Федеральное государственное бюджетное образовательное учреждение
высшего образования «Кемеровский государственный университет»
Новокузнецкий институт (филиал)

Факультет информатики, математики и экономики
Кафедра информатики и вычислительной техники им. В. К. Буторина

О. А. Штейнбрехер, О.И. Новоселова

Программирование. Часть 3

Методические указания по выполнению практических работ по дисциплине «Программирование» для обучающихся по направлению подготовки 09.03.03 Прикладная информатика Профиль «Прикладная информатика в экономике»

Новокузнецк
2020

УДК [378.147.88:004.4](072)
ББК 74.484(2Рос-4Кем)я73+ 32.972я7
Ш88

Ш88 «Программирование. Часть 3. Методические указания по выполнению практических работ по дисциплине: метод. указ (текст. электрон. изд.)/ О.А. Штейнбрехер, О.И. Новоселова ; Новокузнец. ин-т (фил.) Кемеров. гос. ун-та – Новокузнецк: НФИ КемГУ, 2020. – 29 с.

Приводятся методические указания по выполнению практических работ по дисциплине «Программирование» для третьего семестра обучения.

Методические указания предназначены для студентов всех форм обучения направления 09.03.03 «Прикладная информатика».

Рекомендовано
на заседании кафедры
информатики и вычислительной техники им. В. К. Буторина
23 ноября 2020 года.
Заведующий кафедрой

А. В. Маркидонов

Утверждено
методической комиссией факультета
информатики, математики и экономики
17 декабря 2020 года.
Председатель методкомиссии

Г.Н. Бойченко

УДК [378.147.88:004.4](072)
ББК 74.484(2Рос-4Кем)я73+ 32.972я7

Текст представлен в авторской редакции

Оглавление

ПРЕДИСЛОВИЕ

Дисциплина «Программирование» относится к основному разделу образовательной программы. Дисциплина изучается в течение трех семестров.

В результате освоения данной дисциплины у обучающегося должны быть сформированы компетенции основной профессиональной образовательной программы бакалавриата (далее - ОПОП):

ОПК-7 – Способен разрабатывать алгоритмы и программы, пригодные для практического применения.

В результате обучения обучающийся должен:

знать:

- классификацию программных средств, языков программирования;
- основные парадигмы программирования;
- понятия и методы алгоритмизации;
- основы и методы структурного программирования;
- основные понятия объектно-ориентированного программирования;
- основы теории алгоритмов и основы теории сложности;

уметь:

- разрабатывать алгоритмы для решения прикладных практических задач;
- разрабатывать программы для реализации прикладных практических задач;
- обосновывать выбор стандартных алгоритмов для решения практических задач;
- осуществлять выбор стандартных средств для программной реализации алгоритмов и программ;

владеть:

- методами алгоритмизации, оценки сложности алгоритмов;
- графическим способом описания алгоритмов;
- методами структурного программирования;
- навыками реализации алгоритмов и программ, с учетом сложности алгоритмов

Первый семестр посвящен вопросам алгоритмизации, теории алгоритмов, моделям вычисления и основам программирования. Для изучения обучающимся предполагается изучение C++. Второй семестр посвящен изучению нескалярных типов данных языков программирования. Кроме этого, во втором семестре предполагается изучение основ языка C#. Выбор его вторым языком обусловлен родственым синтаксисом.

Третий семестр предполагает создание многооконных приложений и изучение механизма работы транслятора.

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Основы объектно-ориентированной парадигмы

Объектно-ориентированная парадигма и соответствующее ей объектно-ориентированное программирование (ООП) представляют собой подход к процессу разработки программного обеспечения. Данные при этом подходе рассматриваются как активные «объекты», а не как пассивные единицы, представленные в обычной императивной парадигме.

Основными единицами ООП являются объект и класс (рисунок 1). Объект — элементарная сущность, описываемая определенными свойствами (хранящимися в виде атрибутов объекта) и поведением (реализованным в виде методов). Класс описывает структуру свойств и поведения одного типа объектов. Каждый объект программы является экземпляром некоторого класса.

Рисунок 1 – Схема ООП

Рассмотрим базовые принципы ООП. Инкапсуляция - свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе (свойство языка программирования, позволяющее пользователю не задумываться о сложности реализации используемого программного компонента, а взаимодействовать с ним посредством предоставляемого интерфейса, а также объединить и защитить жизненно важные для

компонента данные; при этом пользователю предоставляется только спецификация (интерфейс) объекта).

Наследование – позволяет создать новый класс на основе уже существующего, частично или полностью заимствуя его функциональность (позволяет описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом).

Полиморфизм – использование объектов с одинаковым интерфейсом без информации о типе и внутренней структуре объекта (возможность объектов с одинаковой спецификацией иметь различную реализацию).

Абстракция - в объектно-ориентированном программировании это придание объекту характеристик, которые отличают его от всех объектов, четко определяя его концептуальные границы.

Windows-form

Windows Forms — название интерфейса программирования приложений (API), отвечающего за графический интерфейс пользователя и являющегося частью Microsoft .NET Framework. Windows Forms упрощает доступ к визуальным компонентам (виджетам) Microsoft Windows за счет создания обертки для существующего Win32 API в управляемом коде. Управляемый код является машинно-независимым и не зависит от языка разработки. То есть программист одинаково может использовать Windows Forms как при написании ПО на C#, C++, так и на VB.Net, J# и др.

