Министерство образования и науки Российской Федерации
ФГАО УВО “Севастопольский государственный университет”

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторного практикума по дисциплине
“Теоретические основы построения компиляторов”
для студентов основного профиля
09.03.02 – “Информационные системы и технологии”
всех форм обучения

Севастополь
2018

УДК 004.423 + 453

Методические указания к выполнению лабораторно - вычислительного практикума по дисциплине “Теоретические основы построения компиляторов” для студентов основного профиля 09.03.02 – “Информационные системы и технологии” всех форм обучения / Разраб. В.Ю. Карлусов, С.А. Кузнецов. – Севастополь: Изд-во СевГУ, 2018. – 36 с.

Цель методических указаний: выработка у учащихся необходимых исследовательских и практических навыков применения основных положений теорий конечных автоматов, формальных грамматик и синтаксического анализа и компиляции при решении задач, связанных с построением трансляторов.

Методическое пособие рассмотрено и утверждено на заседании кафедры Информационных систем,
протокол № *10* от *18 мая* 20*18* г.

Рецензент Кожаев Е.А., кандидат техн. наук, доцент кафедры кибернетики и вычислительной техники.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Одним из немаловажных компонентов информационных технологий является компьютерный анализ синтаксиса текстов. К настоящему времени элементы его присутствуют практически повсеместно: в трансляторах языков программирования – от ассемблеров до объектно-ориентированных алгоритмических языков, контроля орфографии редактора “Word”, с использованием которого набраны настоящие методические указания, разнообразных переводчиков типа Prompt, РУМП и им подобных, наконец, квинтэссенции информационных систем – системах искусственного интеллекта. Поэтому трансляторы являются важной практической областью научных исследований, связанных с изучением ЭВМ[4].

В настоящем практикуме изучаются методы, ориентированные на использование теории формальных языков и грамматик при решении задач распознавания синтаксиса. Вопросы генерации объектного кода и его оптимизации опущены по следующим соображениям: наряду со значительными объёмами текстов соответствующих программ, алгоритмы генерации и оптимизации [6, 13] кодов обладают известной степенью инвариантности и универсализма, что, в немалой степени, препятствует формированию нетривиальных заданий при их изучении и исследовании. Последнее обстоятельство, как известно, толкает студента на путь формалистики и поверхностного изучения дисциплины.

Варианты практических заданий сформулированы на основании [8, 10, 14], примеры аналитических преобразований и вычислений, необходимых в ходе выполнения работ, помещены в [10 – 12].

ЛАБОРАТОРНАЯ РАБОТА № 1.
“ИССЛЕДОВАНИЕ ДЕТЕРМИНИРОВАННОГО КОНЕЧНОГО АВТОМАТА”

1. Цель работы

1.1. Научиться производить построение детерминированных конечных автоматов (ДКА), допускающих определённые цепочки символов языка.

1.2. Освоить приёмы описания конечных автоматов (КА) в виде графов, таблиц переходов и регулярных выражений.

1.3. Научиться выполнять построения ДКА по недетерминированным конечным автоматам (НКА).

1.3. Научиться проводить построение минимальных детерминированных конечных автоматов (МДКА).

2. Теоретические сведения

Конечным автоматом (КА), под которым, в дальнейшем, будем понимать конечный автомат Мили, формально называют совокупность следующих объектов [15, 16, 19]:

$$A\ (Q,\ \Sigma,\ \delta,\ q_0,\ F),$$

где Q – конечное множество состояния автомата; Σ - алфавит входных символов (букв или литер); δ - функция переходов КА, определяемая на декартовом произведении $Q \otimes \Sigma$; $q_0 \in Q$ – начальное состояние КА; $F \subset Q$ – множество заключительных состояний КА. Автомат функционирует на основании следующих эмпирических правил [15].

1. Исходным состоянием КА является начальное состояние q_0.

2. Цепочка, составленная из литер алфавита Σ, по одной букве поступает на вход КА. Возврат к начальным литерам цепочки не возможен.

3. Литера S_j допускается КА, находящимся в состоянии q_i, если $\delta(q_i, S_j) \neq \{\}$ – иными словами, образ функции переходов не пуст, определён переход из текущего состояния КА в следующее.

4. Если литера входной цепочки не допускается, то цепочка отвергается, признаётся не принадлежащей входному языку КА, сам автомат остаётся в текущем состоянии.

5. Цепочка литер допускается КА, если после поступления её последнего символа автомат оказывается в одном из заключительных состояний $q_r \in F$. В противном случае цепочка отвергается.

Известны теоремы [16], устанавливающие соответствие между классами конечных автоматов и классами цепочек, называемыми регулярными, которые могут быть проверены на принадлежность языку, порождаемому определённой грамматикой, с помощью КА.

Элементы КА могут быть построены и описаны несколькими способами [7 – 9, 15, 16], указанные процедуры излагаются и в методической литературе [10, с. 9 – 17; 11, с. 23 - 33].

Одним из эффективных способов описания КА является регулярное выражение, особенно когда известен вид цепочек, подлежащих опознаванию.

Правила написания регулярных выражений для типовых конструкций таковы [10, 16], при этом фигурные скобки обозначают итерацию, то, что в них заключено, может повторяться от нуля до бесчисленного числа раз. круглые или квадратные – используют для объединения альтернатив, из которых одна обязательно присутствует.

Пусть имеется алфавит составленный из символов (букв, литер) $\boldsymbol{V_T = \{x_1, x_2, \ldots, x_n\}}$.

1. Множество всевозможных цепочек, составленных из букв $\boldsymbol{x_i \in V_T}$:

$$\boldsymbol{L = \{x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n\}}.$$

2. Множество цепочек, составленных из литер $\boldsymbol{x_i \in V_T}$ и оканчивающихся литерой $\boldsymbol{x_1}$.

$$\boldsymbol{L = \{x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n\} \wedge x_1}.$$

3. Множество цепочек, составленных из литер $\boldsymbol{x_i \in V_T}$ начинающихся цепочкой $\boldsymbol{l_1}$ и оканчивающихся $\boldsymbol{l_2}$.

$$\boldsymbol{L = l_1 \wedge \{x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n\} \wedge l_2}.$$

4. Множество однолитерных цепочек (однобуквенных слов) совпадает с алфавитом

$$\boldsymbol{L = x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n}.$$

5. Множество двулитерных цепочек (двухбуквенных слов)

$$\boldsymbol{L = (x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n) \wedge (x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n)}.$$

6. Множество $\boldsymbol{m}$ – буквенных слов

$$\boldsymbol{L = (x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n) \underbrace{\wedge \ldots \wedge}_{m-2} (x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n)}$$

С точки зрения техники реализации КА на ЭВМ, предпочтительнее является его описание в виде таблиц переходов и выходов, иногда совмещённых, первая из них есть табличный аналог δ, символам входного алфавита соответствуют строки, а состояниям – столбцы.

Иногда функция переходов определена неоднозначно, в этом случае КА будет недерминированным. При этом необходимо получить детерминированный конечный автомат (ДКА) по недетерминированному КА. Приёмы построения ДКА по недетерминированному

КА, иллюстрируются многочисленными примерами, приводимыми в [10, с. 10 – 13]. Известен [10, 16, 19] следующий алгоритм построения ДКА по НКА.

1. Рассмотрим функцию переходов.

Для случаев, когда значение функции определено неоднозначно, введём новое состояние, которое является объединением состояний, образующих неоднозначность, в множество состояний автомата,.

2. Для вновь введенного состояния строим функцию переходов; анализируя функцию переходов в состояниях, определяющих данное состояние.

3. Повторяем п.п. 1 – 2 до тех пор, пока множество состояний автомата не будет изменено, а функция переходов приобретёт однозначность.

Технически оправдано иметь таблицу переходов КА минимального размера. Очевидно, что число строк таблицы изменению не подлежит, как определяемое внешними условиями, а число состояний КА и, соответственно, количество столбцов, может быть уменьшено в соответствии с теоремой Майхилла – Нерода [16]. Примеры такого упрощения приводятся в [10, с. 15 – 21; 11, с. 23 - 27]. Программная реализация КА может быть различной [1, с. 60 – 61, 86; 10, с. 15; 11, с. 26 - 37], тем не менее, неизменно наличие следующих этапов.

1. Начальное состояние – нулевое, соответствующий столбец – первый, он же является текущим.

2. Выполнение действий, связанных с обработкой текущего состояния КА.

3. Определение строки управляющей таблицы, соответствующей поступившей литере.

4. Определение следующего состояния КА и соответствующего номера столбца.

5. Пункты 2 – 3 выполняются, пока не будет достигнуто одно из заключительных состояний, после чего принимается решение о принадлежности или непринадлежности к одному из множеств цепочек.

3. Ход работы

Выполнение работы занимает два аудиторных занятия (4 академических часа) и предполагает интенсивную домашнюю подготовку.

3.1. Домашняя подготовка. На основании кода варианта в виде регулярного выражения, полученного у преподавателя, взять регулярное выражение из приложения А. Выполнить разметку регулярного выражения согласно [10], по необходимости минимизировать. Если КА не является детерминированным, необходимо привести его к детерминированному виду. Построить таблицу переходов МДКА. Выполнить минимизацию по таблице переходов. Построить граф конечного автомата для визуального определения недостижимых состояний, и, если таковые обнаружатся, окончательно минимизировать ДКА, построив окончательную таблицу МДКА. Написать черновой вариант программы на языке С (C^{++}).

3.2. Занятие 1 (2 часа). Предъявить результаты домашней подготовки для проверки преподавателю, устранить ошибки согласно замечаниям преподавателя. Ввести программу, реализующую КА, довести её до успешной компиляции.

3.3. Домашняя подготовка. Разработать тестовые последовательности литер, допускаемые и отвергаемые КА. Найти допускаемые последовательности минимальной длины. Предусмотреть соответствующие диагностические сообщения. Каждая цепочка литер должна быть снабжена соответствующей последовательностью вершин КА, которую он проходит в процессе анализа цепочки.

Подготовить пункты отчёта, касающиеся выполненной работы.

3.4. Занятие 2 (2 часа). Предъявление материалов тестирования преподавателю и отладка программы. Окончательное оформление отчёта и защита результатов выполнения лабораторной работы.

4. Содержание отчёта

4.1. Вариант задания.

4.2. Аналитические выкладки по проведению минимизации и, по необходимости построения ДКА, управляющая таблица КА.

4.3. Тестовые примеры для отладки включая допускаемые последовательности минимальной длины.

4.4. Листинги программы, результаты выполнения тестирования и граф конечного автомата, график числа синтаксических ошибок, выявленных на стадии компиляции программы в функции от номера постановки на компиляцию, график числа логических ошибок, выявленных в ходе отладки программы в функции от номера постановки на счёт.

