МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
БОРИСОГЛЕБСКИЙ ФИЛИАЛ
(БФ ФГБОУ ВО «ВГУ»)

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО УЧЕБНОЙ ДИСЦИПЛИНЕ
Программирование

Методические указания для обучающихся по освоению дисциплины

Приступая к изучению учебной дисциплины, прежде всего обучающиеся должны ознакомиться с учебной программой дисциплины. Электронный вариант рабочей программы размещён на сайте БФ ВГУ.

Знание основных положений, отраженных в рабочей программе дисциплины, поможет обучающимся ориентироваться в изучаемом курсе, осознавать место и роль изучаемой дисциплины в подготовке будущего педагога, строить свою работу в соответствии с требованиями, заложенными в программе.

Основными формами контактной работы по дисциплине являются лекции, практические и лабораторные занятия, посещение которых обязательно для всех студентов (кроме студентов, обучающихся по индивидуальному плану).

В ходе лекционных занятий следует не только слушать излагаемый материал и кратко его конспектировать, но очень важно участвовать в анализе примеров, предлагаемых преподавателем, в рассмотрении и решении проблемных вопросов, выносимых на обсуждение. Необходимо критически осмысливать предлагаемый материал, задавать вопросы как уточняющего характера, помогающие уяснить отдельные излагаемые положения, так и вопросы продуктивного типа, направленные на расширение и углубление сведений по изучаемой теме, на выявление недостаточно освещенных вопросов, слабых мест в аргументации и т.п.

Не следует дословно записывать лекцию, лучше попытаться понять логику изложения и выделить наиболее важные положения лекции в виде опорного конспекта. Рекомендуется использовать различные формы выделения наиболее сложного, нового, непонятного материала, который требует дополнительной проработки: можно пометить его знаком вопроса (или записать на полях сам вопрос), цветом, размером букв и т.п. – это поможет быстро найти материал, вызвавший трудности, и в конце лекции (или сразу же, попутно) задать вопрос преподавателю (не следует оставлять непонятый материал без дополнительной проработки, без него иногда бывает невозможно понять последующие темы). Материал уже знакомый или понятный нуждается в меньшей детализации – это поможет сэкономить усилия во время конспектирования.

На практически занятиях рекомендуется активно участвовать в анализе решаемых задач, обсуждении алгоритма их решения, выборе способов реализации алгоритма на языке программирования. При возникновении затруднений в решении задач важно сразу выяснить все непонятные моменты, задав вопрос преподавателю.

Большое значение при изучении программирования имеет самостоятельная работа.

В ходе выполнения лабораторных работ рекомендуется пользоваться конспектами лекций и записями с практических занятий. При необходимости, за справочной информацией по языку программирования рекомендуется обращаться к встроенной справке среды разработки или к онлайн-справочникам. Важно при решении задач придерживаться правил стилевого оформления кода: это сделает код более «читаемым» и поможет в его анализе (и поиске ошибок при необходимости).

При подготовке к промежуточной аттестации необходимо повторить пройденный материал в соответствии с учебной программой, примерным перечнем вопросов, выносящихся на зачет. Рекомендуется использовать конспекты лекций и источники, перечисленные в списке литературы в рабочей программе дисциплины, а также ресурсы электронно-библиотечных систем. Необходимо обратить особое внимание на темы учебных занятий, пропущенных по разным причинам. При необходимости можно обратиться за консультацией и методической помощью к преподавателю.

План лекционных занятий

Тема №1. Технологии создания программного продукта. Алгоритмы.

- Основные этапы решения задач на ЭВМ.
- Понятие алгоритма, его свойства. Способы описания алгоритмов.

- Понятие языка программирования. Эволюция языков программирования, их классификация.
- Понятие системы программирования. Технологический процесс создания программы, компиляция программы.

Тема №2. Основы языка программирования Pascal.

- Язык программирования Pascal. Структура программы.
- Типы данных: простые и структурные. Структура программы. Оператор присваивания.
- Операторы: простые, структурированные. Условный оператор. Оператор выбора. Организация циклических структур.
- Структурированный тип – массив. Сортировка массивов. Типовые задачи. Строки. Тип-файл.

Тема №3. Процедурное программирование.

- Подпрограммы: основные понятия, фактические и формальные параметры.
- Функции. Формат записи. Примеры использования.
- Процедуры. Формат записи. Примеры использования.
- Рекурсия.
- Использование подпрограмм в решении математических задач.

Тема №4. Модульное программирование. Графика.

- Модульное программирование. Структура модуля.
- Подключение модулей к программе. Использование библиотек подпрограмм.
- Графика в Pascal. Графические операторы.
- Простейшая анимация.

Тема №5. Динамические структуры данных.

- Понятие динамических структур данных.
- Указатели. Объявление. Подпрограммы для работы с указателями.
- Линейный односвязный список: просмотр, добавление, удаление элементов.
- Стек. Очередь. Бинарные деревья.
- Использование динамических структур данных при решении задач: проверка правильности расстановки скобок, перевод выражения в постфиксную форму.

Тема №6. Объектно-ориентированное программирование.

- Объектно-ориентированная парадигма программирования. Объекты, полиморфизм и наследование. Объектно-ориентированное проектирование.
- Визуальные среды программирования. Конструирование программ на основе иерархии объектов. Сообщения. Обработка исключительных ситуаций.

Методические материалы по теме «Основы языка программирования Pascal»

Паскаль (англ. Pascal) — язык программирования общего назначения, один из наиболее известных языков программирования, является базой для ряда других языков. В настоящее время используется главным образом для обучения программированию.

Язык Паскаль был создан Никлаусом Виртом в 1968–69 годах и назван в честь выдающегося французского математика и философа Блеза Паскаля (1623–1662).

Первая версия языка была опубликована Виртом в 1970 году. Автор задумывал Паскаль как небольшой и эффективный язык со строгой типизацией и средствами структурного (процедурного) программирования, который должен способствовать «дисциплинированному» программированию и прививать хороший стиль программирования.

За время существования языка Паскаль у него появилось несколько диалектов.

Мы будем рассматривать одну из наиболее популярных реализаций языка Паскаль: Турбо Паскаль (англ. Turbo Pascal) от фирмы Borland. Турбо Паскаль — интегрированная среда разработки и язык программирования в этой среде. Последняя версия Турбо Паскаля (7.1) была выпущена в 1994 году и с тех пор его разработка и

поддержка прекращена (в дальнейшем разработчики сфокусировались на поддержке «наследников» Турбо Паскаля — Object Pascal и Delphi).
Следует отметить, что при этом другие диалекты Паскаля продолжали развиваться. Например, весьма популярна одна из свободных реализаций языка Паскаль — Free Pascal. Free Pascal имеет реализации под различные платформы, а кроме того обеспечивает режимы совместимости с другими распространёнными диалектами языка.

Основы языка Паскаль

Как известно, описание любого алгоритмического языка содержит описание используемых в нём символов, элементарных конструкций, выражений и операторов (заметим, что перечисленные элементы составляют иерархическую структуру).
Опишем кратко названные элементы для языка Турбо Паскаль.
Символы языка — это основные неделимые знаки, в терминах которых пишутся все тексты на языке.
Описание символов заключается в перечислении допустимых символов языка (*алфавит* языка). Паскаль включает следующий набор основных символов: латинские прописные и строчные буквы, пробел, знак подчёркивания, цифры, знаки операций, ограничители (скобки, точка, запятая и т.п.), спецификаторы (^, #, $) и служебные (зарезервированные) слова (var, for, if и др.).
Элементарные конструкции — это минимальные единицы языка, имеющие самостоятельный смысл. Они образуются из основных символов языка.
Под описанием элементарных конструкций понимают правила их образования из символов языка. Элементарные конструкции языка Паскаль включают в себя имена, числа и строки.
Имя (*идентификатор*) — это последовательность букв и цифр, начинающаяся с буквы, называющая элементы языка — константы, метки, типы, переменные, процедуры, функции, модули, объекты. Не разрешается использовать в качестве имен служебные слова и стандартные имена.
Числа обычно записываются в десятичной системе счисления. Они могут быть целыми и действительными. Действительные числа записываются либо в форме с десятичной точкой, либо в форме с «плавающей» точкой (в этом случае для отделения десятичного порядка используется буква E).
Строка — последовательность символов, записанная между апострофами.
Выражение в алгоритмическом языке состоит из элементарных конструкций и символов, оно задаёт правило вычисления некоторого значения.
Описание выражений — это правила образования любых выражений, имеющих смысл в данном языке.
Выражение состоит из констант, переменных, указателей функций, знаков операций (арифметических, логических, отношений, унарных) и скобок и задает правило вычисления некоторого значения. Порядок вычисления определяется приоритетом содержащихся в нем операций.
Оператор задаёт полное описание некоторого действия, которое необходимо выполнить. Для описания сложного действия может потребоваться группа операторов. В этом случае операторы объединяются в *составной оператор* или *блок* (для этого используют *операторные скобки* — пару служебных слов begin и end).
Объединенная единым алгоритмом совокупность описаний и операторов образует *программу* на алгоритмическом языке.

