ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

КАФЕДРА ІНФОРМАЦІЙНОЇ БЕЗПЕКИ ТА КОМП’ЮТЕРНОЇ ІНЖЕНЕРІЇ

# КУРСОВА РОБОТА

з дисципліни: “Системне програмне забезпечення”

На тему: “Побудова компілятора”

Студент 4 курсу групи КМ-175

Спеціальність 123

«Комп’ютерна інженерія»

Косенко А.В.

Керівник ктн..доцент

Миронюк Т.В.

Національна шкала \_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_ ECTS \_\_\_\_\_

Оцінка:

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

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

Черкаси 2020

Зміст

Вступ

Розліл 1

1.1 Методи створення таблиць ідентифікаторів

1.2 Призначення лексичного аналізатора

1.3 Призначення синтаксичного аналізатора

1.4 Методи і алгоритми генерації та оптимізації коду

Розліл 2

2.1 Таблиця ідентифікаторів

2.1.1 Вхідні дані

2.1.2 Призначення таблиці ідентифікаторів

2.1.3 Метод рехешування

2.1.4 Метод бінарного дерева

2.1.5 Результати

2.2 Проектування лексичного аналізатор

2.2.1 Вхідні дані

2.2.2 Принципи роботи лексичного аналізатора

2.2.3 Результати

2.3 Генерація та оптимізація об'єктного коду

2.3.1 Вхідні дані

2.3.2 Результати

2.4 Результат виконання програми-компілятора

2.4.1 Меню

2.4.2 Результат токенізатора

2.4.3 Абстрактне синтаксичне дерево

Висновки

Додаток 1

Черкаський державний технологічний університет

ФІТІС Кафедра ІБтаКІ

Спеціальність 123 «Комп'ютерна інженерія».

ЗАВДАННЯ

На курсову роботу студента

Косенко Андрiя Віталійовича

1. Тема роботи Побудова компілятора

2. Вхідні дані проекту

Індивідуальне завдання:

1. Тип констант: двійкові

2. Додаткові операції: --

3. Оператор циклу: while <вираз> do <оператор>

4. Метод оптимізації: Виключення зайвих операцій

5. Тип даних: byte

6. Тип коментарів: /\*коментар\*/

1. Зміст розрахунково- пояснювальної записки:

Вступ (тема, мета, постановка завдання)

1. Теорія побудови компіляторів.
2. Програмна реалізація компілятора.
3. Лістинг програми.
4. Результати роботи програми.

Висновки.

Додаток І

Лістинг

**Вступ**

**Мета:** вивчення складових частин, головних принципів побудови і функціонування компіляторів, практичне освоєння методів побудови найпростимих компіляторів для заданої вхідної мови

Курсова робота полягає у створенні компілятора з заданої підмиожини мови Паскаля з незначними модифікаціями. Результатами курсової роботи є програмна реалізація заданого компілятора і пояснювальна записка, оформлена відповідно до вимог стандартів та завдання на курсову роботу. Для програмної реалізації компілятора рекомендується використовувати мову програмування Object Pascal і систему програмування Borland Delphi. Можливо використовувати інші мови та системи програмування за погодженням з викладачем. Компілятор рекомендується побудувати з наступних складових частин: 1. Лексичний аналізатор. 2. Синтаксичний аналізатор. 3. Оптимізатор. 4. Генератор результуючого коду. Для побудови компілятора рекомендується використовувати методи, засвоєні в ході виконання лабораторних робіт з курсу «Системне програмне забезпечення».

**Визначення поняття компілятор**

Компілятор (англ. Compiler від англ. to compile - збирати в ціле) - комп'ютерна програма (або набір к. програм), що перетворює (компілює вихідний код, написаний певною мовою програмування (вихідна мова, англ source language), на семантично еквівалентний код в іншій мові програмування, який зазвичай необхідний для виконання програми машиною, наприклад, комп'ютером.

Компіляція - процес перетворювання високорівненого вихідното тексту програми, і екпіпалентний текст програми, але вже низькорівневою мовою.

Компонувальник (Компоновщщик, Лінкер) - програма, яка генерує виконуваний модуль шляхом зв'язутання об'єктиих файлів проекту.

Коротко компілятор можна визначити як програму або технічний засіб, шо виконує компіляцію. Історично компілятором називалась програма, що зв'язувала підпрограми, чим і зумовлено походження слова. Сьогодні це завдання виконує компонувальник. Для виконання програма не завжди повинна бути перекладена компілятором, існує також інший принцип: покрокове виконання програмних інструкцій інтерпретатором.

**Задачі компілятора**

Комп'ютери самі по собі здатні виконувати тільки дуже обмежений набір операцій, які називаються машинними кодами. У старі часи, коли з'явились перші комп'ютери, програми писалися в машинних кодах, що представляють собою послідовності двійкових чисел, однозначно сприймаються комп'ютером.

В кінці 50-х кодів минулого століття з'явилися перші мови програмування, такі як мова асемблера і Фортаран. Для того, щоб комп’ютер міг зрозуміти програму написану на якійсь мові програмування, необхідний перекладач (транслятор) такої програми в машинні коди, Відзначимо, що, якщо оператор мови асемблер відображається при трансляції частіше всього.

Деякі оператори мови асемблера,наприклад, такі, як оператори введення / виведення, відображаються в кілька машинних команд, в одну машинну інструкцію, пропозиції мов більш високого рівня відображаються, взагалі кажучи, в кілька машинних іиструкший.

Транслятори бувають двох типів: компілятори і інтерпретатори. Процес компіляції складається з двох частими: аналізу і синтезу, Аналізуюча частина компілятора розбиває вихідну програму на складові її елементи (конструкції мови) і створює проміжнє представлення вихідної програми. Синтезує частина з проміжного представлення створює нову програму, яку комп'ютер в змозі зрозуміти. Така програма називається об'єктної програмою. Об'єктна програма можє в подальшому виконуватися без перетрансляції. Як проміжний уявлення зазвичай використовуються дерева, зокрема, так звані дерева розбору. Під деревом розбору розуміється дерево, кожен вузол якого є певною операції, а сини цього вузла - операндам.

**Основні способі реалізації**

Лексичний аналіз - зазвичай є первинною фазою. Задача лексичного аналізатора - розбиття суцільного тексту програми на токени (лексеми). Токен є атомарною одницею мови. ЛЕксичний аналізатор формує токени з набору тексту, перевіряє коректність та допустимість символу, створює новий токен та прив\*язує до нього характеристики - позицю від початку тексту, тип токена та конкретний тип. На вихід аналізатор подає потік токенів.

Синтаксичний аналіз - створення внутрішнього представлення та

перевірка отриманих від лексичного аналізатора потоку токенів на відповідність граматиці мови. Основна задача - створення внутрішнього представлення для подальших етапів аналізу та синтезу. Внутрішнім представленням є абстрактне синтаксичне дерево (AST). Під час побудови дерева відбуваються перевірки на коректність та відповідність вхідних даних граматиці мови.

Семантичний аналіз - перевіряє відповідність типів, наявність змінних, полів, методів, класів. Приймає на вхід А8Т від синтаксичного аналізатора та подає на вихід атрибутивне дерево розбору. Контроль типів керується матрицями (для бінарних операторів) та векторами (для унарних) конверсії. При необхідності у дерево розбору додаються вузли конверсії

Оптимізація коду полягає у видаленні недосяжних блоків та інструкцій, згортці констант, детектуванні та видаленні невикористаних змінних та методів.

Генерація коду - останній етап, під час якого з атрибутивного дерева формується SSА представлення, відбуваються останні оптимізації та формується код мовою асемблера.

**Актуальність теми**

JavaScript в 2020 році - це стратегічний навик. Навіть якщо він не потрібен вам прямо зараз, то напевно знадобиться трохи пізніше.

Можливості використання сучасного JavaScript:

* Створювати клієнтську і серверну частини веб-проектів.
* Писати мобільні і десктопні програми.
* Писати утиліти командного рядка.
* Керувати пристроями інтернету речей,
* Створювати "безсервернії" функції.
* Працювати з величезною кількістю додатків, шо мають JavaScript

Node.js дозволяє виконувати JavaScript-код поза звичним браузерним середовищема. Ця потужна платформа може не тільки створювати сервені додатки. Вона надає практично необмежені можливості для автоматизації розробки і тестування коду.

Переважна більшість розробників (93%) в тій чи іншій мірі мають справу з Node.js. Ця платформа - основний гравець на арені JavaScript в 2019 році.

Зараз JavaScript - найпопулярніша мова програмування для браузерів.

**1 ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО ЕТАПИ РОЗРОБКИ КОМПІЛЯТОРА**

1.1 Методи створення таблиць ідентифікаторів

При використанні семантичного аналізу, генерації коду і оптимізації результуючої програми, компілятор має оперувати характеристиками основних елементів вхідної програми. Для змінної – тип і адреса комірки пам’яті; для константи – її значення; для функції – кількість і тип формальних аргументів, тип повернення результату і адрес виклику коду функції.