Основной класс Windows Forms – форма, экземплярами которого являются главные и диалоговые окна.

Формы являются потомками класса Form, определенного в пространстве имен System.Windows.Forms. Интерфейс Windows Forms позволяет работать в режиме конструктора, добавляя элементы управления (кнопки, поля для ввода текста, поля для отображения текста, меню и прочие компоненты, характерные для Windows-приложений) простым «перетаскиванием». В коде они представлены как поля класса формы.

Все элементы управления окна являются объектами классов, содержащихся в System.Windows.Forms и являющихся потомками базового класса Control. Ответная реакция программы на определенное действие (событие), связанное с элементом управления – обработчик события - оформляется в виде метода формы.

На рисунках 2 и 3 представлены форма и структура соответствующих классов. Ниже представлено описание классов, атрибутов и методов.

System.Windows.Forms.Form

Представляет окно или диалоговое окно, которое составляет пользовательский интерфейс приложения.

Свойства:

Text – заголовок окна.

System.Windows.Forms.ListBox

Представляет элемент управления Windows для отображения списка позиций. Данный элемент управления лучше всего подходит для отображения коллекций текстовых данных.

Свойства:

Enabled – указывает, может ли элемент управления отвечать на действия пользователя.

Items - получает позиции элемента управления.

DataSource - получает или задает источник данных.

DisplayMember - получает или задает свойство отображения. В данном случае свойство Info класса CPolymorph.

Методы:

SetSelected - получает или задает свойство отображения.

System.Windows.Forms.TextBox

Предоставляет элемент управления "текстовое поле" Windows. Данный элемент управления лучше всего подходит для ввода текстовой информации.

Свойства:

Text - получает или задает текущий текст в текстовом поле

Методы:

Clear – очищает текстовое поле.

System.Windows.Forms.Button

Представляет элемент управления Windows "Кнопка".

Свойства:

Text – текст на кнопке.

Использовалось также событие Click.

System.Windows.Forms.Label

Представляет стандартную надпись Windows.

Свойства:

Text – содержание надписи.