4.5. Содержательные выводы, в которых отражено: изменение числа состояний КА в ходе выполнения лабораторной работы; анализ динамики выявленных ошибок; размер исходного модуля, размер исполнимого модуля.

5. Вопросы для самоконтроля и самоподготовки

5.1. Определение, способы описания и построения конечных автоматов.

5.2. Формулировки теорем о связи КА и формальной грамматики, соответствия недетерминированного КА и детерминированного КА.

5.3. Привести алгоритм построения ДКА по НКА.

5.4. Теорема Майхилла – Нерода.

5.5. Минимизация КА по регулярному выражению и таблицам переходов и выходов.

5.6. Что такое конечный автомат?

5.7. Какие гипотезы положены в основу функционирования КА?

5.8. Какой КА называется недетерминированным?

5.9. Правила написания регулярных выражений.

5.10. Какая цепочка входных литер допускается КА?

5.11. Какая цепочка входных литер не допускается КА?

5.12. В чём заключается отношение эквивалентности применительно к цепочкам, обрабатываемым конечным автоматом?

5.13. Почему для программной или схемной реализации КА необходимо его приведение к детерминированному виду?

5.14. Привести примеры задач, решаемых с применением КА[19].

ЛАБОРАТОРНАЯ РАБОТА № 2.
“ИССЛЕДОВАНИЕ СКАНЕРА ПРИ АНАЛИЗЕ ПРОСТЫХ ЯЗЫКОВЫХ КОНСТРУКЦИЙ”

1. Цель работы

1.1. Изучить принципы построения и программирования лексического анализатора на языке С (С$^{++}$) для простых языковых конструкций.

1.2. Получить навыки практического построения лексического анализатора (сканера) на основе теории конечных автоматов.

1.3. Освоить приёмы составления регулярных выражений для описания лексем.

1.4. Закрепить навыки построения минимального КА, осуществляющего сканирование текста программ.

2. Теоретические сведения

Из общей структуры компилятора [3 – 4; 5, с. 19] видно, что лексический анализатор (сканер) занимает заметное место в аналитической части транслятора и является первой (по времени начала выполнения) системной программой, анализирующей исходный модуль пользователя.

Сканер получил свое наименование благодаря особенности обработки входной информации. В его задачу входит: анализ исходной программы на уровне лексем [7, 9, 18]; выдача диагностических сообщений об обнаруженных ошибках; построение информационных таблиц констант, идентификаторов, служебных слов, разделителей; перекодирование исходной программы в соответствии с построенными таблицами.

Входной информацией сканера является литеры исходной анализируемой программы, а выходной - внутренняя, перекодированная форма программы и таблицы лексем [7]. Для построения сканера широко используют регулярные выражения и конечные автоматы, которые описываются регулярными (автоматными грамматиками) [15, 16, 19]. К сожалению, возможности регулярных языков ограничены. Поэтому роль грамматик, порождающих регулярные языки, сведена к распознаванию лексем, используемых в программе [15]. Указанный этап компиляции получил название лексического анализа.

Построение сканера наиболее формально и точно можно осуществить по регулярному выражению, при этом творческая компонента присутствует при составлении регулярного выражения, уступая затем место строгим алгоритмам. Примеры построения регулярного выражения и конечного автомата на его основе, осуществляющего процедуру лексического анализа, приводятся в [10, с. 22 – 24; 11, с. 33 - 36].

Возможен подход к построению сканера на основании диаграммы состояний (графа) автомата. Указанный подход проектирования является, по сути, эмпирическим, его ясное и подробное изложение выполнено Д. Грисом [7]. За исключением построения управляющей таблицы КА сканера, в нём соблюдается вся методология проектирования программных систем[13, 17], что вполне может быть использовано студентами при выполнении настоящей лабораторной работы.

3. Ход работы

Выполнение работы занимает два аудиторных занятия (4 академических часа) и предполагает интенсивную домашнюю подготовку.

3.1. Домашняя подготовка. Согласно выданному заданию составить регулярное выражение, описывающее КА, который осуществляет лексический анализ.

Структура такого выражения примерно такова

$\boldsymbol{Cc_1}L_1 \cup \boldsymbol{Cc_2}L1 \cup \boldsymbol{<iden>}L_1 \cup \boldsymbol{<data>}L_2 \cup \boldsymbol{d}_1 \cup \boldsymbol{d}_2 \cup \ldots \cup \boldsymbol{d}_j \cup \ldots \cup \boldsymbol{d}_n \cup L_3$.

В записи обозначено:

$\boldsymbol{Cc}_i$ – i-тое служебное слово.

<iden> – описание переменной.

<data> – описание константы или литерала.

L_1 – лексема, состоящая в том, что текущая литера не буква и не цифра.

L_2 – текущая литера не принадлежит к образующим константу или литерал.

$\boldsymbol{d}_j$ – разделители j-того типа.

L_3 – литера, не принадлежащая алфавиту конечного автомата.

В ходе домашних экзерциций составить эскизные выражения для описания дизъюнктивных компонент. Построить управляющую таблицу МДКА, Продумать модернизацию программы из предыдущей работы под нужды лексического анализа и текст диагностических сообщений, генерируемых в его ходе. Разработать схему кодирования лексем.

3.2. Занятие 1 (2 часа). Согласовать текст диагностических сообщений с преподавателем, сконструировать текст программы сканера на основании программы МДКА с учётом домашней подготовки. Выявить и устранить синтаксические ошибки.

3.3. Домашняя подготовка. Разработать тестовые примеры правильных и неправильных лексем языка, уточнить диагностические сообщения. Подготовить мероприятия по хронометрированию работы сканера. Продумать структуру выходной информации программы на предмет использования другими программными модулями компилятора. Подготовить пункты отчёта.

3.4. Занятие 2 (2 часа). Тестировать и окончательно отладить программу. Экспериментально определить время работы сканера для качественно различных структур строк и характера ошибок. Завершить оформление отчёта и защитить результаты выполнения лабораторной работы.

4. Содержание отчёта

4.1. Вариант задания.

4.2. Регулярное выражение КА, осуществляющего лексический анализ; аналитические преобразования, выполненные при построении детерминированного минимального КА; управляющая таблица КА.

4.3. Тестовые примеры и тексты диагностических сообщений.

4.4. Листинги сервисных процедур и результатов тестирования.

4.5. Статистика синтаксической и логической отладки программ, выполненная по аналогии п. 4.4. предыдущей работы.

4.6. Осмысленные выводы, в которых должно быть отражено: изменение объёмов (в символах) входной и выходной информации; результаты хронометража - среднее время, затрачиваемое на сканирование строки исходного текста; анализ динамики выявленных ошибок; размер исходного модуля, размер исполнимого модуля.

5. Вопросы для самоконтроля и самоподготовки

5.1. Назначение сканера и его место в схеме трансляции.
5.2. Типы лексем и система их кодирования.
5.3. Виды таблиц, создаваемых и используемых лексическим анализатором и способы их организации [7, 9, 18].
5.4. Характерные особенности функционирования лексического анализатора.
5.5. Методология построения лексических анализаторов.

ЛАБОРАТОРНАЯ РАБОТА № 3.
“ИССЛЕДОВАНИЕ АЛГОРИТМА ПРОСТОГО НИСХОДЯЩЕГО РАСПОЗНАВАТЕЛЯ”

1. Цель работы

1.1. Приобрести навыки описания синтаксиса алгоритмических языков средствами формальных грамматик.
1.2. Получить опыт разработки структуры данных, необходимых для хранения правил грамматики.
1.2. Выполнить оценку эффективности работы универсального нисходящего синтаксического анализатора.

2. Теоретические сведения

Простой нисходящий разбор является исторически одним из первых подходов к осуществлению автоматического синтаксического анализа. При выполнении алгоритма осуществляется попытка построения синтаксического дерева вывода [5; 7; 10, с. 6], и, если таковое удаётся построить, то анализируемая конструкция считается синтаксически корректной. Алгоритм обстоятельно описан в [7, с. 107 – 114], приводится программа на псевдоязыке [5, с. 95 - 108], отмечаются недостатки алгоритма и рассмотрена структура [7, с. 115 – 120] представления грамматики в виде синтаксического графа.

Функционирование алгоритма, в общих чертах, таково (смотри рисунок 3.1).

1. Используются два стека (магазина): магазин поставленных целей (МПЦ) для хранения нетерминальных символов, из которых, предположительно, выведена анализируемая конструкция языка, и магазин достигнутых целей (МДЦ) для хранения нетерминалов, для которых удалось установить такой вывод. Конструкция считается синтаксически корректной, если по окончании разбора в МДЦ будет находиться аксиома формальной грамматики.

2. В ходе обработки правил грамматики, в МПЦ помещаются нетерминальные символы, до тех пор, пока не будет достигнут терминальный.

