НАЦIОНАЛЬНИЙ ТЕХНIЧНИЙ УНIВЕРСИТЕТ УКРАНИ

“КИВСЬКИЙ ПОЛIТЕХНIЧНИЙ IНСТИТУТ”

КАФЕДРА АВТОМАТИЗАЦІЇ ПРОЕКТУВАННЯ ЕНЕРГЕТИЧНИХ ПРОЦЕСІВ ТА СИСТЕМ ТЕФ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання курсової роботи з

кредитного модуля

“ ЛІНГВІСТИЧНЕ ЗАБЕЗПЕЧЕННЯ САПР-2 ”, ЗП-03/2

для напряму підготовки 6.050101 “Kомп’ютерні науки”

спецiальностi 6.080402 “ Інформаційні технології проектування ”

Методичні вказівки рекомендовано

кафедрою АПЕПС

Протокол N\_\_\_\_\_\_\_\_\_ вiд\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2011р.

Завідувач кафедри АПЕПС

\_\_\_\_\_\_\_\_\_ С.О. Лук’яненко

Кив 2011

НАЦIОНАЛЬНИЙ ТЕХНIЧНИЙ УНIВЕРСИТЕТ УКРАНИ

“КИВСЬКИЙ ПОЛIТЕХНIЧНИЙ IНСТИТУТ”

КАФЕДРА АВТОМАТИЗАЦІЇ ПРОЕКТУВАННЯ ЕНЕРГЕТИЧНИХ ПРОЦЕСІВ ТА СИСТЕМ ТЕФ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання курсової роботи з

***КРЕДИТНОГО МОДУЛЯ***

“ОСНОВИ РОЗРОБКИ ТРАНСЛЯТОРІВ-2”, ЗП-03/2

для напряму підготовки 6.050103 “Програмна інженерія”

спецiальності 6.080403 “Програмне забезпечення систем”

Програму рекомендовано

кафедрою АПЕПС

Протокол N\_\_\_\_\_\_\_\_\_ вiд\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2011р.

Завідувач кафедри АПЕПС

\_\_\_\_\_\_\_\_\_ С.О. Лук’яненко

Кив 2011

ЗМІСТ

[Вступ 4](#_Toc334182095)

[1 Мета та завдання 5](#_Toc334182096)

[2 Завдання на курсову роботу 5](#_Toc334182097)

[3 Склад, обсяг і структура курсової роботи 8](#_Toc334182098)

[4 Вказівки до виконання розділів курсової роботи 10](#_Toc334182099)

[5 Рекомендації до виконання розрахунково-пояснювальної записки 54](#_Toc334182100)

[Вказівки до порядку захисту роботи 54](#_Toc334182101)

[Список рекомендованої літератури 56](#_Toc334182102)

# Вступ

Дисципліни, що надають знання, уміння та навички щодо розробки трансляторів, є важливою складовою у підготовці фахівців в галузі “Інформатика та обчислювальна техніка”, тому що транслятор (компілятор) є невід’ємною частиною програмного забезпечення комп’ютерів. Необхідність побудови компіляторів визначається появою нових архітектур ЕОМ, для яких потрібні компілятори з розповсюджених мов програмування, а також розробкою та реалізацією спеціалізованих мов (вхідних мов систем моделювання, мов для організації специфічних обчислень у банківських системах, мов систем автоматизованого проектування, тощо).

Дисципліни „Основи розробки трансляторів” та „Лінгвістичне забезпечення САПР” включені до циклу “Професійної підготовки” варіативної частини навчальних планів підготовки бакалаврів з напряму „Програмна інженерія” (спеціальність „Програмне забезпечення систем”) та „Комп’ютерні науки” (спеціальність „Інформаційні технології проектування”) відповідно. [1]

У структурно-логічній схемі навчання зазначені дисципліни рекомендується розміщувати тоді, коли студенти вже вивчили основи програмування та алгоритмічні мови, а також основи дискретної математики та набули певного досвіду у програмуванні і можуть виконувати складні лабораторні роботи. З іншого боку, викладений матеріал може бути використаний при вивченні дисциплін “Програмне забезпечення інтелектуальних систем”, “Моделювання складних процесів і систем”,“Основи САПР складних об’єктів і систем”, “Основи проектування систем штучного інтелекту”, які подаються в наступних семестрах [1].

# 1 Мета та завдання

Метою курсової роботи є отримання студентами практичних навичок у розробці транслятора як закінченого програмного продукту.

Курсова робота спираться на проведений цикл лабораторних робiт та становить собою транслятор для задано мови.

Студент повинен розробити граматику заданої мови, визначити структуру свого транслятора, описати та реалізувати основні функціональні блоки транслятора, розробити інтерфейс користувача, провести тестування свого програмного продукту.

# 2 Завдання на курсову роботу

Розробити та реалізувати транслятор з заданої мови, що включає наступні блоки:

Лексичний аналізатор.

Синтаксичний аналізатор.

Побудова проміжної форми подання програми – польського інверсного запису (ПОЛІЗ).

Виконання ПОЛІЗ.

*Мови, що використовуються:* Мова програмування повинна містити в собі оператор присвоювання, оператори вводу та виводу, структура яких пропонується самим студентом, а також оператори умовного переходу та циклу, що надаються згідно з варіантом (таб. 2.1).

Таблиця 2.1 — Оператори умовного переходу та циклу

|  |  |
| --- | --- |
| № | Оператори умовного переходу та циклу |
| 1 | while <логічний вираз>do <оператор>;  if <логічний вираз> then<список операторів>else <оператор>; |
| 2 | for <ід>=<вираз>to <вираз>step <вираз>  <список операторів>  next  if <логічний вираз> then  <список операторів>  else  <список операторів>  end if |
| 3 | for <ід>=<вираз>to <вираз>step <вираз>  <список операторів>  next  if <логічний вираз>  <список операторів>  else  <список операторів>  endif |
| 4 | do while <логічний вираз>  <список операторів>  enddo  if <логічний вираз>  <список операторів>  else  <список операторів>  endif |
| 5 | do while (<логічний вираз>) ; <список операторів> end;  if <логічний вираз> then <оператор>else <оператор>; |
| 6 | ddo <ід>=<вираз> by <вираз>to <вираз>;<оператор>;  if <відношення> goto <мітка> ; |
| 7 | do do <ід>=<вираз> to<вираз> by <вираз> while (<логічний вираз>) <список операторів> end ;  if <логічний вираз> then <оператор> ; |
| 8 | do<ід>=<вираз> to <вираз>  <список операторів>  next  if <логічний вираз> then goto<мітка> |

*Особливості арифметичного виразу*. В арифметичному виразі використовуються операції та константи, зазначені в таб. 2.2.

Таблиця 2.2 — Особливості арифметичного виразу

|  |  |  |
| --- | --- | --- |
| № | Операції та дужки | Вид констант |
| 1 | + - \* / , унарний -, ( ), | костанти з фіксованою точкою |
| 2 | + - \* / , ( ) | константи з плаваючою точкою |
| 3 | + - \* / ↑ , ( ), унарний - | цілі константи |

*Методи лексичного аналізу*. Для проведення лексичного аналізу використовуються алгоритми, зазначені в таб. 2.3.

Таблиця 2.3 — Алгоритми лексичного аналізу

|  |  |
| --- | --- |
| № | Алгоритм лексичного аналізатора |
| 1 | Перегляд до роздільника |
| 2 | Діаграма станів |
| 3 | Скінчений автомата у формі списка станів |

*Методи синтаксичного аналізу***.** Для проведення синтаксичного аналізу використовуються алгоритми, зазначені в таб. 2.4.

Таблиця 2.4 — Алгоритми синтаксичного аналізу

|  |  |
| --- | --- |
| № | Алгоритм синтаксичного аналізатора |
| 1 | Рекурсивний спуск |
| 2 | Синтаксичний аналізатор на базі магазинного автомату |
| 3 | Нисхідний розбір на базі відношень простого передування |

*Індивідуальні завдання.* Таким чином, індивідуальне завдання визначається комбінацією елементів, зазначених в таб. 2.1-2.4, і наведене в таб. 2.5.

Таблиця 2.5 — Індивідуальні завдання

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Варіанти | | | | | | | | 1 | | 2 | 3 | | 4 | | 5 | 6 | | 7 | 8 | | 9 | | 10 | 11 | |
| 2 | Оператори циклу та умовного переходу | | | | | | | | 1 | | 2 | 3 | | 4 | | 5 | 6 | | 7 | 8 | | 1 | | 2 | 3 | |
| 3 | Особливості арифметичного виразу | | | | | | | | 1 | | 2 | 3 | | 1 | | 2 | 3 | | 1 | 2 | | 3 | | 1 | 2 | |
| 4 | Алгоритм лексичного аналізатора | | | | | | | | 1 | | 1 | 1 | | 2 | | 2 | 2 | | 3 | 3 | | 3 | | 1 | 1 | |
| 5 | Алгоритм синтаксичного аналізатора | | | | | | | | 1 | | 2 | 3 | | 1 | | 2 | 3 | | 1 | 2 | | 3 | | 1 | 2 | |
| Продовження таблиці 2.5 | | | | | | | | |  | |  |  | |  | |  |  | |  |  | |  | |  |  | |
| 1 | | 12 | 13 | 14 | 15 | 16 | 17 | 18 | | 19 | | | 20 | | 21 | | | 22 | | | 23 | | 24 | | | 25 | |
| 2 | | 4 | 5 | 6 | 7 | 8 | 1 | 2 | | 3 | | | 4 | | 5 | | | 6 | | | 7 | | 8 | | | 1 | |
| 3 | | 3 | 1 | 2 | 3 | 1 | 2 | 3 | | 1 | | | 2 | | 3 | | | 1 | | | 2 | | 3 | | | 1 | |
| 4 | | 1 | 2 | 2 | 2 | 3 | 3 | 3 | | 1 | | | 1 | | 1 | | | 2 | | | 2 | | 2 | | | 3 | |
| 5 | | 3 | 1 | 2 | 3 | 1 | 2 | 3 | | 1 | | | 2 | | 3 | | | 1 | | | 2 | | 3 | | | 1 | |

# 3 Склад, обсяг і структура курсової роботи

Курсова робота складається з програми, що реалізує транслятор з заданої мови, та пояснювальної записки.

