МІНІСТЕРСТВО ОСВІТИ І НАУКИ,
МОЛОДІ ТА СПОРТУ УКРАЇНИ
Національний технічний університет України
“Київський політехнічний інститут”

ОСНОВИ ПРОЕКТУВАННЯ ТРАНСЛЯТОРІВ

Методичні вказівки та завдання до виконання
лабораторних робіт по кредитному модулю
"Інженерія програмного забезпечення – 1.
Основи проектування трансляторів”
для студентів напрямку 6.050102
«Комп’ютерна інженерія»

Рекомендовано вченою радою факультету прикладної математики
НТУУ «КПІ»

Київ 2018

МІНІСТЕРСТВО ОСВІТИ І НАУКИ,
МОЛОДІ ТА СПОРТУ УКРАЇНИ
Національний технічний університет України
“Київський політехнічний інститут”

ОСНОВИ ПРОЕКТУВАННЯ ТРАНСЛЯТОРІВ

Методичні вказівки та завдання до виконання
лабораторних робіт по кредитному модулю
" Інженерія програмного забезпечення – 1.
Основи проектування трансляторів”
для студентів напрямку 6.050102
«Комп’ютерна інженерія»

Затверджено
на засіданні кафедри системного
програмування і спеціалізованих
комп’ютерних систем
ФПМ НТУУ «КПІ»
Протокол №__ від _____

Київ НТУУ «КПІ» 2018

УДК 681.3

Основи проектування трансляторів: методичні вказівки та завдання до виконання лабораторних та розрахунково-графічної робіт по кредитному модулю «Інженерія програмного забезпечення – 1. Основи проектування трансляторів» для студентів напрямку підготовки 6.050102 «Комп'ютерна інженерія» [Електронне видання] / О.І.Марченко. – К.: НТУУ «КПІ», 2018. – 112 с.

Гриф надано вченою радою ФІПМ
(протокол № ___ від _____ 2017 р.)

Методичні вказівки містять теоретичні відомості та завдання до виконання лабораторних робіт і розрахунково-графічної роботи по кредитному модулю «Інженерія програмного забезпечення – 1. Основи проектування трансляторів».

Наведені варіанти індивідуальних завдань. До кожної роботи надаються: теоретичний матеріал, вказівки щодо виконання завдань, тестування програм та оформлення звіту, контрольні питання для підготовки до захисту лабораторних та розрахунково-графічної робіт. Призначені для студентів напряму підготовки 6.050102 «Комп'ютерна інженерія».

Навчальне електронне видання

Автор: *Марченко Олександр Іванович, канд. техн. наук, доцент*

Відповідальний редактор: *Орлова М.М., канд. техн. наук, доцент*
Рецензент: Заболотня *Т.М., канд. техн. наук, доцент*

ЗМІСТ

ВСТУП

Дисципліна "Інженерія програмного забезпечення" є спеціальною нормативною дисципліною у загальній схемі алгоритмічної і програмістської підготовки бакалаврів за напрямом 050102 „Комп‘ютерна інженерія” і складається з двох кредитних модулів. Вивченню першого кредитного модуля "Інженерія програмного забезпечення 1. Основи побудови трансляторів" повинні передувати дисципліни "Структури даних та алгоритми", "Програмування" та “Системне програмування". Цей модуль забезпечує вивчення другого кредитного модуля цієї дисципліни, а також дисциплін „Архітектура комп'ютерів” та „Автоматизація проектування комп'ютерних систем” і призначений для вивчення основ формальних граматик, а також принципів та концепцій, які покладені в основу проектування трансляторів, зокрема їх складових частин: лексичних аналізаторів, синтаксичних аналізаторів та генераторів коду.

Цикл семестрових завдань складається з двох комплексних лабораторних робіт розробницького характеру та розрахунково-графічної роботи і призначений для покриття практичної частини першого кредитного модуля дисципліни «Інженерія програмного забезпечення».

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

Методичні вказівки включають для кожної роботи:

- 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.

1. ЛАБОРАТОРНА РОБОТА «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**>*)**.

3. Для програмування може бути використана довільна алгоритмічна мова програмування високого рівня. Якщо обрана мова програмування має конструкції або бібліотеки для роботи з ***регулярними виразами***, то використання цих конструкцій та/або бібліотек ***строго заборонено***.

4. 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 – . . .

5. Входом ЛА має бути наступне:

- вхідна програма, написана підмножиною мови SIGNAL відповідно до варіанту;
- таблиця кодів ASCII з 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.

6. 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.

Зміст звіту

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

- титульний аркуш;
- індивідуальне завдання згідно до варіанту;
- таблиці або граф автомату, що задає алгоритм ЛА;
- лістинг програми ЛА на мові програмування;
- 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 на довільному носії інформації 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)

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 починається, 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).