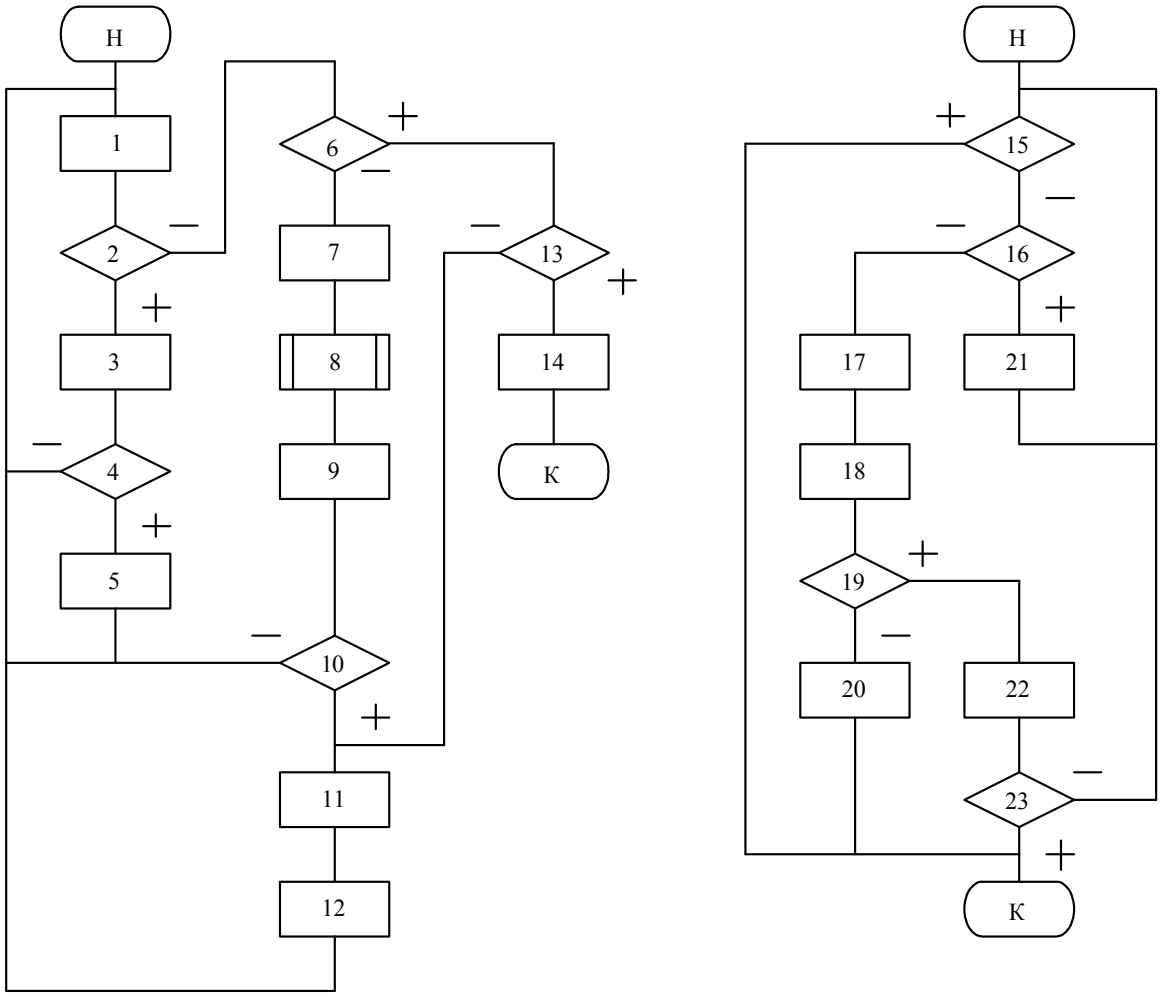

Рисунок 3.1 – Алгоритм простого нисходящего разбора

1. Чтение символа из правил грамматики $S \leftarrow G(z)$
2. Символ S – нетерминал?
3. Запись S в МПЦ
4. Есть ли продукция с S в левой части?
5. Переход на начало соответствующего правила
6, 10. Конец правила?
7. Чтение текущего символа из программы $T \leftarrow L\{G[z]\}$
8. Попытка идентификации терминала S из грамматики и терминала T из кодов программы, (блоки №№ 15 – 23)
9. Продолжение просмотра текущей поставленной цели
11. Перепись последнего символа из МПЦ в МДЦ
12. Возврат к глобальной позиции.
13. Конец анализируемой программы?
14. Перепись содержимого МПЦ в МДЦ
15. Терминалы из грамматики и программы совпали?
16. Есть ли альтернатива терминалу грамматики?
17. Вытолкнуть верхний элемент МПЦ
18. Вернуться на глобальную подцель
19. Есть ль альтернатива глобальной подцели?

20. Генерация сообщения об ошибке
21. Уход на альтернативу терминала
22. Уход на альтернативу глобальной подцели
23. Альтернатива – нетерминальный символ?

3. Этот терминал сопоставляется с очередным символом программы. Если символы несопоставимы, проверяют возможную альтернативу терминалу. Когда все проверки окончатся неудачей, то текущая цель выталкивается из МПЦ, и проверяется нетерминальная альтернатива у глобальной цели.

4. Если альтернативы исчерпаны, то фиксируется ошибка, в противном (удачном) случае удаётся достичь конца правила, и текущая поставленная цель считается достигнутой и переписывается МДЦ.

5. Когда достигнут конец текста, то МДЦ и исходная конструкция образуют синтаксическое дерево.

Входной информацией для анализатора является последовательность кодов, соответствующая тексту исходной программы, которую готовит сканер (смотри лабораторную работу № 2). Информация в синтаксическом графе представляется в этой же системе кодировки терминальных и нетерминальных символов, которая разработана ранее для сканера.

3. Ход работы

Выполнение работы занимает два аудиторных занятия (4 академических часа) и предполагает интенсивную домашнюю подготовку.

3.1. Домашняя подготовка. Для Вашего варианта задания, согласно рекомендации [2, 4, 5, 7 – 12], по образу и подобию [1] описания языков программирования в форме Бекуса – Наура, выполнить построение формальной грамматики. Особое внимание следует на устранение явной левой рекурсии в правилах, паче таковая непременно возникнет:

Исходные правила	Преобразованные правила
$\mathbf{A}\rightarrow a\|\mathbf{A}a$	$\mathbf{A}\rightarrow a\|a\mathbf{B}$
	$\mathbf{B}\rightarrow a\|a\mathbf{B}$.

Для большей устойчивости работы анализатора рекомендуется преобразовать грамматику в нормальную форму Грейбах [10, с. 37 – 41; 11, с.9 - 10; 16], правда, подобное преобразования приведёт к заметному увеличению количества правил грамматики и размера области хранения её синтаксического графа.

Составить текст программы, реализующей алгоритм простого нисходящего разбора. Продумать структуру синтаксического графа и диагностические сообщения, которые должны выдаваться при возникновении синтаксических ошибок.

3.2. Занятие 1 (2 часа). Предъявить формальную грамматику преподавателю, обсудить диагностические сообщения. Ввести программу в компьютер устранить синтаксические ошибки, добиться успешной компиляции.

3.3. Домашняя подготовка. Модифицировать грамматику согласно результатам обсуждения с преподавателем. Подготовить информацию для занесения в синтаксический граф. Продумать ход тестирования программы и разработать тесты [17]. Позаимствовать из предыдущей работы средства хронометража. Подготовить пункты отчёта.

3.4. Занятие 2 (2 часа). Согласовать тесты с преподавателем. Осуществить отладку логики нисходящего анализатора. Зафиксировать время, затрачиваемое анализатором на обработку тестов с разной синтаксической структурой. Предъявить материалы выполнения. Окончательно оформить отчёт и защитить результаты выполнения лабораторной работы.

4. Содержание отчёта

4.1. Вариант задания.

4.2. Формальная грамматика с правилами, свободными от левой рекурсии, порождающая фрагменты программы, соответствующие заданному варианту, дерево программы.

4.3. Разработанную структуру представления правил грамматики.

4.4. Тестовые примеры и тексты диагностических сообщений.

4.5. Листинг результатов тестирования.

4.5. Статистика синтаксической и логической отладки программ, аналогично предыдущим работам.

4.6. Осмысленные выводы, в которых должно быть отражено: минимальное, максимальное и среднее время, затрачиваемое на синтаксический разбор; размер исходного модуля, размер исполнимого модуля.

5. Вопросы для самоконтроля и самоподготовки

5.1. Назначение синтаксического разбора.

5.2. Принцип работы нисходящего распознавателя.

5.3. Основные недостатки простого нисходящего разбора.

5.4. Способы устранения левой рекурсии в правилах формальных грамматик.

5.5. Способы и особенности организации данных для осуществления нисходящего разбора.

ЛАБОРАТОРНАЯ РАБОТА № 4. “ИССЛЕДОВАНИЕ АЛГОРИТМА НИСХОДЯЩЕГО РАСПОЗНАВАТЕЛЯ ЯЗЫКА *LL(к)* - ГРАММАТИКИ”

1. Цель работы

1.1. Изучить процедуру синтаксического анализа, основанную на осуществлении рекурсивного спуска.

1.2. Приобрести навыки в исследовании грамматики на принадлежность к классу *LL(к)*.

1.3. Освоить методы синтеза формальных грамматик класса *LL(к)* и, по необходимости, преобразования грамматики к этому классу

2. Теоретические сведения

В качестве основного недостатка алгоритма простого нисходящего разбора указывается [7, с. 115 – 120] отсутствие прогноза при выборе возможного правила формальной грамматики из множества альтернатив для построения дерева. Следствием неправильного определения правила, гипотетически использованного при выводе и ведущего к неудаче, по-

следующее восстановление правильного хода вывода, приводит к непроизводительным затратам ресурсов ЭВМ и времени выполнения трансляции, варьируемым в широких пределах.

Таким образом, проблемой решаемой на каждом шаге разбора является определение очередного правила подстановки. В этом случае, если по *k* проанализированным символам исходной программы можно однозначно определить, какую продукцию грамматики необходимо использовать, грамматика называется *LL(к)* - грамматикой и, в этом случае, применяют алгоритм рекурсивного спуска [5, 16], а сам алгоритм разбора называют иногда прогнозирующим.

Одной из особенностей данной схемы трансляции является её жёсткая ориентация на конкретную грамматику, поэтому метод относят к группе специализированных методов. При этом на каждый символ формальной грамматики, как терминальный, так и нетерминальный, разрабатывается алгоритм, и пишется процедура его распознавания. Порядок вызова процедур определяется местоположением распознаваемых символов в правилах исходной грамматики. Поэтому алгоритм относят к группе методов синтаксических функций [2, 9, 18].

Исследование грамматики на принадлежность к классу *LL(к)* состоит в отыскании уникальных цепочек терминальных символов, выводимых из головы правой части правила. Изложение алгоритма, иллюстрация его применения и примеры для закрепления приводятся в [10, с. 24 – 26; 11, 38 – 40]. Там же приводится текст программы на C (C^{++}). Заметим, что рекурсивные вызовы при работе распознавателя могут быть организованы как явным, так и неявным способами. При кажущейся простоте алгоритма, он весьма эффективен и широко распространён: достаточно отметить, что компилятор с алгоритмического языка Pascal построен по данной схеме на основании свойств *LL(1)* – грамматики. Как недостаток можно отметить однопроходовость такого транслятора: его работа завершается нахождением первой встреченной от начала программы ошибки. Обычно отдают предпочтение *LL(1)* – грамматике, что упрощает процедуру обработки анализируемого фрагмента строки, а *LL(к)* - грамматики с большим индексом *k* используют вынужденно.

В качестве входной информации нисходящий рекурсивный распознаватель использует перекодированную программу [7, с. 19], принимаемую после работы сканера.

3. Ход работы

Выполнение работы занимает два аудиторных занятия (4 академических часа) и предполагает интенсивную домашнюю подготовку.

3.1. Домашняя подготовка. Для Вашего варианта построить формальную *LL(1)* – грамматику, свободную от левой рекурсии, либо позаимствовать её из предыдущей работы. Если грамматика не является *LL(1)* – грамматикой, то её необходимо преобразовать к такому виду [7]. Типичной проблемой, препятствующей отнесению грамматики к классу *LL(1)*, является наличие альтернативных правых частей правил с одинаковыми головами. Преобразование заключается во введении новых правил.

