 Министерство образования Красноярского края

Краевое государственное бюджетное профессиональное образовательное

учреждение «Красноярский колледж радиоэлектроники и информационных технологий»



**МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ**

ОП.06 Основы алгоритмизации и программирования

интеллектуальных интегрированных систем

По учебной дисциплине \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Курс \_\_\_\_

2

Для специальности (код и наименование)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

09.02.08 Интеллектуальные интегрированные системы

Красноярск, 2025

Методические рекомендации составлены:

Преподавателем высшей категории КГБПОУ СПО «ККРИТ» Е.В. Харитоновой

Преподавателем высшей категории КГБПОУ СПО «ККРИТ» Л.В. Шайхутдиновой

Преподавателем первой категории КГБПОУ СПО «ККРИТ» К.В. Савельевой

РАССМОТРЕНО

на заседании цикловой комиссии преподавателей

профессионального цикла информационно-технического профиля

Протокол №\_\_\_ от «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2025 г.

Председатель ЦК \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.В. Харитонова

Ответственный редактор: зам. директора по учебной работе М.А. Полютова

Одобрено Методическим советом КГБПОУ СПО «ККРИТ»

протокол № \_\_ от «\_\_\_» \_\_\_\_\_\_ 2025 г.

Председатель методического совета

Зам. директора по УР М.А. Полютова

Содержание

Пояснительная записка

Перечень практических работ

Методические указания для студентов по выполнению практических занятий

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Методические указания по выполнению практических занятий по предмету ОП.06 Ос- новы алгоритмизации и программирования разработаны с целью оказания помощи студентам специальности 09.02.08 Интеллектуальные интегрированный системы и преподавателям по ор- ганизации практических занятий по изучаемой дисциплине, в соответствии с требованиями фе- дерального государственного стандарта среднего профессионального образования.

Методические разработка включает в себя краткие теоретические сведения, указания по выполнению практических работ, контрольные вопросы, формы контроля.

В соответствии с учебным планом на практические занятия для студентов отводится

**30** часов.

Учебная дисциплина ОП.06 Основы алгоритмизации и программирования относится к обязательной части общепрофессионального цикла.

Учебная дисциплина ОП.06 Основы алгоритмизации и программирования обеспечивает формирование профессиональных и общих компетенций по всем видам деятельности ФГОС по специальности 09.02.08 Интеллектуальные интегрированный системы.

Методические указания направлены на формирование и развитие у студентов общих и профессиональных компетенций:

ОК 01 Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации, и информационные технологии для выполнения задач профессиональной деятельности

ОК 04 Эффективно взаимодействовать и работать в коллективе и команде;

ОК 05 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста;

ОК 09 Пользоваться профессиональной документацией на государственном и иностранном языках;

ПК 3.1 Разрабатывать программные модули для интеллектуальных интеграционных решений; ПК 3.2 Выполнять отладку программных модулей для интеллектуальных интеграционных решений с использованием специализированных программных средств;

ПК 3.3 Выполнять тестовый запуск программных модулей для интеллектуальных интеграционных решений и обеспечивать их требуемое качество.

В результате освоения учебной дисциплины обучающийся **должен**

**иметь практический опыт:** посторенние алгоритмов для конкретных программ; работать в языках программирования; выполнять проверку, отладку программного кода, оформлять программный код в соответствии со стандартом.

# уметь:

* разрабатывать алгоритмы для конкретных задач;
* использовать программы для графического отображения алгоритмов;
* определять сложность работы алгоритмов;
* работать в среде программирования;
* реализовывать построенные алгоритмы в виде программ на конкретном языке программирования;
* оформлять код программы в соответствии со стандартом кодирования.

# знать:

* понятие алгоритмизации, свойства алгоритмов, общие принципы построения алгоритмов, основные алгоритмические конструкции;
* эволюцию языков программирования, их классификацию, понятие системы программирования;
* основные элементы языка, структуру программы, операторы и операции, управляющие структуры, структуры данных, файлы, классы памяти;
* подпрограммы, составление библиотек подпрограмм;
* объектно-ориентированную модель программирования, основные принципы объектно- ориентированного программирования на примере алгоритмического языка: понятие классов и объектов, их свойств и методов, инкапсуляция и полиморфизма, наследования и переопределения.

