КИПР

ТУСУР

Кафедра конструирования
и производства радиоаппаратуры

Линейные программы с графическим интерфейсом в среде Lazarus

Кобрин Юрий Павлович
Линейные программы с графическим интерфейсом в среде Lazarus. Методические указания к лабораторной работе и по организации самостоятельной работы по дисциплинам «Информатика» и «Информационные технологии» для студентов очного и заочного обучения специальностей 211000.62 (бакалавриат) и 162107.65 «Информатика и информационные технологии» (специалитет). - Томск: Томский государственный университет систем управления и радиоэлектроники (ТУСУР), кафедра КИПР, 2017. – 30 с.

Lazarus — открытая бесплатная кроссплатформенная среда визуальной разработки программного обеспечения для компилятора *Free Pascal*, максимально приближённая к *Delphi*. Методические указания посвящены освоению основ объектно-ориентированного программирования (ООП) в интегрированной среде *Lazarus*, а также с особенностями технологии визуального программирования простейших линейных программ с графическим интерфейсом. Для создания графического интерфейса приложения Lazarus используют готовые компоненты, значки которых находятся на панели компонентов. После того как пользователь помещает компонент на форме, программный код для него генерируется автоматически. Вручную остаётся запрограммировать только те действия, которые будет выполнять это приложение.

Методические указания предназначены для помощи в подготовке бакалавров и магистрантов в Информатике, выполнения курсовых и дипломных проектов.

Оглавление

1 Цели и задачи работы

Цель работы: освоение основных элементов интегрированной среды визуального программирования *Lazarus* и разработка с её помощью простейших приложений.

В ходе выполнения работы следует усвоить:

- структуру программ на *Lazarus*;
- назначение базовых объектов на *Lazarus*.

Научиться:

- работать в интегрированной среде Lazarus;
- усвоить правила использования основных типов данных (констант, переменных) и операторов в *Lazarus*;
- освоить приёмы программирования и отладки простейших линейных алгоритмов.

2 Порядок выполнения работы

Перед выполнением этой работы следует:

1) Ознакомиться с теоретической частью (раздел 5). В качестве дополнительной литературы использовать [1,2,3,4,5,6,7,8].
2) Выполнить индивидуальное задание по своему варианту.
3) Войти в свой личный каталог, загрузить и настроить систему программирования ***Lazarus***.
4) Войти в режим редактирования и набрать текст программы.
5) Запустить программу на трансляцию и выполнение.
6) Исправить ошибки с помощью отладки, чтобы программа давала правильные результаты.
7) Ответить на контрольные вопросы.
8) Оформить отчёт и защитить его у преподавателя.

3 Отчётность

Отчёт должен быть выполнен в соответствии с [9] и состоять из следующих разделов:

1) Тема и цель работы.
2) Индивидуальное задание.
3) Блок-схема алгоритма.
4) Откомпилированный текст программы (в электронном виде).
5) Ответы на контрольные вопросы.
6) Результаты выполнения программы.
7) Выводы.

При защите отчёта по работе для получения зачёта студент должен:

- уметь отвечать на контрольные вопросы;
- обосновать структуру выбранного алгоритма и доказать, что разработанное приложение работает правильно;
- уметь пояснить назначение элементов разработанного приложения;
- продемонстрировать навыки работы в интегрированной среде ***Lazarus***.

4 Контрольные вопросы

1) Дайте определение объектно-ориентированному программированию. Какими достоинствами и недостатками обладает данная технология?
2) Что собой представляют классы, объекты, интерфейсы?
3) В чем заключается сущность принципов инкапсуляции, полиморфизма, наследования?
4) Какие принципы положены в основу технологии ООП?
5) Что собой представляет визуальное проектирование интерфейса?
6) Перечислите основные элементы пользовательского интерфейса Lazarus.
7) Какие функции выполняет палитра компонентов в Lazarus?
8) Для чего используется Инспектор объектов?
9) Разъясните назначение основных файлов, входящих в проект Lazarus.
10) Какой алгоритм называется линейным? Из каких операторов он состоит?

5 Методика разработки линейных программ

5.1 Основные понятия объектно-ориентированного программирования

Lazarus — открытая среда разработки программного обеспечения на *языке Object Pascal* для компилятора *Free Pascal* базируется на наиболее распространённой в настоящее время технологии **объектно-ориентированного программирования (ООП).** При объектно-ориентированном подходе к программированию главными элементами программ являются совокупности взаимодействующих между собой данных – *объектов*.

Классы задают функциональные возможности и структуру объектов, состоящих из полей, методов и свойств. **Объекты** – это экземпляры классов. Класс - это шаблон, описывающий свойства и методы, которые будут доступны у ещё не созданного объекта, построенного по описанию класса.

Компоненты класса:

- **Поле** - переменная любого типа (в том числе и класса), представляющая элементы данных, присутствующие в каждом экземпляре класса. Для скрытия от внешнего мира структуры класса поля обычно размещаются в группе частных описаний private. Доступ к ним осуществляется через свойства.
- **Метод** – это процедура или функция, связанная с классом. Большая часть методов оперирует объектами (т.е. экземплярами классов). Некоторые методы (называемые методами класса) работают с классами как с типами.
- **Свойства** дают доступ к полям, напрямую недоступным. Свойства имеют спецификаторы доступа, которые определяют, каким образом данные могут быть прочитаны или изменены. Из других частей программы, то есть снаружи объекта, свойство в большинстве случаев выглядит как поле.

5.1.1 Наследование, инкапсуляция и полиморфизм

Объекты обладают свойствами *наследования, инкапсуляции и полиморфизма*.

Наследование (от англ. *Inheritance*) - создание новых классов-потомков, которые обладают всеми свойствами классов-предков. Целесообразно при создании нового класса использовать свойства имеющегося класса. Исходный класс называется *родительским* (предком), производный – *потомком* (наследником, дочерним классом). Класс-наследник наследует все описания и методы класса-предка. Для него нужно описать только дополнительные свойства.

Классы образуют *иерархическое дерево* (Рис. 5.1). Общим предком всех классов в Lazarus является абстрактный класс $TObject$, обладающий самыми общими методами, присущими любому объекту. Этот класс описывает основные принципы поведения объектов во время работы программы (создание, уничтожение, обработка событий и т.п.).

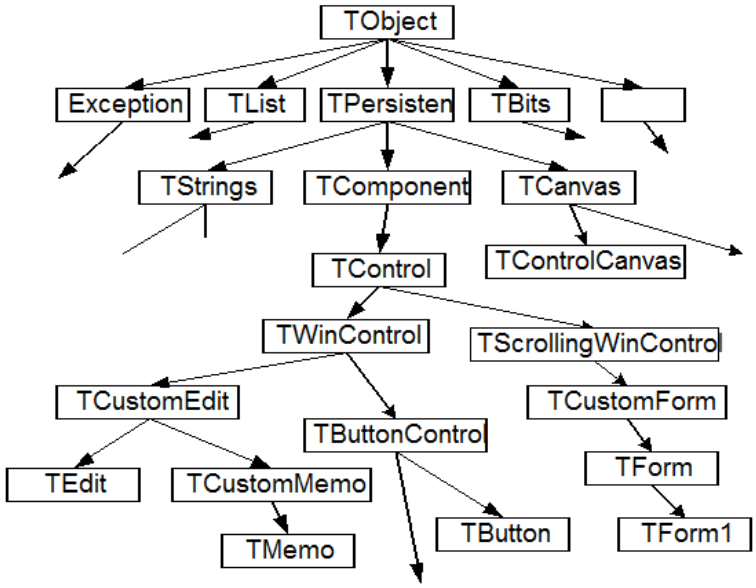
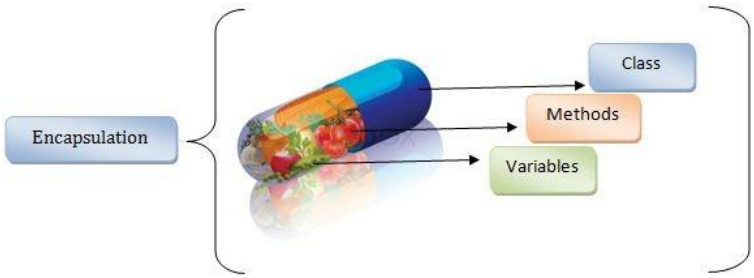