Структура программы

Программа на языке Паскаль состоит из *заголовка*, *раздела описаний* и *раздела операторов* (*тела программы*):

```
program Name;
  {Раздел описаний}
begin
  {Тело программы}
```

```
end.
```

Заголовок программы содержит имя программы (в примере Name), задаваемое пользователем. Отметим, что в большинстве диалектов Паскаля заголовок программы является необязательным.

Раздел описания может включать в себя раздел подключаемых *модулей* (uses), раздел описания *меток* (label), раздел описания *типов* (type), раздел описания *констант* (const), раздел описания *переменных* (var), а также описание пользовательских *процедур* и *функций*. При этом отдельные разделы описаний могут отсутствовать.

Раздел операторов представляет собой составной оператор, который содержит между служебными словами begin и end последовательность операторов. Операторы отделяются друг от друга точкой с запятой.

Текст программы заканчивается точкой.

Кроме описаний и операторов программа может содержать *комментарии* — произвольные последовательности символов, расположенные между скобками {}. Заметим, что также в фигурных скобках в Паскале указываются *директивы компилятора*.

Типы данных

В языке Паскаль тип явно задается в описании переменной или функции, которое предшествует их использованию. При этом в Паскале тип данных определяет множество значений переменной (или выражения, функции), а каждая операция или функция требует аргументов фиксированного типа и выдаёт результат фиксированного типа.

Тип данных в языке Паскаль определяет:

- возможные значения переменных, констант, функций, выражений, принадлежащих к данному типу;
- внутреннюю форму представления данных в памяти компьютера;
- операции и функции, которые могут выполняться над величинами, принадлежащими к данному типу.

В Турбо Паскале имеется разветвлённая структура типов данных:

Кроме того, в Турбо Паскале существует возможность создания пользовательских типов.

В следующей таблице представлены простые типы Турбо Паскаля.

Тип данных	**Диапазон значений**
Целые типы	
Integer	–32 768..32 767
Byte	0..255
Word	0..65 535

ShortInt	–128..127
LongInt	–2 147 483 648..2 147 483 647
Вещественные типы	
Real	$2.9\times10^{-39}..1.7\times10^{38}$
Single	$1.5\times10^{-45}..3.4\times10^{38}$
Double	$5\times10^{-324}..1.7\times10^{308}$
Extended	$3.4\times10^{-4932}..1.1\times10^{4932}$
Comp	$-9.2\times10^{18}..9.2\times10^{18}$
Логический тип	
Boolean	True, False
Символьный тип	
Char	все символы ASCII

Пример описания переменных простых типов:

```
var
  I: Integer;
  R: Real;
```

Структурированные типы имеют четыре разновидности: массивы, множества, записи и файлы. Эти типы подробнее рассмотрены соответствующих разделах. Также в отдельных разделах рассматриваются вопросы о работе с указателями.

К данным различных типов в Паскале можно применять операции приведения типов. Приведение типов позволяет рассматривать одну и ту же величину как принадлежащую разным типам.

Заметим, что для задания начальных значений переменным (инициализации переменных) в Турбо Паскале существует возможность присваивать начальные значения переменным одновременно с их описанием. Такое объявление размещается в разделе описания констант:

```
const
  R: Real = 0.4;
```

Основные операторы Турбо Паскаля

Самым простым действием над переменной является занесение в неё величины соответствующего типа. Эта операция называется присваиванием.

Оператор присваивания — один из самых простых и наиболее часто используемых операторов в любом языке программирования. Он предназначен для вычисления нового значения некоторой переменной, а также для определения значения, возвращаемого функцией.

В языке Паскаль оператор присваивания записывается как := (в отличие от операции сравнения =), например:

```
I := A + B;
```

Оператор выполняется следующим образом: вычисляется значение выражения в правой части присваивания; после этого переменная, указанная в левой части, получает вычисленное значение. Тип переменной и тип выражения должны совпадать кроме случая, когда выражение относится к целому типу, а переменная — к действительному. При этом происходит приведение значения выражения к действительному типу.

Кроме присваивания часто используемыми и необходимыми для любого языка программирования являются операции ввода и вывода.

В Паскале для ввода и вывода данных используются стандартные *процедуры ввода и вывода* Read (Readln) и Write (Writeln).

Особенностью процедур ввода/вывода является переменное число параметров (может быть передан список ввода/вывода через запятую, а может использоваться вызов и вовсе без параметров). Кроме того, первым параметром в эти процедуры может быть передана файловая переменная, связанная с файлом для ввода/вывода. Если этот

параметр опущен, по умолчанию вывод осуществляется на экран, а ввод данных происходит с клавиатуры.
Варианты процедур Readln и Writeln после ввода/вывода данных осуществляют переход на следующую строку.
Переменные, образующие список ввода, могут принадлежать к целому, вещественному или символьному типу.
Переменные, составляющие список вывода, могут относиться к целому, действительному, символьному, строковому или булевскому типу. В качестве элемента списка вывода кроме имен переменных (констант) могут использоваться также выражения и строки.
Заметим, что для величин действительного типа может осуществляться так называемый форматный вывод: элемент списка вывода может иметь вид A:K:M, где A — переменная или выражение действительного типа, K — ширина поля вывода, M — число цифр дробной части выводимого значения.
При организации ввода-вывода данных хорошим тоном считается вывод на экран комментариев, из которых ясен смысл вводимых и выводимых значений.
Далее рассмотрим так называемые *операторы управления потоком*.
Обычно операторы в программе выполняются в том порядке, в каком они записаны. Однако существуют специальные операторы, позволяющие изменять последовательность выполнения операторов.
Оператор безусловного перехода (goto) прерывает естественный порядок выполнения программы и указывает, что дальнейшее выполнение должно продолжаться, начиная с оператора, помеченного меткой, указанной в операторе перехода. Используемая метка должна быть предварительно объявлена в разделе объявления меток (label).
Заметим, что использование оператора безусловного перехода среди программистов считается дурным тоном, так как противоречит принципам структурного программирования, и его использования нужно избегать. Всегда существует возможность обойтись в программе без него и почти всегда это нужно делать.
Оператор условного перехода (*условный оператор*, *оператор ветвления*) — является одной из основных алгоритмических структур.
Заметим, что алгоритм решения любой задачи можно реализовать на основе трёх базовых конструкций: следование, ветвление, цикл.
Оператор условного перехода в Турбо Паскале имеет вид:
```
if Условие then Оператор_1 else Оператор_2;
```
Здесь Условие — это логическое выражение, в зависимости от значения которого выбирается одна из двух альтернативных ветвей алгоритма. Если значение условия истинно (True), то будет выполняться Оператор_1, записанный после ключевого слова then. В противном случае будет выполнен Оператор_2, следующий за словом else, при этом Оператор_1 пропускается.
Отметим, что перед ключевым словом else точка с запятой никогда не ставится.
У условного оператора существует *неполная форма*, в которой ветка else в операторе if отсутствует:
```
if Условие then Оператор_1;
```
Тогда в случае невыполнения логического условия управление сразу передается оператору, стоящему в программе после конструкции if.
В программах на языке Паскаль условия представляют собой выражения, значением которых является величина логического (Boolean) типа. Это может быть, как просто переменная указанного типа, так и сложная последовательность высказываний, связанных логическими операциями.
В программировании часто возникают ситуации, когда приходится осуществлять выбор одного из нескольких (более двух) альтернативных путей выполнения программы. Несмотря на то, что такой выбор можно организовать с помощью нескольких условных операторов, удобнее воспользоваться специальным *оператором выбора* (переключателем):

```
case Выражение of
  Значение_1: Оператор_1;
  ...
  Значение_n: Оператор_n;
  else Оператор;
end;
```

После ключевого слова case записывается Выражение, которое называется *селектором*. Оно может быть любого перечисляемого типа. Далее через двоеточие записываются пары Значений выражения и Операторов, которые должны выполняться, когда Выражение принимает эти значения. Как и в условном операторе «ветка» else является необязательной.

Операторы цикла (с параметром, с пред- и постусловием) и их принципы работы. Вложенные операторы цикла

В большинстве практических задач, решаемых с помощью программирования, необходимо производить многократное выполнение некоторого действия. Такой многократно повторяющийся участок вычислительного процесса называется *циклом*.

В Турбо Паскале существует три вида циклов.

Цикл с параметром (цикл типа «для»).

```
for Переменная := Значение_1 to Значение_2 do Оператор;
```

или

```
for Переменная := Значение_1 downto Значение_2 do Оператор;
```

Оператор for вызывает Оператор (который может быть составным и называется *телом* цикла), находящийся после слова do, по одному разу для каждого значения в диапазоне от Значения_1 до Значения_2.

Переменная цикла, начальное и конечное значения должны иметь порядковый тип. В первом варианте записи цикла (со словом to), значение переменной цикла увеличивается на 1 при каждой итерации цикла. Во втором вариант записи (со словом downto) — значение переменной цикла уменьшается на 1 при каждой итерации цикла. Не следует вручную изменять значение управляющей переменной внутри цикла с параметром.

Цикл с параметром используется тогда, когда известно необходимое количество выполнений тела цикла. Такой цикл называют *арифметическим*.

Если же количество повторений заранее неизвестно, то цикл называют *итерационным*. В Турбо Паскале существуют две разновидности таких циклов: с предусловием и с постусловием.

