# Содержание

# Введение

Авторы книги давно хотели написать учебник по программированию, предназначенный для пользователей различных операционных систем.

Учебник основан на курсе, который авторы читали в Донецком национальном техническом университете (ДонНТУ) студентам общеинженерных специальностей. Многие годы в ДонНТУ в качестве языка обучения программированию будущих инженеров используется Pascal, который является ясным, логичным и гибким языком и приучает к хорошему стилю программирования. Кроме того, в средней школе основы программирования преподают именно на базе Pascal. Вместе с тем именно Pascal лежит в основе современной мощной системы визуального программирования Delphi, с помощью которой разрабатываются многие современные программные продукты.

В учебнике используется язык программирования Free Pascal, компиляторы с которого являются свободно распространяемыми. Free Pascal является очень мощным средством программирования, и вместе с тем за использование компиляторов студенту, школьнику и преподавателю не придется платить. Этим компилятором можно пользоваться абсолютно легально.

Свободно распространяемые компиляторы с языка Free Pascal реализованы во многих дистрибутивах Linux, есть свободные компиляторы и для ОС Windows. Кроме того, в этой книге мы попытались познакомить читателя с принципами создания визуальных приложений в среде Lazarus. Бурно развивающаяся среда визуального программирования Lazarus является серьезным конкурентом Delphi.

В настоящее время существует множество подходов к изучению программирования. По мнению авторов, нельзя изучать программирование на каком-либо языке, не изучив методы разработки алгоритмов. Как свидетельствует опыт авторов, одним из наиболее наглядных методов составления алгоритмов является язык *блок-схем*. Мы попытались написать учебник по алгоритмизации и программированию. Насколько нам это удалось судить читателю.

Авторы надеются, что читатель имеет первоначальные навыки работы на персональном компьютере под управлением ОС Linux или Windows и знаком со школьным курсом математики.

Книга состоит из десяти глав.

В *первой главе* читатель узнает о средствах разработки программ на Free Pascal, напишет свои первые программы.

Во *второй* главе изложены основные элементы языка (переменные, выражения, операторы) Free Pascal. Описаны простейшие операторы языка: присваивания и ввода-вывода. Приведена структура программы на Free Pascal, а также примеры программ линейной структуры.

*Третья глава* является одной из ключевых в изучении программирования. В ней изложена методика составления алгоритмов с помощью блок-схем. Приведено большое количество примеров алгоритмов и программ различной сложности. Авторы рекомендуют внимательно разобрать все примеры и выполнить упражнения этой главы и только после этого приступать к изучению последующих глав книги.

В *четвертой* главе читатель на большом количестве примеров познакомится с подпрограммами. Описан механизм передачи параметров между подпрограммами. Один из параграфов посвящен рекурсивным подпрограммам. В завершении главы рассмотрен вопрос создания личных модулей.

*Пятая* и *шестая* главы посвящены алгоритмам обработки массивов и матриц. Здесь же читатель познакомится и с реализацией этих алгоритмов на языке Free Pascal. Именно эти главы совместно с третьей являются ключом к пониманию принципов программирования.

*Седьмая* глава знакомит читателя с обработкой файлов на языке Free Pascal под управлением операционных систем Linux и Windows. На практических примерах изложен механизм прямого и последовательного доступа к файлам и обработки ошибок ввода-вывода. Описана работа с бестиповыми и текстовыми файлами.

*Восьмая* глава посвящена обработке строк и записей. Приведенные примеры позволят читателю разобраться с принципами обработки таблиц в языке Free Pascal.

В *девятой* главе авторы описали принципы объектно-ориентированного программирования и их реализацию в языке Free Pascal.

В *десятой* главе описаны графические возможности Lazarus, подробно описан алгоритм построения графиков непрерывных функций на экране дисплея. Приведены тексты программ изображения графиков функций с подробными комментариями.

К каждой теме прилагаются 25 вариантов задач для самостоятельного решения, что позволит использовать книгу не только начинающим самостоятельно изучать программирование, но и преподавателям в учебном процессе.

С рабочими материалами книги можно познакомиться на сайтах Евгения Ростиславовича Алексеева — www.teacher.dn-ua.com, www.teacher.ucoz.net.

Авторы выражают благодарность своим родным за помощь и понимание.

Алексеев Е.Р., Чеснокова О.В., Кучер Т.В.
Донецк, ноябрь 2009 г.

## Сведения об авторах

*Алексеев Евгений Ростиславович* — кандидат технических наук, доцент кафедры «Вычислительная математика и программирование» Донецкого национального технического университета, доцент кафедры «Документоведение и информационная деятельность» Донецкого инженерно-экономического колледжа. Преподавательский стаж Алексеева Е.Р. — 17 лет. Евгений Ростиславович — автор двенадцати книг, вышедших в Москве и Донецке, общий тираж которых превышает 65 тыс. экземпляров. Е.Р. Алексеев — автор более 60 научных и методических работ. Область его научных интересов — программирование, вычислительная математика, Интернет-технологии, использование свободно распространяемого программного обеспечения.

*Чеснокова Оксана Витальевна* — старший преподаватель кафедры «Вычислительная математика и программирование» Донецкого национального технического университета. Преподавательский стаж Чесноковой О.В. — 15 лет. Оксана Витальевна — автор девяти книг, общий тираж которых превышает 40 тыс. экземпляров, а также — около 40 научных и методических работ. Область ее научных интересов — программирование, вычислительная математика.

*Кучер Татьяна Викторовна* — ассистент кафедры «Вычислительная математика и программирование» Донецкого национального технического университета, ассистент кафедры «Документоведение и информационная деятельность» Донецкого инженерно-экономического колледжа. Татьяна Викторовна — автор 30 научных и методических работ. Преподавательский стаж Кучер Т. В. — 14 лет. Область ее научных интересов — программирование, вычислительная математика.

# 1 Средства разработки программ на языке Free Pascal

В этой главе мы начинаем знакомство с программированием на языке Free Pascal. *Язык программирования* Free Pascal ведет свое начало от классического языка Pascal, который был разработан в конце 60-х годов XX века Николасом Виртом. Он разрабатывал этот язык как учебный язык для своих студентов. С тех пор Pascal, сохранив простоту и структуру языка, разработанного Н. Виртом, превратился в мощное средство программирования. С помощью современного языка Pascal можно производить простые расчеты, разрабатывать программы для проведения сложных инженерных и экономических вычислений.

## *1.1 Процесс разработки программы*

Разработку программы можно разбить на следующие этапы:

1. Составление алгоритма решения задачи. *Алгоритм* — это описание последовательности действий, которые необходимо выполнить для решения поставленной задачи.

2. Написание текста программы. *Текст программы* пишут на каком-либо языке программирования (например на Free Pascal) и вводят его в компьютер с помощью текстового редактора.

3. Отладка программы. *Отладка программы* — это процесс устранения ошибок из текста программы. Все ошибки делятся на синтаксические и логические. При наличии синтаксических ошибок (ошибок в написании операторов) программа не запускается. Подобные ошибки исправляются проще всего. Логические ошибки — это ошибки, при которых программа работает, но неправильно, выдавая не те результаты, которые ожидает разработчик или пользователь. Логические ошибки исправить сложнее, чем синтаксические, иногда для этого приходится переписывать отдельные участки программы, а иногда перерабатывать весь алгоритм.

4. Тестирование программы. *Тестирование программы* — процесс выявления ошибок в ее работе.

Процессы отладки и тестирования сопровождаются неоднократным *запуском программы на выполнение*. Процесс запуска может быть осуществлен только после того, как введенная в компьютер про-

грамма на алгоритмическом языке Pascal[1] будет переведена в *двоичный машинный код* и создан *исполняемый файл*.

Процесс перевода текста программы в машинный код называют *трансляцией*. Все трансляторы делятся на два класса:

- *интерпретаторы* — трансляторы, которые переводят каждый оператор программы в машинный код и по мере перевода операторы выполняются процессором;
- *компиляторы* переводят всю программу целиком, и если этот перевод прошел без ошибок, то полученный двоичный код можно запускать на выполнение.

Если в качестве транслятора выступает компилятор, то процесс перевода текста программы в машинный код называют *компиляцией*. При переводе программы с языка Pascal в машинный код используются именно компиляторы[2].

Рассмотрим основные этапы обработки компилятором программы на языке Pascal.

1. Компилятор анализирует, какие внешние библиотеки[3] нужно подключить, разбирает текст программы на составляющие элементы, проверяет синтаксические ошибки и в случае их отсутствия формирует объектный код (в Windows — файл с расширением **.obj**, в Linux — файл с расширением **.o**). Получаемый на этом этапе двоичный файл (объектный код) не включает в себя объектные коды подключаемых библиотек.

2. На втором этапе компоновщик подключает к объектному коду программы объектные коды библиотек и генерирует исполняемый код программы. Этот этап называется *компоновкой*, или *сборкой*, *программы*. Полученный на этом этапе исполняемый код можно запускать на выполнение.

На сегодняшний день существует множество компиляторов языка Pascal, среди которых можно выделить компиляторы языка Pascal разработки фирмы Borland (Borland Pascal), а также свободно распространяемые кроссплатформенные компиляторы языка Free Pascal и среду визуального программирования Lazarus.

---

1 Как и на любом другом языке.

2 Вместо термина «компилятор» в литературе иногда используют термин «транслятор компилирующего типа».

3 В библиотеках языка Pascal хранится объектный (двоичный) код стандартных (таких, как sin(x), cos(x) и др.) функций и процедур языка.

## 1.2 Среда программирования Free Pascal

Рассмотрим *процесс установки компилятора* Free Pascal в ОС Linux. Для установки программ в операционной системе Linux служит менеджер пакетов Synaptic. В ОС Ubuntu он вызывается с помощью команды **Система — Администрирование — Менеджер пакетов Synaptic**. В ALT Linux менеджер пакетов вызывается командой **Система — Программа управления пакетами Synaptic**. Окно Synaptic представлено на рис. 1.1.

Обратите внимание, что для установки программ необходимо установить список источников программ (список репозиториев[4]). После установки ОС Ubuntu Linux первоначальный список репозиториев установлен. В ОС ALT Linux первоначальный список репозиториев надо установить.

Для установки окне Synaptic (см. рис. 1.1) необходимо щелкнуть по кнопке **Найти**. И в открывшемся окне ввести **fpc** (см. рис. 1.2). Менеджер программ находит программу Free Pascal, после чего в окне Synaptic необходимо пометить программы *fpc* (Free Pascal Compiler Meta Package) для установки (с помощью контекстного меню или с помощью кнопки **Отметить для обновления**) и начать установку, щелкнув по кнопке **Применить**. После этого начнется процесс скачивания пакетов из Интернета и их установки.

В состав метапакета `fpc` входит компилятор языка Free Pascal `fpc` и среда разработки `fp-ide`. Для запуска среды разработки в Linux необходимо просто в терминале набрать `fp`. На рис. 1.3 представлено окно среды разработки программ на языке Free Pascal в ОС Linux.

Для установки Free Pascal в операционной системе Windows необходимо запустить скачанный со страницы загрузки http://www.-freepascal.org/down/i386/win32.var инсталяционный файл. Первое диалоговое окно сообщит о начале процесса установки Free Pascal на компьютер. Для продолжения процесса установки во всех следующих окнах нужно выбирать кнопку **Next**, для возврата к предыдущему шагу — кнопку **Back**, а для прерывания процесса установки — кнопку **Cancel**. В следующем окне нужно определить путь для установки Free Pascal. По умолчанию установка происходит в корневой каталог диска C. Для выбора другого пути установки можно воспользоваться

4 Список репозиториев — список официальных сайтов, с которых можно устанавливать программы.

кнопкой **Browse...**. Кроме того, в этом окне выводится информация о количестве свободного места на диске. В следующих четырех окнах пользователь сможет выбрать из списка тип установки: **Full Installation** (полная), **Minimum Installation** (минимальная), **Custom Installation** (выбор компонентов), указать название устанавливаемого приложения в главном меню, выбрать типы файлов, поддерживаемых средой, и начать процесс инсталляции Free Pascal, нажав кнопку **Install**. Контролировать процесс установки можно с помощью линейного индикатора, а прервать кнопкой **Cancel**.

Запуск среды программирования Free Pascal в Windows можно осуществить из главного меню: **Пуск — Программы — Free Pascal — Free Pascal**. На экране появится окно, представленное на рис. 1.4.

Установив пакет Free Pascal, мы получили компилятор и среду программирования.

Компилятор Free Pascal работает в командной строке. Для того чтобы создать исполняемый файл из текста программы, написанного на языке Pascal, необходимо выполнить команду

```
fpc name.pas
```

здесь `fpc` — имя исполняемого файла компилятора командной строки Free Pascal, `name.pas` — имя файла с текстом программы. В результате в Linux будет создан исполняемый файл с именем `name` (в Windows — имя исполняемого файла `name.exe`).

При использовании компилятора fpc после компиляции автоматически происходит компоновка программы (запуск компоновщика make).

Технология работы с компилятором Free Pascal может быть такой: набираем текст программы в стандартном текстовом редакторе, затем в консоли запускаем компилятор, после исправления синтаксических ошибок повторно запускаем компилятор. После успешной компиляции запускаем исполняемый файл. При необходимости опять вносим изменения в текст программы и запускаем компилятор. При такой технологии работы с компилятором необходимо не забывать сохранять текст программы, иначе при запуске компилятора будет компилироваться старая версия текста программы.

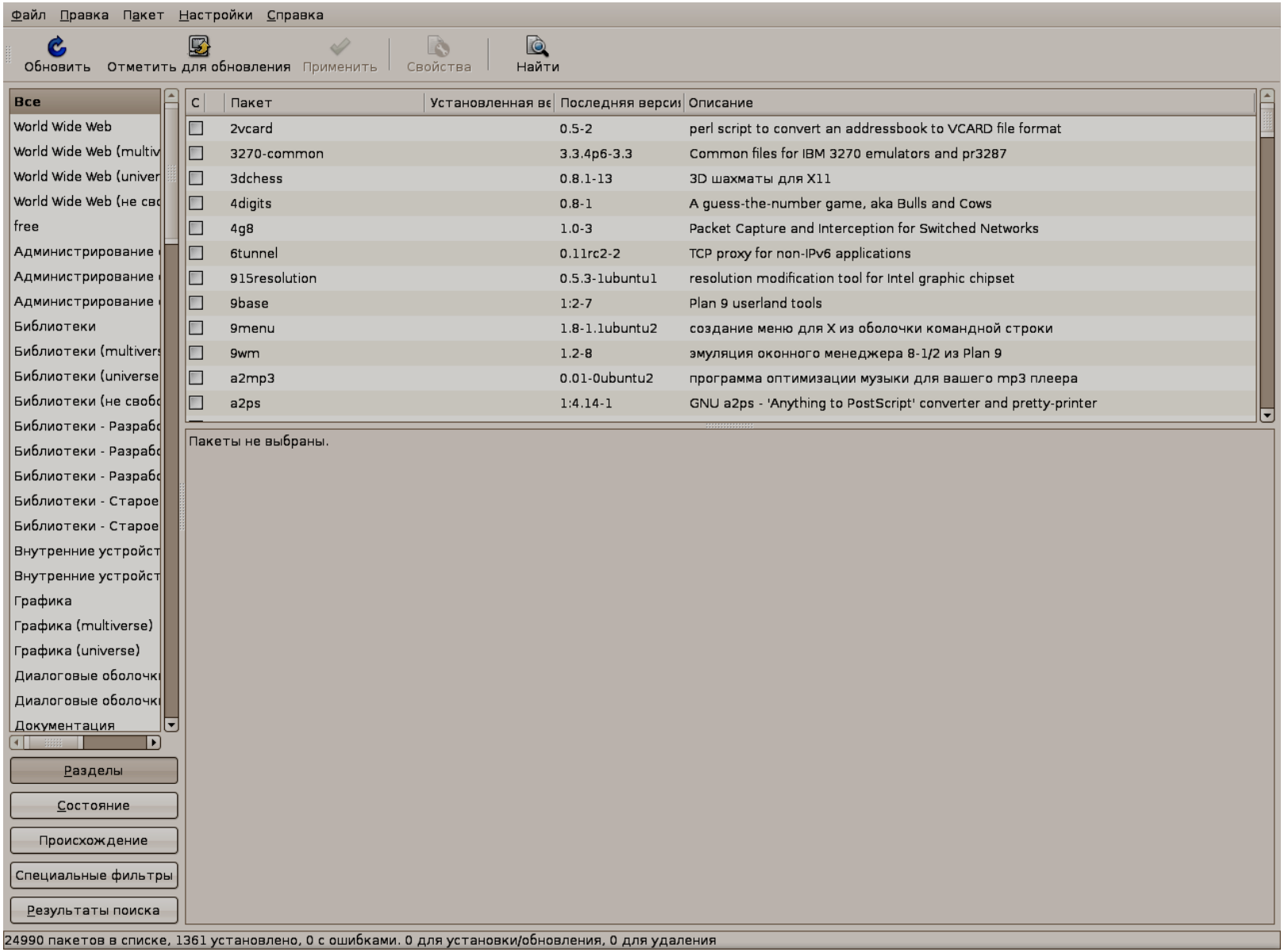


*Рисунок 1.1: Менеджер пакетов Synaptic*

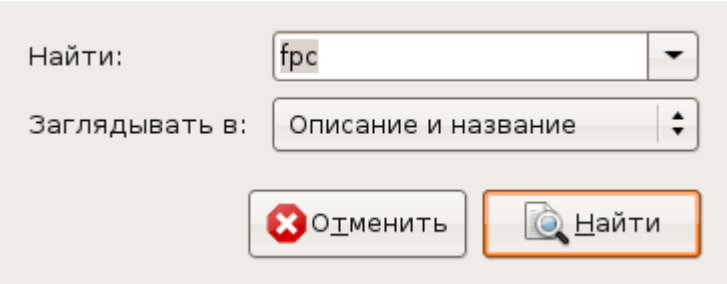


*Рисунок 1.2: Окно поиска компилятора Free Pascal в Synaptic*


*Рисунок 1.3: Среда программирования Free Pascal в ОС Linux*


*Рисунок 1.4: Окно компилятора Free Pascal*

Среда программирования Free Pascal позволяет значительно упростить процесс разработки программ. В состав среды программирования входит текстовый редактор, транслятор и отладчик. Рассмотрим их работу подробнее.

### 1.2.1 Работа в текстовом редакторе Free Pascal

С помощью редактора Free Pascal можно создавать и редактировать тексты программ. После открытия пустого окна (**File — New**) или загрузки текста программы (**File — Open**) пользователь находится в *режиме редактирования*, признаком чего является наличие в окне *курсора* (небольшого мигающего прямоугольника). Для перехода из режима редактирования к главному меню нужно нажать клавишу **F10**, обратно — **Esc**. Кроме того, этот переход можно осуществить щелчком мыши либо по строке главного меню, либо по полю редактора.

Редактор Free Pascal обладает возможностями, характерными для большинства текстовых редакторов. Остановимся на некоторых особенностях.

*Работа с фрагментами текста* (блоками) в редакторе Free Pascal может осуществляться с помощью главного меню и функциональных клавиш.

В главном меню для работы с фрагментами текста предназначены команды *пункта редактирования* **Edit**:

**Copy (Ctrl+C)** — копировать фрагмент в буфер;
**Cut (Ctrl+X)** — вырезать фрагмент в буфер;
**Paste (Ctrl+V)** — вставить фрагмент из буфера;
**Clear (Ctrl+Del)** — очистить буфер;
**Select All** — выделить весь текст в окне;
**Unselect** — отменить выделение.

Команды **Copy** и **Cut** применяют только к выделенным фрагментам текста. *Выделить фрагмент текста* можно с помощью клавиши **Shift** и клавиш перемещения курсора (стрелок). Кроме того, пункт меню **Edit** содержит команды **Undo** и **Redo,** с помощью которых можно отменять и возвращать выполненные действия.

