                LINK <variable-identifier> , <unsigned-
                integer> ; |
                IN <unsigned-integer>;  |
                OUT <unsigned-integer>; |
                RETURN ; |
                ;
26.   <condition-statement> --> <incomplete-condition-
                statement><alternative-part>
27.   <incomplete-condition-statement> --> IF
                <conditional-expression> THEN <statements-
                list>
28.   <alternative-part> --> ELSE<statements-list>  |
                <empty>
29.   <loop-declaration> --> <expression> TO <expression>
                DO <statements-list>
```

```
30.   <actual-arguments> --> ( <expression> <actual-
                arguments-list> )  |
                <empty>
31.   <actual-arguments-list> --> ,<expression> <actual-
                arguments-list> |
                <empty>
32.   <alternatives-list> --> <alternative> <alternatives-
                list> |
                <empty>
33.   <alternative> --> <expression> : / <statements-list>
                \
34.   <conditional-expression> --> <logical-summand>
                <logical>
35.   <logical> --> OR <logical-summand> <logical>  |
                <empty>
36.   <logical-summand> --> <logical-multiplier> <logical-
                multipliers-list>
37.   <logical-multipliers-list> --> AND <logical-
                multiplier><logical-multipliers-list> |
                <empty>
38.   <logical-multiplier> --> <expression> <comparison-
                operator><expression> |
                [ <conditional-expression> ]  |
                NOT <logical-multiplier>
39.   <comparison-operator> -->  <   |
                <=  |
                =   |
                <>  |
                >=  |
                >
40.   <expression> --> <summand> <summands-list>   |
                - <summand> <summands-list>
41.   <summands-list> --> <add-instruction> <summand>
                <summands-list> |
                <empty>
42.   <add-instruction> -->     +   |
                -   |
                !   |
43.   <summand> --> <multiplier><multipliers-list>
44.   <multipliers-list> --> <multiplication-instruction>
                <multiplier><multipliers-list>  |
                <empty>
45.   <multiplication-instruction> -->     *   |
                /   |
                &   |
                MOD
```

```
46.    <multiplier> --> <unsigned-constant>  |
                 <constant-identifier>  |
                 <variable>  |
                 <function-identifier>  |
                 <builtin-function-identifier><actual-
                 arguments>  |
                 ( <expression> )  |
                 - <multiplier>    |
                 ^ <multiplier>
47.    <variable> --> <variable-identifier><dimension>
48.    <dimension> --> [ <expression> <expressions-list> ]
                 |
                 <empty>
49.    <expressions-list> -->  ,<expression><expressions-
                 list>  |
                 <empty>
50.    <unsigned-constant> --> <unsigned-number>
51.    <constant-identifier> --> <identifier>
52.    <variable-identifier> --> <identifier>
53.    <procedure-identifier> --> <identifier>
54.    <function-identifier>  --> <identifier>
55.    <builtin-function-identifier> --> <identifier>
56.    <identifier> --> <letter><string>
57.    <string> --> <letter><string> |
                 <digit><string> |
                 <empty>
58.    <unsigned-number> --> <integer-part><fractional-
                 part>
59.    <integer-part> --> <unsigned-integer>
60.    <fractional-part> --> #<sign><unsigned-integer> |
                 <empty>
61.    <unsigned-integer> --> <digit><digits-string>
62.    <digits-string> --> <digit><digits-string> |
                 <empty>
63.    <sign> --> + |
                 - |
                 <empty>
64.    <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
65.    <letter> --> A | B | C | D | ... | Z
```

Варіант 32

```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
                 <block>.
```

```
3.     <block> --> <declarations> BEGIN <statements-list>
                END
4.     <declarations> --> <label-declarations><constant-
                declarations><variable-declarations>
5.     <label-declarations> --> LABEL <unsigned-integer>
                <labels-list>; |
                <empty>
6.     <labels-list> -->  , <unsigned-integer> <labels-
                list> |
                <empty>
7.     <constant-declarations> --> CONST <constant-
                declarations-list> |
                <empty>
8.     <constant-declarations-list> --> <constant-
                declaration> <constant-declarations-list> |
                <empty>
9.     <constant-declaration> --> <constant-identifier> =
                <constant>;
10.    <constant> --> <complex-constant> |
                <unsigned-constant>   |
                - <unsigned-constant>
11.    <variable-declarations> --> VAR <declarations-list>
                |
                <empty>
12.    <declarations-list> --> <declaration> <declarations-
                list> |
                <empty>
13.    <declaration> --><variable-identifier> <identifiers-
                list>: <attribute><attributes-list> ;
14.    <identifiers-list> --> , <variable-identifier>
                <identifiers-list> |
                <empty>
15.    <attributes-list> --> <attribute> <attributes-list>
                |
                <empty>
16.    <attribute> --> SIGNAL     |
                COMPLEX    |
                INTEGER    |
                FLOAT      |
                BLOCKFLOAT |
                EXT        |
                [<range><ranges-list>]
17.    <ranges-list> -->  ,<range> <ranges-list> |
                <empty>
18.    <range> --> <unsigned-integer> .. <unsigned-integer>
```