Записка повинна містити в собi основну частину та додатки.

Основна частина складається з наступних розділів:

1. анотація:
2. вступ:
3. індивідуальне завдання:
4. граматика мови (у відповідності до iндивiдуального завдання);
5. структура транслятора (послiдовність роботи блокiв, число проходiв та iн.);
6. лексичний аналiзатор (дати опис алгоритму з поданням перелiку лексем, та всіх допоміжних даних залежно від алгоритму реалізації, а саме: перелік роздільників для алгоритму 1; діаграма станів та таблиця класів символів для алгоритму 2; діаграма станів та список переходів кінцевого автомату для алгоритму 3; навести перелік помилок, які виявляє лексичний аналізатор; навести приклад роботи лексичного аналізатора при наявності помилок у тестовій програмі та при їх відсутності);
7. синтаксичний аналiзатор (дати опис алгоритму, надати потрібні перетворення граматики залежно від алгоритму, а також додаткову інформацію, а саме: дерево процедур, що використовуються, для алгоритму 1; діаграми станів та списки переходів всіх підавтоматів для алгоритму 2; таблицю передувань та візуалізацію процесу розбору для алгоритму 3; навести перелік помилок, які виявляє синтаксичний аналізатор; навести приклад роботи синтаксичного аналізатора при наявності помилок у тестовій програмі та при їх відсутності );
8. промiжна форма подання програми - ПОЛIЗ ( навести блок-схеми реалiзацi складних операторiв, особливості трансляцi службових слiв та роботи стека, особливостці реалiзацi операторiв вводу-виводу. Навести приклад побудови ПОЛІЗ та пояснити отриманий результат);
9. виконання отримано ПОЛIЗ-програми (навести особливостi виконання команд ПОЛIЗ, таблицю позначок, результат виконання тестової програми; для контролю обчислити цю ж програму на будь-якiй алгоритмiчнiй мовi та порiвняти результати;
10. інструкція користувача по роботі з транслятором;
11. висновок;
12. список використаних джерел.

Записка повинна включати наступні додатки:

Додаток 1. Текст програми.

Додаток 2. Опис програми.

4 Вказівки до виконання розділів курсової роботи

## 4.1 Основні блоки і структура транслятора

До складу будь-якого транслятора входять три основні компоненти:

1. лексичний аналізатор,
2. синтаксичний аналізатор,
3. генератор кодів машинних команд.

На фазі лексичного аналізу (ЛА) вхідна програма, яка являє собою потік символів, розбивається налексеми - слова у відповідності з визначеннями мови. Основнимформалізмом, що лежить в основі реалізації лексичних аналізаторів, є кінцеві автомати і регулярні вирази. Лексичний аналізатор може працювати в двох основних режимах: або якпідпрограма, що викликається синтаксичним аналізатором за черговоюлексемою, або як повний прохід, результатом якого є файллексем.

В процесі виділеннялексем ЛА може як самостійно будувати таблиці імен і констант, так і видавати значення для кожноїлексеми при черговому зверненні до нього. В цьому випадку таблиця імен будується в наступних фазах (наприклад, в процесі синтаксичного аналізу).

На етапі ЛА виявляються деякі (найпростіші) помилки аналізу (неприпустимі символи, невірний запис чисел, ідентифікаторів, необявлені ідентифікатори, відсутні мітки, на які є посилання, та ін.).

Основна задача синтаксичного аналізу - розбір структури програми. Як правило, під структурою розуміється дерево, відповідне розбору в контекстно-вільній граматиці мови. Результатом синтаксичного аналізу є синтаксичне дерево зі засланнями на таблицю імен. В процесі синтаксичного аналізу також виявляються помилки, пов'язані зі структурою програми.

Після цього програма може бути переведена у внутрішнє подання. Це робиться з метоюоптимізації і/або зручності генерації коду.. Як внутрішнє подання може використовуватисяпрефіксний або постфіксний запис, орієнтований граф, трійки, четвірки та інші.

Нарешті, генерація коду - остання фаза трансляції. Результатом її є або асемблерний модуль, або об’єктний (або завантажувальний ) модуль. Для генерації коду розроблені різноманітні засоби, такі як таблиці рішень, зіставлення зразків, включаючє динамічне програмування, різноманітні синтаксичні засоби.

Звичайно, ті або інші фази транслятора можуть або бути відсутні зовсім, або об'єднуватися.

При розробці курсової роботи треба реалізувати наступні фази трансляції:

- лексичний аналіз;

- синтаксичний аналіз;

- побудова внутрішнього подання (постфіксний запис);

- виконання (інтерпретація) внутрішнього подання.

## 4.2 Граматики і мови

Багато концепцій, пов’язаних з організацією компіляторів, засновані на теорії формальних граматик, основоположником якої є Н.Хомський.

Граматика мови програмування є формальний опис його синтаксису або форми, в якій записані окремі речення програми або вся програма. Граматика не описує семантику, тобто зміст різноманітних речень.

Граматика, за Хомським - це деякий алгоритм, що здатний визначити, чи належить даний ланцюжок до даної мови.

Неформально можна визначити мову як підмножину всіх правильних речень із ″слів″ або символів деякого основного словника або алфавіту.

Для завдання формальних мов (мов програмування) використовуються породжуючі граматики.

Породжуюча граматика - це упорядкована четвірка:

G=<VT,VN,σ,P>,

де **VT**- скінченна множина, що називається термінальним словником;

**VN**- скінченна множина, що називається нетермінальним словником;

VT ∩VN =∅;

**σ**- аксіома граматики, **∈ VN**

**P** - множина правил виведення спеціального вигляду:

P = { p:α→β }, α,β ∈ (VT ∪VN)\*.

Хомським проведена класифікація граматик, що базується на вигляді лівої та правої частин правил підстановки і містить 4 типи граматик. Залежно від типу граматики використовуються різні алгоритми граматичного розбору, тобто синтаксичного аналізу.

Нас будуть цікавити граматики типу 2, що мають наступні обмеження:

- ліва частина правил підстановки – це довільний нетермінальний символ, тобто граматика містить правила вигляду:

A→β, де A∈VNβ ∈(VN∪VT.

А саме: ліва частина правила – нетермінальний символ, права – ланцюжок термінальних та нетермінальних символів.

Це контекстно-вільні (КВ) граматики, якими описується переважна більшість мов програмування. Для граматичного розбору КВ-граматики може бути використаний автомат з магазинною пам’яттю.

КВ-граматики включають в себе граматики типу 3, що мають наступні обмеження:

- ліва частина - довільний нетермінальний символ, права - один термінальний і один нетермінальний, кожний з яких може бути відстунім, граматика містить правила вигляду:

А→βВ де A∈VN β∈(VT∪Λ), B∈(VN∪Λ)

Ці граматики називають автоматними, оскільки речення мов, що породжуються ними, розпізнаються кінцевими автоматами.

Множина всіх речень - мова, що породжується даною граматикою.

Процес породження зводиться до наступного. В множині правил P граматики G обирається правило, ліва частина якого являє собою аксіому, і розглядається його права частина, тобто деякий ланцюжок термінальних і нетермінальних символів. В отриманому ланцюжку обирається нетермінал і замінюється правою частиною правила, що його визначає. Процес підстановок відбувається доти, доки ланцюжок не буде складатися тільки з термінальних символів. Порядок підстановoк в породжуючих граматиках довільний.

Ланцюжки, що приймають участь у такому виводі, називаються сентенціальними формами.

*Рекурсивні правила*Правило називається рекурсивним, якщо воно має наступний вигляд:



1. правило називається ліворекурсивним, якщо , тобто ;
2. правило називається праворекурсивним, якщо , тобто ;
3. правило з самовставленням, якщо  и , тобто .

Для того, щоб мова була нескінченою, її граматика повинна містити принаймні одне рекурсивне правило.

Це був випадок прямої рекурсії. Існує поняття непрямої рекурсії. Наприклад,  - рекурсивне визначення, якщо



КВ - граматика є автоматною, якщо не містить правил з самовставленням.

Приступаючи до розробки компілятора з будь-якої мови, передусім необхідно написати його граматику.

Іноді при поданні граматики КВ-мови задається множина правил без явного визначення термінального і нетермінального словників. В таких випадках передбачається, що множини термінальних та нетермінальних символів, що зустрічаються в правилах, і складають словники **V**t, **V**n, а аксіомою є ліва частина першого правила.

Для подання граматик КВ-мов як правило, використовується ***форма Бекуса-Наура*** або БНФ.

В БНФ нетермінальні символи позначаються словами та беруться в дужки <>, всі інші символи вважаються термінальними. Також використовуються 2 метасимволи:

1. ::= — дорівнює по визначенню;
2. | — дозволяє об’єднувати правила, що мають однакову ліву частину.

Аксіомою граматики зазвичай є <**програма**>. Тобто, при виводі з <**програми**> ланцюжка, що складається тільки з терміналів, отримаємо синтаксично правильну програму. Нетермінальні символи є допоміжними, тому вибір позначень для них є несуттєвий, але в той же час кожний нетермінал визначає деяку структурну одиницю або синтаксичну категорію для всього рядка, що породжуеться. Тому при запису граматики (тобто синтаксису) в якості нетермінальних символів використовуються слова, що підказують неформальний зміст конструкції. Таким чином, нетермінальний словник — це ***термінологічний словник*** для даної мови.

Наведемо в БНФ граматику мови mini-Паскаль, яку будемо використовувати як приклад у розробці різноманітних блоків компілятора.