Характерная черта практических занятий – индивидуальное выполнение заданий, само- стоятельное приобретение знаний. В связи с этим предусмотрены работы по всем основным разделам курса. Перед выполнением практической работы обучающийся получает опережаю- щее теоретическое домашнее задание. На занятии объясняются вопросы, уточняются определе- ния, которые помогают выполнению заданий. Обучающийся может просмотреть запись объяс- нения любой примерной работы по всем темам. И только после этого обучающийся приступает к выполнению практической работы.

При выполнении работы обучающийся должен самостоятельно изучить методические рекомендации по проведению практической работы, подготовить ответы на контрольные во- просы. Все практические задания выполняются за компьютером, теоретические вопросы сда- ются устно или письменно.

# Объем учебной дисциплины и виды учебной работы

|  |  |
| --- | --- |
| **Вид учебной работы** | **Объем в часах** |
| **Объем образовательной программы учебной дисциплины** | 156 |
| **Объем работы обучающихся во взаимодействии с преподавателем** | 112 |
| в том числе: | |
| практические занятия | 46 |
| самостоятельная работа | 30 |
| Промежуточная аттестация в форме экзамена | 10 |
| В т.ч. консультации | 4 |
| **Промежуточная аттестация в форме экзамена** | |

**ПЕРЕЧЕНЬ ПРАКТИЧЕСКИХ РАБОТ**

**Практическое занятие № 1** Анализ данных и формализация поставленной задачи

**Практическая работа № 2.** Знакомство со средой программирования. **Практическая работа № 3.** Составление программ линейной структуры. **Практическая работа № 4.** Составление программ циклической структуры **Практическая работа № 5.** Составление программ обработки одномерных массивов. **Практическая работа № 6.** Составление программ обработки двумерных массивов **Практическая работа № 7.** Обработка двумерных массивов.

**Практическая работа № 8.** Работа со строками.

**Практическая работа № 9.** Работа с данными типа множество. **Практическая работа № 10.** Файлы последовательного доступа. **Практическая работа № 11.** Типизированные файлы.

**Практическая работа № 12.** Применение рекурсивных функций.

**Практическая работа № 13.** Программирование модуля.

**Практическая работа № 14.** Использование указателей для организации связанных списков. **Практическая работа № 15.** Создание проекта с использованием компонентов для работы с текстом.

**Практическое занятие № 16.** Создание проекта с использованием компонентов ввода и отображения чисел, дат и времени

**Практическое занятие № 17.** Создание проекта с использованием кнопочных компонентов

**Практическое занятие № 18.** Создание проекта с использованием компонентов стандартных диалогов и системы меню.

**Практическая работа № 19.** Создание процедур на основе событий.

**Практическая работа № 20.** Разработка оконного приложения с несколькими формами

**Практическое занятие № 21.** Разработка игрового приложения. **Практическая работа № 22.** Тестирование, отладка приложения. **Практическое занятие № 23.** Программирование приложений

# МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ СТУДЕНТОВ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕ- СКИХ ЗАНЯТИЙ

**Практическая работа №1.** Анализ данных и формализация поставленной задачи. Цель занятия:

* научиться анализировать данные и выполнять формализацию поставленной задачи; Оснащение:
* ПК;
* Программное обеспечение: Pascal ABC.

# Краткие теоретические сведения

Вычислительные технологии в современном мире являются самыми перспективными средствами как для выполнения научных исследований, так и для повышения конкурентоспособ- ности промышленности. Для решения сложных реальных задач объединяются силы учёных-ис- следователей и практиков, а также вычислительных мощностей разных компьютеров, иногда и по всему миру. Для синхронизации деятельности людей и вычислительной техники необходима чёткая организация, планирование, разделение творческих и вычислительных процессов. Ко- нечно, не все задачи требуют такой сложной организации. Однако умение планировать сложную деятельность полезно осваивать на простых примерах.

Обсудим процесс решения задачи, разделив его на отдельные этапы.

1. Постановка задачи. Это первый этап решения задачи, на котором определяется цель решения, собираются исходные данные, выясняются формат и область значений результатов. На этом этапе предполагается построение адекватных моделей объектов исследования с характери- стиками, близкими к оригиналу. Здесь важно определить оптимальный набор исходных данных, которые позволят решить задачу, но не сделают её решение трудновыполнимым. На входе — проблема; на выходе — исходные данные и формат результатов.
2. Формализация задачи. На этом этапе осуществляется построение математической мо- дели задачи. Взаимодействие моделей объектов, участвующих в реальном процессе, заменяется известными математическими соотношениями и уравнениями.