Цикл с предусловием (цикл типа «пока»).

```
while Условие do Оператор;
```

Тело цикла — Оператор (который может быть и составным) — будет выполняться пока истинно (True) Условие цикла, представляющее собой логическое выражение. Как только Условие принимает значение «ложь» (False), осуществляется выход из цикла. Если Условие ложно изначально, то тело цикла не будет выполнено ни разу.

Цикл с постусловием (цикл типа «до»).

```
repeat
  Оператор_1;
  ...
  Оператор_n;
until Условие;
```

Тело цикла — операторы между словами repeat и until — повторяются, пока Условие, представляющее собой логическое выражение, является ложным (False). Как только логическое выражение становится истинным (True), происходит выход из цикла.

Так как Условие проверяется после выполнения операторов, то в цикле с постусловием в любом случае операторы выполнятся хотя бы один раз, в отличие от цикла с предусловием.

Заметим, что цикл с предусловием является универсальным, то есть любая задача, требующая использования цикла, может быть решена с применением этой структуры.

Цикл с постусловием и цикл с параметром созданы для удобства программирования некоторых типов задач.
В циклах с предусловием и с постусловием программисту необходимо следить, чтобы вовремя сработало условие выхода из цикла, иначе цикл станет бесконечным и программа «зависнет».
Пример. Приведём решение следующей элементарной задачи с помощью трёх видов цикла: *Найти сумму всех натуральных чисел от 1 до 100*.
1) Решение с помощью цикла с предусловием.
```
program Ex1;
var
  I, S: Integer;
begin
  I := 1;
  S := 0;
  while I <= 100 do
  begin
    S := S + I;
    I := I + 1;
  end;
  Writeln('Сумма: ', S);
end.
```
2) Решение с помощью цикла с постусловием.
```
program Ex2;
var
  I, S: Integer;
begin
  I := 1;
  S := 0;
  repeat
    S := S + I;
    I := I + 1;
  until I > 100;
  Writeln('Сумма: ', S);
end.
```
3) Решение с помощью цикла с параметром.
```
program Ex3;
var
  I, S: Integer;
begin
  S := 0;
  for I := 1 to 100 do
    S := S + I;
  Writeln('Сумма: ', S);
end.
```
Часто при решении практических задач используются *вложенные циклы* — когда телом цикла является циклическая структура. При этом *внешний* и *внутренний* циклы могут быть разных видов. При использовании вложенных циклов необходимо для каждого из них использовать свою переменную цикла, а также следить, чтобы все операторы внутреннего цикла полностью располагались в теле внешнего цикла.
Типичный пример использования вложенных циклов — работа с двумерными (и многомерными) массивами.
В заключение отметим, что для всех операторов цикла выход из цикла может осуществляется как вследствие естественного окончания оператора цикла, так и с помощью *операторов перехода и выхода*: Continue и Break.
Процедура Break выполняет безусловный выход из цикла. Процедура Continue обеспечивает переход к началу новой итерации цикла.

Структурированный тип данных массив. Формат описания и обращения. Примеры задач на массивы

Массив — упорядоченная (пронумерованная) последовательность данных одного типа, объединенных под одним именем. При этом каждый компонент массива может быть явно обозначен и к нему имеется прямой доступ. В Турбо Паскале существуют только *статические массивы*, а это значит, что число элементов массива определяется при его описании и в дальнейшем не меняется.

Проще всего представить себе массив в виде таблицы (матрицы), где каждая величина находится в собственной ячейке. Положение ячейки в таблице однозначно определяется набором координат (индексов). Самой простой является линейная таблица (вектор — строка или столбец), в которой для указания на элемент данных достаточно одного индекса.

Описание массива выглядит так:

```
var Имя: array [Список_индексов] of Тип_элементов;
```

В качестве индексов могут выступать переменные порядковых типов (чаще всего — тип-диапазон). При указании диапазона начальный индекс не должен превышать конечный. Тип элементов массива может быть любым.

Пример описания *линейного* (*одномерного*) массива:

```
var
  Mass: array [1..10] of Real;
```

Единственное действие, которое можно произвести с массивом целиком — присваивание. Однако присваивать можно только массивы одинаковых типов. Даже массивы, описанные одинаково, но в разных строках описания переменных, не могут быть присвоены один другому целиком.

Вместе с тем, над массивами не определены операции отношения. Сравнивать, а также вводить и выводить массивы можно только поэлементно. Для обращения к элементу массива указывают имя массива и индекс элемента в квадратных скобках. Например, обратиться к третьему элементу массива из примера выше можно с помощью записи Mass[3].

Пример ввода элементов одномерного массива с клавиатуры и вывода его на экран:

```
program Ex1;
var
  Mass: array [1..10] of Integer;
  I: Byte;
begin
  Writeln('Ввод элементов массива:');
  for I := 1 to 10 do
    Readln(Mass[I]);
  Writeln('Вывод элементов массива:');
  for I := 1 to 10 do
    Write(Mass[I], ' ');
end.
```

Кроме ввода элементов списка с клавиатуры, также часто используется заполнение массива случайными числами или вычисление значений элементов с помощью математических формул.

Существует несколько стандартных (наиболее часто используемых) операций над массивами:

- поиск значений;
- сортировка элементов в порядке возрастания или убывания;
- подсчет элементов в массиве, удовлетворяющих заданному условию.

Базовые задачи решаются элементарно: сумму элементов массива можно подсчитать с помощью оператора S := S + A[I], первоначально задав S := 0; количество элементов массива — с использованием оператора K := K + 1, первоначально задав K := 0; произведение — оператором P := P * A[I], первоначально задав P := 1.

Рассмотрим несколько более сложную задачу — сортировку массива по возрастанию (неубыванию).
Представленный ниже код производит сортировку в порядке возрастания введенного с клавиатуры массива одним из простейших способов — *выбором минимального*. Идея этого метода состоит в следующем: пусть часть массива (по K-й элемент включительно) отсортирована; нужно найти в неотсортированной части массива минимальный элемент и поменять местами с (K+1)-м.

```
program Ex2;
var
  N, I, J, K, Temp: Integer;
  A: array [1..30] of Integer;
begin
  Write('Введите количество элементов (N<=30): ');
  Readln(N);
  for I := 1 to N do
  begin
    Write('A[', I, '] = ');
    Readln(A[I]);
  end;
  for I := 1 to N - 1 do
  begin
    K := I;
    for J := I + 1 to N do
      if A[J] <= A[K] then
        K := J;
    Temp := A[I];
    A[I] := A[K];
    A[K] := Temp;
  end;
  for I := 1 to N do
    Write(A[I], ' ');
end.
```

При решении практических задач часто приходится иметь дело с различными таблицами данных, содержащими более одной строки (столбца). В таком случае для работы с данными используют *двумерные массивы.* Положение элемента в таком массиве определяется двумя индексами: номером строки и номером столбца. При записи обращения к элементу массива на языке Паскаль индексы разделяются запятой.
Работа с двумерными (и многомерными) массивами почти всегда связана с организацией вложенных циклов.
Пример объявления двумерного массива, заполнения его данными с клавиатуры и вывода элементов на экран:

```
program Ex3;
var
  A: array [1..10, 1..5] of Integer;
  I, J: Integer;
begin
  for I := 1 to 10 do
    for J := 1 to 5 do
    begin
      Write('A[', I, ', ', J, '] = ');
      Readln(A[I, J]);
    end;
  for I := 1 to 10 do
  begin
    for J := 1 to 5 do
```

```
      Write(A[I, J], ' ');
    Writeln;
  end;
end.
```

Кроме двумерных, при написании программ на языке Турбо Паскаль можно использовать массивы и большей размерности: трёхмерные, четырёхмерные и т.д. Принципы объявления и использования массивов при этом не изменяются, просто увеличивается количество индексов. В общем случае N-мерный массив характеризуется N индексами.

Методические материалы по теме «Процедурное программирование» Алгоритм решения любой достаточно сложной задачи можно упростить путём разложения исходной задачи на отдельные подзадачи, и реализации для их решения соответствующих подалгоритмов. Особенно удобно использование подалгоритмов, когда какой-либо подалгоритм неоднократно повторяется в программе или имеется возможность использовать некоторые фрагменты уже разработанных ранее алгоритмов.

В языке Паскаль, как и в большинстве языков программирования, предусмотрены средства, позволяющие оформлять вспомогательный алгоритм как подпрограмму. Кроме того, подпрограммы применяются для разбиения крупных программ на отдельные смысловые части в соответствии с модульным принципом в программировании.

Процедуры и функции

Подпрограмма — это последовательность операторов, которые определены и записаны только в одном месте программы, однако их можно вызвать для выполнения из одной или нескольких точек программы. Каждая подпрограмма определяется уникальным именем. В языке Паскаль существуют два типа подпрограмм — *процедуры* и *функции*.

Процедура и функция — это именованная последовательность описаний и операторов. Функция отличается от процедуры тем, что она должна обязательно выработать значение определенного типа (и вернуть это значение в качестве результата).

Процедуры и функции, используемые в программе, должны быть описаны в разделе описаний программы до первого их упоминания. Вызов процедуры или функции производится по их имени.