*Комбинации клавиш*, предназначенные для работы с блоком:

**Ctrl+K+B** – пометить начало блока;
**Ctrl+K+K** – пометить конец блока;
**Ctrl+K+T** – пометить в качестве блока слово слева от курсора;
**Ctrl+K+Y** – стереть блок;

**Ctrl+K+C** – копировать блок в позицию, где находится курсор;
**Ctrl+K+V** – переместить блок в позицию, где находится курсор;
**Ctrl+K+W** – записать блок в файл;
**Ctrl+K+R** – прочитать блок из файла;
**Ctrl+K+P** – напечатать блок;
**Ctrl+K+H** – снять пометку блока; повторное использование этой комбинации клавиш вновь выделит блок.

*Работа с файлами* в среде Free Pascal осуществляется с помощью команд **File** главного меню и функциональных клавиш:

**New** — открыть окно для создания новой программы;
**Open (F3)** — открыть ранее созданный файл;
**Save (F2)** — сохранить созданный файл;
**Save As** — сохранить файл под другим именем;
**Exit (Alt+X)** — выйти из среды программирования;

При *создании новой программы* ей по умолчанию присваивается стандартное имя NONAMEOO.PAS (NO NAME — нет имени).

При *первом сохранении файла* пользователю будет предложено ввести его имя. При *повторном сохранении* файл сохраняется под тем же именем. Команда **Save As** аналогична первому сохранению. Если файл не был сохранен, то при попытке завершить работу со средой появится запрос о необходимости сохранить изменения в файле. При *открытии ранее созданного файла* его имя выбирают из списка существующих файлов.

В редакторе Free Pascal допускается *работа с несколькими окнами*. Для переключения в окно с номером от первого до девятого нажать комбинацию клавиш **Alt+i**, где i – номер окна (например **Alt+5** — вызов пятого окна). Для вывода списка окон на экран нажать комбинацию клавиш **Alt+0**, появится с*писок активных окон*, в котором необходимо будет выбрать нужное и нажать **Enter**.

### 1.2.2 Запуск программы в среде Free Pascal и просмотр результатов

После того как текст программы был набран, его следует перевести в машинный код. Для этого необходимо *вызвать транслятор* с помощью команды **Compile — Compile** (комбинация клавиш **Alt+F9**). На первом этапе транслятор проверяет наличие синтаксических ошибок. Если в программе нет синтаксических ошибок, то на экране сообщается о количестве строк транслированной программы и объеме доступной оперативной памяти.

Если на каком-либо этапе транслятор обнаружит ошибку, то в окне редактора курсор указывает ту строку программы, где при трансляции обнаружена ошибка. При этом в верхней строке редактора появляется краткое диагностическое сообщение о причине ошибки.

Для *запуска транслированной программы* необходимо выполнить команду **Run — Run** (комбинация клавиш **Ctrl+F9**), после чего на экране появляется окно командной строки, в котором пользователь и осуществляет диалог с программой. После завершения работы программы пользователь опять видит экран среды Free Pascal.

Для *просмотра результатов работы программы* в ОС Windows необходимо нажать комбинацию клавиш **Alt+F5**. Для возврата в оболочку следует нажать любую клавишу.

При использовании среды программирования в ОС Linux существует проблема вывода кириллического текста в консольном приложении. Альтернативой среды программирования Free Pascal является текстовый редактор Geany (http://www.geany.org), который позволяет разрабатывать программы на различных языках программирования.

## 1.3 Текстовый редактор Geany

Текстовый редактор Geany есть в репозитарии большинства современных дистрибутивов Linux[5]. Разработка программ с использованием Geany довольна удобна. Установка Geany также может быть осуществлена с помощью менеджера пакетов Synaptic.

Последовательно рассмотрим основные этапы разработки программы с использованием Geany.

1. Необходимо *создать шаблон* приложения на Pascal (или другом языке программирования) с помощью команды **Файл — New (with Template) — Pascal source file**. После чего появится окно с шаблоном исходного кода (рис. 1.5), в котором необходимо ввести текст программы и сохранить его (рис. 1.6).

2. Для *компиляции* и *запуска программы* на выполнение служит пункт меню **Построить**. Для компиляции программы следует использовать команду **Построить — Собрать** (**F8**). В этом случае будут созданы файл с объектным кодом программы и исполняемый файл. После компиляции программы ниже окна с программой будет представлен подробный отчет (рис. 1.6) о результатах компиляции. Следует его внимательно изучить. Этот отчет позволит быстро исправлять

5 Существует версия Geany и для Windows (http://www.geany.org/Download/Releases)

синтаксические ошибки. В сообщении об ошибке указана строка, где она найдена и ее краткое описание на английском языке. В отчетах по результатам компиляции могут быть *ошибки* (*error*) и *сообщения* (*warning*). Сообщения — это обнаруженные компилятором неточности, при которых возможно создание исполняемого кода программы.

3. Для *запуска программы* следует выполнить команду **Построить — Выполнить (F5)**. После чего на экране появляется окно терминала (рис. 1.7), в котором можно вводить данные и увидеть результаты работы программы.

В редакторе `Geany` (хотя часто его называют и средой программирования) можно настроить команду вызова компиляции, компоновки и запуск. Для это служит команда **Построить — Установить включения и аргументы**. Это окно для работы с файлами с расширением pas представлено на рис. 1.8. При настройке строк **Compile** и **Запуск** следует учитывать, что `%f` — имя компилируемого файла, `%e` — имя файла без расширения.

Какую среду выбрать для разработки консольных программ на Free Pascal — это дело пользователя. Авторы советуют под управлением ОС Linux использовать `Geany`[6]. Хотя можно использовать для набора текста программы обычный текстовый редактор (например, `gedit`, `tea`, `kate` и др.), а компиляцию осуществлять в терминале. Под управлением Windows логичнее использовать `fp-ide`.

## 1.4 Среда визуального программирования Lazarus

Lazarus – это *среда визуального программирования*. Здесь программист получает возможность не просто создавать программный код, но и наглядно (визуально) показывать системе, что бы он хотел увидеть.

Технология визуального программирования позволяет строить интерфейс[7] будущей программы из специальных компонентов, реализующих нужные свойства. Количество таких компонентов достаточно велико. Каждый из них содержит готовый программный код и все необходимые для работы данные, что избавляет программиста от создания того, что уже создано ранее.

6 Это субъективный совет авторов.

7 Интерфейс – диалог, обмен информацией.

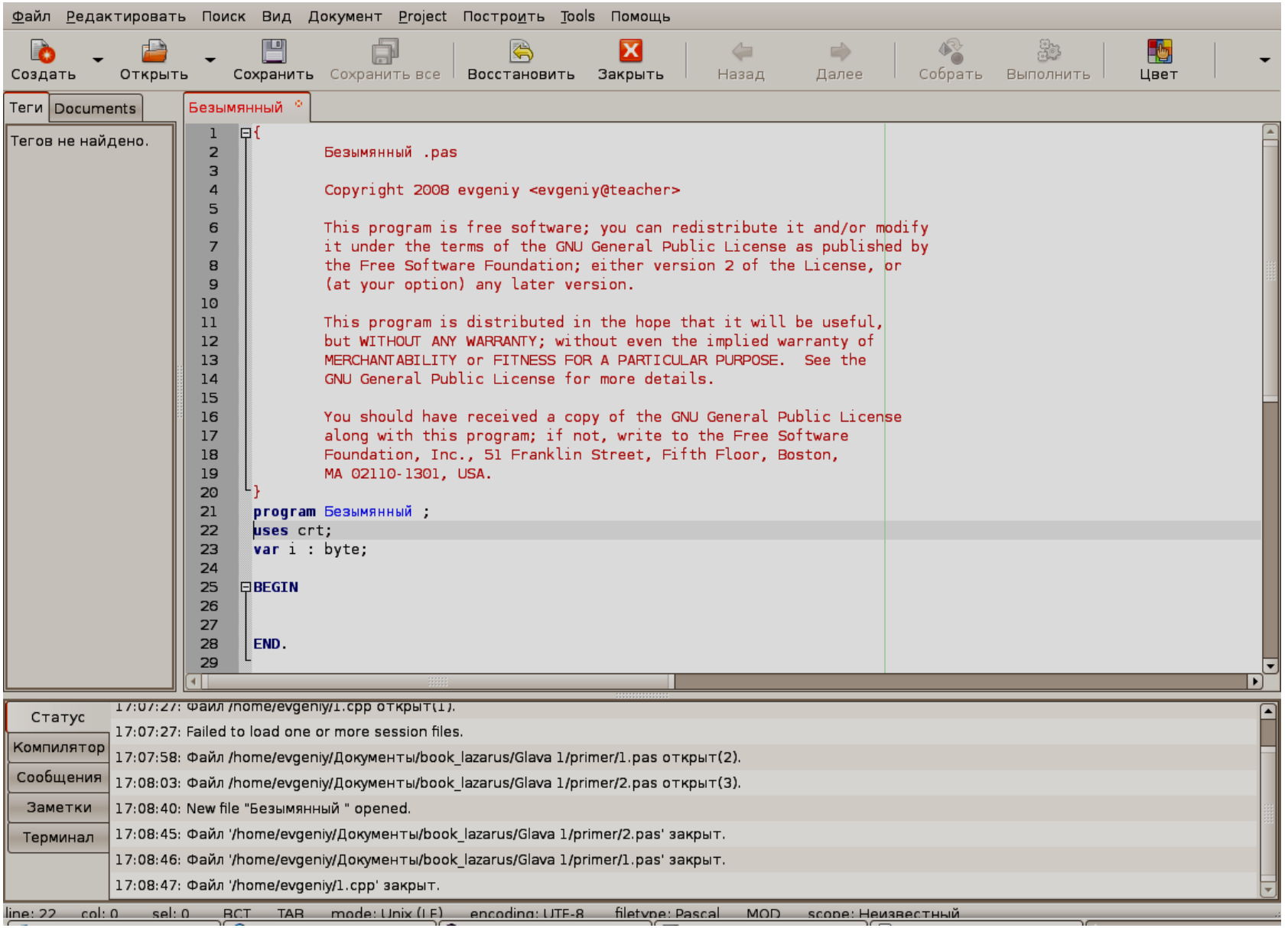


*Рисунок 1.5: Окно Geany с шаблоном программы на Free Pascal*

Подобный подход во много раз сокращает время написания программы. Кроме того, быстрота создания программного кода в Lazarus достигается за счет того, что значительная часть текста формируется автоматически.

Среда визуального программирования Lazarus сочетает в себе компилятор, объектно-ориентированные средства визуального программирования и различные технологии, облегчающие и ускоряющие создание программы.

### 1.4.1 Установка Lazarus в ОС Linux

Для установки Lazarus в окне Synaptic (см. рис. 1.1) необходимо щелкнуть по кнопке **Найти**. В появившемся окне поиска (см. рис. 1.9) вводим имена необходимых программ (*Lazarus, fpc, fpc-source*) и щелкаем по кнопке **Найти**.

Менеджер программ находит программы Lazarus и Free Pascal, после чего в *Lazarus, fpc, fpc-source* для установки (с помощью контекстного меню или с помощью кнопки **Отметить для обновления**) и начинает установку, щелкнув по кнопке **Применить**. После этого Synaptic предложит установить еще несколько пакетов, которые необходимы для установки Lazarus. Надо соглашаться. После этого начнется процесс скачивания файлов пакетов и установки Lazarus на компьютер. После установки запуск программы осуществляется с помощью команды меню **Программирование — Lazarus**[8].

Можно начинать работать. В старых версиях операционной системы Linux (например, Ubuntu 8.10 и более ранних) при запуске Lazarus вместо русских пунктов меню появятся непонятные символы.

Подробно о том, как добиться правильного отображения символов кириллицы в меню Lazarus, описано на следующих страницах:

http://www.freepascal.ru/article//lazarus/20080316091540/,
http://forum.sources.ru/index.php?showtopic=243159,
http://forum.ubuntu.ru/index.php?topic=18539.0;all.

Кроме того, можно использовать и запуск с английским интерфейсом командой терминала `LANG=C startlazarus`.

Но, на взгляд авторов, наиболее универсальным и простым методом добиться корректного отображения символов кириллицы будет следующий.

8 Не исключено, что вызов Lazarus в других дистрибутивах Linux можно осуществлять и с помощью другой команды главного меню.

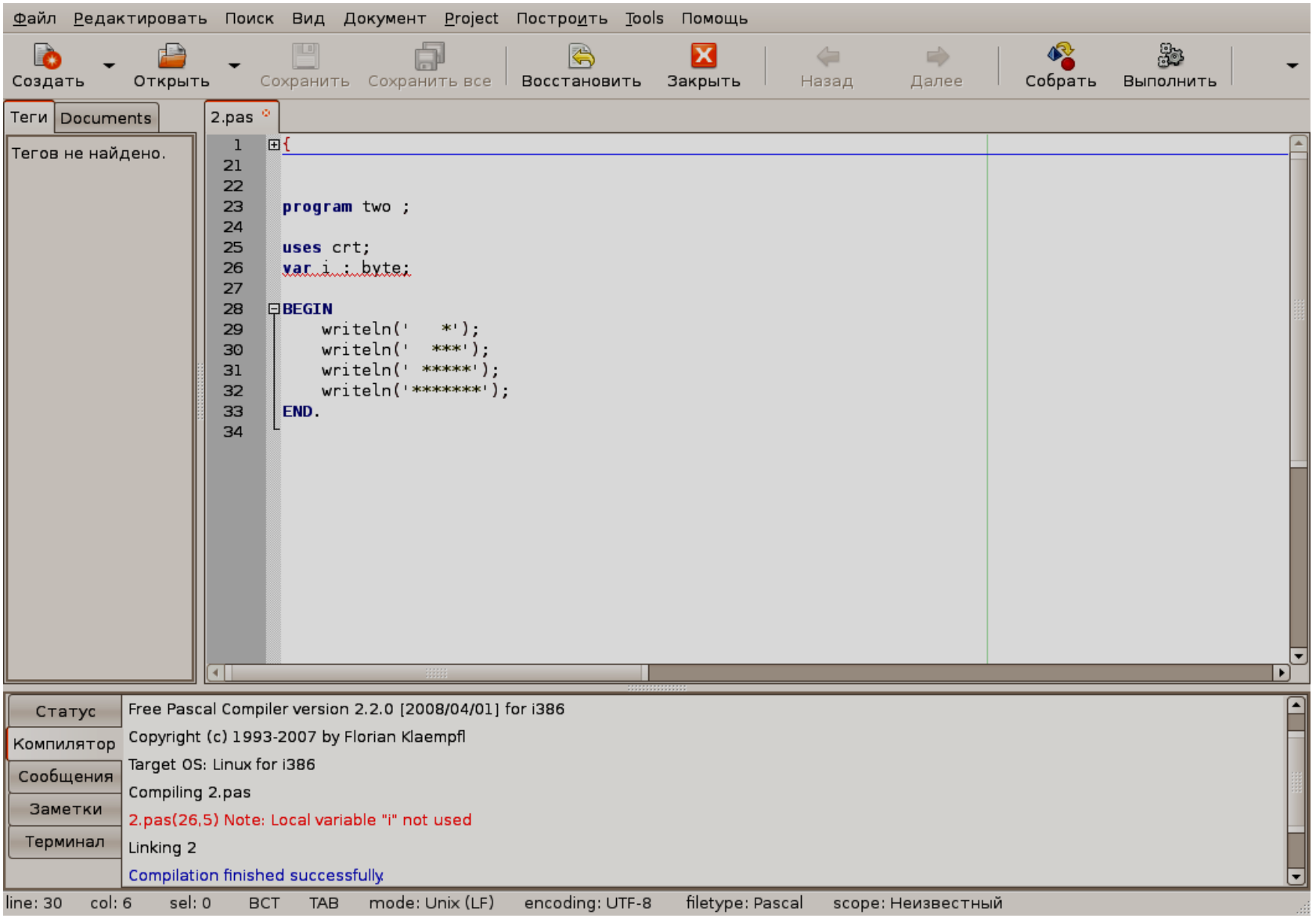


*Рисунок 1.6: Окно Geany с текстом программы на языке Free Pascal*

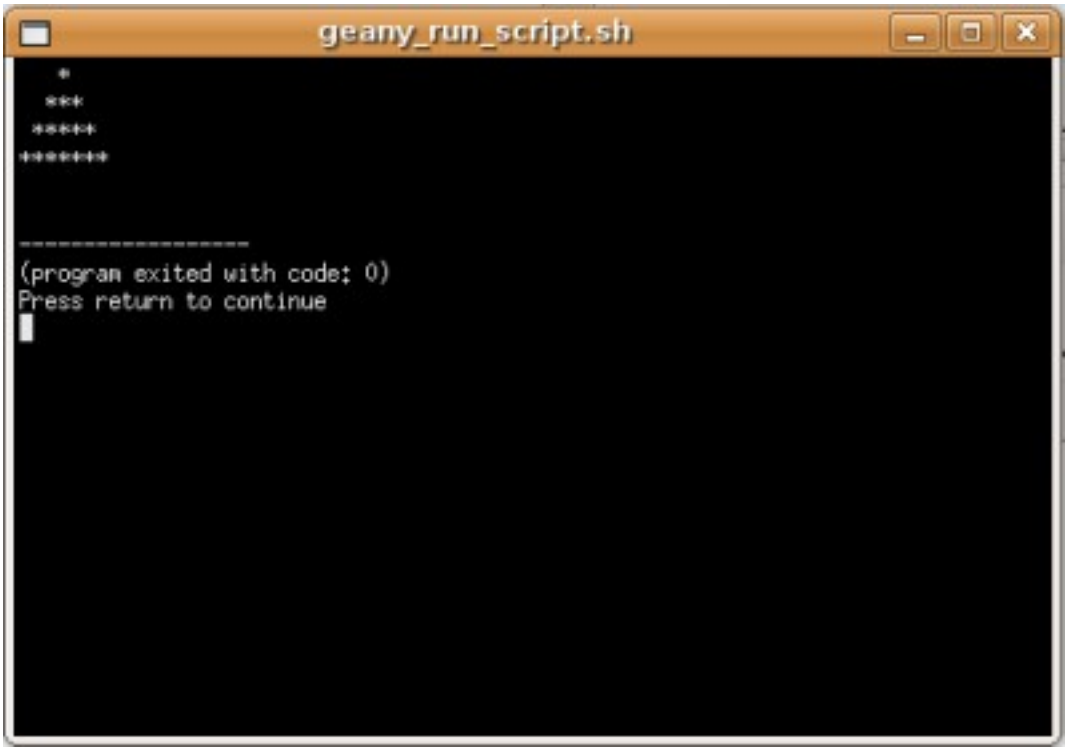


*Рисунок 1.7: Окно терминала с результатами работы программы*

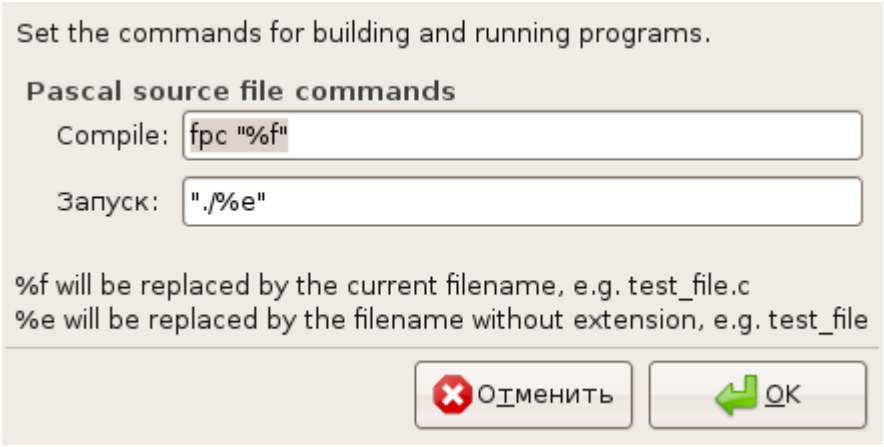


*Рисунок 1.8: Окно **Установить включения и аргументы** для Free Pascal*

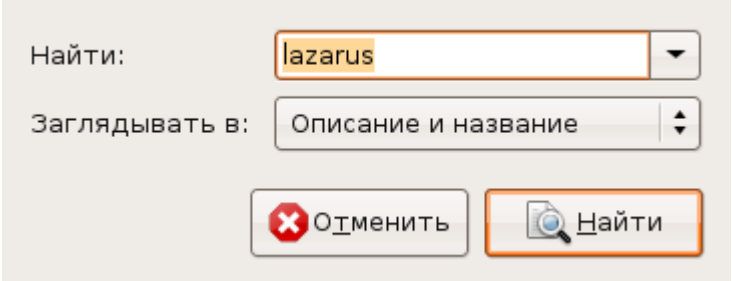


*Рисунок 1.9: Окно поиска пакета Lazarus для установки*

1. Запускаем Lazarus с английским интерфейсом с правами администратора[9].