```
19.   <statements-list> --> <statement> <statements-list>
                |
                <empty>
20.   <statement> --> <unsigned-integer> : <statement> |
                <variable> := <expression> ;  |
                <condition-statement> ENDIF ; |
                WHILE <conditional-expression> DO
                <statements-list> ENDWHILE ; |
                LOOP <statements-list> ENDLOOP ; |
                FOR <variable-identifier> := <loop-
                declaration> ENDFOR ; |
                CASE <expression> OF <alternatives-list>
                ENDCASE ; |
                GOTO <unsigned-integer> ; |
                LINK <variable-identifier> , <unsigned-
                integer> ; |
                IN <unsigned-integer>;  |
                OUT <unsigned-integer>; |
                ;
21.   <condition-statement> --> <incomplete-condition-
                statement><alternative-part>
22.   <incomplete-condition-statement> --> IF
                <conditional-expression> THEN <statements-
                list>
23.   <alternative-part> --> ELSE<statements-list>  |
                <empty>
24.   <loop-declaration> --> <expression> TO <expression>
                DO <statements-list>
25.   <alternatives-list> --> <alternative> <alternatives-
                list> |
                <empty>
26.   <alternative> --> <expression> : / <statements-list>
                \
27.   <conditional-expression> --> <logical-summand>
                <logical>
28.   <logical> --> OR <logical-summand> <logical>  |
                <empty>
29.   <logical-summand> --> <logical-multiplier> <logical-
                multipliers-list>
30.   <logical-multipliers-list> --> AND <logical-
                multiplier><logical-multipliers-list> |
                <empty>
31.   <logical-multiplier> --> <expression> <comparison-
                operator> <expression> |
                [ <conditional-expression> ]  |
                NOT <logical-multiplier>
```

```
32.   <comparison-operator> -->  <   |
              <=  |
              =   |
              <>  |
              >=  |
              >
33.   <expression> --> <summand> <summands-list>   |
              - <summand> <summands-list>
34.   <summands-list> --> <add-instruction> <summand>
              <summands-list> |
              <empty>
35.   <add-instruction> -->     +   |
              -   |
              !
36.   <summand> --> <multiplier><multipliers-list>
37.   <multipliers-list> --> <multiplication-instruction>
              <multiplier><multipliers-list>  |
              <empty>
38.   <multiplication-instruction> -->     *   |
              /   |
              &   |
              MOD
39.   <multiplier> --> <unsigned-constant>  |
              <complex-constant> |
              <constant-identifier>  |
              <variable>  |
              <function-identifier>  |
              <builtin-function-identifier><actual-
              arguments>  |
              ( <expression> )  |
              - <multiplier>    |
              ^ <multiplier>
40.   <variable> --> <variable-identifier><dimension> |
              <complex-variable>
41.   <complex-variable> --> "<complex-number>"
42.   <dimension> --> [ <expression> <expressions-list> ]
              |
              <empty>
43.   <expressions-list> -->  ,<expression><expressions-
              list>  |
              <empty>
44.   <complex-constant> --> '<complex-number>'
45.   <unsigned-constant> --> <unsigned-number>
46.   <complex-number> --> <left-part> <right-part>
47.   <left-part> --> <expression> |
              <empty>
```

```
48.    <right-part> --> ,<expression> |
                 $EXP( <expression> )   |
                 <empty>
49.    <constant-identifier> --> <identifier>
50.    <variable-identifier> --> <identifier>
51.    <builtin-function-identifier> --> <identifier>
52.    <identifier> --> <letter><string>
53.    <string> --> <letter><string> |
                 <digit><string> |
                 <empty>
54.    <unsigned-number> --> <integer-part><fractional-
                 part>
55.    <integer-part> --> <unsigned-integer>
56.    <fractional-part> --> #<sign><unsigned-integer> |
                 <empty>
57.    <unsigned-integer> --> <digit><digits-string>
58.    <digits-string> --> <digit><digits-string> |
                 <empty>
59.    <sign> --> + |
                 - |
                 <empty>
60.    <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
61.    <letter> --> A | B | C | D | ... | Z
```

Варіант 33