Зачастую в определённой сфере человеческой деятельности уже накоплен набор матема- тических уравнений, с помощью которых принято решать те или иные задачи. Умение предло- жить новые модели решения связано с широким кругозором исследователя, его неординарным подходом. Такие же качества требуются от человека и при решении новых задач, тех, которые ранее не решались в конкретной области.

На этом этапе реализуется исключительно творческая деятельность человека, а успех ре- шения задачи зависит от выбора и обоснования метода решения.

На этом же этапе определяются допустимая точность, объём и время вычислений, а также технические средства, с помощью которых будет реализовываться решение.

На входе — исходные данные и формат результатов; на выходе — математическая модель и технические средства для решения.

1. Создание алгоритма решения. На этом этапе реализуются методы, с помощью которых будет осуществляться решение. Здесь необходимо скоординировать работу

Политика обработки персональных данных всех участников проекта решения. Создаются блок-схемы алгоритмов, технологические карты загрузки оборудования, например, распаралле- ливание вычислений. Определяется последовательность выполнения отдельных блоков вычис- лений.

На входе — математическая модель; на выходе — блок-схема или другой способ описания алгоритма.

1. Составление программы для решения задачи. Алгоритм реализуется на конкретном языке программирования. Уточняются формат и область значений результатов в зависимости от выбранного языка программирования.

На входе — блок-схема или другой способ описания алгоритма; на выходе — программа обработки исходных данных.

1. Тестирование и отладка программы. Составление программы обработки исходных дан- ных — процесс трудоёмкий, особенно если необходимо соединить блоки разных разработчиков и параллельно работающих процессов. Отладка программы выполняется в несколько этапов: определяются синтаксические и семантические ошибки. Для обнаружения ошибок проводят те- стирование на специально подготовленных данных.

Внутри программы расставляют «контрольные точки», получая таким образом сообщение об ошибке в ходе выполнения программы до её завершения. Кроме компьютерного тестирования полученные данные должен проанализировать специалист в той области, в которой работает по- ставленная задача. Такой подход позволяет выявить ошибки на предыдущих этапах решения за- дачи. Например, анализ результатов может выявить недочёты, связанные с выбором алгоритма решения, постановкой задачи или со сбором исходных данных. В этом случае процесс решения возвращается на предыдущие этапы и проводится их корректировка.

На входе — программа обработки исходных данных; на выходе — результаты решения задачи.

**Задача**. Движение автомобиля.

|  |  |
| --- | --- |
| **Что моделируется?** | **Процесс движения объекта «автомобиль»** |
| **Вид движения** | Равноускоренное |
| **Что известно о движении?** | Начальная скорость (V0), ускорение (а), макси- мально развиваемая скорость (Vmax) |
| **Что надо найти?** | Скорость (Vi) в заданные моменты времени (ti). |
| **Как задаются моменты времени?** | От нуля через равные интервалы (∆t)? |
| **Что ограничивает расчеты?** | Vi < Vmax |

Такие характеристики объекта, как цвет, тип кузова, год выпуска и общий пробег, степень изношенности шин и многие другие, в данной постановке учитывать не будем.

*Разработка модели*

Этап разработки модели начинается с построения информационной модели в различных знаковых формах, которые на завершающей стадии воплощаются в компьютерную модель. В информационных моделях задача приобретает вид, позволяющий принять решение о выборе программной среды и четко представить алгоритм построения компьютерной модели.

*Информационная модель*

Выбор наиболее существенных данных при формировании информационной модели и ее сложность определяются целью моделирования. Параметры объектов, определенных при форма- лизации задачи, располагаются в порядке убывания значимости. При моделировании учитыва- ются не все, а лишь некоторые свойства, интересующие исследователя.

Если отбросить существенные факторы, то модель будет неверно отражать оригинал (про- тотип). Если оставить их слишком много, модель окажется сложна для построения и исследова- ния. Во многих исследованиях создают несколько моделей одного объекта, начиная от простей- ших, с минимальным набором определяющих параметров. Затем постепенно уточняют модель, добавляя некоторые из отброшенных характеристик.

Иногда задача может быть уже сформулирована в упрощенной форме, цель — четко по- ставлена, а параметры модели, которые надо учесть, — определены. Задачи такого вида вам при- ходилось неоднократно решать на уроках математики и физики. Однако в обычной жизни отбор информации приходится проводить самостоятельно.