Исходные правила	Преобразованные правила
P→<*iden*>+<*data*>	**P**→<*iden*>**R**<*data*>
P→<*iden*>-<*data*>	**R**→+|-.

Рекурсия тоже препятствует причислению грамматики к этому классу. Преобразование заключается в использовании итерационного повторителя { }, который программно осуществляется итерационным циклом типа while или until.

Исходные правила	Преобразованные правила
<*spisok*>→<*iden*>;<*spisok*>|<*iden*>	<*spisok*>→<*iden*>{;<*iden*>}.

Составить текст программы, реализующей рекурсивный алгоритм разбора. Продумать диагностические сообщения, которые должны выдаваться при возникновении синтаксических ошибок.

3.2. Занятие 1 (2 часа). Предъявить формальную *LL(1)* – грамматику, совместно с доказательством её свойств, преподавателю. Ввести программу в компьютер, добиться успешной компиляции. В ходе отладки синтаксиса программы вести учёт синтаксических ошибок, выявляемых в ходе каждой компиляции.

3.3. Домашняя подготовка. Модифицировать грамматику согласно результатам обсуждения с преподавателем, если возникнет необходимость. Подготовить диагностические сообщения и определить места их размещения в тексте анализатора. Продумать ход тестирования программы и разработать тесты. Позаимствовать из предыдущей работы средства хронометража. Подготовить пункты отчёта.

3.4. Занятие 2 (2 часа). Согласовать тесты с преподавателем. Осуществить отладку логики рекурсивного анализатора. Зафиксировать время, затрачиваемое анализатором на обработку тестов с разной синтаксической структурой, сравнить временные характеристики с характеристиками простого нисходящего распознавателя. Предъявить материалы выполнения. Окончательно оформить отчёт и защитить результаты выполнения лабораторной работы.

4. Содержание отчёта

4.1. Вариант задания.

4.2. Формальная грамматика *LL(1)*, порождающая фрагменты программы, соответствующие заданному варианту.

4.3. Текст программы рекурсивного нисходящего распознавателя.

4.4. Тестовые примеры и тексты диагностических сообщений.

4.5. Листинг результатов тестирования .

4.5. Статистика синтаксической и логической отладки программ, полученная по аналогии предыдущих работ.

4.6. Осмысленные выводы, в которых должно быть отражено: среднее время, затрачиваемое на синтаксический разбор, насколько эффективнее данный алгоритм по сравнению с алгоритмом простого нисходящего распознавателя; размер исходного модуля, размер исполнимого модуля.

5. Вопросы для самоконтроля и самоподготовки

5.1. Формулировка теорем о преобразовании формальной грамматики в нормальную форму Грейбах [16].

5.2 Приёмы преобразования грамматик, направленные на устранения левой рекурсии.

5.3. Процедура определения принадлежности грамматики к классу *LL(к)*.

5.4. Сущность и особенности метода синтаксических функций.

5.5. Преимущества нисходящего разбора, основанного на свойствах *LL(к)* грамматик перед обычным нисходящим разбором.

ЛАБОРАТОРНАЯ РАБОТА № 5.
“ИССЛЕДОВАНИЕ АЛГОРИТМА ПРОСТОГО ВОСХОДЯЩЕГО РАСПОЗНАВАТЕЛЯ”

1. Цель работы

1.1. Изучение принципов построения и особенностей функционирования алгоритма простого восходящего разбора.

1.2. Закрепление навыков построения синтаксических анализаторов.

1.3. Оценка эффективности работы программы, реализующей алгоритм

2. Теоретические сведения

Термин “восходящий разбор” характеризует направление построения синтаксического дерева (дерева разбора), предполагая движение от сентенциальной формы (оператора) к вершине, которую представляет аксиома грамматики, предложения которой разбираются [2, 4 - 9, 13, 14, 16]. Простой восходящий разбор инвариантен, то есть обладает универсальностью по отношению к свойствам формальной грамматики, за исключением свойства однозначности. Так как при обработке предложения алгоритмического языка априори отсутствуют сведения о том, какие правила грамматики и в каком порядке использованы для его получения, то в ходе применения алгоритма не избежать использования проб и ошибок. Выбор простой фразы в качестве возможной основы разбора при неудаче обуславливает восстановление (полностью либо частично) исходного текста. Указанные действия в алгоритме соответствуют операциям подстановки и отхода, действие которых наглядно проиллюстрировано [5, с. 113]. Одним из удачных решений является алгоритм [5, с. 109 - 118]. Так же, как и все синтаксические анализаторы, он использует информацию, подготовленную сканером. Алгоритм простого восходящего разбора, изображённый на рисунке 5.1, включает следующие эвристики.

1. Языковая конструкция, проверяемая на правописание, анализируется “слева направо” посимвольно.

2. В ходе анализа все подстроки, от начала и до текущей позиции строки, сопоставляются с правыми частями продукций формальной грамматики.

3. При совпадении осуществляется подстановка, состоящая в замене совпавшей подстроки с левой частью (корнем) соответствующего правила. Таким образом, последний нетерминальный символ строки определяет последнюю выполненную подстановку.

4. Если, по достижении конца строки, в ходе выполнения подстановок, она трансформируется в аксиому, то анализируемая конструкция принадлежит языку, порождаемому формальной грамматикой, и, следовательно, является синтаксически правильной.

5. В противном случае, выполняется постепенный отход, состоящий в замене левого нетерминала, правой частью (аргументом) правила, его определяющего.

6. Чтобы избежать зацикливания, правила нумеруются, и номер использованного правила хранится вместе с нетерминалом, полученным в ходе подстановки, а выбор очередного правила производится в порядке возрастания номеров.

7. Если все подстановки не увенчались успехом, а исходная строка не содержит более основ, языковая конструкция признаётся ошибочной.

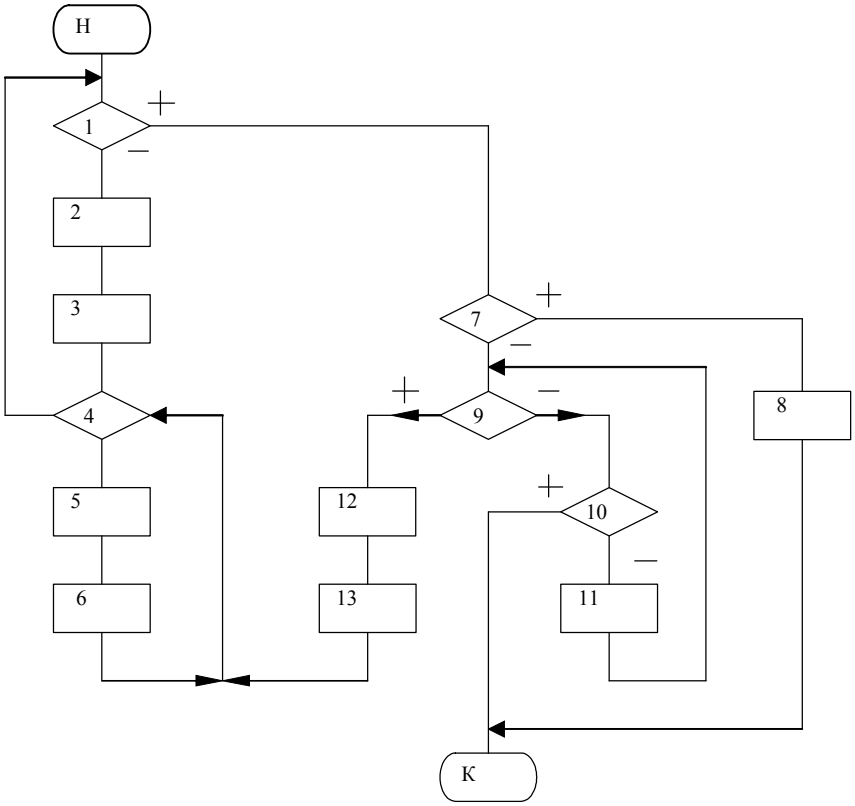

Рисунок 5.1 - Алгоритм простого восходящего разбора (SLR)

1. Стек ВХОД пуст?
2. Поместить в стек ХРАНЕНИЕ символ из ВХОД'а, номер правила Np положить равным нулю.
3. Поиск правила грамматики, номер которого больше Np, у которого правая часть (ПЧП) совпадает с содержимым стека ХРАНЕНИЕ.
4. Есть основа в стеке ХРАНЕНИЕ? (Найдено?)
5. Формирование дерева разбора: ДЕРЕВО←# (разделитель фраз), перепись содержимого стека ХРАНЕНИЕ в ДЕРЕВО.
6. Корень ≡ левая часть правила (ЛЧП)→ХРАНЕНИЕ, запомнить Np.
7. Аксиома на вершине ХРАНЕНИЕ?
8. Перепись аксиомы в дерево.
9. Нетерминал на вершине ХРАНЕНИЕ?
10. Стек ХРАНЕНИЕ пуст?
11. Переписать верхний элемент стека ХРАНЕНИЕ во ВХОД.
12. Восстановить Np и аргумент правила, соответствующего нетерминалу (отход).
13. Переписать содержимое ДЕРЕВО до символа # в ХРАНЕНИЕ.

В заключение заметим, что синтаксическое дерево, получаемое в результате разбора, помимо подтверждения правильности предложения, может быть, в случае ошибки, использовано для её нейтрализации [7, с. 353 -356], так, в частности, работали компиляторы PL/I IBM 360/370.

3. Ход работы

Выполнение работы занимает два аудиторных занятия (4 академических часа) и предполагает интенсивную домашнюю подготовку.

3.1. Домашняя подготовка. Алгоритм простого восходящего разбора является универсальным, поэтому к формальной грамматике не предъявляются иные требование, кроме непустоты порождаемого языка [16, 18] и однозначности самой грамматики [7,9]. Поэтому целесообразно переработать или разработать заново грамматику, составленную в ходе выполнения предыдущих работ, чтобы минимизировать число правил и объём синтаксического графа.

Взяв за основу алгоритм [5, с. 109 - 118], разработайте программу, его реализующую. Продумайте структуры хранения данных, поисковые процедуры, взаимодействие с лексическим анализатором (сканером).

3.2. Занятие 1 (2 часа). Обсудить грамматику с преподавателем, ввести программу в компьютер, добиться успешной компиляции.

3.3. Домашняя подготовка. Если необходимо, внести корректуры в правила формальной грамматики. Разработать тестовые примеры, диагностические сообщения. Позаимствовать из предыдущей работы средства хронометража. Подготовить пункты отчёта.