2. Выполняем команду **Tools — Conigure «Build — Lazarus»...** . В открывшемся окне **Conigure «Build — Lazarus»** на вкладке Quick Build Options устанавливаем следующие параметры компиляции **Build Options — Build All**, **LCL interface — IDE — gtk2 (betta)**, после чего щелкаем по кнопке Build и ждем, пока произойдет перекомпиляция среды Lazarus с новыми параметрами.

3. После этого запускаем Lazarus из меню (или из терминала командой startlazarus) и видим корректную русскоязычную среду Lazarus.

Конечно, для установки Lazarus можно не использовать менеджер пакетов Synaptic, а самостоятельно скачать все необходимые пакеты с сайта http://sourceforge.net/project/showfiles.php?group_id=89339 и затем их вручную установить. Подробно процесс ручной установки можно найти в Интернете, например на странице http://freepascal.ru/article//lazarus/20080316091540.

### 1.4.2 Установка Lazarus под управлением ОС Windows

Рассмотрим особенности установки среды визуального программирования Lazarus для операционной системы Windows. Перед установкой необходимо скачать установочный файл со страницы загрузки http://sourceforge.net/project/showfiles.php?group_id=89339. Установка Lazarus на компьютер осуществляется с помощью ряда окон Мастера установки. Для того чтобы Мастер установки начал свою работу, необходимо запустить программу установки Lazarus Setup. Появится диалоговое окно (рис. 1.10), в котором пользователь может выбрать из списка язык для дальнейшего диалога с Мастером установки. Нажатие кнопки **ОК** приведет к появлению следующего окна[10].

Следующее окно (рис. 1.11) информационное. Оно сообщает пользователю о начале процесса установки. Здесь кнопка **Далее** приведет к следующему шагу Мастера установки, кнопка **Отмена** прервет процесс установки приложения.

---

9 В Ubuntu команда будет такой `sudo LANG=C startlazarus`

10 В этом и во всех последующих окнах использование кнопки **Отмена** приведет к прерыванию работы Мастера установки.

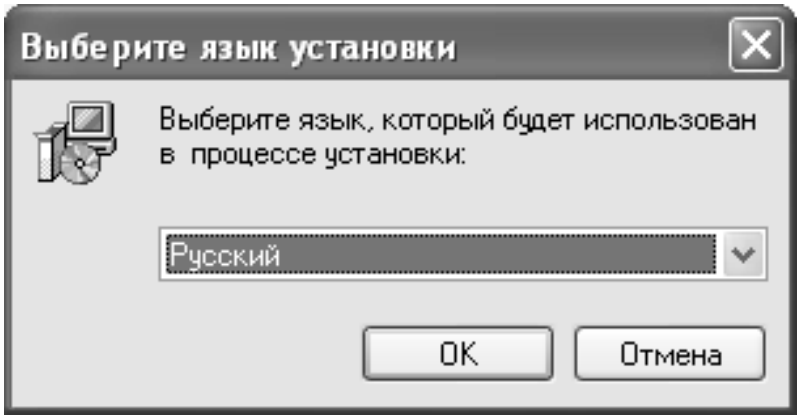


*Рисунок 1.10: Окно Мастера установки. Выбор языка.*

На следующем этапе (рис. 1.12) необходимо выбрать путь для установки Lazarus. По умолчанию программа будет установлена на диск C. Для выбора иного пути для установки следует воспользоваться кнопкой **Обзор**. Щелчок по кнопке **Далее** приведет к следующему шагу процесса установки[11].

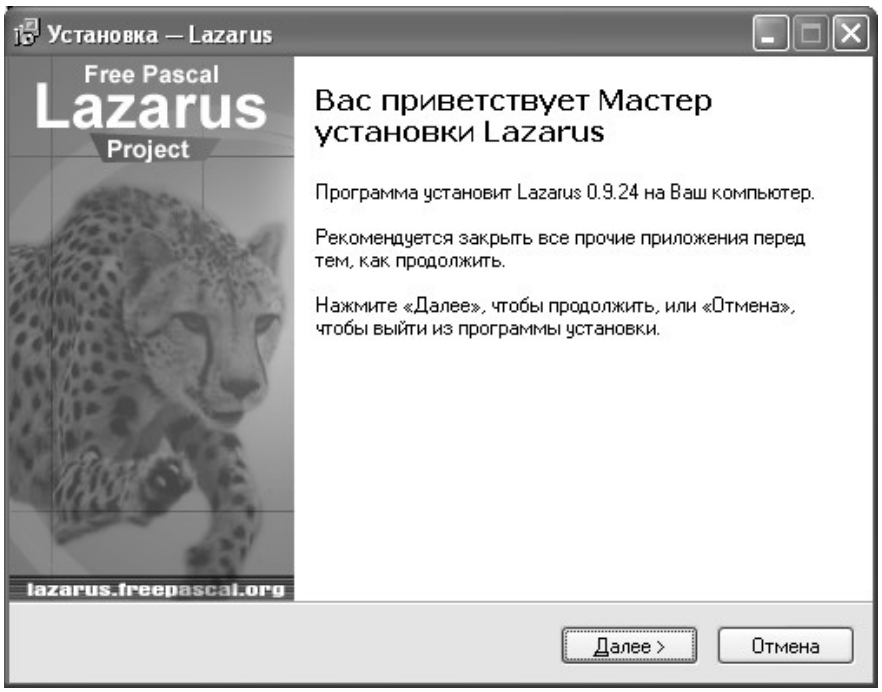


*Рисунок 1.11: Начало процесса установки Lazarus*

11 В этом и во всех последующих окнах с помощью кнопки **Назад** можно вернуться к предыдущему шагу Мастера установки.

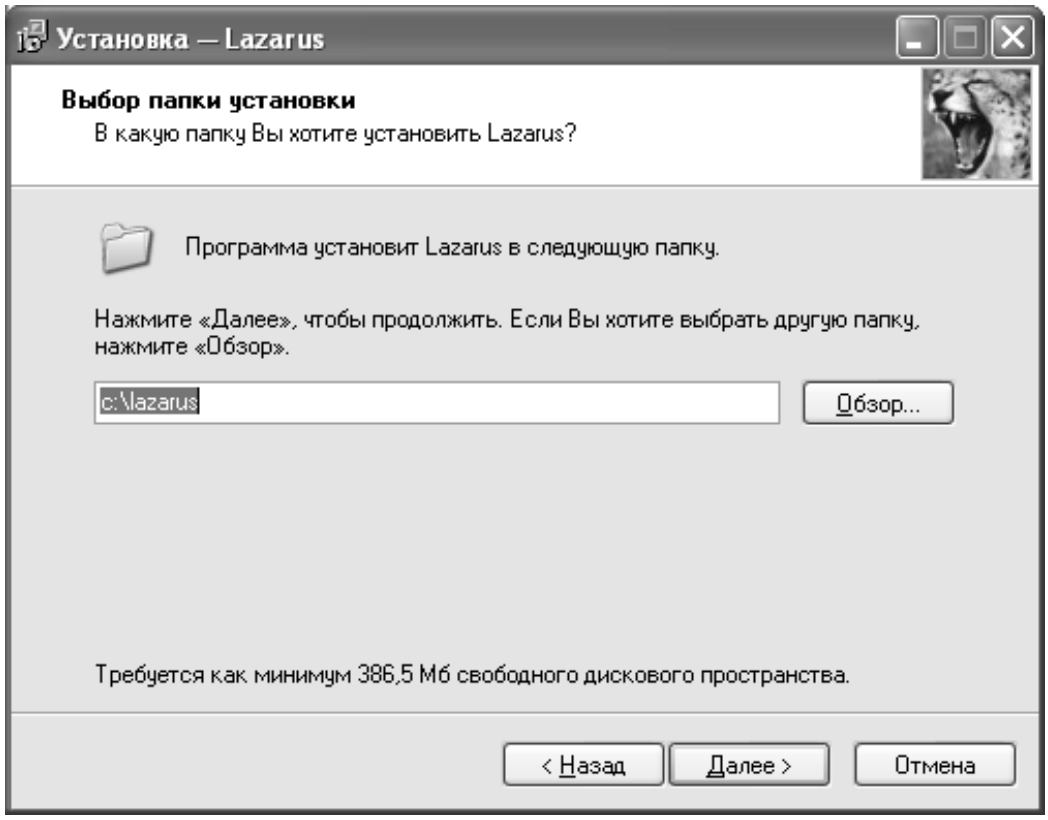


*Рисунок 1.12: Окно выбора пути для установки Lazarus*

Следующий шаг – это выбор необходимых компонентов из полного перечня средств системы (рис. 1.13). По умолчанию будут установлены все компоненты системы. Отменить установку компонента можно, если щелчком мыши убрать метку рядом с его именем. Кнопка **Далее** продолжает процесс установки. В следующих окнах Мастера установки пользователь сможет выбрать папку меню **Пуск**, в которой будет создан ярлык для устанавливаемого приложения[12], и создать на рабочем столе ярлык для приложения Lazarus[13].

Далее Мастер установки сообщит о том, куда будет установлено приложение Lazarus, каков тип установки, какие компоненты системы были выбраны, где будет создан ярлык и будет ли создан значок на рабочем столе. После этого начнется процесс установки приложения на компьютер. Контролировать инсталляцию приложения пользователь может с помощью линейного индикатора, а прервать – кнопкой **Отмена**. Завершается процесс установки щелчком по кнопке **Завершить**. Запуск Lazarus можно осуществить из главного меню командой **Пуск — Программы — Lazarus — Lazarus.**

12 По умолчанию ярлык создается в меню **Пуск — Программы.**

13 С помощью щелчка мыши установить метку в поле рядом с командой **Создать значок на рабочем столе**.

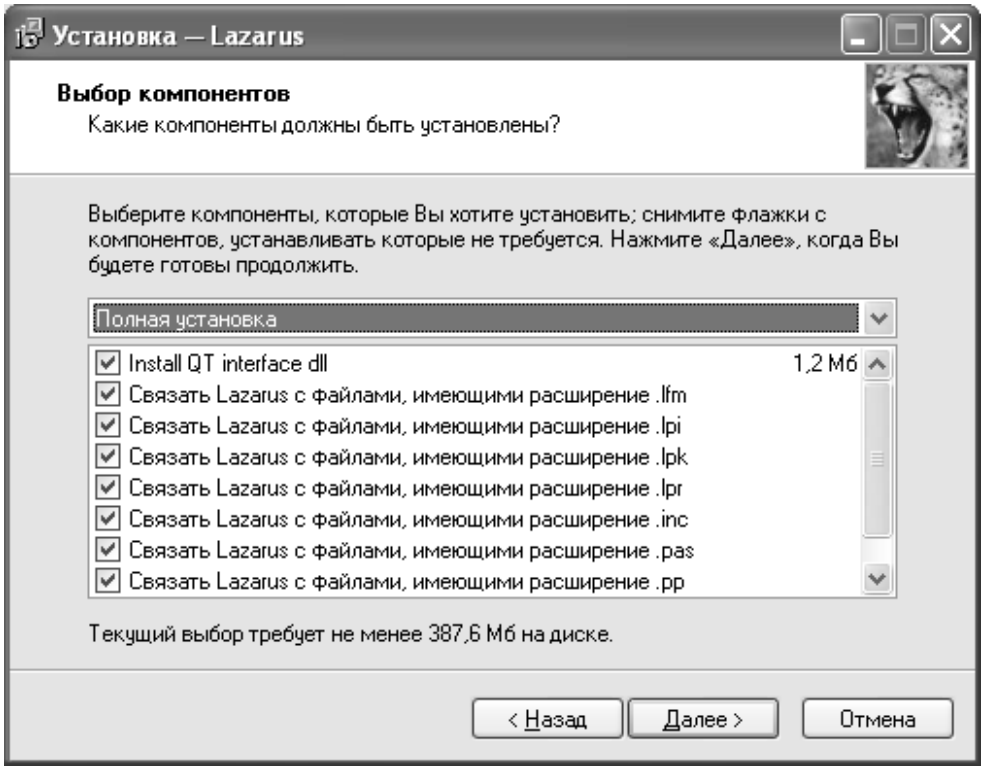


*Рисунок 1.13: Окно выбора компонентов для установки приложения*

### 1.4.3 Среда Lazarus

На рис. 1.15 показано окно, которое появится после запуска Lazarus. В верхней части этого окна размещается *главное меню* и *панель инструментов*. Слева расположено окно *инспектора объектов*, а справа окно *редактора исходного кода*. Если свернуть или сдвинуть окно редактора, то станет доступным *окно формы*, представленное на рис. 1.14.

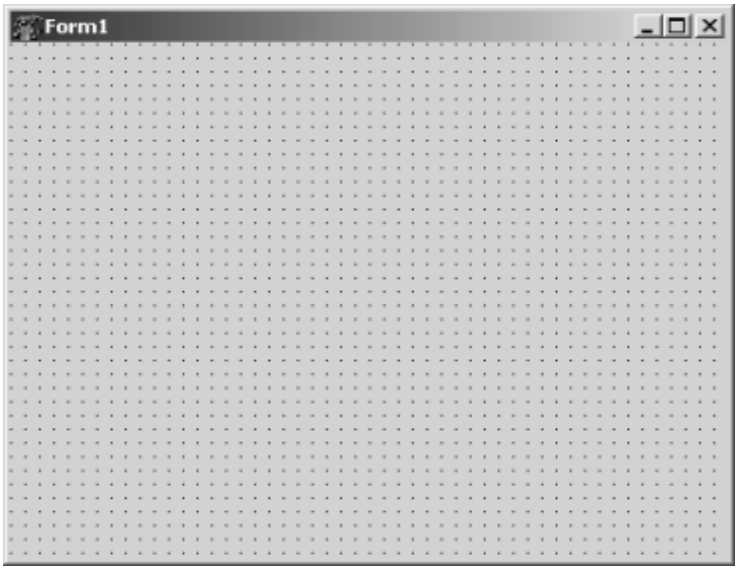


*Рисунок 1.14: Окно формы*

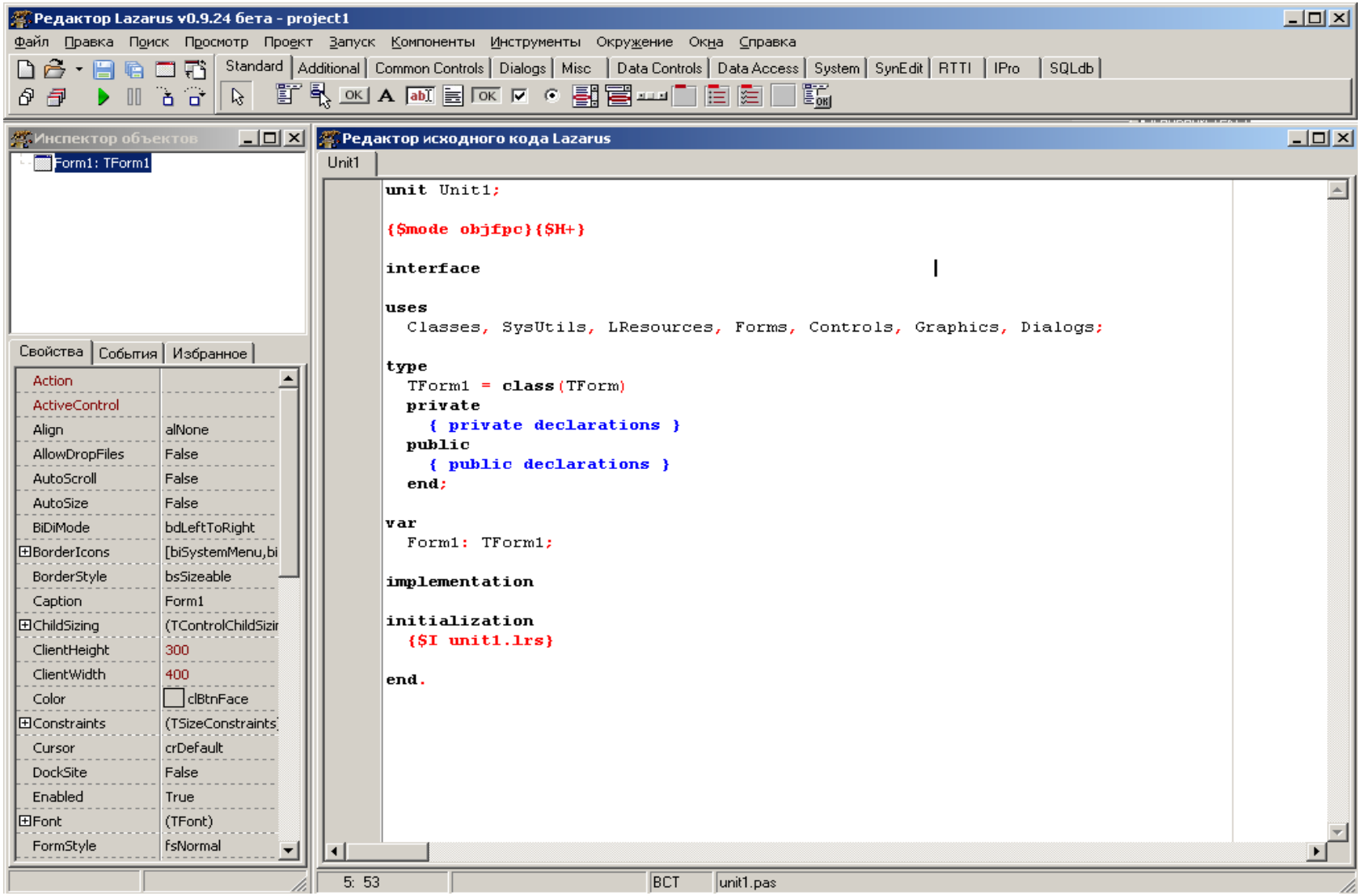


*Рисунок 1.15: Среда визуального программирования Lazarus*

Работу над программой в среде визуального программирования условно можно разбить на две части. Первая это создание внешнего вида (интерфейса) будущей программы, вторая — написание программного кода. Итак, *инспектора объектов* и *окно формы* нужны для создания интерфейса программы, а *редактор исходного кода* — для работы с ее текстом. Файлы, из которых в результате получается программа, называют *проектом*.

### 1.4.4 Главное меню Lazarus

Все команды, необходимые для работы в среде визуального программирования Lazarus, содержатся в *главном меню*. Доступ к командам главного меню осуществляется одинарным щелчком левой кнопкой мыши.

*Работа с файлами* в среде Lazarus осуществляется при помощи пункта меню **Файл**. Команды этого пункта меню можно разбить на группы:

- создание новых файлов – **Создать модуль**, **Создать форму**, **Создать...**;
- загрузка ранее созданных файлов – **Открыть**, **Открыть недавнее**, **Вернуть**;
- сохранение файлов – **Сохранить**, **Сохранить как…**, **Сохранить все**;
- закрытие файлов – **Закрыть**, **Закрыть все файлы редактора**;
- вывод на печать – **Печать…**;
- перезагрузка среды – **Перезапуск**;
- выход из среды – **Выход**.

Команды, предназначенные для *редактирования текста программного кода,* собраны в меню **Правка**. В основном это команды характерные для большинства текстовых редакторов:

- команды отмены или возврата последней операции – **Отменить**, **Вернуть**;
- команды переноса, копирования и вставки выделенного фрагмента текста в буфер – **Вырезать**, **Копировать**, **Вставить**;
- команды, работающие с выделенным блоком текста – **Сдвинуть блок вправо**, **Сдвинуть блок влево**;
- команды смены регистра – **Верхний регистр выделения**, **Нижний регистр выделения**;
- команды выделения фрагмента текста собраны в пункте меню **Выделить**.

Далее будут перечислены специфические команды редактора программного кода Lazarus.