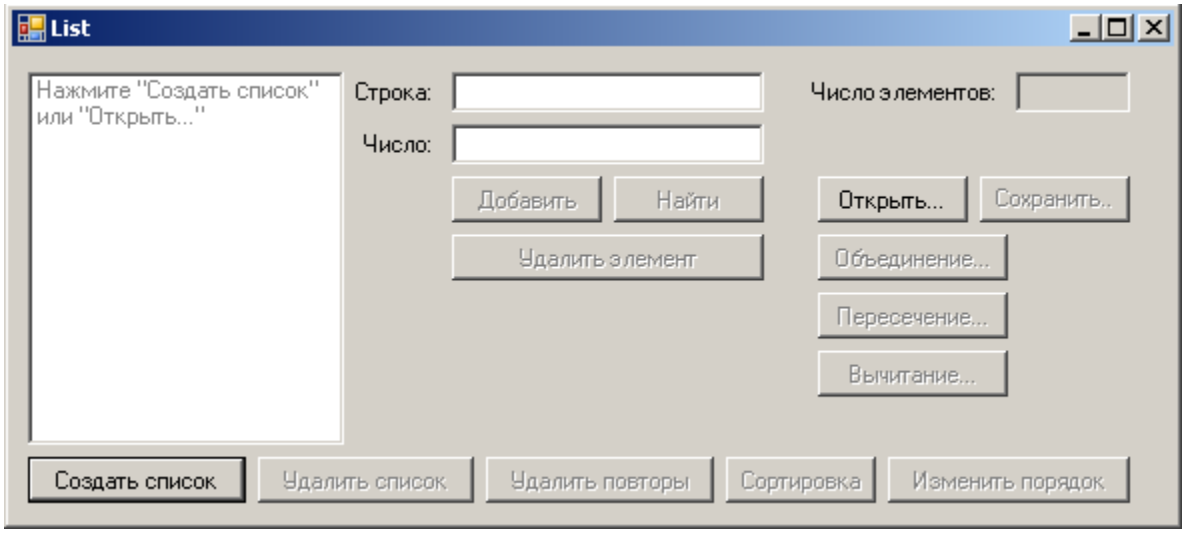

Рисунок 2 – Пример формы

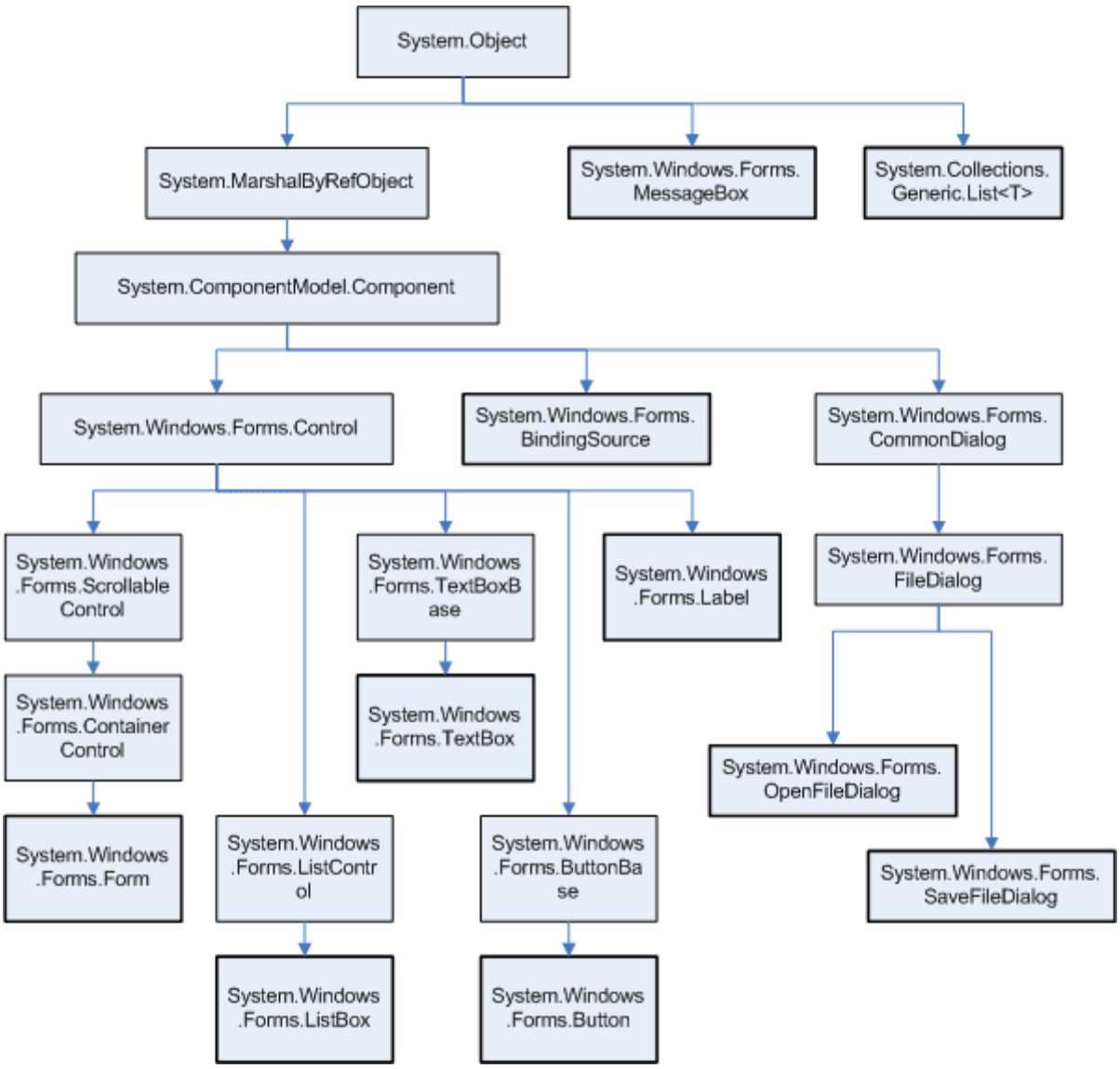

Рисунок 3 – Структура классов

Трансляция

Под трансляцией в самом широком смысле можно понимать процесс восприятия компьютером программы, написанной на некотором формальном языке.

Компиляция (рисунок 4) - преобразование объектов (данных и операций над ними) с входного языка в объекты на другом языке для всей программы в целом с последующим выполнением полученной программы в виде отдельного шага.

Интерпретация (рисунок 5) - анализ отдельного объекта на входном языке с одновременным выполнением (интерпретацией).

Следовательно, компиляция и интерпретация отличаются не характером и методами анализа и преобразования объектов программы, а совмещением фаз обработки этих объектов во времени. То есть при компиляции фазы преобразования и выполнения действий разнесены во времени, но зато каждая из них выполняется над всеми объектами программы одновременно. При интерпретации, наоборот, преобразование и выполнение действий объединены во времени, но для каждого объекта программы.