Результатом построения информационной модели является хорошо знакомая вам таблица характеристик объекта. В зависимости от типа задачи таблица может видоизменяться.

Рассмотрим информационные модели описанной выше задачи.

**Задача.** Движение автомобиля

|  |  |
| --- | --- |
| **Объект моделирования** | **Параметры** |
|  | |
| **Процесс движения автомо-** | V0 – начальная скорость |
| **биля** | ∆t – интервал изменения времени |
|  | а – ускорение |
|  | Vмакс – максимально развиваемая автомобилем скорость |
|  | ti – время движения |
|  | Vi – значения скорости |

При построении компьютерной образно-знаковой модели (текстовый или графический документ) информационная модель будет описывать объекты, их параметры, а также предвари- тельные исходные значения, которые исследователь определяет в соответствии со своим опытом и представлениями, а затем уточняет в ходе компьютерного эксперимента.

Информационная модель, как правило, представляется в той или иной знаковой форме.

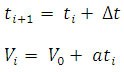
Таблица – один из примеров знаковых моделей.

Иногда полезно дополнить представление об объекте и другими знаковыми формами (схе- мой, чертежом, формулами), если это способствует лучшему пониманию задачи.

Рассмотрим знаковые модели для описанных выше задач.

**Задача.** Движения автомобиля.

Задача о движении автомобиля становится более понятной, если привести рисунок с ука- занием обозначений, используемых в задаче.



Математическая модель движения автомобиля имеет вид:

Правильно составленная математическая модель просто необходима в задачах, где требу- ется рассчитать значения параметров объекта.

Знаковые формы могут иметь и другой вид.

Например, при создании географических или исторических карт разрабатывается система условных обозначений.

И лишь для простых, знакомых по содержанию задач знаковые модели не требуются.

# Компьютерная модель

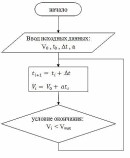
Теперь, когда сформирована информационная знаковая модель можно приступать соб- ственно к компьютерному моделированию - созданию компьютерной модели. Сразу возникает вопрос о средствах, которые необходимы для этого, то есть об инструментах моделирования.

Компьютерная модель - это модель, реализованная средствами программной среды. Существует множество программных комплексов, которые позволяют проводить постро-

ение и исследование моделей (моделирование). Каждая программная среда имеет свой инстру- ментарий и позволяет работать с определенными видами информационных моделей. Поэтому перед исследователем возникает нелегкий вопрос выбора наиболее удобной и эффективной среды для решения поставленной задачи. Надо сказать, что одну и ту же задачу можно решить, используя различные среды.

От выбора программной среды зависит алгоритм построения компьютерной модели, а также форма его представления.

Например, это может быть блок-схема. На рисунке ниже представлен алгоритм задачи о движении автомобиля в виде блок-схемы. Руководствуясь блок-схемой, задачу можно решить в разных средах. В среде программирования - это программа, записанная на алгоритмическом языке. В прикладных средах это последовательность технологических приемов, приводящая к решению задачи.



Например, при моделировании в среде графического редактора или текстового процес- сора алгоритм может быть представлен в словесной форме, описывающей последовательность действий по созданию объектов и, если требуется, технологических приемов. При разработке ал- горитма построения модели в электронных таблицах особое внимание обращается на выделение области исходных и расчетных данных и правила записи формул, связывающих данные разных областей.

Исходя из вышесказанного, можно сделать вывод, что при моделировании на компьютере необходимо иметь представление о классах программных средств, их назначении, инструмента- рии и технологических приемах работы. Разнообразное программное обеспечение позволяет пре- образовать исходную информационную знаковую модель в компьютерную и провести компью- терный эксперимент.

Рассмотрим возможные варианты выбора компьютерной среды для приведенной выше за- дачи. Справедливости ради следует заметить, что предложенная в качестве иллюстрации задача может быть решена и зачастую решается без применения компьютера.

**Задача.** Движение автомобиля.

Для задач, в которых требуется получить расчетные значения, подходит среда электрон- ных таблиц. В этой среде информационная и математическая модели объединяются в таблицу, содержащую три области: исходные данные, промежуточные расчеты и результаты. Электронная таблица позволяет не только рассчитать требуемые скорости, но и построить график движения автомобиля.