Команда **Закомментировать** добавляет в начале каждой строки выделенного фрагмента два символа «косая черта», что превращает выделенный текст в *комментарий*[14], команда **Раскомментировать** выполняет обратное действие.

Команда **Заключить выделение в...** приводит к открытию диалогового окна (рис. 1.16), в этом окне пользователь может выбрать конструкцию языка программирования, в которую будет заключен выделенный фрагмент текста.

Команда **Сортировка выбранного...** открывает диалоговое окно, в котором можно установить параметры сортировки текста в выделенном фрагменте.

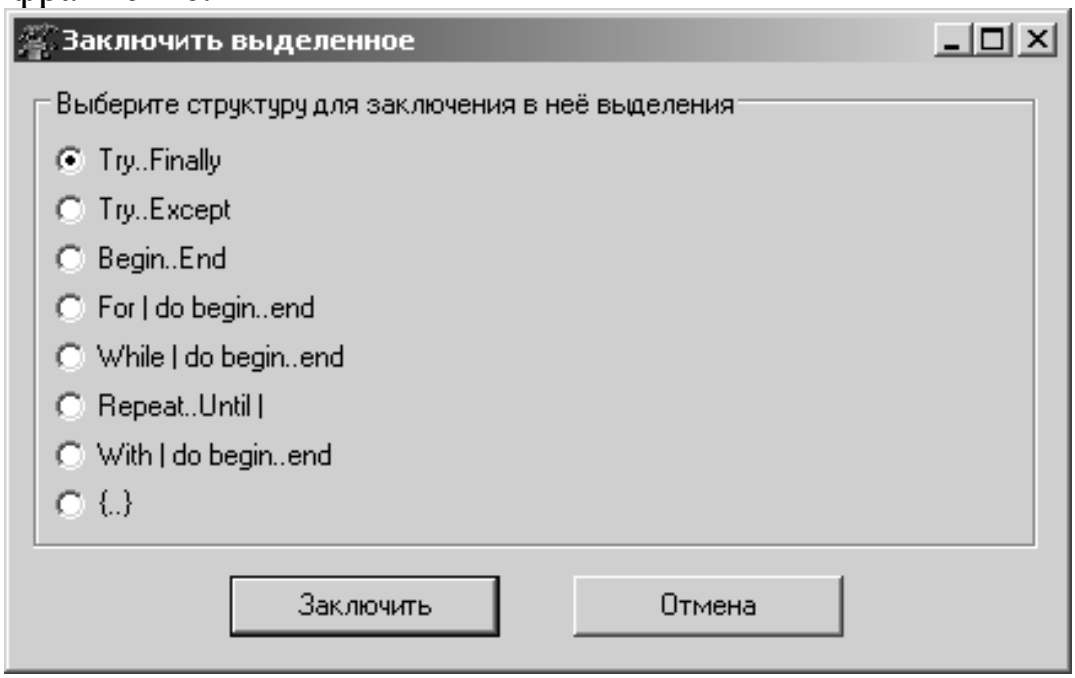


*Рисунок 1.16: Выбор конструкции языка для заключения в нее выделенного фрагмента программного кода*

Команды меню **Поиск** можно разделить на группы. Первая группа — это непосредственно *команды поиска и замены*, вторая - это команды перехода, а третья — работа с закладкой. В четвертой группе объединены команды поиска, замены и перехода в выделенном фрагменте. Большинство из этих команд используются в текстовых редакторах, смысл остальных понятен из названия.

Пункт меню **Просмотр** применяют для *настройки внешнего вида*

14 Комментарий — фрагмент программного кода, который игнорируется компилятором. Обычно комментарии используют для пояснений к программе или для временного исключения фрагментов текста при отладке.

*среды программирования*. Первая группа команд открывает или активизирует следующие окна:

- **Инспектор объектов** — окно, с помощью которого можно описать внешний вид и поведение выделенного объекта (подробно см. п. 1.4.8);
- **Редактор исходного кода** — окно, в котором можно создавать и редактировать текст программы (подробно см. п. Ошибка: источник перекрестной ссылки не найден);
- **Обозреватель кода —** содержит общую информацию о проекте;
- **Редактор LazDoc —** редактор шаблонов;
- **Браузер кода** — окно проводника проекта.

Следующая группа команд пункта меню **Просмотр** тоже открывает диалоговые окна. Эти окна носят информационный характер. Так команды **Модуль...** и **Форма...** выводя список модулей и форм данного проекта. Назначение команд **Показать зависимости модулей** и **Показать сведения о модуле** говорят сами за себя. Последняя команда этой группы **Переключить модуль/форму** активизирует либо окно редактора, либо форму.

В последней группе команд следует отметить команды **Показать палитру компонентов** и **Показать кнопки быстрого доступа**. Устанавливая метки рядом с этими командами, пользователь выводит на экран или наоборот убирает *панели инструментов*. Командой **Окна отладки** пользуются во время отладки программного кода. Здесь можно вызывать **Окно наблюдений**, **Окно отладчика**, просматривать точки останова и значения переменных в процессе выполнения программы.

Команды пункта меню **Проект** предназначены для выполнения различных *операций с проектом*:

- команды создания проекта — **Создать проект** и **Создать проект из файла**;
- команды вызова ранее созданного проекта — **Открыть проект**, **Открыть недавний проект**, **Закрыть проект**;
- команды сохранения проекта — **Сохранить проект**, **Сохранить проект как...**, **Опубликовать проект**;
- команды управления проектом — **Инспектор проекта**, **Настройка проекта...**, **Параметры компилятора** и т.д.

Команды, позволяющие *запускать проект* на выполнение и *выполнять его отладку*, содержатся в пункте главного меню **Запуск**:

- **Собрать** — сборка программы из откомпилированных файлов;
- **Собрать все** — скомпоновать все файлы проекта;
- **Быстрая компиляция** — компиляция файлов программы;
- **Запуск** — запуск проекта с помощью отладчика (компиляция, компоновка и выполнение);
- **Пауза** — приостановка выполнения программы до нажатия любой клавиши;
- **Шаг с входом** — режим пошагового отладочного выполнения программы с входом в вызываемые процедуры и функции;
- **Шаг в обход** — режим пошагового отладочного выполнения программы без входа в вызываемые процедуры и функции;
- **Запуск до курсора** - отладка и выполнение программы в этом режиме осуществляются до оператора, стоящего в строке помеченной курсором;
- **Останов** — прерывание выполнения программы;
- **Сброс отладчика** - сброс всех ранее задействованных отладочных средств и прекращение выполнения программы;
- **Настройка сборки+запуска...** - настройка параметров компоновки и запуска;
- **Вычислить\Изменить** - возможность просмотреть значение переменной и/или найти значение выражения в процессе выполнения программы, при необходимости можно изменить значения любой переменной;
- **Добавить наблюдения** — в открывающемся окне можно указать переменные и/или выражения, за изменением значений которых следует понаблюдать при отладке программы;
- **Добавить точку останова** - установка в текущей строке контрольной точки; после запуска программного кода отладчик прекратит его выполнение перед оператором, помеченным точкой останова; в программе можно указать произвольное количество таких точек. Точку останова также можно добавить щелчком мыши по номеру строки программного кода.

*Работа с компонентами*[15] организована при помощи команд пункта меню **Компоненты**. Пункты меню **Инструменты** и **Окружение** применяют для настройки свойств среды программирования.

15 Компонент – это готовый программный код, который можно использовать при написании программы. Пункт меню **Компоненты** предназначен для расширения стандартного набора за счет добавления компонентов других разработчиков.

*Работа с окнами* в Lazarus выполняется при помощи пункта меню **Окна**. Названия команд этого пункта совпадают с названиями окон, и выбор любой команды приведет к активации соответствующего окна.

Пункт меню **Справка** это справочная информация о системе. Здесь можно вызывать средства справочной системы и выполнять ее настройку.

### 1.4.5 Окно формы

*Окно формы* (рис. 1.14) — это проект интерфейса будущего программного продукта.

Вначале это окно содержит только стандартные элементы – строку заголовка и кнопки развертывания, свертывания и закрытия. Рабочая область окна заполнена точками координатной сетки[16].

Задача программиста – используя *панель компонентов*, заполнить форму различными интерфейсными элементами, создавая тем самым внешний вид своей программы.

Команды настройки окна формы находятся на вкладке **Редактор форм** (рис. 1.17) в меню **Окружение — Настройки окружения...**

### 1.4.6 Окно редактора Lazarus

*Окно редактора* (рис. 1.15) тесно связано с окном формы (рис. 1.18) и появляется вместе с ним при создании нового проекта.

Окно редактора по умолчанию находится на первом плане и закрывает окно формы. Переключаться между этими окнами можно командой **Просмотр - Переключить модуль/форму**, клавишей **F12** или просто щелчком мыши.

Окно редактора предназначено для создания и редактирования текста программы, который создается по определенным правилам и описывает некий алгоритм. Если окно формы определяет внешний вид будущей программы, то программный код, записанный в окне редактора, отвечает за ее поведение.

Вначале окно редактора содержит текст, обеспечивающий работу пустой формы. Этот программный код появляется в окне редактора автоматически, а программист в ходе работы над проектом вносит в него дополнения, соответствующие функциям программы.

16 Координатная сетка отображается только на этапе создания программы.

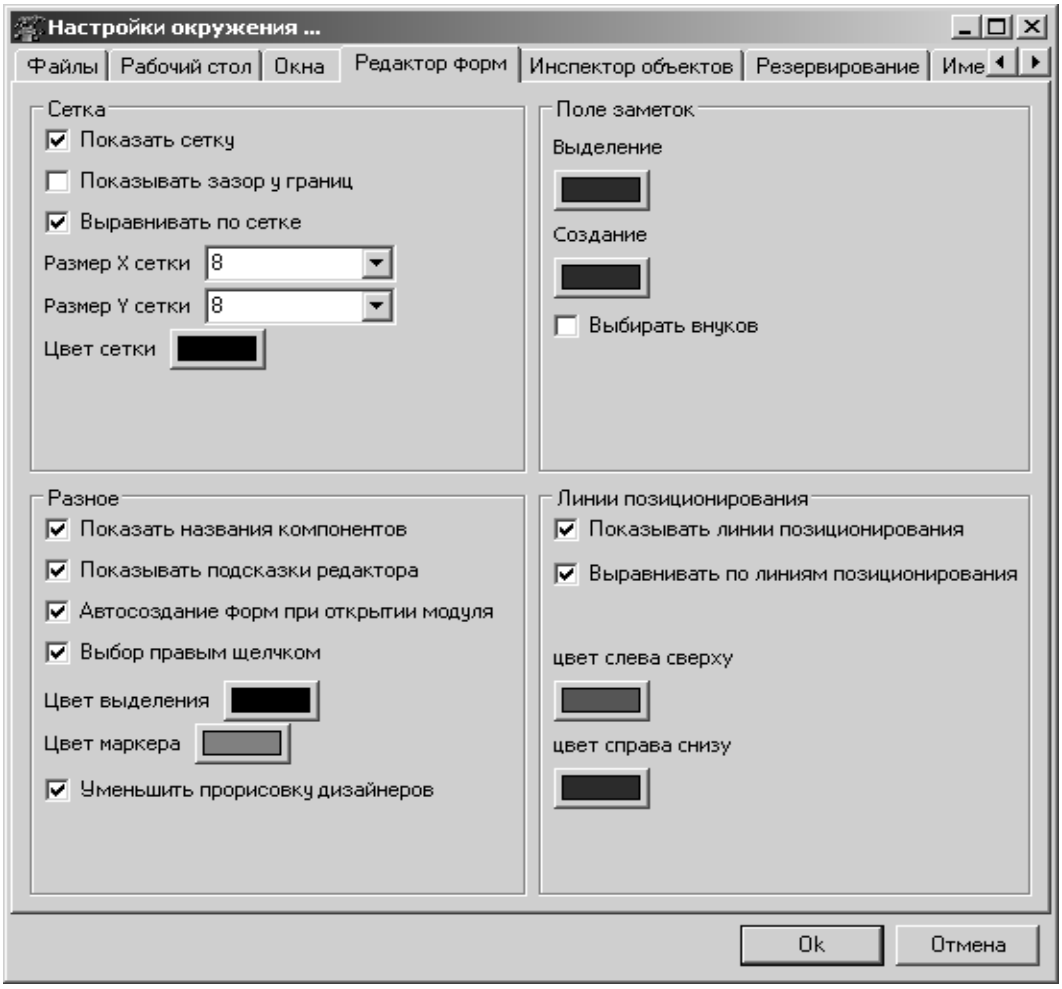


*Рисунок 1.17: Настройка окна формы*

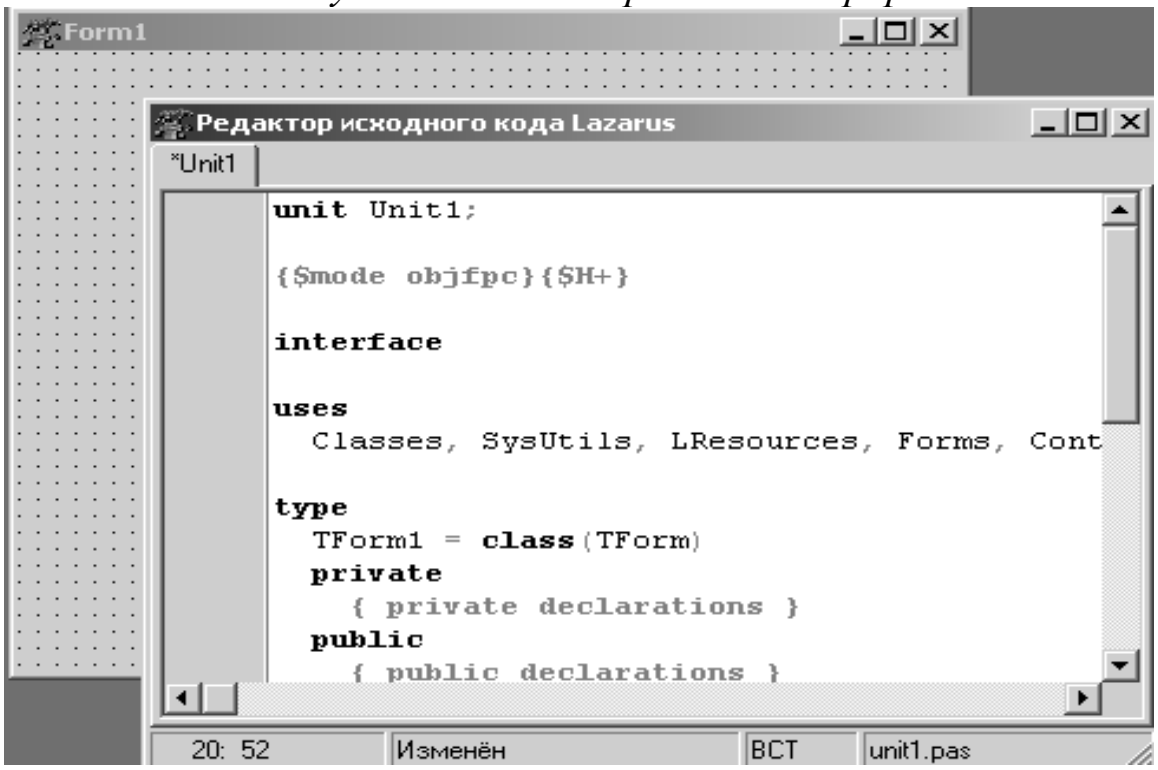


*Рисунок 1.18: Окно формы и окно редактора*

Обратите внимание, что при загрузке Lazarus автоматически загружается *последний проект*, с которым работал пользователь. Происходит это благодаря установке **Открывать последний проект при запуске** (рис. 1.19, 1.20), которая находится на вкладке **Файлы** меню **Окружение — Настройки окружения...**. Если убрать метку рядом с командой **Открывать последний проект при запуске**, то при загрузке Lazarus будет создавать *новый проект*.

*Настроить окно редактора* можно с помощью вкладки **Общие** меню **Окружение — Настройки редактора...**(рис. 1.21). Для того чтобы установить (отменить) ту или иную настройку, достаточно установить (убрать) маркер рядом с ее названием и подтвердить изменения нажатием кнопки **Ok**.