Рис. 5.1 - Фрагмент структуры классов Lazarus

Принцип наследования позволяет поэтапно создавать новые сложные классы. Класс-потомок получает все поля своих предков, но не может их удалять или переопределять. К классу-потомку обычно *добавляют новые поля и методы*. Чем ниже находится класс в дереве иерархии, тем большим набором данных и методов обладают объекты этого класса.

Инкапсуляция (англ. *encapsulation*) – упаковка данных и методов (подпрограмм) работы с ними и в единый компонент, позволяющая скрыть от внешнего мира внутреннюю структуру объекта, что обеспечивает максимально возможную независимость объектов.



Она предоставляет разработчику определённый набор свойств и методов для управления поведением и свойствами объекта, определяемыми внутри класса.

Доступ к данным возможен только с использованием методов работы с ними внутри класса, что исключает возможность неправильных действий над данными. Кроме того, инкапсуляция даёт возможность изменять в процессе разработки каждый класс без влияния на остальную часть программы.

Полиморфизм – использование для разных объектов неодинаковых методов под одинаковыми именами, позволяющее различным объектам реагировать по-разному на одни и те же события. Функционально одинаковые операции с объектами предками и потомками лучше именовать одинаково. При этом можно переопределять (перекрывать) процедуры объекта предка. Одноименные методы разных объектов делаются уникальными введением слова *virtual* (виртуальный)[1] после заголовка метода.

Перед работой с виртуальными методами используется специальный метод - **конструктор** (от англ. *constructor*), который *создаёт и инициализирует*[2] *объект, выделяет под него память*. При компиляции конструктор составляет таблицу с адресами виртуальных методов, по которой на стадии обращения к нему выбирается адрес нужного метода (так называемое *позднее связывание*). Объявление конструктора похоже на объявление процедуры, но в её заголовке вместо слова *procedure* употребляется слово *constructor*.

Метод рекомендуется делать виртуальным, когда предполагается его дальнейшее изменение. Для невиртуальных методов используется *раннее связывание*, когда адрес метода определяется на этапе компиляции. Если в классе-предке есть виртуальный метод, то все методы с аналогичным именем у классов-потомков должны быть виртуальными.

Деструкторы (от англ. *Destructors*) - это специальные методы, которые *разрушают объекты и освобождают занимаемую ими память*. Объявление деструктора сходно с объявлением процедуры, только вместо слова процедура указывается ***destructor.***

5.1.2 Создание класса

Новые классы объявляются в разделе типов *type*.

```
type
  <Имя класса> = class(<Имя предка>)
   описание новых данных;
   заголовки новых методов;
end;
```

---

[1] **Виртуальный метод** - метод класса, который может быть переопределён в классах-наследниках так, что конкретная реализация метода для вызова будет определяться во время исполнения. Необязательно знать точный тип объекта для работы с ним через виртуальные методы - достаточно лишь знать, что объект принадлежит классу или наследнику класса, в котором метод объявлен.

[2] **Инициализация** (англ. *initialization*) - присвоение начальных значений полям объекта.

Класс должен быть объявлен до создания объектов класса. Каждый объект имеет уникальную копию полей класса, но все объекты класса используют одни и те же методы. Объекты создаются при исполнении программы в динамической памяти. Для создания объекта используется метод constructor, а для уничтожения – *destructor*.

Lazarus-приложение выполняется в среде Windows, и как любое Windows- приложение, получает *сообщения* о возникающих для него событиях. **Событие** (англ. *Event*) в ООП - это *сообщение*, которое возникает в результате действий пользователя, аппаратуры компьютера или других программ во время работы приложения. Например, о начале или о завершении некоторой процедуры, нажатии клавиши мыши или клавиатуры, изменении текущих координат мыши и т.п. При возникновении события (у каждого события есть имя) выполняется метод *обработчик события* - процедура, реализующая действия, соответствующие этому событию (Рис. 5.2).