```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
                 <block>.
3.     <block> --> <declarations> BEGIN <statements-list>
                 END
4.     <declarations> --> <constant-declarations><variable-
                 declarations><math-function-declarations>
5.     <constant-declarations> --> CONST <constant-
                 declarations-list> |
                 <empty>
6.     <constant-declarations-list> --> <constant-
                 declaration> <constant-declarations-list> |
                 <empty>
7.     <constant-declaration> --> <constant-identifier> =
                 <constant>;
8.     <constant> --> <complex-constant> |
                 <unsigned-constant>   |
                 - <unsigned-constant>
```

```
9.     <variable-declarations> --> VAR <declarations-list>
                 |
                 <empty>
10.    <declarations-list> --> <declaration> <declarations-
                 list> |
                 <empty>
11.    <declaration> --><variable-identifier> <identifiers-
                 list>: <attribute><attributes-list> ;
12.    <identifiers-list> --> , <variable-identifier>
                 <identifiers-list> |
                 <empty>
13.    <attributes-list> --> <attribute> <attributes-list>
                 |
                 <empty>
14.    <attribute> --> COMPLEX    |
                 INTEGER    |
                 FLOAT      |
                 BLOCKFLOAT |
                 EXT        |
                 [<range><ranges-list>]
15.    <ranges-list> -->  ,<range> <ranges-list> |
                 <empty>
16.    <range> --> <unsigned-integer> .. <unsigned-integer>
17.    <math-function-declarations> --> DEFFUNC <function-
                 list> |
                 <empty>
18.    <function-list> --> <function><function-list> |
                 <empty>
19.    <function> --> <function-identifier> =
                 <expression><function-characteristic> ;
20.    <function-characteristic> --> \ <unsigned-integer> ,
                 <unsigned-integer>
21.    <statements-list> --> <statement> <statements-list>
                 |
                 <empty>
22.    <statement> --> <variable> := <expression> ;  |
                 <condition-statement> ENDIF ; |
                 WHILE <conditional-expression> DO
                 <statements-list> ENDWHILE ; |
                 LOOP <statements-list> ENDLOOP ; |
                 FOR <variable-identifier> := <loop-
                 declaration> ENDFOR ; |
                 CASE <expression> OF <alternatives-list>
                 ENDCASE ; |
                 LINK <variable-identifier> , <unsigned-
                 integer> ; |
                 IN <unsigned-integer>;  |
```

```
             OUT <unsigned-integer>; |
             ;
23.   <condition-statement> --> <incomplete-condition-
             statement><alternative-part>
24.   <incomplete-condition-statement> --> IF
             <conditional-expression> THEN <statements-
             list>
25.   <alternative-part> --> ELSE<statements-list>  |
             <empty>
26.   <loop-declaration> --> <expression> TO <expression>
             DO <statements-list>
27.   <alternatives-list> --> <alternative> <alternatives-
             list> |
             <empty>
28.   <alternative> --> <expression> : / <statements-list>
             \
29.   <conditional-expression> --> <logical-summand>
             <logical>
30.   <logical> --> OR <logical-summand> <logical>  |
             <empty>
31.   <logical-summand> --> <logical-multiplier> <logical-
             multipliers-list>
32.   <logical-multipliers-list> --> AND <logical-
             multiplier><logical-multipliers-list> |
             <empty>
33.   <logical-multiplier> --> <expression> <comparison-
             operator><expression> |
             [ <conditional-expression> ]  |
             NOT <logical-multiplier>
34.   <comparison-operator> -->  <   |
             <=  |
             =   |
             <>  |
             >=  |
             >
35.   <expression> --> <summand> <summands-list>   |
             - <summand> <summands-list>
36.   <summands-list> --> <add-instruction> <summand>
             <summands-list> |
             <empty>
37.   <add-instruction> -->     +   |
             -   |
             !
38.   <summand> --> <multiplier><multipliers-list>
39.   <multipliers-list> --> <multiplication-instruction>
             <multiplier><multipliers-list>  |
             <empty>
```

