МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Державний вищий навчальний заклад
«ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

МЕТОДИЧНІ ВКАЗІВКИ І ЗАВДАННЯ
ДО ЛАБОРАТОРНИХ РОБІТ ЗА КУРСОМ

"ТЕОРІЯ СИНТАКСИЧНОГО АНАЛІЗУ І КОМПІЛЯЦІЇ"

Донецьк – 2009

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Державний вищий навчальний заклад
«ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

МЕТОДИЧНІ ВКАЗІВКИ І ЗАВДАННЯ
ДО ЛАБОРАТОРНИХ РОБІТ ЗА КУРСОМ

"ТЕОРІЯ СИНТАКСИЧНОГО АНАЛІЗУ І КОМПІЛЯЦІЇ"

(для студентів напряму підготовки 6.0501
«Програмна інженерія»)

Затверджено
на засіданні кафедри
прикладної математики і
інформатики
Протокол № 9 від 4.02.2009

Донецьк - 2009

УДК 681.3.06

Методичні вказівки і завдання до лабораторних робіт за курсом "ТЕОРІЯ СИНТАКСИЧНОГО АНАЛІЗУ І КОМПІЛЯЦІЇ" (для студентів напряму підготовки 6.0501 “Програмна інженерія”)./ Укл. Дмитрієва О.А. – Донецьк: ДонНТУ, 2009 – 72 с.

Методичні вказівки і завдання призначені для закріплення знань студентів професійного напряму підготовки 6.0501 «Програмна інженерія» денної форми навчання факультету обчислювальної техніки та інформатики, одержуваних при вивченні лекційного матеріалу. Методичні вказівки містять основні вказівки і завдання до 6 лабораторних робіт по всіх основних розділах курсу, до яких відносяться:

- організація таблиць у трансляторах і робота з ними;
- конструювання сканерів;
- побудова породжуючих граматик для конструкцій мов програмування;
- граматики-розпізнавачи;
- проектування синтаксичного аналізатора для граматики передування;
- проектування розпізнавача для граматики з операторним передуванням.

При виконанні лабораторних робіт необхідно застосовувати викладені в методичних вказівках теоретичні положення для вирішення поставлених задач. Студентам пропонується виконати ряд індивідуальних завдань, варіанти яких приводяться в методичних вказівках. Таким чином, самостійна робота студентів дозволить розвити навички рішення практичних задач.

Для кожної теми приводяться змістовні задачі, що дозволяють проробити на практиці вивчений матеріал. Для виконання задач передбачається розробка індивідуальних програм.

Рецензент, к.т.н., доц. каф. ЕОМ Анопрієнко О.Я.

Лабораторна робота №1

ОРГАНІЗАЦІЯ ТАБЛИЦЬ У ТРАНСЛЯТОРАХ І РОБОТА З НИМИ

Ціль роботи: Одержати практичні навички проектування і реалізації таблиць у трансляторах і операцій з таблицями.

Для виконання роботи необхідно знати способи організації таблиць і методи пошуку інформації в таблицях.

При виконанні лабораторної роботи побудувати і реалізувати конкретну таблицю для транслятора з однієї з пропонованих мов програмування: С, ПАСКАЛЬ, АДА, МОДУЛА, ФОРТРАН, PL/1 та інш.

Необхідні теоретичні знання для виконання роботи викладені в [1, с. 244-277; 2, с. 35-53; 4, с. 129-142; 5, т.2, с. 267-326; 6, с. 227-231; 9, с. 96-115]. У монографії [7] описується семантика понять мов програмування, а також міститься огляд реалізацій мов програмування.

1.1 Теоретичні знання

Таблиця - це множина записів, кожна з яких є набором пойменованих полів. Одне поле запису є ключем, за значенням якого здійснюється пошук запису в таблиці і додавання нового запису в таблицю.

Таблиці є найбільш уживаною структурою даних у трансляторах. При трансляції описової частини програми заповнюються відповідні таблиці транслятора. Трансляція виконавчої частини програми полягає в генерації команд вихідної мови. При генерації здійснюються необхідні семантичні перевірки на основі інформації, що міститься в таблицях. Значна частина часу трансляції затрачається на пошук у таблицях. Тому ефективної організації пошуку інформації в таблицях приділяється особлива увага. Крім того, кожен запис таблиці повинен займати якнайменше пам'яті для того, щоб вистачило пам'яті при трансляції великих програм, і трансляція була швидкою.

Основною таблицею будь-якого транслятора є "ТІ" - *таблиця ідентифікаторів* (імен, символів), у якій зберігаються всі атрибути (характеристики) ідентифікаторів, необхідні для перевірки семантики програми і для генерації машинного коду, ключем для "ТІ" є сам ідентифікатор, а його атрибутами можуть бути тип значення, адреса в пам'яті, ознака ініціалізації значення змінної, що позначається ідентифікатором, і т.д. Атрибути ідентифікатора з'ясовуються з опису і контексту використання ідентифікатора в програмі.

У мовах, що мають кінцеве число типів (АЛГОЛ-60, ФОРТРАН, PL/1), тип можна представити цілим числом. У мовах, що мають потенційно нескінченне число типів (АЛГОЛ-68, ПАСКАЛЬ, АДА, С), інформація про тип представляється покажчиком на запис у спеціальній таблиці типів. У *таблиці типів* зберігають ім'я типу, обсяг пам'яті, займаної одним значенням даного типу, структурні характеристики: для масивів - кількість вимірів і границі зміни індексів по кожному виміру; для структур (записів) - кількість елементів (полів), для кожного елемента (поля) необхідно пам'ятати ім'я і тип значення.

У залежності від реалізованої мови програмування і методів трансляції в трансляторі можуть також використовуватися таблиці позначок, процедур, операцій і т.д.

У *таблиці позначок* зберігають саму позначку, тип визначенності, вид позначки (позначка оператора, позначка процедури і т.п.), адресу команди, з якою зв'язується позначка.

У *таблиці процедур* міститься наступна інформація: ім'я процедури, вид процедури (процедура чи функція), кількість параметрів, їхні імена і тип, адреса першої команди тіла процедури, для функцій - тип результату.

У *таблиці операцій* необхідно мати позначення операції, її пріоритет і асоціативність, зв'язок із процедурою генерації машинних команд для даної операції. Інформація про припустимі типи значення операндів операції може зберігатися в таблиці чи знаходитися у відповідній процедурі генерації

машинних команд.

Усі таблиці транслятора можна розділити на статичні і динамічні.

У *статичних таблицях* незалежно від трансльованої програми є постійна кількість незмінних записів. Прикладами статичних таблиць можуть бути таблиця операцій, визначених у мові, таблиця вбудованих у мову примітивних функцій. Кількість і зміст таких таблиць визначається мовою програмування, для якої конструюється транслятор.

У будь-якому трансляторі є *динамічні таблиці*, для яких кількість і/чи вміст записів таблиці змінюється в ході трансляції. Приклади динамічних таблиць "ТІ", таблиця типів, таблиця констант і т.п. У динамічних таблицях може бути статична частина. Наприклад, у "ТІ" можуть знаходитися ключові слова мови, у таблиці типів - вбудовані в мову типи, у таблиці процедур - вбудовані в мову примітивні функції, а в таблиці констант - константи, обумовлені мовою програмування і її реалізацією. Статичні частини таблиць повинні бути заповнені (проініціалізовані) до початку трансляції програм. Для динамічних таблиць можливе оформлення статичної частини у вигляді окремої таблиці з більш ефективною реалізацією операції пошуку інформації.

Сумарний обсяг пам'яті, необхідної для статичних таблиць, не залежить від величини і вмісту трансльованої програми, тому основну увагу варто звернути на пам'ять для динамічних таблиць транслятора. Вимога мінімального обсягу пам'яті для одного запису таблиці виконується, якщо всі поля запису мінімального розміру. Однак деякі поля можуть представляти інформацію, яку не можна безпосередньо закодувати, використовуючи поле фіксованого розміру. Наприклад, довжина ідентифікатора в багатьох мовах програмування не обмежується. У цьому випадку розмір поля беруть по максимуму, чи в поле поміщають покажчик на додаткову структуру даних (див. рис. 1.1).

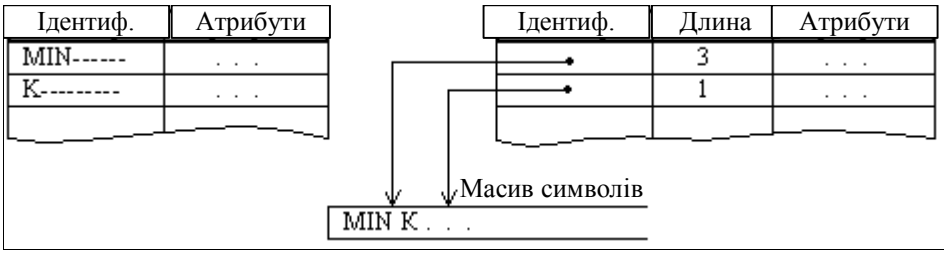


Рисунок 1.1 – Приклади структури "Таблиця ідентифікаторів" для мови АДА.

У "ТІ" кількість інформації, яку потрібно зберігати для ідентифікатора, залежить від об'єкта, що зв'язується з ідентифікатором – простою змінною, масивом, процедурою і т.п. Таким чином, кількість необхідних полів одного запису таблиці може залежати від значень атрибутів запису, у даному випадку можливо кілька способів структурної організації одного запису таблиці транслятора.

1) Альтернативні поля виносять у додаткові структури даних, а у вихідній таблиці зберігають покажчик. Для розглянутого прикладу вигляд об'єкту буде визначати додаткову структуру даних (в окремому випадку таблицю), на яку посилається покажчик.

2) Альтернативні поля "перекриваються" по пам'яті. Для розглянутого прикладу кожен запис буде мати поля, що містять чи атрибути масиву чи атрибути процедури. Зміст "перекритих" полів визначається видом об'єкта, що зв'язується з ідентифікатором.

3) Запис таблиці містить змінну кількість полів, у залежності від об'єкту. При цьому структура і довжина записів в одній таблиці будуть змінними, тому кожний запис повинний містити інформацію про свою структуру і довжину. Наприклад, для транслятора з мови PL/1 можливі зв'язки таблиць-ідентифікаторів (ТІ), позначок (ТМ) і процедур (ТП) - показані на рис. 1.2. Для оператора опису процедури-функції, що обчислює максимальне значення функції F, на інтервалі (A, B).

MAXF: PROCEDURE (F, A, B) RETURNS (REAL (B));

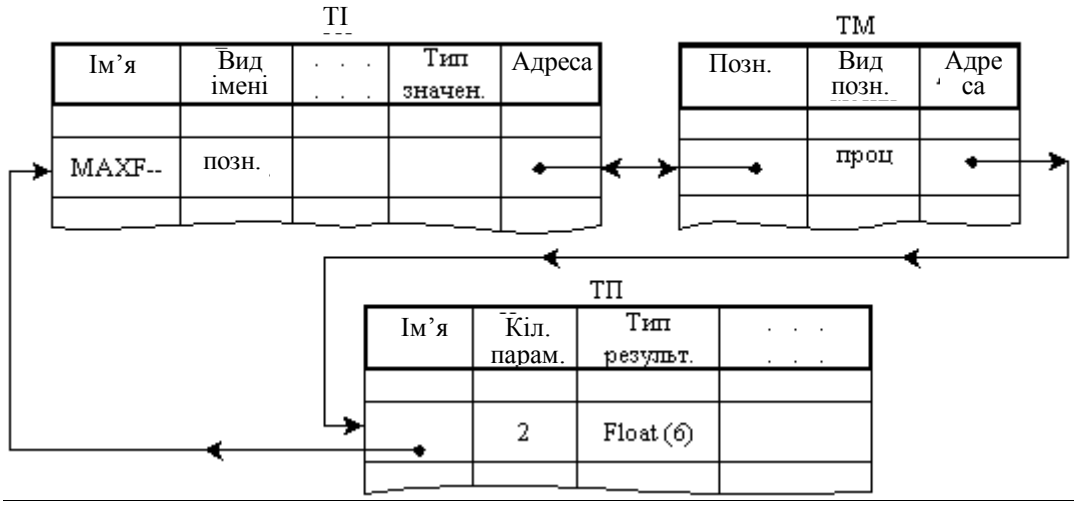


Рисунок 1.2 – Зв'язки таблиць у трансляторі з мови PL/1.

Вибір структурної організації запису таблиці і способів кодування інформації в ній, зв'язків з іншими таблицями в кожному конкретному випадку здійснюється компромісом між суперечливими вимогами мінімальності пам'яті і швидкого виконання операцій.

Для таблиць у трансляторах використовують наступні операції:

1) пошук запису;

2) додавання запису;

3-4) читання/занесення значення атрибута заданого запису;

5) виключення запису;

6) реорганізація таблиці.

Під час трансляції для статичних таблиць визначені операції 1 і 3, а для динамічних таблиць можуть застосовуватися всі перераховані операції.

Пошук запису в таблиці найбільш вживана операція, вона реалізується по-різному для статичних і динамічних таблиць, для статичних таблиць бажаний прямий метод пошуку: за значенням ключа відразу визначається місце відповідного запису в таблиці. Послідовний пошук є припустимим для статичних таблиць з невеликою кількістю записів (наприклад, пошук у таблиці операцій). Для прискорення пошуку в динамічних таблицях використовується хеш-адресація: адреса запису в таблиці виходить

“хешуванням” ключа - виконанням простих арифметичних чи логічних операцій над ключем. За методом організацій пошуку виділяють кілька методів організації таблиць: неупорядкована (проста чи деревоподібна), упорядкована, із входами, що обчислюються, і т.д.

Як правило, додаванню запису в таблицю передує пошук. Якщо пошук не знайшов відповідного запису в таблиці, то, можливо, цей запис необхідно додати в таблицю.

При семантичних перевірках потрібно знати атрибути записів. Для цього використовується операція "читання значення" атрибута заданого запису.

Наприклад, для конструкцій "виклик процедури" необхідно перевірити відповідність фактичних і формальних параметрів по кількості і типу значень. Рекомендується реалізацію операція "читання значення" атрибута запису виконувати у виді макровизначення чи процедури-функції.

У ряді мов програмування для деяких об'єктів (наприклад, позначок) входження конструкції, що використовує об'єкт (оператор GOTO), може в тексті програми передувати визначальному входженню самого об'єкту (оператора з позначкою). У цьому й у ряді інших випадків довизначені значення атрибутів заносяться в таблицю з допомогою операції занесення значення атрибута.

Для мов програмування, що мають блокову структуру програми (С, АДА, ПАСКАЛЬ та інш.), допускається локалізація об'єктів усередині блоку (позначок, змінних, процедур і т.п.), при завершенні трансляції блоку інформація про його локальні об'єкти повинна бути вилучена з таблиць операцією виключення записів.