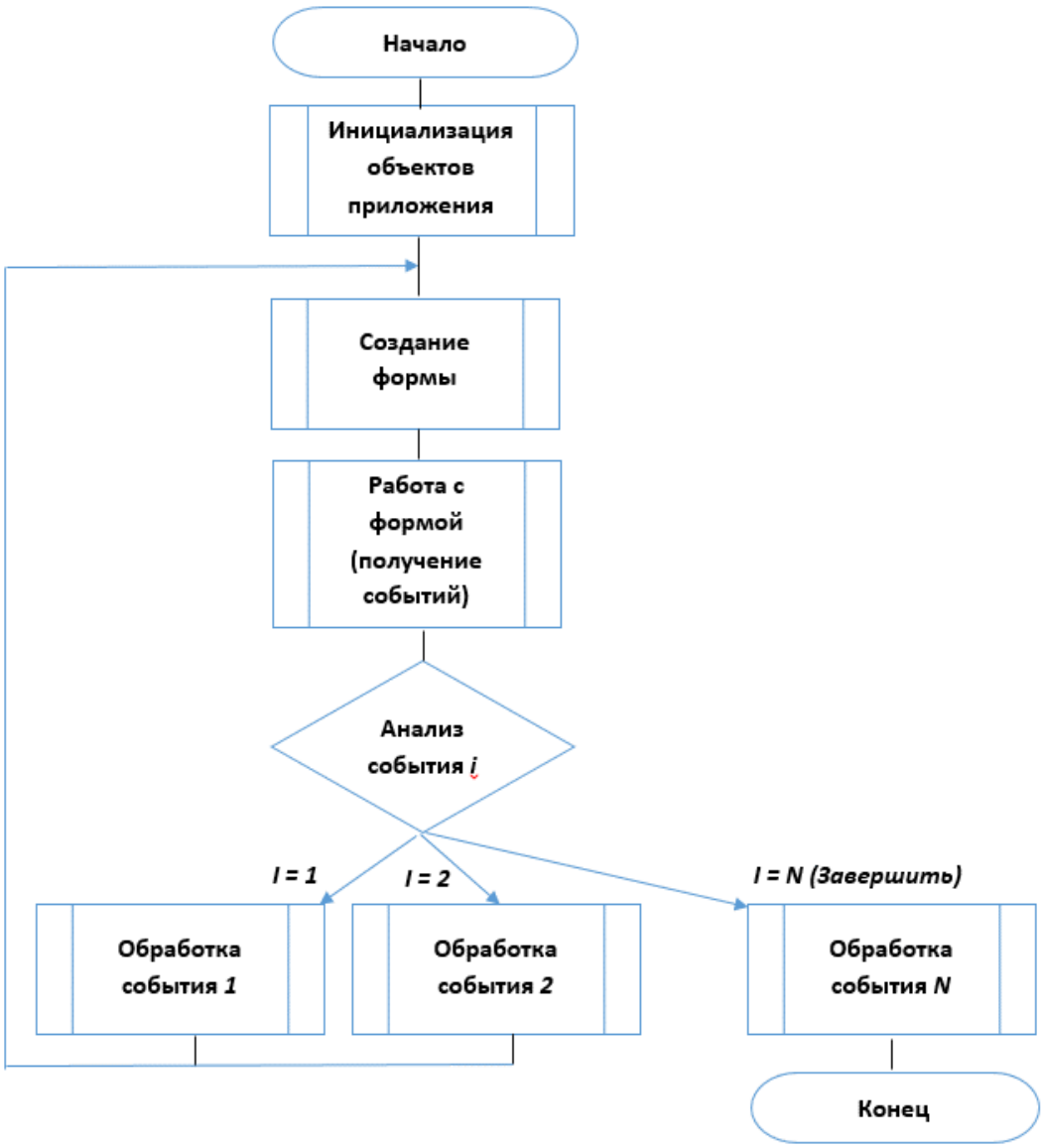


Рис. 5.2 - Организация обработки событий в ООП

Следовательно, управление работой приложения сводится к обработке получаемых сообщений.

Lazarus автоматически генерирует заготовку процедуры обработки событий - обработчики событий для каждого компонента. Имя обработчика событий при этом формируется из имени компонента и названия события (например, ***EditlClick***). Имя обработчика события автоматически квалифицируется именем класса формы. Например: ***TForml.ButtonlClick(Sender: TObject***).

Для каждого компонента предусмотрено одно стандартное событие. Например, для командной кнопки, флажка, списка, поля ввода - это событие ***Click***, а для формы - событие ***FormCreate***. Для того чтобы автоматически добавить в модуль объявление и описание разработчика стандартного события, достаточно выполнить на компоненте формы или самой форме двойной щелчок мышью.

Таким образом, класс — это некий шаблон, описание не только самой структуры данных (чего там может или должно быть), но и поведения, функциональных возможностей объектов этого класса. Объект - это совокупность свойств и методов, а также событий, на которые он может реагировать (Рис. 5.3). Внешнее управление объектом осуществляется через обработчики событий. Эти обработчики обращаются к методам и свойствам объекта. Начальные значения данных объекта могут задаваться также в процессе проектирования установкой различных свойств. В результате выполнения методов объекта могут генерироваться новые события, воспринимаемые другими объектами программы или пользователем.

Рис. 5.3 - Организация объекта

5.2 Структура проекта в Lazarus

Любой проект в Lazarus – это совокупность файлов, из которых создаётся единый выполняемый файл. Список файлов проекта имеет следующий вид (Таблица 5.1).

Таблица 5.1 - Список файлов проекта Lazarus

Название файла	Расширение	Расшифровка расширения	Примечание
Файл проекта	$.lpr$	$lpr = Lazarus\ Project$ на языке FPC	Создаётся автоматически.
Файл описания (конфигурации) проекта	$.lpi$	$Lpi = Lazarus\ Project\ Information$	Создаётся автоматически.
Описание формы	$.lfm$	$lfm = Lazarus\ form$	Создаётся автоматически

Название файла	Расширение	Расшифровка расширения	Примечание
Программный модуль	$.pas$		Код модуля на языке FPC.
Файл ресурсов	$.lrs$	$lrs = Lazarus\ Resource$	Курсоры, иконки и др. Создаётся автоматически
Компилированный	$.compiled$		Содержит конфигурации проекта, нужные для формирования объектного файла проекта. Создаётся компилятором автоматически.
Ассемблерный	$.ppu$	$ppu = p\ p\ unit$	Создаётся компилятором автоматически.
Объектный код модуля	.o	$o = Object$	Код нужен для сборки проекта. Создаётся автоматически.
Исполняемый файл	$.exe$	$exe = Execution$	Создаётся компилятором автоматически.

Во вложенной папке проекта \backup автоматически создаются резервные копии файлов. Для приложения автоматически создаётся файл проекта (.lpr) и файл шаблона кода модуля формы. Они размещаются на разных страницах. Код проекта по умолчанию не отображается, так как его редактировать не рекомендуется.

После компиляции программы из всех файлов проекта создаётся единый выполняемый файл, имя которого совпадает с именем проекта.

5.3 Использование подпрограмм

Подпрограмма - это некоторая последовательность операторов, к которой можно обратиться по имени. Всякий раз, когда мы называем имя подпрограммы, инициируется последовательность запрограммированных в ней действий.

Когда следует использовать подпрограммы?

Логично применить подпрограммы (процедуры или функции) как средство экономии памяти. Если Вы обнаружите, что в Вашей программе одни и те же действия выполняются *многократно, но с разными исходными данными*, то целесообразно один раз записать эту последовательность действий в процедуре или функции, а затем многократно вызывать ее из разных точек программы.

При создании программы для решения сложной задачи возникают те же проблемы, что и при разработке больших и сложных систем. Пути решения этих проблем для систем хорошо известны. Это, прежде всего, *декомпозиция* (разделение) сложной задачи на отдельные подзадачи, подзадач – на ещё более мелкие подзадачи и так далее, пока подзадача не станет понятной и легко доступной для реализации небольшим числом простейших операторов. Каждая функционально законченная подзадача может и должна быть реализована подпрограммой.

Подпрограммы, используемые в языке ***Lazarus***, как раз и служат для разделения логики большой программы на обозримые и управляемые фрагменты. Любая подпрограмма может в свою очередь обращаться как к другим подпрограммам, так и к самой себе (рекурсия).

Программы, состоящие из процедур и функций, называют *модульными*. Они имеют отчётливую *иерархическую структуру*. Опытные программисты обоснованно считают, что модульные программы гораздо легче для разработки и проще для понимания и тестирования, чем *монолитные*.

Как обращаться к подпрограмме-функции в выражениях Вы уже знаете. Подпрограмма-процедура активизируется с помощью оператора процедуры, в котором содержатся имя процедуры, после которого в круглых скобках приводят список необходимых (фактических) параметров. Некоторые процедуры списка параметров могут и не иметь.

В языке ***Lazarus*** существует возможность хранить хорошо зарекомендовавшие себя процедуры и функции в модулях-библиотеках (***unit***). При необходимости Вы можете подключать к разрабатываемой программе на ***Lazarus*** описания данных и подпрограммы как из своих собственных *библиотек-модулей*, так и из *библиотек-модулей*, разработанных другими программистами.

Для подключения одного или более модулей с нужными подпрограммами необходимо в раздел описаний любой программы на ***Lazarus*** ввести предложение

```
Uses
  <список модулей-библиотек>;
```

6 Линейные программы

6.1 Решение задач на компьютере

Решение любой задачи достигается обработкой информации или данных. Поэтому, как программисту, Вам необходимо знать, *как*:

- *осуществить ввод данных* - т.е. ввести информацию (данные) в программу с клавиатуры, диска или из порта ввода/вывода;
- *сохранить данные* (*информацию*) во внутренней (оперативной) или внешней (экран, магнитные и оптические диски и другие устройства ввода-вывода) памяти. Это достигается использованием констант, переменных и структур, содержащих числа (целые и вещественные), текст (символы и строки) или указатели - адреса переменных и структур;

– *правильно задать команды обработки данных* (*операторы*, *инструкции*). С их помощью осуществляют присваивание значений, вычисление выражений, сравнение значений (равно, не равно, больше и т.д.);
– *получить данные из программы* (*вывод данных*) на экран, на диск или в порт

Вы можете написать и упорядочить операторы так, чтобы:

• некоторые из них выполнялись при выполнении определённогоо условия или ряда условий (условное выполнение);

• другие выполнялись определённое число раз (циклы), пока истинно некоторое условие, или пока условие не станет истинным;

• другие собирались в отдельные части, объединённые именем (подпрограммы), которые могут быть выполнены в нескольких местах программы, где есть вызов их по имени.