```
40.   <multiplication-instruction> -->     *   |
               /   |
               &   |
               MOD
41.   <multiplier> --> <unsigned-constant>  |
               <complex-constant> |
               <constant-identifier>  |
               <variable>  |
               <function-identifier>  |
               <builtin-function-identifier><actual-
               arguments>  |
               ( <expression> )  |
               - <multiplier>    |
               ^ <multiplier>
42.   <variable> --> <variable-identifier><dimension> |
               <complex-variable>
43.   <complex-variable> --> "<complex-number>"
44.   <dimension> --> [ <expression> <expressions-list> ]
               |
               <empty>
45.   <expressions-list> -->  ,<expression><expressions-
               list>  |
               <empty>
46.   <complex-constant> --> '<complex-number>'
47.   <unsigned-constant> --> <unsigned-number>
48.   <complex-number> --> <left-part> <right-part>
49.   <left-part> --> <expression> |
               <empty>
50.   <right-part> --> ,<expression> |
               $EXP( <expression> )   |
               <empty>
51.   <constant-identifier> --> <identifier>
52.   <variable-identifier> --> <identifier>
53.   <function-identifier>  --> <identifier>
54.   <builtin-function-identifier> --> <identifier>
55.   <identifier> --> <letter><string>
56.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
57.   <unsigned-number> --> <integer-part><fractional-
               part>
58.   <integer-part> --> <unsigned-integer>
59.   <fractional-part> --> #<sign><unsigned-integer> |
               <empty>
60.   <unsigned-integer> --> <digit><digits-string>
61.   <digits-string> --> <digit><digits-string> |
               <empty>
```

```
62.    <sign> --> + |
               - |
               <empty>
63.    <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
64.    <letter> --> A | B | C | D | ... | Z
```

Варіант 34

```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
               <block>. |
               PROCEDURE <procedure-identifier>
               <parameters-list> ; <block> ;
3.     <block> --> <declarations> BEGIN <statements-list>
               END
4.     <declarations> --> <label-declarations><constant-
               declarations><variable-
               declarations><procedure-declarations>
5.     <label-declarations> --> LABEL <unsigned-integer>
               <labels-list>; |
               <empty>
6.     <labels-list> -->  , <unsigned-integer> <labels-
               list> |
               <empty>
7.     <constant-declarations> --> CONST <constant-
               declarations-list> |
               <empty>
8.     <constant-declarations-list> --> <constant-
               declaration> <constant-declarations-list> |
               <empty>
9.     <constant-declaration> --> <constant-identifier> =
               <constant>;
10.    <constant> -->
               <unsigned-constant>   |
               - <unsigned-constant>
11.    <variable-declarations> --> VAR <declarations-list>
               |
               <empty>
12.    <declarations-list> --> <declaration> <declarations-
               list> |
               <empty>
13.    <declaration> --><variable-identifier><identifiers-
               list>:<attribute><attributes-list> ;
```

```
14.   <identifiers-list> --> , <variable-identifier>
               <identifiers-list> |
               <empty>
15.   <attributes-list> --> <attribute> <attributes-list>
               |
               <empty>
16.   <attribute> --> SIGNAL     |
               INTEGER    |
               FLOAT      |
               BLOCKFLOAT |
               EXT        |
               [<range><ranges-list>]
17.   <ranges-list> -->  ,<range> <ranges-list> |
               <empty>
18.   <range> --> <unsigned-integer> .. <unsigned-integer>
19.   <procedure-declarations> --> <procedure> <procedure-
               declarations> |
               <empty>
20.   <procedure> --> PROCEDURE <procedure-
               identifier><parameters-list> ;
21.   <parameters-list> --> ( <declarations-list> ) |
               <empty>
22.   <statements-list> --> <statement> <statements-list>
               |
               <empty>
23.   <statement> --> <unsigned-integer> : <statement> |
               <variable> := <expression> ;  |
               <procedure-identifier><actual-arguments> ;
               |
               <condition-statement> ENDIF ; |
               WHILE <conditional-expression> DO
               <statements-list> ENDWHILE ; |
               LOOP <statements-list> ENDLOOP ; |
               FOR <variable-identifier> := <loop-
               declaration> ENDFOR ; |
               CASE <expression> OF <alternatives-list>
               ENDCASE ; |
               GOTO <unsigned-integer> ; |
               LINK <variable-identifier> , <unsigned-
               integer> ; |
               IN <unsigned-integer>;  |
               OUT <unsigned-integer>; |
               RETURN ; |
               ;
24.   <condition-statement> --> <incomplete-condition-
               statement><alternative-part>
```

```
25.   <incomplete-condition-statement> --> IF
               <conditional-expression> THEN <statements-
               list>
26.   <alternative-part> --> ELSE<statements-list>  |
               <empty>
27.   <loop-declaration> --> <expression> TO <expression>
               DO <statements-list>
28.   <actual-arguments> --> ( <expression> <actual-
               arguments-list> )  |
               <empty>
29.   <actual-arguments-list> --> ,<expression> <actual-
               arguments-list> |
               <empty>
30.   <alternatives-list> --> <alternative> <alternatives-
               list> |
               <empty>
31.   <alternative> --> <expression> : / <statements-list>
               \
32.   <conditional-expression> --> <logical-summand>
               <logical>
33.   <logical> --> OR <logical-summand> <logical>  |
               <empty>
34.   <logical-summand> --> <logical-multiplier> <logical-
               multipliers-list>
35.   <logical-multipliers-list> --> AND <logical-
               multiplier><logical-multipliers-list> |
               <empty>
36.   <logical-multiplier> --> <expression> <comparison-
               operator><expression> |
               [ <conditional-expression> ]  |
               NOT <logical-multiplier>
37.   <comparison-operator> -->  <   |
               <=  |
               =   |
               <>  |
               >=  |
               >
38.   <expression> --> <summand> <summands-list>   |
               - <summand> <summands-list>
39.   <summands-list> --> <add-instruction> <summand>
               <summands-list> |
               <empty>
40.   <add-instruction> -->     +   |
               -   |
               !
41.   <summand> --> <multiplier><multipliers-list>
```