3.4. Занятие 2 (2 часа). Согласовать тесты с преподавателем. Осуществить отладку логики восходящего анализатора. Зафиксировать время, затрачиваемое анализатором на обработку тестов разной синтаксической структуры, сравнить временные характеристики простого восходящего анализатора с характеристиками простого и рекурсивного нисходящих распознавателей. Предъявить материалы выполнения. Окончательно оформить отчёт и защитить результаты выполнения лабораторной работы.

4. Содержание отчёта

4.1. Вариант задания.

4.2. Формальная грамматика, порождающая фрагменты программы, соответствующие заданному варианту.

4.3. Текст программы простого восходящего распознавателя.

4.4. Тестовые примеры и тексты диагностических сообщений, результаты тестирования.

4.5. Статистика синтаксической и логической отладки программ, полученная по аналогии предыдущих работ

4.5. Осмысленные выводы, в которых должно быть отражено: среднее время, затрачиваемое на синтаксический разбор, и его сопоставление с временными оценками других программ синтаксического разбора; размер исходного модуля, размер исполнимого модуля.

5. Вопросы для самоконтроля и самоподготовки

5.1. Преимущества и недостатки простого восходящего разбора.

5.2. Понятия подстановки и отхода.

5.3. Каким образом осуществляется поиск основы в ходе анализа.

5.4. Особенности структур данных, обеспечивающих функционирование алгоритма.

5.5. Возможен ли точный прогноз времени выполнения простого восходящего анализа?

ЛАБОРАТОРНАЯ РАБОТА № 6.
“ИССЛЕДОВАНИЕ АЛГОРИТМА ВОСХОДЯЩЕГО АНАЛИЗАТОРА ЯЗЫКА ГРАММАТИКИ ПРОСТОГО ПРЕДШЕСТВОВАНИЯ”

1. Цель работы

1.1. Изучить синтаксические анализаторы, базирующиеся на свойствах простого предшествования.

1.2. Освоить описание синтаксиса языков программирования грамматиками предшествования.

1.3. Получить исследовательские навыки построения отношений предшествования.

1.4. Освоить правила преобразования произвольной грамматики в грамматику предшествования

1.5. Исследовать временные показатели программы, реализующие алгоритм разбора предложений языка грамматики простого предшествования.

2. Теоретические сведения

Отношение простого предшествования между рядом стоящими символами алфавита (объединённого словаря) R и S позволяет определить аргумент продукции при поиске основы синтаксического разбора, так как указывает на одну из четырёх ситуаций: а) R и S стоят рядом в каком либо правиле; б) R – конец правила; в) S - начало правила; г) R и S несовместимы в синтаксически правильной программе [2, 5 – 9, 14].

Однозначные отношения существуют для класса грамматик простого предшествования, прилагательное “простой” нередко опускается. Если грамматика принадлежит к такому классу, то возможно построить эффективный распознаватель её языка, который, по отношению к обычному распознавателю, будет обладать повышенными скоростными характеристиками. Иногда представляется возможным построить функции предшествования (когда последние существуют) [7, параграф 5.4.]. В этом случае удаётся минимизировать размеры матрицы, которой представлены отношения предшествования, с размера $n \times n$ до $2 \times n$. Идеи, положенные в основание алгоритма разбора (рисунок 6.1), просты.

1. Символы последовательно просматриваются слева направо до выполнения отношения $\bullet>$. Его выполнение означает, что предпоследний символ является концевым в каком-то правиле формальной грамматики.

2. Правая часть правила определяется отношением эквивалентности $\equiv$ между символами, входящими в него, включая и предпоследний символ. Слева она ограничивается выполнением отношения $<\bullet$.

3. Выполняется подстановка, после чего часть строки сжимается, и проверяется отношение предшествования между текущим анализируемым и подставленным символами.

4. Процесс продолжается итерационно и завершается подстановкой аксиомы, в случае удачи, либо появлением пары символов, отношение между которыми не определено

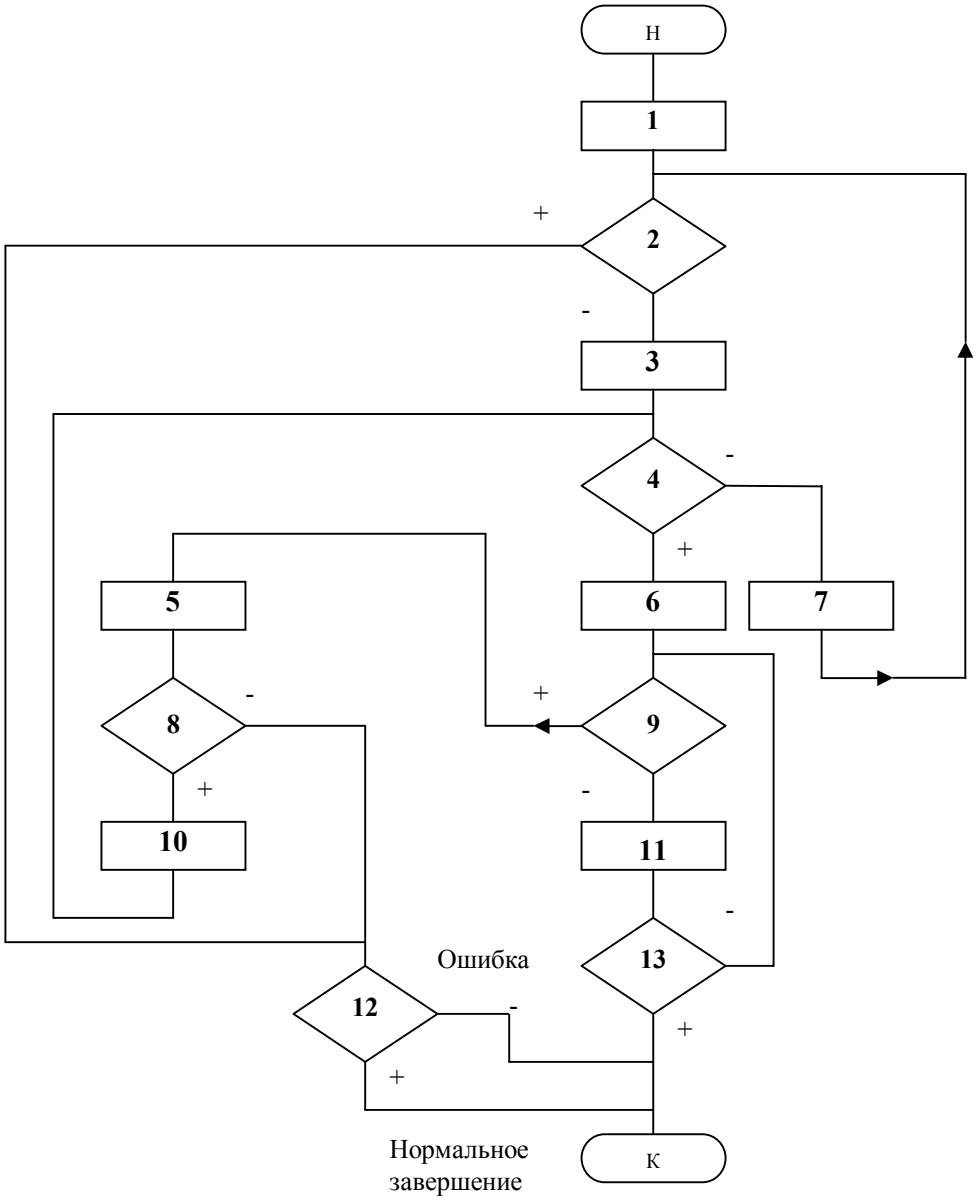

Рисунок 6.1 - Восходящий разбор, основанный на свойствах простого предшествования

1. i←1; МАГ$_i$ ← # (выталкиватель); R←#;
2. Конец разбираемой конструкции?
3. Чтение очередного символа программы S.
4. R •>S?
5. Поиск ПЧП вида МАГj…МАГi
6. j=i;
7. i++; МАГi←S; R←S
8. Найдено соответствующее правило?
9. МАГ$_{j-1}$ <0 МАГj?
10. i←j; МАГ$_i$ ← ЛЧП (подстановка); R←ЛЧП
11. j--;

12. На вершине магазина аксиома или символ #?
13. j=1?

В пояснениях к рисунку 6.1 употреблены сокращения: МАГ – магазин (стек); ЛЧП – левая часть правила; ПЧП – правая часть правила.

Содержимое магазина и его верхушка↓ ↓Текущий символ программы

	<iden>	=	<data>	*
< •	≡	< •	• >	←Отношения между символами

Рисунок 6.2 – Возможное состояние в ходе разбора

3. Ход работы

Выполнение работы занимает два аудиторных занятия (4 академических часа) и предполагает интенсивную домашнюю подготовку.

3.1. Домашняя подготовка. Для позаимствованной из предыдущих работ №№ 3 и 5 или специально разработанной формальной грамматики, построить отношения предшествования. Выполнение необходимых преобразований проиллюстрировано [10, с. 27 – 36; 11, с. 42 – 48]. Если отношение предшествования построить не удалось, необходимо выполнить стратификацию правил грамматики [7, с. 141 – 144]. Стратификация выполняется при неоднозначности отношений и наличии явной левой или правой рекурсии, например

Исходные правила	Преобразованные правила
P→…SU…	**P→…SW…**
U→U…	**W→U**
S≡U, **S< • U**	**U→U…**
	S≡W, **S< • U**

3.2. Занятие 1 (2 часа). Предъявить грамматику и ход построения отношений предшествования преподавателю, обсудить грамматику с преподавателем, ввести программу в компьютер, добиться успешной компиляции. В ходе отладки синтаксиса программы вести учёт синтаксических ошибок, выявляемых в ходе каждой компиляции.

3.3. Домашняя подготовка. Выполнить построение функций предшествования [7, с. 137 – 141; 10, с. 34 – 35; 11, с. 46 -48]. Если функции удалось построить, внести необходимые изменения в программу. Определить содержание и место диагностических сообщений. Разработать тестовые примеры. Позаимствовать из предыдущей работы средства хронометража. Подготовить отчёт.

3.4. Занятие 2 (2 часа). Предъявить тесты преподавателю. Осуществить отладку логики восходящего анализатора. Зафиксировать время, затрачиваемое анализатором на обработку тестов с разной синтаксической структурой, сравнить временные характеристики с характеристиками простого, рекурсивного нисходящих и простого восходящего распознавателей. Продемонстрировать выполнение преподавателю. Окончательно оформить отчёт и защитить результаты выполнения лабораторной работы.

4. Содержание отчёта

4.1. Вариант задания.