6.2 Линейные программы

При любых значениях исходных данных в линейном алгоритме все действия ***A1, A2, …. AN*** выполняются однократно, строго последовательно, одно за другим (Рис. 6.1). Такой порядок выполнения действий называется *естественным*.

Программы с линейной структурой являются простейшими и используются **в чистом виде** на практике достаточно редко: при расчёте обычных арифметических и алгебраических выражений и решении ряда простейших задач.

Алгоритм линейной структуры представляет собой последовательность действий и не содержит каких-либо **условий.**

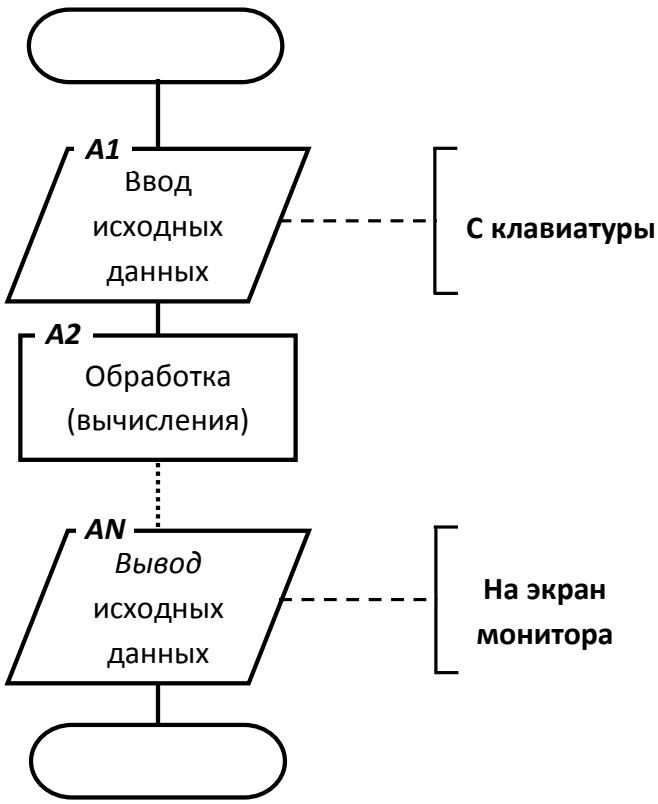


Рис. 6.1 – Линейный алгоритм

В линейных программах могут применяться только:

– операторы (процедуры) ввода,
– операторы (процедуры) вывода,
– операторы присваивания (изменения значения переменных),
– операторы обращения к подпрограммам.

Рассмотрим важнейшие элементы программы на языке ***Lazarus***, используемые при создании линейных программ.

6.3 Оператор присваивания

Оператор присваивания является основным вычислительным оператором. Если в программе надо выполнить вычисление, то нужно использовать оператор присваивания.

В результате выполнения оператора присваивания значение переменной меняется, ей присваивается новое значение.

В общем виде инструкция присваивания выглядит так:

<***Имя*** > ***:=*** <***выражение***>

где:

Имя – имя переменной, значение которой изменяется в результате выполнения оператора присваивания;

: = - символ присваивания.

Выражение - выражение, значение которого присваивается переменной, имя которой указано слева от символа оператора присваивания.

1) Оператор присваивания выполняется следующим образом:

2) Сначала вычисляется значение выражения, которое находится справа от символа присваивания (***: =***).

3) Затем вычисленное значение записывается в переменную, имя которой стоит слева от символа присваивания[3].

Например, в результате выполнения операторов:

```
I := 0;          {значение переменной i становится равным нулю}
a := b + c;      {значением переменной a будет суммf значений переменных b и c}
j := j + 1;      {значение переменной j увеличивается на единицу}
```

Выражение должно быть *совместимо по присваиванию с типом переменной*. Оператор присваивания считается верным, если тип выражения соответствует или может быть приведён к типу переменной, получающей значение. Например, переменной типа ***real*** можно присвоить значение выражения, тип которого ***real*** или ***integer***, а переменной типа ***integer*** можно присвоить значение выражения только типа ***integer***.

Так, например, если переменные ***i*** и ***n*** имеют тип ***integer***, а переменная ***d*** - тип ***real***, то

```
I := n / 10;  i := 1.0;     {операторы неправильные}
d := i +1;                  {операторы записаны правильно}
```

---

[3] Старое значение переменной слева от символа присваивания будет при этом не сохранится.

6.4 Организация ввода и вывода данных

Любая программа в своей работе использует какие-то исходные данные.

Для организации ввода можно использовать компонент TEdit (Поле ввода), для вывода результатов – компонент TLabel (Поле вывода).

6.4.1 Использование компонент TEdit и Tlabel

Компонент (***TEdit***) представляет из себя поле ввода (редактирования) *строки символов*.

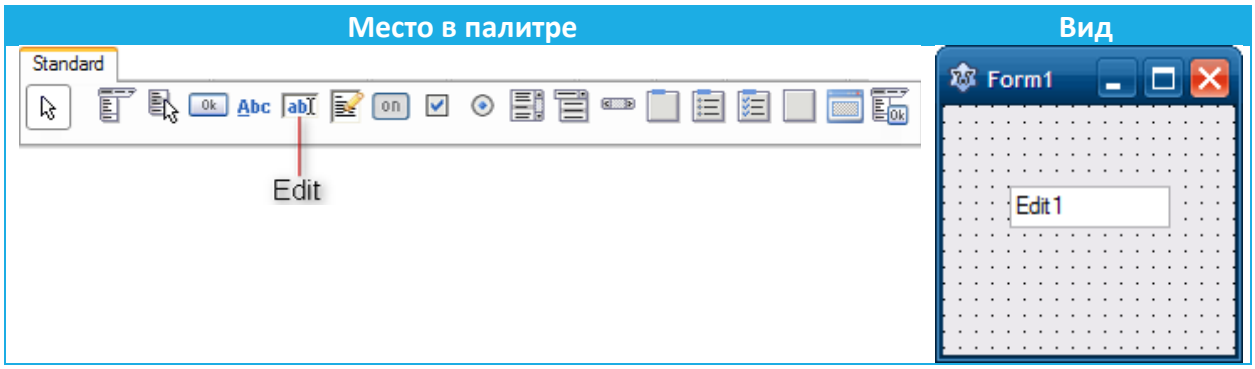


Таблица 6.1 представляет основные свойства компонента ***TEdit.***

Таблица 6.1 - Основные свойства компонента ***TEdit*** (поле ввода строки символов.)

Свойство	Описание
Name	**Имя компонента.** Используется в программе для доступа к компоненту и его свойствам, в частности для доступа к тексту, введённому в поле редактирования.
Text	**Текст**, находящийся в поле ввода и редактирования.
Left	Расстояние от левой границы компонента до левой границы формы.
Top	Расстояние от верхней границы компонента до верхней границы формы.
Width, Height	**Ширина, высота поля.**
Font	**Шрифт**, используемый для отображения вводимого текста.
ParentFont	Признак наследования компонентом характеристик шрифта формы, на которой находиться компонент. Если значение свойства равно *True*, то при изменении свойства *Font* формы автоматически меняется значение свойства *Font* компонента.
Enabled	Используется для ограничения возможности изменить текст в поле редактирования. Если значение свойства равно *False*, то текст в поле редактирования изменить нельзя.
Visible	Позволяет скрыть текст (*False*) или сделать его видимым (*True*).