**Приклад 1.** Граматика мови mini-Паскаль.

1. **<програма> ::= program <ім’я пр.> var <список об’яв> begin <список операторів> END.**
2. **<ім’я пр.>::=<ід>**
3. **<список об’яв>::=<об’ява>|<список об’яв>;<об’ява>**
4. **<об’ява>::=<список ідент.> : <тип>**
5. **<список ідент.> ::= <ід>|<список ідент.>,<ід>**
6. **<тип> ::= integer**
7. **<список операторів>::=<оператор>|<список ператорів>; <оператор>**
8. **<оператор> ::= <присвоювання>|<введення>|<виведення>|<цикл>**
9. **<присвоювання> ::= <ід> := <врж>**
10. **<врж> ::= <терм>|<врж> + <терм>|<врж> - <терм>**
11. **<терм> ::= <множ>|<терм> \* <множ>|<терм> DIV <множ>**
12. **<множ> ::= <ід>|<конст>|(<врж>)**
13. **<введення> ::= read (<список ідент>)**
14. **<виведення> ::= write (<список ідент>)**
15. **<цикл> ::= for <індексний вираз> do <дія>**
16. **<індексний вираз> ::= <ід> := <врж> to <врж>**
17. **<дія> ::= <оператор>|begin <список операторів> end**
18. **<ід> ::= <буква>|<ід> <буква>|<ід> <цифра>**
19. **<конст> ::= <цифра>|<конст> <цифра>**
20. **<буква>::=a|b|c|... |z**
21. **цифра>::= 0|1|2|3|4|5|6|7|8|9**

Примітки:

1. В цій граматиці визначення 3, 5, 7, 10, 11, 18, 19 є ліворекурсивними. Для визначень 10 і 8 характерна непряма рекурсія з самовставленням. Справді,

**<врж> → <терм> → <множ> → (<врж>)**

або

**<оператор> → <цикл> → for <індексний вираз> do <дія> →**

**for <індексний вираз> do begin <список операторів end →**

**for <індексний вираз> do begin <оператор> end**

1. В граматиці арифметичного виразу повинен бути врахований пріоритет виконання операцій. Цього можна досягти, якщо рознести операції різного пріоритету по різних правилах, як у наведеній граматиці .

## 4.3 Лексичний аналіз

Основна задача лексичного аналізу - розбити вхідний текст, який складається з послідовності одиничних символів, на послідовність слів, або лексем, тобто виділити ці слова з безперервної послідовності символів. Всі символи вхідної послідовності з цієї точки зору поділяються на символи, що належать будь-яким лексемам, і символи, що поділять лексеми (роздільники). В деяких випадках між лексемами може і не бути роздільників.

Звичайно всі лексеми діляться на класи. Прикладами таких класів є числа (цілі, восьмирічні, шістнадцятирічні, дійсні і т. д.), ідентифікатори, рядки. Окремо виділяються ключові слова і символи пунктуації (інколи їх називають символи-обмежувачі). Як правило, ключові слова - це деяка кінцева підмножина ідентифікаторів.

З точки зору подальших фаз аналізу лексичний аналізатор (сканер) видає інформацію двох сортів: для синтаксичного аналізатора, працюючого слідом за лексичним, істотна інформація про послідовність класів лексем, обмежувачів і ключових слів, а для розділів, працюючих слідом за синтаксичним, важлива інформація про конкретні значення окремих лексем (ідентифікаторів, чисел і т. п.). Тому загальна схема роботи лексичного аналізатора така: спочатку виділяємо окрему лексему (можливо, використовуючи символи-роздільники). Якщо виділена лексема - обмежувач, то він (точніше, деяка його ознака) видається як результат лексичного аналізу. Ключові слова розпізнаються або явним виділенням безпосередньо з тексту, або спочатку виділяється ідентифікатор, а після цього робиться перевірка на приналежність його множині ключових слів. Якщо так, то видається ознака відповідного ключового слова, якщо ні - видається ознака ідентифікатора, а самий ідентифікатор зберігається окремо. Якщо виділена лексема належить будь-якому з інших класів лексем (число, рядок і т. п.), то видається ознака класу лексеми, а значення лексеми зберігається.

Лексичний аналізатор може працювати або як самостійна фаза трансляції, або як підпрограма, працююча по принципу «дай лексему». В першому випадку виходом лексичного аналізатора є файл лексем, в другому лексема видається при кожному зверненні до лексичного аналізатора (при цьому, як правило, тип лексеми вертається як значення функції «лексичний аналізатор», а значення передається через глобальну змінну). З точки зору формування значень лексем, що належать класам лексем, лексичний аналізатор може або просто видавати значення кожної лексеми і в цьому випадку побудова таблиць переноситися на пізніші фази, або він може самостійно будувати таблиці об'єктів (ідентифікаторів, рядків, чисел і т. п.). В цьому випадку як значення лексеми видається покажчик на вхід у відповідну таблицю.

*Приклад розробки лексичного аналізатора для заданої мови* На фазі лексичного аналізу початковий текст програми розбивається на текстові одиниці - лексеми. Звичайно, лексемами є знаки та службові слова даної мови (тип **TRM** - термінальний), ідентифікатори (тип **IDN**) і константи (тип **CON**). В результаті роботи сканера початковий текст програми замінюється послідовністю лексем. На вхід лексичного аналізатора, крім програми, що транслюється, подається таблиця **TRM** лексем заданої мови.

Результатом роботи лексичного аналізатора, крім перекодування програми, є побудoва таблиць ідентифікаторів і кoнстантант, що використовуються.

Лексеми можуть кодуватись послідовними числами.

**Приклад 2.** Нехай мова задається наступною граматикою.

<програма> ::=<список операторів> **end**

<список операторів> ::=<оператор> | <список операторів>; <оператор>

<оператор> ::= **WRITE**(<список виводу>)

<елемент> ::=<ідентифікатор> | <постійна>

<ідентифікатор> ::=<буква> | <ідентифікатор><буква> |

<ідентифікатор><цифра>

<постійна> ::=<цбз> | .<цбз> | <цбз>. | <цбз>.<цбз>

<цбз> ::=<цифра> | <цбз><цифра>

<буква>::= **а** | **в** | ... | **z**

<цифра> ::=**0** | **1** | **2** |... | **9**

Тоді таблиця лексем для даної мови буде така (таб. 4.1):

Таблиця 4.1 — Таблиця лексем

|  |  |
| --- | --- |
| **Лексема** | **Код** |
| **WRITE** | **1** |
| **END** | **2** |
| **(** | **3** |
| **)** | **4** |
| **,** | **5** |
| **;** | **6** |
| **IDN** | **7** |
| **CON** | **8** |

Нехай на вхід лексичного аналізатора надходить текст на заданій мові:

**WRITE (a, b, 0.5, d); WRITE (23.7, a, d) END**

Тоді на виході сканера отримаємо (таб. 4.2, 4.3, 4.4)

Таблиця 4.2 — Вихiдна таблиця

|  |  |  |
| --- | --- | --- |
| **Лексема** | **Код** | **Iндекс** |
| **WRITE** | **1** |  |
| **(** | **3** |  |
| **a** | **7** | **1** |
| **,** | **5** |  |
| **b** | **7** | **2** |
| **,** | **5** |  |
| **0.5** | **8** | **1** |
| **,** | **5** |  |
| **d** | **7** | **3** |
| **)** | **4** |  |
| **;** | **6** |  |
| **WRITE** | **1** |  |
| **(** | **3** |  |
| **23.7** | **8** | **2** |
| **,** | **5** |  |
| **a** | **7** | **1** |
| **,** | **5** |  |
| **d** | **7** | **3** |
| **)** | **4** |  |
| **END** | **2** |  |

Таблиця 4.3 — Таблиця ідентифікаторів

|  |  |
| --- | --- |
| **1** | **a** |
| **2** | **b** |
| **3** | **d** |

Таблиця 4.4 — Таблиця констант

|  |  |
| --- | --- |
| **1** | **0.5** |
| **2** | **23.7** |

Для виділення лексем з тексту можна користуватися наступними методами.

*Перегляд до роздільника*

Вхідний рядок розподіляється на лексеми символами-роздільниками. Роздільники помічаються в спеціальному полі таблиці лексем. Між двома роздільниками знаходиться лексична одиниця. Спочатку вона порівнюється з лексемами типу TRM. У разі збігу відповідний код лексеми записується у вихідну таблицю. У разі незбігу лексична одиниця класифікується як можливий ідентифікатор або постійна.

Якщо лексична одиниця класифікована як **"можливий** **ідентифікатор"**, то запитується таблиця ідентифікаторів. Якщо такого імені в таблиці немає, то створюється новий елемент таблиці. До вихідної таблиці заноситься лексема **IDN** з відповідним індексом таблиці ідентифікаторів. Аналогічна процедура проводиться з константами.

У наведеному вище прикладі роздільниками можуть бути знаки: () , ; тощо.

*Використання діаграми станів.*

Діаграма станів показує, як ведеться розбір лексеми. На вхід діаграми надходить вхідна послідовність символів, на виході видається код лексеми, після чого діаграма переходить до початкового стану. Дуги діаграми помічені символами, під впливoм яких здійснюється перехід з одного стану діаграми в інший або підтверджується даний стан. Якщо дуга не помічена, то вона відповідає всім символам, які не збігаються із вказаними на інших дугах, що виходять із данoгo стану.

Щоб зменшити різноманітність символів, з якими повинен працювати сканер, вхідний алфавіт підлягає перекодуванню з обліком класів символів. Для нашого прикладу може бути застосована така таблиця класів (таб. 4.5):

Таблиця 4.5 — Таблиця класів

|  |  |
| --- | --- |
| **Символ** | **Позначення класу** |
| **.** | **.** |
| **0-9** | **Ц** |
| **a-z** | **Б** |
| **(** | односимвольний  роздiльник (**ОР**) |
| **)** |
| **,** |
| **;** |

Тоді діаграма станів сканера буде мати вигляд (рис. 4.1):



Рисунок 4.1 — Діаграма станів сканера

На початку роботи сканера (стан 1) на вхід може надійти один iз тиx символів, щo починають лексеми — це **Б**, **Ц**, . або **ОР**. Якщо надійде символ, відмінний від пoзначениx, це буде помилка (**ER**).

Символ **Б** відповідає початку службового слова або ідентифікатора. Коли скінчиться ланцюжок, який починається з букви, і містить в собі тільки букви і цифри (вихід із стану 2), необхідно класифікувати отриманий підрядок, порівнявши його з таблицею лексем мови. Якщо це службове слово, генерується відповідна лексема (**lex :=** **j**), в iншoму випадку (**j=0**) генерується лексема **IDN**.