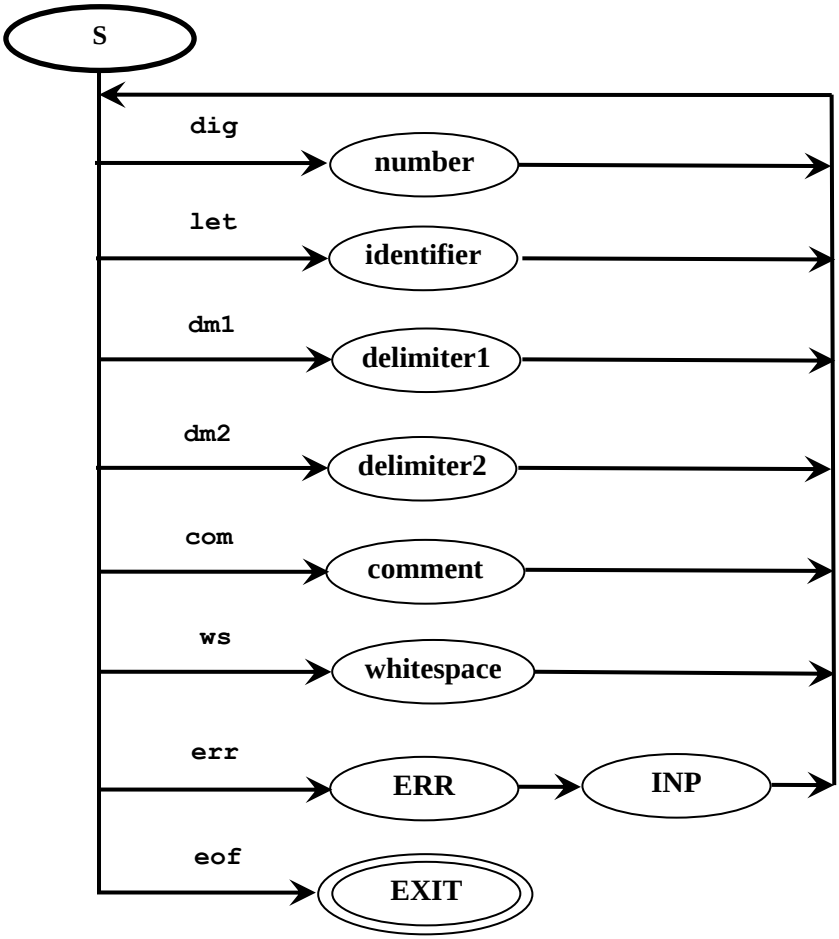

Рис.2 Укрупнений граф автомата ЛА

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.

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**.

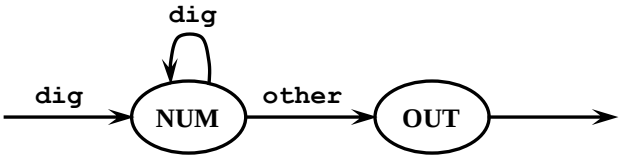

Рис.3 Підграф виділення лексеми (токена) **number**

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

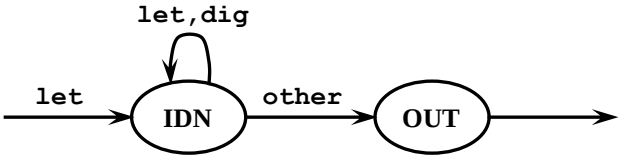

Рис.4 Підграф виділення лексеми (токену) **identifier**

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 виділене ключлве слово, інакше – звичайний ідентифікатор.

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

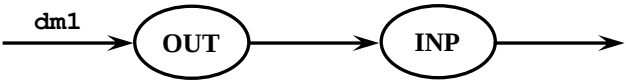

Рис.5 Підграф виділення лексеми (токена) **delimiter1**

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

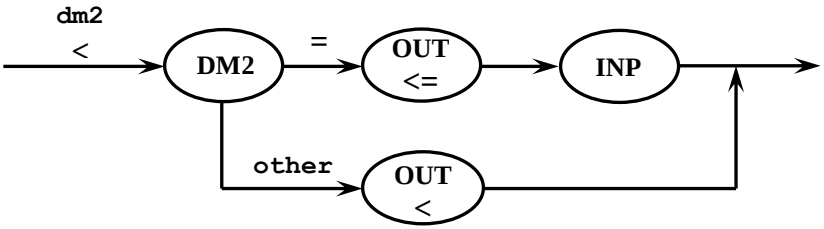

Рис.6 Підграф виділення лексеми (токена) **delimiter2**

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

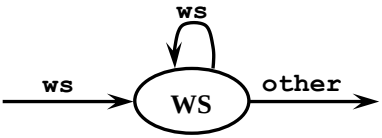

Рис.7 Підграф виділення та обробки **whitespace**

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>*)**.