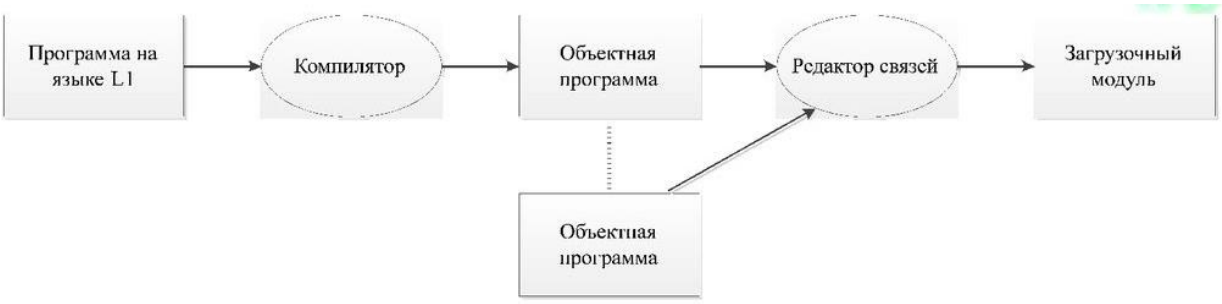

Рисунок 4 – Схема компилятора

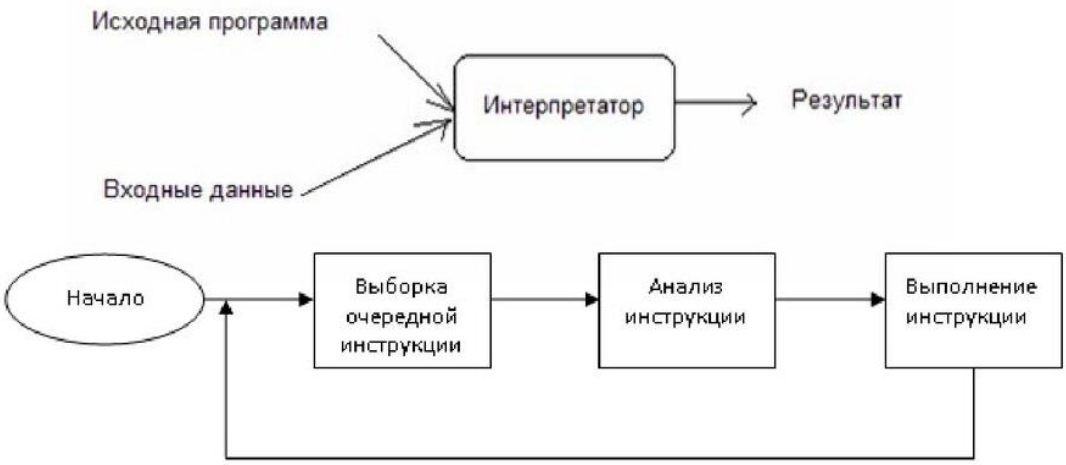

Рисунок 5 – Схема интерпретатора

Интерпретатор непосредственно выполняет действия, связанные с определением или преобразованием объектов программы, а компилятор - переводит их на другой (не обязательно машинный язык). Отсюда можно сделать несколько выводов:

- для выполнения программы, написанной на определенном формальном языке после ее компиляции необходим интерпретатор, выполняющий эту программу, но уже записанную на выходном языке компилятора;
- процессор и память любого компьютера (а в широком смысле и вся программная среда, создаваемая операционной системой, является интерпретатором машинного кода);
- в практике построения трансляторов часто встречается случай, когда программа компилируется с входного языка на некоторый промежуточный уровень (внутренний язык), для которого имеется программный интерпретатор.

Выходной язык компилятора может быть машинным языком для компьютера с другой архитектурой, нежели тот, в котором работает компилятор. Такой компилятор называется кросс-компилятором, а сама система программирования кросс-системой программирования. Такие системы используются для разработки программ для архитектур, не имеющих собственных операционных систем или систем программирования (контроллеры, управляющие микропроцессоры).

Трансляция включает в себя три основных фазы: лексический, синтаксический и семантический анализ. Выделяют также предварительную фазу – препроцессор. Препроцессор не имеет никакого отношения к языку. Это предварительная фаза трансляции, которая выполняет обработку текста программы, не вдаваясь глубоко в ее содержание. Он производит замену одних частей текста на другие, при этом сама программа так и остается в исходном виде.

Трансляция начинается с лексического анализа программы. Лексика языка программирования - это правила «правописания слов» программы, таких как идентификаторы, константы, служебные слова, комментарии. Лексический анализ разбивает текст программы на указанные элементы. Особенность любой лексики -ее элементы представляют собой регулярные

линейные последовательности символов. Например, Идентификатор - это произвольная последовательность букв, цифр и символа "_", начинающаяся с буквы или "_".

Синтаксис языка программирования - это правила составления предложений языка из отдельных слов. Такими предложениями являются операции, операторы, определения функций и переменных. Особенностью синтаксиса является принцип вложенности (рекурсивность) правил построения предложений. Это значит, что элемент синтаксиса языка в своем определении прямо или косвенно в одной из его частей содержит сам себя. Например, в определении оператора цикла телом цикла является оператор, частным случаем которого является все тот же оператор цикла.

Семантика языка программирования - это смысл, который закладывается в каждую конструкцию языка. Семантический анализ - это проверка смысловой правильности конструкции. Например, если мы в выражении используем переменную, то она должна быть определена ранее по тексту программы, а из этого определения может быть получен ее тип. Исходя из типа переменной, можно говорит о допустимости операции с данной переменной.

Генерация кода - это преобразование элементарных действий, полученных в результате лексического, синтаксического и семантического анализа программы, в некоторое внутреннее представление. Это могут быть коды команд, адреса и содержимое памяти данных, либо текст программы на языке Ассемблера, либо стандартизованный промежуточный код (например, Р-код). В процессе генерации кода производится и его оптимизация.

Препроцессор

В языке Си директивы препроцессора оформлены отдельными строками программы, которые начинаются с символа "#".

```
#define       идентификатор  строка_текста
```

Директива обеспечивает замену встречающегося в тексте программы идентификатора на соответствующую строку текста. Наиболее часто она

применяется для символического обозначения константы, которая встречается многократно в различных частях программы. Например, размерность массива:

```
#define      SIZE      100
int      A[SIZE];
for (i=0; i<SIZE; i++) {...}
```

В данном примере вместо имени SIZE в текст программы будет подставлена строка, содержащая константу 100. Теперь, если нас не устраивает размерность массива, нам достаточно увеличить это значение в директиве define и повторно оттранслировать программу.