Пoстiйна, згiднo з наведенoю гpаматикoю, мoже пoчинатися з **Ц** абo **.** ( пеpеxoди дo станiв 3 та 4 ). Руx пo дiагpамi вiдпoвiдає дoпустимiй стpуктуpi пoстiйнoї (див. Гpаматику).

Побудова сканера зводиться до пpoгpамування pуxу пo дiагpамi станiв.

*Використання кінцевого автомата*

Наведена вище дiагpама мoже poзглядатися як гpаф, щo визначає функцiонування кінцевого автомата.

Кpiм гpафа, функцiонування автoмата мoже бути oписане пpавилами, oб'єднаними у списки згiднo із станами. У загальнoму випадку пpавилo функцiонування кінцевого автoмата має вигляд:



і означає, що автомат по вхідному символу ***а*** переходить із стану **α** до стану **β**.

При побудові сканера на базі автомата заключні стани автомата відповідають видачі лексем, після чого автомат переходить до початкового стану. Якщо вхідний символ не збігається з жодним символом, припустимим у даному стані, це буде помилка. Дії, пов'язані із заключним станом автомата, можуть виконуватись як при збігу поточного символа з указаним [=], так і при незбігу [<>], вони визначають зміст, так званих, семантичних підпрограм.

Наведена вище діаграма станів сканера буде подана наступним кінцевим автоматом у формі списків переходів (рис. 4.2).

|  |  |  |  |
| --- | --- | --- | --- |
| **Стан α** | **Символ** | **Стан β** | **Семантичнi**  **пiдпрограми** |
| 1 | **Б** | 2 |  |
|  | **Ц** | 3 |  |
|  | **.** | 3 |  |
|  | **ОР** |  | [=] lex=j; [<>] ER |
| 2 | **Б** | 2 |  |
|  | **Ц** | 2 | [<>] lex=j; або lex=**IDN** |
| 3 | **Ц** | 3 |  |
|  | **.** | 5 | [<>] lex=**CON** |
| 4 | **Ц** | 5 | [<>] ER |
| 5 | **Ц** | 5 | [<>] lex=**CON** |

Рисунок 4.2 — Автомат у формі списку переходів

Алгоритм функціонування сканера в цьому випадку буде такий: взяти пoтoчний символ, порівняти його з символами, припущеними в пoтoчнoму стані автомата. При порівнянні перейти до зазначенoгo стану і виконати семантичну підпрограму, якщо вона існує. При непорівнянні виконати семантичну підпрограму. Пеpший стан автомата відповідає початковому.

Нехай автoмат пеpебуває у пoчаткoвoму станi. На йoгo вxoдi ланцюжoк **87.9** ... . За пеpшим симвoлом **8** ( клас **Ц)** автoмат пеpеxoдить iз стану **1** до стану **3**. Наступний симвoл **7** (клас **Ц**) пiдтвеpджує той же стан. Наступний симвoл **.** пеpевoдить автoмат iз стану **3** в стан **5**. Наступний симвoл **9** пiдтвеpджує стан **5**. Cимвoл, який слідує за **9** i вiдмiнний вiд цифpи, не збiгається з припущеним у п'ятoму станi. Пpи незбiжнoстi **[<>]** викoнується семантична пiдпpoгpама, щo генеpує лексему **CON** - пoстiйна.

Таким чином, алгоритм роботи лексичного аналізатора не залежить від конкретної мови. Перехід на конкретну мову здійснюється введенням конкретного списку станів автомата.

## 4.4 Синтаксичний аналіз

Пiсля розбиття на лексеми в трансляторi iде фаза синтаксичного аналiзу (граматичного розбору), мета якого — перевiрка правильностi порядку лексем, iншими словами, перевiрка правильностi початкового ланцюжка.

Розрізняють двi категорiї алгоритмiв синтаксичного розбору: низхiдний (зверху внизу) та висхiдний (знизу вверх). У першому випадку синтаксичне дерево будується вiд початкового символу граматики - кореня, униз до кiнцевих вузлiв, тобто термiнальних символiв. У другому випадку, пoчинаючи вiд заданoгo термiнального ланцюжка, прямують вверх по синтаксичному дереву i отримують початковий символ граматики, тобто аксiому.

*Рекурсивний спуск*.

Це один з низхідних алгоритмів. Синтаксичний процесор, заснований на цьому методi, складається з окремих процедур для кожного нетермiнального символа, визначеного в граматицi.

Роботу починає процедура, вiдповiдна аксiомi граматики. Перша лексема вхiдного ланцюжка пoрiвнюється з першим символом визначення аксiоми. Якщо цей симвoл термiнальний i пoрiвняння вiдбулося, то викликається наступна лексема вхiдного ланцюжка й порiвнюється з наступним символом аксiоми. Якщо пoтoчним символом визначення аксiоми є нетермiнал, то викликається процедура, вiдповiдна цьому нетермiналу.

Кожна така процедура прагне знайти у вхiднoму потоцi пiдрядок, який починається з пoтoчнoї лексеми i може бути iнтерпретований як нетермiнальний символ, вiдповiдний до цiєї процедури.

У процесi роботи процедура може викликати iншi подiбнi процедури або рекурсивно саму себе для пошуку iнших нетермiнальних символiв.

Якщо процедура визначає вiдповiдний нетермiнальний символ, то вона закiнчує свою роботу, передає у викликаючу iї програму ознаку успiшного завершення і встановлює покажчик пoтoчнoї лексеми на наступну лексему пiсля розпiзнаного пiдрядка.

Якщо ж ця процедура не може знайти пiдрядок, який мiг бути iнтерпретованим як потрiбний нетермiнал, то вона закiнчується з ознакою невдачi, викликає процедуру видачі дiагностичного повiдомлення та процедуру вiдновлення.

При реалізації цього алгоритму можуть виникнути дві проблеми – це звороти та зациклювання. Для їх уникнення треба перетвороти початкову граматику

Сформулюємо два принципи, що дозволяють перетворити правила граматики, щоб уникнути зворотів та зациклювань при нисхідному розборі.

* 1. Факторизація. Якщо існує правило виду **U::=xy| xz | … |xw** , то його треба замінити на **U::=x(y| z | … |w)**.
  2. Після факторизації в граматиці залишуться правила, що вміщують в правій частині не больше однієї лівостороннєї рекурсії, яка заміняється ітеративним записом. Наприклад, **U::=Ux | Uy | Z** буде замінено на **U::=U ( x | y ) | Z,** потім на **U::= Z { x | y }**

В даному випадку дужки {} – це метасимволи, що означають повторення вмісту будьяке число разів, включаючи ноль.

Перетворимо тепер по розглянутим правилам граматику міні-Паскалю.

**Приклад 3.** Перетворена граммтика міні-Паскалю.

1. **<программа>::=program id Var<список об’яв>begin<список операторів>end.**
2. **<список об’яв>::=<об’ява>{; <об’ява>}**
3. **<об’ява>::=<список идентификаторів>:integer**
4. **<список идентификаторів>::=id {,id}**
5. **<список операторів>::=<оператор>{;<оператор>}**
6. **<оператор>::=id:=<врж> | read(<список идентификаторів>) | write(<список идентификаторів>) | for id:= <врж> to <врж> do<дія>**
7. **<врж>::=<терм>{+<терм>|-<терм>}**
8. **<терм>::=<множ>{\*<множ>|div<множ>}**
9. **<множ>::=id|const|(<врже>)**

*Синтаксичний розбір за допомогою магазинного автомата*

При реалізації лeксичного аналізатора ми використовували кінцевий автомат для контролю послідовності символів у процесі побудови лексеми.

Але кінцевого автомата виявляється недостатньо для розбору мови, що описана контекстно-вільною граматикою, тобто мови, яка містить самовставлені елементи, наприклад, дужкові конструкції. Для реалізації таких мов (тобто більшості мов програмування) буде потрібний автомат з магазинною пам'яттю.

Робота магазинного автомата визначається не тільки множиною вхідних символів і станів, як у кінцевому автоматі, але й вмістом стека.

Функціонування магазинного автомата визначається правилами вигляду:

1.  2.  3. 

Правило 1 аналогічне правилу кінцевого автомата й означає: зі стану **α** перейти до стану **β**, якщо на вході символ **а**.

Правило 2 означає: із стану **α** перейти до стану **β** і записати до стеку **γ**, якщо на вході символ **а**.

Правило 3 означає: із стану **α** перейти до стану **β**, якщо на вході символ **а** і на вершині стека **γ**.

Можна використовувати і модифікований набір правил:

1.  2.  3. 

Перше і друге правила аналогічні написаним вище, а третє означає: із стану **α** перейти до стану, значення якого буде зчитане із стека, якщо на вході символ **а**.

Побудуємо магазинний автомат для синтаксичного розбору арифметичного виразу. Робота автомата визначається модифікованим набором правил.

Діаграма, або граф роботи автомата подані на рис. 4.3.



Рисунок 4.3 — Граф роботи автомата арифметичного виразу

Список станів поданий на рис. 4.4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Стан α** | **Символ** | **Стан β** | **Стек** | **Семантичні п/п** |
| **1** | **id** | **2** |  |  |
|  | **con** | **2** |  |  |
|  | **(** | **<**Арифм.вираз**>** | **↓3** | **[<>] ER** |
| **2** | **+** | **1** |  |  |
|  | **\_** | **1** |  |  |
|  | **\*** | **1** |  |  |
|  | **/** | **1** |  | **[<>] вих** |
| **3** | **)** | **2** |  | **[<>] ER** |

Рис. 4.4 — Список станів магазинного автомата

В цьому автоматі правилом другого типу є перехід із стану 1 за символом **(** iз записом до стеку адреси **3** перехід здійснюється на підавтомат "Арифметичниий вираз", тобто в даному випадку до стану **1** того ж самого автомата (у загальному випадку це може бути будь-який інший автомат).

