



Кафедра конструирования
и производства радиоаппаратуры

Консольные программы на *Free Pascal*



Кобрин Юрий Павлович

Консольные программы на *Free Pascal*. 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 знакомству со структурой программы на *Free Pascal/Lazarus*, а также с особенностями работы с простейшими программными объектами (константами, переменными, выражениями и операторами) в консольной версии *Free Pascal*.

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

©Кафедра КИПР федерального государственного бюджетного образовательного учреждения высшего профессионального образования «Томский государственный университет систем управления и радиоэлектроники (ТУСУР)», 2017.

© Кобрин Ю.П. 2017

Оглавление

1	Цели и задачи работы.....	4
2	Порядок выполнения работы	4
3	Отчётность.....	4
4	Контрольные вопросы	5
5	Краткие теоретические сведения	5
5.1	Основные этапы решения задач на компьютере	5
5.2	Формы Бэкуса–Наура	7
5.3	Программы на Free Pascal/Lazarus.....	8
5.3.1	Общие сведения.....	8
5.3.2	Консольные приложения	10
5.4	Подключение модулей.....	12
5.5	Константы.....	12
5.5.1	Типизированные именованные константы	14
5.6	Переменные	15
5.7	Выражения.....	16
5.7.1	Последовательность выполнения операций.....	16
5.7.2	Результаты вычисления выражений	18
5.7.3	Некоторые встроенные функции	19
5.7.4	Функции модуля Math	20
5.8	Операторы языка.....	21
5.8.1	Оператор присваивания	21
5.8.2	Операторы ввода/вывода	22
5.9	Работа в редакторе исходного текста <i>Free Pascal</i>	24
6	Индивидуальные задания	25
6.1.1	Варианты заданий.....	25
6.1.2	Рекомендации по составлению программы:	28
6.1.3	Пример разработки программы.....	28
7	Список литературы.....	30

1 Цели и задачи работы

Знакомство с основными элементами языка программирования *Free Pascal/Lazarus*: переменными, их типами, основными операциями и функциями.

В ходе выполнения работы следует усвоить:

- структуру программ на *Free Pascal/Lazarus*;
- назначение основных объектов на *Free Pascal/Lazarus*.

Научиться:

- выбирать и объявлять основные типы данных в программах *Free Pascal*;
- усвоить правила использования констант, переменных и операторов в *Free Pascal/Lazarus*;
- освоить правила формирования выражений в *Free Pascal/Lazarus*;
- работать в консольной версии *Free Pascal*.

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].
- 2) Выполнить индивидуальное задание по своему варианту.
- 3) Загрузить систему программирования *Free Pascal/Lazarus*.
- 4) Войти в режим редактирования и набрать текст программы.
- 5) Запустить программу на трансляцию и выполнение.
- 6) Исправить ошибки, получить и проанализировать результаты.
- 7) Ответить на контрольные вопросы.
- 8) Оформить отчёт и защитить его у преподавателя.

3 Отчётность

Отчёт должен быть выполнен в соответствии с [7] и состоять из следующих разделов:

- 1) Тема и цель работы.
- 2) Индивидуальное задание.
- 3) Откомпилированный текст программы (в электронном виде).
- 4) Ответы на контрольные вопросы.
- 5) Выводы.

Защита лабораторной состоит в ответах на контрольные вопросы по теоретическому и практическому разделам работы. При защите отчёта по работе для получения зачёта студент должен:

- уметь отвечать на контрольные вопросы;
- показать, как работает разработанное приложение;
- продемонстрировать навыки работы в консольной версии *Free Pascal*.

4 Контрольные вопросы

- 1) Что такое консольное приложение? Чем оно отличается от визуального?
- 2) Перечислите стандартные типы языка *Free Pascal*. Каковы диапазоны допустимых значений для целых и вещественных типов данных?
- 3) Какова структура консольной программы *Free Pascal*?
- 4) Как описать именованные константы?
- 5) Для какой цели используются типизированные константы?
- 6) В каком порядке выполняются операции в выражениях?
- 7) Как работает оператор присваивания?
- 8) Как ввести в консольную программу данные с клавиатуры?
- 9) Как записываются операторы вывода на экран в языке *Free Pascal*?
- 10) С какой целью осуществляется форматирование результатов вывода?

5 Краткие теоретические сведения

5.1 Основные этапы решения задач на компьютере

При решении на компьютере любой задачи можно выделить несколько этапов:

- математическая постановка задачи;
- выбор метода решения задачи;
- разработка алгоритма;
- программирование;
- отладка программы;
- подготовка исходных данных и непосредственное выполнение программы.

Математическая постановка задачи. На этом этапе всем физическим величинам, участвующим в задаче, необходимо дать математические обозначения: уяснить их размерность, возможный диапазон значений. Целесообразно использовать общепринятые обозначения (*Summa* - сумма, *h* - высота, *l* - длина, *S* - площадь и т.д.) из соответствующей предметной области.

Затем следует установить - к какой из следующих категорий относится каждая переменная:

- **исходные данные** - эти величины известны из условия задачи;
- **результат** - эти величины требуется найти;
- **промежуточные данные** - эти величины заранее не известны, они используются в ходе решения задачи (разумно анализировать состояние этих величин при пошаговой тестовой отладке программы).

После этого следует записать известные ограничения для величин и связи между ними в виде неравенств, формул и уравнений, а также определить условия окончания алгоритма.

Выбор метода решения задачи. Для огромного большинства задач методы решения уже разработаны, и зачастую их можно найти в литературе в нескольких вариантах. Остаётся только предпочесть метод, который больше всего отвечает условиям задания (ми-

нимальный объем памяти, минимальная трудоёмкость, максимальная эффективность, достаточная точность, допустимая погрешность вычислений и др.). Заметим, что разработано множество методов точных и приближенных (численных) вычислений (для поиска корней линейных и нелинейных алгебраических уравнений, решения обыкновенных уравнений и уравнений в частных производных, вычисления определённых интегралов, интерполяции, оптимизации и т.п.).

Нередко для реализации одного и того же численного метода можно применить несколько алгоритмов, различающихся по точности, сложности, объёму вычислительной работы, быстродействию, затратам памяти и т.д. Постарайтесь выбрать алгоритм, обеспечивающий наиболее эффективное использование компьютера.

Разработка алгоритма. На этом этапе разрабатывается подробный алгоритм решения задачи. Тщательная проработка алгоритма позволяет избежать многих логических ошибок.

Именно на этом этапе определяется успешность будущей программы. И если к разработке алгоритма отнестись недостаточно внимательно, то в дальнейшем, на этапе программирования появятся трудности, алгоритм потребует дополнительной доработки, затраты новых усилий и т.п. А на этапе отладки программы может выясниться, что, к сожалению, алгоритм даёт ошибки или вообще не выполним, и нужно разработать другой.

Удобнее всего сначала записать алгоритм в виде некоторого общего плана, состоящего из крупных этапов. Затем каждый этап разрабатывается более подробно. Изображать алгоритм целесообразнее всего в виде графической схемы. Необходимо продумать, в какой форме будет вводиться исходная информация для решения задачи, какие нужны промежуточные результаты, определить вид и структуру выходных данных (графики, таблицы и т.п.).

Программирование. Это процесс записи алгоритма на одном из алгоритмических языков программирования (Free Pascal, СИ, и др.). Если алгоритм разработан достаточно качественно, то на написание программы и её отладку не придётся тратить много времени.