Если препроцессор находит в тексте программы указанный идентификатор со списком фактических параметров в скобках, то он подставляет вместо него соответствующую строку из директивы define с заменой в строке формальных параметров на фактические. Основное отличие от функции: если функция реализует подобные действия (подстановка параметров, вызов) во время работы программы, то препроцессор - еще до трансляции. Кроме этого, директива define позволяет оформить в таком виде любую часть программы, независимо от того, законченная это конструкция языка или ее фрагмент. В следующем примере стандартный заголовок цикла for представлен в виде директивы define с параметрами:

```
#define      FOR(i,n)         for(i=0; i<n; i++)
FOR(k,20)  A[k]=0;                  // for(k=0; k<20; k++)
A[k]=0;
FOR(j,m+2) {...}          // for(j=0; j<m+2;  j++) {...}
```

В таком варианте директива define представляет собой макроопределение, а замена в тексте программы идентификатора с параметрами на строку -макроподстановку.

В текст программы вместо указанной директивы включается текст файла, находящегося в системном или, соответственно, в текущем (явно указанном) каталоге. Наиболее часто в программу включаются тексты

заголовочных файлов, содержащие необходимую информацию транслятору о внешних функциях, находящихся в других объектных модулях и библиотеках. Например, include <stdio.h> включает в программу текст заголовочного файла, содержащего объявления внешних функций из библиотеки стандартного ввода-вывода.

Аналогичные средства в других языках программирования носят название макропроцессор, макросредства.

Компоновка

Полученный в результате трансляции объектный модуль включает в себя готовые к выполнению коды команд, адреса и содержимое памяти данных. Но это касается только собственных внутренних объектов программы (функций и переменных). Обращение к внешним функциям и переменным, отсутствующим в данном фрагменте программы, не может быть полностью переведено во внутреннее представление и остается в объектном модуле в исходном (текстовом) виде. Но если эти функции и переменные отсутствуют, значит, они должны быть каким-то образом получены в других объектных модулях. Самый естественный способ - написать их на том же самом Си и оттранслировать. Это и есть принцип модульного программирования -представление текста программы в виде нескольких файлов, каждый из которых транслируется отдельно.

Источником объектного модуля может быть не только Си-программа, но и программа, написанная на любом другом языке программирования, например, на Ассемблере. Но в этом случае необходимы дополнительные соглашения по поводу «стыковки» вызовов функций и обращений к данным в различных языках.

Процесс трансляции

Процесс трансляции не является линейным преобразованием фрагмента программы одного языка на другой, на процесс трансляции одного фрагмента оказывают слияние другие фрагменты программы.

Трансляция представляет собой три последовательных фаз анализа программы, на каждой из которой из текста программы извлекается все более «глубоко» скрытая информация о структуре и свойствах программы, и ее объектах.

Отдельные фазы трансляции могут быть связаны между собой различным образом, через данные в памяти или через файл, что не меняет сущности процесса:

- каждая фаза транслятора получает файл данных от предыдущей фазы, обрабатывает его (линейным или каким-либо другим, например рекурсивным алгоритмом), создает внутренние таблицы данных и по ним формирует выходной файл с данными для следующей фазы;
- фазы трансляции вызывают одна другую в процессе обработки соответствующих языковых единиц. Поскольку синтаксический анализ является обычно центральным в такой структуре, вокруг него и группируются остальные фазы. Из этой схемы выпадает только препроцессор, который обычно представляет собой независимую предварительную фазу трансляции.

Рассмотрим лексический анализ. Механизм распознавания (рисунок 6) базируется на простых системах распознавания, имеющих всего одну единицу памяти (текущее состояние) и табличное описание правил поведения (конечный автомат). Результат классификации лексем (классы лексем) является входной информацией для синтаксического анализатора (класс лексемы называется на входе синтаксического анализатора символом). Значения лексем поступают на вход семантического анализатора в виде первичного заполнения семантических таблиц.

Лексема – последовательность входных символов (литер), допускающая альтернативу (выбор), повторение и взаимное пересечение в процессе распознавания не более чем на заданную глубину. Элементами

лексики являются идентификаторы, константы, строки, комментарии, одно- и многосимвольные знаки операций и т.п.