Задача компілятора в тому, щоб зберігаючи деяку інформацію пов’язану з кожним елементом програми і мати доступ до цієї інформації за іменем елемента. Для вирішення цієї задачі компілятор організує спеціальні сховища даних, так званих таблиць ідентифікаторів або таблиць символів.

Одним з ефективних методів пошуків у впорядкованому переліку з N елементів є бінарне дерево.

Алгоритм бінарного пошуку. Шуканий символ порівнюється з елементом (N+1)/2 в середині таблиці. Якщо цей елемент не є шуканим, то ми маємо продивитися тільки блок елементів, що пронумеровані від 1 до ((N+1)/2)–1 або блок елементів від ((N+1)/2)+1 до N, в залежності від того, менше або більше шуканий елемент від того з яким його порівнюють. Далі процес повторюється над необхідним блоком в 2 рази меншого за розміром. Так продовжується до того моменту, доки або шуканий елемент не буде знайдено, або алгоритм не дійде до наступного блоку, що містить 1 або 2 елементи з якими можливо виконувати пряме порівняння шуканого елемента.

Побудова таблиці ідентифікаторів за методом бінарного дерева. Таблиця має вигляд бінарного дерева: кожна вершина в ньому не має більше двох гілок (ліва і права). Перший ідентифікатор розміщується в вершині дерева. Всі наступні ідентифікатори потрапляють в дерево за алгоритмом:

Вибрати наступний ідентифікатор з вхідного потоку даних. Якщо наступного ідентифікатора немає, то побудову дерева завершено.

Зробити поточним вузлом дерева кореневу вершину.

Порівняти ім’я наступного ідентифікатора з іменем ідентифікатора, що міститься в поточному вузлі дерева.

Якщо ім’я наступного ідентифікатора менше то перейти до кроку 5. Якщо дорівнює – припинити виконання алгоритму (Двох однакових ідентифікаторів не має бути). Інакше перейти до кроку 7.

Якщо у поточного вузла існує ліва вершина, то зробити її поточним вузлом і перейти до кроку 3, інакше – до кроку 6.

Створити нову вершину, розмістити в ній інформацію про наступний ідентифікатор. Зробити цю нову вершину лівою вершиною поточного вузла і повернутися до кроку 1.

Якщо у поточного вузла існує права вершина, то зробити її поточним вузлом і повернутись до кроку 3. Інакше перейти до кроку 8.

Створити нову вершину, розмістити в ній інформацію про наступний ідентифікатор. Зробити цю нову вершину правою вершиною поточного вузла і повернутися до кроку 1.

Хеш адресація з рехешируванням. Якщо n0=h(A) вказує на вже зайняту комірку, то необхідно обчислити значення n1=h1(A) і перевірити зайнятість комірки за адресою n1. Якщо ж вона зайнята то обчислити значення n2 і так поки не знайдеться вільна комірка або наступне значення hi(A) не співпаде з h(A).

В останньому випадку вважається що таблиця ідентифікаторів заповнена,і місця в ній більше немає, і видається інформація про пошук розміщення ідентифікаторів в таблицю. Тоді пошук елемента А в таблиці ідентифікаторів виконується за таким алгоритмом:

1. Обчислити значення функції N=h(A) для шуканого елементу А.

2. Якщо комірка за адресою n вільна, то елемент не знайдено, алгоритм завершено інакше необхідно порівняти ім’я елементу в комірці n з ім’ям шуканого елементу А. Якщо вони співпадають, то елемент знайдено і алгоритм завершено, інакше і:=1 і перейти до кроку 3.

3. Обчислити ni=hi(A) якщо комірка за адресою ni вільна або n=ni, то елемент не знайдено і алгоритм завершено, інакше порівнюється ім’я елементу ni з ім’ям шуканого елементу А. Якщо вони співпадають, то елемент знайдено, алгоритм завершено, інакше і:=і+1 і повторити крок 3.

1.2 Призначення лексичного аналізатора

Лексичний аналізатор або сканер – це частина компілятору, яка читає літери програми вхідною мовою і будує з них слова (лексеми) вихідної мови. На вхід лексичного аналізатора надходить текст вхідної програми, а вихідна інформація передається для подальшої обробки компілятором на етапі синтаксичного аналізу і розбору.

Лексема (лексична одиниця мови) – структурна одиниця мови, яка складається з елементарних символів мови і не містить у своєму складі інших структурних одиниць мови. Лексемами є ідентифікатори, константи, ключові слова, знаки операцій і т.д. Склад можливих лексем кожної конкретної мови програмування визначається синтаксисом цієї мови. З теоретичної точки зору лексичний аналізатор не є обов’язковою необхідною частиною компілятора. Його функції можуть виконуватись на етапі синтаксичного аналізу. Але є деякі причини за яких до складу практичних компіляторів обов’язково включають лексичний аналізатор:

1. Спрощується робота з текстом вхідної програми на етапі синтаксичного розбору і скорочується об’єм інформації, що обробляється, оскільки лексичних аналізатор структурує вхідний текст програми і видаляє всю зайву неважливу інформацію.

2. Для виділення в тексті і розбитті лексем можна застосувати просту ефективну техніку аналізу, в той час як синтаксичний аналіз вхідної мови використовує складні алгоритми мови.

3. Лексичний аналізатор відділяє складний за конструкцією синтаксичний аналіз від роботи безпосередньо з текстом вхідної програми, структура якого може змінюватись в залежності від версії вхідної мови. За такої конструкції компілятору при переході від однієї версії мови до іншої достатньо тільки перебудувати відносно легкий лексичний аналізатор.

Результатом роботи лексичного аналізатора є перелік всіх знайдених в тексті вхідної програми лексем з урахуванням характеристик кожної лексеми. Цей перелік можна представити у вигляді таблиць лексем

1.3 Призначення синтаксичного аналізатора

Синтаксичний аналізатор – це частина компілятора, яка відповідає за виявлення і перевірку синтаксичних конструкцій вхідної мови.

У завдання синтаксичного аналізатора входить:

* знайти і виділити синтаксичні конструкції в тексті вхідної програми;
* встановити тип і перевірити правильність кожної синтаксичної конструкції;
* представити синтаксичні конструкції у вигляді, зручному для подальшої генерації тексту результуючої програми.

Синтаксичний аналізатор – це основна частина компілятора на етапі аналізу. Без виконання синтаксичного аналізатору робота компілятора безглузда, у той час як лексичний розбір, в принципі, не є обов'язковою фазою компіляції. Усі завдання з перевірки синтаксису вхідної мови можуть бути вирішені на етапі синтаксичного розбору. Лексичний аналізатор тільки дозволяє позбавити складний за структурою синтаксичний аналізатор від рішення примітивних завдань з виявлення та запам'ятовування лексем вхідної програми.

Виходом лексичного аналізатора є таблиця лексем. Ця таблиця утворює вхід синтаксичного аналізатора, який досліджує тільки один компонент кожної лексеми – її тип. Решта інформації про лексеми використовується на більш пізніх фазах компіляції при семантичному аналізі, підготовці до генерації та генерації коду результуючої програми.

Синтаксичний аналізатор сприймає вихід лексичного аналізатора і розбирає його відповідно до граматики вхідної мови. Однак у граматиці вхідної мови програмування звичайно не уточнюється, які конструкції слід вважати лексемами. Прикладами конструкцій, які зазвичай розпізнаються під час лексичного аналізу, служать ключові слова, константи і ідентифікатори. Але ці ж конструкції можуть розпізнаватися і синтаксичним аналізатором. На практиці не існує жорсткого правила, що визначає, які конструкції повинні розпізнаватися на лексичному рівні, а які треба залишати синтаксичному аналізатору.

1.4 Методи і алгоритми генерації та оптимізації коду

Генерація об'єктного коду – це переклад компілятором внутрішнього представлення вхідної програми в ланцюжок символів вихідної мови. Оскільки вхідною мовою компілятора (на відміну від транслятора) може бути тільки або мова асемблера, або мова машинних кодів, то генерація коду породжує результуючу об'єктну програму мовою асемблера або безпосередньо машинною мовою (в машинних кодах).

Генерація об'єктного коду виконується після того, як виконані лексичний і синтаксичний аналіз програми і всі необхідні дії з підготовки до генерації коду: перевірені семантичні узгодження вхідної мови (Семантичний аналіз), виконана ідентифікація імен змінних і функцій, розподілено адресний простір під функції і змінні і т. д.

Генерацію коду можна вважати функцією, що визначена на синтаксичному дереві, побудованому в результаті синтаксичного аналізу, і на інформації, що міститься в таблиці ідентифікаторів. Характер відображення вхідної програми в послідовність команд, що виконується генерацією, залежить від вхідної мови, архітектури цільової обчислювальної системи, на яку орієнтована результуюча програма, а також від якості бажаного об'єктного коду.

В ідеалі компілятор повинен виконати синтаксичний аналіз всієї вхідної програми, потім провести її семантичний аналіз, після чого приступати до підготовки генерації і безпосередньо генерації коду.