Как правило, именно на отладку программы уходит подавляющая часть времени, поэтому программа должна быть наглядной, легко читаться, сопровождаться комментариями. В комментариях рекомендуется указывать назначение отдельных участков программы, смысл используемых переменных, пояснение наиболее сложных участков.

Отладка программы. Это процесс поиска (диагностики) и устранения ошибок в программе путём решения ее на контрольных (тестовых) примерах. Надо подобрать разные наборы исходных данных, чтобы проверить правильность работы программы при всех возможных исходах. Вместе с тем, исходные данные должны быть упрощёнными настолько, чтобы на каждом этапе сверить значения промежуточных результатов с результатами ручного подсчёта. Если решение задачи вручную невозможно или требует больших затрат, то для сравнения можно взять результаты решения, полученные по другому алгоритму.

Главное правило подбора тестовых примеров состоит в следующем: тесты следует подбирать таким образом, чтобы они не подтверждали правильность программы, а *выявляли имеющиеся в ней ошибки*.

Ошибки программы бывают двух видов:

- **синтаксические ошибки** записи операторов и ошибки ввода (описки). Такие ошибки обычно достаточно легко выявляются при трансляции программы. Большинство современных программ-трансляторов выводит на экран компьютера сообщение о характере ошибки и указывает место в программе, где эта ошибка была обнаружена.

- **семантические (смысловые, алгоритмические) ошибки** - выявляются на этапе отладки работающей программы, когда программа не производит ожидаемых действий, а сообщений об ошибках при этом нет. Обнаруживаются такие ошибки только при сравнении результатов машинного и ручного подсчёта.

При отладке программы высокой сложности используются следующие методы:

- вывод на экран или распечатка значений промежуточных данных помогает найти ошибки в вычислениях;
- пошаговое выполнение программы с просмотром на экране содержимого памяти (выполняется с помощью специальной программы - диалогового отладчика);
- трассировка программы (англ. *trace* - след) - вывод номеров операторов по мере их фактического выполнения, даёт возможность проследить последовательность выполнения отдельных действий и частей программы.

5.2 Формы Бэкуса–Наура

Для определения элементов языка и синтаксиса конструкций объектов программы на языках программирования обычно употребляют *формы Бэкуса–Наура* (сокр. **БНФ**, *Бэкуса - Наура форма*), в которых одни синтаксические категории последовательно определяются через другие категории.

В формах Бэкуса–Наура придерживаются следующих соглашений:

определяемое понятие отделяется от своего **определения**, находящегося в правой части формулы, знаком ::=, который заменяет выражение «определяется как» или «это есть»

определяемые понятия, обозначения и неделимые единицы текста заключаются в **треугольные скобки** < >. Текст внутри угловых скобок характеризует элемент, однако не описывает синтаксис этого элемента

допускаются **рекурсивные определения** (определение выражается через само себя в правой части)

необязательные элементы заключаются в квадратные скобки [];

элементы списков или альтернативные варианты конструкций (ИЛИ) отделяются друг от друга вертикальной чертой |;


стандартные множества задаются своими граничными значениями, отделенными друг от друга многоточием .. (т.е. указывают диапазон значений).

5.3 Программы на Free Pascal/Lazarus

5.3.1 Общие сведения

В программе программист согласно алгоритму решения задачи записывает последовательность действий, выполняемых над определёнными данными с помощью определённых операторов (инструкций) для реализации заданной цели. Набор текста программы осуществляется с помощью встроенного редактора текстов системы программирования Lazarus или любого другого редактора. В языке *Free Pascal* должны быть описаны все переменные, типы, константы, которые будут использоваться программой.

Операторы¹ в *Free Pascal* разделяются точкой запятой. Строка с оператором может начинаться с любой позиции, и это позволяет самим программистом устанавливать величину отступа от левой границы экрана для каждой строки, чтобы получить наиболее удобный для чтения текст программы. Количество операторов в строке произвольно, но если в строке записывается один оператор, то такая программа легче читается и проще отлаживается. Кроме того, удобнее записать и поясняющий комментарий к такому оператору.

Запустить среду *Lazarus* в Windows можно либо с помощью пиктограммы  на рабочем столе, либо из главного меню: **Пуск** ⇒ **Все программы** ⇒ **Lazarus** ⇒ **Lazarus**.

Любой проект в *Lazarus* – это совокупность файлов, из которых формируется единый выполняемый файл. Количество и типы файлов проекта зависят от вида программы и используемых режимов работы при создании программы.

Чтобы файлы разных проектов не смешивались, рекомендуется для каждого проекта создавать свою папку.

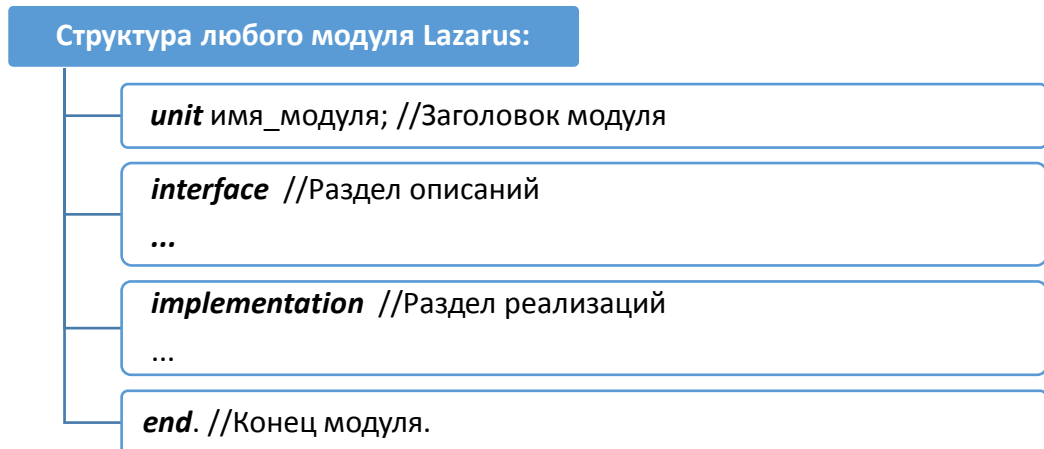
Список файлов проекта Lazarus имеет вид:

- файл описания проекта (**.lpi**);
- файл проекта (**.lpr**);
- файл ресурсов (**.lrs**);
- модуль формы (**.lfm**);
- программный модуль (**.pas**);

¹ **Оператор (инструкция, директива)** (от англ. *Statement* - предложение) – это единое и неделимое предложение, выполняющее **в программе** какие-либо определённые алгоритмические действия.

После компиляции² программы из всех файлов проекта создаётся единый исполняемый файл с расширением *.exe, имя которого совпадает с именем проекта.

Программный модуль, или просто **модуль**³, – это отдельно компилируемая программная единица, которая представляет собой набор типов данных, констант, переменных, процедур и функций. Процедуры и функции в Lazarus также построены по модульному принципу.



Заголовок модуля – это зарезервированное слово **unit**, за которым следует имя модуля и точка с запятой.

В **разделе описаний**, который открывается служебным словом **interface**, описывают используемые программные элементы: типы, классы, константы, переменные, процедуры и функции:

interface

```

uses список_модулей;
type список_типов;
const список_констант;
var список_переменных;
procedure имя_процедуры;
...
function имя_функции;
...

```

Порядок объявлений и описаний не регламентируется.