```
42.    <multipliers-list> --> <multiplication-instruction>
                 <multiplier><multipliers-list>  |
                 <empty>
43.    <multiplication-instruction> -->     *   |
                 /   |
                 &   |
                 MOD
44.    <multiplier> --> <unsigned-constant>  |
                 <constant-identifier>  |
                 <variable>  |
                 <builtin-function-identifier><actual-
                 arguments>  |
                 ( <expression> )  |
                 - <multiplier>    |
                 ^ <multiplier>
45.    <variable> --> <variable-identifier><dimension>
46.    <dimension> --> [ <expression> <expressions-list> ]
                 |
                 <empty>
47.    <expressions-list> -->  ,<expression><expressions-
                 list>  |
                 <empty>
48.    <unsigned-constant> --> <unsigned-number>
49.    <constant-identifier> --> <identifier>
50.    <variable-identifier> --> <identifier>
51.    <procedure-identifier> --> <identifier>
52.    <builtin-function-identifier> --> <identifier>
53.    <identifier> --> <letter><string>
54.    <string> --> <letter><string> |
                 <digit><string> |
                 <empty>
55.    <unsigned-number> --> <integer-part><fractional-
                 part>
56.    <integer-part> --> <unsigned-integer>
57.    <fractional-part> --> #<sign><unsigned-integer> |
                 <empty>
58.    <unsigned-integer> --> <digit><digits-string>
59.    <digits-string> --> <digit><digits-string> |
                 <empty>
60.    <sign> --> + |
                 - |
                 <empty>
61.    <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
62.    <letter> --> A | B | C | D | ... | Z
```

Варіант 35

```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
                <block>. |
                PROCEDURE <procedure-identifier>
                <parameters-list> ; <block> ;
3.     <block> --> <declarations> BEGIN <statements-list>
                END
4.     <declarations> --> <constant-declarations><variable-
                declarations><math-function-
                declarations><procedure-declarations>
5.     <constant-declarations> --> CONST <constant-
                declarations-list> |
                <empty>
6.     <constant-declarations-list> --> <constant-
                declaration> <constant-declarations-list> |
                <empty>
7.     <constant-declaration> --> <constant-identifier> =
                <constant>;
8.     <constant> --> <complex-constant> |
                <unsigned-constant>   |
                - <unsigned-constant>
9.     <variable-declarations> --> VAR <declarations-list>
                |
                <empty>
10.    <declarations-list> --> <declaration> <declarations-
                list> |
                <empty>
11.    <declaration> --><variable-identifier> <identifiers-
                list> : <attribute><attributes-list> ;
12.    <identifiers-list> --> , <variable-identifier>
                <identifiers-list> |
                <empty>
13.    <attributes-list> --> <attribute> <attributes-list>
                |
                <empty>
14.    <attribute> -->
                COMPLEX    |
                INTEGER    |
                FLOAT      |
                BLOCKFLOAT |
                EXT
15.    <math-function-declarations> --> DEFFUNC <function-
                list> |
                <empty>
```

```
16.   <function-list> --> <function><function-list> |
               <empty>
17.   <function> --> <function-identifier> =
               <expression><function-characteristic> ;
18.   <function-characteristic> --> \ <unsigned-integer> ,
               <unsigned-integer>
19.   <procedure-declarations> --> <procedure> <procedure-
               declarations> |
               <empty>
20.   <procedure> --> PROCEDURE <procedure-
               identifier><parameters-list> ;
21.   <parameters-list> --> ( <declarations-list> ) |
               <empty>
22.   <statements-list> --> <statement> <statements-list>
               |
               <empty>
23.   <statement> --> <variable> := <expression> ;  |
               <procedure-identifier><actual-arguments> ;
               |
               <condition-statement> ENDIF ; |
               WHILE <conditional-expression> DO
               <statements-list> ENDWHILE ; |
               LOOP <statements-list> ENDLOOP ; |
               FOR <variable-identifier> := <loop-
               declaration> ENDFOR ; |
               CASE <expression> OF <alternatives-list>
               ENDCASE ; |
               RETURN ; |
               ;  |
               ($ <assembly-insert-file-identifier> $)
24.   <condition-statement> --> <incomplete-condition-
               statement><alternative-part>
25.   <incomplete-condition-statement> --> IF
               <conditional-expression> THEN <statements-
               list>
26.   <alternative-part> --> ELSE<statements-list>  |
               <empty>
27.   <loop-declaration> --> <expression> TO <expression>
               DO <statements-list>
28.   <actual-arguments> --> ( <expression> <actual-
               arguments-list> )  |
               <empty>
29.   <actual-arguments-list> --> ,<expression> <actual-
               arguments-list> |
               <empty>
```