Структура описания процедур и функций похожа на структуру Паскаль-программы: у них также имеются заголовок, раздел описаний и исполняемая часть. Раздел описаний содержит те же подразделы, что и раздел описаний программы: описания констант, типов, переменных и т.д. Исполняемая часть содержит собственно операторы процедур и функций.

Формат описания процедуры:

```
procedure Имя_процедуры(Параметры: Тип);
  {Раздел описаний процедуры}
begin
  {Тело процедуры}
end;
```

Формат описания функции:

```
function Имя_функции(Параметры: Тип): Тип;
  {Раздел описаний функции}
begin
  {Тело функции}
end;
```

Подпрограммы в языке Паскаль могут иметь *параметры* (значения, передаваемые в процедуру или функцию в качестве аргументов). При описании указываются *формальные параметры* (имена, под которыми будут фигурировать передаваемые данные внутри подпрограммы) и их типы.

При вызове подпрограммы вместе с её именем должны быть заданы все необходимые параметры в том порядке, в котором они находятся в описании. Значения, указываемые при вызове подпрограммы, называются *фактическими параметрами*.

Одна и та же подпрограмма может вызываться неоднократно, выполняя одни и те же действия с разными наборами входных данных.

В теле функции обязательно должна быть хотя бы команда присвоения такого вида:

```
Имя_функции := Выражение;
```

Указанное выражение задаёт возвращаемый функцией результат и должно приводить к значению того же типа, что и тип результата функции, описанный выше.

Вызов процедуры производится оператором, имеющим следующий формат:

```
Имя_процедуры(Список_фактических_параметров);
```

Список фактических параметров — это их перечисление через запятую. При вызове фактические параметры как бы подставляются вместо формальных, стоящих на тех же местах в заголовке. Типы фактических параметров должны быть такими же, что и у соответствующих им формальных.

Вызов функции производится аналогичным образом, однако, как правило, он должен входить в некое выражение. При вычислении значения такого выражения функция будет вызвана, действия, находящиеся в её теле, будут выполнены, в выражение будет подставлено значение результата функции.

Итак, при вызове процедур и функций необходимо соблюдать следующие правила:

- количество фактических параметров должно совпадать с количеством формальных;
- соответствующие фактические и формальные параметры должны совпадать по порядку следования и по типу.

В языке Турбо Паскаль передача параметров может производиться двумя способами: *по значению* и *по ссылке*. Параметры, предаваемые по ссылке, отличаются тем, что в заголовке процедуры (функции) перед ними ставится служебное слово var.

При первом способе (передача по значению) значения фактических параметров копируются в соответствующие формальные параметры. При изменении этих значений в ходе выполнения процедуры (функции) исходные данные (фактические параметры) измениться не могут. При этом в качестве фактических параметров можно использовать и константы, и переменные, и выражения.

При втором способе (передача по ссылке) все изменения, происходящие в теле процедуры (функции) с формальными параметрами, приводят к немедленным аналогичным изменениям соответствующих им фактических параметров. Изменения происходят с переменными вызывающего блока. При вызове соответствующие фактические параметры могут быть только переменными.

Обратим внимание ещё на один важный момент: имена, описанные в заголовке или разделе описаний процедуры или функции, называются *локальными* для этого блока. Имена, описанные в блоке, соответствующем всей программе, называют *глобальными*. Следует помнить, что формальные параметры процедур и функций всегда являются локальными переменными для соответствующих блоков.

Локальные имена доступны (считаются известными, «видимыми») только внутри того блока, где они описаны. Сам этот блок, и все другие, вложенные в него, называют областью видимости для этих локальных имен. Имена, описанные в одном блоке, могут совпадать с именами из других, как содержащих данный блок, так и вложенных в него.

Глобальные имена хранятся в области памяти, называемой сегментом данных программы. Они создаются на этапе компиляции и действительны на все время работы программы.

Рекомендуется все имена, которые имеют в подпрограммах чисто внутреннее, вспомогательное назначение, делать локальными. Это предохраняет от изменений глобальные объекты с такими же именами.

Рассмотрим использование процедур и функций на примере программы поиска максимума из двух целых чисел.
В первом примере будем использовать процедуру, которая печатает на экран максимальное из двух переданных в неё значений.
```
program Ex1;
var
  X, Y: Integer;

procedure PrintMaxNumber(A, B: Integer);
begin
  if A > B then
    Writeln(A)
  else
    Writeln(B);
end;

begin
  Write('Введите X и Y: ');
  Readln(X, Y);
  Write('Максимальное число: ');
  PrintMaxNumber(X, Y);
end.
```
Во втором примере аналогичную задачу решим с использованием функции, которая возвращает максимальное из двух переданных в неё значений:
```
program Ex2;
var
  X, Y: Integer;

function FindMaxNumber(A, B: Integer): Integer;
begin
  if A > B then
    FindMaxNumber := A
  else
    FindMaxNumber := B;
end;

begin
  Write('Введите X и Y: ');
  Readln(X, Y);
  Writeln('Максимальное число: ', FindMaxNumber(X, Y));
end.
```
Дополнительные задания для самостоятельной работы

Реализовать с использованием типа запись базу данных в соответствии с вариантом (необходимо сохранять данные в файл и читать из файла):

1. Биржа труда. База безработных содержит следующую информацию: анкетные данные, профессия, образование, место и должность последней работы, причина увольнения, семейное положение, контактная информация. Обеспечить поиск безработных с высшим образованием и вычислить процент таковых от общего числа безработных.
2. Человек. Информация о человеке содержит ФИО, пол, национальность, рост, вес, дата рождения (год, месяц, число), номер телефона, домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира). Обеспечить сортировку списка людей по фамилии в алфавитном порядке.
3. Справочник банков. Справочник по каждому банку содержит наименование, статус (форма собственности), условия хранения средств на лицевом счете (годовые проценты на различных видах вкладов). Обеспечить выбор банка с наибольшим процентом для заданного вида вклада.

4. Справочник туриста. Справочник содержит список турагентств и предлагаемых ими услуг (страна, город или маршрут круиза, условия проживания и проезда, экскурсионное обслуживание, сервис принимающей стороны, стоимость путевки). Привести список турагентств, предлагающих туры в страны еврозоны.
5. Справочник любителя живописи. Справочник содержит список художников с анкетными данными и стилями, список картин со ссылкой на художников, датой создания, жанром. Оригинал картины находится в музее или в частной коллекции. Определить процент картин определенного художника, находящихся в частной коллекции.
6. Справочник астронома. Для каждой из зарегистрированных звезд известны название, созвездие, видимая звездная величина, расстояние, координаты на небосклоне. Обеспечить поиск звезд и созвездий в заданной точке земного шара в заданное время.
7. Владелец автомобиля. Каждый владелец автомобиля регистрирует транспортное средство в ГИБДД. База владельцев содержит ФИО, дату рождения владельца, регистрационный номер автомобиля, номер технического паспорта, телефон, отделение регистрации ГИБДД. Определить список владельцев, автомобили которых зарегистрированы в определенном ГИБДД.
8. Касса автовокзала. Расписание автобусов: номер рейса, конечный и промежуточные пункты, время отправления, количество свободных мест на каждом рейсе. Выбрать ближайший рейс до заданного пункта (при наличии свободных мест). Замечание: пункт может являться как конечным, так и промежуточным пунктом автобуса.

Методические рекомендации к самостоятельной работе по теме «Динамические структуры данных. Указатели»

Основные понятия и определения темы

Все переменные, объявленные в программе, размещаются в одной непрерывной области оперативной памяти, которая называется *сегментом данных*. Длина сегмента данных составляет 65 536 байт, что может вызвать известные затруднения при обработке больших массивов данных. С другой стороны, объем памяти ПК достаточен для успешного решения задач с большой размерностью данных. Выходом из положения может служить использование так называемой динамической памяти.

Динамическая память — это оперативная память ПК, предоставляемая программе при ее работе, за вычетом сегмента данных (64 Кбайт), стека (обычно 16 Кбайт) и собственно тела программы. Размер динамической памяти можно варьировать в широких пределах.

Динамическое размещение данных означает использование динамической памяти непосредственно при работе программы. В отличие от этого, *статическое размещение* осуществляется компилятором Turbo Pascal в процессе компиляции программы. При динамическом размещении заранее не известны ни тип, ни количество размещаемых данных, к ним нельзя обращаться по именам, как к статическим переменным.

Оперативная память ПК представляет собой совокупность элементарных ячеек для хранения информации — байтов, каждый из которых имеет собственный номер. Эти номера называются *адресами*, они позволяют обращаться к любому байту памяти.

Turbo Pascal предоставляет в распоряжение программиста гибкое средство управления динамической памятью — так называемые указатели.

Указатель — это переменная, которая в качестве своего значения содержит адрес байта памяти.

Как правило, в Turbo Pascal указатель связывается с некоторым типом данных. Такие указатели будем называть *типизированными*. Для объявления типизированного указателя используется значок ^, который помещается перед соответствующим типом, например:

```
var
     p1: ^Integer;
     p2: ^Real;
```

Кроме того, в Turbo Pascal можно объявлять указатель и не связывать его при этом с каким-либо конкретным типом данных. Для этого служит стандартный тип Pointer, например:

```
var
     p: Pointer;
```

Указатели такого рода будем называть *нетипизированными*. Поскольку нетипизированные указатели не связаны с конкретным типом, с их помощью удобно динамически размещать данные, структура и тип которых меняются в ходе работы программы.