Раздел **implementation** содержит программный код, реализующий механизм работы описанных программных элементов (тексты процедур обработки событий, процедур и функции, созданные программистом).

² **Компиляторы** переводят всю программу целиком, и если перевод всей программы прошёл без ошибок, то полученный двоичный код можно запускать на выполнение уже без интегрированной среды.

³ **Модульность** - свойство программ, при котором объекты заключают в себе полное определение их характеристик. Никакие определения методов и свойств не должны располагаться вне объекта, что делает возможным свободное копирование и внедрение одного объекта в другие.

5.3.2 Консольные приложения

Кроме **визуальных приложений** (англ. *Application*), имеющих графический интерфейс под Windows, Lazarus позволяет разрабатывать и обычные **консольные приложения** (англ. *Custom Program*), имитирующие работу под операционной системой MS DOS с текстовым пользовательским интерфейсом (без графики), которые также могут быть созданы в оболочке Free Pascal.

Запуск среды программирования консольного приложения *Free Pascal* в Lazarus можно осуществить с помощью меню *Файл* ⇒ *Создать...* В появившемся диалоговом окне (Рис. 5.1) выбрать *Проект* ⇒ *Программа* и нажать кнопку *Ок*.

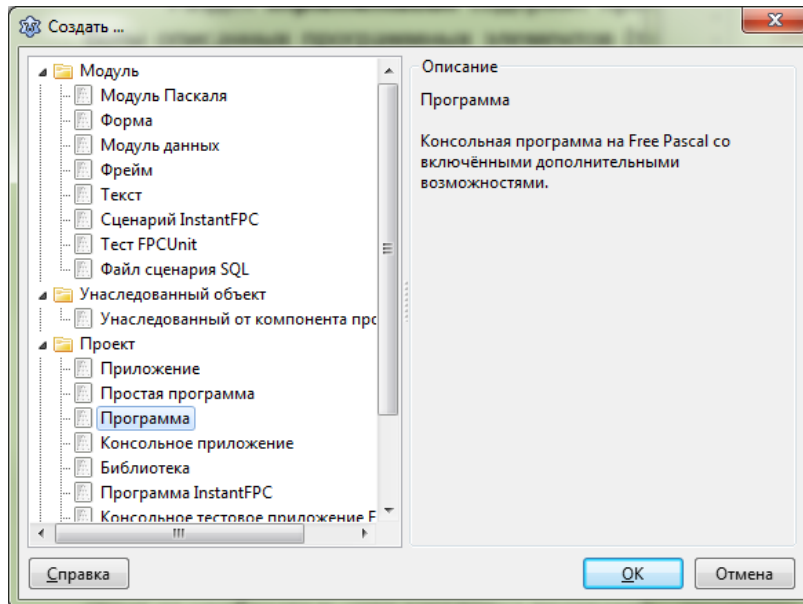


Рис. 5.1 - Диалоговое окно "Создать..."

На экране появится окно редактора исходного кода (Рис. 5.2), в котором представлена заготовка программы на языке *Free Pascal*.

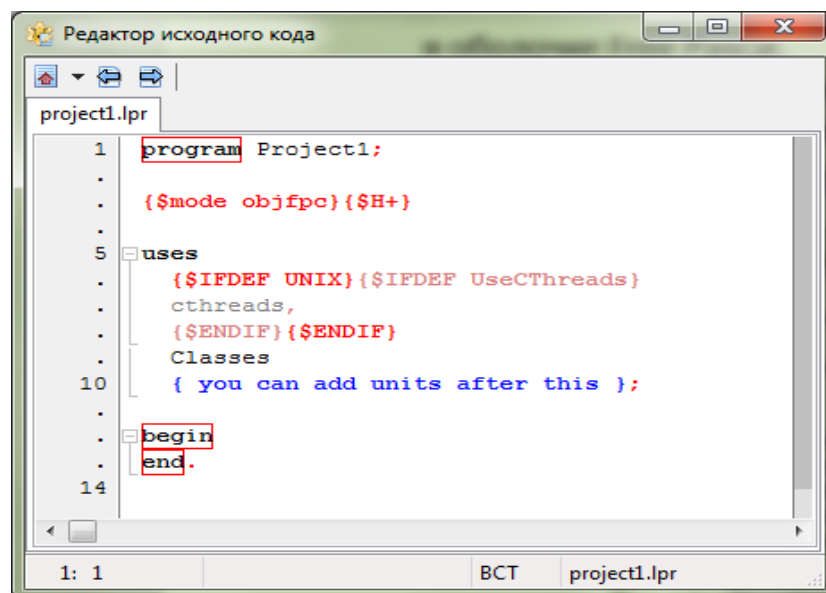


Рис. 5.2 – Заготовка программы в Редакторе исходного кода

Структурно консольная программа на *Free Pascal* состоит из необязательного заголовка программы (**program**), который называет программу, и основного программного блока, в котором между ключевыми словами **begin** и **end** находятся операторы, описывающие исполняемые программой действия (Таблица 5.1).

Таблица 5.1 - Структура консольной программы

Оператор начала раздела	Назначение	Примечание
Program Имя_программы;	Заголовок	Имя_программы (по умолчанию Project1) при желании можно изменить.
Uses Имя_модуля_1, ..., Имя_модуля_N;	Подключение модулей, библиотек	Разделы описаний программного блока, в котором должны быть описаны все объекты, используемые в программе (в любом порядке).
Label Имя_метки_1, ..., Имя_метки_N;	Описание меток	
Const Описание_константы_1, ..., Описание_константы_N;	Описание констант	
Type Описание_типа_1, ..., Описание_типа_N;	Описание типов данных	
Var Описание_переменной_1, ..., Описание_переменной_N;	Описание переменных	
Procedure Заголовок_процедуры; ... begin ... {Тело процедуры} end.	Описание процедур, используемых в программе	
Function Заголовок_функции; ... begin ... {Тело функции} end.	Описание функций, используемых в программе	
begin ... end.	Тело программы	Раздел исполняемых операторов

Рассмотрим подробнее элементы программы.

5.4 Подключение модулей

Основной блок любой программы на *Free Pascal* может включать в себя строку **uses** (англ. *Uses* - использовать), позволяющую программе использовать один или более модулей. К примеру, если Вы хотите изменить цвета выводимого на экран текста, то к Вашей программе необходимо использовать стандартный модуль *Crt*, являющийся стандартным модулем *Free Pascal*. Строка *uses Crt* подключит модуль *Crt* с нужными подпрограммами к Вашей программе, и станет допустимым, например, следующий фрагмент программы:

Uses

Crt; {Подпрограммы модуля *Crt* обеспечивают контроль над текстовыми режимами экрана, расширенными кодами клавиатуры, цветами, окнами и звуком}

...

{Подготовка экрана дисплея}

{Вызываем процедуры из модуля *CRT* для настройки режима экрана: }

TextColor(Blue); {установка цвета символов на экране (синего)}

TextBackGround(LightGray); {установка цвета фона экрана (светло

5.5 Константы

Константа характеризуется фиксированным именем, фиксированным типом и фиксированным значением.

Константами называются данные программы, значения которым присваиваются компилятором перед *началом* выполнения программы и не могут быть изменены до завершения её работы.

В качестве констант в *Free Pascal* можно использовать целые, вещественные и шестнадцатеричные числа, логические константы, символы, строки символов и др.