# Выполнить тест

1. В таблице приведена стоимость перевозок между соседними железнодорожными стан- циями. Укажите схему, соответствующую таблице.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | A | | | | B | | | C | | | D | |
| A | | | |  | | | | 4 | | |  | | | 5 | |
| B | | | | 4 | | | |  | | | 3 | | | 6 | |
| C | | | |  | | | | 3 | | |  | | |  | |
| D | | | | 5 | | | | 6 | | |  | | |  | |
|  | 1) | | | | 2) | | | | 3) | | | 4) | | | |
|  |  |  | |  |  |  | |  |  |  |  |  | |  |

1. Путешественник пришел в 08:00 на автостанцию поселка ЛЕСНОЕ и увидел следующее расписание автобусов:

*Отправление из Прибытие в*  *Время отправления Время прибытия ЛЕСНОЕ ОЗЕРНОЕ 07:45 08:55*

*ЛУГОВОЕ ЛЕСНОЕ 08:00 09:10*

*ПОЛЕВОЕ ЛЕСНОЕ 08:55 11:25*

*ПОЛЕВОЕ ЛУГОВОЕ 09:10 10:10*

*ЛЕСНОЕ ПОЛЕВОЕ 09:15 11:45*

*ОЗЕРНОЕ ПОЛЕВОЕ 09:15 10:30*

*ЛЕСНОЕ ЛУГОВОЕ 09:20 10:30*

*ОЗЕРНОЕ ЛЕСНОЕ 09:25 10:35*

*ЛУГОВОЕ ПОЛЕВОЕ 10:40 11:40*

*ПОЛЕВОЕ ОЗЕРНОЕ 10:45 12:00*

Определите самое раннее время, когда путешественник сможет оказаться в пункте ПОЛЕВОЕ согласно этому расписанию*.*

1) 10:30 2) 11:25 3)11:40 4) 11:45

1. Между населёнными пунктами A, B, C, D, E, F построены дороги, протяжённость кото- рых приведена в таблице. (Отсутствие числа в таблице означает, что прямой дороги между пунктами нет.)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F |
| A |  | 5 |  |  |  |  |
| B | 5 |  | 9 | 3 | 8 |  |
| C |  | 9 |  |  | 4 |  |
| D |  | 3 |  |  | 2 |  |
| E |  | 8 | 4 | 2 |  | 7 |
| F |  |  |  |  | 7 |  |

*Определите длину кратчайшего пути между пунктами A и F (при условии, что передвигаться можно только по построенным дорогам).*

1) 11 2) 13 3) 15 4) 17

1. Транспортная фирма осуществляет грузоперевозки разными видами транспорта между четырьмя городами: ЧЕРЕПОВЕЦ, МОСКВА, КУРСК, ПЕРМЬ. Стоимость доставки грузов и время в пути указаны в таблице:

*Стоимость (у.е.)*  *Время в*

*Пункт отправления*

*пути*

*Пункт назначения*

|  |  |  |  |
| --- | --- | --- | --- |
| *МОСКВА* | *ПЕРМЬ* | *100* | *70* |
| *МОСКВА* | *КУРСК* | *30* | *10* |

|  |  |  |
| --- | --- | --- |
| *МОСКВА ЧЕРЕПОВЕЦ* | *50* | *15* |
| *ПЕРМЬ МОСКВА* *100* | *69* |  |
| *ЧЕРЕПОВЕЦ ПЕРМЬ* | *140* | *80* |
| *ЧЕРЕПОВЕЦ МОСКВА* | *50* | *15* |
| *ЧЕРЕПОВЕЦ КУРСК* | *100* | *80* |
| *КУРСК ПЕРМЬ* *60* | *40* |  |
| *КУРСК МОСКВА* *30* | *10* |  |
| *КУРСК ЧЕРЕПОВЕЦ* | *100* | *80* |
| *КУРСК ЧЕРЕПОВЕЦ* | *90* | *100* |

Определите маршрут наиболее дешевого варианта доставки груза из ЧЕРЕПОВЦА в ПЕРМЬ. Если таких маршрутов несколько, в ответе укажите наиболее выгодный по времени вариант*.*

1. ЧЕРЕПОВЕЦ – ПЕРМЬ
2. ЧЕРЕПОВЕЦ – КУРСК – ПЕРМЬ
3. ЧЕРЕПОВЕЦ – МОСКВА – ПЕРМЬ
4. ЧЕРЕПОВЕЦ – МОСКВА – КУРСК – ПЕРМЬ
5. В таблицах приведена стоимость перевозки грузов между соседними станциями. Если пересечение строки и столбца пусто, то соответствующие станции не являются соседними. Ука- жите номер таблицы, для которой выполняется условие «Максимальная стоимость перевозки грузов от пункта В до пункта D не больше 6».