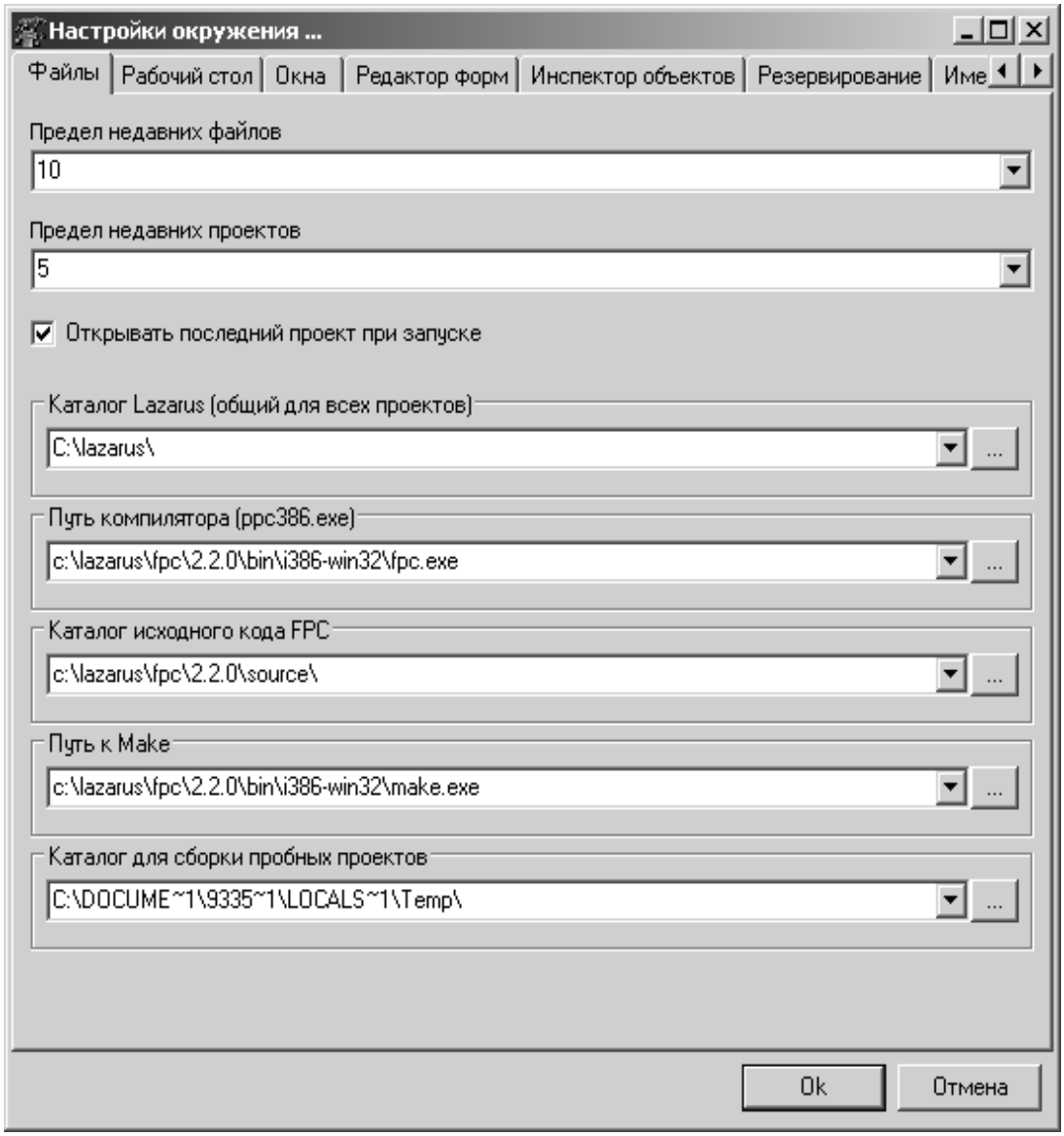


*Рисунок 1.19: Окно настройки файлов в среде Windows*

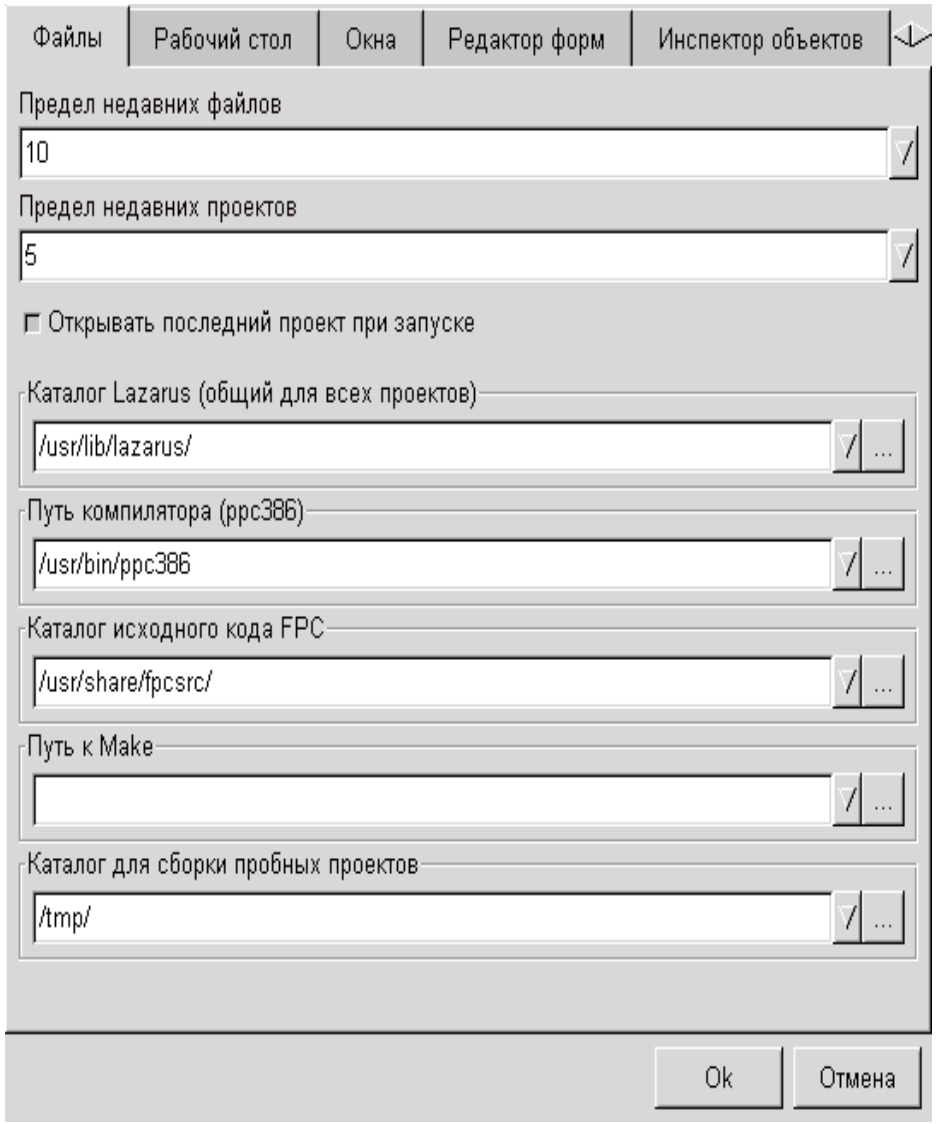


*Рисунок 1.20: Окно настройки файлов в Linux*

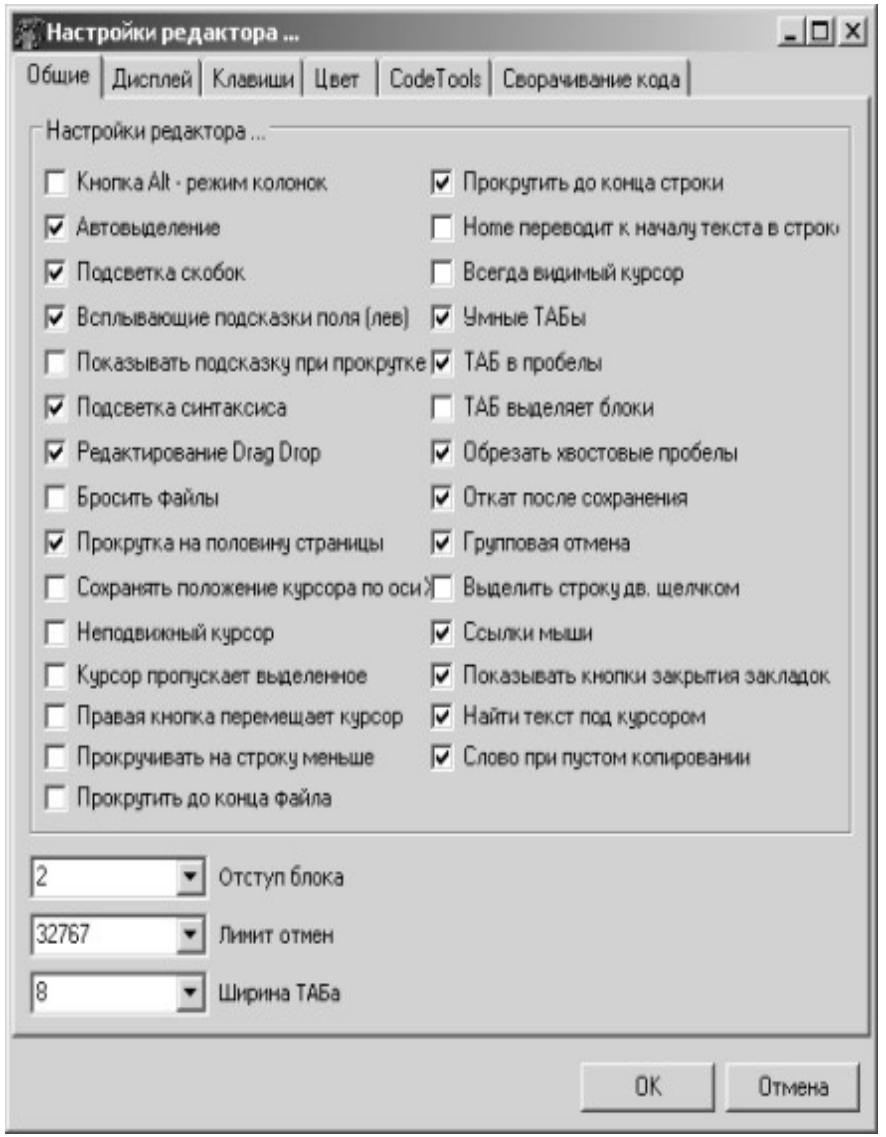


*Рисунок 1.21: Настройки окна редактора*

На вкладке **Дисплей** меню **Окружение — Настройки редактора...**(рис. 1.22) можно *изменить шрифт* текста программного кода.

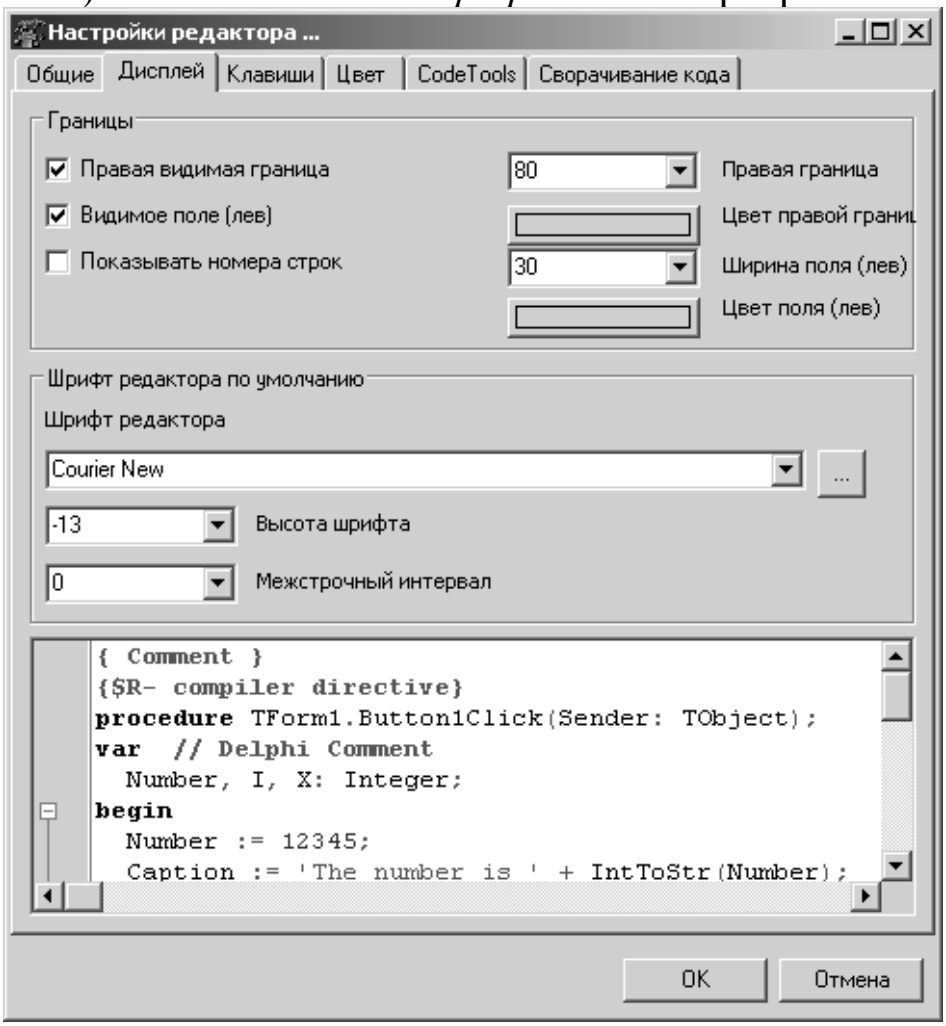


*Рисунок 1.22: Окно настройки редактора программного кода, вкладка Дисплей*

Для корректного отображения символов кириллицы в окне редактора кода под управлением Ubuntu Linux необходимо в окне настроек редактора в поле **Шрифт редактора** установить тип шрифта — **Fixed Misc**, после чего выбрать кодировку **ISO 10646-1** (рис. 1.23).

Как и в большинстве текстовых редакторов, в редакторе программного кода Lazarus предусмотрена *работа с шаблонами*. Здесь под шаблоном подразумевается автоматический ввод конструкций языка программирования.

Например, пользователь вводит символ `b`, затем символ пробел, а на экране появляется конструкция `begin ... end`. Настройку шаблонов можно выполнить в диалоговом окне (рис. 1.24, 1.25), которое появится после выполнения команды **Окружение — Шаблоны кода...**. Например, чтобы заработал рассмотренный шаблон, нужно

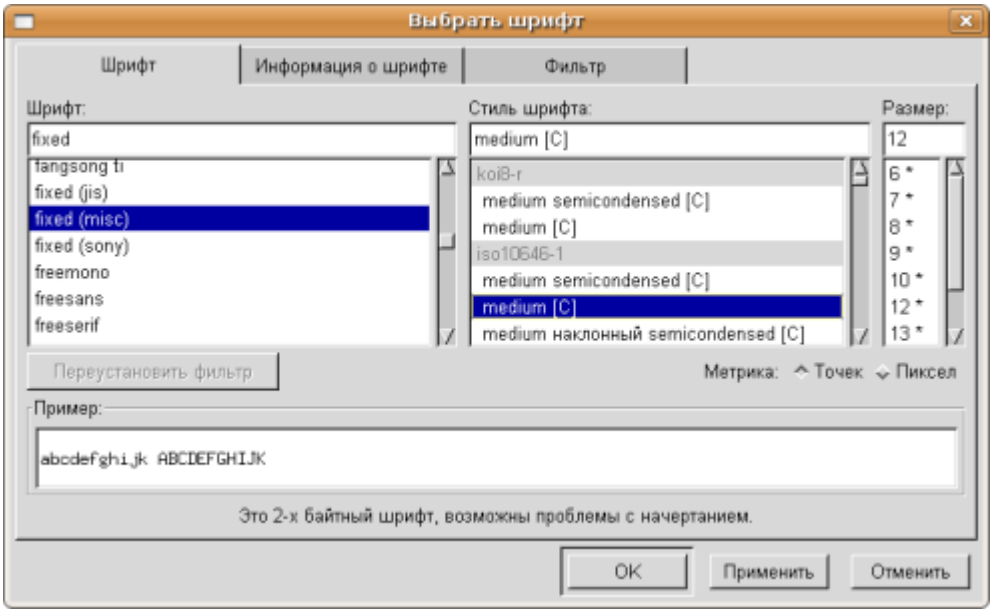


*Рисунок 1.23: Настройка кириллицы в ОС Ubuntu*

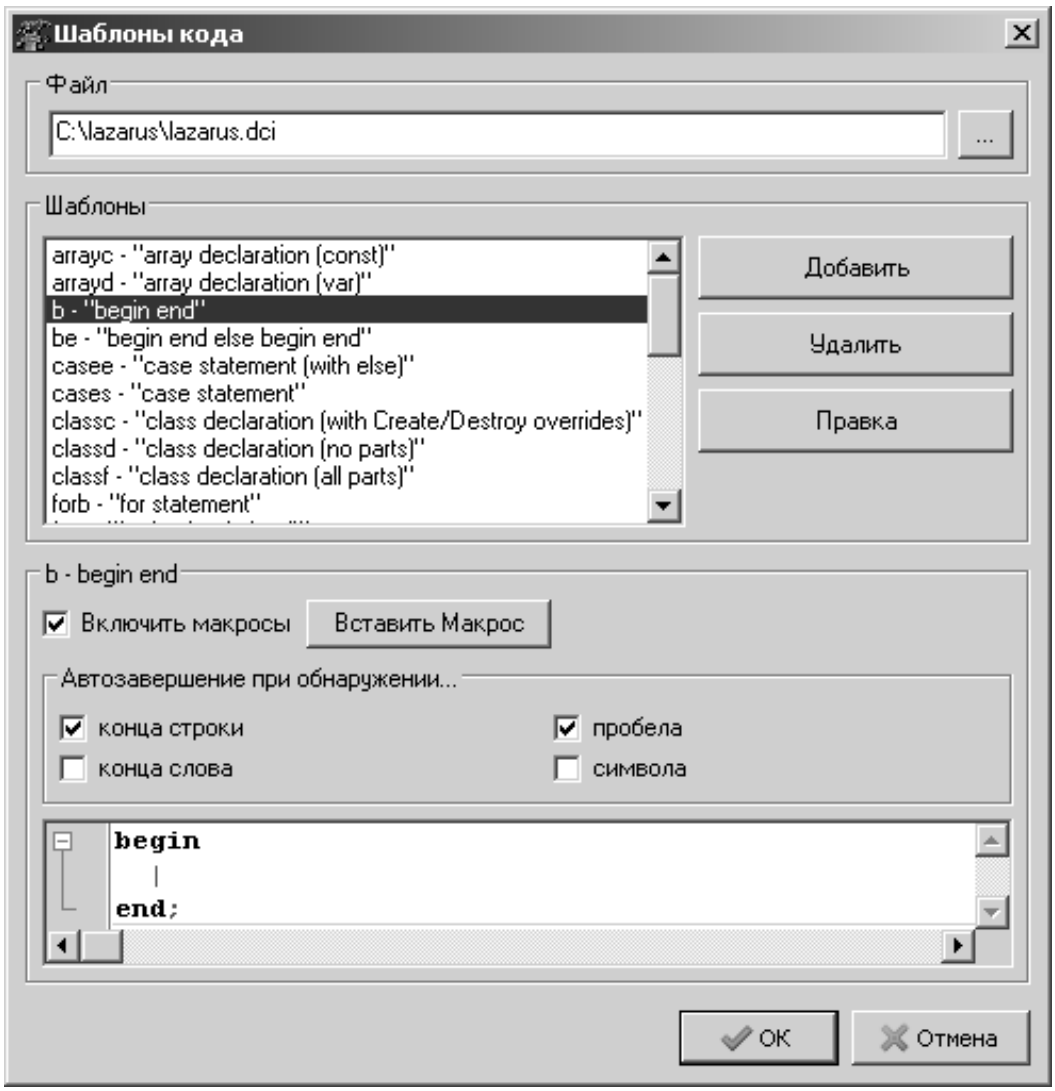


*Рисунок 1.24: Окно настройки шаблонов редактора программного кода в среде Windows*

выбрать (щелкнуть по нему мышкой) в списке шаблонов строку `b - begin ... end` и установить маркер рядом с командой **Включить макросы**. Затем в группе команд **Автозавершение при обнаружении...** включить флажок возле слова **пробела**. Если среди команд автозавершения выбрать **конец строки**, то шаблон будет появляться в тексте программы после ввода символа `b` и нажатия клавиши **Enter**.

Шаблоны можно добавлять, удалять и править. Для этого в окне **Шаблоны кода** (рис. 1.25) предусмотрены специальные кнопки. В нижней части окна, над кнопками **Ok** и **Отмена**, расположено поле ввода и редактирования шаблонов с активным курсором.

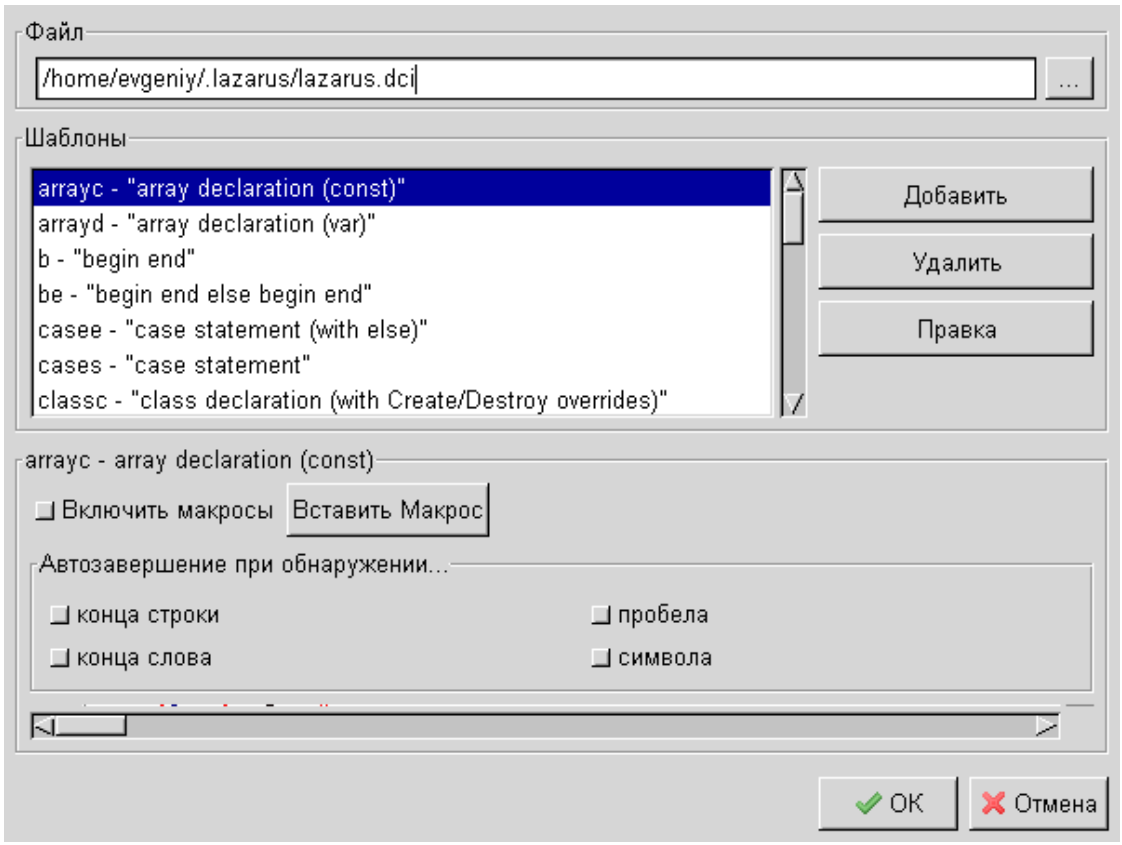


*Рисунок 1.25: Окно настройки шаблонов редактора программного кода в среде Linux*

Если выделить любой шаблон из списка, щелкнув по нему мышкой, то в поле ввода появится соответствующая конструкция. Если воспользоваться кнопкой **Добавить**, то появится диалоговое окно (рис. 1.26) для *создания шаблона*. Здесь в поле **Элемент** следует указать символ или группу символов, ввод которых будет связан с создаваемым шаблоном.