В языке *Free Pascal* можно использовать значение константы непосредственно **в явном виде** (без имени, как операнд выражения), например:

100 'A' (2.5 + 1) / (2.5 - 1) 'Free ' + 'Pascal'

Тип констант автоматически распознается компилятором без предварительного описания.

Значения констант можно задать в описательной части программы. В *Free Pascal* есть возможность дать константе имя, и затем в последующем тексте программы всюду вместо этой константы обращаться к ней по имени.

Именованная константа - это имя (идентификатор), которое в программе используется вместо самой константы.

Например, если Вы многократно используете в программе номер Вашего телефона, то его лучше всего задать именованной константой, так как он не меняет своего значения.

Все именованные константы должны быть описаны в специальном разделе, который начинается зарезервированным словом **const** (англ. *constant* – постоянный). В правой части допускается использовать не только значения константы, но и выражения⁴, в которых могут быть использованы ранее объявленные константы.

```
<объявление константы> ::= const <список>;
<список> ::= <элемент>[;<список>]
<элемент> ::= <имя> = <выражение>
```

Так как тип константы в объявлении не указан, то её значение в программе *менять нельзя*.

В выражениях-константах допускается использовать следующие стандартные функции:

Abs, Chr, Hi, High, Length, Lo, Low, Odd, Ord, Pred, Ptr, Round, SizeOf, Succ, Swap, Trunc.

Примеры описания констант:

```
const //Раздел объявления констант
{Числовые константы - записываются обычным образом}
Gamma = 5.6712E - 3;
Eps = 1e - 5;
Ln10 = 2.302585092994095684;
Ln10R = 1 / Ln10;
Min = 0; Max = 100;
Center = (Max - Min) div 2;
```

$\pi = 3.1415$
 92653589793
 238462643383
 279502884197169
 39937510582097494
 4592307816406286208998

В *Free Pascal* имеется ряд констант, к значениям которых можно обращаться без предварительного определения. Их называют **зарезервированными константами**.

Например, для вызова числа π можно вызвать функцию **Pi** (без параметров)

⁴ Значения всех выражений в разделе объявлений констант вычисляются один раз на шаге компиляции, перед выполнением программы.

В разделе описания констант можно также описать символьные и строковые константы:

const //Раздел объявления констант

{Символьные константы}

Char7 = '7'; {один символ заключается в *апострофы*}

CharCr = #13; {задается символ с соответствующим номером по таблице *ASCII* в десятичной системе счисления}

Beta = **Chr**(187); {функция **Chr** преобразовывает целое значение аргумента в символ с соответствующим номером по таблице *ASCII*}

Cod = \$124; {задается символ с *соответствующим номером* по таблице *ASCII* в шестнадцатеричной системе счисления}

{Строковые константы }

Address = 'г. Томск, проспект Ленина, ТУСУР, кафедра КИПР';

Message = 'Out of memory';

ErrStr = 'Error: ' + **Message** + '.' + **CharCr**;

Alpha = '2.4'; {Это именно *строковая константа*, т.е. строка символов, которая изображает число "две целые четыре десятых", а не число 2,4!}

5.5.1 Типизированные именованные константы

Типизированные именованные константы представляют собой *переменные (!) с начальными значениями*, уже известными к началу выполнения программы .

В отличие от именованных констант, значение типизированных констант в программе *МОЖНО МЕНЯТЬ*.

Типизированные именованные константы также описывают в разделе объявления констант:

const

<идентификатор>: <тип> = <значение>;

Пример:

const

Oценка: byte = 4;

Predmet: string = 'Информатика';

Eps: real = 0.00001;

c: char = '@';

Error: boolean = true;

Maximum : integer = 1000;

5.6 Переменные

Переменная подобно ящичку, который можно заполнить *различными значениями*, которые можно хранить и читать.

Все переменные, используемые в программе, описываются в специальном **разделе описания переменных** после зарезервированного слова **Var**. С помощью описания пере-

Переменная — это именованная часть памяти, в которую могут помещаться разные значения переменной.

Переменная характеризуется именем, типом и переменным значением из множества допустимых значений данного типа.

В каждый момент времени переменная имеет единственное значение.

В процессе выполнения программы значение переменной может изменяться

менных компилятор получает информацию о том, сколько байт необходимо выделить для хранения данной переменной, диапазон допустимых значений, способ кодирования и набор допустимых операций.

В общем виде раздел объявления переменных выглядит так:

Var //Раздел объявления переменных

<список> : <тип>;

<список> ::= <имя>[,<список>]

где: **тип** - тип данных, для хранения которых предназначена переменная;

имя - имя переменной.

Следует обратить внимание, что компилятор языка **Lazarus** не различает прописные и строчные буквы в именах переменных, отчего имена **SUMMA**, **Summa** и **summa** обозначают одну и ту же переменную!

Разумно, чтобы имя переменной было логически связано с ее назначением!

Например, если в программе есть переменные, предназначенные для хранения суммы покупки и величины скидки, то этим переменным можно присвоить имена:

TotalSumm и **Discount** или **ObSumma** и **Skidka**.

В тексте программы объявление каждой переменной, как правило, помещают *на отдельной строке*. Это даёт возможность сделать комментарий к этой переменной: указать ее назначение, размерность, диапазон значений и т.п.

Если в программе имеется несколько переменных, относящихся к одному типу (список переменных), то имена этих переменных можно перечислить через запятую, а тип переменных указать после имени последней переменной через двоеточие.

Пример объявления переменных:

```

Var //Объявляем переменные
i, j, k: Integer; //Индексы массивов
Day = 1..31; //Дни месяца
Result, //Результат вычислений
SummaDay, //Сумма выручки за день, руб.
Power: Real; //Мощность, потребляемая РЭС, Вт
Flag, //Флаг события: True – свершилось, False - нет
Error: Boolean; //Ошибка: True – случилась, False - нет
  
```

5.7 Выражения

С помощью выражений задаются правила вычисления новых значений на основе ранее уже известных. Арифметические выражения в *Free Pascal* состояются из имён переменных и функций, констант, знаков операций и скобок. К моменту выполнения операций всем переменным, входящим в выражение, тем или иным способом должны быть заданы конкретные значения.

Ниже приведены (Таблица 5.2 **Ошибка! Источник ссылки не найден.**) примеры записи некоторых математических выражений.

Таблица 5.2 – Примеры записи арифметических выражений на *Free Pascal*

Математическая запись	Запись выражений
$\frac{xy}{z}$	$x * y / z$
$\frac{x}{yz}$	$x / (y * z)$ или $x / y / z$
$\frac{a^2 + \sqrt[3]{b}}{bc}$	$(sqr(a) + exp(ln(b) / 3)) / (b * c)$
$\frac{a_{i+1} + b_{i-1}}{2\pi y}$	$(a[i + 1] + b[i - 1]) / (2 * pi * y)$
$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$	$(-b + sqrt(sqr(b) - 4 * a * c)) / (2 * a)$
a^b	$exp(b * ln(a))$

5.7.1 Последовательность выполнения операций

Последовательность выполнения операций в выражениях определяется:

- приоритетом операций;
- порядком расположения операций в выражениях;
- использованием скобок.

Для задания нужного порядка выполнения операций в выражении можно использовать скобки, например:

$$\frac{r1 + r2 + r3}{r1 \cdot r2 \cdot r3} \Rightarrow (r1 + r2 + r3) / (r1 * r2 * r3)$$

Выражение, заключённое в скобки, трактуется как один операнд. Это означает, что операции над операндами в скобках будут выполняться в обычном порядке, но раньше, чем операции над операндами, находящимися за скобками.