Чтобы ввести численные данные с помощью компонент ***TEdit*** (Поле ввода), а для вывода численных результатов – компонент ***TLabel*** (Поле вывода) следует подключить специальные функции перевода из символьного представления в численное (Таблица 6.2) и наоборот (Таблица 6.3).

Таблица 6.2 - Преобразование строк в другие типы

Обозначение	Тип аргументов	Тип результата	Действие
StrToDateTame(S)	Строка	Дата и время	Преобразует символы из строки ***S*** в дату и время
StrToFloat(S)	Строка	Вещественное число	Преобразует символы из строки ***S*** в вещественное число
StrToInt(S)	Строка	Целое число	Преобразует символы из строки ***S*** в вещественное число
Val(S, X, Kod)	Строка	Число	Преобразует троку символов ***S*** во внутреннее представление числовой переменной X. Если преобразование прошло успешно, ***Kod*** = 0.

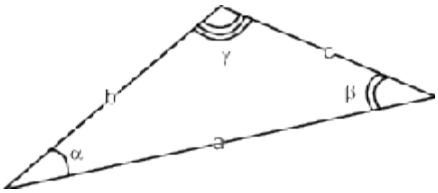
Таблица 6.3 - Обратное преобразование

Обозначение	Тип аргументов	Тип результата	Действие
DataTimeToStr(V)	Дата и время	Строка	Преобразует дату и время в строку ***S***
FloatToStr(V)	Вещественное число	Строка	Преобразует вещественное число в строку ***S***
IntToStr(V)	Целое число	Строка	Преобразует целое число в строку ***S***
FloatToStrF(V, F, P, D)	Вещественное число	Строка	Преобразует вещественное число ***V*** в строку символов с учётом формата ***F*** и параметров ***P, D***.

6.5 Пример разработки программы

Последовательность стандартных шагов: *ввести, сохранить, отредактировать и выполнить* - определяет общий сценарий разработки практически любой программы. Рассмотрим, как можно выполнить любой из этих шагов.

ЗАДАЧА. Известны длины сторон треугольника a, b и c. Вычислить площадь S, периметр P и величины углов α, β и γ треугольника.



Прежде чем приступить к написанию программы, вспомним математические формулы, необходимые для решения задачи.

Для вычисления площади треугольника применим теорему Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

где полупериметр:

$$p = \frac{a = b = c}{2}$$

один из углов найдём по теореме косинусов:

$$\cos(\alpha) = \frac{b^2 + c^2 - a^2}{2bc};$$

второй — по теореме синусов:

$$\sin(\beta) = \frac{b}{a}\sin(\alpha)$$

третий — по формуле:

$$\gamma = \pi - (\alpha + \beta)$$

Решение задачи можно разбить на следующие этапы:

1. Определение значений a, b и c (ввод величин *a, b* и *c* в память компьютера).
2. Расчёт значений S, P, α, β и γ по приведённым формулам.
3. Вывод значений S, P, α, β и γ.

Внешний вид программы каждый может разработать самостоятельно. Например, разместить на форме десять меток, три поля ввода и одну кнопку (Рис. 6.2).

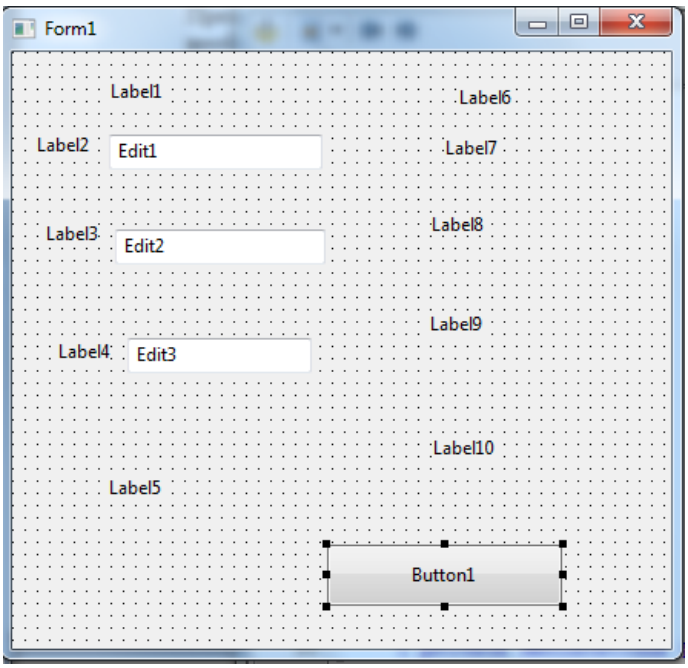


Рис. 6.2 - Вид формы с компонентами

Потребуется изменить заголовки компонентов (свойство ***Caption***) в соответствии с Рис. 6.3. Как это делать – мы уже знаем из предыдущей лабораторной работы.

Рис. 6.3 - Интерфейс программы

Двойной щелчок по кнопке **Вычислить** приведёт к созданию процедуры $TForm1.Button1Click$.

Задача программиста заполнить шаблон описаниями и операторами. Все команды, указанные в процедуре между словами $begin$ и end, будут выполнены при щелчке по кнопке **Выполнить**.

В нашем случае процедура $TForm1.Button1Click$ может иметь вид:

```
procedure TForm1.Button1Click(Sender: TObject);
 //Описание переменных (все переменные вещественного типа):
 var
   a, b, c, {стороны треугольника}
   alfa, betta, gamma, {углы треугольника}
   S,  площадь {треугольника}
   p: real; {полупериметр треугольника}
begin
//Из полей ввода Edit1, Edit2, Edit3 считываются введенные строки,
//с помощью функции StrToFloat(x) преобразовываются в вещественные числа
//и записываются в переменные a, b, c.
   a := StrToFloat(Edit1.Text);
   b := StrToFloat(Edit2.Text);
   c := StrToFloat(Edit3.Text);
   p := (a + b + c) / 2; //Вычисление значения полупериметра
//При вычислении значения площади применяется
// функция  sqrt(x) – корень квадратный из x.
    S := sqrt(p * (p-a) * (p-b) * (p-c));
//При вычислении значения угла alfa в радианах применяем функции:
// arccos(x) - арккосинус x и  sqr(x) – возведение x в квадрат
   alfa := arccos((sqr(b) + sqr(c) - sqr(a)) / 2 / b / c);
```

```
//При вычислении значения угла betta в радианах применяем функцию:
//arcsin(x) - арксинус x;
   betta := arcsin(b / a * sin(alfa));
//Вычисление значения угла gamma в радианах.
//Математическая постоянная определена функцией без аргумента pi.
   gamma := pi - (alfa + betta);
//Перевод радиан в радусы
   alfa := alfa * 180 / pi;
   betta := betta * 180 / pi;
   gamma := gamma *180 / pi;
//Для вывода результатов вычислений используем
//операцию слияния строк «+» и функцию FloatToStrF(x), которая
//преобразовывает вещественную переменную x в строку
//и выводит ее в указанном формате.
//В нашем случае под переменную отводится три позиции,
//включая точку и ноль позиций после точки.
//Величины углов в градусах выводятся на форму
//в соответствующие объекты типа надпись.
   Label6.Caption := 'alfa = ' + FloatToStrF(alfa, ffFixed, 3,0);
   Label7.Caption := 'betta = ' + FloatToStrF(betta,ffFixed,3,0);
   Label8.Caption:='gamma='+FloatToStrF(gamma,ffFixed,3,0);
//Используем функцию FloatToStrF(x) для форматированного вывода.
//В нашем случае под все число отводится пять позиций,
//включая точку, и две позиций после точки.
//Значения площади и периметра выводятся на форму.
   Label9.Caption := 'Периметр P = ' + FloatToStrF(2 * p, ffFixed, 5, 2);
   Label10.Caption := 'Площадь S = ' + FloatToStrF(S, ffFixed, 5, 2);
end;
```

Обратите внимание, что было написано всего десять команд, предназначенных для решения поставленной задачи, все остальное комментарий, который писать необязательно.