Вся динамическая память в Турбо Паскале рассматривается как сплошной массив байтов, который называется *кучей*. Физически куча располагается в старших адресах сразу за областью памяти, которую занимает тело программы.

Память под любую динамически размещаемую переменную выделяется процедурой New. Параметром обращения к этой процедуре является типизированный указатель. В результате обращения указатель приобретает значение, соответствующее динамическому адресу, начиная с которого можно разместить данные, например:

```
New(i);
```

После того как указатель приобрел некоторое значение, т. е. стал указывать на конкретный физический байт памяти, по этому адресу можно разместить любое значение соответствующего типа. Для этого сразу за указателем без каких-либо пробелов ставится значок ^, например:

```
i^: = 2; {В область памяти по адресу i помещено значение 2}
```

Таким образом, значение, на которое указывает указатель, т. е. собственно данные, размещенные в куче, обозначаются значком ^, который ставится сразу за указателем. Если за указателем нет значка ^ то имеется в виду адрес, по которому размещены данные.

Динамическую память можно не только забирать из кучи, но и возвращать обратно. Для этого используется процедура Dispose. Например, следующий оператор вернёт в кучу память, которая ранее была выделена указателю i:

```
Dispose(i);
```

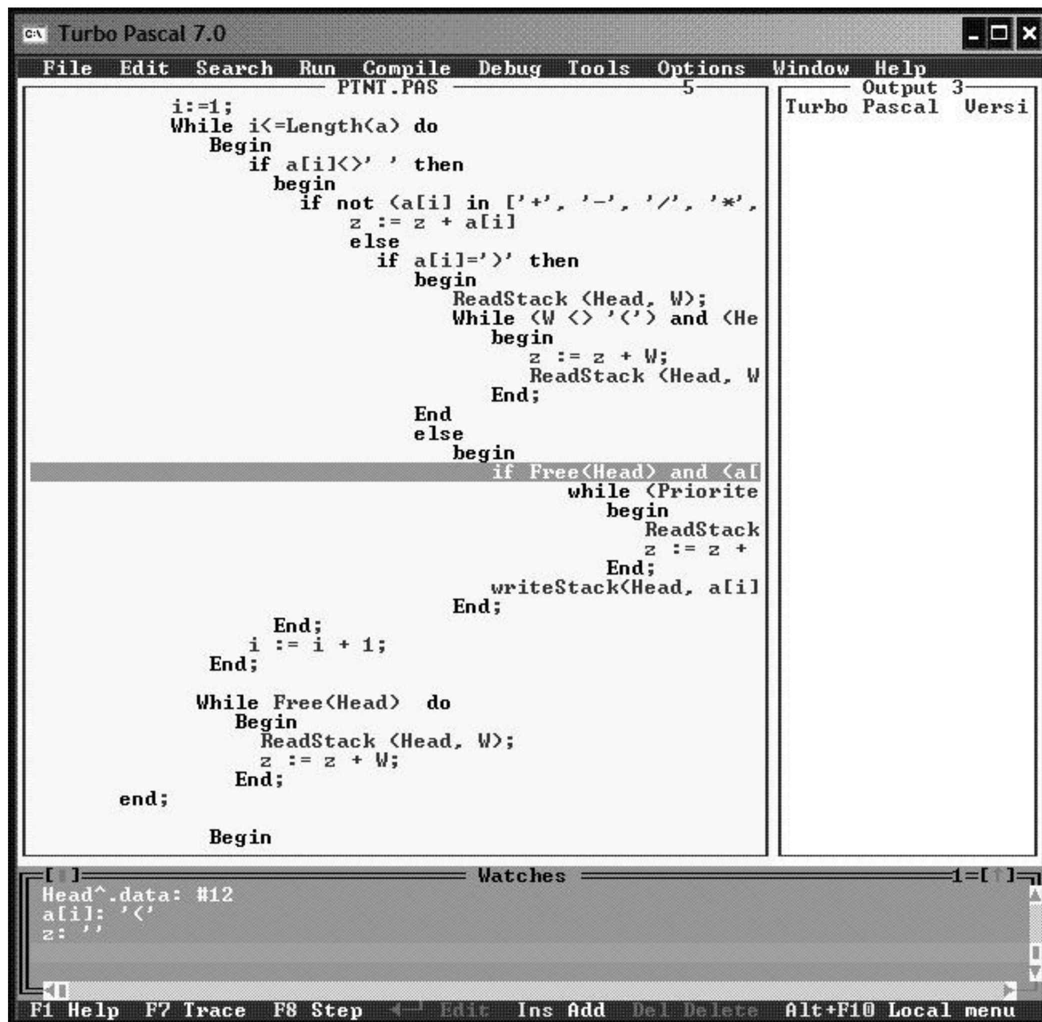
Освободившийся указатель программист может пометить зарезервированным словом nil. Если указатель содержит значение предопределенной константы nil, то такой указатель называют *пустым*, то есть он не указывает ни на какую переменную.

Настройка среды Turbo Pascal

Для отладки программы удобно использовать пошаговое выполнение совместно с контролированием текущего значения переменных в окне Watch. Это позволяет видеть, какое значение принимает та или иная переменная в каждый момент выполнения программы.

Рекомендуется настроить среду Turbo Pascal 7.0 так, как это описано ниже.

Чтобы вызвать окно Watch выполните **Debug→Watch** (пункт **Debug** в главном меню, далее команду **Watch**). Также удобно при отладке иметь перед глазами окно Output. Для его вызова выполните **Debug→Output**. Расположите окна как показано ниже



Для перемещения и изменения размеров любого окна сначала выделите его, после чего выполните **Window→Size/Move** (или нажмите сочетание клавиш *Ctrl+F5*). Теперь для перемещения окна используйте стрелки на клавиатуре, а для изменения его размеров — те же стрелки при нажатой клавише *Shift*.

После того, как программа будет набрана, можно приступать к отладке.

Для этого, прежде всего, необходимо выбрать те переменные, наблюдение за значением которых в процессе выполнения программы поможет определить, правильно ли она работает и, при необходимости, найти и устранить ошибку. Определившись с перечнем необходимых переменных, нужно добавить эти переменные в окно наблюдения Watch. После этого программа запускается на выполнение в пошаговом режиме.

Чтобы добавить в окно Watch переменные, за значением которых необходимо наблюдать, выполните **Debug→Add Watch…** (или нажмите сочетание клавиш *Ctrl+F7*), и введите в окно запроса имя переменной. Как правило, для списков имеет смысл наблюдать за изменением значения поля data текущего элемента списка. Так, если в программе осуществляется «проход» по элементам списка p с помощью переприсваивания p:=p^.next в цикле, то в окно Watch целесообразно добавить значение p^.data.

Для запуска программы на выполнение в отладочном режиме используйте клавиши:

- F8 — для пошагового выполнения программы без захода в подпрограммы (процедуры и функции);
- F7 — для пошагового выполнения программы с заходом в процедуры;
- F4 — для выполнения программы до строки, на которой расположен курсор.

При запуске на выполнение программы Turbo Pascal может выдавать два типа ошибок: *ошибки периода компиляции* и *ошибки периода выполнения*.

Динамические списки

Под *списком* обычно понимают конечный упорядоченный набор объектов произвольных размера и природы.

Связанные списки используются в двух основных случаях:

- при создании в оперативной памяти набора данных, размер которого заранее неизвестен.
- в базах данных. Связанный список позволяет быстро выполнять вставку и удаление элемента данных без реорганизации всего дискового файла.

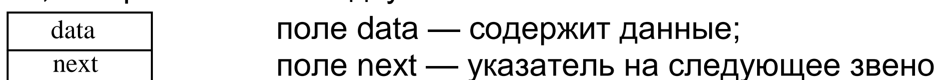
Связанные списки могут иметь одиночные или двойные связи.

Список с одной связью (*однонаправленный список*) содержит элементы, каждый из которых имеет связь со следующим элементом данных.

В списке с двойной связью (*двунаправленный список*) каждый элемент имеет связь, как со следующим элементом, так и с предыдущим элементом.

Линейный (однонаправленный) список является динамической структурой данных переменного размера, данные в которую могут включаться и извлекаться в произвольно выбранном месте.

Строить список удобно при помощи определенной структуры данных, состоящей из динамических переменных. Каждый элемент (также используются термины звено и узел) списка будем представлять записью языка Pascal, схематично показанной на рисунке ниже, которая состоит из двух полей: data и next.



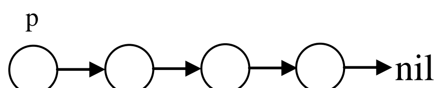
Описание каждого узла списка будет выглядеть следующим образом:

```
type PNode = ^Node;
     Node = record
          data: Integer;
          next: PNode;
     end;
```

Таким образом, список представляет собой последовательность узлов, в поле next каждого из которых хранится указатель на следующий узел списка или на nil, если данный узел является последним.

Тогда схематически однонаправленный список можно изобразить так:

Для удобства можно использовать упрощённое схематическое изображение списка, показанное ниже.