При записи выражений, содержащих скобки, должна соблюдаться парность скобок, т.е. число открывающих скобок должно быть равно числу закрывающих скобок.

Нарушение парности скобок -

наиболее распространённая ошибка при записи выражений!!

Операции более высокого ранга (умножение и деление) выполняются раньше, чем операции более низкого ранга (сложение и вычитание). Операции одного ранга выполняются *слева направо*.

Наивысший приоритет имеют *унарные* (содержащие один операнд) операции:

- 1) **+** (сохранение знака)
- 2) **-** (отрицание знака)
- 3) **not** (логическое отрицание).

Например,

$$-7 \Rightarrow -7 \quad -(-6) \Rightarrow 6 \quad \text{not False} \Rightarrow \text{True}$$

Затем выполняются *бинарные* (содержащие два операнда) операции *типа умножения*:

- 4) ***** (умножение);
- 5) **/** (деление);
- 6) **div** (целочисленное деление);
- 7) **mod** (остаток от деления целых чисел);
- 8) **and** (логическое И);
- 9) **shl** (поразрядный сдвиг влево, операция над двоичным кодом);
- 10) **shr** (поразрядный сдвиг вправо, операция над двоичным кодом).

Например,

$$3.14 * 2 \Rightarrow 6.28 \quad 5 / 2 \Rightarrow 2.5 \quad 5 \text{ div } 2 \Rightarrow 2 \quad 23 \text{ mod } 5 \Rightarrow 3$$

Вслед за тем выполняются бинарные операции *типа сложения*:

- 11) **+** (сложение);
- 12) **-** (вычитание);
- 13) **or** (логическое ИЛИ, операция над двоичным кодом);
- 14) **xor** (логическое исключающее ИЛИ, операция над двоичным кодом).

Например,

$$5 + 7 \Rightarrow 9 \quad 5.37 - 2.15 \Rightarrow 3.22 \quad \text{True or False} \Rightarrow \text{True}$$

Самый низший приоритет имеют операции типа *бинарных отношений*:

- 15) = (равно);
- 16) <> (не равно);
- 17) > (больше);
- 18) < (меньше);
- 19) >= (больше или равно);
- 20) <= (меньше или равно).

Например,

$$2 = 2 \Rightarrow True \quad 2 <> 2 \Rightarrow False \quad 3 > 2 \Rightarrow True \quad 3 <= 4 \Rightarrow True$$

5.7.2 Результаты вычисления выражений

В следующей таблице приведены типы операндов и результатов для бинарных арифметических операций:

Таблица 5.3 - Бинарные арифметические операции

Операция	Действие	Типы операндов	Тип результата
+	Сложение	Целый Вещественный	Целый Вещественный
-	Вычитание	Целый Вещественный	Целый Вещественный
*	Умножение	Целый Вещественный	Целый Вещественный
/	Деление	Целый Вещественный	Вещественный Вещественный
div	Целочисленное деление	Целый	Целый
mod	Остаток от деления	Целый	Целый

Конкретный тип результата можно определить, руководствуясь следующими правилами:

- если оба операнда в операциях **+**, **-**, *****, **div** или **mod** являются операндами целого типа, то тип результата будет таким же, как общий тип обоих операндов;
- если один или более операндов в операциях **+**, **-**, или ***** имеют вещественный тип, то тип результата будет *вещественным*;
- если при использовании унарной операции сохранения знака или операции отрицания знака операнд имеет целый тип, то результат будет тоже целого типа. Если операнд вещественного типа, то тип результата будет вещественным или типом с повышенной точностью (*extended*).
- значение выражения **x/y** всегда будет вещественного типа (*real*) или с повышенной точностью (*extended*), независимо от типов операндов. Если **y** равно 0, то результат будет ошибочным;

- значение выражение $i \text{ div } j$ (деление нацело) представляет собой математическое частное от i/j , округлённое в меньшую сторону до значения целого типа. Если j равно 0 , результат будет ошибочным;
- операция **mod** возвращает остаток, полученный путём деления двух её операндов, то есть:

$$i \text{ mod } j = i - (i \text{ div } j) * j$$

Знак результата операции **mod** будет тем же, что и знак i . Если j равно нулю, то возникает ошибка.

5.7.3 Некоторые встроенные функции

Для выполнения часто встречающихся вычислений и преобразований язык **Lazarus** предоставляет программисту ряд стандартных подпрограмм-функций (Таблица 5.4).

Таблица 5.4 - Арифметические функции

Заголовок функции	Назначение	Тип результата	Примеры использования
Abs (X : <i>real/integer</i>): <i>real/integer</i>	Вычисление абсолютного значения аргумента X	Совпадает с типом X	Abs (-2.3) \Rightarrow 2.3 Abs (-157) \Rightarrow 157
ArcTan (X : <i>real</i>): <i>real</i>	Вычисление угла, тангенс которого равен X радиан	Вещественный	ArcTan (1) \Rightarrow \Rightarrow 7.854e-01
Cos (X : <i>real</i>): <i>real</i>	Вычисление косинуса X ; параметр X задает значение угла в радианах	Вещественный	Cos (Pi) \Rightarrow \Rightarrow -9.9999e-01
Exp (X : <i>real</i>): <i>real</i>	Вычисление экспоненты X , т.е. значение e^X ($e = 2.718282$ – основание натурального логарифма)	Вещественный	Exp (1) \Rightarrow \Rightarrow 2.71828e-01
Frac (X : <i>real</i>): <i>real</i>	Вычисление дробной части X	Вещественный	Frac (123.456) \Rightarrow \Rightarrow 0.456 Frac (-123.456) \Rightarrow \Rightarrow -0.456
Int (X : <i>real</i>): <i>real</i>	Вычисление целой части X	Вещественный	Int (123.456) \Rightarrow 123.0 Int (-123.456) \Rightarrow \Rightarrow -123.0
Ln (X : <i>real</i>): <i>real</i>	Вычисление натурального логарифма X (по основанию e)	Вещественный	Ln (1) \Rightarrow \Rightarrow 2.718282e+00
Pi : <i>real</i>	Возвращает значение числа π ($\pi = 3.1415926535897932385$)	Вещественный	Pi \Rightarrow \Rightarrow 3.141592653e+00
Sin (X : <i>real</i>): <i>real</i>	Вычисление синуса X ; параметр задаёт значение угла в радианах	Вещественный	Sin (Pi) \Rightarrow \Rightarrow 0.0000e+00

Заголовок функции	Назначение	Тип результата	Примеры использования
<i>Sqr(X)</i>	Вычисление X^2	Совпадает с типом X	<i>Sqr(5) ⇒ 25</i> <i>Sqr(2.5) ⇒ 6.25</i>
<i>Sqrt(X: real): real</i>	Вычисление \sqrt{X}	Вещественный	<i>Sqrt(2) ⇒</i> <i>⇒1.41421356</i>
<i>Round(X: real): longint</i>	Возвращает значение X , округлённое до ближайшего целого числа	Целый	<i>Round(1.4) ⇒ 1</i> <i>Round(1.5) ⇒ 2</i> <i>Round(-1.4) ⇒ -1</i> <i>Round(-1.5) ⇒ -2</i>
<i>Trunc(X: real): longint</i>	Возвращает ближайшее целое число, меньшее или равное X , если $X ≥ 0$, и большее или равное X , если $X < 0$.	Целый	<i>Trunc(1.4) ⇒ 1</i> <i>Trunc(1.5) ⇒ 1</i> <i>Trunc(-1.4) ⇒ -1</i> <i>Trunc(-1.5) ⇒ -1</i>
<i>Random(Range: word): word</i>	Генерирует значение целого случайного числа из диапазона $0 .. Range$	Целый	<i>Random(1000) ⇒ ⇒</i> <i>745</i> (случайное число)
<i>Random: real</i>	Генерирует значение случайного числа ⁵ из диапазона $0 .. 0.99$	Вещественный	<i>Random ⇒ 0.4876</i> (случайное число)

Значение функции связано с ее именем. Поэтому функцию можно использовать в качестве операнда выражения, например в инструкции присваивания. Так, чтобы вычислить квадратный корень, достаточно записать $k := Sqrt(X)$, где *Sqrt* - функция вычисления квадратного корня, X — переменная, которая содержит число, квадратный корень которого надо вычислить.