4.2. Грамматика, порождающая фрагменты программы, соответствующие заданному варианту. Выкладки, связанные с построением отношений и функций предшествования, матрицы и функции предшествования.

4.3. Текст программы восходящего распознавателя, основанного на свойствах простого предшествования.

4.4. Тестовые примеры и тексты диагностических сообщений, результаты тестирования.

4.5. Статистика синтаксической и логической отладки программ, полученная по аналогии предыдущих работ

4.5. Осмысленные выводы, в которых должно быть отражено: среднее время, затрачиваемое на синтаксический разбор, и его сопоставление с временными оценками других программ синтаксического разбора; анализ динамики выявленных ошибок; размер исходного модуля, размер исполнимого модуля.

5. Вопросы для самоконтроля и самоподготовки

5.1. Что представляют собой отношения простого предшествования?

5.2. Поясните свойства грамматики простого предшествования.

5.3. Способы и алгоритмы, применяемые при построении отношений.

5.4. Транзитивное замыкание отношений: сущность и построение. Алгоритм S. Warshall.

5.5. Функционирование алгоритма разбора на основании свойств грамматики простого предшествования.

ЛАБОРАТОРНАЯ РАБОТА № 7.
“ИССЛЕДОВАНИЕ АЛГОРИТМА ВОСХОДЯЩЕГО АНАЛИЗАТОРА ЯЗЫКА ГРАММАТИКИ ОПЕРАТОРНОГО ПРЕДШЕСТВОВАНИЯ”

1. Цель работы

1.1. Изучение синтаксического анализатора на основании свойств операторного предшествования.

1.2. Получить исследовательские навыки построения отношений операторного предшествования.

1.3. Приобрести опыт построения операторных грамматик и преобразования обычных грамматик к операторному виду.

1.4. Получить экспериментальные оценки показателей эффективности функционирования алгоритма.

2. Теоретические сведения

Между продукциями операторных грамматик и арифметическими выражениями существует некоторая аналогия [7, с. 146], если представить, что роли знаков операций играют терминальные символы, а роли операндов отведены нетерминалам. Подобно тому, как приоритеты старшинства знаков операций арифметического выражения определяют порядок его вычисления, так и терминальные символы должны определять порядок разбора. Формальная грамматика, в продукциях которой не стоят два нетерминала подряд, называется операторной [7], а если между терминалами определено не более одного отношения предшествования, то операторная грамматика называется грамматикой операторного предшествования. Сами отношения предшествования суть таковы: а) *r* и *s* стоят либо рядом, в каком ни будь правиле, либо их разделяет один нетерминал; б) *r* – находится в конце правила или перед концевым нетерминалом; в) *s* – находится в начале правой части правила или сразу же за начальным нетерминалом; г) *r* и *s* несовместимы в синтаксически правильной программе [2, 4 – 6].

По сравнению с грамматикой простого предшествования, грамматика с операторным предшествованием позволяет построить более компактный компилятор, за счёт того, что размер словаря терминальных символов грамматики меньше размера совокупного словаря. Алгоритм анализатора с операторным предшествованием [7] несколько сложнее, чем для анализатора с простым предшествованием, так как в ряде случаев проверку отношений приходится осуществлять через символ. Общая же структура алгоритмов разбора для грамматик простого и операторного предшествования подобна друг другу. Существующие отличия, определяются спецификой правил операторной грамматики.

3. Ход работы

Выполнение работы занимает два аудиторных занятия (4 академических часа) и предполагает интенсивную домашнюю подготовку.

3.1. Домашняя подготовка. Построить операторную грамматику, либо преобразовать обычную грамматику в операторную. Формально это можно сделать, заменяя отдельные нетерминалы терминалами, их определяющими.

Исходные правила	Преобразованные правила
P→<*iden*>**R**<*data*>	**P**→<*iden*>+<*data*>
R→+|-	**P**→<*iden*>-<*data*>

Построить отношения операторного предшествования. Построение отношений операторного предшествования качественно не отличается от построения отношения простого предшествования, а выделяется только объёмом вычислений. Разработать алгоритм анализатора языка грамматики с операторным предшествованием.

3.2. Занятие 1 (2 часа). Предъявить грамматику и ход построения отношений предшествования преподавателю, обсудить грамматику с преподавателем, ввести программу в компьютер, добиться успешной компиляции. В ходе отладки синтаксиса программы вести учёт синтаксических ошибок, выявляемых в ходе каждой компиляции.

3.3. Домашняя подготовка. Выполнить построение функций предшествования. Если функции удалось построить, внести необходимые изменения в программу. Определить содержание и место диагностических сообщений. Разработать тестовые примеры. Позаимствовать из предыдущей работы средства хронометража. Подготовить отчёт.

3.4. Занятие 2 (2 часа). Предъявить тесты преподавателю. Осуществить отладку логики восходящего анализатора. Зафиксировать время, затрачиваемое анализатором на обработку тестов разной синтаксической структуры. Сравнить временные характеристики полученного анализатора с характеристиками простого, рекурсивного нисходящих, простого и с учётом предшествования восходящих распознавателей. Продемонстрировать выполнение преподавателю. Окончательно оформить отчёт и защитить результаты выполнения лабораторной работы.

4. Содержание отчёта

4.1. Вариант задания.

4.2. Грамматика, порождающая фрагменты программы, соответствующие заданному варианту. Выкладки, связанные с построением отношений и функций операторного предшествования, матрицы и функции предшествования.

4.3. Текст программы восходящего распознавателя, основанного на свойствах операторного предшествования.

4.4. Тестовые примеры и тексты диагностических сообщений, результаты тестирования.

4.5. Статистика синтаксической и логической отладки программ, полученная по аналогии предыдущих работ

4.5. Осмысленные выводы, в которых должно быть отражено: среднее время, затрачиваемое на синтаксический разбор, и его сопоставление с временными оценками других программ синтаксического разбора; размер исходного модуля, размер исполнимого модуля.

5. Вопросы для самоконтроля и самоподготовки

5.1. Каким образом отношения операторного предшествования используются для определения простых фраз в ходе разбора?

5.2. Поясните свойства грамматики операторного предшествования.

5.3. Каким образом осуществляется построение отношений и функций операторного предшествования?

5.4. Как осуществляется функционирование алгоритма разбора на основании свойств грамматики операторного предшествования?

5.5. Приведите достоинства и недостатки алгоритма распознавания на основании операторного предшествования по сравнению с известными Вам распознавателями другого типа.

5.6. Дайте определения свойства грамматики операторного предшествования.

5.7. Что качественно означают отношения предшествования операторов?

5.8. Почему синтаксический анализатор, основанный на свойствах грамматики операторного предшествования, не в состоянии анализировать конструкции вида $\mathbf{P \rightarrow R}$?

5.9. Возможно ли построить функции операторного предшествования?

ЛАБОРАТОРНАЯ РАБОТА № 8.
“ИССЛЕДОВАНИЕ АЛГОРИТМОВ ТРАНСЛЯЦИИ АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ”

1. Цель работы

1.1. Изучение методов трансляции выражений.

1.2. Изучение принципов построения модуля подготовки к генерации объектного кода на примере арифметических выражений.

2. Теоретические сведения

Арифметические и логические выражения присутствуют практически во всех программах. Общность правил записи выражений, безотносительно синтаксиса и назначения языков программирования, породило специфические методы их трансляции. Суть трансляции арифметических выражений сводится к замене исходного выражения последовательностью тетрад или триад состоящих из кода операции и двух (триады) либо трех (тетрады) операндов. При этом известны следующие методы преобразования: скобочные (итеративные и рекурсивные) и бесскобочные, основанные на префиксной или постфиксной польской записи [4, 5, 14, 18, 20]. Все они обладают теми или иными присущими достоинствами и недостатками. Достаточно распространена схема Бауэра и Замельзона, представленная на рисунке 8.1.

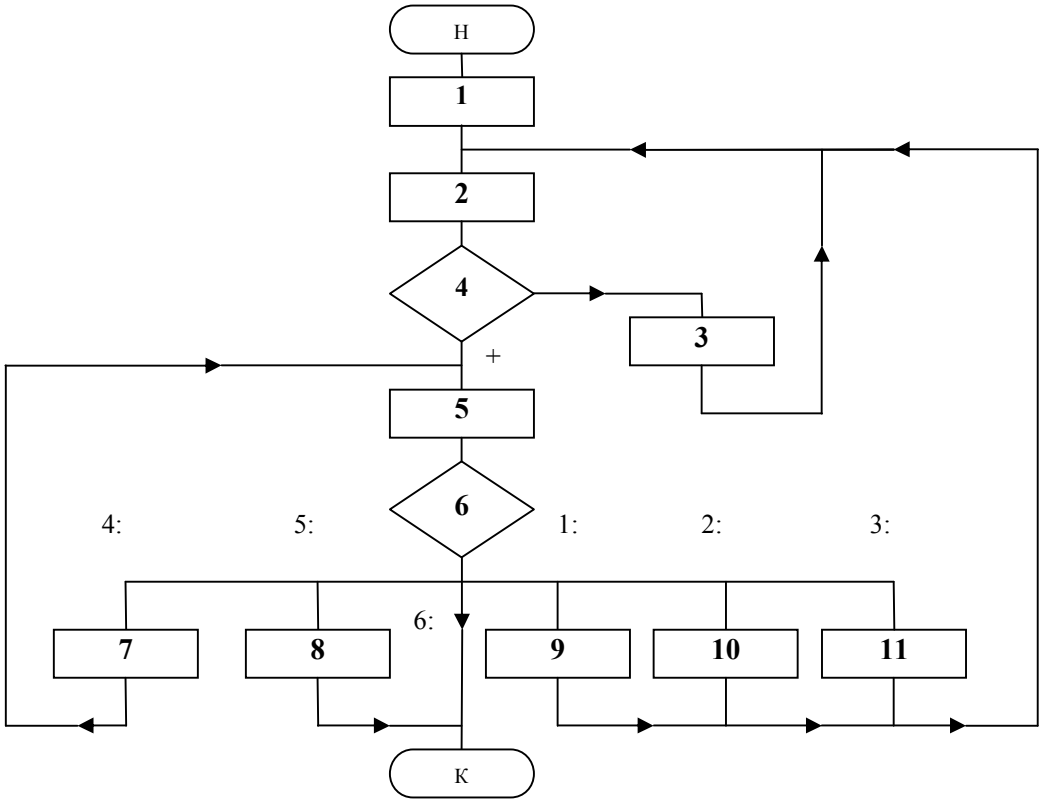

Рисунок 8.1 – Структурная схема алгоритма Бауэра и Замельзона

Цифрами на рисунке обозначены следующие операции.