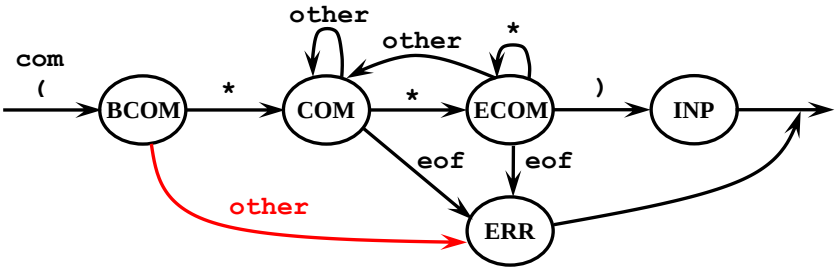

Рис.8 Підграф виділення та обробки коментаря

Важливе зауваження: ***в даному підграфі символ "other" у стані BCOM призводить до помилки (червоне ребро). Однак це вірно лише для граматики даного прикладу, у якій символ "(" використовується тільки як символ початку коментаря і не є односимвольним роздільником. 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.

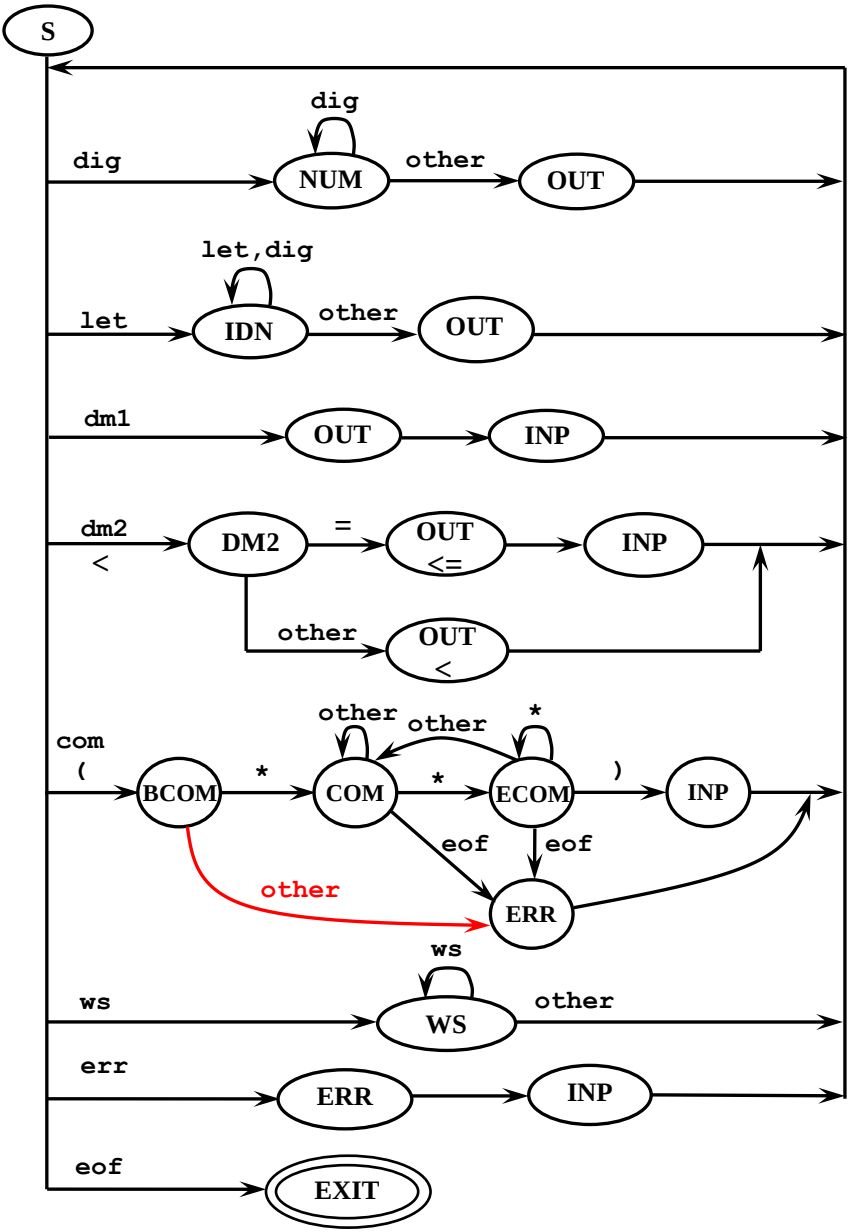

Рис.9 Граф автомата ЛА для граматики умовного оператора

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).

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.

2. РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА «РОЗРОБКА СИНТАКСИЧНОГО АНАЛІЗАТОРА»

Мета розрахунково-графічної роботи

Метою розрахунково-графічної роботи «Розробка синтаксисного аналізатора» є засвоєння теоретичного матеріалу та набуття практичного досвіду і практичних навичок розробки синтаксичних аналізаторів (парсерів).

Постановка задачі

1. Розробити програму синтаксичного аналізатора (СА) для підмножини мови програмування SIGNAL згідно граматики за варіантом.

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 (алгоритм синтаксичного аналізатора вибирається за варіантом);
- 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.

3. Для програмування може бути використана довільна алгоритмічна мова програмування високого рівня. Якщо обрана мова програмування має конструкції або бібліотеки для роботи з ***регулярними виразами***, то використання цих конструкцій та/або бібліотек ***строго заборонено***.

4. 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).

5. 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.

Зміст звіту

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

- титульний аркуш розрахунково-графічної роботи;
- індивідуальне завдання згідно до варіанту;

- перетворену граматику (якщо це необхідно для реалізації заданого алгоритму СА);
- таблиці або граф автомату синтаксичного аналізатора (крім варіантів з синтаксичним аналізом за методом рекурсивного спуску);
- лістинг програми синтаксичного аналізатора на мові програмування;
- 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;
- ***в якості графічної частини розрахунково-графічної роботи*** включити до звіту рисуки дерев розбору, побудованих для всіх контрольних прикладів (мінімум 5 прикладів і дерев).

До звіту на довільному носії інформації 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. Зокрема були визначені чотири відносно великих класи КВ-граматик, які дозволяють побудувати СА без повернень:

1. **LL(k)** – граматики;
2. **LR(k)** – граматики;
3. **RL(k)** – граматики;
4. **RR(k)** – граматики.

Перша літера (наприклад **L**) в позначенні означає напрямок перегляду вхідного рядку (тобто в даному випадку ввід рядку зліва направо). Друга літера (**L** або **R**) в позначенні означає вид розбору (лівий чи правий відповідно), тобто **L –** розбір зліва направо, а **R –** розбір справа наліво. Літера **k** в дужках означає кількість символів вхідного рядку, які аналізуються наперед при виконанні розбору.

З використанням **LL(k)** і **RR(k)**-граматик реалізується низхідна (зверху вниз) стратегія синтаксичного розбору, а з використанням **LR(k)** і **RL(k)**-граматик – висхідна (знизу вверх) стратегія

синтаксичного розбору. На практиці використовуються в основному **LL(k)** та **LR(k)**-граматики.

Існує багато різноманітних алгоритмів синтаксичного розбору. Розглянемо алгоритми, які запропоновано реалізувати в даній розрахунково-графічній роботі.

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.

Розглянемо низхідний розбір за алгоритмом аналізуючої машини Кнута (АМК).

Аналізуюча машина – це абстрактна машина для аналізу рядків в деякому алфавіті. Вона читає із «вхідного рядка» кожен раз по одній літері згідно з деякою програмою. Програма аналізуючої машини являє собою набір процедур, що рекурсивно викликають одна одну; сама програма по суті – це одна із цих процедур. Кожна процедура намагається виявити у вхідному рядку присутність деякої конкретної синтаксичної конструкції і по закінчені роботи повертає значення «істина» чи «фальш» в залежності від того, чи був пошук успішним.

Нехай S_1 S_2 ... S_n – вхідний рядок, і S_h – «поточна» літера, що читається машиною.

Команда цієї машини складається із трьох полів: поля коду операції і двох адрес, AT і AF. Процедури записуються з

використанням двох типів команд, що відповідають двом різним видам кодів операцій.

Тип 1: Код операції – літера *a* із алфавіту.

Тип 2: Код операції – адреса процедури, що стоїть в квадратних дужках [A].

Ці команди виконуються наступним чином:

Тип 1: якщо $S_h = a$, то пропустити *a* (тобто встановити $h := h + 1$) і перейти до AT, інкше перейти до AF.

Тип 2: викликати процедуру, що починається в комірці A (рекурсивно); якщо вона повернула значення true, то перейти до AT, інакше, якщо вона повернула значення false, то перейти до AF.

Кожне з полів AT і AF може містити або адресу команди, або один зі спеціальних символів: T, F або пусте поле. Пусте поле адреси відповідає адресі команди, що записана в наступному рядку. Якщо він містить T, здійснюється вихід із процедури із значенням «істина». Якщо воно містить F, то відбувається вихід із процедури із значенням «фальш» і змінна *h* знову набуває того значення, яке вона мала до входу в процедуру.

Зауваження: Таким чином, мається на увазі, що при виклику процедури по коду операції типу 2, завжди разом з адресою повернення зберігається і значення змінної *h*.

Розглянемо наступну граматику мови, що нагадує мову «логічних виразів». Граматика записується в модифікованих позначеннях БНФ: замість «::=» використовується знак «→», синтаксичні класи позначаються великими літерами, а не беруться в дужки.