Функция характеризуется типом значения и типом параметров. Тип переменной, которой присваивается значение функции, должен соответствовать типу функции. Точно так же тип фактического параметра функции, т.е. параметра, который указывается при обращении к функции, должен соответствовать типу формального параметра. Если это не так, компилятор выводит сообщение об ошибке.

5.7.4 Функции модуля Math

Для расширения математических возможностей программ во *Free Pascal* имеется подключаемый модуль *Math*. Перед началом работы следует подключить этот модуль в разделе *Uses*.

Модуль *Math* содержит следующие математические функции (Таблица 5.5).

⁵ Для инициализации генератора случайных чисел случайным значением можно предварительно воспользоваться процедурой *Randomize*.

Таблица 5.5 - Функции дополнительного модуля Math

Функция	Описание
<i>ArcSin(X)</i>	Арксинус аргумента X (-1 .. 1) (Возвращает значение в радианах)
<i>ArcCos(X)</i>	Арккосинус аргумента X (-1 .. 1) (Возвращает значение в радианах)
<i>Tan(X)</i>	Тангенс аргумента (угол X в радианах)
<i>Cotan(X)</i>	Котангенс угла (угол X в радианах)
<i>Log10(X)</i>	Вычисление десятичного логарифма
<i>Log2(X)</i>	Вычисление двоичного логарифма числа X
<i>LogN(y, x)</i>	Вычисление логарифма числа x по основанию y
<i>Max(a, b)</i>	Возвращает максимальное число из двух чисел
<i>Min(a, b)</i>	Возвращает минимальное число из двух чисел
<i>Power(X, N)</i>	Возведение числа X в произвольную степень N
<i>Hypot(X: Y)</i>	Вычисление гипотенузы по длине катетов
<i>RadToDeg(X)</i>	Преобразование радиан в градусы
<i>DegToRad(X)</i>	Преобразование градусов в радианы
<i>Floor</i>	Округление в меньшую сторону
<i>Ceil</i>	Округление в большую сторону

5.8 Операторы языка

Минимальная конструкция языка, представляющая собой законченную мысль, называется **предложением**.

В языках программирования предложение, задающее полное описание некоторого выполняемого действия, называется **оператором**.

Оператор - это составная часть программы, фраза алгоритмического языка, предписывающая определённый порядок преобразования информации.

Любая программа содержит операторы. Операторы образуют программу так же, как обычные предложения образуют текст книги.

В **простых операторах** можно присваивать значение, активизировать процедуру или функцию или *передать управление* на любой выполняемый оператор.

Структурные операторы могут быть *составными* и содержать несколько операторов, операторы *цикла* или *условные операторы*, управляющие логикой программы.

5.8.1 Оператор присваивания

Оператор присваивания является основной вычислительной инструкцией. Если в программе надо выполнить вычисление, то нужно использовать оператор присваивания, имеющий вид:

имя := выражение;

где:

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

:= (заменить, присвоить) - символ присваивания;

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

Оператор присваивания выполняется следующим образом:

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

2) Затем вычисленное значение записывается в переменную, имя которой стоит слева от символа присваивания⁶.

Примеры.

$i := 0;$ {значение переменной i становится равным нулю}

$a := b + c;$ {значением переменной a будет число, равное сумме значений переменных b и c }

$i := i + 1;$ {увеличение текущего значения переменной i на единицу}.

Выражение должно быть *совместимо по присваиванию с типом переменной* (желательно одного типа). 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 записаны неправильно}

$d := i + 1;$ {операторы записаны правильно}

5.8.2 Операторы ввода/вывода

Во время исполнения программа может выводить информацию на экран, печать или внешнюю память, вводить информацию с клавиатуры или диска. Для этого используются операторы ввода и вывода.

Операторы ввода могут иметь вид:

read(список переменных ввода);

или

readln(список переменных ввода);

Имена переменных ввода могут быть любого числового, символьного или строкового типа. Число имён переменных может быть любым (в том числе и пустым).

Отличие работы оператора **Readln** от **Read** заключается в том, что после выполнения **Readln** осуществляется пропуск до начала следующей строки исходных данных.

⁶ Старое значение переменной слева от символа присваивания при этом будет утеряно.

Операторы вывода могут иметь вид:

write(список элементов вывода); //после вывода курсор остается на этой же строке
или

writeln(список элементов вывода); //после вывода переводит курсор на новую строку
где:

- **<список элементов вывода> ::= < элемент вывода >[,<список элементов вывода>]**
- **< элемент вывода > ::= Expr [: MinField [: DecDigits]]**
- **Expr** – выводимое выражение символьного, целого, вещественного строкового или булевого типа.

Необязательный параметр, позволяющий отформатировать любой элемент вывода:

- **MinField** – выражение целого типа, задающее минимальную ширину поля вывода, которая должна быть больше нуля.

Необязательный параметр, позволяющий отформатировать выводимое число вещественного типа:

- **DecDigits** - выражение целого типа, задающее число *десятичных знаков, выводимых на экран после десятичной точки*. **DecDigits** указывается только для **Expr** вещественного типа, если указан параметр **MinField**.

- Если **DecDigits** указывается, то число выводится в формате с фиксированной точкой, а если не указывается, то в формате с плавающей точкой.

Формат вывода с фиксированной точкой:

- [**<пробелы>**] [-] **<цифры>** [**<цифры дробной части>**]

Формат вывода с плавающей точкой:

- [-] **<цифра>** [**<цифры дробной части>**] **E** [+|-]**<показатель степени>**]

Примеры.

Пусть объявлены две типизированные константы (переменные с начальными значениями):

Const

I: integer = 12345; {объявлена константа целого типа}

R: real = -123.1234567; {объявлена константа вещественного типа}

Печать без форматирования⁷:

Writeln(I, R);

1	2	3	4	5	-	1	.	2	3	1	2	3	4	5	6	7	0	0	0	0	E	+	0	0	0	2
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Как здесь разобраться с результатами?



⁷ Клеточка обозначает позицию на экране или печатающем устройстве - знакоместо для вывода одного символа.

Форматированная печать (по десять позиций для вывода каждого числа; для вещественного числа отведено три позиции для дробной части):

Writeln(I: 10, R: 10: 3);

					1	2	3	4	5			-	1	2	3	.	1	2	3
--	--	--	--	--	---	---	---	---	---	--	--	---	---	---	---	---	---	---	---

Вывод вещественного числа в плавающем формате (отведено 11 позиций, число знаков в дробной части не указано):

Writeln(R: 11);

-	1	.	2	3	E	+	0	0	0	2
---	---	---	---	---	---	---	---	---	---	---

5.9 Работа в редакторе исходного текста *Free Pascal*

Редактор *Free Pascal* обладает возможностями, характерными для большинства текстовых редакторов. С его помощью можно создавать и редактировать тексты программ. Кроме этого, редактор обладает возможностями подсветки синтаксиса, а также рядом других удобств.