Оптимізація програми – це обробка, пов'язана з переупорядкуванням і зміною операцій в модульній програмі з метою отримання більш ефективної результуючої об'єктної програми. Оптимізація виконується на етапах підготовки до генерації і безпосередньо при генерації об'єктного коду.

Як показники ефективності результуючої програми можна використовувати два критерії: тобсяг пам'яті, необхідний для виконання результуючої програми, і швидкість виконання (швидкодія) програми. Далеко не завжди вдається виконати оптимізацію так, щоб вона задовольняла обом цим критеріям. Найчастіше скорочення необхідного програмі обсягу даних веде до зменшення її швидкодії, і навпаки. Тому для оптимізації зазвичай вибирається один зі згаданих критеріїв. Вибір критерію оптимізації зазвичай виконується в налаштуваннях компілятора.

Принципово розрізняють два основних види оптимізуючих перетворень:

1. Перетворення початкової програми (у формі її внутрішнього подання в компіляторі), не залежні від результуючої об'єктної мови;

2. Перетворення результуючої об'єктної програми.

Розділ 2

**2.1 Таблиця ідентифікаторів**

2.1.1 Вхідні дані

Компілятор поповнює записи в таблиці ідентифікаторів протягом аналізу вхідної програми і знаходження в ній нових елементів, які потребують розміщення в таблиці.

Пошук інформації в таблиці виконується кожного разу, коли компілятору необхідні відомості про той чи інший елемент програми. Пошук елемента таблиці буде виконуватись компілятором значно частіше ніж розміщення в ній нових елементів. Кожному додаванню елемента в таблицю ідентифікаторів передує операція пошуку, щоб переконатися, що такого елемента немає. Тому таблиця ідентифікаторів організована таким чином, щоб компілятор мав можливість максимально швидко виконувати пошук необхідного йому запису в таблиці за іменем елемента з яким пов'язаний цей запис.

Способи організації таблиці ідентифікаторів:

1. Прості впорядковані списки.
2. Бінарне дерево.
3. Хеш-адресація з рехешируванням.
4. Хеш-адресація за методом ланцюжків.
5. Комбінація хеш-адресації зі списком або бінарним деревом.

2.1.2 Призначення таблиці ідентифікаторів

Перевірка правильності семантики і генерація коду вимагають знання характеристик ідентифікаторів, використовуваних в програмі мовою оригіналу. Ці характеристики з'ясовуються з описів і з того, як ідентифікатори використовуються в програмі і накопичуються в таблиці символів або таблиці ідентифікаторів. Будь-яка таблиця символів складається З набору полів, кількість яких дорівнює кількості ідентифікаторів програми. Кожне поле містить в собі повну інформацію про даному елементі таблиці. Під ідентифікаторами маються на увазі константи, змінні, імена процедур і функцій формальні і фактичні параметри.

2.1.3 Метод рехешування

Для вирішення проблеми колізії можна використовувати метод рехешування. Відповідно до цього методу, якщо для елемента А адреса п ()=h (А), обчислений за допомогою хеш-функції h, вказує на вже зайняту комірку, тo необхідно обчислити значення функціїn1 = h1 (A) і перевірити зайнятість комірки за адресою n1 . Якщо і вона зайнята, то обчислюється значення h2 (А), так до тих пір, поки або не буде знайдена вільна осередок, або чергове значенні hi (А) не співпадає з h (А). В останньому випадку вважається, що таблиця ідентифікаторів заповнена і місця в ній більше немає - видається інформація про помилку розміщення ідентифікатора в таблиці.

Пошук елемента А в таблиці ідентифікаторів буде виконуватися за наступним алгоритмом:

Крок 1. Обчислити значення хеш-функції п = h(A) для шуканого елемента

Крок 2. Якщо осередок за адресою п порожня, то елемент не знайдений,алгоритм завершений, інакше необхідно порівняти ім'я елемента в комірці п з ім'ям шуканого елемента А. Якщо вони збігаються, то слемент знайдений і алгоритм завершений, інакше і:=1 і перейти до кроку 3.

Крок 3. Обчислити П - hі (А). Якщо осередок за адресою nі порожня або п = Пі, то елемент не знайдений і алгоритм завершений, інакше - порівняти ім'я елемента в комірці Пі з ім'ям шуканого елемента А. Якщо вони збігаються, то елемент знайдений і алгоритм завершений, інакше і := i+1 і повторити крок 3.

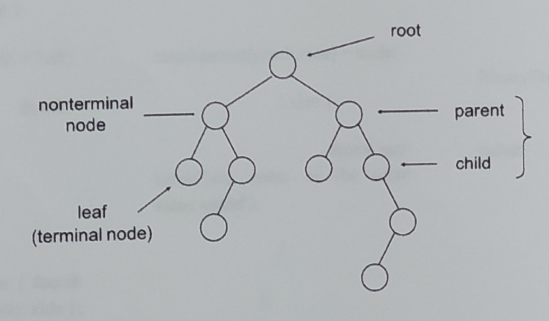
Крок 1. Вважати перший елемент у вхідному списку, створити кореневий вузол, помістити в нього перший елемент і перейти до кроку 4.

Крок 2. Якщо поточний вузол порожній, заносимо в нього значення надійшов на вхід елемента, переходимо до кроку 4. Інакше переходимо до кроку 3

Крок 3. Якщо значення в поточному вузлі збігається з котрі вступили на вхід елементом, даний елемент вже є в таблиці ідентифікаторів, перейти до кроку 4. Інакше порівняти елемент зі значенням в поточному вузлі. Якщо це значення менше значення, що зберігається в поточному вузлі, зробити поточним лівий дочірній вузол поточного вузла і перейти до кроку 2. Інакше зробити поточним правий дочірній вузол і перейти до кроку 2.

Крок 4. Якщо більше немає ідентифікаторів, які треба розмістити в таблиці, то виконання алгоритму закінчено, інакше вважати наступний ідентифікатор і перейти до кроку 2.

Графічне представлення алгоритму бінарного дерева. (рисунок 2.2)



2.1.5 Результати

Для тестування програми було обрано вихідний текстовий файл наступного змісту :

(

while (true) (

(

subtract (add 1 3)2 //Subtract 2 from 4

)

)

while (false) {

subtract (add (1,3), divide (6,2));

};

2.2 Проектування лексичного аналізатора

2.2.1 Вхідні дані

Для виконання даної частини курсової роботи потрібно написати програму, яка викjнує лексичний аналіз вхідного тексту відповідно до завдання і породжує таблицю лексем із зазначенням їх типів і значень. Текст на вхідній мові задається у вигляді символьного (текстового) файлу. Програма повинна видавати повідомлення про наявність у вхідному тексті помилок, які можуть бути виявлені на єтапі лексичного аналізу.

Програма повинна допускати наявність коментарів необмеженої довжини у вхідному файлі.

2.2.2 Принципи роботи лексичного аналізатора

Лексичний аналізатор має справу з різного роду константами і ідентифікаторами (до останніх відносяться і ключові слова). Мова констант і ідентифікаторів в більшості випадків є регулярним - тобто, може бути описаний за допомогою регулярних граматик. Розпізнавачами для регулярних мов є кінцеві автомати. Існують правила, за допомогою яких для будь-якої регулярної граматики може бути побудований недетермінований кінцевий автомат, що розпізнає ланцюжка мови, заданого цієї граматикою.

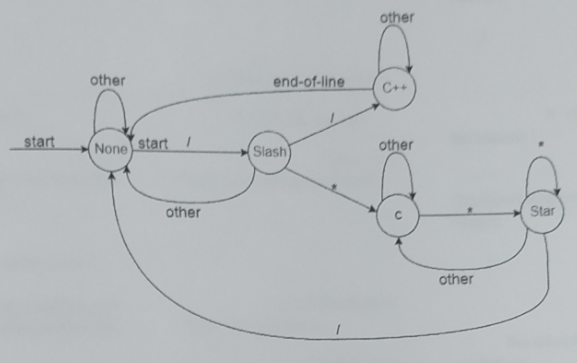


Рисунок 2.4 - Граф алгоритму лексичного аналізу

2.2.3 -Результати

|  |  |  |
| --- | --- | --- |
| constructor(language) {  this.ignore = {};  this.customOut = {};  this.functions = {};  this.errTok = 'ERROR';  if (language === ") {  this. language = ‘Custom’,  this.tokens = {};  } else if (!(language in  languages)) {  throw new  LanguageNotFoundException(lan  guage),  } else {  this.tokens =  languages[language];  this.language = language;  this.strict = true;  ignoreWhiteSpace() {  (!Object.keys(this.tokens).inclu  s("WHITESPACE")) {  this.addToken("WHITESPACE  new  RegExp(/(\u0009)\u000B}\u000  \u0020|\u00A0)\uFEFF/));  this.addl gnore("WHITESPACR")  ignoreNewLine() {  if  (Object. keys(this.tokens). include  s("NEW\_LINE")) {  this.addToken("NEW\_LINE",  New | this.addl gnore("NEW\_LINE");  }  addTokenSet(tokenSet) {  if ((tokenSet instanceof  Object)) {  throw new  TypeError(‘addTokenSet expects  an object of token names to regex  patterns’);  }  if (this.strict) {  throw new  NoCustomTokensException(this.|  anguage),  }  for (const key in tokenSet) {  if (key in this.tokens) {  throw new  DuplicateTokenException(key);  }  if (!(tokenSet[key]  instanceof RegExp)) {  throw new  TypeError('value of key should  be regexp in tokenset’);  }  this.tokens[key] =  tokenSet[key];  } | addToken(tokenName,  regPattern) {  if ({(regPattern instanceof  RegExp)) {  throw new  TypeError(‘Second = Argument  must be of type RegExp’);  if (this strict) {  throw new  NoCustomTokensE xception( this |  anguage);  }  if (tokenName in  this.tokens) {  throw new  DuplicateTokenException(token  Name);  }  this.tokens[tokenName]} =  regPattern;  }  removeToken(tokenName) {  if (this.strict) {  throw new  NoCustomTokensException(this.!  anguage);  }  retum delete  this.tokens[tokenName}; |