7 Подводя итог

1. **Объектно-ориентированное программирование (ООП)** – это технология создания сложного программного обеспечения, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого типа (класса), а классы образуют иерархию с наследованием свойств. Взаимодействие программных объектов в такой программе осуществляется путём передачи сообщений.

2. **Основными понятиями ООП являются:**

– класс – способ описания сущности, определяющий состояние и поведение, зависящее от этого состояния, а также правила для взаимодействия с данной сущностью;

– объект – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом; описывается совокупностью свойств и методов, а также событий, на которые он может реагировать;

– интерфейс – набор методов класса, доступных для использования другими классами;

– инкапсуляция – свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя;

– абстрагирование – способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые; абстракция – набор всех таких характеристик;

– полиморфизм (многообразие) – свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта;

– наследование – свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью; класс, от которого производится наследование, называется базовым или родительским, а новый класс – потомком, наследником или производным классом.

3. **В основу ООП положены следующие принципы:**

– абстрагирование – процесс выделения существенных характеристик некоторого объекта, которые отличают его от всех других видов объектов;

– ограничение доступа – сокрытие отдельных элементов реализации абстракции, не затрагивающих существенных характеристик ее как целого;

– модульность – принцип разработки программной системы, предполагающий реализацию ее в виде отдельных частей (модулей);

– иерархия – ранжированная или упорядоченная система абстракций;

– типизация – ограничение, накладываемое на свойства объектов и препятствующее взаимозаменяемости абстракций различных типов;

– параллелизм – свойство нескольких абстракций одновременно находиться в активном состоянии, т.е. выполнять некоторые операции;

– устойчивость – свойство абстракции существовать во времени независимо от процесса, породившего данный программный объект, и/или в пространстве, перемещаясь из адресного пространства, в котором он был создан.

4. **Интегрированная Среда Разработки** (ИСР или *Integrated Development Environment — IDE*) — это система программных средств, используемых программистами для разработки программного обеспечения. Обычно среда разработки включает в себя текстовый редактор, компилятор и/или интерпретатор, средства автоматизации сборки, отладчик.

8 Индивидуальные задания

8.1.1 Использование встроенных диалоговых окон InputBox, ShowMessage

Другой способ организации ввода и вывода данных – использование встроенных диалоговых окон ***InputBox***, ***ShowMessage***. Эти диалоговые окна не устанавливаются программистом на форму во время разработки. Операторы их активации нужно вставлять в программный код.

Функция ***InputBox()*** выводит на экран диалоговое окно, в котором можно ввести данные.

Аргументами этой функции являются три строки, а значением функции – строка введённая пользователем.

В общем виде строка программного кода с использованием функции ***InputBox*** выглядит так:

```
Переменная := InputBox('Заголовок', 'Подсказка', 'Значение по умолчанию');
```

где*:*

Переменная – переменная строкового типа, значение которой должно быть получено от пользователя;

Заголовок – текст заголовка окна;

Подсказка – текст поясняющего сообщения;

Значение по умолчанию – текст, который будет находиться в поле ввода, когда окно появиться на экране.

Например,

```
n := InputBox('Ввод числа', 'Введите число:', ' ');
```

Результат показан на рисунке:

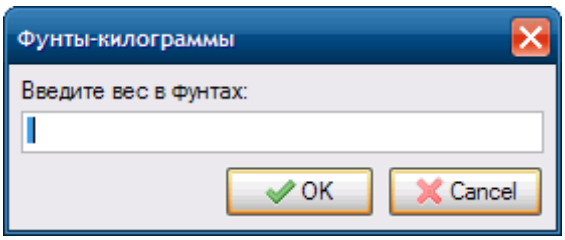
Если пользователь щёлкнет по кнопке OK, то значением функции станет строка, введённая пользователем в текстовое поле. Если пользователь щёлкнет по кнопке ***Cancel***, то значением функции станет строка «*Значение по умолчанию*».

Значение функции ***InputBox*** всегда строкового типа (***String***), поэтому в случае, если нужно ввести число, то введённая строка должна быть преобразована в число при помощи соответствующей функции преобразования.

В качестве примера возьмём задачу пересчёта веса из фунтов в килограммы

Ввод исходных данных из окна ввода и последующее преобразование данных может выглядеть так:

```
funtStr := InputBox('Фунты-килограммы', 'Введите вес в фунтах:', ' ');
funtFloat:=StrToFloat(funtStr);
```



Для вывода результата может использоваться процедура ***ShowMessage()***, которая выводит на экран диалоговое окно с текстом и командной кнопкой OK.

Общий вид инструкции вызова процедуры ***ShowMessage***:

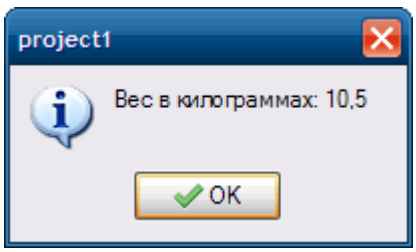
```
ShowMessage('Сообщение');
```

где ***Сообщение*** – текст, который будет выведен в окне.

Например, для того чтобы вывести результат в программе пересчета веса из фунтов в килограммы, можно использовать следующую строку кода:

```
ShowMessage('Вес в килограммах: '+ FloatToStr(kg));
```

Результат показан на рисунке.



При использовании для вывода компонента ***TLabel*** (Поле вывода), в который будет осуществляться вывод, устанавливают на **Форме** во время разработки. Содержание этого поля определяется значением свойства ***Caption***.

Для того чтобы вывести результаты в это поле, нужно в программном коде поместить оператор присваивания, который будет изменить значение свойства ***Caption*** на нужное вам значение.

Например, для того чтобы вывести результат в поле вывода ***Label1*** в рассмотренной выше задаче, нужно использовать следующий оператор присваивания:

```
Label1.Caption:=FloatToStr(kg) + 'кг';
```

8.2 Рекомендации по составлению программы

После запуска IDE Lazarus создайте подкаталог для Вашего нового проекта. Затем загрузите ранее сделанный Вами шаблон программы. Возможно у Вас появятся новые мысли по модернизации программ-шаблона. Откорректируйте его, проверьте и сохраните под тем же именем.

Чтобы не испортить шаблон программы, переименуйте его и запишите в созданный подкаталог под новым именем, которое Вы пожелаете дать разрабатываемой программе.

При составлении программы обязательно предусмотреть:

– разумный выбор идентификаторов:

– многократный ввод данных при исполнении программы, т.е. возможность повторного счета при других исходных данных;

– простейший диалог типа «запрос-ответ» при вводе данных;

– необходимые комментарии в тексте программы;

– вывод результатов в удобном для пользователя виде (отформатированные результаты, размерность, цвет и т.п.);

– подготовку тестового примера, позволяющего доказать правильность работы Вашей программы.