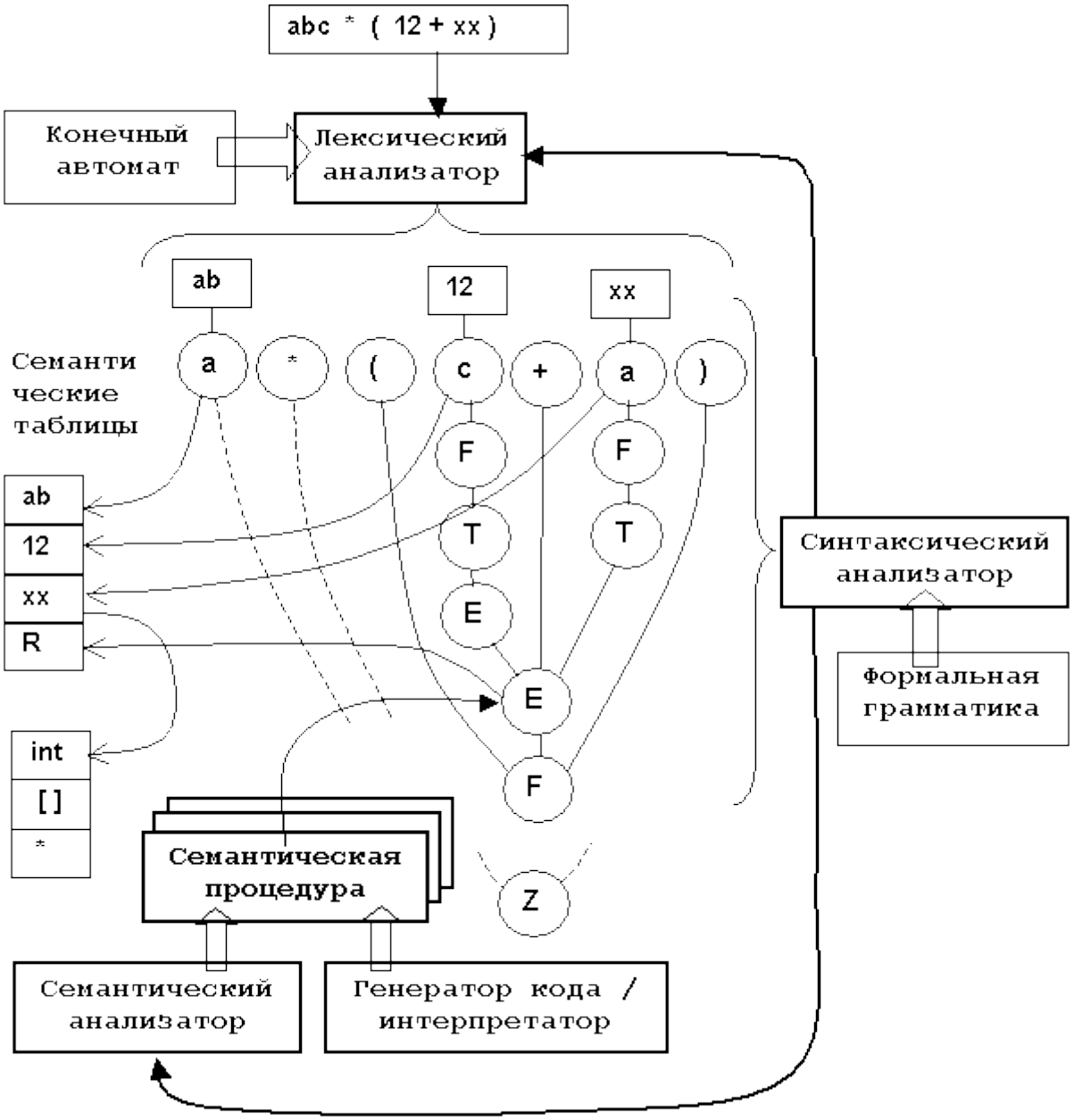

Рисунок 6 – Лексический анализатор

В языке программирования ЛА соответствует фаза трансляции, в которой из последовательности отдельных литер (символов, букв языка) выделяются слова (лексемы или символы - как сони именуются на сходе следующей фазы - синтаксического анализа). Типичными словами в языке программирования являются такие компоненты как комментарии, идентификаторы, константы, служебные слова, знаки операций.

Лексика имеет ряд принципиальных ограничений, которая делает возможной ее распознавание описанными ниже методами:

- лексемы распознаются независимо друг от друга. Единственное, что возможно, это «перекрытие» процесса распознавания двух рядом расположенных лексем на заданное количество литер (которые участвуют в завершении распознавания текущей лексемы и начале распознавания следующей);
- определение лексем включает элементы выбора одного из нескольких вариантов продолжения и повторения;
- вложенность одной лексемы в другую или наличие произвольной вложенности элементов лексемы не допускается.

ЛА является наиболее простой и формализованной фазой трансляции. Любой алгоритм ЛА базируется на последовательном просмотре текста, с возвратом и перечитыванием из входной последовательности не фиксированного числа литер, поэтому программу ЛА иногда называют сканером. Формальной основой описания процесса ЛА являются конечные автоматы.

Формальной основой ЛА являются конечные автоматы (КА). Конечный автомат – алгоритмическая компонента программы «без данных», моделирующая «инстинктивное» поведение, неадаптируемое к последовательности воздействий внешней среды.

Конечный автомат определяется шестью компонентами A = { I,O,S,S0,P,D }:

- множество (алфавит) входных символов I={Ii};
- множество (алфавит) входных символов O={Oj};
- множество состояний автомата S={Sk};
- начальное состояние S0ÍS;
- функция переходов P(S,I) => S , ставящая в соответствие каждой паре «состояние - входной символ» новое состояние;

- функция выходов D(S,I) => O , ставящая в соответствие каждой паре «состояние - входной символ» выходной символ.

Работа КА заключается в преобразовании входной цепочки символов в выходную. Закон этого преобразования определяет поведение КА. Задача проектирования КА состоит в создании КА, обеспечивающем заданные правила преобразования цепочек. Очевидно, что отсутствие у КА внутренней памяти ограничивает его возможности по преобразованию цепочек (общепринятый термин - моделирующая способность). С другой стороны, эти же ограничения позволяют решать в общем случае многие проблемы (термин – алгоритмическая разрешимость, т.е. принципиальная возможность разработать соответствующий алгоритм).

Задачей синтаксического анализа является построение структуры – синтаксического дерева, отражающего взаимное расположение всех синтаксических элементов программы (их порядка следования, повторяемости, вложенности, приоритетов). Средством описания синтаксиса являются формальные грамматики, а их свойства используются для построения распознавателей, базирующихся на табличном описании правил их поведения в сочетании со стековой памятью (магазинные автоматы).

Лексический и синтаксический анализ являются формализованными компонентами языка, как с точки зрения их описания (конечные автоматы и формальные грамматики), так и с точки зрения проектирования распознавателей для них (для чего используются известные методы и алгоритмы).