1. B → R | (B)
2. R → E = E
3. E → a | b | (E+E)

Позначення класів:

- B – логічний вираз;
- R – відношення;
- E – вираз.

Відповідна програма для аналізуючої машини Кнута наведена на рис. 8. Необхідно звернути увагу на відповідність між граматикою та програмою АМК. Останні два рядки програми відповідають ще одному правилу граматики

Start → B ┤

де «┤» - спеціальний символ-обмежувач, який зустрічається тальки в кінці рядка, що аналізується.

Адреса	Код операцхї	AT	AF
B	[R]	T	
	(		F
	[B]		F
	)	T	F
R	[E]		F
	=		F
	[E]	T	F
E	a	T	
	b	T	
	(		F
	[E]		F
	+		F
	[E]		F
	)	T	F
Start	[B]		ПОМИЛКА
	┤	OK	ПОМИЛКА

Рис.10 Приклад програми АМК

Процедура *Start* виконає перехід на мітку «ОК», лише якщо весь рядок, що аналізується являє собою В (логічний вираз), за яким слідує символ „┤“, інакше вона виконає перехід на мітку «ПОМИЛКА».

Низхідний розбір за алгоритмом рекурсивного спуску

Метод рекурсивного спуску реалізує безповоротний аналіз за рахунок наступних обмежень на правила граматики:

1. Правила не повинні містити лівобічної рекурсії. Якщо такі правила є, то вони замінюються правилами з правосторонньою рекурсією або представляються в ітераційній формі.

2. Якщо є декілька правил з однаковою лівою частиною, то права частина повинна починатися з різних термінальних символів (нормальна форма Грейбах).

Суть методу: кожному нетерміналу граматики ставиться у відповідність процедура або функція.

Контрольні питання

3. Принцип виконання низсхідного аналізу (розбору).
4. Принцип виконання висхідного аналізу (розбору).
5. Визначення автомату з магазинною (стековою) пам’яттю (МП-автомату).
6. Конфігурація і такт роботи МП-автомата.
7. Детерміновані та не детерміновані МП-автомати.
8. Відповідність між КВ-граматикою та МП-автоматом.

9. Визначення LL(k), LR(k), RL(k) і RR(k) граматик.

10.Основні алгоритми низсхідної стратегії синтаксичного розбору та переваги використання LL(1)-граматик.

11.Принцип роботи синтаксичного аналізу за алгоритмом аналізуючої машини Кнута (АМК).

12.Формування таблиці АМК (програмування АМК).

13.Принцип роботи синтаксичного аналізу за алгоритмом рекурсивного спуску.

14. Принцип роботи синтаксичного аналізу за алгоритмом граматик передування.

15.Принцип роботи синтаксичного аналізу за алгоритмом табличнo-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.

Алгоритм синтаксичного аналізу вибирається за таблицею 2.

Числа в таблиці означають номер алгоритму синтаксичного аналізу для даного варіанту згідно з наступним списком:

1 – низхідний розбір за алгоритмом аналізуючої машини Кнута;

2 – низхідний розбір за алгоритмом рекурсивного спуску.

Таблиця 2. Варіанти алгоритму синтаксичного аналізу

Номер варіанту	**Номер алгоритму синтаксичного аналізу**	**Номер варіанту**	**Номер алгоритму синтаксичного аналізу**
1	1	19	1
2	2	20	2
3	1	21	1
4	2	22	2
5	1	23	1
6	2	24	2
7	1	25	1
8	2	26	2
9	1	27	1
10	2	28	2
11	1	29	1
12	2	30	2
13	1	31	1
14	2	32	2
15	1	33	1
16	2	34	2
17	1	35	1
18	2		

Варіант визначається відповідно порядковому номеру студента в списку журнала групи.

3. ЛАБОРАТОРНА РОБОТА «РОЗРОБКА ГЕНЕРАТОРА КОДУ»

Мета лабораторної роботи

Метою лабораторної роботи «Розробка генератора коду» є засвоєння теоретичного матеріалу та набуття практичного досвіду і практичних навичок розробки генераторів коду.

Постановка задачі

1. Разробити програму генератора коду (ГК) для подмножини мови програмування SIGNAL, заданої за варіантом.
2. Програма має забезпечувати:

- читання дерева розбору та таблиць, створених синтаксичним аналізатором, який було розроблено в розрахунково-графічній роботі;
- виявлення семантичних помилок;
- генерацію коду та/або побудову внутрішніх таблиць для генерації коду.

3. Входом генератора коду (ГК) мають бути:

- дерево розбору;
- таблиці ідентифікаторів та констант з повною інформацією, необхідною для генерації коду;
- вхідна програма на підмножині мови програмування SIGNAL згідно з варіантом (необхідна для формування лістингу програми).