Внимание!

- В большинстве случаев вычисления ведутся не в системе СИ, а в *согласованной системе единиц*!
- В формулах ничего менять не нужно.
- В то же время, чтобы получить значения реактивных сопротивлений в Омах, необходимо в формулы для расчёта индуктивного и ёмкостного сопротивлений все данные подставлять в системе СИ!

8.3 Варианты заданий для составления линейных программ

1. Рассчитать индуктивность L_0 [нГн] круглого витка со средним диаметром D [см] и диаметром провода d [см] и его реактивное сопротивление X_{L_0} на частоте f по формулам

$$L_0 = 2\cdot\pi\cdot D\cdot\left(ln(8D/d)-1.75\right),\ X_{L_0} = 2\cdot\pi\cdot f\cdot L_0.$$

2. Рассчитать число витков w однослойной тороидальной катушки круглого сечения с индуктивностью L_0 [нГн] и ее реактивное сопротивление X_{L_0} на частоте f по формулам

$$w = \sqrt{L_0 \Big/ \left(2\cdot\pi\cdot\left(D-\sqrt{D^2-D_1^2}\right)\right)},\ \text{где}$$

D [см] – средний диаметр тора, а D_1 [см] – диаметр сечения тора.

3. Рассчитать число витков w однослойной тороидальной катушки прямоугольного сечения с индуктивностью L_0 [нГн] и ее реактивное сопротивление X_{L_0} на частоте f по формулам:

$$w = \sqrt{L_0/\left(2\cdot h\cdot ln(D_2/D_1)\right)},\quad X_{L_0} = 2\cdot\pi\cdot f\cdot L_0,\ \text{где}$$

D_1 [см] – внутренний диаметр тора, а D_2 [см] – внешний диаметр тора, h [см] - сторона квадрата сечения тора.

4. Рассчитать число витков w многослойной тороидальной катушки круглого сечения с индуктивностью L_0 [нГн] и ее реактивное сопротивление X_{L_0} на частоте f по формулам:

$$w = \sqrt{L_0/\left(2\cdot\pi\cdot D\cdot\left(ln(8\cdot D/D_1)-1.75\right)\right)},\ X_{L_0} = 2\cdot\pi\cdot f\cdot L_0,\ \text{где}$$

D [см] – средний диаметр тора, а D_1 [см] – диаметр сечения тора.

5. Рассчитать индуктивность L_0 [мкГн] однослойной катушки круглого сечения (без сердечника) и ее реактивное сопротивление X_{L_0} на частоте f по формулам:

$$L_0 = 0.394\cdot r^2\cdot w^2/(9\cdot r+10\cdot l),\quad X_{L_0} = 2\cdot\pi\cdot f\cdot L_0,\ \text{где}$$

w - число витков, r [см] – радиус катушки, l [см] – длина намотки ($l > 0.666\cdot r$)

6. Рассчитать число витков w однослойной катушки круглого сечения (без сердечника) и ее реактивное сопротивление X_{L_0} на частоте f по формулам:

$$w = \sqrt{L_0\cdot(9\cdot r+10\cdot l)/0.394}\,/\,r,\quad X_{L_0} = 2\cdot\pi\cdot f\cdot L_0,\ \text{где}$$

L_0 [мкГн] - индуктивность, r [см] – радиус катушки индуктивности, l [см] – длина намотки ($l \geq 0.666\cdot r$).

7. Рассчитать число витков w однослойной катушки круглого сечения (без сердечника) с индуктивностью L_0 [мкГн] и ее реактивное сопротивление X_{L_0} на частоте f по формулам:

$$w = 12.7\cdot L_0\cdot\left(1+\sqrt{\cdot 0.14\cdot r^3\cdot p^2/L_0}\right)\Big/\left(p\cdot r^2\right),\quad X_{L_0} = 2\cdot\pi\cdot f\cdot L_0,\ \text{где}$$

r [см] – радиус катушки, $p = w / l$ – количество витков на единицу длины намотки катушки, l [см] (определяется размерами провода и толщиной изоляции).

8. Рассчитать индуктивность L_0 [мкГн] короткой катушки (без сердечника) по формуле и ее реактивное сопротивление X_{L_0} на частоте f по формулам:

$$L_0 = 0.394 \cdot r^2 \cdot w^2 / (r \cdot (9 - 0.2 \cdot r/l) + 10 \cdot l), \quad X_{L_0} = 2 \cdot \pi \cdot f \cdot L_0,$$ где

w - число витков, *r* [см] – радиус катушки, *l* [см] – длина намотки (*0.1·r ≤ l ≤ 0.666·r*).

9. Рассчитать число витков *w* короткой катушки (без сердечника) по формуле:

$$w = \sqrt{(r \cdot (9 - 0.2 \cdot r/l) + 10 \cdot l) \cdot L_0 / 0.394} / r, \quad X_{L_0} = 2 \cdot \pi \cdot f \cdot L_0,$$ где

L_0 [мкГн] - индуктивность, *r* [см] – радиус катушки, *l* [см] – длина намотки катушки (*0.1·r ≤ l ≤ 0.666·r*).

10. Рассчитать индуктивность L_0 [мкГн] катушки с многослойной обмоткой (без сердечника) и ее реактивное сопротивление X_{L_0} на частоте *f* по формулам:

$$L_0 = 0.315 \cdot r^2 \cdot w^2 / (6 \cdot r + 9 \cdot l + 10 \cdot d), \quad X_{L_0} = 2 \cdot \pi \cdot f \cdot L_0,$$ где

w - число витков, *r* [см] – средний радиус катушки, *l* [см] – длина намотки, *d* [см] – радиальная толщина обмотки.

11. Рассчитать число витков *w* катушки с многослойной обмоткой (без сердечника) и ее реактивное сопротивление X_{L_0} на частоте *f* по формулам:

$$w = \sqrt{(6 \cdot r + 9 \cdot l + 10 \cdot d) \cdot L_0 / 0.315} / r, \quad X_{L_0} = 2 \cdot \pi \cdot f \cdot L_0,$$ где

L_0 [мкГн] - индуктивность, *r* [см] – средний радиус катушки, *l* [см] – длина намотки, *d* [см] – радиальная толщина обмотки.

12. Рассчитать число витков *w* катушки с замкнутым сердечником и ее реактивное сопротивление X_{L_0} на частоте *f* по формулам:

$$w = \sqrt{L_0 l_m (1 + \mu d_l / a l_m) / 4\pi\mu F_c}, \quad X_{L_0} = 2 \cdot \pi \cdot f \cdot L_0,$$ где

L_0 [мкГн] - индуктивность, d_l[см] – средний радиус катушки, l_m [см] – длина средней силовой линии, *a* [см] – толщина сердечника, F_c. [см^2]- площадь поперечного сечения сердечника, μ - магнитная проницаемость.

13. Рассчитать индуктивность L_0 [мкГн] прямого провода длиной *l* [см] и диаметром *d* [см] и его реактивное сопротивление на частоте *f* по формулам:

$$L_0 = 0.002 \cdot l \cdot [2.3 log(4 \cdot l / d - 0.75)], \quad X_{L_0} = 2 \cdot \pi \cdot f \cdot L_0.$$

14. Определить ёмкость *C* [пФ] бумажного конденсатора с параллельными прямоугольными пластинами и его реактивное сопротивление X_C на частоте *f*:

$$C = \varepsilon \cdot S / (11.3 \cdot d), \quad X_C = 1 / (2 \cdot \pi \cdot f \cdot C),$$ где

$S = a \cdot b$ [см^2] - площадь пластины, *a* и *b* [см] – размеры пластины, *d* [см] – расстояние между ними, ε - относительная диэлектрическая проницаемость (для бумаги $\varepsilon = 4$).