Текст в редакторе можно выделять, копировать, вырезать, вставлять. Кроме того, в редакторе можно осуществлять поиск заданного фрагмента текста, выполнять вставку и замену.

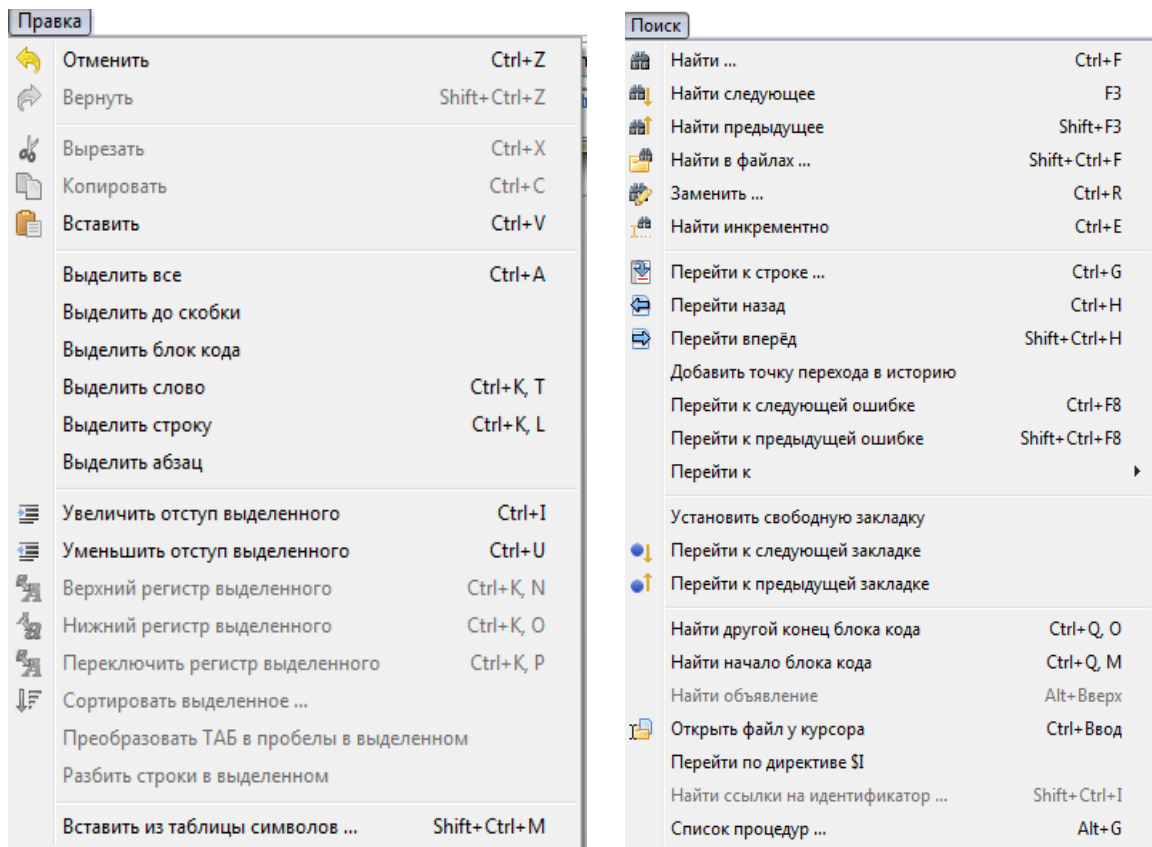


Рис. 5.3 - Меню **Правка** и **Поиск** Редактора исходного текста

Все допустимые операции в редакторе собраны в меню **Правка** и **Поиск** главного меню Lazarus (Рис. 5.3). Там же приведены и «Горячие клавиши», соответствующие клавишам меню.

Для доступа к меню можно:

- 1) использовать для выбора пункта меню мышь;
- 2) нажать **F10**, чтобы переключить фокус на меню. Затем можно использовать клавиши со стрелками для навигации по меню. Для выбора пункта меню используется клавиша Enter.

Контекстное меню можно вызвать щелчком правой кнопки мыши (Рис. 5.4).

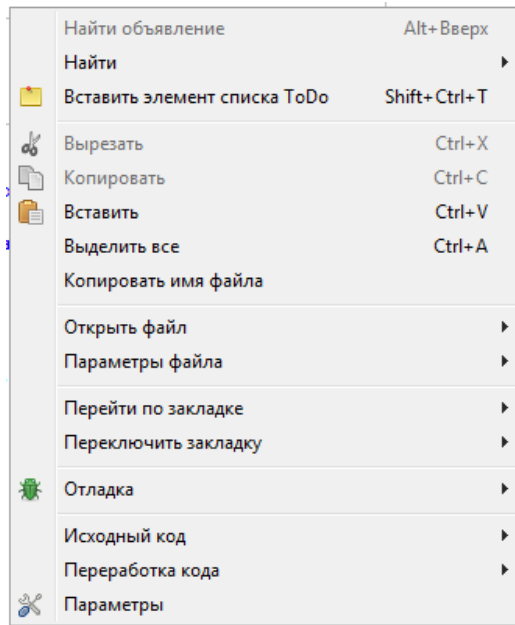


Рис. 5.4 - Контекстное меню Редактора

Для выхода из меню без выполнения каких-либо действий следует нажать клавишу Esc.

6 Индивидуальные задания

6.1.1 Варианты заданий

- 1) Для своего варианта (Таблица 6.1) написать консольную программу на языке *Free Pascal* для вычисления функции, значение аргумента x для которой вводят с клавиатуры.
- 2) Определить порядок выполнения операций в формуле, пронумеровав их.
- 3) Произвести тестирование программы, созданной в консольной версии *Free Pascal* при заданных значениях констант и двух значениях переменной x .

Таблица 6.1 - Варианты индивидуальных заданий

№ варианта	Формула	Константы		Переменная	
		a	b	x_1	x_2
1.	$Y := \sqrt[3]{\frac{\lg(a+x)}{b^{a+x} + b}} + e^{5 \cdot a}$	6	10	0.5	0
2.	$Y := \frac{\sin(\ln(a + b \cdot x)) + (b + x)^3}{a \cdot x^2 + b \cdot x - 1}$	1.7	0.5	1.5	1.8
3.	$Y := a - \frac{\lg(a+x) \cdot x^b}{\cos(\pi \cdot x)}$	3.0	-0.5	1.3	2.0
4.	$Y := \frac{b^5 - \operatorname{tg} x}{\ln x}$		1.5	0.3	2.0
5.	$Y := \frac{\left(\frac{a}{b} - 3^{a \cdot x}\right) \cdot \cos x}{\sqrt{x+2}}$	1.5	-100	0.8	1.3
6.	$Y := \frac{(a + \sqrt[3]{x+3}) \cdot (x+b)}{\ln b}$	1.0	2.0	-0.5	0.5
7.	$Y := \frac{b \cdot e^{(x-b)}}{\operatorname{tg}(5 \cdot x)} + \frac{a}{b}$	π	5.0	6.0	6.5
8.	$Y := 3 \cdot \frac{x^b - a^x}{b^2 + a^x}$	$\pi/2$	-0.9	1.0	2.0
9.	$Y := \frac{b \cdot \ln(a+x)}{\sqrt{a^b} - \sin b}$	$\pi/3$	4.0	-3.0	3.0
10.	$Y := \frac{\sin^2 x - a^x}{\sqrt{\pi \cdot x - 4} - a}$	$2\pi/3$		2.0	2.5
11.	$Y := \frac{\sqrt[3]{x} + \frac{1}{a}}{\lg\left(\left \sqrt[3]{x} + \frac{1}{a}\right \right)}$	π		1.5	3.5
12.	$Y := \frac{\frac{1}{\sqrt{ x+1 +a}} + \frac{1}{a}}{\operatorname{arctg}(a-x)}$	4.0		-6.0	-5.0
13.	$Y := \frac{\lg^2(x+a) - x+a }{\cos^2(x+a)}$	5.0		10.0	0
14.	$Y := \frac{\frac{1}{\cos x} + \frac{1}{\cos^3(x^2)} + b}{e^{2 \cdot b \cdot x}}$	0.5	-3.0	0.9	1.2