В данном примере указатель на первый узел списка хранится в переменной p.

13. Методические материалы для обучающихся по теме «Визуальные среды программирования. Основы работы в интегрированной среде разработки Lazarus»

Lazarus — открытая среда разработки программного обеспечения (интегрированная среда разработки: ИСР или IDE от англ. Integrated Development Environment) на языке Object Pascal для бесплатно распространяемого компилятора Free Pascal. Lazarus относится к так называемым инструментам быстрой разработки (Rapid Application Development — RAD) и сочетает визуальное проектирование и объектно-ориентированное программирование.

При разработке Lazarus большое внимание было уделено совместимости с коммерческой средой разработки Delphi: Lasarus основан на библиотеке визуальных компонентов Lazarus Component Library (LCL), хорошо совместимой с библиотекой визуальных компонентов Delphi (VCL).

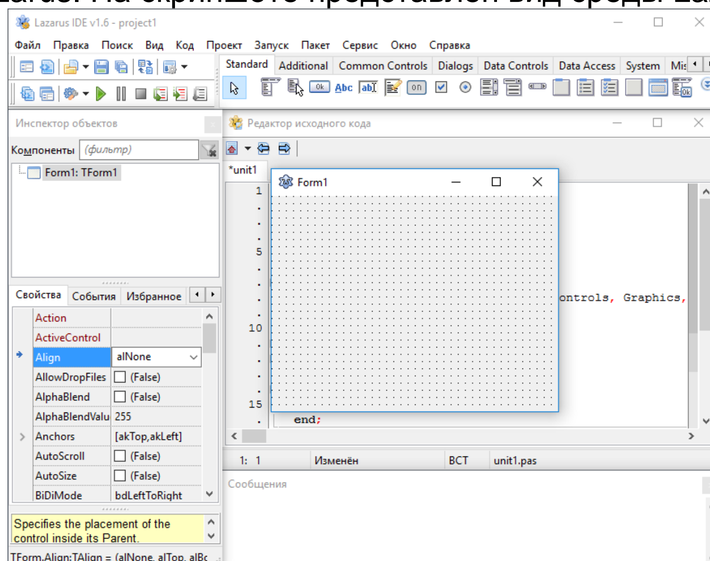
Интерфейс Lazarus внешне очень похож на «классический» интерфейс среды Delphi, но при этом среда Lazarus включает достаточно мощный редактор кода, по своим возможностям близкий к другим современным средам программирования.

К преимуществам Lazarus относится поддержка (на уровне компилятора) множества типов синтаксиса языка Pascal: Object Pascal, Turbo Pascal, Mac Pascal, Delphi, а также кроссплатформенность: программы могут быть скомпилированы для операционных систем семейства Linux, Microsoft Windows (Win32, Win64), Mac OS X, FreeBSD, WinCE, OS/2.

Lazarus — система с открытым исходным кодом, распространяемая на условиях GNU General Public License.

Начало работы

Запустите среду Lazarus. На скриншоте представлен вид среды Lazarus.



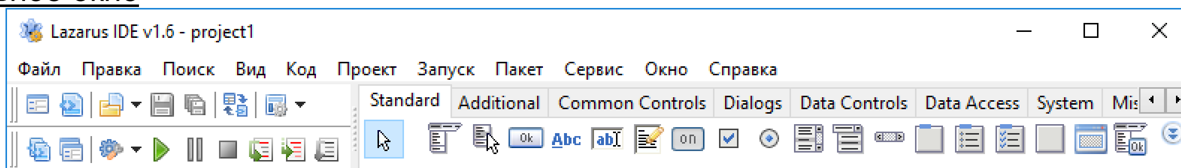
Обратите внимание, что вместо одного, на экране появились пять окон:

главное окно — Lazarus IDE vXX – project1 (заголовок окна включает название и версию среды разработки, а также имя проекта — до первого сохранения это будет автоматически назначаемое имя по умолчанию);

- окно стартовой формы — Form1;
- окно редактора свойств объектов — Инспектор объектов (Object Inspector), включающий древовидное отображение списка объектов и вкладки со свойствами (Properties) выбранного объекта и событиями (Events), с ним связанными;
- окно Редактора исходного кода (Lazarus Source Editor), в котором открыта одна вкладка с текущим редактируемым модулем unit1;
- окно Сообщений.

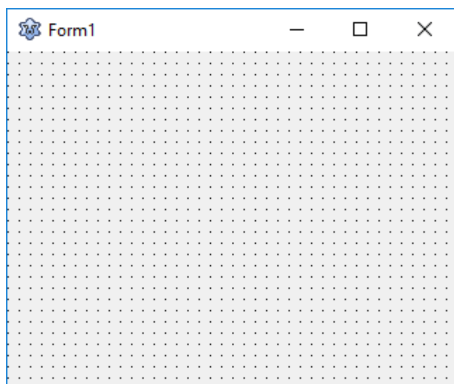
Окно редактора кода частично перекрыто окном стартовой формы.

Главное окно

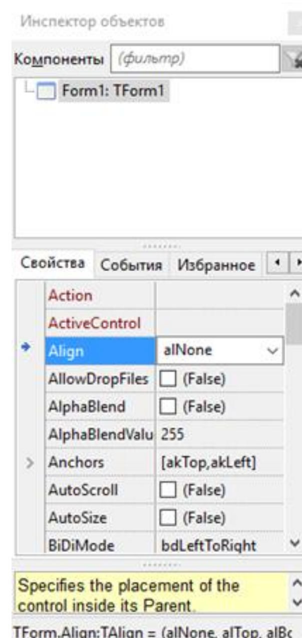


В главном окне находится главное меню, панели инструментов и палитра компонентов. В палитре компонентов отображаются компоненты, с помощью которых разработчик создаёт свои приложения. Пиктограммы стандартных компонентов Lazarus разделены на группы, и каждая группа расположена на отдельной вкладке (Standard, Additional и др.).

Проектировщик форм



В окне проектировщика форм отображается форма (заготовка окна будущего приложения) как визуальный объект. Здесь программист определяет, как будет выглядеть приложение с точки зрения пользователя, и создаёт графический интерфейс пользователя. Для этого выбираются компоненты из палитры компонентов и перетаскиваются на форму. Управление внешним видом и поведением каждого компонента осуществляется с помощью Инспектора объектов и Редактора исходного кода.



По умолчанию для каждого оконного приложения в Lazarus создаётся заготовка главного окна разрабатываемого приложения — стартовая форма (Form1).

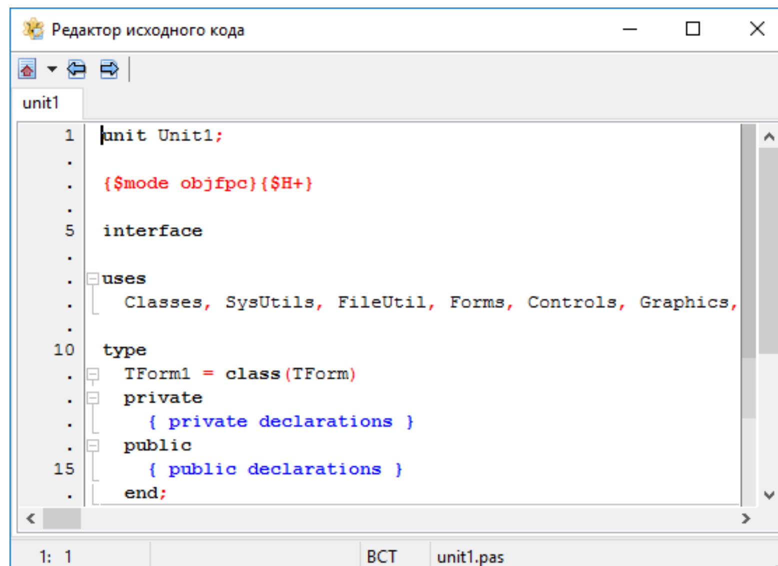
В режиме редактирования на заготовке формы отображается точечная сетка, предназначенная для облегчения процесса размещения визуальных компонентов на форме.

Инспектор объектов (Object Inspector)

С помощью объектного инспектора можно изменять свойства компонентов формы и определять события, на которые должна реагировать форма или её компоненты.

Окно инспектора объектов имеет две вкладки: «Свойства» (Properties), с помощью которой можно настроить внешний вид и другие параметры компонентов, и «События» (Events), позволяющую определить «реакцию» компонента на действия пользователя и другие события — *обработчики событий*. Именно написание обработчиков событий и составляет основу программирования в визуальной среде.

Редактор исходного кода (Lazarus Source Editor)



Окно редактора кода предназначено для ввода кода разрабатываемого приложения, а также для редактирования кода, сгенерированного средой Lazarus автоматически. Обратите внимание, что такого автоматически сгенерированного кода будет довольно много.

Сгенерированный средой код не следует изменять без необходимости!

В начале работы над новым проектом окно редактора кода содержит сформированный средой Lazarus шаблон программы. В дальнейшем в код автоматически будут добавляться заготовки обработчиков событий, для которых необходимо будет написать код. Кроме того, программист может определять собственные подпрограммы, объявлять переменный, константы и т.д., если это необходимо.