Реорганізація таблиць може використовуватися, наприклад, для упорядкування записів у таблиці з метою прискорення пошуку чи для ущільнення записів у таблиці.

1.2 Варіанти завдань

1.2.1 Таблиці для транслятора з мови С

1) Таблиця ідентифікаторів і її ініціалізація.

2) Таблиця типів і її ініціалізація.

3) Таблиця констант і її ініціалізація.

4) Таблиця операцій і її ініціалізація.

5) Таблиця позначок.

6) Таблиця процедур і її ініціалізація.

1.2.2 Таблиці для транслятора з мови ПАСКАЛЬ

7) Таблиця ідентифікаторів і її ініціалізація.

8) Таблиця типів і її ініціалізація.

9) Таблиця констант і її ініціалізація.

10) Таблиця операцій і її ініціалізація.

11) Таблиця позначок.

12) Таблиця процедур і її ініціалізація.

1.2.3 Таблиці для транслятора з мови АДА

13) Таблиця ідентифікаторів і її ініціалізація.

14) Таблиця типів і її ініціалізація.

15) Таблиця констант і її ініціалізація.

16) Таблиця операцій і її ініціалізація.

17) Таблиця позначок.

18) Таблиця процедур і її ініціалізація

1.2.4 Таблиці для транслятора з мови ФОРТРАН

19) Таблиця ідентифікаторів і її ініціалізація.

20) Таблиця підпрограм і підпрограм-функцій. Ініціалізація таблиці примітивними функціями.

21) Таблиця масивів.

22) Таблиця позначок.

23) Таблиця операцій і її ініціалізація.

24) Таблиця операторів-функцій.

25) Таблиця констант.

1.3 Порядок виконання роботи

1. Відповісти на контрольні питання і виконати вправи даної лабораторної роботи.

2. Для обраної мови програмування усвідомити семантику понять, для яких конструюється таблиця.

3. Визначити структуру записів таблиці і її зв'язків з іншими таблицями, а також операції над записами таблиці.

4. Вибрати й обґрунтувати метод організації таблиці з урахуванням її зв'язків.

5. Реалізувати таблицю й операції над нею за допомогою інструментальної мови програмування. Реалізація роботи з таблицею повинна бути виконана у виді декількох процедур, що відповідають операціям над записами таблиці: пошук запису, додавання нового запису, читання/занесення значення деякого атрибута запису, виключення запису, реорганізація таблиці.

1.3.1 Загальні зауваження

1. Результати даної лабораторної роботи використовуються для виконання наступних робіт. У зв'язку з цим можлива корекція структури таблиці і програм, що реалізують операції над записами таблиці. Тому всі програми повинні розроблятися з урахуванням можливих наступних модифікацій.

2. Для ініціалізації таблиці ідентифікаторів вибрати найбільш уживані ключові слова (15-25) мови, серед яких обов'язково повинні бути ключові слова, пов'язані з описами (див. варіанти завдань лабораторної роботи 2).

3. Для ініціалізації таблиці процедур (підпрограм) взяти 5-8 функцій.

4. Таблиця операцій повинна містити інформацію про всі операції мови програмування.

5. Таблицю типів необхідно проініціалізувати всіма вбудованими в мову типами.

1.4 Зміст звіту

1) Назва лабораторної роботи і завдання.

2) Перелік конструкцій обраної мови програмування, що визначають і використовують поняття чи об'єкт, для яких конструюється таблиця.

3) Структура запису таблиці і зв'язки з іншими таблицями, операції над записами таблиці.

4) Метод організації таблиці: неупорядкована (проста чи деревоподібна), упорядкована, перемішана (з відкритим доступом чи з ланцюжками переповнення).

5) Текст програм інструментальною мовою програмування, що описують таблицю і реалізують операції над записами таблиці.

6) Набір тестових даних і очікуваних результатів контролю правильності програм.

1.5 Контрольні питання

1. Яким вимогам повинні задовольняти організація таблиць транслятора і реалізація операцій роботи з таблицями?

2. Наведіть приклади статичних і динамічних таблиць транслятора.

3. Які методи прискорення пошуку можна ефективно використовувати для статичних таблиць транслятора?

4. Якими методами можна прискорити пошук у динамічних таблицях транслятора?

5. Запропонуйте кілька хеш-функцій для таблиці імен.

6. Сформулюйте достоїнства і недоліки різних структур таблиць транслятора.

7. Існують мови (ФОРТРАН, PL/1), у яких тип значення перемінної можна визначити по першому символу ідентифікатора. Незважаючи на це, обґрунтуйте необхідність таблиці ідентифікаторів у трансляторах для зазначених мов програмування.

8. Запропонуйте організацію таблиці ідентифікаторів для мов програмування з блоковою структурою, якщо в блоці дозволені описи локальних перемінних.

Лабораторна робота №2

КОНСТРУЮВАННЯ СКАНЕРІВ

Ціль роботи: Одержати практичні навички розробки компонент лексичного аналізатора.

Для виконання роботи необхідно знати інтерфейс взаємодії з таблицями транслятора, у яких міститься чи в які заноситься інформація про розпізнавані лексеми.

Виконання роботи полягає в розробці програм лексичного аналізатора для заданих типів лексем у трансляторі для обраної при виконанні лабораторної роботи №1 мови програмування.

Необхідні теоретичні зведення для виконання роботи викладені в [1, с. 69-100; 2, с. 122-129; 3, с. 101-124; 4, с. 41-53; 5, т.1, с. 283-294; 6, с. 104-109, 302-303; 9, с. 297-298; 312-316].

2.1 Теоретичні знання

У будь-якій мові програмування є послідовності символів, що утворюють єдині синтаксичні об'єкти, які позначаються лексемами: ключові слова, ідентифікатори, константи, операції і т.п. Виділення лексем у вихідному тексті програми є основною задачею лексичного аналізатора чи сканера.

Усі лексеми в трансляторі приводяться до єдиного формату і представляються дескриптором, що містить два поля (тип лексеми, значення).

Перше поле визначає тип об'єкта, що зв'язується з лексемою: ідентифікатор, константа і т.д. Друге поле містить інформацію чи покажчик на інформацію про дану лексему в таблиці, яка визначається типом лексеми.

Які послідовності символів вважати лексемами, і які типи лексем виділяти, залежить не тільки від мови програмування, але і від обраних алгоритмів трансляції.

Синтаксис лексем описується автоматними граматиками. У такий спосіб сканер можна розглядати як кінцевий автомат, входом якого служить ланцюжок символів вихідної програми, а виходом - послідовність лексем і таблиці лексем. Послідовність лексем часто називають лексичною згорткою програми. У лексичну згортку програми не включаються коментарі і пробіли.

Приклад 2.1. Розглянемо, які лексеми виділить сканер в операторі присвоювання мови ПАСКАЛЬ.

Вклад := 1,02 * Вклад;

Лексична згортка цього оператора представлена на рис. 2.1.

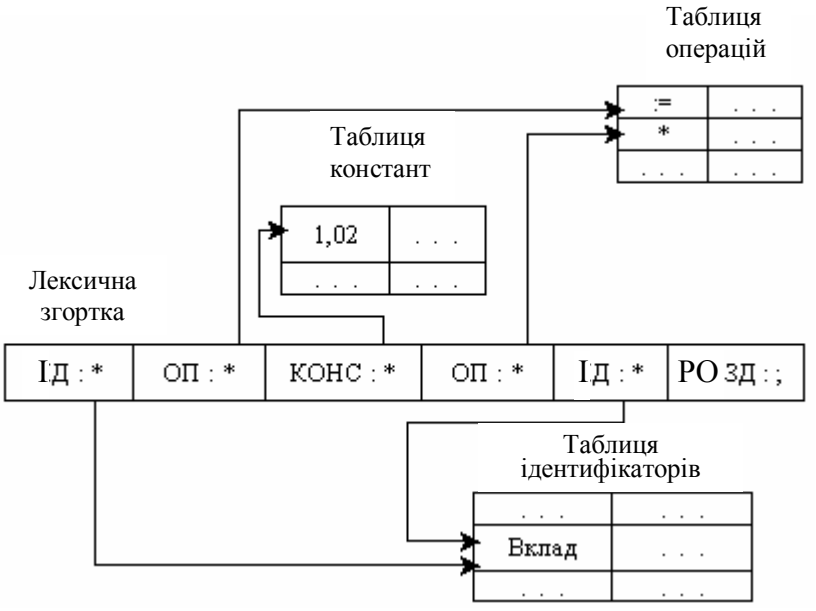


Рисунок 2.1 – Лексична згортка оператора присвоювання.

ЗАУВАЖЕННЯ. Використано наступні позначення класів лексем:

ІД – ідентифікатор; ОП - операція; КОНС – константа; РОЗД – роздільник. У мові ПАСКАЛЬ пробіли є роздільниками (крім вживання

усередині текстової константи). Тому в лексичній згортці програми пробіли відсутні.

Лексичний аналіз програми можна реалізувати у вигляді окремого перегляду усього вихідного тексту програми.

У цьому випадку необхідна пам'ять для лексичної згортки всієї програми. Звичайно сканер програмується як модуль, до якого звертається синтаксичний аналізатор усякий раз, коли йому потрібна наступна лексема.

Обов'язковими вхідними даними для сканера є послідовність символів тексту вихідної програми і пояснення покажчика, що фіксує поточний символ. Виділяють два крайніх способи організації роботи лексичного аналізатора: прямий і непрямий. На практиці також може використовуватися комбінація цих способів.

1) Сканер працює *прямо*, якщо він визначає лексему, лівий символ якої є поточним символом, і зрушує покажчик на перший символ, що не входить у лексему.

2) Сканер працює *непрямо*, якщо при звертанні до сканера задається тип очікуваної лексеми, у цьому випадку сканер перевіряє, чи утворюють символи вхідного ланцюжка, починаючи з поточного символу, лексему заданого типу. Якщо так, то покажчик зрушується за цю лексему.

Час роботи сканера значно впливає на загальний час трансляції, оскільки сканер обробляє вихідний текст програми по одному символу. Тому на практиці програму лексичного аналізатора часто оптимізують.

2.2 Варіанти завдань

2.2.1 Типи лексем мови АДА

1. Ідентифікатор: ключове слово, ім'я змінної, ім'я вбудованої функції, ім'я оператора-функції.

2. Константи: цілі, дійсні, комплексні, логічні, текстові, шістнадцятирічні.

3. Операції: арифметичні, відносини, логічні, присвоювання.

4. Позначки.

5. Роздільники.

2.2.1.1 Завдання для сканера з мови АДА

1) Ідентифікатор, ключове слово.
2) Ідентифікатор, дійсна константа.
3) Ідентифікатор, комплексна константа.
4) Ідентифікатор, ціла константа.
5) Позначки, коментарі.
6) Операції, ідентифікатор.
7) Ідентифікатор, роздільники.
8) Константи.

2.2.2 Типи лексем мови PL/1

1. Ідентифікатор: ключове слово, ім'я змінної, ім'я вбудованої функції, позначка процедури, позначка оператора (мітка-константа чи мітка-ім'я крапки входу в процедуру).

2. Константи: DEC FIXED, BIN FIXED, DEC FLOAT, BIN FLOAT, комплексні, рядок символів, рядок бітів.

3. Операції: арифметичні, порівняння, логічні, зчеплення, присвоювання.

4. Роздільники.

2.2.2.1 Завдання для сканера мови PL/1

1) Ідентифікатор, рядок біт.
2) Ідентифікатор, константа DEC FLOAT.
3) Ідентифікатор, константа DEC FIXED.
4) Ідентифікатор, константа BIN FLOAT.
5) Ідентифікатор, константа BIN FIXED.
6) Ідентифікатор, рядок літер.

7) Операції, ідентифікатор.

8) Позначки, мнимі константи.

9) Константи.

ПРИМІТКА. При виконанні всіх завдань повинні оброблятися коментарі і пробіли.

2.2.3 Типи лексем мови ПАСКАЛЬ

1. Ідентифікатор: ключове слово, позначення операції, ім'я константи, ім'я типу, ім'я змінної, ім'я процедури.

2. Константи: логічні, цілі, дійсні, строкові.

3. Операції: арифметичні, логічні, відносини, присвоювання.

4. Роздільники.

5. Позначки.

2.2.3.1 Завдання для сканера мови ПАСКАЛЬ

1) Ідентифікатор, дійсні константи.

2) Ідентифікатор, роздільники.

3) Ідентифікатор, цілі константи.

4) Ідентифікатор, строкові константи.

5) Константи.

6) Операції, ідентифікатор.

7) Позначки, дійсні константи.

8) Ідентифікатор, логічні константи.

ПРИМІТКА. Виконання всіх завдань повинне включати обробку коментарів і пробілів.

2.3 Порядок виконання роботи

1. Відповісти на контрольні питання.

2. Формально описати синтаксис розпізнаваних лексем.

3. Розробити алгоритми розпізнавання лексем.

4. Визначити дескриптори розпізнаних лексем для лексичної згортки програми і логіку взаємодії з необхідними таблицями транслятора.

5. Запрограмувати інструментальною мовою алгоритми розпізнавання лексем.

6. Вибрати представницькі приклади вихідних даних для автономного тестування створюваних програм.

7. Налагодити програми.

ПРИМІТКА. Програми розбору лексем обов'язково повинні взаємодіяти з програмами, що реалізують операції над записами у таблицях з першої лабораторної роботи. Допускається взаємодія з іншими таблицями на рівні інтерфейсу і найпростіших заглушок.

2.4 Зміст звіту

1. Назва лабораторної роботи.
2. Завдання.
3. Опис синтаксису розпізнаних лексем.
4. Вхідні і вихідні дані сканера.
5. Тестові приклади й очікувані вихідні дані.
6. Результати виконання розроблених програм для тестових даних.
7. Аналіз результатів.

2.5 Контрольні питання і вправи

1. Дайте визначення лексеми.
2. Якими граматиками описується синтаксис лексем?
3. Які способи завдання синтаксису лексем ви знаєте?
4. Які дані обробляє сканер, і що є результатом, його роботи?
5. Які функції в лексичного аналізатора?
6. Які причини відділення лексичного аналізу від синтаксичного аналізу?
7. Яким чином лексичний аналізатор може взаємодіяти із синтаксичним

аналізатором?

8. Які способи організації роботи сканера вам відомі?

9. Які лексеми мови ФОРТРАН виділить сканер для наступних операторів:

```
D0 10 I = 1,15
D0 10 I = 1.15
```

Розгляньте, як сканер прямо і непрямо для типів лексем ідентифікатор і ключове слово "DO" буде переглядати текст операторів. Врахуйте, що пробіли в мові ФОРТРАН не є роздільниками.