4. Виходом ГК мають бути:

- асемблерний код згенерований для вхідної програми та/або внутрішні таблиці для генерації коду;
- внутрішні таблиці генератора коду (якщо потрібні).

5. Зкомпонувати повний компілятор, що складається з розроблених раніше лексичного та синтаксичного аналізаторів і генератора коду, який забезпечує наступне:

- генерацію коду та/або побудову внутрішніх таблиць для генерації коду;
- формування лістингу вхідної програми з повідомленнями про лексичні, синтаксичні та семантичні помилки.

6. Входом компілятора має бути програма на підмножині мови програмування SIGNAL згідно з варіантом;

7. Виходом компілятора мають бути:

- асемблерний код згенерований для вхідної програми та/або внутрішні таблиці для генерації коду;
- лістинг вхідної програми з повідомленнями про лексичні, синтаксичні та семантичні помилки.

8. Для програмування може бути використана довільна алгоритмічна мова програмування високого рівня. Якщо обрана мова програмування має конструкції або бібліотеки для роботи з ***регулярними виразами***, то використання цих конструкцій та/або бібліотек ***строго заборонено***.

Зміст звіту

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

- титульний аркуш;
- індивідуальне завдання згідно із варіантом;
- лістинг програми ГК;
- контрольні приклади, необхідні для демонстрації всіх конструкцій заданої граматики, а також всіх можливих помилкових ситуацій. Опис кожного контрольного прикладу має містити вхідне дерево розбору і таблиці сформовані синтаксичним аналізатором, а також згенерований код та додаткові внутрішні таблиці ГК (якщо необхідно).

До звіту на носії інформації має бути додано наступне:

- файл з текстом звіту;
- проект працюючої програми (компілятора, що включає ЛА, СА та ГК) з вхідними кодами, завантажувальний модуль та необхідними файлами даних;
- набір тестів, що показують коректність роботи компілятора, та відповідні лістинги з результатами компіляції.

Методичні вказівки

Неформальна семантика конструкцій мови програмування SIGNAL.

1. **Визначення головної програми (PROGRAM) та процедур (PROCEDURE) без параметрів.**

Дії:

- генерація заголовку для виклику програми чи процедури;
- генерація інструкцій повернення з підпрограми.

Обмеження:

- не дозволяється використовувати однакові імена для двох і більше процедур;
- не дозволяється використовувати однакові імена для процедури та довільної змінної або константи.

2. **Визначення процедур (PROCEDURE) з параметрами.**

Дії:

- генерація заголовку для виклику програми чи процедури;
- генерація інструкцій повернення з підпрограми;
- виділення кадру стеку для зберігання параметрів.

Обмеження:

- не дозволяється використовувати однакові імена для двох і більше процедур;
- не дозволяється використовувати однакові імена для процедури та довільної змінної або константи;

- має бути відповідність за типом і порядком розташування між формальними параметрами, заданими в списку параметрів, та фактичними параметрами, заданими у операторі виклику.

3. **Визначення математичних функцій (DEFFUNC)**

Семантика:

- визначення математичної функції задає ініціалізований масив, заданий за допомогою формули в заданих межах.

Дії:

- виділення пам’яті для масиву заданого розміру;
- ініціалізація масиву значеннями згідно до заданої формули.

Обмеження:

- не дозволяється використовувати однакові імена для двох і більше функцій;
- імена функцій, описаних в розділі DEFFUNC фактично є іменами масивів, а тому не мають збігатися з іменами інших змінних та констант.

4. **Оголошення змінних**

Дії:

- генерація коду виділення пам’яті для змінних.

Обмеження:

- не дозволяється використовувати однакові імена для двох і більше змінних в одній підпрограмі чи на глобальному рівні.

5. **Оголошення констант**

Дії:

- генерація коду виділення пам’яті для констант;
- ініціалізація виділеної пам’яті заданими значеннями констант.

Обмеження:

- не дозволяється використовувати однакові імена для двох і більше констант в одній підпрограмі чи на глобальному рівні;
- не дозволяється використовувати однакові імена для конятанти та довільної змінної або процедури;
- не дозволяється змінювати значення константи в програмі.

6. **Оголошення міток**

Дії:

- генерація таблиці міток;
- перетворення міток до вигляду ідентифікаторів для можливості вставки в асемблерний код.

Обмеження:

- не дозволяється використовувати однакові імена для двох і більше міток;
- імена згенерованих міток не мають збігатися з іменами змінних та констант в одній підпрограмі та з зовнішніми іменами на глобальному рівні.

7. **Використання змінних**

Дії:

- конструкція виду <змінна>[<вираз>,<список-виразів>] задає звернення до масиву;
- числова константа в одинарних лапках задає комплексну константу, яка повинна обчислюватись на етапі згортки у лексичному аналізаторі;
- конструкція виду “<вираз>,<вираз>” задає комплексну змінну (тобто в виразі можуть бути присутні змінні);
- конструкція виду “<вираз>$EXP(<вираз>)” задає комплексну змінну в експоненційній формі.