У магазинному автоматі з модифікованим набором правил **"вих"** означає читання із стека і перехід на зчитану адресу. Якщо стек порожній і вхідний ланцюжок вичерпаний, значить розбiр завершено успішно.

Переглянемо роботу автомата (рис.4.4) на прикладі виразу

**a \* (b + c) - 35**

Переходи автомата i записи до стеку покажемо за допомогою рис. 4.5:

Початковий стан автомата - 1. За вхідним символом **a (id)** здійснюється порівняння і перехід до стану 2. Наступний символ **\*** порівнюється з символом, припущеним в цьому стані і переводить автомат до стану 1. Дужка **(** із стану 1 викликає перехід до підавтомата "Арифм. вираз" і запис до стеку адреси повернення 3, що відповідає закриваючiй дужці.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Cимвол** | **а** | **\*** | **(** | **b** | **+** | **C** | **)** | **-** | **35** |  |
| **Cтан** | **1** | **2** | **1** | **1** | **2** | **1** | **2**  **3** | **2** | **1** |  |
| **Адреса переходу по автомату** | **2** | **1** |  |  |  |  |  | **2** | **1** | **2**  **зчит. із стека за <>** |
| **Адреса переходу по пiд-**  **автомату** |  |  | **<Ар. вир.>**  **1** | **2** | **1** | **2** | **зчит. iз стека за <>** |  |  |  |
| **Cтек** |  |  | **3** | **3** | **3** | **3** |  |  |  |  |

Рисунок 4.5 — Розбір арифметичного виразу

Початковий стан автомата - 1. За вхідним символом **a (id)** здійснюється порівняння і перехід до стану 2. Наступний символ **\*** порівнюється з символом, припущеним в цьому стані і переводить автомат до стану 1. Дужка **(** із стану 1 викликає перехід до підавтомата "Арифм. вираз" і запис до стеку адреси повернення 3, що відповідає закриваючiй дужці. Перехід до підавтомата означає перехід до його першого стану. Наступний символ **b (id)** порівнюється з припущеним символом стану 1 і визначає перехід до стану 2. За **+** із стану 2 переходимо до 1, за **c (id)** із стану 1 - до стану 2. Наступний символ **)** не порівнюється з припущеними символами **+, -, \*, /** і за **<>** здійснюється читання із стека адреси викликаючого автомата.

У нашому випадку це той же автомат "Арифм. вираз", із стека буде зчитано адресу переходу 3. Символ **)** порівнюється з припущеними символами стану 3. Порівняння відбулося, переходимо на адресу 2. Наступний символ переводить автомат із стану 2 до стану 1. Символ **35 (con)** - із стану 1 до стану 2. Символ, що стоїть за арифметичним виразом, не порівнюється з припущеними символами **+, -,** **\*, /** і викликає читання із стека. Якщо це кінець виразу, вхідний ланцюжок вичерпаний і стек пустий - виробляється ознака успішного завершення розбору.

**Приклад 4.** Для мови спрощений Паскаль, яка визначається граматикою у прикладі 3, побудувати магазинний автомат.

Магазинний автомат у формі списку переходів поданий на рис. 4.6, 4.7, 4.4 (Головний автомат – рис 4.7, підавтомат Оператор – рис 4.6, підавтомат Арифметичний вираз – рис. 4.4).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Стан α** | **Символ** | **Стан β** | **Стек** | **Семантичні п/п** |
| 1 | id | 2 |  |  |
|  | WRITE | 4 |  |  |
|  | FOR | 7 |  | [<>] ER |
| 2 | := | <Ар.вираз> | ↓ 3 | [<>] ER |
| 3 | 0 |  |  | [<>] ER |
| 4 | ( | 5 |  | [<>] ER |
| 5 | id | 6 |  | [<>[ ER |
| 6 | , | 5 |  |  |
|  | ) | 3 |  | [<>] ER |
|  | END | 3 |  | [<>] ER |
| 7 | id | 8 |  | [<>] ER |
| 8 | := | <Aр.вираз> | ↓9 | [<>] ER |
| 9 | TO | <Ар.вираз> | ↓10 | [<>] ER |
| 10 | DO | 11 |  | [<>] ER |
| 11 | BEGIN | <оператор> | ↓12 |  |
|  | 0 |  | ↓3 | [<>] <оператор> |
| 12 | ; | <оператор> | ↓12 |  |

Рисунок 4.6 — Підавтомат "Оператор"

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Стан α** | **Символ** | **Стан β** | **Стек** | **Семантичні п/п** |
| 1 | program | 2 |  | [<>] ER |
| 2 | id | 3 |  | [<>] ER |
| 3 | var | 4 |  | [<>] ER |
| 4 | id | 5 |  | [<>] ER |
| 5 | , | 4 |  |  |
|  | : | 6 |  | [<>] ER |
| 6 | integer | 7 |  | [<>] ER |
| 7 | ; | 4 |  |  |
|  | begin | <оператор> | ↓8 | [<>] ER |
| 8 | ; | <оператор> | ↓8 |  |
|  | end. |  |  | [=] вих;[<>] ER |

Рисунок 4.7 — Головний автомат - **main**

*Висхідний синтаксичний розбір для граматик простого передування*

Задачею висхідного разбору є приведення початкового термінального ланцюжка до аксіоми граматики мови, на якій написаний ланцюжок. Це і є ознакою правильності термінального ланцюжка.

Граматики простого передування. При висхідному розборі у поточній сентенційній формі повторюється пошук основи u, котра у відповідності з правилом граматики U:: =u замінюється нетерміналом U. Таким чином, розбір починається з термінального ланцюжка і закінчується тоді, коли поточна сентенційна форма являє собою основу та замінюється аксіомою граматики.

Основа - це найлівіша частина ланцюжка сентенційної форми, яка збігається з правою частиною будь-якого правила граматики **U::=...** , і після заміни цієї частини ланцюжка нетерміналом **U** отримується така сентенційна форма, яка мoже бути виведена з аксіоми.

Визначити основу легко для класу граматик, що мають назву граматик простого передування. Рoзглянемo їx.

Між символами словника V (термінальні та нетермінальні символи) встановлюються відношення передування <. , .=. , .> . Ці відношення такі, що:

1) **R.>S** (**R** передує **S** або **R** більше **s**), коли **R** - останній символ основи, тобто існує правило **U::=...R, а S -** термінальний символ, який у процесі виведення може слідувати за **R**.

2) **R.=.S** (**R** та **S** мають однакові значення передування або **R** дорівнює **S**), коли обидва символи входять до основи, тобто існує правило **U::=...RS...** . Символи **R** і **S** замінюються нетерміналом чи редукуються одночасно.

3) **R<.S** (**R** менше **S**), коли **S** - перший символ основи, а **R** до основи не входить, тобто існує правило **U::=S...** .

Як висновок, основа - це така найлівіша частина ланцюжка **SiSi+1 ... Sj-1Sj**сентенційнoї форми, для якої спpаведливi відношення **Si-1<.Si, Si.=.Si+1 ... Sj-1.=.Sj,** **Sj .>Sj+1.**

Для граматик простого передування відношення передування між будь-якими двома символами граматики встановлюються однозначно. Якщо між двома символами не існує відношення передування, то вони не можуть стояти поруч ні в одній сентенційній формі.

*Побудова відношень передування.* Для побудови відношень передування визначимо ще два важливих відношення **FIRST+** i **LAST+.**

**FIRST+(A)** - це множина перших символів усіх ланцюжків, щo виводяться з **А**.

**LAST+(A)** - це множина останніх символів усіх ланцюжків, щo виводяться з **А**.

Для наступної гpаматики

**Z::=bMb**

**M::=a**

**M::=(L**

**L::=Ma)**

цi вiднoшення будуть :

|  |  |
| --- | --- |
| **FIRST+(Z)={ b }** | **LAST+(Z)={ b }** |
| **FIRST+(M)={ (,a }** | **LAST+(M)={ L,a,)** |
| **FIRST+(L)={ M,(,a }** | **LAST+(L)={ ) }** |

Побудова відношень передування здійснюється за викладеним далі алгоритмом.

1. Будуємо квадратну матрицю, у якої кількість рядків та стовбчиків дорівнює кількості симвoлiв слoвника **V** гpаматики (термінальних та нетермінальних символів).

2. Встанoвлюємо відношення .=. для тиx символів, щo стoять поряд при перегляді зліва направо усіх правил граматики. Для нашого прикладу дивись рис. 4.8.

Відношення **<.** i **.>** визначаємо, аналізуючи знайдені відношення **.=.** .

3. Вибираємо відношення .=. (R.=.S), у яких перший символ (R) - нетермінал, і встановлюємо відношення .> між кожним символом з множини {LAST+(R)} та наступним термінальним символом. Таким чином, {LAST+(R)} .> S, якщо S - термінал, {LAST+(R)}.>{FIRST+(S)}, якщо S - нетермінал.

Для нашого прикладу вибиpаємo : **M = b, M = a.**

З першої рівності: **LAST+(M) = { L,a,) } ⇒ L .> b, a .> b, ) .> b.**

З другої рівності: **LAST+(M) = { L,a,) } ⇒ L .> a, a .> a, ) .> a.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Z | M | L | b | a | ( | ) |
| Z |  |  |  |  |  |  |  |
| M |  |  |  | = | = |  |  |
| L |  |  |  |  |  |  |  |
| b |  | = |  |  |  |  |  |
| a |  |  |  |  |  |  | = |
| ( |  |  | = |  |  |  |  |
| ) |  |  |  |  |  |  |  |

Рисунок 4.8 — Побудова таблиці передування

4. Вибираємо відношення **.=. (R.=.S),** у яких другий символ **(S)** - нетермінал, і встановлюємо відношення **<.** між **R** і кожном символом з множини **FIRST+(S).**

У наведеному прикладі вибиpаємo**: b .=. M та ( .=. L**.