Лабораторна робота № 3

ПОБУДОВА ПОРОДЖУЮЧИХ ГРАМАТИК ДЛЯ КОНСТРУКЦІЙ МОВ ПРОГРАМУВАННЯ

Ціль роботи: Одержати практичні навички побудови граматик, що породжують, для конструкцій мов програмування.

Для виконання роботи необхідно знати поняття термінального та нетермінального символів, описання формальної граматики, поняття синтаксичного дерева, алгоритм низхідного і висхідного розбору речень.

Виконання роботи полягає в описі БНФ граматики та синтаксису конструкцій мов програмування, побудові синтаксичних дерев для конструкцій мов, проведенні низхідного та висхідного розбору речень.

Необхідні теоретичні знання для виконання роботи викладені в [5; 6; 7; 11; 12; 13; 14; 17].

3.1 Теоретичні знання

Опис будь-якої формальної мови здійснюється звичайно на іншій мові, яка зветься **метамовою**. Метамова може описувати або синтаксис (тобто форму конструкцій) формальної мови, або семантику (зміст цих конструкцій), або і те й інше разом.

Для мов програмування найбільш розповсюдженою метамовою для опису синтаксису є нормальна форма Бекуса (чи форма Бекуса-Наура), скорочено - БНФ. Виділяють основні поняття і конструкції цієї метамови:

- **термінальний символ** – символ, що складається тільки з букв алфавіту описуваної мови. У загальному випадку символом може бути одна буква чи кілька букв, що мають разом певне значення (наприклад, ключове слово *END*);

- **нетермінальний символ** – сформульоване на українській (чи будь-якій іншій) мові поняття описуваної мови програмування. Нетермінальні

символи беруться в кутові дужки: < >. Наприклад, <програма>, <символічне ім'я>, <арифметичний вираз>. Іноді нетермінальні символи називають металінгвістичними змінними (тобто змінними метамови).

Для того, щоб розкрити поняття мови, що позначається нетермінальним символом, використовуються так називані правила підстановки, чи металінгвістичні формули.

У загальному випадку правило підстановки записується у виді:

U ::= u ,

де U, u – довільні (кінцеві) послідовності (ланцюжок) нетермінальних і термінальних символів. А знак ::= означає “є по визначенню” чи “являє собою”.

При описі синтаксису мов програмування звичайно U – один нетермінальний символ, а u – кожна (у тому числі порожня) послідовність нетермінальних і термінальних символів, що розкриває (можливо не до кінця) сутність нетермінального символу, що стоїть в лівій частині.

Наприклад, правило підстановки, що визначає синтаксис умовного оператора мови PL/1:

<умовний оператор> ::= *IF* *<скалярний вираз>* *THEN*

<оператор>; *ELSE* *<оператор>*;

Тут u у правій частині складається з 4-х термінальних символів: *IF*, *THEN*, *ELSE* і 2-х нетермінальних: <скалярний вираз>, <оператор>. Для того, щоб цілком визначити умовний оператор, необхідно розкрити ці нетермінальні символи, причому робити це доти, поки в правих частинах правил підстановки не залишиться невизначених нетермінальних символів.

Якщо права частина правила підстановки може приймати кілька різних форм, для скорочення запису використовується символ | , що означає “чи”. Наприклад:

<колір> ::= *червоний* | *синій* | *жовтий*

<оператор> ::= *<оператор присвоювання>* |

<умовний оператор> | *<оператор циклу>*

У першому прикладі слова “*червоний*”, ”*синій*”, ”*жовтий*” записані як термінальні символи.

Синтаксис усієї мови (чи її частини – декількох конструкцій) визначається послідовністю правил підстановки. Одне з цих правил, що визначає (найбільш загальне) поняття мови, стоїть першим; нетермінальний символ, що позначає це поняття, називається початковим символом. Згадана послідовність правил підстановки називається граматикою, що породжує, тому що вона описує процедуру одержання (породження) правильних пропозицій мови. Граматика, що породжує, має наступне формальне визначення.

Формальна граматика G, що породжує, являє собою четвірку об'єктів (кортеж)

$$G = (N, T, \Sigma, П)$$

де N – безліч нетермінальних символів, що позначають конструкцію описуваної мови;

T – безліч термінальних символів, що складаються з букв алфавіту описуваної мови;

Σ – початковий символ граматики; $\Sigma \in N$;

П – набір правил підстановки.

Таким чином, щоб описати синтаксис формальної мови, досить задати N, T, Σ, П.

БНФ - не єдина метамова опису синтаксису формальних мов, однак її терміни і поняття стали загальновживаними і використовуються звичайно при визначенні граматик.

Приклад 1. Опис синтаксису (граматика) символічного імені.

<символічне ім'я> ::= <буква> | <символічне ім'я> <бц>

<бц> ::= A | B | C | D | ... | Z

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

У цьому прикладі використаний один дуже простий прийом – рекурсивне визначення. У першому правилі підстановки <символічне ім'я> визначається спочатку найпростішим образом як <буква>, а потім, після роздільника | (“чи”), нетермінальний символ <символічне ім'я> використовується з визначеним значенням. Після того, як <символічне ім'я> одержало значення <буква> <бц>, на наступному кроці воно може бути визначене як <буква> <бц> <бц> і так далі. Цей покроковий процес і є процесом породження конструкцій мови за допомогою граматики, що породжує. У теорії формальних мов такий процес називається *виведенням*.

Приклади виведення символічних імен:

1) <символічне ім'я> => <символічне ім'я> <бц> => <символічне ім'я> <бц> <бц> => <буква> <бц> <бц> => *X*<бц> <бц> => *X* <буква> <бц> => *XP*<бц> => *XP*<цифра> => *XP3*

2) <символічне ім'я> => <буква> => *A*

Зверніть увагу на те, що на будь-якому кроці виведення можна по-різному розкривати нетермінальні символи.

Це пов'язано з тим, що в прикладі 1 у правих частинах усіх правил підстановки присутні роздільники | (“чи”), що дозволяють довільний вибір. Виведення закінчується, коли в отриманому ланцюжку всі символи термінальні.

Використання рекурсивного визначення нетермінальних символів дозволяє одержувати як завгодно довгі речення мови, тому що рекурсія природним образом забезпечує повторюваний (циклічний) процес.

Узагалі, якщо необхідно описати синтаксис як завгодно довгої послідовності об'єктів, варто застосовувати рекурсію:

<послідовність об'єктів> ::= <об'єкт> | <послідовність об'єктів>

<об'єкт>

У якості об'єктів можуть виступати, наприклад, букви і цифри в іменах і константах, імена з комами в операторах опису, списках параметрів процедури, списках введення-висновку й інші конструкції.

Нижче приведений приклад застосування БНФ для повного опису синтаксису простої мови програмування. У цьому описі можна знайти практично всі особливості використання БНФ, зокрема, опис синтаксису арифметичних і логічних виразів, що є вже класичним.

Приклад 2. Граматика мови програмування.

Спочатку дамо її короткий неформальний опис.

Мова дозволяє оперувати з даними цілого (integer) і логічного (logical) типів. Логічні значення: true, false. Дане може бути константою, змінною чи одномірним масивом цілих чисел. Масив описується в такий спосіб: *array* <ім'я> [n_1 : n_2], де n_1, n_2 – цілі константи, що визначають нижню і верхню границі зміни індексу. Елемент масиву (змінна з індексом) записується у вигляді: [індексний вираз], наприклад: A[i], B[k*m+1].

Оператори мови S_i:

- оператор присвоювання: X := E;
- умовний оператор: *if* <умова> *then* <оператор> *else* <оператор> *fi* чи *if* <умова> *then* <оператор> *fi*;
- оператор циклу: *while* <умова> *do* <*оператор*> *od*;
- порожній оператор.

де S_i – будь-який з перерахованих вище операторів.

Формальний опис синтаксису мови БНФ:

<програма> ::= <опис> <оператор>

<опис> ::= <оператор опису>*;* | <опис> <оператор опису>*;*

<оператор опису> ::= <оператор опису типу> | <оператор опису масиву>

<оператор опису типу> ::= <тип> <список типу>

<список типу> ::= <ім'я> | <список типу> , <ім'я>

<оператор опису масиву> ::= *array* <список масивів>

<список масивів>::= <елемент списку> | <список масивів> , <елемент списку>

<елемент списку> ::= <ім'я> [<ціле> : <ціле>]

<оператор> ::= <оператор присвоювання> | <умовний оператор> | <оператор циклу> | <порожній оператор> | <оператор>; <оператор>

<оператор присвоювання> ::= <перемінна> := <вираз>

<умовний оператор> ::= *if* <логічна умова> *then* <оператор> *else* <оператор> *fi* | *if* <логічна умова> *then* <оператор> *fi*

<оператор циклу> ::= *while* <логічний вираз> *do* <оператор> *od*

<порожній оператор> ::=

<змінна> ::= <проста змінна> | <змінна з індексом>

<проста змінна> ::= <ім'я>

<змінна з індексом> ::= <ім'я> [<арифметичний вираз>]

<вираз> ::= <арифметичний вираз> | <логічний вираз >

<арифметичний вираз> ::= <доданок> | + <доданок> | - <доданок> | <арифметичний вираз> | + <доданок> | <арифметичний вираз> - <доданок>

<доданок> ::= <множник> | <доданок> * <множник> | <доданок> \ <множник>

<множник> ::= <ціле> | <змінна> | *(*<арифметичний вираз>*)* | <множник> ** <множник>

<логічний вираз> ::= <логічний доданок> | <логічний доданок> ∪ <логічний вираз>

<логічний доданок> ::= <логічний множник> | <логічний множник> ∩ <логічний доданок>

<логічний множник> ::= <логічна константа> | <проста змінна> | <арифметичний вираз> <операція відносини> <арифметичний вираз> | *(*<логічний вираз>*)* | ¬ <логічний множник>

<логічна константа> ::= *true* | *false*

<операція відносини> ::= = | ≠ | > | < | ≥ | ≤

<ім'я> ::= <буква> | <ім'я> <буква> | <ім'я> <цифра>

<ціле> ::= <цифра> | <ціле> <цифра>

<буква> ::= *A* | *B* | *C* | *D* | ... | *Z* | *a* | *b* | *c* | ... | *z*

<цифра> ::= *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9*

У багатьох випадках для скорочення і підвищення наочності опису синтаксису використовують так називану модифіковану БНФ. Модифікація

полягає в додаванні нових символів метамови. Найбільш часто застосовуються квадратні і фігурні дужки.

Частина речення, укладена в квадратні дужки, може бути або відсутньою, або присутньою.

Приклад 3. Використання квадратних дужок у БНФ.

1) <умовний оператор> ::= if <логічна умова> then <оператор> [else <оператор>] fi

2) <дійсна константа> ::= [<знак>] <ціле>.<ціле> [E [<знак>] <ціле>]

< знак > ::= + | -

<ціле> ::= <цифра> | <ціле> <цифра>

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Частина речення, укладена у фігурні дужки, може або бути відсутньою, або повторюватися будь-яке число разів.

Приклад 4. Використання фігурних дужок у БНФ.

<символічне ім'я> ::= <буква> { <буква> | <цифра> }

<ціле> ::= <цифра> { <цифра> }

<список імен> ::= <ім'я> { , <ім'я> }

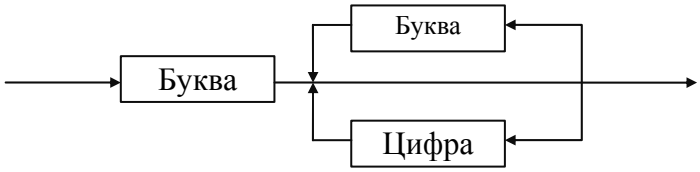
Число повторень частини речення, укладеної у фігурні дужки, може бути обмежено знизу і зверху підрядковими і надрядковими індексами. Це записується в такий спосіб: $\{\ldots\}_m^n$, де m – мінімальне, n – максимальне число повторень. Наприклад, ланцюжок символів $<буква>\{<буква>|<цифра>\}_0^5$ визначає символічне ім'я довжиною не більш 6-ти символів.

Інший розповсюджений спосіб опису синтаксису мов програмування – синтаксичні діаграми, що є фактично графічним зображенням нормальної форми Бекуса. Нетермінальні символи записуються на діаграмі в

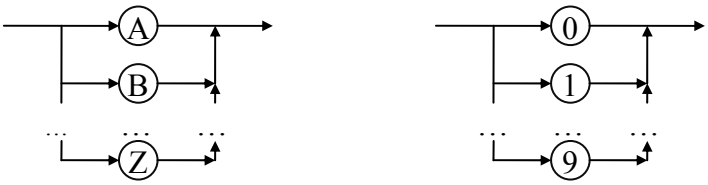
прямокутниках, термінальні – у кружках (довгі термінальні символи можуть зображуватися в овалах). Символи з'єднуються стрілками, що вказують послідовність читання символів, що утворюють ланцюжок. Символ | (“чи”) на діаграмі зображується петлею.

Приклад 5. Визначення синтаксису символічного імені.

Ім'я

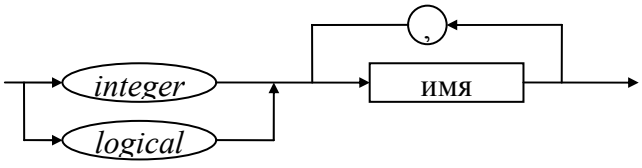


Буква Цифра



Приклад 6. Опис синтаксису оператора опису в мові з приклада 2.

Оператор опису



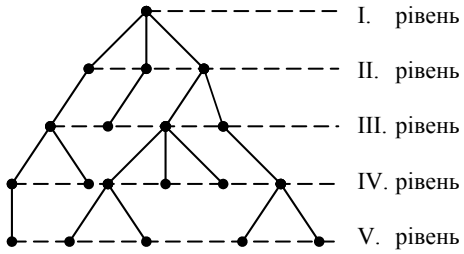
В свою чергу, нетермінальний символ <Ім'я> визначений вище в прикладі 5.

Синтаксичний аналіз конструкцій мов програмування

Синтаксичний аналіз мовних конструкцій являє собою задачу, протилежну задачі породження (виведення). Задача синтаксичного аналізу (задача розбору) формулюється в такий спосіб: визначити, чи відповідає дана конструкція деякої мови граматиці цієї мови або, іншими словами, чи є дана конструкція правильним (не утримуючим синтаксичних помилок) реченням мови. Задача розбору має широке практичне застосування: кожен транслятор мови програмування має у своєму складі блок синтаксичного аналізу.