**2.3 Генерація та оптимізація об’єктного коді**

2.3.1 Вхідні дані

Для виконання заключної частини курсової роботи потрібно написати програму, яка на підставі дерева синтаксичного розбору породжує об’єктний код і потім виконує його оптимізацію. В якості вихідного дерева синтаксичного розбору рекомендується взяти дерево, яке породжує програма, побудована за завданням попереднього розділу роботи.

2.3.2 Результати

Лістинг згенерованого асемблерного коду.

|  |  |  |
| --- | --- | --- |
| tak.  (module  (type $t0 (func  (param i32)))  (type $t1 (func))  (type $t2 (func  (result i32)))  (type $t3 (func  (Param i32 i32 i32) (result i32)))  (import "env"  "pute \_js" (func $putc\_js (type  $t0)))  (func  S\_wasm\_call\_ctors (type $11))  (func $main (export  "main") (type $t2) (result i32)  i32.const 0)  (fune $writey\_c  (export "writev\_c") (type $t3)  (param $p0 i32) (param $p1 i32)  (param $p2 i32) (result i32) | (local $10 i32)  (local $11 i32) (local $12 i32)  (local $13 i32) (local $14 i32)  (local $15 i32)  block $B0  get\_local $p2  132.const |  i32.It\_s  br\_if SBO  i32.const 0  set\_local $10  132.const 0  set\_local $11  loop SLI0  offset=4  i32.const 0  set\_local $12  block $B2  get\_local $p]  get\_local $10  i32.const 3  i32.shl  i32.add  tee\_local $13  132.load | i32.eqz  br\_if $B2  get\_local $13  i32.const 4  i32.add  set\_local $14  i32.const 0  set\_local $15  loop $L3  get\_local $13  i32.load  i32.add  i32.load8\_s  call $pute\_js  get\_local $15  i32.const 1  i32.add  tee\_local $15  get\_local $14  i32.load  tee\_local $12  i32.1t\_u  br\_if $L3  end  end  get\_local $12  get local $11  i32.add |
| set\_local $11  get\_local $10  j32.const |  132.add  tee\_local $10  get\_local $p2  i32.ne  br\_if $L]  end | get\_local $11  return  end  i32.const 0)  (table $TO 1 1  anyfunc)  (memory $memory  (export "memory") 2)  (global $g0 (mut  132) (i32.const 66560)) | (global  $\_\_heap\_base (export  "heap\_base") i32 (i32.const  66560))  (global  $\_\_data\_end (export  "\_data\_end") i32 (432.const  1024)) |

**2.4 Результати виконання програми-компілятора**

2.4.1 Меню

Програма має меню для кращого користуванна та наглядності її роботи. На рисунку 2.4.1 можемо бачити вигляд головного меню програми.

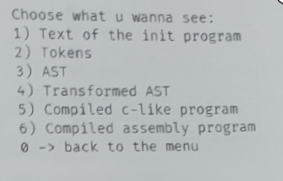


Рисунок 2.4.1 - Меню програми

2.4.2 Результат токенізатора

В ході роботи токенізатора на виході створюється масив токенів вхідної програми. Створений масив токенів показаний на рисунку 2.4.2

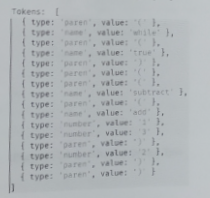
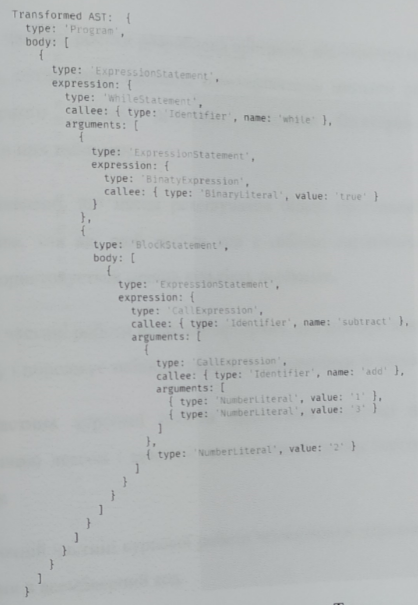


Рисунок 2.4.2 - Токенізатор

2.4.3 Абстрактне синтаксичне дерево

AST - кінцеве позначене орієнтоване дерево, в якому внутрішні вершини зіставлені (позначені) з операторами мови програмування, а листя - з відповідними операндами.

На рисунку 2.4.3 зображене абстрактне синтаксичне дерево вихідної програми (AST)



**ВИСНОВКИ**

Під час виконання даної курсової роботи, я розглянув основні складові частини компілятора, практично дослідив основні принципи побудови і функціонування компіляторів. Освоїв методи побудови простих компіляторів для заданої вихідної мови.

Створений компілятор працює з введеним кодом програми на мові Delphi, виконує аналіз та створює таблицю лексем. Програма має можливість виявляти помилки розбору та виводити інформацію про помилку. Під час синтаксичного аналізу програма розпізнає змінні та додає їх до таблиці ідентифікаторів. Компілятор, на базі таблиці лексем виконує перевірку синтаксису введеної програми. Синтаксичний аналізатор визначає, чи в правильній послідовності знаходяться оператори, звернувшись до таблиці операторного передування. Після вдалого проходження синтаксичного аналізу програма виконує створення списку тріад. З даним списком легко виконувати оптимізацію програми, а саме легко знайти операції, які повторюються і т.д. Після виконання всіх основних етапів компіляції програма генерує низькорівневий код.