В поле **Комментарий** можно дать краткое описание шаблона и нажать кнопку **Добавить**. Элемент и комментарий к нему будут добавлены в список шаблонов в окне **Шаблоны кода**. Теперь нужно создать сам шаблон, то есть указать, какая именно конструкция языка появится при вводе элемента в текст программы.

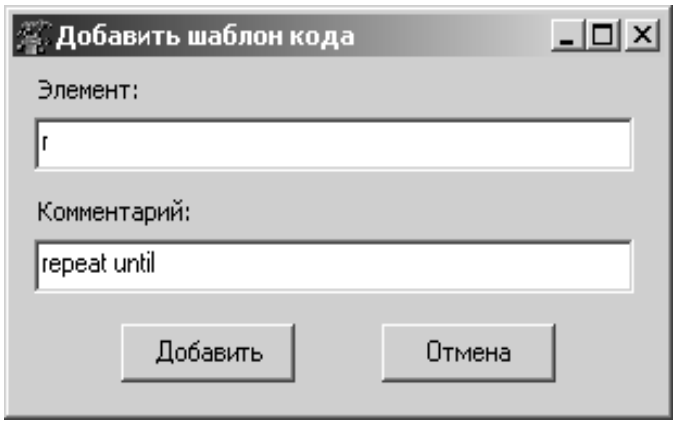


*Рисунок 1.26: Окно создания шаблона*

Для этого надо выделить вновь созданный шаблон и в поле ввода, расположенном в нижней части окна **Шаблоны кода,** ввести соответствующий текст. Например, создадим шаблон для конструкции `repeat ... until`, которая будет добавлена в текст, если пользователь введет символ `r`. Выполним следующие действия:

- щелкнем по кнопке **Добавить** в окне **Шаблоны кода**;
- в диалоговом окне **Добавить шаблон кода** (рис. 1.26) в поле **Элемент** введем символ `r`, а в поле **Комментарий** фразу `repeat until` и нажмем кнопку **Добавить**;
- в окне **Шаблоны кода** (рис. 1.27, 1.28) выделим в списке шаблонов строку `r - repeat until`;
- введем в поле ввода, расположенном в нижней части окна, конструкцию языка `repeat until;`;
- нажмем кнопку **Ok** в окне **Шаблоны кода**.

Для *изменения шаблона* служит кнопка **Правка**. Шаблон, который нужно откорректировать, должен быть выделен. Дальнейшие действия аналогичны тем, которые выполняются при создании шаблона.

Для того чтобы *удалить шаблон* из списка, его нужно выделить, а затем нажать кнопку **Удалить**.

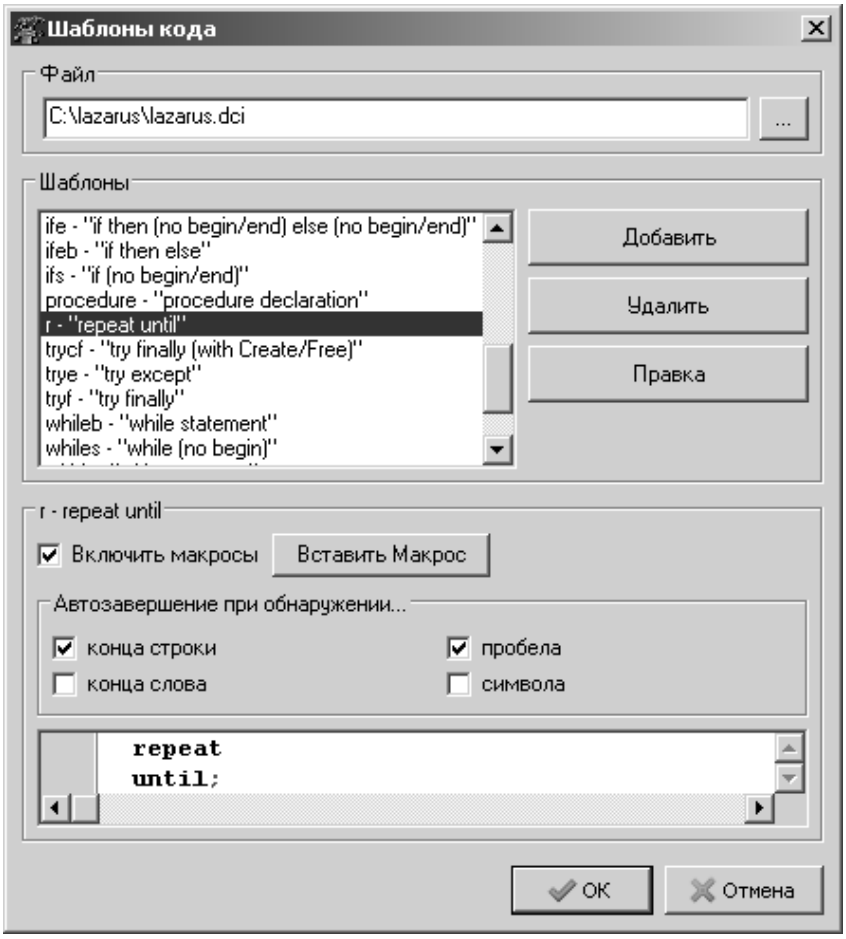


*Рисунок 1.27: Создание шаблона в Windows*

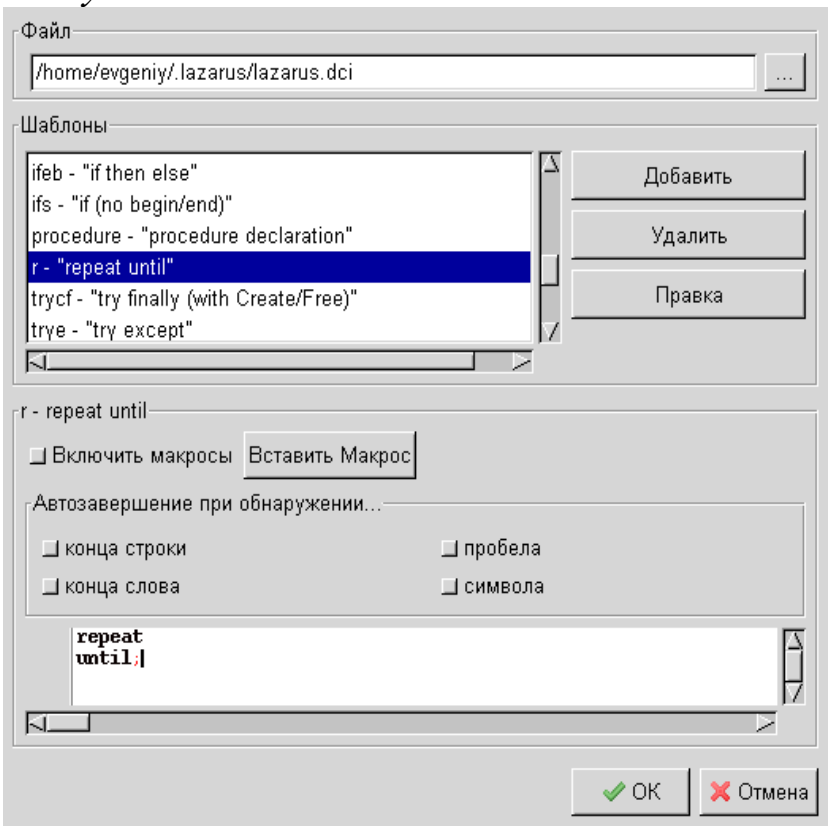


*Рисунок 1.28: Создание шаблона в Linux*

### 1.4.7 Панель компонентов

*Панель компонентов* расположена[17] под главным меню (рис. 1.15). Она состоит из большого числа групп, в которых располагаются соответствующие компоненты (рис. 1.29).


*Рисунок 1.29: Панель компонентов*

*Компонент* – это некий функциональный элемент интерфейса, обладающий определенными свойствами. Размещая компоненты на форме, программист создает внешний вид своей будущей программы – окна, кнопки, переключатели, поля ввода и т.п.

Для *внедрения нового компонента* на форму нужно сделать два щелчка мышкой:

- в панели компонентов, для выбора компонента;
- в рабочем пространстве формы, для указания положения левого верхнего угла компонента.

Компоненты объединяются в группы по функциональному признаку. После создания проекта по умолчанию открывается список группы **Standard**, содержащий основные элементы диалоговых окон. Просмотреть другие группы можно, раскрывая их щелчком по соответствующей вкладке.

### 1.4.8 Инспектор объектов

Окно *инспектора объектов* располагается слева от окна редактирования. Как правило, оно содержит информацию о выделенном объекте. На рис. 1.30 представлен инспектор объектов с информацией о вновь созданной форме. Окно инспектора объектов имеет три вкладки: **Свойства**, **События**, **Избранное.** Эти вкладки используются для редактирования свойств объекта и описания событий, на которые будет реагировать данный объект. Совокупность *свойств* отображает внешнюю сторону объекта, совокупность *событий* – его поведение.

Вкладки инспектора объектов представляют собой таблицу. В левой колонке расположены названия свойств или событий, в правой – конкретные значения свойств или имена подпрограмм, обрабатывающих события. Чтобы выбрать свойство или событие, необходимо щелкнуть левой кнопкой мыши по соответствующей строке.

17 Включить (выключить) **Панель компонентов** можно, установив (убрав) маркер рядом с командой **Показать палитру компонентов** меню **Просмотр**.

Свойства, отображенные в таблице, могут быть простыми или сложными. Простые свойства определены единственным значением.

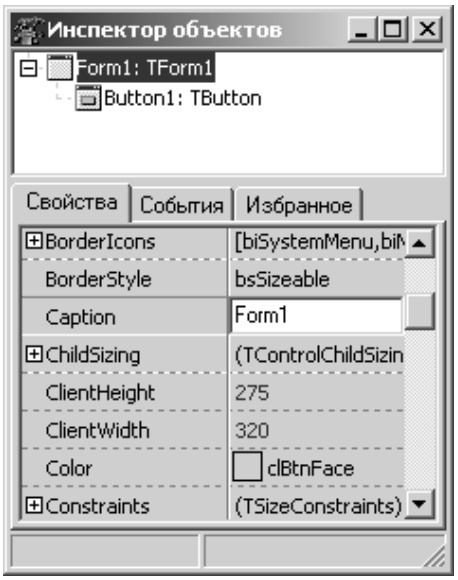


*Рисунок 1.30: Окно инспектора объектов*

Например, свойство `Caption` (Заголовок) определяется строкой символов, свойства `Height` (Высота) и `Width` (Ширина) – числом, свойство `Enabled` (Доступность) – значениями `True` (Истина) и `False` (Ложь). Сложные свойства определяются совокупностью значений. Например, свойство `Font` (Шрифт). Слева от имени этого свойства стоит знак «+». Это означает, что свойство сложное, и при щелчке по значку «+» откроется список его составляющих.

Активизировать значение любого свойства можно обычным щелчком мыши. При этом в конце строки может появиться либо кнопка с символом `троеточия`, либо кнопка со стрелкой, направленной вниз. Щелчок по `троеточию` откроет диалоговое окно для установки значений сложных свойств (например, `Font`). Обращение к стрелке приведет к раскрытию списка возможных значений простого свойства (например, `Enabled`).

### 1.4.9 Первая программа в Lazarus

Процесс создания программы в Lazarus состоит из двух этапов: формирование внешнего вида программы, ее интерфейса и написание программного кода на языке программирования Free Pascal, заставляющего работать элементы интерфейса.

Как мы уже выяснили, для создания интерфейса программы существует *окно формы*, а для написания программного кода – *окно редактора*. Эти окна тесно связаны между собой, и размещение *компонентов* на форме приводит к автоматическому изменению программного кода.

Начнем знакомство с визуальным программированием с создания простой программы про кнопку, которая хочет, чтобы по ней щелкнули. После чего появляется сообщение о работе программы.

Начнем с *создания нового проекта*. Для этого выполним команду

главного меню **Проект — Создать проект...**. В появившемся диалоговом окне (рис. 1.31, 1.32) выберем из списка слово **Приложение** и нажмем кнопку **Создать**. Результатом этих действий будет появление *окна формы* и *окна редактора программного кода*.

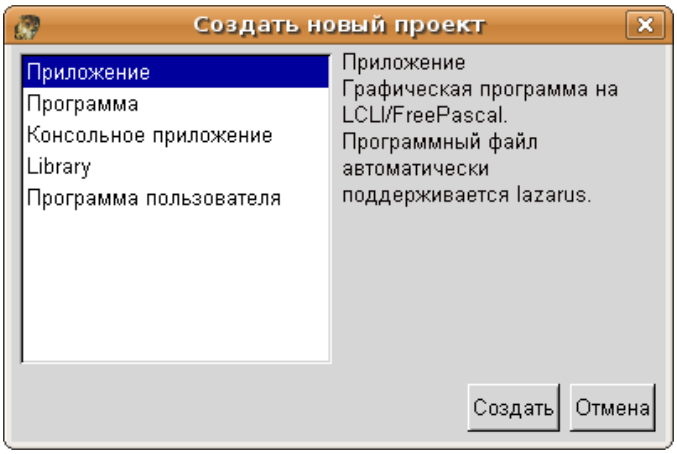


*Рисунок 1.31: Создание нового проекта в Linux*

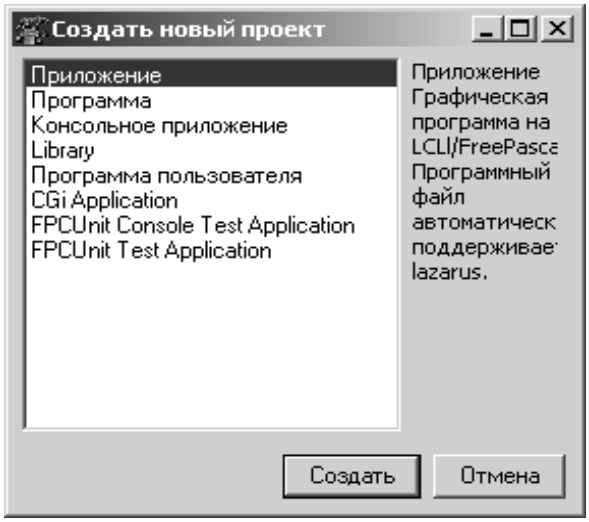


*Рисунок 1.32: Создание нового проекта в среде Windows*

*Сохраним созданный проект*, воспользовавшись командой **Проект — Сохранить проект как…**. Откроется окно сохранения программного кода **Сохранить Unit1** (рис. 1.33, 1.34). Создадим в нем новую папку **Primer_1** (можно воспользоваться кнопкой **Создание новой папки (Создание каталога)**), откроем ее и щелкнем по кнопке **Сохранить**. Тем самым мы сохраним файл **Unit1.pas**, содержащий текст программы.

Сразу же откроется окно **Сохранить проект** (рис. 1.35, 1.36), в котором также необходимо щелкнуть по кнопке **Сохранить**.

Теперь мы сохранили файл **Project1**[18], содержащий общие сведения о проекте.

На самом деле все наши манипуляции привели к сохранению более чем двух файлов. Теперь в каталоге **Primer_1** (рис. 1.37, 1.38) хранится файл с текстом программы **Unit1.pas**, файл **Unit1.lfm**[19], со сведениями о форме **Form1**, а также файлы **Project1.lpn** и **Project1.lpi,** содержащие настройки системы программирования и параметры конфигурации проекта.

18 Имена **Unit1** и **Project1** могут быть любыми другими.
19 Имена файлов можно задать и отличные от стандартных Unit1.