Обмеження:

- не дозволяється використання комплексних змінних в лівій частині виразу (ліворуч від знаку присвоєння), заданих у вигляді “<вираз>,<вираз>” або “<вираз>$EXP(<вираз>)”, тобто у лівій частині оператора присвоєння дозволяється використовувати тільки одиничні ідентифікатори комплексних змінних.

8. **Оператори**

Дії:

- правило <statements-list> --> <empty> відповідає інструкції nop;
- конструкція <variable> := <expression> задає оператор присвоєння;
- конструкція <procedure-identifier><actual-arguments> задає оператор виклику процедури;

- конструкція IF-THEN-ELSE-ENDIF задає умовний оператор (існує варіант як з однією, так і з двома гілками умовного оператора);
- конструкція WHILE-DO задає цикл з передумовою;
- конструкція LOOP-ENDLOOP задає нескінчений цикл;
- конструкція FOR-ENDFOR задає цикл з лічильником;
- конструкція CASE-OF-ENDCASE задає оператор вибору;
- конструкція GOTO задає оператор безумовного переходу до заданої мітки;
- конструкція LINK <variable-identifier> , <unsigned-integer> зв'язує задану змінну з одним з регістрів (портів) вводу-виводу;
- конструкції IN <unsigned-integer> и OUT <unsigned-integer> задають властивості регістру, зв'язаного зі змінною оператором LINK. Властивості задаються віртуально і є необхідними на етапі компіляції для перевірки семантичної коректності. Оператор IN задає регістру властивість „тільки читання", а оператор OUT – властивість „тільки запис".
- конструкція RETURN задає оператор повернення з процедури;
- пустому оператору „ ; " відповідає інструкція nop;
- конструкція **($** <assembly-insert-file-identifier> **$)** відповідає вставці тексту асемблерного коду, який знаходиться в заданому файлі, в програму без будь-яких перевірок.

Обмеження:

- поле мітки оператора безумовного переходу має містити тільки імена оголошених та існуючих міток;
- оператори IN та OUT не можуть задавати властивості одного і того ж регістру в одній програмі;
- не дозволяється запис до регістру, якщо для нього була встановлена властивість „тільки читання” за допомогою оператору IN;
- не дозволяється читання з регістру, якщо для нього була встановлена властивість „тільки запис” за допомогою оператору OUT;
- в операторах LINK можуть використовуватись тільки змінні, оголошені зі специфікатором SIGNAL.

9. **Операції в виразах**

Дії:

- результатом логічних операцій (OR, AND, NOT) та операцій порівняння є 0 чи 1;
- операція ! відповідає побітовому OR над операндами типу INTEGER;
- операція & відповідає побітовому AND над операндами типу INTEGER;
- операція ^ відповідає побітовому NOT над операндом типу INTEGER;
- результатом операції MOD є залишок від ділення.

Обмеження:

- операції мають виконуватись тільки над даними сумісних типів.

10. **Типи даних**

Базові типи:

- INTEGER – ціле число;
- FLOAT – число з плаваючою комою. В пам'яті таке число міститься у вигляді <мантиса><порядок>;
- BLOCKFLOAT – використовується тільки для масивів чисел з плаваючою комою, що мають однаковий порядок. В пам'яті міститься у вигляді <порядок><мантиса1>…< мантисаN>.

Специфікатори:

- Довільна змінна чи константа може мати тільки один з базових специфікаторів INTEGER, FLOAT або BLOCKFLOAT;
- специфікатор COMPLEX задає комплексний тип даних для одного з базових типів;
- специфікатор SIGNAL вказує на те, що змінна є вхідною чи вихідною. Така змінна обов'язково має бути пов'язана з регістром (портом) вводу-виводу за допомогою оператору LINK, після чого дії вводу-виводу виконуються операторами IN або OUT;
- один регістр (порт) вводу-виводу в рамках однієї програми не може бути одночасно і портом вводу, і портом виводу;
- специфікатор SIGNAL можна використовувати разом з одним з числових типів;
- всі змінні з однаковим ідентифікатором, оголошені в різних процедурах зі специфікатором EXT, пов'язані з однією коміркою

пам’яті. Специфікатор EXT можна використовувати з будь-яким з базових або комплексним типом, а також з масивом.

Масиви:

- для оголошення масиву необхідно описати атрибут [<список діапазонів>], який задає розмірності масиву (кількість розмірностей та їхні індексні діапазони).