15. Определить ёмкость *C* [пФ] коаксиального кабеля длиной *l* [м] и его реактивное сопротивление X_C на частоте *f*:

$$C = 24.5 \cdot \varepsilon \cdot l / log(D/d), \quad X_C = 1 / (2 \cdot \pi \cdot f \cdot C),$$ где

D [мм] – наружный диаметр кабеля, *d* [мм] – внутренний диаметр кабеля, ε - относительная диэлектрическая проницаемость ($\varepsilon = 2.23$).

16. Определить реактивное X_C и полное Z_C сопротивления конденсатора ёмкостью *C* с внутренним сопротивлением $R_{посл}$ на частоте *f* (все параметры заданы в системе Си), а также его затухание *d* и добротность *Q*:

$$X_C = 1/(2\cdot\pi\cdot f\cdot C),\ \ Z_c = \sqrt{R_{посл}^2 + X_C^2},\ d = \frac{R_{посл}}{X_C}\cdot 100\%,\ Q = \frac{1}{d}.$$

17. Определите резонансную частоту f_P, добротность Q, ширину полосы пропускания Δf, ток I и напряжения $U_{Rпосл}$, U_L и U_C при резонансе на элементах L (индуктивность), C (емкость) и $R_{ПОСЛ}$ (внутреннее сопротивление) последовательного резонансного контура, питающегося от генератора с напряжением $U_Г$:

$$f_p = \frac{1}{2\pi\sqrt{LC}},\ I = \frac{U_Г}{R_{посл}},\ Q = \frac{2\pi f L}{R_{посл}},\ U_L = U_C = U_Г\cdot Q,\ \Delta f = \frac{f_p}{Q}.$$

18. Вычислить медианы треугольника со сторонами a, b, c по формулам:

$$m_a = 0.5\cdot\sqrt{2b^2 + 2c^2 - a^2},\ \ m_b = 0.5\cdot\sqrt{2a^2 + 2c^2 - b^2},$$
$$m_c = 0.5\cdot\sqrt{2a^2 + 2b^2 - c^2}.$$

19. Вычислить биссектрисы треугольника со сторонами a, b, c по формулам:

$$\beta_a = 2\cdot\sqrt{b\cdot c\cdot p\cdot(p-a)}/(b+c),\ \beta_b = 2\cdot\sqrt{a\cdot c\cdot p\cdot(p-b)}/(a+c),$$
$$\beta_c = 2\cdot\sqrt{a\cdot b\cdot p\cdot(p-c)}/(b+a),\ \text{где}\ \ p = (a+b+c)/2.$$

20. Вычислить координаты центра тяжести трех материальных точек с массами m_1, m_2, m_3 и координатами (x_1, y_1), (x_2, y_2), (x_3, y_3) по формулам:

$$x_c = (m_1\cdot x_1 + m_2\cdot x_2 + m_3\cdot x_3)/(m_1 + m_2 + m_3),$$
$$y_c = (m_1\cdot y_1 + m_2\cdot y_2 + m_3\cdot y_3)/(m_1 + m_2 + m_3).$$

21. Вычислить координаты точки, делящей отрезок a_1a_2 в отношении $n_1: n_2$ по формулам:

$$x = (x_1 + \lambda\cdot x_2)/(1+\lambda),\ \ y = (y_1 + \lambda\cdot y_2)/(1+\lambda\cdot),\ \ \text{где}\ \lambda = \mathrm{n}_1/n_2.$$

22. Вычислить площадь поверхности $S = \pi\cdot(R+r)\cdot l + \pi\cdot R^2 + \pi\cdot r^2$ и объем $V = \pi\cdot h\cdot(R^2 + r^2 + Rr)/3$ усечённого конуса.

23. Составить программу для вычисления расстояний между двумя точками с координатами (x_1, y_1, z_1) и (x_2, y_2, z_2) в трёхмерном пространстве по формуле:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}.$$

24. Составить программу для вычисления значений функций:

$$y = \frac{e^{-x_1} + e^{-x_2}}{2}\ \text{и}\ z = \frac{a\sqrt{x_1} - b\sqrt{x_2}}{c},\ \ \text{где}$$
$$x_1 = \left(b + \sqrt{|b^2 - 4ac|}\right)/(2a),\ \ x_2 = \left(b - \sqrt{|b^2 - 4ac|}\right)/(2a).$$

25. Написать программу вычисления стоимости поездки на автомобиле на дачу (туда и обратно). Исходными данными являются: расстояние до дачи (в километрах); количество бензина, которое потребляет автомобиль на 100 км пробега; цена одного литра бензина.

26. Три сопротивления $R1, R2$ и $R3$ соединены параллельно. Найти общее сопротивление соединения $R0$

27. Даны два числа. Найти среднее арифметическое кубов этих чисел и среднее геометрическое модулей этих чисел.

28. Вычислить расстояние между двумя точками с данными координатами ($x1, y1$) и ($x2, y2$).

29. Дана сторона равностороннего треугольника. Найти площадь этого треугольника, его высоту, радиусы вписанной и описанной окружностей.

30. Заданы координаты трёх вершин треугольника ($x1, y1$), ($\mathrm{x}2, y2$), ($x3, y3$). Найти его периметр и площадь.

9 Список литературы

1. Алексеев Е.Р., Чеснокова О.В., Кучер Т.В. Free Pascal и Lazarus: Учебник по программированию / Е.Р. Алексеев, О.В. Чеснокова, Т.В. Кучер. - М.: Издательский дом ДМК-пресс, 2010. - 440 с.

2. Алексеев Е.Р., Чеснокова О.В., Кучер Т.В.. Самоучитель по программированию на Free Pascal и Lazarus.. - Донецк: ДонНТУ, Технопарк ДонНТУ УНИТЕХ, 2011. - 503 с.

3. Кетков Ю.Л. Свободное программное обеспечение. FREE PASCAL для студентов и школьников / Ю.Л. Кетков, А.Ю. Кетков. — СПб.: БХВ-Петербург, 2011. — 384 с.

4. Мансуров К.Т. Основы программирования в среде Lazarus. - М.: Нобель пресс, 2013. – 772 с.

5. Фаронов В.В. Turbo Pascal. Наиболее полное руководство (в подлиннике). — СПб.: БХВ-Петербург, 2004. — 1056 с.

6. Фленов М.Е. Библия Delphi. — 2-е изд., перераб. и доп. - СПб.: БХВ-Петербург, 2008. - 800 с.

7. Lazarus Tutorial/ru [Электронный ресурс] // База знаний о Free Pascal, Lazarus и родственных проектах: [сайт]. URL: http://wiki.freepascal.org/Lazarus_Tutorial/ru

8. Программирование на Lazarus [Электронный ресурс] // «ИНТУИТ» Национальный открытый университет: [сайт]. URL: http://www.intuit.ru/studies/courses/13745/1221/lecture/23276?page=1

9. ОС ТУСУР 01-2013 (СТО 02069326.1.01-2013). Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления. - Томск: ТУСУР, 2013. – 57 с.

10. Кобрин Ю.П. Основные понятия языка Free Pascal. - Томск: ТУСУР, кафедра КИПР, 1916. - 37 с/.