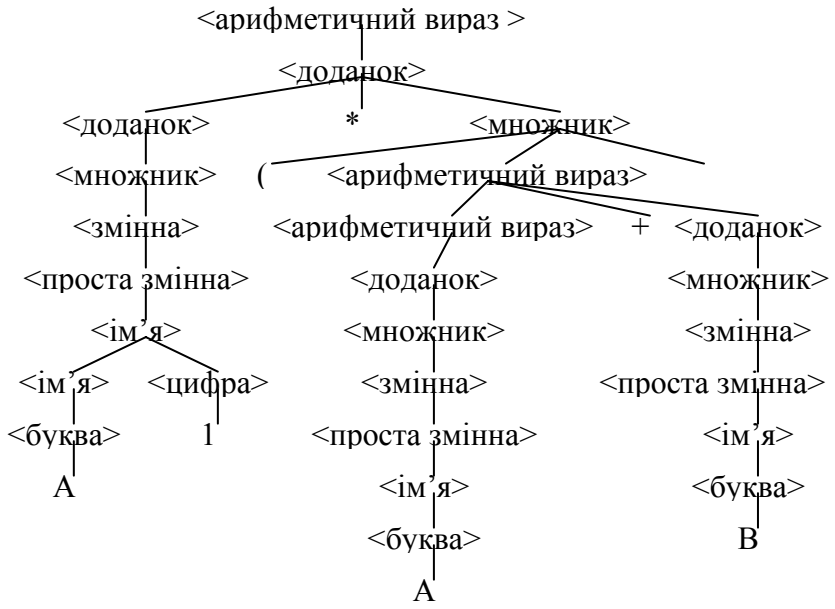
Перш ніж розглядати алгоритми синтаксичного аналізу, введемо поняття **синтаксичного дерева** як графічного способу зображення висновку конкретного речення мови.

Дерево являє собою ієрархічну структуру, що складається з вузлів різних рівнів; кожний з вузлів деякого рівня (крім останнього) зв'язаний з декількома вузлами наступного рівня і не більш, ніж з одним вузлом попереднього рівня.



На першому рівні може бути тільки один вузол, який називається **коренем**. Вузли останнього рівня називаються **листами**. **Кущ** вузла – безліч підлеглих йому вузлів нижніх рівнів. У синтаксичних деревах вузли являють собою нетермінальні і термінальні символи, причому листи є термінальними символами, а інші вузли – нетермінальними. Корінь дерева – початковий символ граматики. Кущ являє собою праву частину правила підстановки для даного нетермінального символу.

Приклад 7. Синтаксичне дерево для арифметичного виразу A1*(A+B).



Достоїнством синтаксичних дерев є наочність опису синтаксису як самого речення, так і його окремих складових. З іншого боку, для реальних конструкцій мов програмування дерево виявляється дуже громіздким. Тому, якщо дерево використовується для одержання наочності опису, то, або опускаються деякі проміжні рівні, несуттєві для цього опису, або воно складається для простих речень. Сама ідея побудови дерев широко використовується в трансляторах, тому що існують ефективні методи відображення дерев у пам'яті ЕОМ.

Синтаксичний аналіз речень, записаних на мовах програмування, фактично зводиться до побудови синтаксичних дерев різними методами. Існують два базових алгоритми синтаксичного аналізу (рішення задачі розбору):

- низхідний розбір (розгорнення);
- висхідний розбір (згортка).

При низхідному розборі синтаксичне дерево будується від кореня до листів. Його відмінна риса – цілеспрямованість, тому що, відправляючись від нетермінального символу, ми прагнемо знайти таку підстановку, що привела б до частини необхідного ланцюжка термінальних символів. У найпростіших випадках синтаксичний аналізатор (розпізнавач) намагається досягти цього

шляхом спрямованого перебору різних варіантів. Розпізнавач розглядає дерево зверху вниз і зліва направо і вибирає перший нетермінальний символ, що не має підлеглих вузлів (що є “листом”). У списку правил підстановки відшукується правило (чи набір правил), що у лівій частині містить цей нетермінальний символ, а в правій частині (у правих частинах) – чергові (рахуючи зліва направо) символи аналізованого речення. Якщо таке правило є, дерево нарощується, й описаний процес повторюється. Якщо правило не знайдене, розпізнавач повертається на один чи кілька кроків назад, намагаючись змінити вибір, зроблений раніше; процес розбору закінчується в одному з двох випадків:

1) Побудовано дерево, усі листи якого є термінальними символами і при читанні зліва направо утворюють аналізоване речення. У цьому випадку результат розпізнавання позитивний, тобто синтаксис розглянутого речення відповідає граматиці мови.

2) Розпізнавач переробив усі можливі варіанти послідовностей підстановок, але так і не прийшов до дерева, яке описане в п.1. Це означає, що аналізоване речення не належить даній мові (тобто містить помилки). Варто підкреслити, що помилка вважається виявленою тільки в тому випадку, коли розглянуті всі припустимі варіанти підстановки.

Розглянемо цей процес на прикладі.

Приклад 8. Провести низхідний синтаксичний розбір числа +2.4 по граматиці.

(1) <константа> ::= <КФТ> | <знак> <КФТ>
(2) <КФТ> ::= .<ціле> | <ціле>. | <ціле>.<ціле>
(3) <ціле>::= <цифра> | <цифра> <ціле>
(4) <знак> ::= + | –
(5) <цифра> ::= *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9*

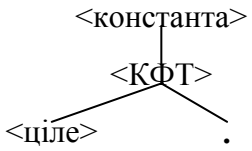
Процес розбору покажемо у виді послідовності кроків, на кожнім з який будується якась частина дерева.

Крок 1.

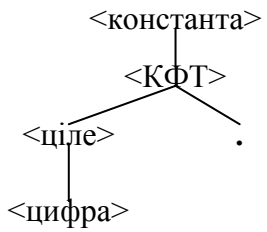
<константа> – корінь дерева.

Крок 2. Оскільки вираз (1) не містить у правій частині термінальних символів, обираємо перший зліва ланцюжок символів: <константа> ::= <КФТ>. Одержуємо

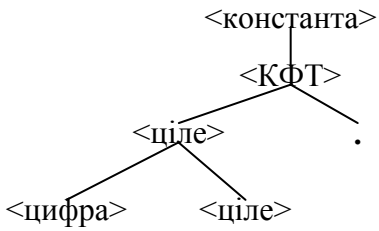
Крок 3. У правилі (2) перший ланцюжок починається з термінального символу ".", а тому що перший символ аналізованого речення – "+", цей ланцюжок не підходить. Вибираємо наступну.



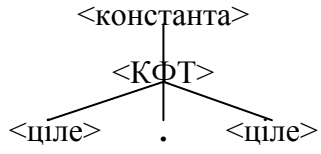
Крок 4. У правилі (3) вибираємо перший ланцюжок <цифра>:



Крок 5. Для визначення нетермінального символу <цифра> за правилом (5) є 10 термінальних символів, але жоден з них не збігається із символом "+". Тому здійснюємо повернення на один крок назад і, знову користуючись правилом (3), одержуємо:

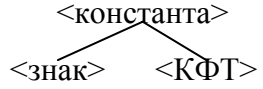


Крок 6. З двох гілок дерева вибираємо, відповідно до алгоритму, ліву і знову намагаємося застосувати правило (5). Тому що це знову не приводить до успіху, повертаємося назад до дерева, яке отримали на етапі 2 (оскільки спроби застосування правила (3) вичерпані. У правилі (2) вибираємо ланцюжок <ціле>.<ціле>. Одержуємо дерево:

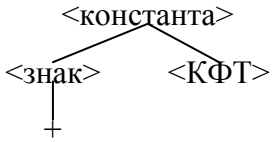


На цьому шляху ми, мабуть, зробимо ще 2 кроки, перш ніж переконатися в його хибності. Проробіть їх самостійно.

Крок 9. Вертаємося до правила (1) і вибираємо для підстановки другий ланцюжок <знак><КФТ>.

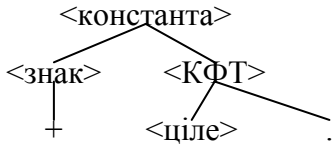


Крок 10. Застосовуємо правило (4) до нетермінального символу <знак>. Одержуємо.



Отриманий термінальний символ збігається з першим символом аналізованого речення, отже, гілка отриманого дерева правильна. Далі намагаємося одержати наступний символ –“2”.

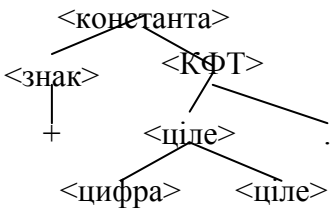
Крок 11. Розкриваємо нетермінальний символ <КФТ> за правилом (2), при цьому вибираємо другий ланцюжок, тому що в першому символ “.” очевидно не збігається із символом “2”.



Кроки 12, 13. Застосовуємо правила (3) і (5).

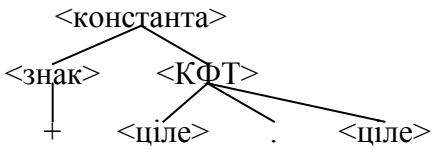
Одержали відразу два термінальних символи аналізованого речення “2” і “.”. Однак дерево вже побудоване, тому що всі листи є термінальними символами. Якби усі варіанти підстановок були вже випробувані, можна було б зробити висновок, що в реченні +2.4 допущена помилка. Оскільки це не так, робимо два кроки назад, повертаючись до дерева, отриманого на кроці 11.

Крок 14. Снову застосовуємо правило (3), але його другий ланцюжок.

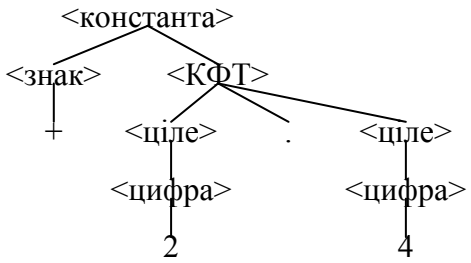


Очевидно, що на цьому шляху ми зробимо ще 3 кроки (переконаєтеся в цьому!), перш ніж зайдемо в тупик. Пропускаємо ці три кроки і повертаємося, нарешті, до дерева, отриманого на кроці 10.

Крок 18. Застосовуємо до символу <КФТ> на цьому дереві правило (2) (третій ланцюжок).



Кроки 19, 20, 21, 22. Розкриваємо послідовно лівий і правий символи <ціле>.



Листи цього дерева (зліва направо) утворюють вихідне речення +2.4, отже воно відповідає заданій граматиці, тобто не містить помилок.

Завжди варто мати на увазі, що тільки неухильне, точне проходження алгоритму низхідного розбору дозволяє гарантувати правильність розбору, особливо для складних граматик. Спроби пропустити кілька кроків як очевидні чи зробити їх “у розумі” часто приводять до помилок.

Примітка. Описаний алгоритм низхідного розбору не можна застосовувати, якщо граматика є ліво-рекурсивною, тобто містить правила підставлення виду

$$U ::= Uu$$

оскільки при цьому спроби розкрити самий лівий нетермінальний символ приведуть до нескінченного породження нетермінального символу U. У такому випадку необхідно видозмінити алгоритм, зробивши його, наприклад, правостороннім, чи змінити граматику.

В алгоритмі низхідного розбору не обов'язково будувати дерево. На кожному кроці можна записувати ланцюжок нетермінальних і термінальних символів, отриманих після чергової підстановки.

При висхідному розборі дерево будується від листів до кореня, тобто алгоритм, відправляючись від заданого рядка, намагається, застосовуючи правила підстановки справа наліво, привести її до початкового символу граматики.

Частина рядка, яку можна привести до нетермінального символу, називається **фразою**. Якщо приведення здійснюється застосуванням одного правила підстановки, фраза називається безпосередньо **зводимою**. Сама ліва фраза, що безпосередньо зводиться, називається **основною**.

Алгоритм висхідного розбору полягає в наступному. У вихідному реченні відшукується основа (тобто перегляд, як і в низхідному розборі, йде зліва направо) і зводиться до нетермінального символу. Ця операція застосовується доти, поки або отриманий єдиний символ, що є початковим символом граматики (у цьому випадку розбір закінчується і робиться висновок про правильність речення, що розбирається,), або в отриманому ланцюжку не може бути знайдена фраза. В останньому випадку робиться повернення на один чи кілька кроків назад і вибирається інша основа. Якщо всі можливі варіанти перебрані, а корінь дерева так і не отриманий, робиться висновок про наявність помилки в реченні, що розбирається, і алгоритм припиняє роботу.

Таким чином, висхідний розбір являє собою перебір варіантів, однак не цілеспрямований, тому що тут немає можливості відкидати свідомо неправильні підстановки, як це має місце в низхідному розборі (ланцюжок “.<ціле>” у правилі (2), невідповідні цифри в правилі (5) у прикладі 9).

Процес висхідного розбору записують або у вигляді побудови дерева від листів до кореня, або у вигляді послідовності ланцюжків термінальних і нетермінальних символів, що починаються з аналізованого речення, яке

закінчується (якщо речення не містить помилок) початковим символом граматики.

Головним недоліком як низхідного, так і висхідного алгоритмів розбору є велика кількість кроків, що визначається переборним характером алгоритмів.

3.2 Варіанти завдань

Варіанти мов для опису БНФ:

a) безліч парних чисел в алфавіті {0,1};

b) безліч непарних чисел в алфавіті {0,1};

c) безліч парних чисел в алфавіті {0,1,2,3,4,5,6,7,8,9};

d) безліч непарних чисел в алфавіті {0,1,2,3,4,5,6,7,8,9};

e) мови з a), b), c), d), крім чисел з незначними нулями;

f) мова, що складається з речень виду $ab^na|n=0,1,2,\ldots$;

g) мова, що складається з речень виду $a^nb^n|n=0,1,2,\ldots$;

h) мова, що складається з речень виду $a^mb^nc^p|m,n,p=0,1,2,\ldots$;

i) мова, що складається з речень виду $a^mb^nc^m|m,n=0,1,2,\ldots$;

j) мова, що складається з речень виду x^nay^n, x^nby^n $|m,n=0,1,2,\ldots$;

k) мова, що складається з ланцюжків 0 і 1, і утримуючих принаймні три підряд 1 (наприклад 010111101);

l) граматика класу 3 імен мови програмування, що складаються не більш, ніж з 6 букв і (чи) цифр;

m) граматику класу 3, що генерує ту ж мову, що і граматика

<S> ::= <A><B>;

<A> ::= <X> | <Y>;

<X> ::= $\underline{x}$ | $\underline{x}$<X>;

<Y> ::= $\underline{y}$ | $\underline{y}$<Y>;

<B> ::= $\underline{b}$ | $\underline{b}$<B>.

Таблиця 3.1 – Варіанти граматик для опису БНФ.