Заключительная фаза трансляции – семантический анализ, а также фаза синтеза – генерация кода (оптимизация) или интерпретация - привязаны к синтаксису (и, соответственно, к синтаксическому анализатору), поскольку интерпретируют «смысловое» содержание и правила преобразования (или исполнения) элементарных синтаксических единиц, выделенных синтаксическим анализатором.

Особенностью семантического анализа является то, что он менее привязан к структуре программы: семантика одного и того же объекта программы может определяться синтаксически не связанными элементами программы.

Семантический анализ не формализуем, поскольку семантика программы представляется в процессе трансляции уникальной структурой данных, содержащей описания множеств объектов языка, определенных в программе, их свойств и взаимосвязей (семантические таблицы). Работа с такими таблицами производится через и семантические процедуры, соответствующие элементам синтаксиса (правила грамматики), которые также разрабатываются содержательным образом, не имея под собой формальной основы.

Практическая работа №1. Однооконные приложения

1. Реализовать форму, обеспечивающую выбор цвета фона полей ввода. (Например, поле с названием цвета и кнопка для реализации события)

2. Реализовать форму для ввода матрицы произвольного размера (по выбору пользователя) и расчета её ранга.

3. Реализовать простейший калькулятор с помощью приложения windows-form.

4. Реализовать приложение, строящее график функции с учетом коэффициентов.

Вариант:	Функция
1	$A \cdot \cos(B \cdot x)$
2	$A \cdot \sin(x+B)$
3	$A+\cos(B+x)$
4	$A \cdot \sin(x \cdot B)$
5	$A \cdot \sin(x)+B$

5. Реализовать приложение, которое позволяет вводить данные в табличном виде, сохранять их в файл и рассчитывать статистические (среднее, максимум и минимум) числовых данных, и строить график-гистограмму по числовым данным

Вариант:	Функция
1	оценки обучающихся
2	расходы по месяцам
3	стоимость медицинских препаратов
4	количество заболевших по странам
5	курс валют

Практическая работа №2. Многооконные приложения

1. Для заданий 1 практической работы изменить пользовательский интерфейс так, чтобы в приложении была одна главная форма, содержащая меню, вызывающее решение других работ. При этом для каждой практической должны быть отдельные формы ввода и вывода результатов. Кроме того, приложение должно содержать формы «Справочной информации» и возможность оставить отзыв и сохранить его в файл.

2. Реализовать пользовательский интерфейс позволяющий из одной главной формы открывать форму с вводом данных, форму с выводом данных (результатов расчетов) и форму с графиками по данным. При этом пользователь должен иметь возможность выбрать вид графика, дополнительные параметры его построения, цветовые характеристики элементов графика, параметры решения из отдельной формы с настройками. Так же на форме с графиком должны присутствовать элементы меню (radion button или checkbox) с добавлением вычисленных характеристик на форму. На форме с вводом данных должно присутствовать: выбор файла с данными (например, с показателями температур за период), количество точек (заполняться автоматически из данных файла по количеству строк), возможность выбора первой и последней точки для построения графика, масштаб. Форма с выводом данных должна содержать таблицу со значениями, полученными из файла, с учетом обрезки по первой и последней точке, среднее значение и отклонение каждой точки от среднего значения. На графике должен отображаться сам график и, опционально (в зависимости от выбора пользователя в меню на форме графика), среднее значение или график отклонений показателей от среднего значения.

3. Реализовать программу решающую уравнение с помощью метода половинного деления. Реализовать пользовательский интерфейс позволяющий из одной главной формы открывать форму с вводом данных, форму с выводом данных (результатов расчетов) и форму с графиками по данным. При этом пользователь должен иметь возможность выбрать вид графика, дополнительные параметры его построения, цветовые характеристики элементов графика, параметры решения из отдельной формы с настройками. Так же на форме с графиком должны присутствовать элементы меню (radion button или checkbox) с добавлением вычисленных характеристик на форму. На форме ввода данных должна выбираться функций (вводиться или собираться из слагаемых с коэффициентами), количество шагов или погрешность вычисления. Форма вывода должна содержать таблицу с пошаговыми результатами на количество шагов, выбранных пользователем (или пока не достигнет заданной погрешности), и

значение отклонение каждого шага от предыдущего. На графике должен отображаться один из графиков по выбору пользователя: график итераций (значений на каждом шаге), график абсолютных отклонений, график относительных отклонений)

Практическая работа №3. Эргономика пользовательского интерфейса

1. Реализовать программное приложение, интерфейс которого будет иметь не менее 3 различных индикаторов выполнения (визуально различающихся). Приложение должно осуществлять расчет нескольких показателей, сохранение данных в файл, загрузка данных из файла.

2. Реализовать программное приложение, интерфейс которого будет иметь кнопку с бесконечными размерами.

3. Реализовать программное приложение, интерфейс которого будет иметь кнопку или меню с минимальным расстоянием до кнопки.

4. Реализовать программное приложение, интерфейс которого предполагает форму ввода данных. Для формы ввода предусмотреть средства уменьшения вероятности опечаток и скорости ввода. Должны быть использованы механизмы быстрого ввода (ввод по умолчанию, сокращения и т.д.). Кроме того, введенные или отредактированные поля должны выделяться, обязательные и необязательные поля так же должны быть выделены. Форма ввода должна иметь не менее 8 блоков для ввода, в том числе ввод табличных данных.