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1) | | | | | | | 2) | | | | | | | 3) | | | | | | | 4) | | | | | | |
|  |  | A | B | C | D |  |  |  | A | B | C | D |  |  |  | A | B | C | D |  |  |  | A | B | C | D |  |
| A |  | 2 |  | 2 | A |  | 2 | 1 | 1 | A |  | 1 | 3 | 6 | A |  | 3 | 2 | 1 |
| B | 2 |  | 4 | 3 | B | 2 |  | 4 |  | B | 1 |  | 2 | 4 | B | 3 |  | 2 |  |
| C |  | 4 |  | 4 | C | 1 | 4 |  | 1 | C | 3 | 2 |  |  | C | 2 | 2 |  | 4 |
| D | 2 | 3 | 4 |  | D | 1 |  | 1 |  | D | 6 | 4 |  |  | D | 1 |  | 4 |  |
|  | | | | | |  | | | | |  | | | | |  | | | | | |

# Контрольные вопросы

1. Что происходит на этапе формализации задачи?
2. Какие этапы включает в себя процесс решения задач?
3. Какой этап решения задачи следует за этапом построения математической модели?
4. Какой этап является заключительным при решении задач на компьютере?

**Практическая работа №2.** Знакомство со средой программирования Цель занятия:

* + научиться применять основные операторы и редактировать программы с линейной струк- турой;
  + сформировать навыки работы в среде программирования; Оснащение:
  + ПК;
  + Программное обеспечение: Pascal ABC.

# Краткие теоретические сведения

Основными операторами являются:

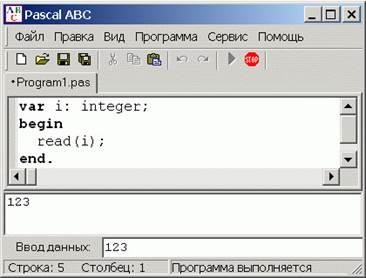
* + - Read, Readln – оператор ввода данных;
    - : = – оператор присваивания;
    - Write, Writeln - оператор вывода.

Program - зарезервированное слово для обозначения заголовка программы. Var - зарезервированное слово для обозначения раздела переменных.

Begin и End - служебные слова, заключающие между собой раздел операторов.



Структура программы с линейной структурой

[](http://gospodaretsva.com/wp-content/uploads/2011/11/urok8-1-Pascal-ABC.jpg)

Окно программы Pascal ABC состоит из окна редактора кода, окна ввода и окна вывода.

# Окно редактора кода

Большую часть рабочей области, её верхнюю часть занимает окно редактора кода. В него вводится исходный текст программы.

**Горячие клавиши**, которые можно использовать при работе с текстом программы: F2, Ctrl-S- сохранить файл.

F3, Ctrl-O- загрузить файл.

F12- сохранить файл под новым именем. Ctrl-Shift-S- сохранить все открытые файлы.

Ctrl-Tab, Ctrl-Shift-Tab – перейти к следующему/предыдущему окну редактора. Ctrl-Shift-I- увеличить отступ выделенного блока.

Ctrl-Shift-U – уменьшить отступ выделенного блока.

Под окном редактора расположено окно вывода. Оно предназначено для вывода данных процедурами write и writeln, а также для вывода сообщений об ошибках и предупреждений во время работы программы.

Окно вывода может быть скрыто. Клавиша F5 показывают/скрывают окно вывода. Для скрытия окна вывода используется также клавиша Esc.

Окно вывода обязательно открывается при любом выводе в него.

Для очистки окна вывода следует нажать комбинацию клавиш Ctrl-Del или кнопку [](http://gospodaretsva.com/wp-content/uploads/2011/11/Urok8-2-Pascal-ABC1.jpg) . Окно ввода открывается при выполнении процедур read и readln в ходе работы про-

граммы.

Ввод данных в окно ввода сопровождается эхо-выводом в окно вывода. После нажатия клавиши Enter данные из окна ввода попадают в соответствующие переменные, окно ввода за- крывается, и программа продолжает работать дальше.