№ варіанта	Граматики	№ варіанта	Граматики
1	a,k	**14**	m,f
2	b,m	**15**	l,e
3	c,e	**16**	k,d
4	d,k	**17**	j,c
5	e,g	**18**	i,b
6	f,j	**19**	h,a
7	g,i	**20**	g,b
8	h,f	**21**	h,c
9	i,e	**22**	i,d
10	j,d	**23**	j,e
11	k,c	**24**	k,f
12	e,b	**25**	l,g
13	m,a		

Таблиця 3.2 – Варіанти конструкцій мови.

№	Нетермінальний символ	Речення
1.	<вираз>	(X+A1[i])/C**3
2.	<оператор опису масиву>	*array* P[0:10], MAX[45:100]
3.	<умовний оператор>	*if* A>B *then* x:=0 *else* x:=y *fi*
4.	<вираз>	(A ∨ ¬ (B ≥ C))
5.	<описання>	*logical* x1,x2; *integer* N;
6.	<оператор присвоювання>	T[K]:=B-(C-D)*E
7.	<оператор циклу>	*while* x≠0 *do* x:=x-1; A[x]:=1 *od*
8.	<вираз>	X**(A+B)-Z
9.	<програма>	*integer* A,B; A:=B;
10.	<описання>	*logical* A,B,C; *integer* XZ,MAC;
11.	<вираз>	A+B[i]≥C ∧ ¬ P
12.	<програма>	*integer* I,A; X[I]:=A;
13.	<оператор опису масиву	*array* N[5:10],Mkd[1:15]
14.	<арифметичний вираз>	X**Z**A-5*A
15.	<оператор опису>	*locical* A,AL,ALPHA,BETA
16.	<програма>	*integer* X1,YP; X1:=2; YP:=X1;
17.	<оператор циклу>	*while* I≤N *do* A[I]:=0; I:=I+1; *od*

18.	<умовний оператор>	*if* C *then* X:=1; Y:=A+B *fi*
19.	<логічний вираз>	¬(B>X+Y∨C∧D)
20.	<оператор присвоювання>	B:=X*MAX[i+1]-C
21.	<оператор циклу>	*while* B *do* X:=X+A[N] *od*
22.	<оператор присвоювання>	C[1]:=D[(A+B)-K[i]]
23.	<оператор опису масиву>	*array* ABC[5:5], X[20:10]
24.	<програма>	*logical* T; *integer* X; X:=T+1;
25.	<оператор опису типу>	*integer* M1,M2,MIN,LP

Таблиця 3.3 – Варіанти завдань для висхідного розбору.

№	Пункт із задачі 1.	Речення		№	Пункт із задачі 1.	Речення	
		1	2			1	2
1.	f)	Abbbaa	Abba	14.	g)	Aabaa	Ab
2.	g)	Abbaa	Aaabbb	15.	h)	Aaacccc	Bbbcca
3.	h)	Abccc	Aabbca	16.	i)	Ababcc	Aabb
4.	i)	Aaabb	Ab	17.	j)	Xxxayyy	Xayxay
5.	j)	Xxxayyy	Xayy	18.	k)	11111	110
6.	k)	10111	101101	19.	f)	Aabb	Abbbba
7.	f)	Abbba	Aaabba	20.	g)	aaaabbbb	Aaaab
8.	g)	Aabb	Aabb	21.	h)	Bbccc	Aaaacb
9.	h)	A	Abbbac	22.	i)	Cccab	Abcccc
10.	i)	Ccc	Aabccc	23.	j)	A	Xxaay
11.	j)	Xxayy	Xay	24.	k)	1010	1101110
12.	k)	111110	110110	25.	f)	Aabba	Bbba
13.	f)	Aa	Aabbaa				

Таблиця 3.4 - Варіанти завдань для низхідного розбору.

№	Речення	№	Речення	№	Речення
1.	Max[5]	10.	Z[5]:=0	19.	A:=B2
2.	*array* X1[1:10]	11.	*logical* X1,X2	20.	156-C
3.	0=1	12.	BA1085	21.	*integer*
4.	Z:=X+Y	13.	A[I+5]	22.	X=Y
5.	*integer* A,B1	14.	1=A	23.	*array*
6.	ZX:=1	15.	*logical* X,Y,Z	24.	A>B
7.	X[3+K]	16.	*while* x *do* *od*	25.	K:=K+1
8.	A+B	17.	X1:=1	26.	BT[I]
9.	AB=CD	18.	*if* B *then* *fi*	27.	X:=X

Таблиця 3.5 – Варіанти завдань для висхідного розбору по заданій граматиці.

№	Числа		№	Числа		№	Числа	
	I	II		I	II		I	II
1	5.0E+4	+3.1	**10**	0.1E8	11.1E-1	**19**	0.202E-02	15
2	3.2E+5	3.23	**11**	7.77	5.E-8	**20**	46.46E-46	-36.1
3	2.6E1	7.8	**12**	2.80	7.2E8	**21**	+5.6E+10	17.1E-5
4	4.E-5	0.0	**13**	15E15	0.0E+0	**22**	E-8	1.1E+555
5	22.1E-01	+3E5	**14**	7.7E-7	2.5-3	**23**	7E-3	25.2255
6	35.482	-53	**15**	5.33E-2	20	**24**	100.001	17E17
7	-2.0E-5	35.1E+1	**16**	9.9E-99	-1E-2	**25**	-4E-5	2.3E+45
8	01.1	5E-4	**17**	1E1	22.22			
9	20.E1	11.111	**18**	5.4	2.3E-1.0			

3.3 Порядок виконання роботи

1) Опишіть БНФ граматики наступних мов (укажіть також клас мови за Хомським). Варіанти завдань приведені у таблиці 3.1.

2) Опишіть у БНФ синтаксис конструкцій мови програмування, для яких будувалися таблиці в першій лабораторній роботі.

3) Опишіть граматики мов і конструкцій завдань 1) і 2) за допомогою модифікованої БНФ і синтаксичних діаграм.

4) Приведіть виведення наступних конструкцій мови по граматиці з прикладу 2.

5) Побудуйте синтаксичні дерева для конструкцій мови з прикладу 2, заданих у таблиці 3.2.

6) Проведіть низхідний розбір речень мов, заданих у пункті 1.

Примітка. Якщо граматика мови ліворекурсивна, модифікуйте алгоритм так, щоб він був застосовний до даної граматики.

7) Проведіть висхідний розбір речень з таблиці 3.3.

8) Проведіть низхідний розбір речень, заданих у таблиці 3.4 по граматиці мови з прикладу 2.

Примітка. Корінь дерева виберіть самостійно.

9) Проведіть висхідний розбір речень (чисел), заданих у таблиці 3.5 по граматиці:

<число> ::= <мантиса> | <мантиса> <порядок>

<мантиса> ::= <ціле>.<ціле>

<порядок> ::= *E*<знак><ціле>

<знак>::= + | –

<ціле> ::= <цифра> | <ціле> <цифра>

<цифра> ::= *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9*

3.4 Зміст звіту

1) Назва лабораторної роботи і завдання.
2) Опис БНФ граматики мови.
3) Опис у БНФ синтаксису конструкцій мови програмування.
4) Опис граматики мов і конструкцій за допомогою модифікованої БНФ і синтаксичних діаграм.
5) Вивід конструкцій мов по граматиці.
6) Побудовані синтаксичні дерева для конструкції мови.
7) Низхідний розбір речень мови.
8) Висхідний розбір речень мови.
9) Низхідний розбір речень по заданій граматиці.
10) Висхідний розбір чисел по заданій граматиці.

3.5 Контрольні питання

1. Що таке метамова?
2. Чим відрізняються нетермінальні символи від термінальних?
3. Дайте визначення породжувальної граматики.
4. Укажіть відповідність між БНФ і синтаксичними діаграмами.

5. Чому граматики класу 1 називаються контекстно-залежними, а класу 2 – контекстно-вільними?

6. Поясніть зміст наступних термінів: ланцюжок, правило підстановки, початковий символ граматики, висновок, корінь, кущ, фраза, основа.

7. Які правила підстановки в граматиці з приклада 2 відносяться до класу 2 за Хомським, а які – до класу 3?

8. Поясніть різницю між задачею породження і задачею розбору.

9. Які методи синтаксичного аналізу вам відомі? Чим вони відрізняються один від одного?

Лабораторна робота № 4

ГРАМАТИКИ - РОЗПІЗНАВАЧИ

Ціль роботи: Одержати практичні навички побудови кінцевих автоматів-розпізнавачей та автоматів з магазинною пам'яттю (МП-автоматів).

Для виконання роботи необхідно знати, як задаються кінцевий автомат-розпізнавач та МП-автомат, принципи їх побудови.

Виконання роботи полягає в побудові кінцевого автомата-розпізнавача для обраної мови, а також МП-автомата для розпізнавання обраної конструкції.

Необхідні теоретичні знання для виконання роботи викладені в [5; 6; 7; 11; 12; 13; 14; 17].

4.1 Теоретичні знання

Граматики-розпізнавачи (граматики, що розпізнають чи просто розпізнавачи) являють собою принципово інший спосіб завдання синтаксису формальних мов. Побудова і застосування цих граматик цілком орієнтовані на рішення задачі розбору, а не породження мови.

Кінцевий автомат - розпізнавач для мов классу 3 за Хомським.

Визначення. Кінцевий автомат-розпізнавач К – це п'ятірка;

$$К = (A, Q, \delta, q_1, z),$$

де A – вхідний алфавіт (включаючи порожній символ λ);

Q – множина станів керуючого пристрою;

δ – функція переходів автомата; $q'_j = \delta(q_j, a_i)$;

$q_1 \in Q$ – початковий стан керуючого пристрою;

$z \subseteq Q$ – множина заключних станів керуючого пристрою.

Множина Z складається з 2-х елементів: $\{q_{z_1}, q_{z_2}\}$; q_{z_1} – відповідає успішному закінченню розбору, q_{z_2} – закінчення розбору помилкового речення.

Кінцевий автомат – розпізнавач задається звичайно таблицею переходів. Заключні стани в клітках цієї таблиці звичайно позначаються: q_{z_1} – знак "+"; q_{z_2} – знак "–" чи порожня клітка.

Приклад 1. Побудувати розпізнавач для мови, що складається з ланцюжків 0 та 1, які обов'язково містять два підряд стоячих нуля.
Таблиця переходів має вигляд:

Таблиця 4.1 – Таблиця переходів.

		A		
		0	**1**	**λ**
Q	**q_1**	q_2	q_1	–
	q_2	q_3	q_1	–
	q_3	q_3	q_3	+

Тут A = {0,1}; Q = {q_1, q_2, q_3, "+", "–"}; Z = {"+", "–"}.
λ – порожній символ, який фактично позначає кінець ланцюжка. Припустимо, на вхід автомата поступає ланцюжок 1011001λ…. Послідовність станів керуючого пристрою для цього ланцюжка: $q_1q_1q_2q_1q_1q_2q_3q_3+$. Для ланцюжка 01101λ… послідовність станів: $q_1q_2q_1q_1q_2q_3q_3-$.

Одним з найважливіших принципів, яким варто керуватися при побудові кінцевих автоматов-розпізнавачей, є запам'ятовування інформації про аналізоване речення за допомогою станів пристрою керування. Оскільки КА не може записувати проміжну інформацію на стрічку, єдиним способом запам'ятовування інформації є перехід у новий стан при виявленні у вхідному

рядку символу (групи символів), наявність якого необхідно запам'ятати. Розглянемо на прикладі.

Приклад 2. Побудувати розпізнавач для дійсних констант, які записуються в однієї з наступних форм:

[±]<цифри>E[±]<цифри>,

[±]<цифри>.<цифри>,

[±]<цифри>.<цифри>E[±]<цифри>.

Наприклад: 5.32, 7E-2, 17.0E10.

При перегляді вхідного рядку необхідно запам’ятовувати наявність цифр цілої частини (можливо зі знаком), крапки, цифр дрібної частини, символу “E”, знака, цифр порядку. Тому з урахуванням початкового стану q_1 КА буде мати 8 станів (не рахуючи двох заключних). Побудуємо таблицю переходів КА.

Таблиця 4.2 – Таблиця переходів.

		A				
		“0”…“9”	**“.”**	**“E”**	**“±”**	**λ**
Q	**q_1**	q_3			q_2	
	q_2	q_3				
	q_3	q_3	q_4	q_6		
	q_4	q_5				
	q_5	q_5		q_6		+
	q_6	q_8			q_7	
	q_7	q_8				
	q_8	q_8				+

В таблиці q_{z_1} відповідає знак “+” (число записано без помилок), q_{z_2} – порожня клітинка (запис помилкова).

Стан q_2 “зберігає” інформацію про наявність на стрічці “±”; стан q_3 – цифр цілої частини; стан q_4 – про наявність крапки; стан q_5 – цифр дрібної частини; стан q_6 – символу “E”; стан q_7 – знаку порядку. Робота КА закінчується або при зчитуванні порожнього символу, або при виявленні помилки.

Автомат з магазинною пам’яттю для розпізнавання мов классу 2 за Хомським

МП-автомат має робочу пам'ять, організовану за принципом стека чи магазина автоматичної зброї. Магазин (стек) являє собою лінійну структуру даних, відкриту для доступу тільки з одного кінця:

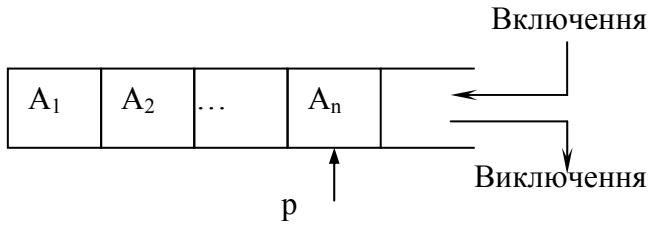


Рисунок 4.1 - Структура автомату з магазинною пам’яттю

Комірка, що містить A_n, називається вершиною магазина, змінна p – покажчиком вершини. При включенні нового елементу p збільшується на 1, при виключенні зменшується на 1. Робота магазина описується формулою: “останнім прийшов – першим пішов”.

Визначення – МП-автомат – це сімка:

$$M = (A, Q, P, \delta, q_1, p_1, Z),$$

де A, Q, q_1, Z – те ж, що й у КА;

δ– функція переходів і запису в робочу пам'ять, що задає відображення $Q^*A^*P \rightarrow Q^*P^*$;

P – алфавіт робочої пам'яті;

P^* – множина слів в алфавіті P, включаючи порожнє;

p_1 – символ, що находиться в робочій пам'яті в початковий момент (початковий символ).

Функціонування МП-автомата являє собою послідовність кроків (тактів), у кожнім з яких керуючий пристрій виконує наступні елементарні операції:

1) читає символ у вершині магазина і можливо (але не обов'язково) на вхідній стрічці;

2) по q_j, a_i і p_l переходить у новий стан q'_j;

3) виробляє слово $\mu = p^{(1)}p^{(2)}\ldots p^{(n)}$(можливо порожнє);

4) виключає p_l з магазина і записує в магазин слово μ так, що б $p^{(1)}$ виявився на вершині магазина;

5) пересувається по вхідній стрічці на одну комірку вправо чи залишається на місці.