1. Начальные установки : ***i=0***; ***j=0***; ***МАГ$_i$*** ← #;
2. Чтение очередного символа анализируемого выражения ***S***.
3. Помещение операнда в постфиксную строку: ***V$_j$=S***; *j*++;
4. ***S*** – разделитель?
5. Вычисление номера стековой операции ***N = T(S , МАГ$_i$)***
6. Оператор варианта (выбора) – номера соответствуют номерам стековых операций.
7. ***V$_j$=МАГ$_i$***; ***j***++; ***i--***;
8. Выдача диагностических сообщений.
9. ***i***++; ***МАГ$_i$*** ←***S***;
10. ***V$_j$*** ← ***МАГ$_i$***; ***МАГ$_i$*** ← ***S***; ***j***++;
11. ***i--***;

Литерами и сокращениями обозначено: ***МАГ*** – магазин для хранения разделителей ; ***V*** – постфиксная строка (результат преобразования) ; ***T*** – таблица управления стеком ; ***S*** – текущий символ программы.

Эффективность работы алгоритма обеспечивается использованием управляющей таблицы алгоритма (рисунок 8.2), которая неявно учитывает приоритеты операций, и позволяет проводить преобразование исходного выражения в ПОЛИЗ как при наличии скобок, так и в бесскобочной записи.

	#	(	+	-	*	/
#	6	5	4	4	4	4
(	1	1	1	1	1	1
+	1	1	2	2	2	2
-	1	1	2	2	2	2
*	1	1	1	1	2	2
/	1	1	1	1	2	2
)	5	3	4	4	4	4

Рисунок 8.2 – Управляющая таблица стековых операций

1. Помещение текущего разделителя в стек.
2. Обмен вершины стека и текущего разделителя местами, вершина помещается в постфиксную строку.
3. Удаление вершины стека с уничтожением её.
4. Удаление вершины стека с помещением в постфиксную строку.
5. Ошибка.
6. Заключительное состояние.

В процессе трансляции выражений, после его преобразования в ПОЛИЗ, используется следующая динамическая схема.

1. Отыскание знака, определяющего операцию.
2. Определение арности (количества операндов) операции.
3. Выбор операндов, относящихся к конкретной операции.
4. Генерация триады или тетрады, которые будут, в дальнейшем, использованы при построении исполнимого кода.

Указанные триады или тетрады выстраиваются в порядке, определённом приоритетами операций. Отсюда следует, что для работы необходимо знание приоритета каждой отдельной операции, независимо от того, какой тип преобразования выражения используется: в одном случае, руководствуясь старшинством операций, выставляются скобки, в другом – оп-

ределяется позиция польской инверсной записи (ПОЛИЗ). В зависимости от типа используемого алгоритма, задание отношения старшинства может быть как явным, так и неявным.

3. Ход работы

Выполнение работы занимает два аудиторных занятия (4 академических часа) и предполагает интенсивную домашнюю подготовку.

3.1. Домашняя подготовка. Выбрать задание на тип преобразования в дополнение к сквозному заданию для работ №№ 2 – 7, согласовать его с преподавателем. Изучить алгоритм преобразования, составить эскиз программы.

3.2. Занятие 1 (2 часа). Ввести программу в компьютер, добиться устранения синтаксических ошибок. В ходе отладки синтаксиса программы вести учёт синтаксических ошибок, выявляемых в ходе каждой компиляции.

3.3. Домашняя подготовка. Разработать тесты – выражения разной сложности и насыщенности, продумать взаимодействие с лексическим и синтаксическим анализаторами. Определить содержание и расположение диагностических сообщений. Подготовит пункты отчёта.

3.4. Занятие 2 (2 часа). Отладить программу, предъявить преподавателю. В процессе отладки программы фиксировать возникающие логические ошибки, а также синтаксические ошибки, появляющиеся в ходе внесения изменений. Оценить среднее число триад, приходящееся на одно выражение. Окончательно оформить отчёт, защитить работу.

4. Содержание отчёта

4.1. Вариант задания.

4.2. Структурная схема алгоритма преобразования.

4.3. Текст программы, соответствующий алгоритму.

4.4. Тестовые примеры и тексты диагностических сообщений, результаты тестирования.

4.5. Статистика синтаксической и логической отладки программ, полученная по аналогии предыдущих работ

4.5. Осмысленные выводы, в которых должно быть отражено: среднее число триад (тетрад), приходящихся на оператор языка высокого уровня; анализ динамики выявленных ошибок; размер исходного модуля, размер исполнимого модуля.

5. Вопросы для самоконтроля и самоподготовки

5.1. Принципы преобразования арифметических выражений, основанные на итерационных и рекурсивных подходах.

5.2. Основные подходы к трансляции выражений в скобочной записи.

5.3. Принципы осуществления процедуры трансляции выражений в бесскобочной записи.

5.4. Алгоритмы перевода инфиксной записи в префиксную или постфиксную формы записи.

5.5. В чём отличия префиксной, инфиксной и постфиксной форм записи выражения?

5.6. Особенности итеративного и рекурсивного способов преобразования выражений.

5.7. Тактика использования стеков (магазинов) при трансляции выражений.

5.8. Учёт приоритета операций в различных методах трансляции и преобразования.

ЗАКЛЮЧЕНИЕ

Предлагаемый Вашему вниманию лабораторно – вычислительный практикум охватывает весьма незначительную часть теоретической области, принадлежащей научной информатике и посвящённой вопросам построения трансляторов с языков программирования [2 – 9, 13]. Тем не менее, в ходе его выполнения закладывается фундамент, позволяющий ориентироваться в проблемных вопросах и продолжать дальнейшее самообразование, а приобретённые навыки, вне сомнения, позволят, руководствуясь [7, 8, 13, 18], писать собственные трансляторы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Алкок Д. Язык Паскаль в иллюстрациях [Текст] /Д. Алкок. – М.: Мир, 1991. – 192 с.
2. Ахо А. Теория синтаксического анализа, перевода и компиляции. [Текст]: Т.1. Синтаксический анализ / А. Ахо, Дж. Ульман. - М.: Мир, 1978. - 619 с.
3. Ахо А. Теория синтаксического анализа, перевода и компиляции. [Текст]: Т.2. Компиляция / А. Ахо, Дж. Ульман. - М.: Мир,1978. - 560 с.
4. Ахо А. Компиляторы: принципы, технологии и инструменты [Текст] / А. Ахо, Р. Сети, Дж. Ульман. . – М.: Издательский дом «Вильямс», 2002. – 512 с.
5. Вайнгартен Ф. Трансляция языков программирования [Текст] / Ф. Вайнгартен - М.: Мир, 1977. - 190 с.
6. Гордеев А.В. Системное программное обеспечение [Текст] / А.В. Гордеев, А.Ю. Молчанов. – СпБ.: Питер, 2003. – 736 с.
7. Грис Д. Конструирование компиляторов для ЦВМ [Текст] / Д. Грис - М.: Мир, 1975. - 544 с.
8. Компаниец Р.И. Основы построения трансляторов [Текст] / Р.И. Компаниец, Маньков Е.В., Н.Е. Филатов. – Спб.: Корона-принт, 2000, - 256 с.
9. Льюис Ф. Теоретические основы проектирования компиляторов [Текст] / Ф. Льис, Д. Розенкранц, Р. Стирнз. –М.: Мир, 1979. – 564 с.
10. Методические указания для студентов всех форм обучения специальности 07.080401 - "Информационные управляющие системы и технологии" при выполнении практических и контрольных работ по дисциплинам «Системное программирование» и "Основы дискретной математики" [Текст] / С.А Качур, В.Ю. Карлусов. - Севастополь: Изд-во СевГТУ, 1999. - 44 с.
11. Методические указания к выполнению практических занятий, индивидуальной подготовки, самоконтроля и проведению контрольных работ по дисциплине «Системное программирование» для студентов специальности 07.080401 -"Информационные

управляющие системы и технологии" всех форм обучения [Текст] / Разраб. В.Ю. Карлусов, С.А. Качур. - Севастополь: Изд-во СевНТУ, 2002. – 57 с.
12. Методические указания к выполнению контрольных работ по дисциплине “Операционные системы и системное программирование. Часть II. Системное программирование” для студентов заочной формы обучения по направлению 0804 – “Компьютерные науки” [Текст] / Разраб. В.Ю. Карлусов, Л.П. Старобинская. - Севастополь: Изд-во СевНТУ, 2007. – 32 с.
13. Молчанов А.Ю. Системное программное обеспечение. [Текст]: учебник для вузов / А.Ю. Молчанов. – СпБ.: Питер, 2006. – 396 с.
14. Молчанов А.Ю. Системное программное обеспечение. [Текст]: лабораторный практикум / А.Ю. Молчанов. – СпБ.: Питер, 2005. – 284 с.
15. Оллонгрен А. Определение языков программирования интерпретирующими автоматами [Текст] / А. Оллонгрен - М.: Мир, 1972. - 288 с.
16. Рейуорд-Смит В.Дж. Теория формальных языков. Вводный курс / В.Дж. Рейуорд-Смит- М.: Мир, 1986. - 128 с.
17. Тамре Л. Ведение в тестирование программного обеспечения [Текст] / Л. Тамре - М.: Издательский дом “Вильямс”, 2003. - 368 с.
18. Хантер Р. Основные концепции компиляторов [Текст] / Р. Хантер. - М.: Издательский дом “Вильямс”, 2002. - 256 с.
19. . Хопкрофт Дж. Введение в теорию автоматов, языков и вычислений [Текст] / Дж. Хопркрофт, Р. Мотвани, Дж. Ульман. – М.: Издательский дом “Вильямс”, 2002. – 528 с.
20. Шураков В.В. Математическое обеспечение ЭВМ [Текст] / В.В. Шураков. - Киев: Наукова думка, 1971. - 342 с.

ПРИЛОЖЕНИЕ А
(обязательное)

Варианты, по усмотрению преподавателя, могут быть выданы по номерам зачётной книжки с использованием таблицы А.1.

Таблица А.1 – Список вариантов задания к лабораторной работе № 1.