**Додаток 1**

|  |
| --- |
| unit Unit1;  interface  uses  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ComCtrls, Vcl.Menus, Vcl.Grids,  Vcl.ExtCtrls, Unit11, DateUtils;  type  syntax = class;  triad = class;  ruls1 = array[0..7] of syntax;  cases = (itslex, itssyntax);  optype = (oconst, ovar, olink);  triadtype = (itsIf, itsMenshe, itsBolshe, itsSub, itsAdd, itsPrisvoit, itsJmp, itsConst, itsSame, itsNop);  noterm = record  case types: cases of  itslex: (term: string[255]);  itssyntax: (noterm\_list: TList);  end;  syntax = class(TObject)  protected  getinfo: noterm;  rulnum: integer;  public  constructor makeTerm(a: string);  constructor makeNoTerm(a, b: integer; arr: ruls1);  function toString: string;  function getItem(a: integer): syntax;  function count: integer;  property items[id: integer]: syntax read getItem;  default;  end;  operands = record  case choise: optype of  oconst: (conval: integer);  ovar: (vars: string[255]);  olink: (link: integer;);  end;  arropers = array[1..2] of operands;  triad = class(TObject)  private  change: boolean;  changetype: integer;  TriadType: triadtype;  operands: arropers;  public  constructor create(types: triadtype; const Ops: arropers);  function getlink(id: integer): integer;  procedure setlink(id: integer; num: integer);  property links[id: integer]: integer read getlink write setlink;  property trdtype: triadtype read triadtype;  function getoptype(id: integer): optype;  procedure setoptype(id: integer; opt: optype);  function getoperand(id: integer): operands;  procedure setoperand(id: integer; op: operands);  property optypes[id: integer]: optype read getoptype write setoptype;  property opers[od: integer]: operands read getoperand write setoperand; default;  end;  triadlist = class(TList)  public  function gettriad(id: integer): triad;  procedure vuvod(opt: boolean);  property triads[id: integer]: triad read gettriad;  default;  end;  TForm1 = class(TForm)  GroupBox1: TGroupBox;  Edit1: TEdit;  Button1: TButton;  Button2: TButton;  Edit2: TEdit;  Label1: TLabel;  Label2: TLabel;  Button3: TButton;  CheckBox1: TCheckBox;  GroupBox2: TGroupBox;  Label3: TLabel;  Label4: TLabel;  Button4: TButton;  CheckBox2: TCheckBox;  Memo1: TMemo;  OpenDialog1: TOpenDialog;  SaveDialog1: TSaveDialog;  Memo2: TMemo;  procedure Button4Click(Sender: TObject);  procedure FormCreate(Sender: TObject);  procedure open(Sender: TObject);  procedure save(Sender: TObject);  end;  TTriadStr = array[triadtype] of string;  const  TriadStr: TTriadStr = ('if','<','>','-','+',':=','jmp','const','reit','nop');  var  Form1: TForm1;  value, temp, dsf: string;  hshcou, trscou, hesrav, trsrav, nocl, i, ds1, cstack, countident, countsy, hashmin, hashmax: integer;  arr1: array[0..255] of string;  arr4: array[0..255] of string;  arr5: array[0..255] of string;  arr7: array[0..255] of string;  arr8: array[0..255] of integer;  arr6: array[0..255] of syntax;  arrLex: array [0..17, 0..17] of char;  pravila: array[1..16] of string = ('E', 'E;E', 'E;', 'ifEthenEelseE', 'ifEthenE', 'beginEend', 'whileEdoE', 'a:=E', 'E<E', 'E>E', 'E', 'E-E', 'E+E', 'E', '(E)', 'a');  listTriad: triadlist;  T1: TDateTime;  MS: Integer;  implementation  {$R \*.dfm}  constructor triad.create(types: triadtype; const ops: arropers);  var i: integer;  begin  inherited create;  triadtype:= types;  for i:=1 to 2 do operands[i]:= ops[i];  end;  function triad.getoperand(id: integer): operands;  begin  result:= operands[id];  end;  procedure triad.setoperand(id: integer; Op: operands);  begin  operands[id]:= op;  end;  function triadlist.gettriad(id: integer): triad;  begin  result:= triad(items[id]);  end;  procedure triad.SetOpType(id: integer; OpT: optype);  begin  operands[id].choise := OpT;  end;  function triad.getoptype(id: integer): optype;  begin  result:= operands[id].choise;  end;  function triad.getlink(id: integer): integer;  begin  result:= operands[id].link;  end;  procedure triad.setlink(id: integer; num: integer);  begin  operands[id].link:= num;  end;  function syntax.count:integer;  begin  result:=getinfo.noterm\_list.count;  end;  function syntax.toString: string;  begin  if getinfo.types = itssyntax then result:= 'E'  else result := getinfo.term;  end;  function syntax.getItem(a: integer): syntax;  begin  result:=syntax(getinfo.noterm\_list[a])  end;  constructor syntax.makeTerm(a: string);  begin  inherited create;  getinfo.types:= itslex;  getinfo.term:= a;  rulnum:= 0;  end;  constructor syntax.makeNoTerm(a, b: integer; arr: ruls1);  var i: integer;  begin  inherited create;  getinfo.types:= itssyntax;  getinfo.noterm\_list:= TList.Create;  for i:= b-1 downto 0 do getinfo.noterm\_list.add(arr[i]);  rulnum:= a;  end;  function getIndex(a: string; b: integer): integer;  begin  result:= (ord(a[1]) + ord(a[(Length(a) + 1) div 2]) - hashmin) mod (hashmax - hashmin + 1) + hashmin;  end;  function varhash(const sname: string): longint;  begin  result:= (ord(sname[1]) + Ord(sname[(length(sname)+1) div 2]) - hashmin) mod (hashmax-hashmin+1)+hashmin;  if (result < hashmin) then result:= hashmin;  end;  function addToHesh(a: string): string;  var temp, i: integer;  begin  Result:= '';  hesrav:= 0;  for i:=0 to 225 do  begin  temp:= getIndex(UpperCase(a), i);  if arr1[temp] = '' then  begin  arr1[temp]:= a;  Result:= a;  Break;  end;  hesrav:= hesrav + 1;  if (UpperCase(arr1[temp]) = UpperCase(a)) then  begin  Result:=a;  Break;  end;  end;  end;  function getFromHesh(a: string): string;  var temp, i: integer;  begin  Result:= '';  hesrav:= 0;  for i:=0 to 225 do  begin  temp:= getIndex(UpperCase(a), i);  if arr1[temp] = '' then Break;  hesrav:= hesrav + 1;  if UpperCase(arr1[temp]) = UpperCase(a) then  begin  Result:=a;  Break;  end;  end;  end;  function checkstr(s:string):boolean;  begin  checkstr:=UpCase(s[Length(s)]) in ['A'..'Z', '\_'];  if (Length(s)<>1) and (UpCase(s[Length(s)]) in ['A'..'Z', '\_']) then checkstr:=checkstr(Copy(s,1,Length(s)-1))  end;  function getTypeLex1(a: string): string;  var t:integer;  begin  if (TryStrToInt(a, t) AND (a <> 'E')) then Result:= 'CONST'  else if checkstr(a) AND (a <> 'E') then Result:= 'VAR'  else Result:= '';  end;  function getTypeLex(symbOp: syntax): string;  begin  case symbOp.rulnum of  0: Result:= getTypeLex1(symbOp.toString);  16: Result:= getTypeLex1(symbOp[0].toString);  15: Result:= getTypeLex(symbOp[1])  else Result:= '';  end;  end;  function getVar(a: string): string;  var t: integer;  begin  if ((a <> 'if') and (a <> 'then') and (a <> 'begin') and (a <> 'end') and (a <> 'while') and (a <> 'do') and (a <> 'and') and (a <> 'or') and (a <> '!') and (a <> 'else')) and (checkstr(a) or (TryStrToInt(a, t))) then Result:='a'  else Result:=a;  end;  function addToStack(a: string; b: boolean; c: syntax; normal: string): syntax;  begin  if b = true then Result:= syntax.makeTerm(normal)  else Result:= c;  cstack:= cstack+1;  arr4[cstack]:= a;  arr6[cstack]:= Result;  end;  procedure delFrStack;  begin  arr4[cstack]:= '';  cstack:= cstack-1;  end;  function getFrStack: string;  begin  if arr4[cstack] <> 'E' then result:= arr4[cstack]  else result:= arr4[cstack - 1];  end;  function getIn(a: string): integer;  begin  if (a = ';') then