МП-автомат вважається заданим, якщо визначені множини Q, A, P, Z, символ p_l і набір виразів, що задають функцію δ і записуваних у виді:

$$\delta(q_j, a_i, p_l) = (q'_j, \mu). \quad (1)$$

Якщо ліва частина виразу (1) має вид $\delta(q_j, \lambda, p_l)$, це значить, що керуючий пристрій не оглядає символ на вхідній стрічці і не пересувається. МП-автомат **допускає** вхідний рядок, якщо він зупиняється в заключному стані q_z, і при цьому магазин порожній; вхідний рядок вважається помилковим, якщо автомат зупиняється при не порожньому магазині, чи серед виразів (1) не знайшлося жодного, ліва частина якого відповідала б поточній конфігурації автомата.

У МП-автоматі, на відміну від кінцевого автомата, нагадування інформації про аналізоване речення виробляється як переходом у новий стан, так і записом слів у магазинну пам'ять. Особливістю мов класів 2 (контекстно-вільних мов), що відрізняє їх від мов класу 3, є наявність вкладених конструкцій з довільною глибиною вкладеності. Прикладами таких конструкцій у мовах програмування є арифметичні вирази загального виду, вкладені умовні оператори, оператори циклу, складені оператори, блоки. Обробку вкладених конструкцій зручно здійснювати за допомогою магазина (стека), при цьому у вершині магазина знаходиться інформація про внутрішніх, оброблюваних у даний момент, елементах цих конструкцій, а в глибині – про зовнішніх, що включають у себе оброблювані внутрішні. Після завершення обробки внутрішнього елемента, інформація про нього виключається з магазину.

В арифметичних виразах такого типу вкладеними конструкціями, що вимагають для своєї обробки магазинну пам'ять, є підвирази, ув'язнені в дужки. Оскільки інші конструкції не можуть бути вкладеними, алфавіт магазинної пам'яті може складатися усього лише з одного символу, запис якого в магазин здійснюється, коли зустрічається черговий підвираз, укладений в дужки.

Приклад 3. Побудувати МП-автомат для розпізнавання арифметичних виразів, синтаксис яких описується наступною граматикою, що породжує:

<вираз> ::= <доданок> | <доданок>+<вираз> | <доданок>–<вираз>

<доданок> ::= <множник> | <доданок>*<множник> |

<доданок>/<множник>

<множник> ::= <имя> | <множник>**<множник> | (<вираз>)

<і’мя> ::= A | B | C | D

Правильними в цій граматиці є, наприклад, вирази: A*(B–D)**C; (((B–D)*A+C)/B)*D-C.

В арифметичних виразах такого типу укладеними конструкціями, потребуючими для своєї обробки магазинної пам’яті, є підвирази, які взяті в лапки. Оскільки інші конструкції не можуть бути вкладеними, алфавіт магазинної пам’яті може складатися лише з одного символу, запис якого у магазин робиться, коли зустрічається черговий підвираз, який взятий у лапки. Функція переходів автомату:

$\delta(q_1,b,s)=(q_2,s)$;

$\delta(q_2,z,s)=(q_1,s)$;

$\delta(q_1,"(",s)=(q_1,ss)$;

$\delta(q_2,")",s)=(q_2,\lambda)$;

$\delta(q_2,\perp,s)=(q_z,\lambda)$.

Тут b – одна з букв A, B, C, D; z – знак операції +, –, *, /, **; s – символ алфавіту магазинної пам’яті; $\perp$ – умовний символ-ознака кінця вхідного рядку. У початковому стані в магазині записан символ s.

Промоделюємо роботу автомата. Візьмемо вираз A*(((B–D)*C–D)/B)+D⊥. В ньому 20 символів, а оскільки керуючий пристрій МП-автомата на кожному кроці зсовується на одну комірку вхідної стрічки, автомат повинен зробити максимум 20 кроків. Наведемо їх в у вигляді таблиці.

Таблиця 4.3 – Моделювання роботи автомату

№ шага	a_i	q_j	стек	№ шага	a_i	q_j	стек
1.	A	q_1	s	11.	C	q_1	sss
2.	*	q_2	s	12.	–	q_2	sss
3.	(	q_1	s	13.	D	q_1	sss
4.	(	q_1	ss	14.	)	q_2	sss
5.	(	q_1	sss	15.	/	q_2	ss
6.	B	q_1	ssss	16.	B	q_1	ss
7.	–	q_2	ssss	17.	)	q_2	ss
8.	D	q_1	ssss	18.	+	q_2	s
9.	)	q_2	ssss	19.	D	q_1	s
10.	*	q_2	sss	20.	⊥	q_2	s

На останньому, 21-м кроці автомат переходить в q_z, очищає магазин і зупиняється.

Розберемо ще один вираз: ((B+D)*A–C⊥

Таблиця 4.4 – Моделювання роботи автомату.

№ кроку	a_i	q_j	крок	№ кроку	a_i	q_j	крок
1.	(	q_1	s	7.	*	q_2	ss
2.	(	q_1	ss	8.	A	q_1	ss
3.	B	q_1	sss	9.	–	q_2	ss
4.	+	q_2	sss	10.	C	q_1	ss
5.	D	q_1	sss	11.	⊥	q_2	ss
6.	)	q_2	sss	12.		q_z	s

Автомат зупинився в заключному стані, але стек не порожній. Значить початкове припущення помилкове (помилка – відсутність закриваючої дужки).

4.2 Варіанти завдань

a) множина парних чисел в алфавіті {0,1};

b) множина непарних чисел в алфавіті {0,1};

c) множина парних чисел в алфавіті {0,1,2,3,4,5,6,7,8,9};

d) множина непарних чисел в алфавіті {0,1,2,3,4,5,6,7,8,9};

e) мови з a), b), c), d), крім чисел з незначними нулями;

f) мова, що складається з речень виду $ab^na|n=0,1,2,\ldots$;

g) мова, що складається з речень виду $a^nb^n|n=0,1,2,\ldots$;

h) мова, що складається з речень виду $a^mb^nc^p|m,n,p=0,1,2,\ldots$;

i) мова, що складається з речень виду $a^mb^nc^m|m,n=0,1,2,\ldots$;

j) мова, що складається з речень виду x^nay^n, x^nby^n $|m,n=0,1,2,\ldots$;

k) мова, що складається з ланцюжків 0 і 1, і утримуючих принаймні три підряд 1 (наприклад 010111101);

l) граматика класу 3 імен мови програмування, що складаються не більш, ніж з 6 букв і (чи) цифр;

m) граматику класу 3, що генерує ту ж мову, що і граматика

<S> ::= <A><B>;

<A> ::= <X> | <Y>;

<X> ::= $\underline{x}$ | $\underline{x}$<X>;

<Y> ::= $\underline{y}$ | $\underline{y}$<Y>;

<B> ::= $\underline{b}$ | $\underline{b}$<B>.

Таблиця 4.5 – Варіанти завдань для побудови кінцевих автоматов-розпізнавачей.

№ варіанта	Граматики	№ варіанта	Граматики
1	a,k	**14**	m,f
2	b,m	**15**	l,e
3	c,e	**16**	k,d
4	d,k	**17**	j,c
5	e,g	**18**	i,b
6	f,j	**19**	h,a
7	g,i	**20**	g,b

8	h,f	**21**	h,c
9	i,e	**22**	i,d
10	j,d	**23**	j,e
11	k,c	**24**	k,f
12	e,b	**25**	l,g
13	m,a		

Таблиця 4.6 - Варіанти завдань для побудови МП-автоматов.

№	Нетермінальний символ	№	Нетермінальний символ
1	<вираз>	**14**	<арифметичний вираз>
2	<оператор опису масиву>	**15**	<оператор опису>
3	<умовний оператор>	**16**	<програма>
4	<вираз>	**17**	<оператор циклу>
5	<описи>	**18**	<умовний оператор>
6	<оператор присвоювання>	**19**	<логічний вираз>
7	<оператор циклу>	**20**	<оператор присвоювання>
8	<вираз>	**21**	<оператор циклу>
9	<програма>	**22**	<оператор присвоювання>
10	<описи>	**23**	<оператор опису масиву>
11	<вираз>	**24**	<програма>
12	<програма>	**25**	<оператор опису типу>
13	<оператор опису масиву>		

4.3 Порядок виконання роботи

1. Побудуйте (чи покажіть, що це неможливо) кінцеві автомати-розпізнавачі для вказаних мов (таблиця 4.5).

2. “Програйте” вручну роботу автомата з задачі 1 для різних вихідних даних.

3. Побудуйте МП-автомати для розпізнавання конструкцій (таблиця 4.6) з прикладу 2 лабораторной роботи № 3 (можливо з обмеженнями).

4. “Програйте” вручну роботу автоматів із задачі 3 для різних вихідних даних (за завданням викладача).

4.4 Зміст звіту

1) Назва лабораторної роботи і завдання.

2) Побудова кінцевого автомату-розпізнавача.

3) Моделювання роботи кінцевого автомата-розпізнавача.

4) Побудова МП-автомату.

5) Моделювання роботи МП-автомата.

4.5 Контрольні питання

1. Для чого призначені граматики-розпізнавачи?

2. Приведіть структуру автоматов-розпізнавачей у загальному виді й в окремих випадках;

3. Яка відповідність має місце між класами мов за Хомським й автоматами-розпізнавачами?

4. Дайте визначення кінцевого автомата;

5. Приведіть приклад граматиці, що породжує, для якої можна побудувати кінцевий автомат-розпізнавач;

6. Дайте визначення МП-автомата;

7. Опишіть функціонування МП-автомата.

Лабораторна робота № 5

ПРОЕКТУВАННЯ СИНТАКСИЧНОГО АНАЛІЗАТОРА ДЛЯ ГРАМАТИКИ ПЕРЕДУВАННЯ

Ціль роботи: Одержання практичних навичок у складанні граматики передування і проектуванні розпізнавача для синтаксичного розбору операторів вхідної мови.

Для виконання лабораторної роботи необхідно знати ідею висхідного аналізу контекстно-вільних мов, визначення граматики передування, як однієї з представниць класу контекстно-вільних граматик, метод передування, а також вміти знаходити матрицю передування, будувати алгоритм розпізнавача в класі граматик передування, оцінювати швидкодію, складність алгоритму й обсяг необхідної пам'яті для розпізнавача.

Необхідні теоретичні зведення знаходяться в [1, с.125-144; 2, с.232-252].

5.1 Теоретичні знання

Синтаксис більшості сучасних мов програмування описується контекстно-вільними граматиками (КВ-граматиками). До цього класу граматик відноситься граматика (простого) передування, що дозволяє конструювати як вручну, так і автоматично досить прості синтаксичні аналізатори (розпізнавачи). Алгоритм розпізнавача для граматики передування використовує лівосторонній висхідний метод діалізу, що дозволяє знаходити в поточній сентенціальній формі основу (тобто саму ліву фразу, що безпосередньо приводиться,) і замінювати її нетермінальним символом. Для того щоб спосіб відшукання основи був однобічним без "тупиків" і "повернень", КВ-граматика повинна мати визначені властивості. Граматика передування, наприклад, відрізняється тим, що в ній:

1) Для кожної упорядкованої пари термінальних і нетермінальних символів (S_i, S_j) виконується не більш ніж одна з трьох відносин передування:

а) $S_i \doteq S_j$, якщо і тільки якщо існує правило

$$U \to xS_iS_jy$$

б) $S_i <\cdot S_j$, якщо і тільки якщо існує правило

$$U \to xS_iDy \text{ і виведення } D \overset{*}{\Rightarrow} S_jy;$$

в) $S_i \cdot > S_j$, якщо і тільки якщо існує правило

$$U \to xCS_iy \text{ і виведення } C \overset{*}{\Rightarrow} zS_i, \text{ чи – правило}$$

$$U \to xCDy \text{ і виведення } C \overset{*}{\Rightarrow} zS_i,\ D \overset{*}{\Rightarrow} S_jw$$

2) Різні правила, що породжують, мають різні праві частини,

де U, C, D - нетермінальні символи,

x, y, z, w - будь-які рядки, можливо, порожні,

$\overset{*}{\Rightarrow}$ - знак породження рядка.

Ці характерні властивості даної граматики дозволяють на кожному кроці синтаксичного аналізу легко виділяти основу шляхом відшукання в рядку $S_i \ldots S_j$ самої лівої групи символів, зв'язаних відносинами передування виду

$$S_{i-1} <\cdot \underbrace{S_i \doteq \ldots \doteq S_j}_{основа} \cdot > S_{j+1}$$

Висловлена ідея пошуку основи може бути використана для побудови лівостороннього розпізнавача у вигляді автомата з магазинною пам'яттю (рис. 1).

Програма роботи керуючого пристрою повинна виконувати наступний алгоритм:

а) символи вхідного рядка по черзі переписуються в стек доти, поки між символом у вершині стека S_i і черговим символом вхідного рядка S_j не виникне відношення $S_i \cdot > S_j$;

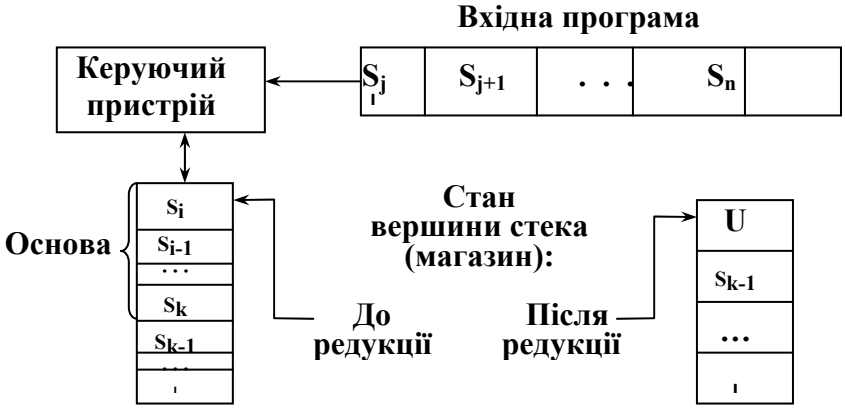


Рисунок 5.1 – Розпізнавач для граматики передування

б) тоді стік проглядається в напрямку від вершини до початку доти, поки між двома черговими символами не з'явиться відношення $S_{k-1} < \cdot S_k$ а з цього випливає, що частина стека від символу S_k до символу у вершині S_i є основа;

в) тепер серед правил, що породжують, знаходиться правило виду $U \to S_k \ldots S_i$ і у стеку основа прямо редукується до нетермінального символу U. При необхідності викликається семантична підпрограма, що обробляє основу, переводячи її на проміжну чи вихідну мову, і описаний вище процес повторюється.