1-я цифра	2-я цифра									
	1	2	3	4	5	6	7	8	9	0
1	A∨B∨C	D∨EF	GH∨I	J∨K∨L	KL∨A	E∨A∨L	B∨KC	JD∨I	F∨H∨G	G∨EA
2	B∨CD	E∨F∨G	HI∨J	H∨EB	I∨A∨L	LB∨K	C∨J∨D	I∨FH	GH∨K	B∨E∨I
3	J∨FC	DG∨K	I∨A∨J	F∨GH	BD∨E	L∨CF	JA∨D	F∨G∨E	L∨FG	EH∨D
4	A∨D∨G	B∨IK	CL∨H	F∨J∨E	G∨BI	I∨C∨J	B∨KA	AL∨F	G∨E∨H	D∨IC
5	KC∨L	H∨FJ	E∨A∨D	L∨HF	JE∨A	JB∨K	J∨E∨K	A∨D∨F	H∨BI	LC∨G
6	D∨G∨B	I∨KC	AL∨J	F∨C∨D	G∨KH	G∨J∨E	K∨AD	FH∨B	I∨L∨C	C∨GJ
7	EB∨I	C∨D∨E	D∨IF	IA∨L	J∨F∨C	BA∨D	I∨C∨J	E∨FK	GL∨H	H∨B∨A
8	D∨GK	AL∨B	K∨C∨J	H∨GE	HK∨B	D∨IC	JE∨F	K∨G∨L	E∨KA	DF∨H
9	E∨I∨L	C∨FJ	AD∨G	D∨G∨H	K∨BE	B∨I∨L	L∨HB	AD∨I	C∨J∨E	F∨KG
0	IL∨C	F∨J∨A	H∨DI	CJ∨B	K∨A∨L	BE∨D	G∨L∨H	C∨IL	JA∨H	I∨C∨A

В таблице А.1 приняты следующие обозначения:

A	-	*ab{b∨a}*	G	-	*b{a∨ac}*
B	-	*{ab}c{c}*	H	-	*c{bb∨ba}*
C	-	*ca{b}*	I	-	*b{b}a*
D	-	*b{c∨a}*	J	-	*a{a∨ab}*
E	-	*{a∨c}b*	K	-	*b{ab}c{a}*
F	-	*c{a∨b}*	L	-	*b{ba∨bb∨bc}.*

ПРИЛОЖЕНИЕ Б
(обязательное)

Варианты, по усмотрению преподавателя, могут быть выданы по номерам зачётной книжки с использованием таблицы Б.1. Содержимое первой цифры варианта (описание синтаксиса идентификатора учебной программы) раскрывается в таблице Б.2, второй (описание синтаксиса констант) – в Б.3, третья (служебные слова, разделители и фрагменты программ) - в Б.4.

Таблица Б.1 – Список вариантов общего контрольного задания к лабораторным работам № 2 – 7 по теории синтаксического анализа.

1-я цифра	2-я цифра									
	1	2	3	4	5	6	7	8	9	0
1	1.1.15	14.8.2	3.8.8	13.7.4	15.6.5	16.4.3	17.3.6	1.2.7	2.1.8	3.8.9
2	12.2.10	2.7.11	13.3.12	4.2.13	5.7.14	6.6.1	9.5.1	10.4.9	13.3.2	4.2.4
3	8.3.5	11.6.6	3.4.14	9.1.7	12.6.8	5.7.9	11.4.9	12.3.10	14.2.11	5.1.12
4	7.4.14	10.5.13	6.5.15	4.8.1	14.8.2	11.7.4	15.6.5	16.5.6	17.4.3	6.3.7
5	17.8.8	1.7.9	2.6.10	3.7.1	5.2.2	15.1.4	8.4.14	1.2.5	2.3.6	7.4.10
6	16.7.7	11.4.14	15.5.8	14.6.9	13.5.10	6.7.11	16.6.12	10.8.13	3.1.14	8.2.1
7	7.6.2	8.3.4	9.6.5	10.5.6	11.3.7	12.4.10	7.5.8	17.6.9	11.7.10	4.8.11
8	6.5.12	5.2.13	4.7.14	3.4.1	2.1.2	1.2.4	17.3.5	8.4.3	1.5.6	12.6.7
9	9.4.10	10.1.8	11.8.9	12.3.10	13.7.11	14.8.12	15.1.13	16.2.14	9.3.1	2.4.9
0	8.3.2	7.2.4	7.1.5	6.2.6	5.5.7	4.6.8	3.7.9	2.8.10	1.6.11	10.8.12

Таблица Б.2- Типы идентификаторов

№ варианта	Описание типа
(1)	(2)
1	Содержит чередующиеся пары букв и цифр, заканчивается последовательностью символов "1"
2	Содержит чередующиеся буквы и цифры, заканчивается последовательностью символов "0"
3	Если встречается символ "a", то за ним всегда следует символ "c"
4	За встреченным символом "b" следует последовательность нулей и единиц
5	Оканчивается символом "m", которому не предшествует буква
6	Оканчивается символом "s", которому не предшествует цифра
7	Включает последовательность "abs"
8	Содержит последовательность символов "1" и оканчивается цифрой
9	Содержит последовательность символов "0" и оканчивается буквой
10	Оканчивается последовательностью цифр и включает символ “a”
11	Оканчивается последовательностью букв и включает последовательность "11"
12	Оканчивается символом "0" и включает последовательность "0100"
13	Следом за начальной буквой идентификатора следует "111"
14	Цифры, встречающиеся в идентификаторе, если они присутствуют в его записи, упорядочены по возрастанию
15	Цифры, включенные в состав идентификатора, сгруппированы в одну последовательность
16	В идентификаторе не встречается более трех букв, идущих подряд
17	В идентификаторе не встречается более трёх цифр, идущих подряд
18	Никакие две одинаковые буквы или цифры не стоят рядом

Таблица Б.3 - Типы констант

№ варианта	Тип	Пояснения	Формат
1	F	С фиксированной точкой	±000.000
2	E	С плавающей точкой	±000.000E±00
3	B	Двоичная	±10….001B
4	C	Символьная	'a…Z0…9'
5	X	Шестнадцатеричная	±0x12…9a…f
6	O	Восьмеричная	±0o012…7
7	D	Удвоенная, с плавающей точкой	±000.000D±000
8	L	Удвоенная, целая	±1001…L

Примечание. Для типа F задаются: максимальный размер целой части P и максимальный размер дробной части q. Для типов B, X, O, L - задается число цифр в записи. Тип E имеет максимально 6 цифр в мантиссе и две в показателе степени. Тип D имеет максимально 16 цифр в мантиссе и 3 в показателе степени.

Таблица Б.4 - Фрагмент программ для анализа

№ вари-анта	Служебные слова	Разделители		Фрагмент программы для анализа
		однолитерные	двулитерные	
(1)	(2)	(3)	(4)	(5)

1	BEGIN
	END
		;
		=
		*
		/
				BEGIN;
				x=4;
				B=4.5*x/3;
				END;

2	START STOP	* + , -	:=	START, x:=-0.5, y:=3*x+1, STOP
3	PUSK OFF	& ! =	\\	PUSK \\ x='ABC' \\ y=x&'A'!B \\ OFF
4	PRINT SCAN	, ; + - ( )		PRINT (A+B,'C'); SCAN (F)
5	IF THEN	( ) & + -	== :=	IF (A==B) & (C==0x0f) THEN K:=K+1;
6	DO WHILE END	= - + , ( )	<= >=	WHILE (K<=0o171) DO K=K+1, A=A-1, END,
7	FOR TO	( ) = ; * /		FOR (I=1) TO 4 (K=K*5; C=C/2E+01;);

8	CASE
	SWITCH
		{
		}
		(
		)
		+
		-
		;
		=
		:
				K=5;
				C=5+x;
				SWITCH(C)
				{
				case 1: K=K+1;
				case 2: K=K+2;
				}

Таблица Б.4 – продолжение.

(1)	(2)	(3)	(4)	(5)
9	CHAR INTEGER	, ; = ( )	!! << >>	CHAR (A,B); INTEGER (C); A='SANTAS DUMON'; B=A!!A;
10	PROC END	= ,	&& !! << >>	PROC, A=0xA1011, B=0xae13, C=A&&B, END,
11	READ WRITE	( ) * / - + , $		WRITE (A*B, C+23) $ READ (D) $
12	IF THEN ELSE	< > = + - ( ) & :		IF (A>B & C<0) THEN (A=A+B+7) ELSE (A=A-B; C=D+B);
13	DO UNTIL	( ) = + - * / < > #	<>	DO x=3+y # z=2/x+1 C=K*C+1 UNTIL(z<>0.01)#

14	FUNC
	FINAL
		:
		\
		!
		{
		}
		(
		)
			:=
			&&
			!!
				A: FUNC (x)\
				{
				X:=(x&&0xfe)\
				A:=!(x!!0x13)\
				}
				FINAL

15	WAIT SIGNAL	( ) > = + %	>= <=	WAIT(S) K=K+5.3 % C=K+C+1 % SIGNAL(S<=0xf)

Таблица Б.4 – продолжение.

(1)	(2)	(3)	(4)	(5)
16	FOR TO NEXT	( ) = ; + - * /		FOR I=0 TO 8 K=K+5*(I-3); C=C/2E+01; NEXT I
17	FORALL DO END FORALL	- + , * / ( )	:=	FORALL X DO X:=A+0o17*(B+2), C=K/C+1, END FORALL
18	SUB ENDSUB	: ( ) = ; + - * /	:=	MODUL1: SUB(A, B, C); F:=A+B*7; C:=(F+3.14)*A; ENDSUB;

Примечание – Вид разбираемой инструкции, приводимой в графе (5) таблицы Б.4, весьма условен в части идентификаторов и констант. Они должны находиться в полном соответствии с вариантом, приводимым в таблицах Б.2 и Б.3.

ПРИЛОЖЕНИЕ В
(обязательное

Варианты, по усмотрению преподавателя, могут быть выданы по последней цифре номера зачётной книжки с использованием таблицы В.1.

Таблица В.1. – Метод преобразования выражения

№ варианта	Содержание задания
(1)	(2)
1	Преобразование выражения в скобочной форме записи в последовательность тетрад (триад) по алгоритму Рутисхаузера
2	Преобразование выражения в скобочной записи в префиксную польскую запись
3	Преобразовать выражение в скобочной форме записи в постфиксную польскую запись
4	Преобразовать выражение в скобочной форме записи в последовательность тетрад (триад) с использованием алгоритма Бауэра и Замельзона
5	Преобразовать выражение в бесскобочной записи в польскую префиксную запись
6	Преобразовать выражение в скобочной форме записи в последовательность триад (тетрад) рекурсивным методом
7	Преобразовать выражение в бесскобочной форме записи в польскую инверсную запись
8	Преобразовать выражение в смешанной форме записи в последовательность триад (тетрад) по алгоритму Бауэра и Замельзона
9	Преобразовать префиксную форму записи в последовательность триад (тетрад)
0	Преобразовать постфиксную польскую запись в последовательность тетрад (триад)

Заказ № от “ “ 2010 . Тираж экз.
Изд-во СевНТУ