```
30.    <alternatives-list> --> <alternative> <alternatives-
                list> |
                <empty>
31.    <alternative> --> <expression> : / <statements-list>
                \
32.    <conditional-expression> --> <logical-summand>
                <logical>
33.    <logical> --> OR <logical-summand> <logical>  |
                <empty>
34.    <logical-summand> --> <logical-multiplier> <logical-
                multipliers-list>
35.    <logical-multipliers-list> --> AND <logical-
                multiplier><logical-multipliers-list> |
                <empty>
36.    <logical-multiplier> --> <expression> <comparison-
                operator><expression> |
                [ <conditional-expression> ]  |
                NOT <logical-multiplier>
37.    <comparison-operator> -->  <   |
                <=  |
                =   |
                <>  |
                >=  |
                >
38.    <expression> --> <summand> <summands-list>   |
                - <summand> <summands-list>
39.    <summands-list> --> <add-instruction> <summand>
                <summands-list> |
                <empty>
40.    <add-instruction> -->     +   |
                -   |
                !
41.    <summand> --> <multiplier><multipliers-list>
42.    <multipliers-list> --> <multiplication-instruction>
                <multiplier><multipliers-list>  |
                <empty>
43.    <multiplication-instruction> -->     *   |
                /   |
                &   |
                MOD
44.    <multiplier> --> <unsigned-constant>  |
                <complex-constant> |
                <constant-identifier>  |
                <variable>  |
                <function-identifier>  |
                <builtin-function-identifier><actual-
                arguments>  |
```

```
               ( <expression> )  |
               - <multiplier>    |
               ^ <multiplier>
45.   <variable> --> <variable-identifier><dimension> |
               <complex-variable>
46.   <complex-variable> --> "<complex-number>"
47.   <dimension> --> [ <expression> <expressions-list> ]
               |
               <empty>
48.   <expressions-list> -->  ,<expression><expressions-
               list>  |
               <empty>
49.   <complex-constant> --> '<complex-number>'
50.   <unsigned-constant> --> <unsigned-number>
51.   <complex-number> --> <left-part> <right-part>
52.   <left-part> --> <expression> |
               <empty>
53.   <right-part> --> ,<expression> |
               $EXP( <expression> )   |
               <empty>
54.   <constant-identifier> --> <identifier>
55.   <variable-identifier> --> <identifier>
56.   <procedure-identifier> --> <identifier>
57.   <function-identifier>  --> <identifier>
58.   <builtin-function-identifier> --> <identifier>
59.   <assembly-insert-file-identifier> --> <identifier>
60.   <identifier> --> <letter><string>
61.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
62.   <unsigned-number> --> <integer-part><fractional-
               part>
63.   <integer-part> --> <unsigned-integer>
64.   <fractional-part> --> #<sign><unsigned-integer> |
               <empty>
65.   <unsigned-integer> --> <digit><digits-string>
66.   <digits-string> --> <digit><digits-string> |
               <empty>
67.   <sign> --> + |
                 - |
                 <empty>
68.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
69.   <letter> --> A | B | C | D | ... | Z
```

ДОДАТОК 2. ГРАМАТИКА МОВИ SIGNAL [7]