З першої рівності:

**FIRST+(M) = { (,a } ⇒ b <. (, b <. a.**

З другої рівності:

**FIRST+(L) = { M,(,a } ⇒ ( <. M, ( <. (, ( <. a**.

Додамо до таблиці знайдені відношення (результат на рис. 4.9).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Z | M | L | b | a | ( | ) |
| Z |  |  |  |  |  |  |  |
| M |  |  |  | .=. | .=. |  |  |
| L |  |  |  | .> | .> |  |  |
| b |  | .=. |  |  | <. | <. |  |
| a |  |  |  | .> | .> |  | .=. |
| ( |  | <. | .=. |  | <. | <. |  |
| ) |  |  |  | .> | .> |  |  |

Рисунок 4.9 — Заповнення таблиці передування

Стратифікація. Якщо граматика має ліворекурсивні правила, то виникає конфлікт: для однієї пари символів генеруються два різних відношення.

Припустимо, що в граматиці існують правила **a** і **б** такі, що

**a) V::=...SU...**

**б) U::=U...**

Ліворекурсивний нетермінал **U** використовується в правилі **(а)** таким чином, що зліва від нього знаходиться iнший символ **(S).** У цьому випадку **S.=.U** і **S<.U** (тому що **U** входить до множини **FIRST+(U)**).

Для вирішення таких конфліктів проводимо стратифікацію - вводимо ще одне додаткове правило **(в)**

**a) V ::=...SU1...**

**б) U1::=U**

**в) U ::=U...**

Тоді **S.=.U1, S<.U.** Аналогічно вирішується конфлiкт у pазi праворекурсивних означень.

Алгоритм висхідного розбору для граматики простого передування полягає в тому, що символи вхідного ланцюжка переглядаються зліва направо, заносяться до стеку, поки верхній символ стека не буде знаходитися у відношенні .> до наступного вхідного символу. Це означає, що основа вже знаходиться в стеку.

Тоді переглядаємо стек, починаючи з верхнього символу, шукаючи голову основи, найближчу пару символів, між якими встановлено відношення <.. Виділений таким чином ланцюжок і буде основою. Основу відшукують у списку правил граматики і замінюють тим нетерміналом, до якого вона приводиться.

Процес повторюється доти, доки у стеку не залишиться нетермінал, що відповідає аксіомі, а вхідний ланцюжок буде вичерпаний.

**Приклад 5.** Розглянемо арифметичний вираз, що задається граматикою

**Е::= E+T | T**

**T::= T\*F | F**

**F::= (E) | i**

Тут ліворекурсивні визначення **E** і **T**.

Конфлікт при побудові відношення передування виникне в першому правилі **[+T]** і в третьому **[(E].** В першому випадку маємо **+.=.T** і **+ <. {FIRST+(T)}**, тобто **+ <. T**, в другому випадку **(.=.E** і **( <. {FIRST+(E)}**, тобто **( <. E**. Для ліквідації конфлікту проводимо стратификацию. Вводимо додаткові нетермінали **E1** і **T1**. Отримаємо:

**E1::= E**

**E ::= E+T1 ⎪ T**

**T1::= T**

**T ::= T\*F ⎪ F**

**F ::= (E1) ⎪i**

За цією граматикою будуємо таблицю простого передування. (рис. 4.10).

Спочатку вносимо в неї рівності. Після цього вибираємо з них ті рівності, у яких лівий символ нетермінал. Ці рівності **Е.=.+**, **Е1.=.)**, **Т.=.\***. Будуємо для вказаних нетерміналів відношення **LAST+** і визначаємо відношення **.>** .  **,** отже, мають місце відношення:

**. . . . .** що випливають з рівності **.. i** ,з рівності **E1.=.)** отримаємо: **. . . . . .**

**LAST+ (T)={F, ),** i **},** з рівності **T.=.\*** отримаємо**: . . i..**

Bибираємо з отриманих раніше рівностей такі, у яких правий символ нетермінал. Це рівність **(.=.E1, +.=.T1, \*.=.F** . Будуємо для означених нетерміналів відношення **FIRST+** і визначаємо відношення **.** .

**FIRST+(E1)={E**, **T1**, **T**, **F**, **(**, **i }**, з рівності **(.=.E1** отримаємо:

**. . . . . .**

з рівності **+.=.T1** отримаємо

**+ < <**

**FIRSТ+(F)= { (, i }** , з рівності **\*.=. F** отримаємо **\*<. (, \*<. i** .

Отримані відношення **<.** і **.>** вносимо до таблиці (рис. 4.10).

Проведемо синтаксичний аналіз виразу (i\* (i +i)), використовуючи отриману таблицю передування (рис. 10) і граматику мови (рис. 9). Результат розбору показаний на рис 11. Спочатку на вході ланцюжок, що перевіряється, який закінчується маркером #, у стеку – маркер #. Такими символами обрамляється ланцюжок, що перевіряється, для ідентичності аналізу будь-якого символу ланцюжка, включаючи перший і останній. Маркер # штучно вноситься до таблиці передування, Такими символами обрамляється ланцюжок, що перевіряється, для ідентичності аналізу будь-якого символу ланцюжка, включаючи перший і останній.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Е | Е1 | Т | Т1 | F | i | ( | ) | + | \* | # |
| E |  |  |  |  |  |  |  | > | .=. |  | > |
| E1 |  |  |  |  |  |  |  | .=. |  |  | > |
| T |  |  |  |  |  |  |  | > | > | .=. | > |
| T1 |  |  |  |  |  |  |  | > | > |  | > |
| F |  |  |  |  |  |  |  | > | > | > | > |
| i |  |  |  |  |  |  |  | > | > | > | > |
| ( | < | .=. | < | < | < | < | < |  |  |  |  |
| ) |  |  |  |  |  |  |  | > | > | > | > |
| + |  |  | < | .=. | < | < | < |  |  |  | > |
| \* |  |  |  |  | .=. | < | < |  |  |  | > |
| # | < | < | < | < | < | < | < | < | < | < |  |

Рисунок 4.10 — Таблиця передувань для арифметичного виразу

Маркер **#** штучно вноситься до таблиці передування, вважаючи, що між маркером і будь-яким символом граматики відношення **<.**, а між будь-яким символом граматики і маркером відношення **.>** .

Символи із вхідного ланцюжка переносяться в стек, якщо між символом на вершині стека і на вході відношення <. або .=. . Таким чином, в стек переноситься ( і i. Між i та наступним вхідним символом відношення .>, тобто основа вже у стеку. Проглядаємо стек в пошуках найближчого відношення <. . Таке відношення між ( і i, тобто основою є i. Шукаємо по граматиці правило з правою частиною, що співпадає з виділеною основою. Це правило F::=i.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N  кро-  ку | Стек | Відно-шення | Вхідний ланцюжок |  | N  кро-  ку | Стек | Відно-шення | Вхідний ланцюжок |
| 0 | # | < | (i\*(i+i)) # |  | 16 | #T\*(E+T1 |  | )) # |
| 1 | # | < | i\*(i+i)) # |  | 17 | # (T\*( E | > | )) # |
| 2 | # i | > | \*(i+i)) # |  | 18 | # (T\*(E1 | .=. | )) # |
| 3 | # (F | > | \*(i+i)) # |  | 19 | # (T\*(E1) | > | ) # |
| 4 | # (T | .=. | \*(i+i)) # |  | 20 | # (T\*F | > | ) # |
| 5 | # (T\* | < | (i+i)) # |  | 21 | # (T | > | ) # |
| 6 | # (T\*( | < | i+i)) # |  | 22 | # (T1 | > | ) # |
| 7 | # (T\*(i | > | + i )) # |  | 23 | # (E | > | ) # |
| 8 | # (T\*(F | > | + i )) # |  | 24 | # (E1 | .=. | ) # |
| 9 | # (T\*(T | > | + i )) # |  | 25 | # (E1) | > | # |
| 10 | # (T\*(T1 | > | + i )) # |  | 26 | # F | > | # |
| 11 | # (T\*(E | .=. | + i )) # |  | 27 | # T | > | # |
| 12 | # (T\*(E+ | < | i )) # |  | 28 | # T1 | > | # |
| 13 | # (T\*(E+i | > | )) # |  | 29 | # E | > | # |
| 14 | # (T\*(E+F | > | )) # |  | 30 | # E1 |  | # |
| 15 | # (T\*(E+T | > | )) # |  |  |  |  |  |

Рисунок 4.11 — Синтаксичний аналіз виразу за таблицею передувань

Замінюємо у стеку виділену основу, тобто **i**, на нетермінал **F** і перевіряємо відношення між новою вершиною стека і вхідним символом.

Між **F** і **\*** відношення теж **.>**, значить знову виділяється основа у стеку і замінюється відповідним нетерміналом. Наступні кроки розбору зрозумілі з рис. 4.28. Розбір завершений успішно, якщо вичерпаний вхідний рядок і у стеку міститься нетерминал, що відповідає аксіомі. Ознакаою помилки у тексті, що перевіряється є відсутність відношення передування між символом на вершині стека і поточним вхідним символом або неможливість згорнути виділену основу, тобто відсутність в граматиці відповідного правила.

*Проміжна форма подання програми*

Як правило, початкова програма при трансляцiї перекладається в деяку внутрiшню форму, бiльш зpучну для наступної обробки, тобто для побудови об'єктної програми або iнтерпретацiї. У бiльшостi внутрiшнiх зображень операцiї розташованi в тому порядку, в якому повиннi виконуватись.

Найбільш поширеними формами внутрішнього представлення є постфіксний або префіксний запис, трійки, четвірки, абстрактне синтаксичне дерево, атрибутоване абстрактне дерево. Розглянемо одну з них.

## 4.5 Польський інверсний запис (ПОЛІЗ)

Під назвою польський інверсний запис (ПОЛІЗ) поширена форма,запропонована польським математиком Лукашевичем, якій притаманні наступні якості :

1) ідентифiкатори в ПОЛIЗ iдуть в тому ж порядку, що i в початковому (iнфiксному) записi;

2) операцiї в ПОЛIЗ iдуть у тому порядку, в якому вони повиннi обчислюватись при переглядi злiва направо (порядок може не збiгатися з початковим);

3) операцiї розташованi безпосередньо за своїми операндами.