Result:= 0  else if (a = 'if') then Result:= 1  else if (a = '(') then Result:= 2  else if (a = ')') then Result:= 3  else if (a = 'else') then Result:= 4  else if (a = 'begin') then Result:= 5  else if (a = 'end') then Result:= 6  else if (a = 'do') then Result:= 8  else if (a = 'while') then Result:= 7  else if (a = ':=') then Result:= 11  else if (a = '<') then Result:= 12  else if (a = '>') then Result:= 13  else if (a = '+') then Result:= 14  else if (a = '-') then Result:= 15  else if (a = '!') then Result:= 16  else if (a = 'then') then Result:= 17  else Result:= 9;  end;  function makeHide: syntax;  var current: string;  tmpArray: ruls1;  i, tmpCount: integer;  str: string;  procedure AddToRule(a, b: string);  begin  current := b;  tmpArray[tmpCount] := arr6[i];  str := a + str;  delFrStack;  Inc(tmpCount);  end;  begin  Result := nil;  tmpCount := 0;  current:= '';  str:= '';  for i:= cstack downto 0 do  begin  if arr4[i] = 'E' then AddToRule(arr4[i], current)  else if current = '' then AddToRule(arr4[i], arr4[i])  else if arrLex[getIn(arr4[i]), getIn(current)] = '=' then AddToRule(arr4[i], arr4[i])  else Break;  if tmpCount > 7 then Break;  end;  if tmpCount <> 0 then  begin  for i:=1 to 16 do  if pravila[i] = str then  begin  countsy:= countsy + 1;  Result := syntax.makeNoTerm(i, tmpCount, tmpArray);  addToStack('E', false, Result, '');  Break;  end  else if (i = 16) then  begin  Application.MessageBox(PWideChar('Ошибка синтаксического разбора. Созданое правило "' + str + '" не отвечает парамитрам компилятора!'), 'Мой курсачь');  exit;  end;  end;  end;  procedure addToLeksem(a:string; b:string);  begin  Form11.add(a, b, ds1);  arr5[ds1]:=b;  inc(ds1);  end;  function getSpes(a:string; b:boolean): string;  var t:integer;  begin  if (a = ':=') then Result:= 'Присвоение значения'  else if (a = '>') then Result:= 'Знак больше'  else if (a = '!') then Result:= 'Конец файла'  else if (a = '+') then Result:= 'Знак сложения'  else if (a = '-') then Result:= 'Знак вычитания'  else if (a = ';') then Result:= 'Разделитель'  else if (a = '(') then Result:= 'Открывающая скобка'  else if (a = ')') then Result:= 'Закрывающая скобка'  else if (a = '<') then Result:= 'Знак меньше'  else if (TryStrToInt(a, t)) then Result:= 'Константа'  else if (a = 'if') or (a = 'then') or (a = 'begin') or (a = 'end') or (a = 'while') or (a = 'do') or (a = 'and') or (a = 'or') or (a = 'else') then Result:= 'Ключевое слово'  else if checkstr(a) then  begin  Result:= 'Переменная';  if b=true then  begin  if addToHesh(a) = '' then Application.MessageBox(PWideChar('Ошибка рехеширования идентификатора "' + a + '"'), 'Мой курсачь');  Inc(hshcou, hesrav);  count:= count + 1;  end;  end  else Result:= '';  end;  function analisFromLex: syntax;  var cnt, i: integer;  begin  addToLeksem(getSpes('!', true), '!');  cnt:= cnt1;  addToStack('!', false, nil, '!');  i:=0;  while i <= cnt do  begin  if (getFrStack = '!') and (getVar(arr5[i]) = '!') then Break  else  begin  case arrLex[getIn(getFrStack), getIn(getVar(arr5[i]))] of  '<', '=':  begin  Result:= addToStack(getVar(arr5[i]), true, nil, arr5[i]);  i:= i + 1;  end;  '>': Result:= makeHide  else  begin  Application.MessageBox(PWideChar('Ошибка синтаксического разбора. Нету отношений между "' + getFrStack + '" и "' + arr5[i] + '"'), 'Мой курсачь');  Break;  end;  end;  end;  end;  end;  procedure makeTree(tmp: TTreeNode; link: syntax);  var  i: integer;  tmp1: TTreeNode;  begin  for i:=0 to link.Count-1 do  begin  tmp1:= Form11.TreeView1.Items.AddChild(tmp, link[i].toString);  if link[i].toString = 'E' then makeTree(tmp1, link[i]);  end;  end;  function MakeTriadListNOP(symbTop: syntax; listTriad: triadlist): string;  var  Opers: arropers;  iIns1,iIns2,iIns3: integer;  function MakeOperand(iOp, iSymOp, iMin, iSymErr: integer; var iIns: integer): string;  var lexTmp: string;  begin  lexTmp := getTypeLex(symbTop[iSymOp]);  if lexTmp <> '' then  begin  if lexTmp = 'VAR' then  begin  Opers[iOp].choise:= ovar;  Opers[iOp].vars:= symbTop[iSymOp][0].toString;  end  else  if lexTmp = 'CONST' then  begin  Opers[iOp].choise:= oconst;  Opers[iOp].conval:= StrToInt(symbTop[iSymOp][0].toString);  end  else  begin  Result:= lexTmp;  Exit;  end;  iIns:= iMin;  Result:= '';  end  else  begin  if MakeTriadListNOP(symbTop[iSymOp],listTriad) <> '' then Exit;  iIns := listTriad.Count;  if iIns <= iMin then  begin  Result:= symbTop[iSymErr].toString;  Exit;  end;  Opers[iOp].choise:= olink;  Opers[iOp].link:= iIns-1;  end;  end;  function MakeOperation( Trd: triadtype): string;  begin  if MakeOperand(1, 0, listTriad.Count, 1, iIns1) <> '' then Exit;  if MakeOperand(2, 2, iIns1, 1, iIns2) <> '' then Exit;  listTriad.Add(triad.Create(Trd,Opers));  end;  begin  case symbTop.rulnum of  4:  begin  if MakeOperand(1, 1, listTriad.Count, 0, iIns1) <> '' then Exit;  Opers[2].choise:= olink;  Opers[2].link:= 0;  listTriad.Add(triad.Create(itsIf, Opers));  if MakeOperand(2, 3, iIns1, 2, iIns2) <> '' then Exit;  Opers[1].choise:= oconst;  Opers[1].conval:= 1;  Opers[2].choise:= olink;  Opers[2].link:= 0;  listTriad.Add(triad.Create(itsJmp,Opers));  listTriad[iIns1].Links[2]:= iIns2 + 1;  if MakeOperand(2, 5, iIns2, 4, iIns3) <> '' then Exit;  listTriad[iIns2].Links[2]:= iIns3;  end;  7:  begin  iIns3 := listTriad.Count;  if MakeOperand(1, 1, iIns3, 1, iIns1) <> '' then Exit;  iIns1 := listTriad.Count;  Opers[1].choise:= olink;  Opers[1].conval:= iIns1 - 1;  Opers[2].choise:= olink;  Opers[2].link:= 0;  listTriad.Add(triad.Create(itsIf,Opers));  if MakeOperand(2, 3, iIns1, 4, iIns2) <> '' then Exit;  Opers[1].choise:= oconst;  Opers[1].conval:= 1;  Opers[2].choise:= olink;  Opers[2].link:= iIns3;  listTriad.Add(triad.Create(itsJmp,Opers));  listTriad[iIns1].Links[2]:= listTriad.Count;  end;  5:  begin  if MakeOperand(1, 1, listTriad.Count, 0, iIns1) <> '' then Exit;  Opers[2].choise:= olink;  Opers[2].link:= 0;  listTriad.Add(triad.Create(itsIf, Opers));  if MakeOperand(2, 3, iIns1, 2, iIns2) <> '' then Exit;  listTriad[iIns1].Links[2]:= iIns2;  end;  8:  begin  if getTypeLex(symbTop[0]) <> 'VAR' then  begin  Result:= symbTop[0].toString;  Exit;  end;  Opers[1].choise:= ovar;  Opers[1].vars:= symbTop[0].toString;  if MakeOperand(2, 2, listTriad.Count, 1, iIns1) <> '' then Exit;  listTriad.Add(triad.Create(itsPrisvoit, Opers));  end;  9: Result := MakeOperation(itsMenshe);  10: Result := MakeOperation(itsBolshe);  12: Result := MakeOperation(itsSub);  13: Result := MakeOperation(itsAdd);  6, 15: Result:= MakeTriadListNOP(symbTop[1],listTriad);  2:begin  Result:= MakeTriadListNOP(symbTop[0],listTriad);  Result:= MakeTriadListNOP(symbTop[2],listTriad);  end;  16: Result:= '';  else Result:= MakeTriadListNOP(symbTop[0],listTriad);  end;  end;  function MakeTriadList(symbTop: syntax; listTriad: triadlist): string;  var  i: integer;  Opers: arropers;  begin  if MakeTriadListNOP(symbTop,listTriad) = '' then  with listTriad do  begin  Opers[1].choise:= oconst;  Opers[1].conval:= 0;  Opers[2].choise:= oconst;  Opers[2].conval:= 0;  Add(triad.Create(itsNop,Opers));  end;  end;  function GetOperStr(Op: operands): string;  begin  case Op.choise of  oconst: Result:= IntToStr(Op.conval);  ovar: Result:= Op.vars;  