```
1.     <signal-program> --> <program>
2.     <program> --> PROGRAM <procedure-identifier> ;
                 <block>. |
                 PROCEDURE <procedure-
                 identifier><parameters-list> ; <block> ;
3.     <block> --> <declarations> BEGIN <statements-list>
                 END
4.     <declarations> --> <label-declarations><constant-
                 declarations><variable-declarations><math-
                 function-declarations><procedure-
                 declarations>
5.     <label-declarations> --> LABEL <unsigned-integer>
                 <labels-list>; |
                 <empty>
6.     <labels-list> -->  , <unsigned-integer> <labels-
                 list> |
                 <empty>
7.     <constant-declarations> --> CONST <constant-
                 declarations-list> |
                 <empty>
8.     <constant-declarations-list> --> <constant-
                 declaration> <constant-declarations-list> |
                 <empty>
9.     <constant-declaration> --> <constant-identifier> =
                 <constant>;
10.    <constant> --> <complex-constant> |
                 <unsigned-constant>   |
                 - <unsigned-constant>
11.    <variable-declarations> --> VAR <declarations-list>
                 |
                 <empty>
12.    <declarations-list> --> <declaration> <declarations-
                 list> |
                 <empty>
13.    <declaration> --><variable-identifier><identifiers-
                 list>:<attribute><attributes-list> ;
14.    <identifiers-list> --> , <variable-identifier>
                 <identifiers-list> |
                 <empty>
```

```
15.   <attributes-list> --> <attribute> <attributes-list>
               |
               <empty>
16.   <attribute> --> SIGNAL     |
               COMPLEX    |
               INTEGER    |
               FLOAT      |
               BLOCKFLOAT |
               EXT        |
               [<range><ranges-list>]
17.   <ranges-list> -->  ,<range> <ranges-list> |
               <empty>
18.   <range> --> <unsigned-integer> .. <unsigned-integer>
19.   <math-function-declarations> --> DEFFUNC <function-
               list> |
               <empty>
20.   <function-list> --> <function> <function-list> |
               <empty>
21.   <function> --> <function-identifier> =
               <expression><function-characteristic> ;
22.   <function-characteristic> --> \ <unsigned-integer> ,
               <unsigned-integer>
23.   <procedure-declarations> --> <procedure> <procedure-
               declarations> |
               <empty>
24.   <procedure> --> PROCEDURE <procedure-
               identifier><parameters-list> ;
25.   <parameters-list> --> ( <declarations-list> ) |
               <empty>
26.   <statements-list> --> <statement> <statements-list>
               |
               <empty>
27.   <statement> --> <unsigned-integer> : <statement> |
               <variable> := <expression> ;  |
               <procedure-identifier><actual-arguments> ;
               |
               <condition-statement> ENDIF ; |
               WHILE <conditional-expression> DO
               <statements-list> ENDWHILE ; |
               LOOP <statements-list> ENDLOOP ; |
               FOR <variable-identifier> := <loop-
               declaration> ENDFOR ; |
               CASE <expression> OF <alternatives-list>
               ENDCASE ; |
               GOTO <unsigned-integer> ; |
               LINK <variable-identifier> , <unsigned-
               integer> ; |
```

```
              IN <unsigned-integer>;  |
              OUT <unsigned-integer>; |
              RETURN ; |
              ;  |
              ($ <assembly-insert-file-identifier> $)
28.   <condition-statement> --> <incomplete-condition-
              statement><alternative-part>
29.   <incomplete-condition-statement> --> IF
              <conditional-expression> THEN <statements-
              list>
30.   <alternative-part> --> ELSE <statements-list>  |
              <empty>
31.   <loop-declaration> --> <expression> TO <expression>
              DO <statements-list>
32.   <actual-arguments> --> ( <expression> <actual-
              arguments-list> )  |
              <empty>
33.   <actual-arguments-list> --> ,<expression> <actual-
              arguments-list> |
              <empty>
34.   <alternatives-list> --> <alternative> <alternatives-
              list> |
              <empty>
35.   <alternative> --> <expression> : / <statements-list>
              \
36.   <conditional-expression> --> <logical-summand>
              <logical>
37.   <logical> --> OR <logical-summand> <logical>  |
              <empty>
38.   <logical-summand> --> <logical-multiplier> <logical-
              multipliers-list>
39.   <logical-multipliers-list> --> AND <logical-
              multiplier> <logical-multipliers-list> |
              <empty>
40.   <logical-multiplier> --> <expression><comparison-
              operator><expression> |
              [ <conditional-expression> ]  |
              NOT <logical-multiplier>
41.   <comparison-operator> -->  <   |
              <=  |
              =   |
              <>  |
              >=  |
              >
42.   <expression> --> <summand> <summands-list>   |
              - <summand> <summands-list>
```

```
43.   <summands-list> --> <add-instruction> <summand>
               <summands-list> |
               <empty>
44.   <add-instruction> -->     +   |
               -   |
               !
45.   <summand> --> <multiplier><multipliers-list>
46.   <multipliers-list> --> <multiplication-instruction>
               <multiplier><multipliers-list>  |
               <empty>
47.   <multiplication-instruction> -->     *   |
               /   |
               &   |
               MOD
48.   <multiplier> --> <unsigned-constant>  |
               <complex-constant> |
               <constant-identifier>  |
               <variable>  |
               <function-identifier>  |
               <builtin-function-identifier><actual-
               arguments>  |
               ( <expression> )  |
               - <multiplier>    |
               ^ <multiplier>
49.   <variable> --> <variable-identifier><dimension> |
               <complex-variable>
50.   <complex-variable> --> "<complex-number>"
51.   <dimension> --> [ <expression> <expressions-list> ]
               |
               <empty>
52.   <expressions-list> -->  ,<expression><expressions-
               list>  |
               <empty>
```

Лексична частина граматики