Якщо при аналізі виявиться, що між сусідніми символами S_i і S_j не існують відносини передування, то це свідчить про синтаксичну помилку.

Для знаходження відносин передування необхідно, насамперед, визначити дві допоміжні множини: множина самих лівих L(U) і множина самих правих R(U) символів щодо нетермінального символу U. Ці множини можна визначити рекурсивно:

$$\begin{aligned} L(U) &= \{S \mid \exists(U \to SZ) \vee \exists(U \to U'Z' \wedge S \in L(U'))\}, \\ R(U) &= \{S \mid \exists(U \to ZS) \vee \exists(U \to Z'U' \wedge S \in R(U'))\} \end{aligned} \qquad (1)$$

Дані множини використовуються для виявлення відносин передування за наступними правилами:

$$
\begin{aligned}
&S_i \doteq S_j \leftrightarrow \exists(U \to xS_iS_jY) \\
&S_i <\cdot S_j \leftrightarrow \exists(U \to xS_iDY) \wedge S_j \in L(D) \\
&S_i \cdot > S_j \leftrightarrow \exists(U \to xCS_jY) \wedge S_i \in R(C) \\
&\vee \exists(U \to xCDY) \wedge S_i \in R(C) \wedge S_j \in L(D)
\end{aligned}
\qquad (2)
$$

Відносини передування варто записати у вигляді матриці передування, що представляє собою таблицю з двома входами. Входами в таблицю є попередній S і наступний символи рядка, що приводиться, а в її клітинах записуються відносини передування.

Приклад 1. Побудувати методом передування розпізнавач для обраної мови програмування на прикладі синтаксичного розбору одного чи декількох операторів, описаних граматикою передування.

1. Для заданих операторів вхідної мови складемо множину правильних конструкцій. Синтаксис одержання цих можливих конструкцій запишемо у видгляді граматики передування.

Граматика **G = (T, N, P, A)** ,

де T={+,-,*,/,цф} – множина термінальних символів;

N={<у>,<ав>,<з>,<м>,<ц>} - множина нетермінальних символів;

A = <у> - початковий символ граматики;

P - множина правил, що породжують, а саме

<у> ::= ⊥<ав>⊥

<ав> ::= <с> | + <с> | - <с> | <ав> + <с> | <ав> - <с>

<с> ::= <м> | <с> * <м> | <с> / <м>

<м> ::= <ц> | (<ав>)

<ц> ::= цф | <ц> цф.

2. Побудуємо розпізнавач для синтаксичного аналізу трансльованих операторів. З цією метою на початку для всіх нетермінальних символів граматики за допомогою правил (1) визначимо допоміжні множини L(U) і R(U).

Таблиця 5.1 – Множини L(U) і R(U)

U	**L(U)**	**R(U)**
<y>	⊥	⊥
<ав>	<с>,+,-, <ав>,<м>,<ц>,(,цф	<с>,<м>,<ц>,),цф
<с>	<м>,<с>,<ц>,(,цф	<м>,<ц>,),цф
<м>	<ц>,(,цф	<ц>,),цф
<ц>	цф,<ц>	цф

3. Складемо матрицю передування, що містить відносини передування для кожної упорядкованої пари символів рядка, що приводиться. Алгоритм заповнення матриці визначається правилами (2).

Таблиця 5.2 – Матриця передування.

	⊥	**<ав>**	**<с>**	**<м>**	**<ц>**	**цф**	**z1**	**z2**	**(**	**)**
⊥		=,<	<	<	<	<	<		<	
<ав>	=						=			=
<с>	>						>	=		>
<м>	>						>	>		>
<ц>	>					=	>	>		>
цф	>					>	>	>		>
z1			=,<	<	<	<			<	
z2				=	<	<			<	
(		=,<	<	<	<	<	<		<	
)	>						>	>		>

де z1={- , +} і z2={* , /}

4. Т.к. існують пари символів, для яких відносини передування неоднозначні, то **G** не є граматикою передування. Для усунення неоднозначності потрібно змінити граматику, ми усунемо рекурсивність у визначених <ав> і <с>. Визначимо нову граматику **G'** наступними правилами:

Граматика **G’ = (T, N, P, A)** ,

де T={+,-,*,/,цф} - множина термінальних символів;

N={<у>,<в~><ав>,<з>,<с~>,<м>,<ц>} - множина нетермінальних символів;

A = <у> - початковий символ граматики;

P - множина правил, що породжують, а саме

<у> ::= ⊥<в~>⊥
<в~> ::= <ав>
<ав> ::= <с~> | + <с~> | - <с~> | <ав> + <с~> | <ав> - <с~>
<с~>::=<с>
<с> ::= <м> | <с> * <м> | <с> / <м>
<м> ::= <ц> | (<ав>)
<ц> ::= цф | <ц> цф

5. Складемо остаточні множини L(U) і R (U).

Таблиця 5.3 – Множини L(U) і R(U).

U	**L(U)**	**R(U)**
<у>	⊥	⊥
<в~>	<с~>,+,-, <ав>,<м>,<с>,<ц>,(,цф	<с~>,<м>,<ц>,),цф
<ав>	<с~>,+,-, <ав>,<м>,<с>,<ц>,(,цф	<с~>,<м>,<ц>,),цф
<с~>	<м>,<с>,<ц>,(,цф	<м>,<ц>,),цф
<с>	<м>,<с>,<ц>,(,цф	<м>,<ц>,),цф
<м>	<ц>,(,цф	<ц>,),цф
<ц>	цф,<ц>	цф

6. Складемо матрицю передування.

Таблиця 5.4 – Матриця передування.

	⊥	**<в~>**	**<ав>**	**<с~>**	**<с>**	**<м>**	**<ц>**	**цф**	**z1**	**z2**	**(**	**)**
⊥		=		<	<	<	<	<	<		<	
<в~>	=											
<ав>									=			=
<с~>	>								>			>
<с>												
<м>	>								>	>		>
<ц>	>							=	>	>		>
цф	>							>	>	>		>
z1				=	<	<	<	<			<	
z2						=	<	<			<	
(			=	<	<	<	<	<	<			
)	>								>	>		>

де z1={- , +} і z2={* , /}

Приклад 2. Виконати за допомогою граматики G', таблиці правил, що породжують і матриці передування, побудованих в прикладі 1, синтаксичний розбір наступного умовного вхідного рядка:

⊥(-23)+5*6⊥

Таблиця 5.5 – Таблиця правил граматики, що породжують.

Номер правила	**Правило, що породжує**
1	<у> ::= ⊥<в~>⊥
2	<в~> ::= <ав>
3	<ав> ::= <с~> | + <с~> | - <с~> | <ав> + <с~> | <ав> - <с~>
4	<с~>::=<с>
5	<с> ::= <м> | <с> * <м> | <с> / <м>
6	<м> ::= <ц> | (<ав>)
7	<ц> ::= цф | <ц> цф

Таблиця 5.6 – Тестовий приклад. Трансляція рядка методом передування.

Стек розпізнавача	**Відношення**	**Вхідний рядок**	**Правило, яке використане**
⊥	<	(-23)+5*6 ⊥	-
⊥ (	<	-23)+5*6 ⊥	-
⊥ (-	<	23)+5*6 ⊥	-
⊥ (-2	>	3)+5*6 ⊥	цифра в терм. симв.
⊥ (- цф	>	3)+5*6 ⊥	-
⊥ (- <ц>	=	3)+5*6 ⊥	7
⊥ (-<ц>3	>	)+5*6 ⊥	цифра в терм. симв.
⊥ (-<ц> цф	>	)+5*6 ⊥	7
⊥ (-<ц>	>	)+5*6 ⊥	6
⊥ (-<м>	>	)+5*6 ⊥	5
⊥ (-<з>	>	)+5*6 ⊥	4
⊥ (-<с~>	=	)+5*6 ⊥	-
⊥ (-<с~>)	>	+5*6 ⊥	3
⊥ <ав>	=	+5*6 ⊥	-
⊥ <ав>+	<	5*6 ⊥	-
⊥ <ав>+5	>	*6 ⊥	цифра в терм. симв.
⊥ <ав>+цф	>	*6 ⊥	7

⊥ <ав>+<ц>	>	*6 ⊥	6
⊥ <ав>+<м>	>	*6 ⊥	5
⊥ <ав>+<с>	=	*6 ⊥	-
⊥ <ав>+<с>*	<	6 ⊥	-
⊥ <ав>+<с>*6	>	⊥	цифра в терм. симв.
⊥ <ав>+<с>* цф	>	⊥	7
⊥ <ав>+<с>* <ц>	>	⊥	6
⊥ <ав>+<с>* <м>	>	⊥	5
⊥ <ав>+<с>	>	⊥	4
⊥ <ав>+<с~>	>	⊥	3
⊥ <ав>	>	⊥	2
⊥ <в~>	=	⊥	1
⊥ <у>		⊥	

У реальних мовах програмування число символів граматики досить велике, тому матриця передування виходить дуже великих розмірів. Для скорочення обсягу пам'яті матрицю варто зберігати в "упакованому" вигляді (по 2 біти на елемент) чи використовувати функції передування разом з матрицею сполучуваності символів.

Алгоритм розпізнавача універсальний у класі граматик передування. Якщо, припустим, необхідно змінити вихідну мову, то в трансляторі досить поміняти лише семантичні підпрограми. А якщо змінюється вхідна мова, то необхідно змінити тільки таблицю правил, що породжують, матрицю передування і семантичні підпрограми.

Практичне застосування методу передування ускладнюється неоднозначністю відносин передування, що часто зустрічається в граматиках реальних мов програмування. Загального алгоритму приведення вихідної КВ-граматики до граматики передування немає. На практиці застосовуються приватні алгоритми усунення конфліктів передування [2] чи використовується розширене передування [1], що дозволяє забезпечити однозначність відносини шляхом використання контексту не з двох, а з трьох символів граматики.

5.2 Завдання до лабораторної роботи

Побудувати методом передування розпізнавач для обраної мови програмування на прикладі синтаксичного розбору одного чи декількох операторів, описаних граматикою передування. Варіанти завдань приведені в таблиці 5.7.

Таблиця 5.7 – Варіанти завдань.

№	Нетермінальний символ
1	<вираз>
2	<оператор опису масиву>
3	<умовний оператор>
4	<вираз>
5	<описи>
6	<оператор присвоювання>
7	<оператор циклу>
8	<вираз>
9	<програма>
10	<описи>
11	<вираз>
12	<програма>
13	<оператор опису масиву
14	<арифметичний вираз>
15	<оператор опису>
16	<програма>
17	<оператор циклу>
18	<умовний оператор>
19	<логічний вираз>
20	<оператор присвоювання>
21	<оператор циклу>
22	<оператор присвоювання>
23	<оператор опису масиву>
24	<програма>
25	<оператор опису типу>

5.3 Порядок виконання роботи

1. Для заданих операторів вхідної мови скласти множину правильних конструкцій. Синтаксис одержання цих можливих конструкцій записати у вигляді граматики передування.

2. Побудувати розпізнавач для синтаксичного аналізу трансльованих операторів. З цією метою на початку для всіх нетермінальних символів граматики за допомогою правил (1) визначити допоміжні множини L(U) і R(U).

3. Скласти матрицю передування, що містить відносини передування для кожної упорядкованої пари символів рядка, що приводиться. Алгоритм заповнення матриці визначається правилами (2). Якщо при заповненні матриці для якої-небудь пари символів виникне неоднозначність відносин передування, то це означає, що побудована граматика не є граматикою передування. У цьому випадку варто скористатися розширеним передуванням [1,2] чи виконати еквівалентне перетворення вихідної граматики, замінивши в ній окремі правила виведення, і повернутися до другого пункту.

4. Скласти таблицю правил, що породжують, по граматиці передування, доповнивши її семантичними підпрограмами обробки трансльованих операторів.

5. Побудувати загальну схему транслятора, заснованого на методі передування [2], з урахуванням специфіки варіанта завдання.

6. Виконати ручний синтаксичний розбір декількох операторів (правильних і з помилками) методом передування. Розбір представити у вигляді таблиці, що відбиває динаміку зміни вмісту стеку розпізнавача, вхідного і вихідного рядків у процесі синтаксичного аналізу.

7. Порівняти ефективність методу простого передування з іншими методами аналізу по швидкодії й обсягу необхідної для розпізнавача пам'яті.

5.4 Зміст звіту

1. Формулювання завдання з вказівкою загальних форматів запису операторів вхідної мови.

2. Граматика передування, що породжує задані оператори.

3. Множини L(U) і R(U) у вигляді таблиць.

4. Матриця передування.

5. Схема транслятора з таблицею породжувальних правил.

6. Тестові приклади синтаксичного розбору.

7. Порівняльний аналіз методів розбору.

5.5 Контрольні питання

1. Які граматики і породжувані ними мови називаються контекстно-вільними?

2. У чому складається ідея висхідного аналізу речень вхідної мови?

3. Що називається відносинами передування?

4. Дайте визначення граматики передування.

5. Яка частина вхідного рядка називається основою?

6. Яка головна ідея алгоритму розбору, заснованого на відносинах передування?

7. Як знаходиться матриця передування й усувається неоднозначність відносин передування?

8. З якою метою застосовуються функції передування?

Лабораторна робота №6

ПРОЕКТУВАННЯ РОЗПІЗНАВАЧА ДЛЯ ГРАМАТИКИ З ОПЕРАТОРНИМ ПЕРЕДУВАННЯМ

Ціль роботи: Закріплення навичок конструювання порівняно простих і швидких розпізнавачів методом операторного передування.

При підготовці до заняття необхідно вивчити основні ознаки КВ-граматики з операторним передуванням, знати алгоритм розбору на основі відшукання первинної фрази, вміти будувати матрицю відносин операторного передування і записувати інструментальною мовою програмування алгоритм синтаксичного розбору операторів мови, породженого граматикою з операторним передуванням.

Необхідні теоретичні зведення приведені в [1, с.145-180; 2, с.253-264].