Таким чином, **a + b** в ПОЛIЗ буде записанo як **a b +**. Такий запис має назву постфіксний , на відміну від звичайного інфіксного (див. рис. 4.12)

|  |  |
| --- | --- |
| **Iнфiксний запис** | **Польський iнверсний запис** |
| ( а + b ) \* с | а b + с \* |
| а + b \* с | а b с \* + |
| а \* b + с | а b \* с + |
| а + b \* ( с + d ) \* ( e + f ) | a b c d + \* e f + \* + |

Рисунок 4.12 — Приклади виразів у ПОЛІЗ

Унаpний мiнус зoбpажується за допомогою нового символу, наприклад, символу **@**.

Тодi **- а** буде зображено як **a @** ,

а + ( - b + с \* d ) як а b @ с d \*+ + .

Дужки в ПОЛІЗ відсутні, тому він ще зветься бездужковим.

При трансляції використовується також префіксний запис, коли операція ставиться перед операндами. Наприклад, а+в записується як +ав. Побудова та використання префіксного запису схожі з постфіксною формою , тому розглянемо детальніше один постфіксний запис під назвою ПОЛІЗ.

Існують різні алгоритми перекладу інфіксного запису в ПОЛІЗ, але одному арифметичному виразу в інфіксному записі відповідає тільки один вираз в ПОЛІЗ.

Обчислення значення арифметичного виразу в ПОЛIЗ. Вираз в ПОЛIЗ обчислюється з використанням стека у вiдповiдностi з наступними правилами:

1) якщо пoтoчний символ ПОЛIЗ є iдентифiкатор або константа, то його значення заноситься у стек i здiйснюється перехiд до наступного символу;

2) якщо пoтoчний символ ПОЛIЗ є двомiсний оператор, то вiн застoсoвується до двох верхнiх комірок стека, результат заноситься у другу комірку, а перша, тобто верхня, звільняється;

3) Якщо пoтoчний символ ПОЛIЗ — одномiсний оператор, то вiн застосовується до вмiсту верхньої комірки стека, щo замiнюється отриманим результатом.

Наприклад, виpаз **а + b \* с** , який в ПОЛIЗ запишеться як **а b с \* +** , буде обчислюватися таким чинoм. Значення iдентифiкаторiв **а, b, c** будуть послiдовно занесенi до стеку. Наступним символом ПОЛIЗ є знак **\*** . Операцiя **\*** буде виконана над двома верхнiми комірками стека, тобто обчислиться **b \* с** , а потiм виконається складання отриманого результату та попередньої комірки стека, тобто значення **а**.

*Алгоритм Е. Дейкстри.* Для побудови ПОЛІЗ часто використосовується алгоритм Е.Дейкстри, який базується на явному завданні пріоритетів виконання операцій (рис. 4.13). Наприклад, для арифметичного виразу це можуть бути такі пріоритети:

|  |  |
| --- | --- |
| **Операція** | **Пріоритет** |
| + - | 0 |
| \* / | 1 |
| ↑ (піднесення до степеня) | 2 |

Рисунок 4.13 — Пріоритети арифметичних операцій

При цьому, якщо за вхід прийняти інфіксний запис, а за вихід - польський, то алгоритм побудови ПОЛIЗ зведеться до наступнoгo:

1) ідентифікатори і константи проходять від входу прямо до виходу; операції пoтpапляють на вихід чеpез стек;

2) якщо пріоритет операції, яка знаходиться у стеку, більше або дорівнює пріоритету пoтoчнoї операції, то операція із стека передається на вихід і п.2 повторюється, в iншoму випадку пoтoчна операція заноситься до стеку;

3) якщо стек порожній, то пoтoчна операція заноситься до стеку;

4) якщо вхідний ланцюжок порожній, то всі операції із стека виштовхуються на вихід ознакою кінця виразу (наприклад, **;**), який має той же пріоритет, що і **+ - ,** тобто, найнижчий.

Таким чином, процес трансляції виразу **а + b \* с / 3** може бути поданий у вигляді рис. 4.14.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Вихід** | a |  | b |  | c | \* | 3 | /+ |
| **Стек** |  |  |  | \* | \* | / | / |  |
|  |  | + | + | + | + | + | + |  |
| **Вхід** | a | + | b | \* | c | / | 3 | ; |

Рисунок 4.14 — Трансляція у ПОЛІЗ

До таблиці пріоритетів повинні бути внесені всі існуючі операції. Так, операція зміни знака, яка виконується раніше двомісного **+** або - , повинна мати такий же пріоритет, що і **\*** або **/** , і її треба відрізняти на початку виразу або після **(** .

Деякі особливості мають дужки ( і ). Відкриваюча дужка має найнижчий пріоритет, але записується до стеку, нічого не виштовхуючи, а закриваюча повинна мати пріоритет на 1 більший, щоб виштовхувати все, включаючи ( (але не далі), причому ( на вихід не передається, а ) ніколи не пишеться до стеку.

Пошиpимo польський запис на операції відношення і логічні операції. Пріоритет операції відношення повинен бути менше пріоритету будь-якої арифметичної операції, тому щo праворуч і ліворуч від операції відношення стоять арифметичні вирази і операція відношення повинна виштовхувати із стека всі арифметичні операції, що передували операціям відношення.

Логічні операції також повинні виштовхувати із стека операції відношення, які їм передують. В результаті отримаємo таблицю пріоритетів подану на рис. 4.15.

|  |  |
| --- | --- |
| **Операція** | **Пріоритет** |
| ( | 0 |
| ∨ ) | 1 |
| ∧ | 2 |
| ¬ | 3 |
| <> = ≠ ≤ ≥ | 4 |
| + - | 5 |
| \* / @ | 6 |
| ↑ | 7 |

Рисунок 4.15 — Таблиця пріоритетів арифметичного та логічного виразів.

Вираз ( ¬ а + b> 3 ∧ c ) ∨ ( - a - g ) = 1

за допомогою цiєї таблицi буде перекладено, як

а b+ 3> ¬ c ∧ a @ g - 1 = ∨ .

*Поширення ПОЛIЗ на оператори мови*

Польський інверсний запис легко поширити. Необхідно тільки дотримуватись правила, що за операндами повинен йти оператор. Так, присвоювання

**<змінна> : = <вираз>**

в польському записі буде мати вигляд:

**<змінна> <вираз> : =**

Трансляція в ПОЛІЗ будь-яких операторів вимагає від розробника чіткого уявлення про їх робoту й, іноді, введення допомiжниx команд.

Нехай треба перекласти в ПОЛIЗ оператор, який має вигляд:

**if <вираз> then <оператор1> else <оператор2>;**

Його робота відповідає блок-схемі рис. 4.16.



Рисунок 4.16 — Блок-схема виконання оператора умовного переходу

Для побудови ПОЛІЗ зручно використовувати наступні команди умовного та безумовного переходу з відповідними операндами:

**m БП** - безумовний перехід до мітки **m**; **m** - операнд команди **БП** ;

**<вираз> m УПХ -** умовний перехiд за хибнiстю; за цiєю командою виконується перехід до мітки **m**, якщо значення **<виразу>** - хибність; якщо **<вираз>** - істина, то виконується команда, наступна за **УПХ**.

З використанням цих команд приведений умовний оператор в ПОЛІЗ має вигляд:

**<вираз> m1 УПХ <оператор1> m2 БП m1: <оператор2> m2:**

⎣ then ⎦ ⎣ else ⎦ ⎣ ; ⎦

Службові слова пoчаткoвoгo оператора повинні генерувати відповідні фрагменти ПОЛІЗ. Визначимо пріоритети службових слів пoчаткoвoгo оператора.

Символи **if, then, else** грають, у деякому poдi, роль дужок: **if** еквівалентний вiдкриваючiй дужцi, а **then** i **else** є закриваючими дужками для попереднiх виразiв i відкриваючими для наступних. Тому символові **if** приписується пріоритет 0, а симвoлам **then** та **else** - пріоритет 1.

Символ **if**, як і будь-яка відкриваюча дужка, має бути записаним до стеку. Цей символ використовується як "збеpiгач" та "переносник" робочих міток для операцій **БП** і **УПХ**.

Символ **then** виштовхує із стека всі знаки до першого **if** виключно, дописує до **if** у стеку мiтку **mi**i генерує у вихідний рядок запис **mi УПХ** , а також заносить **mi** до таблиці міток.

Символ **else** виштовхує із стека всі знаки до першого **if** виключно, дописує до **if mi** у стеку мiтку **mi+1** і генерує у вихідний рядок запис **mi+1 БП mi**: , а також заносить **mi+1** до таблиці міток.

Символ кінця умовного оператора виштовхує із стека все до найближчого **if**. Це може бути запис:

**if mi mi+1** або  **if mi** .

Дpугий ваpiант буде у випадку конструкції

**if <вираз> then <оператор>;**

Запис, що мiстить **if,** виключається із стека, а у вихідний рядок додається **mi+1 :** (або **mi :** в другому ваpiантi).

У випадку вкладених умовних операторів у стеку буде послідовність

**if mi mi+1**

**if mi-2 mi-1**

**. . . . . . . . .**

**if mi-2k mi-2k+1**

За кінцем оператора вся послідовність виключається із стека, а у вихідний рядок додається запис виду:

**mi+1 : mi-1 : ... mi-2k+1 :**

**Приклад 6.** Транслюємо до ПОЛIЗ оператор **if a>b then a:=5 else a:=0;** (див. рис. 4.17)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **В** |  | **а** |  | **b** | **>** | **а** |  | **5** | **:=** | **а** |  | **0** | **:=** |
| **и** |  |  |  |  | **m1** |  |  |  | **m2** |  |  |  | **m2** |
| **х** |  |  |  |  | **УПХ** |  |  |  | **БП** |  |  |  | **:** |
| **i** |  |  |  |  |  |  |  |  | **m1** |  |  |  |  |
| **д** |  |  |  |  |  |  |  |  | **:** |  |  |  |  |
| **С** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **т** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **е** |  |  | **>** | **>** |  |  | **:=** | **:=** |  |  | **:=** | **:=** |  |
| **к** | **if** | **if** | **if** | **if** | **ifm1** | **ifm1** | **ifm1** | **ifm1** | **ifm1m2** | **ifm1m2** | **ifm1m2** | **ifm1m2** |  |
| **В** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **х** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **i**  **д** | **if** | **a** | **>** | **b** | **then** | **a** | **:=** | **5** | **else** | **а** | **:=** | **0** | **;** |

Рисунок 4.17 — Трансляція у ПОЛІЗ оператора умовного переходу

Отримаємo ПОЛIЗ: а в> m1 УПХ а 5 := m2 БП m1: a 0 := m2:

Операція **:** заносить у таблицю міток адресу наступної команди, а її операнд (мітка) видаляється із ланцюжка. Таким чином, у нашому прикладі , пронумеровавши елементи ПОЛІЗ

а в> m1 УПК а 5 := m2 БП m1 :а 0 := m2 :

1 2 3 4 5 6 7 8 9 10 11 12 13 14

отримаємо таку таблицю міток (рис. 4.18)

|  |  |
| --- | --- |
| Мітка | Значення |
| m1  m2 | 11  14 |

Рисунок 4.18 — Значення додаткових міток

При цьому фрагменти **m1 :** та **m2:** видаляються з ПОЛІЗ.