# Запуск и остановка программы

Для запуска программы в текущем окне редактора следует нажать клавишу F9 или кнопку [](http://gospodaretsva.com/wp-content/uploads/2011/11/Urok8-4-Pascal-ABC.jpg) панели инструментов.

Программа вначале компилируется во внутреннее представление, после чего, если не найдены ошибки, программа начинает выполняться. При выполнении программы кнопка за- пуска программы становится неактивной, кнопка останова программы, наоборот, активной и в строке статуса отображается информация “Программа выполняется”.

Выполнение программы можно в любой момент прервать нажатием комбинации клавиш Ctrl-F2 или кнопки [](http://gospodaretsva.com/wp-content/uploads/2011/11/Urok-8-3-Pascal-ABC.jpg) При этом в окне вывода появится сообщение: • Программа прервана пользователем.

# Практические задания

## Содержание отчета по результатам выполнения практической работы

Отчет должен содержать:

1. Название работы.
2. Цель работы.
3. Результат выполнения задания № 1 (записать программу на компьютер и в тетрадь).
4. Результат выполнения задания № 2 (записать программу на компьютер и в тетрадь)
5. Результат выполнения задания № 3 (записать программу на компьютер и в тетрадь)
6. Результат выполнения задания № 4 (записать программу на компьютер и в тетрадь)
7. Результаты выполнения задания № 5 (записать программу на компьютер и в тетрадь)
8. Результаты выполнения задания № 6 (ответить письменно на вопросы)
9. Вывод по работе (результат выполнения № 7).

**Задание 1.** Наберите текст программы, описывающей решение примера С=А+В. Выполните компиляцию программы и тестирование.

* 1. На рабочем столе запустите ярлык Pascal ABC.
  2. В окне текстового редактора программы Pascal ABC наберите текст программы: Program Z1;

Var А, В, С: integer;

Begin

Writeln ('Задача 1');

Writeln ('Введите число А'); Readln (A);

Writeln ('Введите число В'); С:=А+В;

Writeln ('с=' , с ); Readln ;

End .

* 1. Проверьте программу на ошибки (компиляция) - нажать *Программа/Компилиро- вать* (или Ctrl+F9).
  2. Протестируйте программу:
     1. запустить программу, выбрав в меню пункт *Программа/Выполнить (или F9).*
     2. в ответ на приглашение *«Введите число А»* набрать *23,*
     3. в ответ на приглашение *«Введите число В»* набрать *17,*
     4. если программа работает правильно, то в результате будет выведено – С=40
  3. Самостоятельно протестируйте программу с другими числовыми данными: А=2; В= -12;

А = -10; В= -17;

А = 0; В= - 20;

Запишите результаты в тетрадь.

**Задание 2.** Измените исходную программу, чтобы она находила **частное** двух чисел.

**Задание 3.** Измените исходную программу, чтобы она находила сумму четырех чисел.

https://documents.infourok.ru/ee5040e9-9033-47ae-b098-e66e7d2d98d7/0/image006.png**Задание 4.** Наберите текст программы, описывающей решение примера Выполните компиляцию программы и тестирование.

1. В окне текстового редактора программы Pascal наберите текст программы: Program Z2;

Var А, В, D : real; Begin

Writеln ('Задача 2'); Writеln ('Введите А'); Readln (A);

Writeln ('Введите В'); Readln ( В );

D:= (6\*A+2\*B)/ (4\*A);

Writeln ('D= ', D ); Readln ;

End.

1. Проверить программу на ошибки - нажать *Программа/Компилиро- вать* (или Ctrl+F9).
2. Протестировать программу:
   1. запустить программу, выбрав в меню пункт *Программа/Выполнить (или F9)*
   2. в ответ на приглашение *«Введите число А»* набрать *2,*
   3. в ответ на приглашение *«Введите число В»* набрать *2,*
   4. если программа работает правильно, то в результате будет выведено –

*Е=2.0000000000Е+0000.*

1. Введите ограничение для вещественного числа, для этого внесите изменения в текст программы Writeln ('D = ', D:4:2);
2. Самостоятельно протестируйте программу с другими числовыми данными: А=2; В=1,5;

А = 3; В= -7;

А = 0; В= - 2;

Запишите результаты в тетрадь.

**Задание 5.** Создайте и протестируйте программу вычисления переменной **D=(a\*b)+2**

## Контрольные вопросы:

1. Как открыть новое окно программы?
2. Как сохранить новый текст программы?
3. Как «запустить» программу?
4. Назовите оператор вывода в Pascal.
5. Назовите оператор ввода в Pascal.
6. Какие типы данных применимы в этой практической работе.