6.1 Теоретичні знання

Метод операторного передування застосуємо тільки до граматик з операторним передуванням. Граматикою з операторним передуванням називають граматику класу 2 за Хомським, у якій:

1) праві частини правил, що породжують, не містять поруч стоячих нетермінальних символів, тобто в граматиці немає правил виду

$U \to xCDy$

де U, C, D - нетермінальні символи,

x, y - будь-які рядки, можливо порожні,

2) для кожної упорядкованої пари термінальних символів T_i и T_j виконується не більш ніж одне з трьох наступних відносин передування:

а) $T_i \doteq T_j$, якщо і тільки якщо існують правила

$$U \to xT_iT_jy \quad \text{або} \quad U \to xT_iCT_jy$$

б) $T_i <\cdot T_j$, якщо і тільки якщо існує правило

$$U \to xT_iCy \text{ і виведення } C \overset{*}{\Rightarrow} T_j z \text{ або } C \overset{*}{\Rightarrow} DT_j z;$$

в) $T_i \cdot > T_j$, якщо і тільки якщо існує правило

$$U \to xCT_j y \text{ і виведення } C \overset{*}{\Rightarrow} zT_i \text{ або } C \overset{*}{\Rightarrow} zT_i D;$$

3) різні правила, що породжують, мають різні праві частини;

Де U, C, D - нетермінальні символи; x, y, z – будь-які рядки, у тому числі порожні. Цією граматикою можна описати будь-яку реальну мову програмування.

Перша вимога дозволяє обмежитися установленням відносин передування тільки для термінальних символів, що значно скорочує розмір матриці передування і підвищує ефективність алгоритму розбору в порівнянні з методом простого передування. Алгоритм розбору відшукує для редукції не основу, а так називану первинну фразу.

Первинна фраза - це фраза, що не містить ніякої іншої фрази, крім самої себе, і утримуюча, принаймні, один термінальний символ. Алгоритм розбору майже такий ж як у методі простого передування (див. лабораторну роботу 5). Різниця полягає в тому, що в методі операторного передування, по-перше, використовуються тільки відносини термінальних символів і, по-друге, у знайденому для редукції правилі U права частина може відрізнятися від виділеної первинної фрази нетермінальними символами, що не гарантує однозначності даного розбору. Для усунення неоднозначності розбору варто використовувати семантичні підпрограми. Один зі станів стека і вхідний рядок у процесі синтаксичного розбору методом операторного передування показано на рис. 6.1.

Для знаходження відносин операторного передування необхідно попередньо визначити множину самих лівих L.(U) і множину самих правих R.(U) термінальних символів щодо нетермінального символу U.

Алгоритм відшукання цих множин визначається наступними формулами:

$$L_t(U)=\{t\mid\exists(U\to tz)\vee\exists(U\to Ctz)\vee(U'\in L(U)\wedge t\in L_t(U'))\} \quad (1)$$

$$R_t(U)=\{t\mid\exists(U\to zt)\vee\exists(U\to ztC)\vee(U'\in R(U)\wedge t\in R_t(U'))\} \quad (2)$$

де t – термінальний символ.

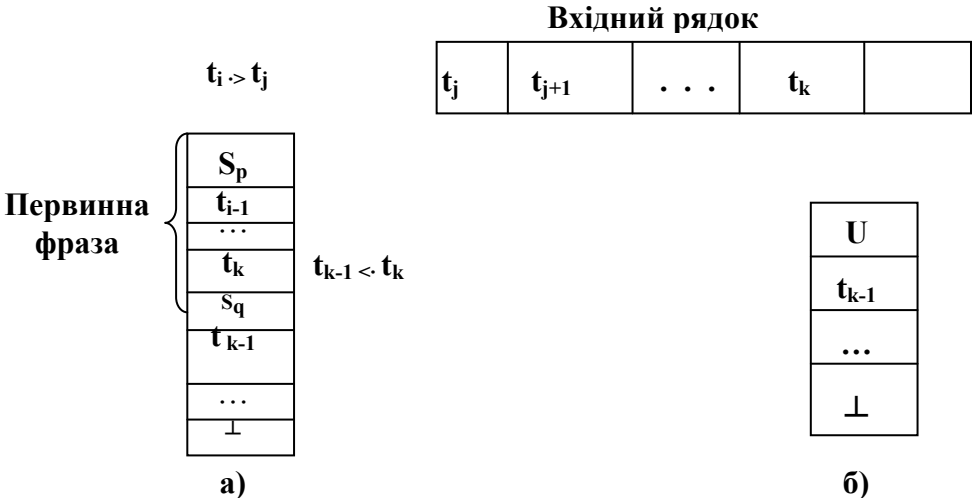


Рисунок 6.1 - Стан вхідного рядка і стека: а) до редукції; б) після редукції.

За допомогою множин (1-2) і правил граматики, що породжують, тепер можна легко знайти відносини операторного передування за наступними правилами:

$$t_i \doteq t_j \leftrightarrow \exists(U\to xt_it_jy)\vee\exists(U\to xt_iCt_jy); \quad (3)$$

$$t_i <\cdot t_j \leftrightarrow \exists(U\to xt_iCy)\&\, t_j\in L_t(C); \quad (4)$$

$$t_i \cdot> t_j \leftrightarrow \exists(U\to xCt_jy)\&\, t_i\in R_t(C). \quad (5)$$

Приклад 1. Побудувати матрицю передування для граматики G_1, яка має наступний вигляд:

<П>::=⊥<В>⊥

<В>::=<Т> | <В>+<Т>

<Т>::=<М> | <Т>×<М>

<М>::= и | (<В>)

Виконання цього прикладу будемо здійснювати на основі наступного алгоритму

Алгоритм побудови матриці операторного передування

1. Для кожного нетермінального символу U граматики G_1 будуються множини $L_t(U)$ та $R_t(U)$. Для цього

а) відшукуються множини L(U) та R(U).

б) для кожного нетермінального символу U :

- відшукуються правила граматики G_1 з правими частинами виду *tz* або *Ctz*. Термінальні символи *t* фіксуються в якості елементів множини $L_t(U)$;

- відшукуються правила з правими частинами виду *zt* та *ztC*. Термінальні символи *t* фіксуються в якості елементів множини $R_t(U)$.

Таблиця 6.1 - Таблиця множин $L_t(U)$ и $R_t(U)$.

U	**$L_t(U)$**	**$R_t(U)$**
<П>	⊥	⊥
<В>	+	+
<Т>	×	×
<М>	и, (	и,)

в) для кожного нетермінального символу U :

- проглядається множина $L_t(U)$ та відшукуються вхідні в неї нетермінальні символи U’, U’’, … . Далі множина $L_t(U)$ доповнюється символами, які входять у $L_t(U')$, $L_t(U'')$, … та які не входять у $L_t(U)$.

- проглядається множина $R_t(U)$ та відшукуються вхідні в неї нетермінальні символи U’, U’’, … . Далі множина $R_t(U)$ доповнюється символами, які входять у $R_t(U')$, $R_t(U'')$, … та які не входять у $R_t(U)$.

В результаті отримуємо таблицю 6.2.

Таблиця 6.2 - Таблиця множин $L_t(U)$ и $R_t(U)$.

U	**$L_t(U)$**	**$R_t(U)$**
<П>	⊥	⊥
<В>	+, × , и, (	+, ×, и ,)
<Т>	×, и, (	×, и,)
<М>	и, (	и,)

2. Складається матриця операторного передування. Для цього проглядаються праві частини правил граматики G_1, що породжують.

- за правилом (3) визначаються відносини $\dot{=}$.

- за правилом (4) з використанням таблиці 6.2 визначаються відносини $<\bullet$.

- за правилом (5) з використанням таблиці 6.2 визначаються відносини $\bullet>$.

Отримані результати зведемо до таблиці 6.3

Таблиця 6.3 - Матриця операторного передування для граматики G_1.

t_i / t_j	(	**и**	×	+	)	⊥
)			$\bullet>$	$\bullet>$	$\bullet>$	$\bullet>$
и			$\bullet>$	$\bullet>$	$\bullet>$	$\bullet>$
×	$<\bullet$	$<\bullet$	$\bullet>$	$\bullet>$	$\bullet>$	$\bullet>$
+	$<\bullet$	$<\bullet$	$<\bullet$	$\bullet>$	$\bullet>$	$\bullet>$
(	$<\bullet$	$<\bullet$	$<\bullet$	$<\bullet$	$\dot{=}$	
⊥	$<\bullet$	$<\bullet$	$<\bullet$	$<\bullet$		$\dot{=}$

Одержана матриця операторного передування має вигляд

$$\begin{bmatrix} & \dot{=} & \bullet> \\ <\bullet & & \end{bmatrix},$$

отже, існують функції операторного передування, які дозволяють зберігати матрицю в упакованому вигляді.

6.2 Завдання до лабораторної роботи

Побудувати методом операторного передування розпізнавач для обраної мови програмування на прикладі синтаксичного розбору одного чи декількох операторів, описаних граматикою операторного передування.

Варіанти завдань обрати з лабораторної роботи 5.

6.3 Порядок виконання роботи

1. Породження правильних конструкцій заданих операторів вхідної мови програмування (по лабораторній роботі 5) описати граматикою з операторним передуванням. При складанні набору синтаксичних правил варто скористатися визначенням граматики з операторним передуванням.

2. Для кожного нетермінального символу U складеної граматики G визначаються множини $L_t(U)$ і $R_t(U)$. Для цього застосовується наступний алгоритм, визначений формулами (1):

2.1 Знаходяться множини L(U) і R(U) за правилами, описаними у лабораторній роботі 5.

2.2 Для кожного нетермінального символу U:

а) відшукуються правила граматики із правими частинами виду *tz* і *Ctz*, де C - нетермінальний символ, а *z* – будь-який рядок, бути може, порожній. Термінальні символи *t* фіксуються як елементи множини $L_t(U)$;

б) відшукуються правила з правими частинами виду *zt* і *ztC*. Термінальні символи *t* фіксуються як елементи множини $R_t(U)$.

2.3. Для кожного нетермінального символу U:

а) проглядається множина $L_t(U)$ і відшукуються вхідні в нього нетермінальні символи U', U'',... . Потім множина $L_t(U)$ доповнюється символами, що входять у $L_t(U')$, $L_t(U'')$... і не вхідними в $L_t(U)$;

б) аналогічно проглядається множина $R_t(U)$ і відшукуються вхідні в неї нетермінальні символи U, U',... . Потім множина $R_t(U)$ доповнюється символами, що входять у $R_t(U')$, $R(U_t)$... і не вхідними в $R_t(U)$.

3. Побудувати матрицю операторного передування.

Для цього необхідно переглянути праві частини правил граматики, що породжує. За правилом (3) визначаються відносини $\doteq$. За правилом (4) з використанням множини $L_t(U)$ визначаються відносини $<\bullet$. За правилом (5) і множиною $R_t(U)$ визначаються відносини $\bullet>$.

4. Скласти таблицю правил, що породжують, для розпізнавача стосовно до граматики G. З таблиці виключити правила, що містять у правій частині

тільки нетермінальні символи, а в правих частинах всіх інших правил нетермінальні символи замінити символом N. Крім того, таблицю доповнити семантичними підпрограмами обробки трансльованих операторів.

5. Написати інструментальною мовою програмування програму, що реалізує алгоритм роботи розпізнавача у вигляді автомата з магазинною пам'яттю.

6. Виконати на ЕОМ чи вручну моделювання отриманого автомата як однобічного розпізнавача, що аналізує вихідні оператори вхідної мови.

7. Оцінити ефективність методу операторного передування.

6.4 Зміст звіту

1. Формулювання завдання на проектування розпізнавача.

2. Граматика з операторним передуванням, що описує правильні конструкції заданих операторів.

3. Таблиці множин L(U) і R(U), $L_t(U)$ і $R_t(U)$.

4. Матриця операторного передування і таблиця правил, що породжують.

5. Програма роботи розпізнавача і результати моделювання синтаксичного розбору заданих операторів.

6. Оцінки по швидкодії (кількість затрачуваних кроків на розбір) і по займаній пам'яті (кількість байт під бази даних) у порівнянні з методом простого передування, табличним методом і методом, заснованим на стеку з пріоритетами.

6.5. Контрольні питання

1. Чим відрізняється граматика з операторним передуванням від граматики простого передування?

2. Що називається первинною фразою і чим вона відрізняється від основи (відмінність показати на конкретному дереві розбору)?

3. З яких основних елементів складається розпізнавач для граматики з операторним передуванням, і яка ідея алгоритму синтаксичного розбору?

4. Як виглядає таблиця правил, що породжують, використовувана в розпізнавачі для граматики з операторним передуванням?

5. Чим компенсується неповнота синтаксичного розбору в класі граматик з операторним передуванням?

6. Які достоїнства і недоліки методів, заснованих на передуванні?

Список рекомендованої літератури

1. Грис Д. Конструирование компиляторов для ЦВМ.- М.: Мир, 1975. - 544 с.
2. Лебедев В.Н. Введение в системы программирования. - М.: Статистика, 1975. - 312 с.
3. Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов, - М.: Мир, 1979.- 654 с.
4. Хантер Р. Проектирование и конструирование компиляторов - М.: Финансы и статистика, 1984.-232 с.
5. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. – М.: Мир, 1978.- т.1 - 612 с., т.2 - 487 с.
6. Касьянов В.Н., Поттосин И.В. Методы построения трансляторов - Новосибирск: Наука, 1986.-344 с.
7. Пратт Т. Языки программирования: разработка и реализация. - М.: Мир, 1979. - 574 с.
8. Болье Л. Методы построения компиляторов.//Языки программирования, - М.: Мир, 1972, с. 87-277.
9. Донован Дж. Системное программирование. - М.: Мир, 1975. - 540 с.
10. Васючкова Т.Л. и др. Языки программирования ДОС ЕС ЭВМ. Краткий справочник. - М.: Статистика, 1977.-152 с.
11. Йенсен К., Вирт Н. Паскаль. Руководство для пользователя и описание языка. - М.: Финансы и статистика, 1982.-151 с.
12. Хопгуд Ф. Методы компиляции. – М.: Мир, 1972. – 160 с.
13. Фостер Дж. Автоматический синтаксический анализатор. – М.: Мир, 1975. – 72с.
14. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. – М.: Мир, 1979. – 536 с.
15. Ингерман П. Синтаксически ориентированный транслятор. – М.: Мир, 1969. – 176 с.

16. Савинков В.М., Сидоренко Л.А. Синтаксический транслятор анализирующего типа. – М.: Статистика, 1972. – 88 с.
17. Ахо А., Ренди С., Ульман Дж. Компиляция. – М.: Мир, 2002. – 834 с.
18. Василеску Ю. Прикладное программирование на языке Ада. – М.: Мир, 1990. – 352 с.
19. Павловская Т.А. С/С++ Программирование на языке высокого уровня: учебник для вузов. – С.-П.: Питер, 2005. – 272 с.
20. Прата Стивен. Язык программирования С++. Лекции и упражнения. – К.: ДиаСофт, 2005. – 1104 с.

ЗМІСТ

Навчальне видання

Методичні вказівки і завдання

до лабораторних робіт за курсом

“Теорія синтаксичного аналізу і компіляції”

(для студентів напряму підготовки 6.0501

“Програмна інженерія”)

Укладач: Ольга Анатоліївна Дмитрієва