olink: Result:= '^' + IntToStr(Op.link + 1);  end;  end;  procedure triadlist.vuvod(opt: boolean);  var  i: integer;  tmp: TStringGrid;  begin  for i:=0 to Count-1 do  begin  tmp:=TStringGrid.create(nil);  if opt = true then tmp:= Form11.StringGrid3  else tmp:= Form11.StringGrid3;  tmp.RowCount:= i + 3;  tmp.Cells[0,i + 1]:= IntToStr(i + 1);  tmp.Cells[1,i + 1]:= TriadStr[triad(Items[i]).TriadType];  tmp.Cells[2,i + 1]:= GetOperStr(triad(Items[i]).Opers[1]);  tmp.Cells[3,i + 1]:= GetOperStr(triad(Items[i]).Opers[2]);  end;  end;  function getVal(a: string): integer;  begin  for i := 0 to 255 do  begin  Result:=-1;  if arr7[i] = a then  begin  Result:= arr8[i];  break;  end;  end;  end;  procedure optumDel (listTriad: triadlist);  var Trd, trdtemp: triad;  i, j, num: integer;  test: boolean;  begin  for i:=0 to listTriad.Count - 1 do  begin  Trd := listTriad[i];  if i <> 0 then  begin  test:= false;  for j := i - 1 downto 0 do  begin  trdtemp:= listTriad[j];  if (trd.TriadType = trdtemp.TriadType) and (trd.TriadType = itsPrisvoit)  and (trd.Operands[1].vars = trdtemp.Operands[1].vars)  and (trd.Operands[1].choise = trdtemp.Operands[1].choise)  and (trd.Operands[1].conval = trdtemp.Operands[1].conval)  and (trd.Operands[2].vars = trdtemp.Operands[2].vars)  and (trd.Operands[2].choise = trdtemp.Operands[2].choise)  and (trd.Operands[2].conval = trdtemp.Operands[2].conval)  then  begin  test:= true;  num:= j;  end;  end;  end;  if (test = true) then  begin  trd.TriadType:= itsSame;  trd.Operands[1].vars:= inttostr(num + 1);  trd.Operands[2].conval:= 0;  trd.change:= true;  trd.changetype:= 1;  if i < listTriad.Count - 1 then  begin  for j := i + 1 downto 0 do  begin  if (listTriad[j].Operands[1].choise = olink)  and (listTriad[j].Operands[1].link = i ) then  begin  listTriad[j].Operands[1].link:= listTriad[j].Operands[1].link + 1;  end;  if (listTriad[j].Operands[2].choise = olink)  and (listTriad[j].Operands[2].link = i ) then  begin  listTriad[j].Operands[2].link:= listTriad[j].Operands[2].link + 1;  end;  end;  end;  end;  end;  end;  procedure lexem (Memo1: TMemo);  var tmp: string;  TS: TStringList;  i, j: integer;  begin  for i:= 0 to Memo1.lines.count do  begin  tmp := Memo1.Lines[i];  tmp := tmp.Remove(pos('/\*', tmp) - 1, pos('\*/', tmp) - pos('/\*', tmp) + 3);  //ShowMessage(tmp);  tmp := tmp.Replace('\n', '');  tmp := tmp.Replace(' ', '');  tmp := tmp.Replace('(', '( ');  tmp := tmp.Replace(')', ' )');  tmp := tmp.Replace(';', ' ;');  tmp := tmp.Replace('if', 'if ');  tmp := tmp.Replace('then', ' then ');  tmp := tmp.Replace(':=', ' := ');  tmp := tmp.Replace('else', ' else ');  tmp := tmp.Replace('while', 'while ');  tmp := tmp.Replace('do', ' do ');  tmp := tmp.Replace('+', ' + ');  tmp := tmp.Replace('-', ' - ');  tmp := tmp.Replace('>', ' > ');  tmp := tmp.Replace('<', ' < ');  tmp := tmp.Replace(' ', ' ');  TS := TStringList.Create;  TS.Delimiter := ' ';  TS.DelimitedText := tmp;  for j := 0 to TS.Count - 1 do  begin  TS.Strings[j] := TS.Strings[j].Replace(' ', '').Replace('\n', '');  if (TS.Strings[j] <> '') then addToLeksem(getSpes(TS.Strings[j], true), TS.Strings[j]);  end;  end;  end;  procedure islink(a: integer);  begin  if a <> 0 then  begin  for i:= 0 to listTriad.Count - 1 do  begin  if (listTriad[i].Opers[1].choise = olink) and (listTriad[i].Opers[1].link = a) AND ((listTriad[i].TriadType = itsJmp) OR (listTriad[i].TriadType = itsIf)) then  begin  Form1.Memo2.Lines.Add('a' + (a+1).ToString() + ':');  break;  end;  if (listTriad[i].Opers[2].choise = olink) and (listTriad[i].Opers[2].link = a) AND ((listTriad[i].TriadType = itsJmp) OR (listTriad[i].TriadType = itsIf)) then  begin  Form1.Memo2.Lines.Add('a' + (a+1).ToString() + ':');  break;  end;  end;  end;  end;  function BinToDec (bin: string): integer;  var  j:integer;  error: boolean;  dec: integer;  begin  dec:= 0;  error:= false;  for j:= 1 to length(bin) do  begin  if (bin[j] <> '0') and (bin[j] <> '1') then  error := true;  if bin[J] = '1' then  dec := dec + (1 shl (length(bin) - j));  end;  if error then BinToDec:= 0  else BinToDec:= DEC;  end;  procedure genAsm();  var i, j: integer;  tmp: string;  begin  j:= 1;  Form1.Memo2.Lines.Add('code segment');  Form1.Memo2.Lines.Add('assume cs:code, ds:code');  Form1.Memo2.Lines.Add('org 100h');  Form1.Memo2.Lines.Add('');  for i:= 0 to 255 do if (arr1[i] <> '') then  begin  countident:= countident + 1;  Form1.Memo2.Lines.Add(arr1[i] + ' db 0');  Form11.StringGrid1.Cells[0,j]:=j.ToString;  Form11.StringGrid1.Cells[1,j]:=i.ToString;  Form11.StringGrid1.Cells[2,j]:=arr1[i];  inc(j);  end;  Form1.Memo2.Lines.Add('');  Form1.Memo2.Lines.Add('start:');  Form1.Memo2.Lines.Add('');  Form1.Memo2.Lines.Add('mov ax, cs');  Form1.Memo2.Lines.Add('mov ds, ax');  for i:= 0 to listTriad.Count - 1 do  begin  if listTriad[i].TriadType = itsPrisvoit then  begin  islink(i);  if (listTriad[i].Opers[2].choise <> olink) then Form1.Memo2.Lines.Add('mov ax, ' + BinToDec(inttostr(listTriad[i].Opers[2].conval)).toString());  Form1.Memo2.Lines.Add('mov ' + listTriad[i].Opers[1].vars + ', ax');  end;  if listTriad[i].TriadType = itsSub then  begin  islink(i);  if (listTriad[i].Opers[1].choise = oconst) then Form1.Memo2.Lines.Add('mov al, ' + BinToDec(inttostr(listTriad[i].Opers[1].conval)).toString())  else Form1.Memo2.Lines.Add('mov al, ' + listTriad[i].Opers[1].vars);  if (listTriad[i].Opers[2].choise = oconst) then Form1.Memo2.Lines.Add('mov bx, ' + BinToDec(inttostr(listTriad[i].Opers[2].conval)).toString())  else Form1.Memo2.Lines.Add('mov bx, ' + listTriad[i].Opers[2].vars);    Form1.Memo2.Lines.Add('sub ax, bx');  end;  if listTriad[i].TriadType = itsAdd then  begin  islink(i);  if (listTriad[i].Opers[1].choise = oconst) then Form1.Memo2.Lines.Add('mov al, ' + BinToDec(inttostr(listTriad[i].Opers[1].conval)).toString())  else Form1.Memo2.Lines.Add('mov al, ' + listTriad[i].Opers[1].vars);  if (listTriad[i].Opers[2].choise = oconst) then Form1.Memo2.Lines.Add('mov bx, ' + BinToDec(inttostr(listTriad[i].Opers[2].conval)).toString())  else Form1.Memo2.Lines.Add('mov bx, ' + listTriad[i].Opers[2].vars);    Form1.Memo2.Lines.Add('add ax, bx');  end;  if listTriad[i].TriadType = itsNop then  begin  islink(i);  Form1.Memo2.Lines.Add('mov ax, 4ch');  Form1.Memo2.Lines.Add('int 21h');  end;  if listTriad[i].TriadType = itsBolshe then  begin  //islink(i);  if (listTriad[i].Opers[1].choise = oconst) then tmp:= BinToDec(inttostr(listTriad[i].Opers[1].conval)).toString() + ', '  else tmp:= listTriad[i].Opers[1].vars + ', ';  if (listTriad[i].Opers[2].choise = oconst) then tmp:= tmp + BinToDec(inttostr(listTriad[i].Opers[2].conval)).toString()  else tmp:= tmp + listTriad[i].Opers[2].vars + ', ';  Form1.Memo2.Lines.Add('cmp ' + tmp);  Form1.Memo2.Text:= Form1.Memo2.Text + #13#10 + 'ja ';  end;  if listTriad[i].TriadType = itsMenshe then  begin  //islink(i);  if (listTriad[i].Opers[1].choise = oconst) then tmp:= BinToDec(inttostr(listTriad[i].Opers[1].conval)).toString() + ', '  else tmp:= listTriad[i].Opers[1].vars + ', ';  if (listTriad[i].Opers[2].choise = oconst) then tmp:= tmp + BinToDec(inttostr(listTriad[i].Opers[2].conval)).toString()  else tmp:= tmp + listTriad[i].Opers[2].vars + ', ';  Form1.Memo2.Lines.Add('cmp ' + tmp);  Form1.Memo2.Text:= Form1.Memo2.Text + 'jb ';  end;  if listTriad[i].TriadType = itsJmp then  begin  islink(i);  Form1.Memo2.Lines.Add('jmp a' + (listTriad[i].Opers[2].link + 1).toString());  end;  if listTriad[i].TriadType = itsIf then  begin  islink(i);  Form1.Memo2.Text:= Form1.Memo2.Text + 'a' + (i + 2).toString();  Form1.Memo2.Lines.Add('jmp a' + (listTriad[i].Opers[2].link + 1).toString());  Form1.Memo2.Lines.Add('a' + (i + 2).toString() + ':');  end;  end;  Form1.Memo2.Lines.Add('');  Form1.Memo2.Lines.Add('code ends');  Form1.Memo2.Lines.Add('end start');  end;  procedure TForm1.Button4Click(Sender: TObject);  var tmp, resul: string;  j, ggt: integer;  res: syntax;  tmpTree: TTreeNode;  begin  try  if (Edit1.Text <> '') OR (Memo1.Text <> '') then  begin  count:= 0;  hshcou:= 0;  trsrav:= 0;  lexem(Memo1);  res:=analisFromLex;  tmpTree := Form11.TreeView1.Items.Add(nil, res.toString);  makeTree(tmpTree, res);  tmpTree.Expand(True);  listTriad := triadlist.Create;  MakeTriadList(res, listTriad);  snd:=TStringList.Create;  optumDel(listTriad);  listTriad.vuvod(true);  Form1.Memo2.text:= '';  genAsm();  Form11.StringGrid3.Cells[0,0]:='Номер триады:';  Form11.StringGrid3.Cells[1,0]:='Тип:';  Form11.StringGrid3.Cells[2,0]:='Операнд 1:';  Form11.StringGrid3.Cells[3,0]:='Операнд 2:';  cnt55:= countident;  if (CheckBox1.Checked = true) or (CheckBox2.Checked = true) then Form11.ShowModal;  if Edit2.Text <> '' then Memo2.Lines.SaveToFile(Edit2.Text);  end;  except  Application.Terminate;  Exit;  end;  end;  procedure TForm1.FormCreate(Sender: TObject);  begin  hashmin:= Ord('0') + Ord('0');  hashmax:= Ord('z') + Ord('z');  arrLex[0, 0] := '>'; arrLex[0, 1] := '<'; arrLex[0, 5] := '<';  arrLex[0, 6] := '>'; arrLex[0, 7] := '<'; arrLex[0, 9] := '<';  arrLex[1, 2] := '<'; arrLex[2, 2] := '<'; arrLex[2, 3] := '=';  arrLex[2, 9] := '<'; arrLex[2, 10] := '<'; arrLex[2, 12] := '<';  arrLex[2, 13] := '<'; arrLex[2, 14] := '<'; arrLex[2, 15] := '<';  arrLex[3, 0] := '>'; arrLex[3, 1] := '<'; arrLex[3, 3] := '>';  arrLex[3, 4] := '='; arrLex[3, 5] := '<'; arrLex[3, 6] := '>';  arrLex[3, 7] := '<'; arrLex[3, 8] := '>'; arrLex[3, 9] := '<';  arrLex[3, 12] := '>'; arrLex[3, 13] := '>'; arrLex[3, 14] := '>';  arrLex[3, 15] := '>'; arrLex[4, 0] := '>'; arrLex[4, 1] := '<';  arrLex[4, 4] := '>'; arrLex[4, 5] := '<'; arrLex[4, 6] := '>';  arrLex[4, 7] := '<'; arrLex[4, 9] := '<'; arrLex[5, 0] := '<';  arrLex[5, 1] := '<'; arrLex[5, 5] := '<'; arrLex[5, 6] := '=';  arrLex[5, 7] := '<'; arrLex[5, 9] := '<'; arrLex[6, 0] := '>';  arrLex[6, 4] := '>'; arrLex[6, 6] := '>'; arrLex[6, 16] := '>';  arrLex[7, 2] := '<'; arrLex[9, 0] := '>'; arrLex[9, 3] := '>';  arrLex[7, 5] := '<'; arrLex[7, 7] := '<'; arrLex[8, 1] := '<';  arrLex[8, 5] := '<'; arrLex[8, 9] := '<'; arrLex[8, 9] := '<';  arrLex[9, 6] := '>'; arrLex[9, 11] := '='; arrLex[9, 12] := '>';  arrLex[9, 13] := '>'; arrLex[9, 14] := '>'; arrLex[9, 15] := '>';  arrLex[9, 17] := '<'; arrLex[10, 17] := '<'; arrLex[17, 4] := '=';  arrLex[10, 0] := '>'; arrLex[10, 3] := '>'; arrLex[10, 4] := '>';  arrLex[10, 6] := '>'; arrLex[10, 12] := '>'; arrLex[10, 13] := '>';  arrLex[10, 14] := '>'; arrLex[10, 15] := '>'; arrLex[11, 0] := '>';  arrLex[11, 2] := '<'; arrLex[11, 4] := '>'; arrLex[11, 6] := '>';  arrLex[11, 9] := '<'; arrLex[11, 10] := '<'; arrLex[11, 14] := '<';  arrLex[11, 15] := '<'; arrLex[12, 2] := '<'; arrLex[12, 3] := '>';  arrLex[12, 9] := '<'; arrLex[12, 10] := '<'; arrLex[12, 14] := '<';  arrLex[12, 15] := '<'; arrLex[13, 2] := '<'; arrLex[13, 3] := '>';  arrLex[13, 9] := '<'; arrLex[13, 10] := '<'; arrLex[13, 14] := '<';  arrLex[13, 15] := '<'; arrLex[14, 0] := '>'; arrLex[14, 2] := '<';  arrLex[14, 3] := '>'; arrLex[14, 4] := '>'; arrLex[14, 6] := '>';  arrLex[14, 9] := '<'; arrLex[14, 10] := '<'; arrLex[14, 12] := '>';  arrLex[14, 13] := '>'; arrLex[14, 14] := '>'; arrLex[14, 15] := '>';  arrLex[15, 0] := '>'; arrLex[15, 2] := '<'; arrLex[15, 3] := '>';  arrLex[15, 4] := '>'; arrLex[15, 6] := '>'; arrLex[15, 9] := '<';  arrLex[15, 10] := '<'; arrLex[15, 12] := '>'; arrLex[15, 13] := '>';  arrLex[15, 14] := '>'; arrLex[15, 15] := '>';arrLex[16, 1] := '<';  arrLex[16, 7] := '<'; arrLex[16, 9] := '<'; arrLex[16, 5] := '<';  arrLex[17, 1] := '<'; arrLex[17, 7] := '<'; arrLex[17, 9] := '<';  arrLex[3, 17] := '>'; arrLex[1, 17] := '='; arrLex[17, 0] := '>';  arrLex[8, 0] := '>'; arrLex[7, 8] := '='; arrLex[9, 4] := '>';  arrLex[0, 16] := '>'; arrLex[1, 9] := '<';  end;  procedure TForm1.open(Sender: TObject);  begin  if OpenDialog1.Execute then  begin  Edit1.Text:= OpenDialog1.FileName;  Memo1.Lines.LoadFromFile(OpenDialog1.FileName);  end;  end;  procedure TForm1.save(Sender: TObject);  begin  if SaveDialog1.Execute then  begin  Edit2.Text:= SaveDialog1.FileName;  end;  end;  initialization  for i:= 0 to 255 do arr1[i] := '';  end.  unit Unit11;  interface  uses  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Grids, Vcl.StdCtrls, Vcl.ComCtrls;  type  TForm11 = class(TForm)  PageControl1: TPageControl;  TabSheet1: TTabSheet;  StringGrid3: TStringGrid;  TabSheet2: TTabSheet;  StringGrid1: TStringGrid;  TabSheet3: TTabSheet;  TreeView1: TTreeView;  TabSheet4: TTabSheet;  StringGrid2: TStringGrid;  procedure FormShow(Sender: TObject);  procedure FormCreate(Sender: TObject);  procedure StringGrid3DrawCell(Sender: TObject; ACol, ARow: Integer;  Rect: TRect; State: TGridDrawState);  private  { Private declarations }  public  procedure add(a:string; b:string; var c:integer);  end;  var  Form11: TForm11;  cnt55: integer;  arr3\_1: array[0..255] of string;  arr4\_1: array[0..255] of string;  cnt1, count1, i: integer;  num: integer;  Nodes: array [0..255] of tTreeNode;  tmpwt, tmphg, count: integer;  snd: TStrings;  color: TColor;  implementation  {$R \*.dfm}  procedure TForm11.add(a:string; b:string; var c:integer);  begin  arr3\_1[c]:=a;  arr4\_1[c]:=b;  cnt1:=c;  end;  procedure TForm11.FormCreate(Sender: TObject);  var i: integer;  begin  count:= 1;  tmpwt:= Form11.Width;  end;  procedure TForm11.FormShow(Sender: TObject);  begin  num:= 0;  StringGrid1.RowCount:= cnt55 + 1;  StringGrid1.RowCount:=StringGrid1.RowCount + 1;  StringGrid1.Cells[0,0]:='Номер';  StringGrid1.Cells[1,0]:='Идентификатор';  StringGrid1.Cells[2,0]:='Имя переменной';  StringGrid2.RowCount:= cnt1 + 1;  StringGrid2.RowCount:=StringGrid2.RowCount + 1;  StringGrid2.Cells[0,0]:='Номер';  StringGrid2.Cells[1,0]:='Лексема';  StringGrid2.Cells[2,0]:='Текстовое представление';  count1:= 1;  for i := 0 to cnt1 - 1 do  begin  StringGrid2.Cells[0,count1]:=IntToStr(count1);  StringGrid2.Cells[1,count1]:=arr3\_1[i];  StringGrid2.Cells[2,count1]:=arr4\_1[i];  count1:=count1+1;  end;  end;  procedure TForm11.StringGrid3DrawCell(Sender: TObject; ACol, ARow: Integer;  Rect: TRect; State: TGridDrawState);  begin  if ARow = 0 then exit;  if (StringGrid3.Cells[1, ARow] = 'reit') or (StringGrid3.Cells[1, ARow] = 'const') then StringGrid3.Canvas.Font.Color := clSilver  else StringGrid3.Canvas.Font.Color := clBlack;  StringGrid3.Canvas.FillRect(Rect);  StringGrid3.Canvas.TextOut(Rect.Left + 3, Rect.Top + 3, StringGrid3.Cells[ACol, ARow]);  end;  end. |