Редактор кода Lasarus поддерживает подсветку исходного кода, а также включает достаточно большое число инструментов, облегчающих написание кода: автозавершение, подсказка параметров, шаблоны кода, и др.

Обработка событий

События — это свойства процедурного типа, предназначенные для создания пользовательской реакции на те или иные входные воздействия.

Имена всех событий в Lazarus принято начинать с «On». Двойной щелчок в Инспекторе объектов на странице События в поле напротив названия любого события позволяет получить в программе заготовку метода — обработчика этого события. При этом его имя будет состоять из имени текущего компонента и имени события (без «On»), а относиться он будет к текущей форме. Например, если на форме (Form1) размещена кнопка (компонент типа TButton с именем Button1), то заготовка обработчика события OnClick (щелчок по кнопке) будет иметь вид:

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

end;
```

Заготовку обработчика события OnClick можно также получить, выполнив двойной щелчок по размещенной на форме кнопке.

Форма в Lazarus — это синоним окна. Обычно программа имеет как минимум одну форму, и она появляется на экране в момент старта программы.

Основной способ, используемый для закрытия формы — это вызов метода *Close*. Если закрываемая форма в программе главная, вызов этого метода приведёт также к завершению самой программы.

Создание простейшего приложения

<u>Разработать простейшее приложение, содержащее кнопки и компоненты для вывода текста.</u>

Разработка нового приложения в Lazarus начинается с создания проекта. *Проект* — это набор связанных файлов различного типа, из которых после компиляции получается программа.
Обратите внимание, что для графического приложения, разрабатываемого в Lazarus, программе соответствует не один исходный файл, а несколько. Для сохранения программы нужно скопировать их все.
Когда вы в первый раз открываете среду Lasarus, в ней уже по умолчанию загружен пустой проект. Если в среде был загружен непустой проект (среда при загрузке может открывать последний проект, с которым шла работа) или проект был закрыт, то можно самостоятельно создать заготовку нового проекта. Для этого в меню «Файл» нужно выбрать команду «Создать…», в открывшемся окне в разделе «Проект» выделить пункт «Приложение» и нажать кнопку «OK».

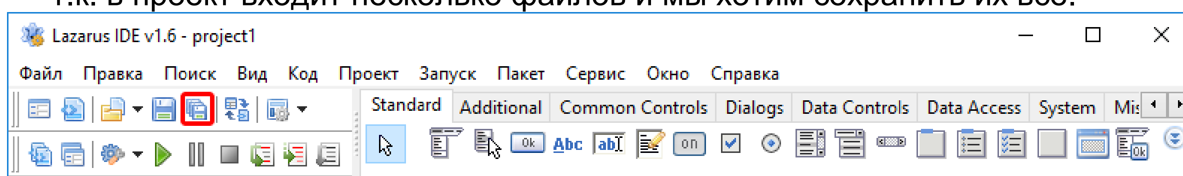
1.1.1. Проконтролируйте, что в среде загружен пустой проект, при необходимости создайте новый.

Сохраните свой (пока ещё пустой) проект.

1.1.2. Для этого создайте на диске рабочий каталог, в котором в дальнейшем будут храниться все ваши программы, а в нём — каталог с именем Project0 для своего первого приложения.

Обратите внимание! Каждый проект нужно сохранять в отдельном каталоге!

1.1.3. На панели инструментов Lazarus нажмите кнопку «Сохранить все» («Save all») — т.к. в проект входит несколько файлов и мы хотим сохранить их все.



1.1.4. В открывшемся диалоге сохранения выберите свой каталог Project0 и отредактируйте предложенное по умолчанию имя проекта «project1», заменив его на «ProjectHello» (обратите внимание на поле «Тип файла» — там выбран тип «*.lpi»).

Внимание! Старайтесь всегда давать файлам проекта осмысленные имена!

1.1.5. Нажмите кнопку «Сохранить». Диалоговое окно закроется и тут же откроется новое — с предложением сохранить модуль (в поле «Тип файла» выбраны типы «*.pas, *.pp»). *Модуль* — это отдельная единица исходного кода программы. Именно в модуле вы в дальнейшем будете писать свой код. В одной программе может быть несколько модулей.

1.1.6. Отредактируйте предложенное по умолчанию имя модуля «unit1», заменив его на «MainUnit» — т.к. это будет главный (а в данном случае и единственный) модуль вашего проекта — и нажмите кнопку «Сохранить».

С помощью файлового менеджера перейдите в каталог Project0 и изучите состав проекта.

1.1.7. После сохранения проекта в каталоге будет создано несколько файлов (после редактирования и компиляции проекта появятся и другие файлы):

– файл с иконкой проекта *.ico;
– информационный файл проекта *.lpi — это «главный» файл проекта, именно его нужно запускать, если вы хотите открыть проект для редактирования;
– исходный файл проекта *.lpr;
– файл конфигурации проекта *.lps;
– файл ресурсов *.res;
– файл формы модуля *.lfm;
– исходный код модуля на языке Free Pascal *.pas.

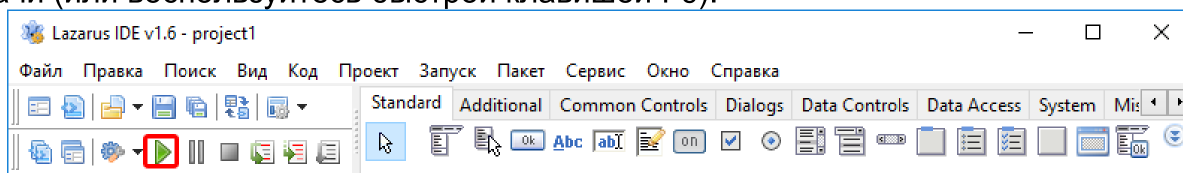
1.1.8. В проекте Lazarus могут присутствовать и другие файлы:

– файлы резервных копий *.bak — создаются при редактировании проекта;
– исполняемый файл программы *.exe — создаётся при компиляции;

– и другие служебные файлы.

Вернитесь в среду Lazarus. В проектировщике форм должна быть открыта пустая заготовка главной формы, а в редакторе исходного кода — заготовка исходного текста модуля, который ассоциирован с данной формой. Мы пока ещё не приступили к редактированию программы, однако созданная по умолчанию заготовка уже может быть скомпилирована.

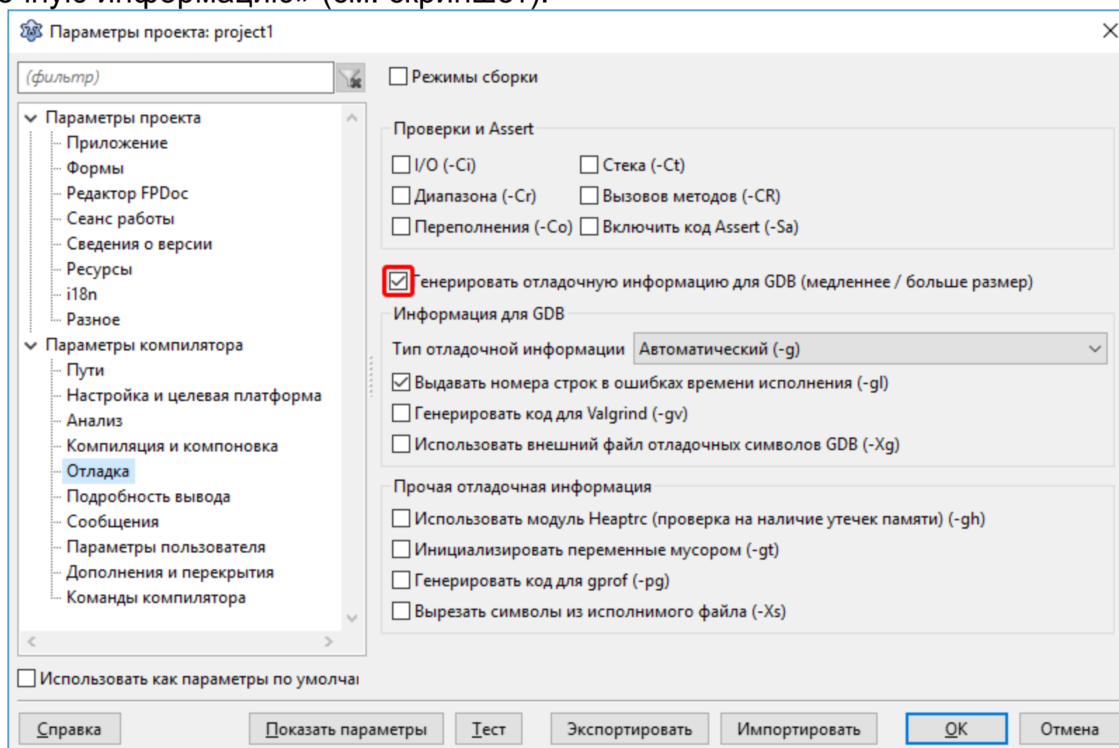
Чтобы запустить программу на выполнение нажмите кнопку «Запустить» на панели задачи (или воспользуйтесь быстрой клавишей F9).



Lazarus осуществит компиляцию приложения и запуск полученного исполняемого файла. Наше приложение пока содержит «пустое» окно без каких-либо элементов, однако это окно уже обладает базовой функциональностью — его можно перемещать, изменять размер, сворачивать, разворачивать, а также закрыть: при нажатии крестика в верхнем углу окна программа будет завершена, и вы вернётесь к редактированию проекта в среде Lazarus.

Вновь с использованием файлового менеджера перейдите в каталог своего проекта. После компиляции в нём появился исполняемый файл *.exe, а также папка lib, в которой среда хранит подключаемые к проекту данные и информацию о компиляции. Размер исполняемого файла получился достаточно большим (около 20 МБ) — и это притом, что в нашем приложении пока только одна пустая форма! Это произошло из-за того, что по умолчанию Lazarus в исполняемом файле сохраняет отладочную информацию.

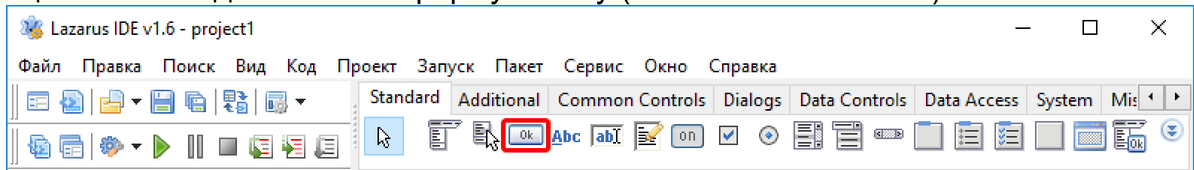
Для уменьшения размера исполняемого файла в главном окне Lazarus выберите пункт меню «Проект», затем «Параметры проекта», а в открывшемся окне параметров перейдите на страницу «Отладка» и снимите флажок напротив пункта «Генерировать отладочную информацию» (см. скриншот).



Вновь запустите программу и проверьте размер исполняемого файла — он должен стать около 2,5 МБ.

Итак, у вас на данный момент есть заготовка (прототип) будущего приложения. Реализуем для этого приложения простейший функционал. Добавьте на форму кнопку, при нажатии на которую будет появляться окно с надписью «Hello, world!»

1.1.9. Для этого со страницы Standard палитры компонентов выберите с помощью мыши и добавьте на форму кнопку (объект типа TButton).



1.1.10. При выделенной кнопке на форме (кнопка при выделении будет по границам «обрамлена» квадратными маркерами, потянув за которые можно изменить её размер) перейдите в Инспектор объектов и на вкладке «Свойства» найдите свойство Caption, которое отвечает за отображаемую на кнопке надпись. Измените надпись на кнопке по умолчанию с Button1 на Hello!.

1.1.11. Найдите в Инспекторе объектов для кнопки свойство Name — имя кнопки, позволяющее обращаться к ней в коде. Измените заданное по умолчанию имя Button1 на более «говорящее» имя ButtonHello.

Обратите внимание! Всегда старайтесь давать объектам и переменным осмысленные имена!

1.1.12. Действуя аналогичным образом, измените у формы Form1 свойство Caption (заголовок формы) на «Hello, world!», а имя формы на MainForm.

1.1.13. Создайте *обработчик события* «Щелчок по кнопке». Для этого выделите кнопку ButtonHello, перейдите в Инспектор объектов на вкладку «События» и выполните двойной щелчок в поле напротив события OnClick. При этом откроется окно редактора кода с заготовкой обработчика события OnClick:

```
procedure TMainForm.ButtonHelloClick(Sender: TObject);
begin

end;
```

1.1.14. Чтобы задать те действия, которые будут выполняться, когда пользователь щёлкнет на кнопке в запущенной программе, нужно в теле этой процедуры (между зарезервированными словами **begin** и **end**) написать необходимый код.

1.1.15. Запрограммируйте появление информационного окна с текстом «Hello, world!» по нажатию на кнопку. Для этого в теле обработчика наберите следующий код (процедура ShowMessage создаёт стандартное окно для вывода сообщений):

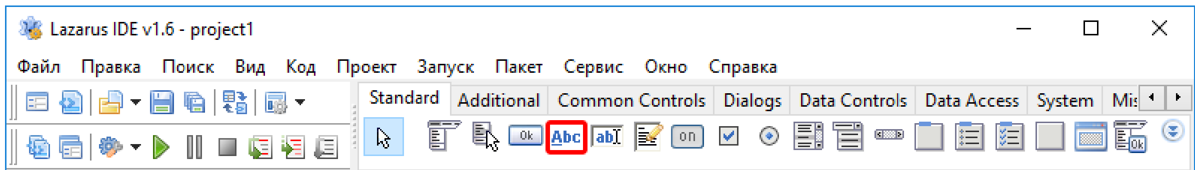
ShowMessage('Hello, world!');

Протестируйте приложение. Для этого запустите его (команда «Запустить» или F9), и после запуска щёлкните на кнопке. На экране появиться небольшое диалоговое окно с надписью «Hello, world!» и одной кнопкой «OK».

Это диалоговое окно является модальным — пока оно не будет закрыто (крестиком или щелчком по кнопке «OK»), выполнение программы будет приостановлено и другие окна программы будут недоступны.

На практике использовать для вывода информации модальные окна далеко не всегда удобно. Гораздо чаще используются менее навязчивые способы информирования пользователя. Внесите изменения в программу, таким образом, чтобы для вывода сообщения использовался компонент «Метка», позволяющий отобразить произвольный текст.

1.1.16. Для этого выберите из палитры компонентов (страница Standard) метку (объект типа TLabel) и разместите её на форме.

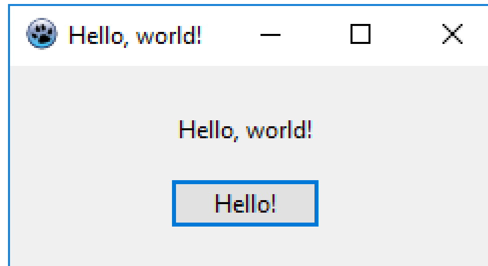


1.1.17. С помощью Инспектора объектов измените имя метки на LabelHello, а отображаемый текст на пустую строку.

1.1.18. В редакторе кода замените тело обработчика щелчка по кнопке — процедуры TMainForm.ButtonHelloClick — на следующий код:

LabelHello.Caption := 'Hello, World!';

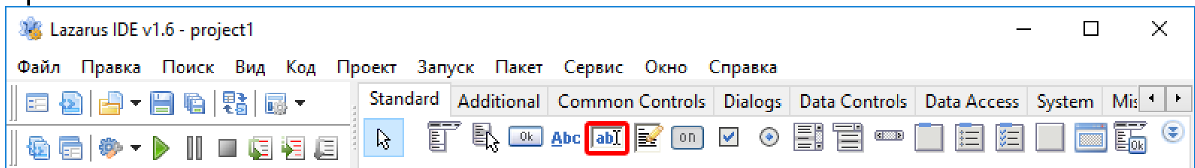
1.1.19. Протестируйте программу. После нажатия на кнопку вид окна должен быть примерно следующий:



Обратите внимание! При проектировании интерфейса программы используйте возможности проектировщика форм по изменению размеров и положения компонентов для создания аккуратного и удобного интерфейса.

Сделаем наше приложение более интерактивным — добавим для пользователя возможность менять текст, выводимый программой: пусть имя того, с кем программа будет «здороваться», определяет пользователь.

1.1.20. Добавьте на форму поле редактирования (компонент типа TEdit — позволяет пользователю вводить и редактировать текст) со страницы Standard палитры компонентов.



1.1.21. Измените у поля редактирования свойство Name на EditHello.

1.1.22. У поля редактирования нет свойства Caption. Отображаемый текст (заданный по умолчанию, введённый или отредактированный пользователем) хранится в свойстве Text. Через Инспектор объектов установите значение этого свойства по умолчанию: «world».

1.1.23. Отредактируйте обработчик нажатия на кнопку ButtonHello таким образом, чтобы программа брала имя для приветствия из поля редактирования EditHello.

Указание: в свойствах Caption и Text хранятся строковые данные, и работать с ними нужно как с переменными типа string.

1.1.24. У вас должен получиться следующий код:

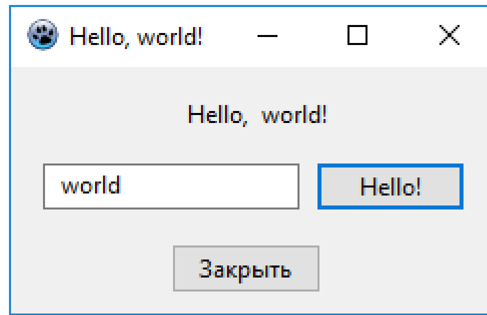
LabelHello.Caption := 'Hello, ' + EditHello.Text + '!';

1.1.25. Запустите программу и протестируйте её работу.

Самостоятельно добавьте на форму ещё одну кнопку, по нажатию на которую будет происходить выход из приложения (закрыть приложение позволяет процедура Close).

Обратите внимание! Более быстрый способ создать обработчик события OnClick для кнопки по сравнению с описанным выше — это просто в проектировщике форм выполнить двойной щелчок по кнопке.

Примерный вид окна итоговой программы после запуска и нажатия на кнопку «Hello!»:


Hello, world!
Hello, world!
world
Hello!
Закрыть