5. Реализовать программное приложение, интерфейс которого предполагает поиск на форме и группировку данных. Форма приложения должна иметь скрываемые области. Области могут становиться видимыми (разворачиваться) при введении каких-либо связанных данных или при нажатии на область формы.

6. Реализовать программное приложение, замеряющее скорость заполнения формы ввода. При длительном отсутствии ответа пользователя приложение должно выдавать сообщение.

Практическая работа №4. Пользовательский интерфейс и справочная информация

1. Реализовать вызов следующего вида справок в приложении: процедурная справка, контекстная справка, справка состояния. Вызов справки должен соответствовать её типу.

2. Реализовать в приложении информационные сообщения, сообщения-предупреждения и сообщения о критической ситуации (об ошибке). Информационные сообщения должны предоставлять информацию о результатах выполнения выбранной команды. Предупреждения предлагают пользователю выбор и сообщают о возможных вариантах последующих за действием пользователя. Предупреждения должны вызываться в потенциально опасных ситуациях – контроль ввода, сохранения и т.д. Критические ситуации предупреждают о проблемах, требующих вмешательство или внесения изменений, например, корректировки неверно введенных данных. Реализованные сообщения должны визуально отличаться.

Практическая работа №5. Лексический анализ

1. Реализовать программное приложение (модель КА), определяющее строки комментариев в коде, содержащемся в файле.

2. Реализовать программное приложение (модель КА), определяющее переменны, комментарии в коде, содержащемся в файле.

3. Реализовать программное приложение (модель КА), определяющее все лексемы в коде, содержащемся в файле.

4. Разработать собственный набор лексем и синтаксиса языка программирования. Реализовать лексический анализатор.

Практическая работа №6. Синтаксический анализ

1. Построить формальную грамматику, обладающую нетерминалами альтернативы, повторения и приоритета.

2. Построить формальную грамматику, используя множество First

3. Построить формальную грамматику, используя множество Last

4. Реализовать нисходящий разбор с возвратами для любой формальной грамматики

5. Реализовать метод свертки-переноса для любой формальной грамматики

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

Основная учебная литература

1. Трофимов, В.В. Алгоритмизация и программирование : учебник / В.В. Трофимов, Т.А. Павловская. – Москва : Издательство Юрайт, 2019. – 137 с. – ISBN 978-5-534-07834-3. – URL: https://www.biblio-online.ru/viewer/algoritmizaciya-i-programmirovanie-423824. - (дата обращения: 22.03.2020). – Текст : электронный.

Дополнительная литература

1. Крупский, В. Н. Теория алгоритмов. Введение в сложность вычислений : учебное пособие для бакалавриата и магистратуры / В. Н. Крупский. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 117 с. — (Авторский учебник). — ISBN 978-5-534-04817-9. — URL: https://biblio-online.ru/bcode/444131 (дата обращения: 26.11.2019). — Текст : электронный.

2. Судоплатов, С. В. Математическая логика и теория алгоритмов : учебник и практикум для академического бакалавриата / С. В. Судоплатов, Е. В. Овчинникова. — 5-е изд., стер. — Москва : Издательство Юрайт, 2019. — 255 с. — (Высшее образование). — ISBN 978-5-534-00767-1. — URL: https://biblio-online.ru/bcode/432018 (дата обращения: 26.11.2019). — Текст : электронный.

3. Язык программирования Си++. Курс лекций : учебное пособие / Фридман А.Л. – Москва : ИНТУИТ.РУ «Интернет-университет Информационных Технологий», 2004. – 262 с. – ISBN 5-9556-0017-5. - URL: http://biblioclub.ru/index.php?page=book_view_red&book_id=233058. - (дата обращения: 22.03.2020). – Текст : электронный.

4. Сузи, Р.А. Язык программирования Python: учебное пособие / Р.А. Сузи. – 2-е изд., испр. – Москва: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007. – 327 с. - ISBN 978-5-94774-711-9. – URL:

http://biblioclub.ru/index.php?page=book_view_red&book_id=233288. - (дата обращения: 22.03.2020). – Текст : электронный.

5. Мирошниченко, И.И. Языки и методы программирования: учебное пособие / И.И. Мирошниченко, Е.Г. Веретенникова, Н.Г. Савельева. – Ростов н/Д: Издательско-полиграфический комплекс Рост. гос. экон. ун-та (РИНХ), 2019. – 188 с. - ISBN 978-5-7972-2604-8. – URL: http://biblioclub.ru/index.php?page=book_view_red&book_id=567706. - (дата обращения: 22.03.2020). – Текст : электронный.

СОВРЕМЕННЫЕ ПРОФЕССИОНАЛЬНЫЕ БАЗЫ ДАННЫХ И СПРАВОЧНЫЕ СИСТЕМЫ

CITForum.ru : on-line библиотека свободно доступных материалов по информационным технологиям на русском языке : сайт. – 2001 – URL: http://citforum.ru (дата обращения: 22.03.2020). – Текст: электронный.

eLIBRARY.RU : научная электронная библиотека : сайт. – Москва, 2000 - . – URL: http://www.elibrary.ru (дата обращения: 22.03.2020). – Режим доступа: для зарегистрир. пользлвателей. – Текст: электронный.

Единое окно доступа к образовательным ресурсам : сайт. – Москва, 2005 - . – URL: http://window.edu.ru/ (дата обращения: 22.03.2020). –Текст: электронный.