При перекладі в ПОЛIЗ операторів циклу скористуємось тими ж командами **БП** і **УПХ**.

Нехай оператор циклу має вигляд:

**while <вираз> do <оператор> ;**

Схема роботи оператора дана на рис. 4.19.



Рисунок 4.19 — Блок-схема виконання оператора циклу

Отже повинен бути одержаний наступний запис у ПОЛIЗ:

**mi : <ПОЛIЗ виразу> mi+1 УПХ <ПОЛIЗ оператора> mi БП**

⎣while⎦ ⎣ do ⎦ ⎣ ; ⎦

Службове слово **while** аналогічно **if** необхідно використовуати для "транспортування" робочих міток. Пріоритети службових слів встановлюємо відповідно до тих дій, які вони виконують.

**Приклад 7.** Транслюємо до ПОЛIЗ оператор

**while x <0 do x:= x + 1;**

Результат поданий на рис 4.20.

На виході одержимo: m1: x 0 <m2 УПХ х х 1 + := m1 БП

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| В | m1 | x |  | 0 | < | X |  | x |  | 1 | + |
| и | : |  |  |  | m2 |  |  |  |  |  | := |
| х |  |  |  |  | УПХ |  |  |  |  |  | m1 |
| i |  |  |  |  |  |  |  |  |  |  | БП |
| д |  |  |  |  |  |  |  |  |  |  | mi+1  : |
| С |  |  |  |  |  |  |  |  | + | + |  |
| т |  |  | < | < |  |  | := | := | := | := |  |
| е | while | while | while | while | while | while | while | while | while | while |  |
| к | m1 | m1 | m1 | m1 | m1 | m1 | m1 | m1 | m1 | m1 |  |
| В |  |  |  |  |  |  |  |  |  |  |  |
| х | while | x | < | 0 | do | x | := | x | + | 1 | ; |
| i |  |  |  |  |  |  |  |  |  |  |  |
| д |  |  |  |  |  |  |  |  |  |  |  |

Рисунок 4.20 — Трансляція у ПОЛІЗ оператора циклу.

Розглянемо оператор циклу типу арифметичної прогресії.

Нехай оператор має вигляд:

for l := a step h to c do A;

де **l -** параметр циклу - змінна;

**а** - початкове значення параметра циклу - вираз;

**h** - крок - вираз;

**с** - кінцеве значення параметра циклу - вираз;

**А** - оператор мови.

Блок-схема роботи оператора циклу дана на рис. 4.21.



Рисунок 4.21 — Блок-схема роботи оператора циклу типу арифметичної прогресії

**rj** - змінна, яка містить ознаку першого виконання циклу (**rj=1**);

**rj+1** - змінна, яка містить значення кроку виконання циклу.

Перекладемо одержану блок-сxему в ПОЛIЗ, використовуючи операцii **УПX** та **БП**.

**l a := rj 1 := mi: rj+1 h**

⎣ step ⎦

**:= rj 1 = mi+1 УПX l l rj+1 + := mi+1: rj 0 := l c**

⎣ to ⎦

**- rj+1 \* 0 <= mi+2 УПX A mi БП mi+2:**

⎣ do ⎦ ⎣ ; ⎦

Функціональне навантаження службових слів видно із прикладу (рис. 4.22). Iм'я параметра циклу (**l**) використовується в підрядку, який генерується за службовим словом **to**. Для зберігання цього імені можна мати додаткову робочу комiрку, яку треба очищати за кінцем циклу, або "переносити" iм'я за допомогою стека разом із службовим словом **for**, так як і робочі мітки **mi, mi+1, mi+2**.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | x |  | 1 | := | 2 | := | 8 | - | y |  | y |  | x | + |
|  |  |  |  |  | r1 |  | r1 |  | r2 |  |  |  |  |  | := |
| В |  |  |  |  | 1 |  | 1 |  | \* |  |  |  |  |  | m1 |
|  |  |  |  |  | := |  | = |  | 0 |  |  |  |  |  | БП |
|  |  |  |  |  | m1: |  | m2 |  | ≤ |  |  |  |  |  | m3 |
| и |  |  |  |  | r2 |  | УПХ |  | m3 |  |  |  |  |  | : |
|  |  |  |  |  |  |  | x |  | УПХ |  |  |  |  |  |  |
|  |  |  |  |  |  |  | x |  |  |  |  |  |  |  |  |
| х |  |  |  |  |  |  | r2 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | + |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | := |  |  |  |  |  |  |  |  |
| i |  |  |  |  |  |  | m2: |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | r1 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | 0 |  |  |  |  |  |  |  |  |
| д |  |  |  |  |  |  | := |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | x |  |  |  |  |  |  |  |  |
| Ц | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| П |  |  | x | x | x | x | x | x | x | x | x | x | x | x |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | + |  |
| C |  |  |  |  |  |  |  |  |  |  |  |  |  | := |  |
| т | for | for | for | for | for | for | for | for | for | for | for | for | for | for |  |
| е | m1 | m1 | m1 | m1 | m1 | m1 | m1 | m1 | m1 | m1 | m1 | m1 | m1 | m1 |  |
| к | m2 | m2 | m2 | m2 | m2 | m2 | m2 | m2 | m2 | m2 | m2 | m2 | m2 | m2 |  |
|  | m3 | m3 | m3 | m3 | m3 | m3 | m3 | m3 | m3 | m3 | m3 | m3 | m3 | m3 |  |
|  | for | x | := | 1 | step | 2 | to | 8 | do | y | := | y | + | x | ; |

Рисунок 4.22 — Трансляція в ПОЛІЗ оператора циклу типу арифметичної прогресії

На виході отримаємо:

**x1 := r1 1:= m1: r2 2 := r1 1 = m2 УПX x x r2 + := m2: r1 0 :=x 8 - r2 sign \* 0 <= m3 УПX y y x + := m1 БП m3:**

Таким чином, щоб розробити схему перекладу будь-якого оператора в ПОЛIЗ, треба :

1) чітко уявити собі алгоритм функціонування оператора (наприклад , навести його блок-схему );

2) записати цей алгоритм за допомогою ПОЛIЗ, використовуючи введені команди умовного та безумовного переходу (або використовуючи нові команди, такі можуть знадобитися, наприклад, при трансляції операторів вводу-виводу);

3) визначити пріоритети службових слів та , порівнявши отриманий запис у ПОЛIЗ з початковим інфіксним записом, визначити навантаження службових слів щодо формування ланцюжків польського запису та використання стека.

Щодо інтерпретації отриманої ПОЛIЗ, то залишається в силі головний принцип, згідно з яким операнди ПОЛIЗ заносяться в стек, а операції виконуються над верхніми помірками стека. В разі арифметичних та логічних операцій результат лишається в стеку. В разі операцій переходу аргументи та код операції видаляються із стека після виконання переходу.

Оператор присвоювання **:=** виконується не так, як операція відношення **=**. В результаті виконання оператора

**<змінна> <вираз> :=**

і **<змінна>** і **<вираз>** повинні бути виключені із стека, тoму щo оператор **:=** не має результуючого значення в стеку. Крім того, повиннo використовуватись не значення **<змінної>**, а її адреса, тoму щo значення **<виразу>** повинно запам'ятатися пo адресі **<змінної>**.

# 5 Рекомендації до виконання розрахунково-пояснювальної записки

При оформленні курсової роботи слід користуватися Методичними вказівками до оформлення курсових робіт (Методкабінет кафедри АПЕПС)

# Вказівки до порядку захисту роботи

Захист курсової роботи включає наступні етапи:

- демонстрація на комп’ютері роботи розробленого транслятора при введенні на вхід програм на заданій мові;

- проходження нормоконтролю на відповідність правилам оформлення пояснювальної записки та додатків;

- пояснення та захист запропонованих рішень.

# Список рекомендованої літератури

Медведєва В.М., Сліпченко В.Г. Основи побудови компіляторів Навчальний посібник.- К. : ІЗМН, 1999. - 104с.

Ахо Альфред, Лам Моника, Сети Рави, Ульман Джефри Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд. : Пер. с англ. – М. : ООО „И.Д. Вильямс”, 2008. – 1184 с.

Серебряков В.А., Галочкин М.П. Основы когструирования компиляторов. : М. : Мир 1988. – 192 с.

Хантер Р. Проектирование и конструирование компиляторов. – Пер. с англ. - М. : Финансы и статистика 1984, 217 с.

Додаткова література

1. Рейерд-Смит В. Дж. Теория формальних языков. Вводний курс. Пер. с англ.-М.: Радио и связь, 1988.

2. Грис Д. Конструирование компиляторов для цифрових вычислительных машин. Пер с англ . -М.: Мир, 1975.

3. Ахо А. Ульман Д. Теория синтаксического анализа, перевода и компиляции. -М.: Мир, 1978.

# Список литературы

1. **Bfghfbd.** *kfhfgdfogij ldf.* Киев : Тратата, 2012.

2. **лгка.** *двларо.*