```
53.   <complex-constant> --> '<complex-number>'
54.   <unsigned-constant> --> <unsigned-number> |
               <unsigned-integer>
55.   <complex-number> --> <left-part> <right-part>
56.   <left-part> --> <expression> |
               <empty>
57.   <right-part> --> ,<expression> |
               $EXP( <expression> )   |
               <empty>
58.   <constant-identifier> --> <identifier>
59.   <variable-identifier> --> <identifier>
```

```
60.   <procedure-identifier> --> <identifier>
61.   <function-identifier>  --> <identifier>
62.   <builtin-function-identifier> --> <identifier>
63.   <assembly-insert-file-identifier> --> <identifier>
64.   <identifier> --> <letter><string>
65.   <string> --> <letter><string> |
               <digit><string> |
               <empty>
66.   <unsigned-number> --> <integer-part><fractional-
               part>
67.   <integer-part> --> <unsigned-integer> |
               <empty>
68.   <fractional-part> --> #<sign><unsigned-integer>
69.   <unsigned-integer> --> <digit><digits-string>
70.   <digits-string> --> <digit><digits-string> |
               <empty>
71.   <sign> --> +      |
               -      |
               <empty>
72.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
73.   <letter> --> A | B | C | D | ... | Z
```

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

Основна література

1. Основи проектування трансляторів: Конспект лекцій : [Електронний ресурс] : навч. посіб. для студ. спеціальності 123 – «Комп'ютерна інженерія» / О. І. Марченко ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 2,64 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 108 с.

2. Основи проектування трансляторів: Інструкції та завдання до виконання лабораторних робіт та розрахунково-графічної роботи з дисципліни «Основи проектування трансляторів» : [Електронний ресурс] : навч. посіб. для студ. спеціальності 123 – «Комп'ютерна інженерія» / О. І. Марченко, О. О. Марченко ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 1,08 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 118 с.

3. Транслятори: лексичний та синтаксичний аналізатори [Текст] : навч. посіб. / В. М. Медведєва, В. А. Третяк ; Нац. техн. ун-т України "Київ. політехн. ін-т". - К. : НТУУ "КПІ", 2012. - 148 с. : рис., табл. - Бібліогр.: с. 146-148.

4. Проектування трансляторів [Текст] : навч. посібник для студентів спеціальності "Програмне забезпечення автоматизованих систем" / В. Х. Власюк ; Вінницький держ. технічний ун-т. - Вінниця : ВДТУ, 2002. - 108 с.

Додаткова література

1. Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. Compilers: Principles, Techniques, & Tools. Pearson Education, Inc., 2007. – 1009 p.

2. David Gries. Compiler Construction for Digital Computers. John Wiley & Sons, Inc., 1971. – 542 p.

3. Donald Knuth. Top-Down Syntax Analysis, Acta Informatica, volume 1, 1971. – pp. 79-110.

4. Leland L. Beck. System Software. An Introduction to systems programming. Addison-Wesley Publishing Company, Inc., 1985. – 437 p.

5. P.M. Lewis, D.J. Rosenkrantz, R.E. Stearns, Compiler Design Theory. Addison-Wesley Publishing Company, Inc., 1976. – 645 p.

6. W.M. McKeeman, J.J. Horning, D.B. Wortman. A Compiler Generator. Prentice-Hall, Inc., 1979. – 513 p.

7. V.J. Rayward-Smith. A first Course in Formal Language Theory. Blackwell Scientific Publications, 1983. – 124 p.

8. Alfred V. Aho, Jeffrey D. Ullman. The Theory of Parsing, Translation and Compiling. Volume 1. Parsing. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1972. – 582 p.

9. Alfred V. Aho, Jeffrey D. Ullman. The Theory of Parsing, Translation and Compiling. Volume 2. Compiling. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1973. – 471 p.

Інформаційні ресурси

13. Youtube канал «Олександр Іванович Марченко» https://www.youtube.com/channel/UCerCwtA87I0yuq3tTQCk7vw

14. Електронний кампус НТУУ «КПІ». Матеріали з дисципліни «Основи проектування трансляторів». – Режим доступу : http://login.kpi.ua