№ варианта	Формула	Константы		Переменная	
		a	b	x_1	x_2
15.	$Y := \frac{\sin(a \cdot x) + x^{10} }{\cos(b \cdot x) + \frac{1}{5}}$	$-\pi$	π	1.5	1.7
16.	$Y := \frac{\ln(x^{2.5} + a) - \sqrt{\lg(x^2)}}{\operatorname{arctg} \frac{x}{b}}$	14.0	200.0	10.0	5.0
17.	$Y := \frac{e^{0.01 \cdot a} - \cos(b \cdot x) \cdot x }{\lg^5 x + \frac{1}{x}}$	50.0	$\pi/3$	5.5	-4.0
18.	$Y := \frac{e^{\sin(a \cdot x)} + \cos(b \cdot x)}{5 \cdot \cos^2(a \cdot x) - a} + \frac{1}{x}$	π	2π	0.8	0.6
19.	$Y := \frac{\lg x + x }{a \cdot x^2} \cdot b + x^{2.5}$	-2.0	3.0	4.0	3.0
20.	$Y := \left \frac{\cos(2 \cdot x)}{a + \sin x} \right + x^2 \cdot \sqrt{ \sin x }$	-1.0	2.3	8.0	5.0
21.	$Y := \frac{x^3}{\cos(a \cdot x) + \sin(b \cdot x)} + x \cdot e^x $	$-\pi$	π	5.0	4.0
22.	$Y := \sin(x) \cdot a + \frac{\lg(b+x)}{\sqrt{b}}$	-1.0	1.5	0	1.0
23.	$Y := \frac{\frac{\operatorname{tg} x}{\ln x} + \frac{\lg x}{a}}{ b \cdot x }$	2	-2.5	2.0	3.0
24.	$Y := \operatorname{tg}(x) \cdot \frac{\lg(b+x)}{\sqrt{a \cdot x + b}}$	10.0	-9.0	0	2.2
25.	$Y := \frac{\sqrt{(a \cdot x)^2 + b}}{a^{0.1 \cdot b} - \lg(b \cdot x)}$	5.0	6.0	-4.0	3.0
26.	$Y := \frac{x^b \cdot \cos x}{\sqrt{b-x} + a}$	-1.5	3.0	0.5	1.0
27.	$Y := \frac{\operatorname{arctg}(b \cdot x) + x }{\sqrt{b-x} + a}$	-3.0	5π	1.8	2.0
28.	$Y := \frac{\frac{1}{a+x} + e^{ \sin x }}{\lg(b-x)}$	-8.0	7.0	0.1	0.3
29.	$Y := \frac{\operatorname{tg}(a+x) - \frac{1}{\lg(a \cdot x)}}{\sqrt[5]{b \cdot x}}$	1.0	2.0	3.0	4.0

№ варианта	Формула	Константы		Переменная	
		<i>a</i>	<i>b</i>	<i>x</i> ₁	<i>x</i> ₂
30.	$Y: = \frac{a \cdot b - e^{a \cdot x} + \cos x}{\lg(x) + 1}$	1.3	2.5	2.2	1.3

6.1.2 Рекомендации по составлению программы:

1) Составление программы целесообразно начать с создания файла консольной программы и сохранения её в предварительно подготовленном каталоге проекта. 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.

2) Так как исходные данные, промежуточные и окончательные результаты арифметических вычислений обыкновенно содержат и дробную часть, то разумно использовать данные вещественного типа.

3) Для возведения в степень ***b*** неотрицательного и не равного нулю выражения ***a*** (не существует логарифмов от отрицательных чисел) во *Free Pascal* можно воспользоваться следующим преобразованием:

$$a^b \Rightarrow \exp(b \cdot \ln(a))$$

4) Извлечь корень степени ***n*** из числа ***a*** (при ***a* > 0**) означает возвести число ***a*** в степень $1/n$. Для извлечения корня ***n*** из неотрицательного и не равного нулю выражения ***a*** можно воспользоваться во *Free Pascal* следующим преобразованием:

$$\sqrt[n]{a} \Rightarrow \exp(\ln(a)/n)$$

5) Выразить десятичный логарифм через натуральный можно преобразованием:

$$\lg(a) \Rightarrow \ln(a)/\ln(10)$$

6.1.3 Пример разработки программы

```

Program Ivanov; //Заголовок программы;
{Назначение программы – Освоение работы с Free Pascal в консольном режиме}
{Автор - Иванов И.И., студент группы 236-1}
{Вариант задания - N° 77}
{Условия задания – Рассчитать по формуле значение функции Y = f((x))}
{$mode objfpc}{$H+}
uses //Используемые модули
{$IFDEF UNIX}{$IFDEF UseCThreads}
{$ENDIF}{$ENDIF}
Classes
{ you can add units after this };

```

```

{Разделы описаний данных}
Const      //Используемые константы:
  a = 2.5;
  b = Pi;

Var        //Используемые переменные:
  Y,       //Результат
  x: real; //Аргумент функции

begin {Начало основного блока программы Ivanov}
  {Вывод на экран заголовка программы}
  writeln('Расчёт функции Y: = sqrt(abs(sin(x)/(a - b)))');
  writeln; //Пропуск строки

  write('Введите аргумент функции x = '); //Вывод на экран запроса
  Readln(x);                               //Ввод с клавиатуры аргумента функции
  Y := sqrt(abs(sin(x)/(a - b)));           //Расчёт по формуле

  writeln; //Пропуск строки
  writeln('Результат Y =', Y: 12: 5);

  writeln; //Пропуск строки
  writeln('Для завершения программы нажмите Enter...');
  Readln

end. {Конец основного блока программы Ivanov}

```

Внимание!

По умолчанию в среде Lazarus используется кодировка UTF-8. Однако консольные приложения в ОС Windows используют старую кодировку MS DOS - CP866. Чтобы в консольных приложениях, создаваемых с помощью среды разработки Lazarus, правильно выводились русские символы, нужно:

- 1) щёлкнуть правой кнопкой мыши в окне редактора исходного кода
- 2) выбрать **Параметры файла** ⇒ **Кодировка** ⇒ **CP866**
- 3) в появившемся окне нажать кнопку с надписью **изменить файл**, после чего не забыть сохранить изменённый файл.

7 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. 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.