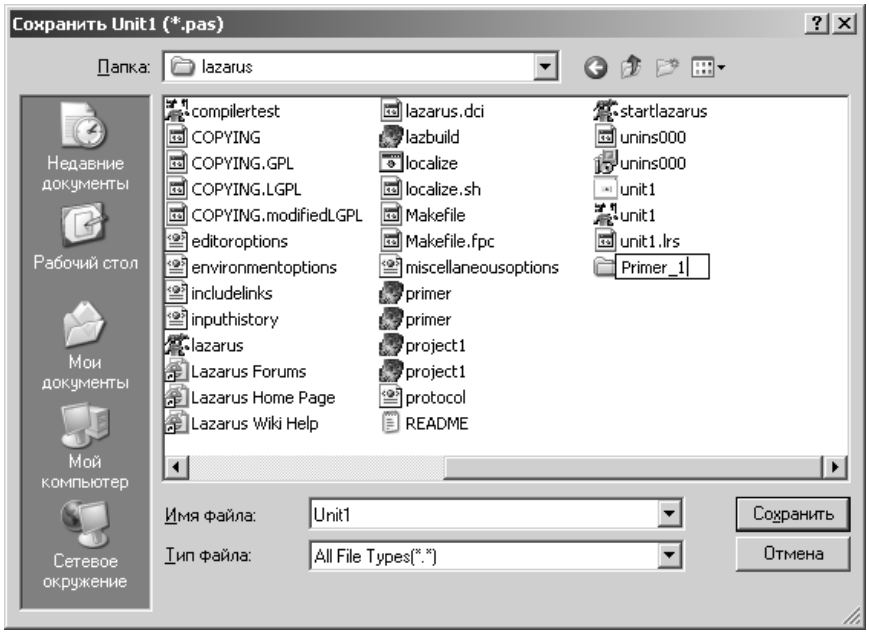


*Рисунок 1.33: Сохранение файла в Windows*

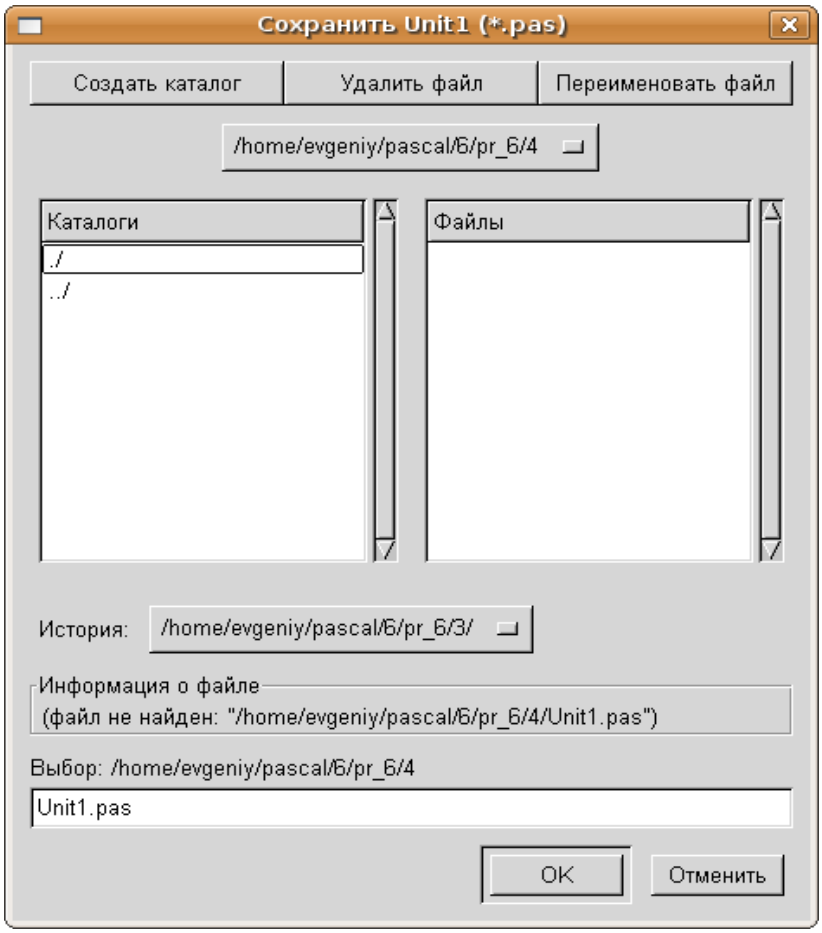


*Рисунок 1.34: Сохранение файла в Linux*

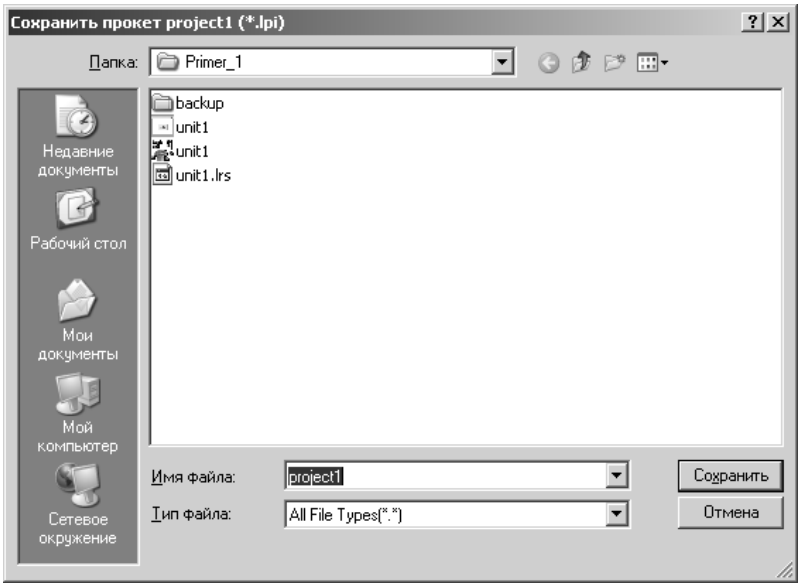


*Рисунок 1.35: Сохранение проекта в Windows*

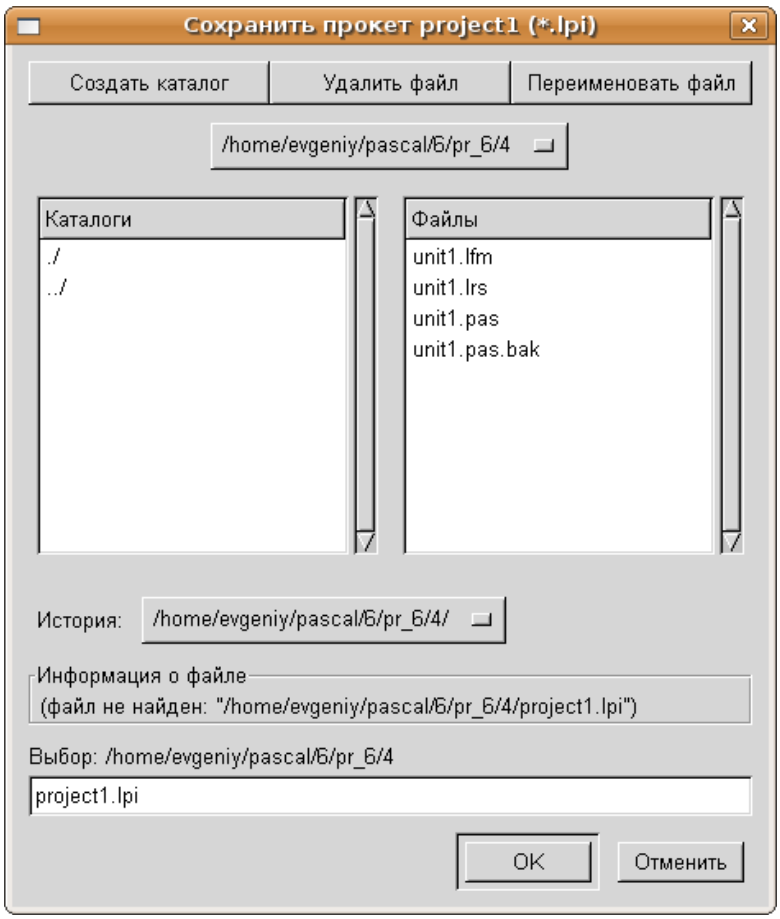


*Рисунок 1.36: Сохранение проекта в Linux*

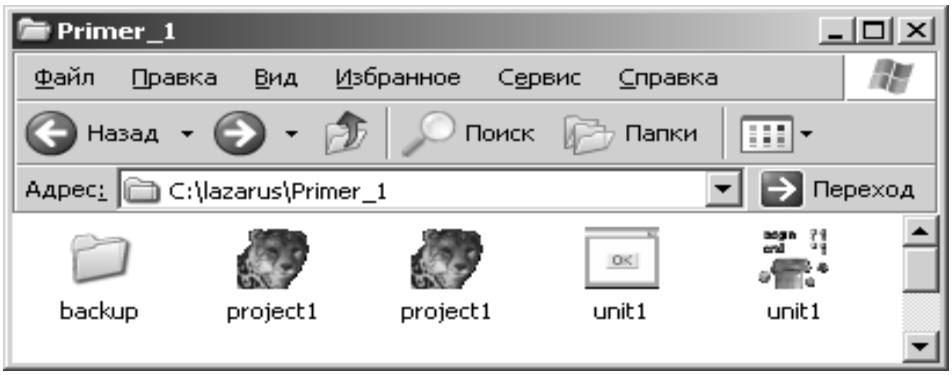


*Рисунок 1.37: Файлы проекта в Windows*

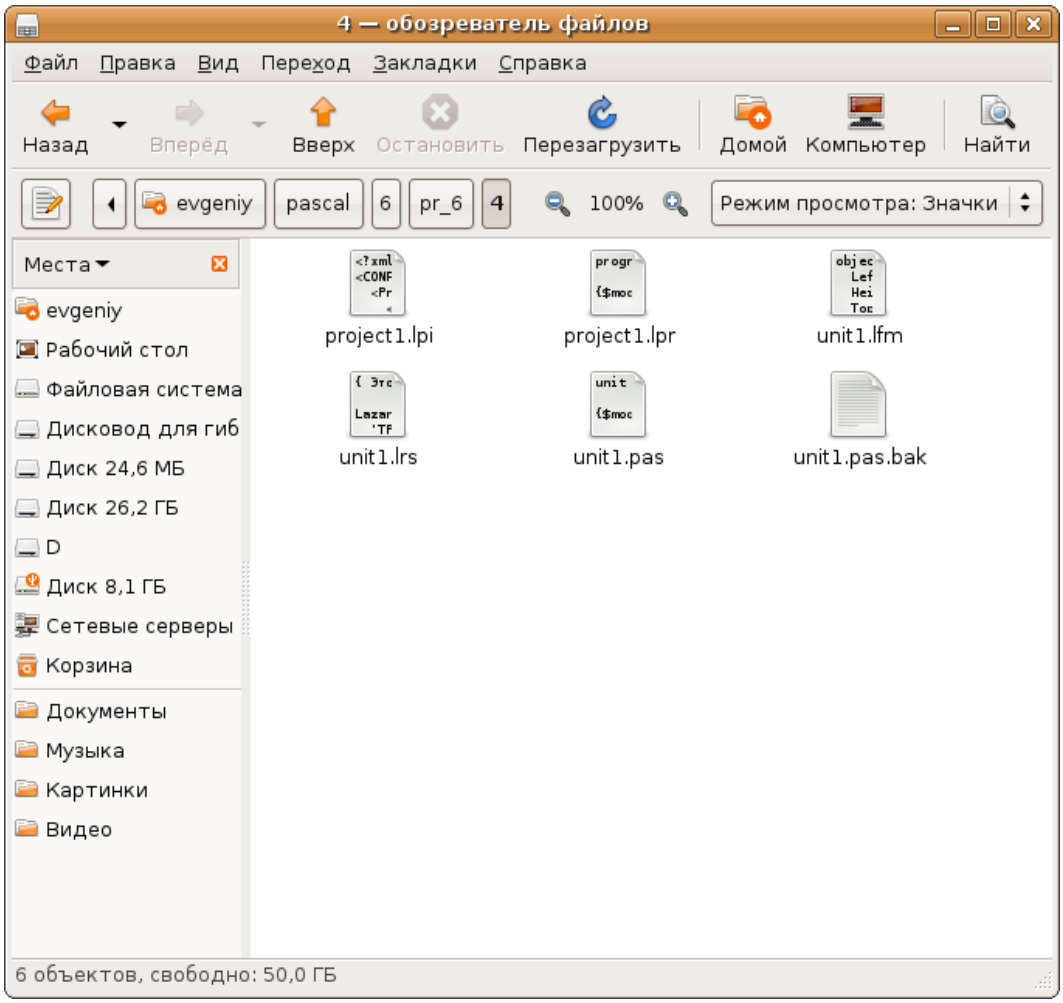


*Рисунок 1.38: Файлы проекта в Linux*

Итак, проект сохранен. Все дальнейшие изменения будем сохранять командой **Проект - Сохранить проект**.

Теперь можно приступить к визуальному программированию. У нас есть один объект – форма `Form1`. Изменим некоторые его свойства с помощью инспектора объектов. Перейдем в окно инспектора объектов и найдем там свойство `Caption` (Заголовок). По умолчанию это свойство имеет значение `Form1`. Изменим его на слово ПРИ-

МЕР 1 (рис. 1.39). Это изменение сразу же отразится на форме – в поле заголовка появится надпись – ПРИМЕР 1 (рис. 1.40). Аналогично можно изменить размеры формы, присвоив новые значения свойствам `Height` (Высота) и `Width` (Ширина), еще проще это сделать, изменив положение границы формы с помощью кнопки мыши, как это делается со всеми стандартными окнами.

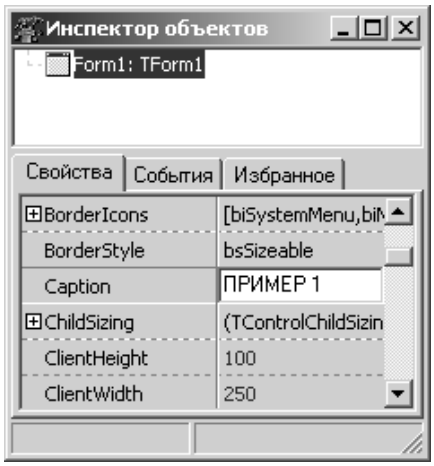


*Рисунок 1.39: Изменение свойства формы Caption*

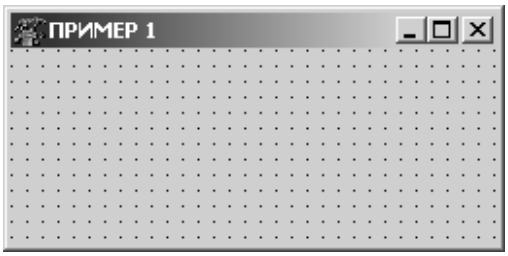


*Рисунок 1.40: Изменение заголовка формы*

Изменим еще одно свойство формы – `Position` (Положение формы на экране). Значения этого свойства не вводятся вручную, а выбираются из списка (рис. 1.41). Если выбрать свойство `poScreenCenter`, то форма всегда будет появляться в центре экрана.

Напомним, что мы пишем программу про кнопку. Пришло время поместить ее на форму. Для этого обратимся к панели компонентов (рис. 1.29) и найдем там компонент `Tbutton`. Чтобы *разместить компонент* на форме, нужно сделать два щелчка мышкой: первый — по компоненту, второй — по окну формы. В результате форма примет вид, представленный на рис. 1.42.

Теперь у нас два объекта: форма `Form1` и кнопка `Button1`. Напомним, что изменения значений свойств в окне инспектора объектов относятся к выделенному объекту. Поэтому выделим объект `Button1` и изменим его заголовок `Caption` и размеры `Height`, `Width`. В заголовке напишем фразу «`Щелкни мышкой!`», установим ширину 135 пикселей, а высоту – 25 пикселей. *Сохраним* проект (**Проект — Сохранить проект**).

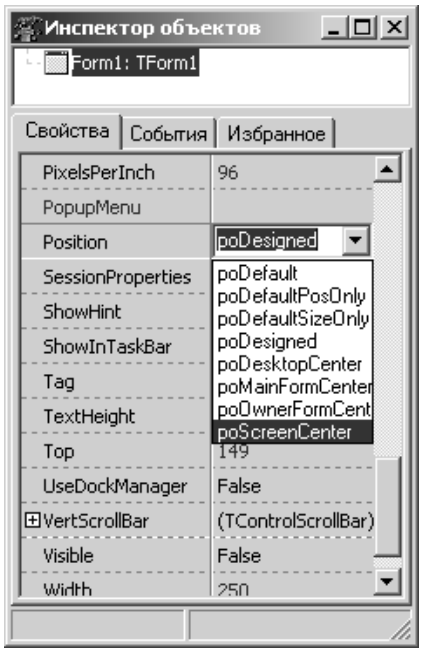


*Рисунок 1.41: Изменение свойства формы Position*

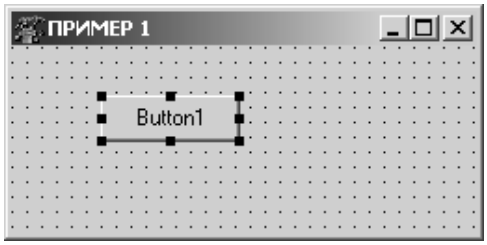


*Рисунок 1.42: Размещение кнопки на форме*

Для того чтобы посмотреть, как работает наша программа, ее необходимо *запустить на выполнение*. Сделать это можно командой **Запуск — Запуск**, функциональной клавишей **F9** или кнопкой **Запуск** в панели инструментов (рис. 1.43). Созданное окно с кнопкой должно выглядеть, как на рис. 1.44.


*Рисунок 1.43: Кнопка Запуск в панели инструментов Lazarus*

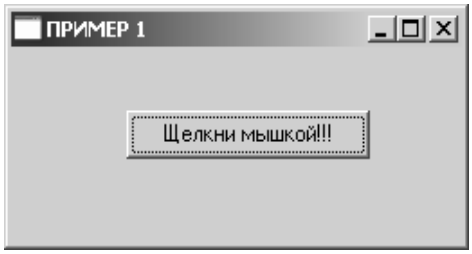


*Рисунок 1.44: Результат работы программы*

Теперь щелкните по кнопке и убедитесь, что ничего не произойдет. В этом нет ничего удивительного, ведь пока мы не написали ни одной строчки программного кода. Можно сказать, что мы разработали только внешнюю часть программы, но не позаботились о функциональной. Несмотря на это, в окне редактора появился текст программы:

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
 Classes, SysUtils, LResources, Forms,
   Controls,   Graphics, Dialogs, StdCtrls;
type
 { TForm1 }
 TForm1 = class(TForm)
 Button1: TButton;
 private
     { private declarations }
 public
     { public declarations }
end;
var
 Form1: TForm1;
implementation
initialization
 {$I unit1.lrs}
end.
```

О назначении всех перечисленных в листинге команд мы поговорим позже. Вернемся к программе и подумаем, что же мы хотим от нашей кнопки. Скорее всего, это должна быть реакция на какое-то событие, например, на щелчок мыши. После запуска программы на экране появилось окно, на котором расположена кнопка с надписью «Щелкни мышкой!». Предположим, что если мы щелкнем по ней, она «обрадуется» и сообщит: «УРА! ЗАРАБОТАЛО!».

Для воплощения этой идеи завершим работу программы, закрыв окно с кнопкой обычным способом (щелчком по крестику в правом верхнем углу) и перейдем в режим редактирования. Убедимся в том, что объект кнопка `Button1` выделен, и обратимся к вкладке **События** инспектора объектов. Напомним, что здесь расположены описания событий. Выберем событие `OnClick` – *обработка щелчка мыши* и дважды щелкнем в поле справа от названия (рис. 1.45).

В результате описанных действий в окне редактора появиться следующий текст:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
end;
```

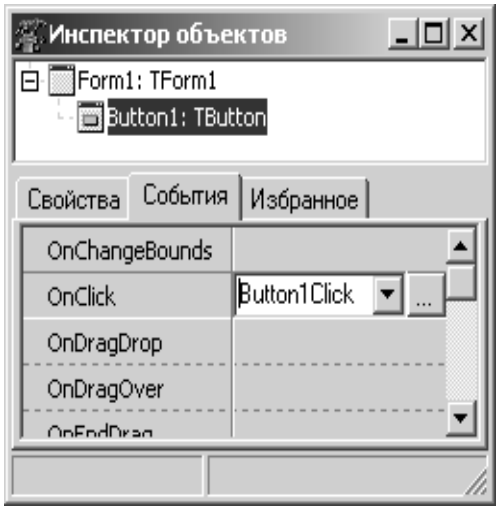


*Рисунок 1.45: Выбор события OnClick*

На данном этапе изложения материала данный текст – это фрагмент программного кода, именуемый *подпрограммой*. О назначении этой подпрограммы можно догадаться по ее имени `TForm1.Button1Click`: на форме `Form1` для объекта кнопка `Button1` обрабатывается событие «щелчок мыши» `Click`. Все команды, написанные между словами `begin` и `end`, будут выполняться при наступлении указанного события.

Теперь установим курсор между словами `begin` и `end` созданной подпрограммы и напишем:

```
Button1.Caption:='УРА! ЗАРАБОТАЛО!';
```

Эта запись означает изменение свойства кнопки. Только теперь мы выполнили его не с помощью инспектора объектов, а записав оператор языка программирования. Прочитать его можно так: присвоить (`:=`) свойству `Caption` объекта `Button1` значение `'УРА! ЗАРАБОТАЛО!'`. Поскольку присваиваемое значение – строка, оно заключено в одинарные кавычки. Теперь, текст подпрограммы в окне редактора имеет вид:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
     Button1.Caption:='УРА! ЗАРАБОТАЛО!';
end;
```


*Рисунок 1.46: Окончательный результат работы программы*

Сохраним созданную программу, запустим ее на выполнение и убедимся, что кнопка реагирует на щелчок мыши (рис. 1.46). *Закрыть проект* можно командой **Проект — Закрыть проект**.

### 1.4.10 Полезная программа

Итак, первая программа написана. Она не несет особой смысловой нагрузки. Главная ее задача – показать, что же такое визуальное программирование. Следующая программа также будет несложной, но полезной. Мы автоматизируем процесс перевода старой русской меры веса в современную. Эта задача формулируется так: выполнить перевод пудов и фунтов в килограммы, если известно, что 1 пуд = 40 фунтам = 16.38 кг.

Итак, наша программа должна позволить пользователю ввести два числа (пуды и фунты) и после щелчка по кнопке сообщить, каково значение в килограммах.

Создадим новый проект, выполнив команду **Проект — Создать проект...**, и сохраним его в папке **Primer_2** командой **Проект — Сохранить проект как..**.

Воспользуемся вкладкой **Standard** панели компонентов и разместим на форме следующие компоненты. Четыре *объекта типа надпись* `Label`. Этот компонент используется для размещения в окне не очень длинных однострочных надписей. Два *поля ввода* `Edit`. Это редактируемое текстовое поле, предназначенное для ввода, отображения или редактирования одной текстовой строки. И одну *кнопку* `Button`. Этот компонент обычно используют для реализации некоторой команды. Компоненты должны располагаться на форме примерно так, как показано на рис. 1.47.

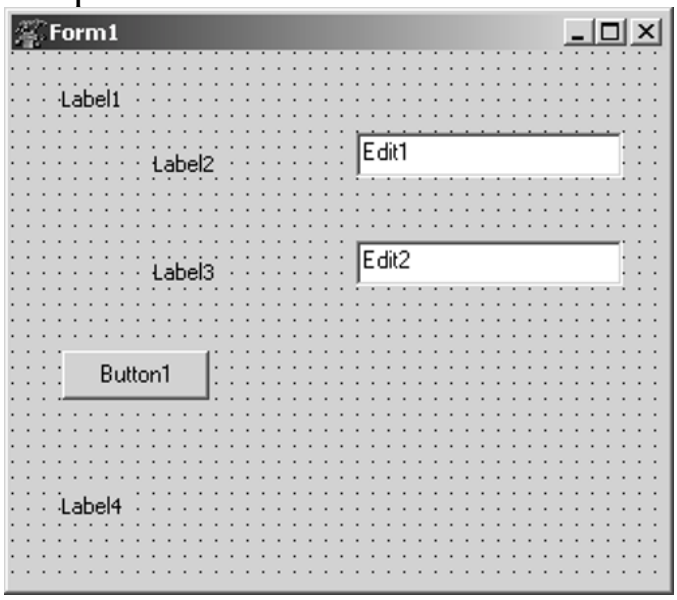


*Рисунок 1.47: Конструирование формы*

В первой надписи будет храниться текст «Меры веса». Две надписи `Label2` и `Label3` нужны для пояснения того, какие именно данные должны вводиться, надпись `Label4` понадобится для вывода результата.

В поля ввода `Edit1` и `Edit2` будут введены конкретные значения: вес в пудах и фунтах.

Вычисление веса в килограммах произойдет при щелчке по кнопке `Button1` и будет выведено в `Label4`.

Изменим свойства формы и размещенных на ней компонентов в соответствии с табл. 1.1-1.4.

Таблица 1.1 Значение свойств формы

| **Свойство** | **Значение** | **Описание свойства** |
|---|---|---|
| `Caption` | Перевод из старой русской меры веса в современную | Заголовок формы |
| `Height` | 230 | Высота формы |
| `Width` | 400 | Ширина формы |
| `Font.Name` | Arial | Название шрифта |
| `Font.Size` | 10 | Размер шрифта |

Таблица 1.2. Свойства компонентов типа надпись

| **Свойство** | **Label1** | **Label2** | **Label3** | **Label4** | **Описание свойства** |
|---|---|---|---|---|---|
| `Caption` | Меры веса | пуды | фунты | В килограммах: | Заголовок компонента |
| `Height` | 15 | 15 | 15 | 15 | Высота компонента |
| `Width` | 60 | 30 | 35 | 85 | Ширина компонента |
| `Top` | 20 | 50 | 105 | 25 | Координата верхнего края |
| `Left` | 25 | 115 | 115 | 200 | Координата левого края |

Таблица 1.3. Свойства компонентов поле ввода

| Свойство | Edit1 | Edit2 | Описание свойства |
|---|---|---|---|
| Text | пробел | пробел | Текст в поле ввода |
| Height | 25 | 25 | Высота компонента |
| Width | 135 | 135 | Ширина компонента |
| Top | 40 | 95 | Координата верхнего края |
| Left | 175 | 175 | Координата левого края |

Таблица 1.4. Свойства компонента кнопка

| Свойство | Значение | Описание свойства |
|---|---|---|
| Caption | ВЫЧИСЛИТЬ | Текст на кнопке |
| Height | 25 | Высота компонента |
| Left | 200 | Ширина компонента |
| Top[20] | 150 | Координата верхнего края |
| Width | 80 | Координата левого края |

После всех изменений форма будет иметь вид, представленный на рис. 1.48.


*Рисунок 1.48: Изменение свойств формы и ее компонентов*

Интерфейс программы готов. Займемся вычислительной частью задачи. Для этого создадим для кнопки Button1 обработчик событий OnClick, для чего дважды щелкнем по объекту Button1 левой кнопкой мыши. Между словами begin и end созданной подпрограм-

20 Свойства Top и Left определяют расположение компонента относительно верхнего левого угла формы.

мы запишем несколько команд, чтобы получился следующий программный код:

```
procedure TForm1.Button1Click(Sender: TObject);
var
pud, funt: integer; kg: real;
begin
     pud:=StrToInt(Edit1.Text);
     funt:=StrToInt(Edit2.Text);
     kg:=pud*16.38+funt*16.38/40;
     Label4.Caption:='В килограммах:
                              '+FloatToStr(kg);
end;
```

Рассмотрим программный код построчно.

Строка 1. *Заголовок подпрограммы*. Он состоит из ключевого слова `procedure` и имени подпрограммы `TForm1.Button1Click`, а в скобках обычно указывают список параметров подпрограммы, если таковые имеются.

Строка 2. Открывает *блок описания переменных* (`var` – от `variable` переменная);

Строка 3. *Описывает переменные* `pud` и `funt`. Компьютер понимает эту запись как команду выделить в памяти место для хранения двух целых чисел[21].

Строка 4. Описывает одной вещественной переменной `kg`.

Строка 5. Начинает программный блок (`begin` – *начало*).

Строка 6. Команда, выполняющая следующие действия. Из свойства `Text` поля ввода `Edit1` считывает введенную туда информацию. Эта информация воспринимается компилятором как строка текста (например, '360' – это строка из трех символов), а поскольку нам для вычислений нужны числовые значения, то функция `StrToInt` преобразовывает ее в целое число. Результат этих преобразований записывается в память компьютера под именем `pud`. Символы «:=» обозначают *оператор присваивания*. Например, запись `a:=3.14` читается так: переменной `a` присвоить значение `3.14`. Выражение `3.14:=a` не имеет смысла.

Строка 7. Информация из поля ввода `Edit2` преобразовывается в целое число и записывается в переменную `funt`. Фактически ко-

21 О переменных и их типах подробнее рассказывается в следующей главе.

манды, записанные в шестой и седьмой строках, осуществляют *ввод исходных данных*.

Строка 8. *Вычисляется значение выражения,* результат которого присваивается переменной `kg`. Обратите внимание, что *умножение* здесь обозначается звездочкой, *деление* – наклонной чертой, *сложение* – знаком «+». Записывая математическое выражение на языке программирования, нужно четко указывать все операции. Нельзя, например, опустить знак умножения, как это принято в математике.

Строка 9. Изменяет свойство `Caption` объекта `Label4`. В него будет помещен текст, состоящий из двух строк. Первая – `'В килограммах:'`, вторая значение переменной `kg`. Знак «+» в этом выражении применяют для конкатенации (слияния) строк. Кроме того, поскольку значение переменной `kg` – вещественное число, оно преобразовывается в строку функцией `FloatToStr`. Фактически в этой строке был выполнен *вывод результатов*.

Строка 10. Конец программного блока (`end` – *конец*).

Теперь наша программа может быть откомпилирована и запущена на выполнение. Процесс *компиляции* представляет собой перевод программы с языка программирования в машинные коды. Для компиляции нужно выполнить команду **Запуск — Быстрая компиляция**. Во время компиляции проверяется наличие синтаксических ошибок. В случае их обнаружения процесс компиляции прерывается. Мы специально допустили ошибку в восьмой строке программного кода (пропустили двоеточие в операторе присваивания). Результат компиляции программы с синтаксической ошибкой показан на рис. 1.49. Строка с ошибкой выделена красным цветом, а под окном редактора появилось окно **Сообщение**[22], в котором описана допущенная ошибка. Ошибку нужно исправить, а программу еще раз откомпилировать.

После успешной компиляции всех файлов, входящих в проект, необходимо *собрать* (скомпоновать) проект из объектных (откомпилированных) файлов (с расширением **.o —** для Linux, .**obj** — для Windows). Для этого следует выполнить команду **Запуск — Собрать** или воспользоваться комбинацией клавиш **Ctrl+F9**.

В результате компоновки будет сгенерирован выполнимый файл, имя которого совпадает с именем проекта.

22 Если окно **Сообщение** отсутствует, его можно вывести на экран командой **Просмотр — Сообщения**.

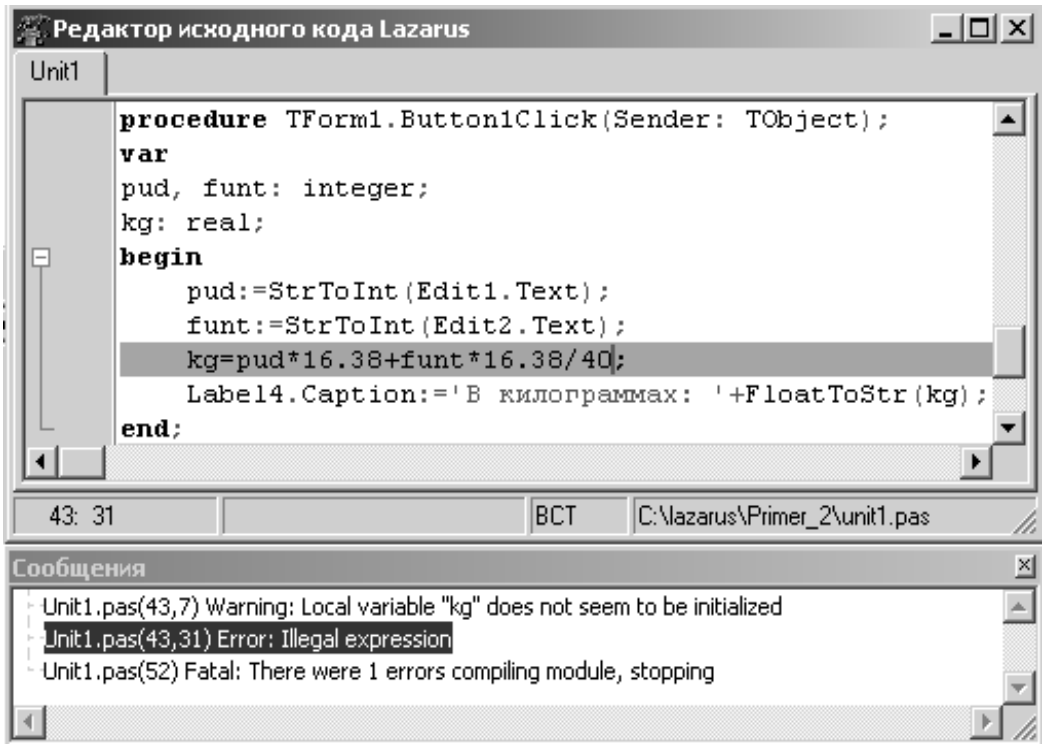


*Рисунок 1.49: Сообщение об ошибке в программе*

Вообще говоря, если команду **Запуск — Быстрая компиляция** опустить, а сразу выполнить команду **Запуск — Собрать**, то произойдет и компиляция всех файлов проекта и его компоновка.

После успешной компиляции и сборки программу нужно *сохранить* (**Проект - Сохранить**) и *запустить*.

Запуск осуществляется из среды разработки (**Запуск - Запуск**) или выполнением файла, Linux — **./project1**, в Windows — **projec1.exe**. На рис. 1.50 показаны результаты работы нашей программы.

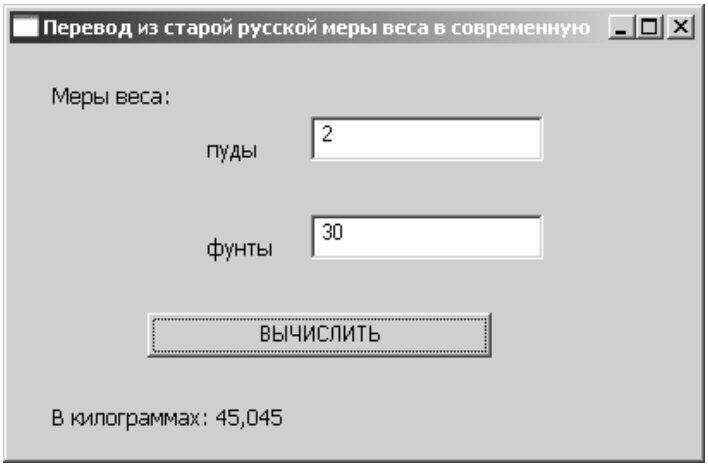


*Рисунок 1.50: Результат работы программы*

Обратите внимание, что ошибки могут возникать и после запуска программы. Например, в нашем примере исходные данные – целые числа. Если ввести дробь, появится аварийное сообщение. В подобной ситуации следует *прервать выполнение программы* командой **Запуск—Останов** (**Ctrl+F2**).

*Вернуться к созданному проекту*, например с целью его усовершенствования, несложно. Нужно выполнить команду **Проект — Открыть проект**… (**Ctrl+F11**), перейти к папке с проектом и выбрать из списка файл с расширением **.lpi** (в нашем случае — **project1.lpi**).

### 1.4.11 Консольное приложение среды Lazarus

В Lazarus можно создавать не только *визуальные приложения* с графическим интерфейсом, но и писать программы на Free Pascal в *текстовом режиме*. Для этого существует *консольное приложение*. Чтобы *создать новое консольное приложение,* нужно выполнить команду **Проект — Создать проект...**, в появившемся диалоговом окне (рис. 1.31, 1.32) выбрать фразу **Программа пользователя** и нажать кнопку **Создать**.

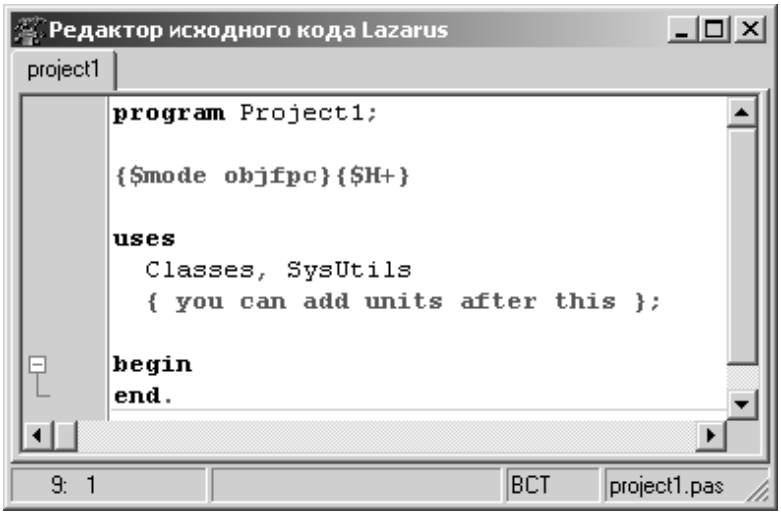


*Рисунок 1.51: Окно редактора в консольном приложении*

На экране появится окно редактора программного кода (рис. 1.51), в котором уже представлена *общая структура программы* на языке Паскаль.

В общем виде *программа на языке Free Pascal* состоит из заголовка программы, раздела описаний и непосредственно тела программы.

*Заголовок программы* состоит из ключевого слова `program` и имени программы. В нашем случае (рис. 1.51) это `Project1`. Имя было присвоено программе автоматически. При желании его можно изменить.

*Раздел описаний* обычно включает в себя описание констант, типов, переменных, процедур и функций. На рис. 1.51 видно, что раздел описаний начинается со слова `uses`. Это раздел подключения модулей. *Модуль* — это специальная программа, которая расширяет возможности языка программирования. В нашем случае происходит подключение модуля `SysUtils`, который позволяет работать с файлами и каталогами, и модуля `Classes`, который работает с компонентами.

За разделом описаний следует исполняемая часть программы, или *тело программы*. Оно начинается со служебного слова `begin` и заканчивается служебным словом `end` и точкой. Тело программы содержит операторы языка, предназначенные для реализации поставленной задачи.

Кроме того в тексте программы могут встречаться комментарии. *Комментарий* — это текст, заключенный в фигурные скобки или начинающийся с двух наклонных. Этот текст не является программным кодом, а носит информационный характер. Например, в нашем случае текст заключенный в фигурные скобки, сразу после описания модулей сообщает пользователю о том, что остальные элементы языка он может добавить самостоятельно.

Рассмотрим **пример**. Пусть нужно решить *задачу перевода градусной меры угла в радианную*. Эта задача известна из школьного курса и формулируется так: чтобы найти радианную меру какого-нибудь угла по данной градусной мере, нужно умножить число градусов на $\frac{\pi}{180}$ , число минут на $\frac{\pi}{180 \cdot 60}$ и найденные произведения сложить.

Текст программы для решения поставленной задачи в консольном приложении будет иметь вид:

```
program Project1;
{$mode objfpc}{$H+}
uses
 Classes, SysUtils
     { you can add units after this };
var
 gradus,minuta:integer; radian: real;
begin
 write('gradus=');
 readln(gradus);
 write('minuta=');
 readln(minuta);
 radian:=gradus*pi/180+minuta*pi/(180*60);
 writeln('radian=', radian);
end.
```

*Сохранить*, *открыть*, *откомпилировать*, *скомпоновать* и *запустить* на выполнение программу в консольном приложении можно так же, как в визуальном проекте.

Результаты работы нашей программы будут иметь вид:

```
gradus=165
minuta=30
radian=2.88851991205O6E+000
```

Нетрудно заметить, что к тексту, созданному автоматически, мы добавили описание переменных (все используемые в программе переменные должны быть описаны):

```
Var
 //Описаны две целочисленные переменные.
 gradus,minuta:integer;
 //Описана вещественная переменная.
 radian: real;
```

и тело программы:

```
begin                 //Начало тела программы.
 //Вывод на экран строки символов gradus=
 write('gradus=');
 //Ввод переменной gradus.
 readln(gradus);
 //Вывод на экран символов minuta=.
 write('minuta=');
 //Ввод переменной minuta.
 readln(minuta);
 //Вычисление.
 radian:=gradus*pi/180+minuta*pi/(180*60);
 //Вывод результата вычислений.
 writeln('radian=', radian);
end. //Конец программы.
```

Так как в этой программе графический интерфейс отсутствует, мы разработали элементарный текстовый диалог компьютер — пользователь. Для этого были использованы операторы *ввода* (`read`) и *вывода* (`write`) данных.

### 1.4.12 Операторы ввода - вывода данных

*Ввод информации с клавиатуры* осуществляется с помощью оператора `read`. Он может иметь один из следующих форматов:

```
read (x1, x2, …, xn); readln (x1, x2, …, xn);
```

где `x1`, `x2`, …, `xn` – список вводимых переменных. При вводе вещественных значений целую и дробную часть числа следует разделять точкой.

Когда в программе встречается оператор `read`, ее действие приостанавливается до тех пор, пока не будут введены исходные данные. При вводе числовых значений два числа считаются разделенными, если между ними есть хотя бы один пробел, символ табуляции или конца строки (**Enter**). После ввода последнего значения следует нажать **Enter**.

Оператор `readln` аналогичен оператору `read`, разница заключается в том, что после считывания последнего в списке значения для одного оператора `readln` данные для следующего оператора `readln` будут считываться с начала новой строки. Но следует помнить, что **Enter** переведет курсор на новую строку независимо от того, как именно происходит считывание данных.

Для *вывода информации* на экран служат операторы `write` и `writeln`. В общем случае эти операторы имеют вид:

```
write(x1, x2, …, xn); writeln(x1, x2, …, xn);
```

где `x1`, `x2`, …, `xn` представляют собой список выводимых переменных, констант, выражений. Если элемент списка — текстовая информация, ее необходимо взять в кавычки.

Операторы `write` и `writeln` последовательно выводят все переменные. Если используется оператор `writeln`, то после вывода информации курсор перемещается в новую строку.

Итак, в нашем примере оператор `write('gradus=');` выводит на экран символы `gradus=`, которые подсказывают пользователю, что он должен ввести значение переменной `gradus`, а оператор `readln(gradus);` предназначен для ввода значения переменной `gradus`. Оператор `writeln('radian=', radian);` выводит на экран два значения: строку `radian=` и значение переменной `radian`.

Как правило, вещественные данные выводятся в *формате с плавающей точкой* # . # # # # # # # # # # # # E ± # # #, где # - любая десятичная цифра от 0 до 9. Для того чтобы перейти к *формату с фиксированной точкой* нужно, число, расположенное до символа E, умножить на 10, возведенное в степень, значение которой указано после числа E. Например,

0.35469000000E-01=0.35469000000*$10^{-1}$=0.035469,

-5.43286710000E+02=-5.43286710000*$10^{2}$=-543.28671.

Чтобы выводить числа в формате с фиксированной точкой, необходимо использовать *форматированный вывод*. Для этого оператор `write` или `writeln` нужно задать следующим образом:

```
write(идентификатор:ширина_поля_вывода:дробная_часть);
```

где

`идентификатор` — это имя переменной, которая будет выводится на экран;

`ширина_поля_вывода` — количество позиций (целая часть, точка, дробная часть), которое будет занимать значение переменной при выводе;

`дробная_часть` — количество позиций, необходимых для дробной части числа.

Например, пусть результат работы нашей программы имеет вид

```
gradus=165
minuta=30
radian=2.8885199120506E+000
```

Понятно, что оператор `writeln('radian=', radian);` вывел на экран значение переменной `radian` в формате с плавающей точкой.

Если воспользоваться оператором

```
writeln('radian=', radian:5:3);
```

то результат работы программы будет таким:

```
gradus=165
minuta=30
radian=2.889
```

где значение переменной `radian` представлено в формате с фиксированной точкой (все число занимает пять позиций и три из них после запятой).