Контрольні питання

1. Поняття неформальної та формальної семантики мови програмування.
2. Визначення мета семантичної мови та приклади існуючих мета семантичних мов.
3. Види семантичної відповідності у простій мета семантичній мові.
4. Структура програми генератора коду (семантичного процесора) та структура рекурсивних семантичних процедур/функцій.
5. Генерація коду для оператора присвоєння.
6. Генерація коду для неповної умовної конструкції if-then.
7. Генерація коду для повної умовної конструкції if-then-else.
8. Генерація коду для конструкції циклу з передумовою while.
9. Генерація коду для конструкції циклу з післяумовою repeat.
10. Генерація коду для конструкції циклу з лічильником (параметром) for.
11. Генерація коду для керуючої конструкції вибору case.

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-declaration>
6.    <math-function-declaration> --> 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.   <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 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>. |
                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-declaration> --> 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> --> 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> --> <expression> |
             <empty>
30.  <right-part> --> ,<expression> |
             $EXP( <expression> )   |
             <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 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-declaration> --> DEFFUNC
                <function-list> |
                <empty>
12.   <function-list> --> <function><function-list> |
                <empty>
13.   <function> --> <function-identifier> =
                <expression><function-characteristic> ;
14.   <function-characteristic> --> \ <unsigned-
                integer> , <unsigned-integer>
15.   <procedure-declarations> --> <procedure>
                <procedure-declarations> |
                <empty>
16.   <procedure> --> PROCEDURE <procedure-
                identifier><parameters-list> ;
17.   <parameters-list> --> ( <attributes-list> ) |
                <empty>
18.   <attributes-list> --> <attribute> ,
                <attributes-list> |
                <empty>
19.   <attribute> --> INTEGER    |
                FLOAT      |
                BLOCKFLOAT
20.   <statements-list> --> <statement> <statements-
                list> |
                <empty>
21.   <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> $)
22.   <actual-arguments> --> ( <variable-identifier>
                <identifiers-list> )  |
                <empty>
23.   <variable-identifier> --> <identifier>
24.   <procedure-identifier> --> <identifier>
25.   <function-identifier>  --> <identifier>
26.   <assembly-insert-file-identifier> -->
                <identifier>
27.   <identifier> --> <letter><string>
28.   <string> --> <letter><string> |
                <digit><string> |
                <empty>
29.   <unsigned-integer> --> <digit><digits-string>
30.   <digits-string> --> <digit><digits-string> |
                <empty>
31.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
                9
32.   <letter> --> A | B | C | D | ... | Z
```

Варіант 29

```
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 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-declaration> --> 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-declaration> --> 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-declaration> --> 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]

Copyright (c) Oleksandr Marchenko, 1987-1988

```
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-declaration> --> 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. Ахо А., Сети Р., Ульман Д. Компиляторы: принципы, технологии и инструменты.: Пер. с англ. – М.: Издательский дом «Вильямс», 2003, – 768 с. : ил.

2. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. В 2-х томах. – М.: Мир, 1978. – 1105 с.

3. Грисс Д. Конструирование компиляторов для цифровых вычислительных машин. – М.: Мир, 1975. – 544 с.

4. Д.Э.Кнут. Нисходящий синтаксический анализ. В кн.: Кибернетический сборник. Выпуск 15. – М.: Мир, 1978. – с.101-142.

5. Бек Л. Введение в системное программирование. – М.: Мир, 1988. – 448 с.

6. Опалева Э.А., Самойленко В.П. Языки программирования и методы трансляции. – СПб: БХВ-Петербург, 2005. – 476 с.

7. Марченко А.И. Средства автоматизации программирования процессоров цифровой обработки сигналов Диссертация на соискание ученой степени кандидата технических наук. Кафедра вычислительной техники, Киев, КПИ, 1993, 129 с.

Додаткова література

8. Компаниец Р.И., Маньков Е.В., Филатов Н.Е. Системное программирование. Основы построения трансляторов./Учебное

пособие для высших и средних учебных заведений – СПб.: КОРОНА принт, 2000. – 256 с.

9. Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. – М.: Мир, 1979. – 656 с.

10.Касьянов В.Н., Поттосин И.В. Методы построения трансляторов. – Новосибирск: Наука, 1986. – 344 с.

11.Глушков В.М., Цейтлин Г.М., Ющенко Е.Л. Алгебра, языки, программирование. – К.: Наукова думка, 1978. – 320 с.

12. Янг С. Алгоритмические языки реального времени. Конструирование и разработка. – М.: Мир, 1985. – 400 с.

13. Ингерман П. Синтаксически ориентированный транслятор. – М.: Мир, 1969. – 176 с.

14. Маккиман У., Хорнинг Дж., Уортман Д. Генератор компиляторов. – М.: Статистика, 1980. – 528 с.

15. Языки и автоматы. Сборник переводов статей. – М.: Мир, 1975. – 360с.

16. Кибернетический сборник. Выпуск 15. – М.: Мир, 1978. – 224 с.