Сделайте вывод о проделанной практической работе.

**Практическая работа №3.** Составление программ линейной структуры

Цель:

 освоить методы разработки алгоритмов ипрограммирования задач линейной структуры.

# Краткие теоретические сведения

**Алгоритм линейной структуры** – это алгоритм, в котором все действия выполняются последовательно друг за другом и только одинраз. Блок-схема алгоритма представляет собой по- следовательностьблоков, которые располагаются сверху вниз в порядке их выполнения. Все промежуточные или исходные данные влияния на направление процесса выполнения не оказы- вают.

Пример. Вычислить высоты треугольника со сторонами a, b, c, используя формулы:

2

ℎ = 𝑎 √𝑝(𝑝 − 𝑎)(𝑝 − 𝑏)(𝑝 − 𝑐)

2

ℎ = 𝑏 √𝑝(𝑝 − 𝑎)(𝑝 − 𝑏)(𝑝 − 𝑐)

2

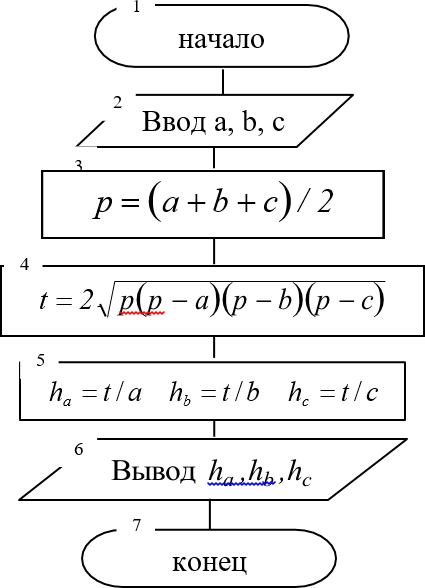
ℎ = 𝑐 √𝑝(𝑝 − 𝑎)(𝑝 − 𝑏)(𝑝 − 𝑐)

При решении данной задачи для исключения повторений следует вычислять высоты не по приведенным выше формулам непосредственно, а используя промежуточную переменную:

𝑡 = 2√𝑝(𝑝 − 𝑎)(𝑝 − 𝑏)(𝑝 − 𝑐)

*тогда ha* = *t* / *a*, *hb* = *t* / *b*, *hc* = *t* / *c*

Схема алгоритма решения задачи имеет следующий вид:



Для записи программы линейной структуры необходимы операторы присваивания, ввода исходных данных и вывода результатов вычислений.

Программа на языке Pascal состоит из заголовка, разделаописаний и раздела операторов: Program имя;

раздел описаний begin

раздел операторов end.

Заголовок программы начинается со служебного слова Program, после которого указыва-

ется имя программы. Раздел описаний предназначен для объявления в программе всех данных, которые встречаются, а также их характеристик.

Существует определенный порядок в разделе описаний:

* раздел меток Label;
* раздел констант Const;
* раздел типов Type;
* раздел переменных Var;
* раздел процедур и функций (Function и Procedure).

Раздел операторов заключается в операторные скобки вида: Begin (начать) и End (закон- чить), при этом после End ставится точка. Точка – это признак окончания программы. В разделе операторов записывается последовательность выполняемых операторов. Операторы отделяются друг от друга символом ";"

Оператор присваивания – это основной оператор любого языка программирования. Он предназначен для замены текущего значения переменной новым значением.

Форма оператора присваивания имеет вид:

## Переменная : = выражение;

При выполнении этого оператора значение выражения вычисляется и присваивается переменной.

Например, а := b+c;

w := sin(sqr(t))/(s + ln(v));s := „строка‟;

Имя переменной и результат выражения должны принадлежать к одному типу. Для ввода данных используются операторы:

***read (элемент 1, элемент 2, ...);***

# readln (элемент 1, элемент 2, ...);

При выполнении оператора read (элемент1, элемент2, ...) происходят следующие дей- ствия: программа приостанавливает свою работу и ждет, пока на клавиатуре будут набраны дан- ные и нажата клавиша Enter. После нажатия клавиши Enter, введенные значения присваиваются переменным, имена которых указаны в операторе read. Числовые значения должны быть набраны в одной строке иразделены пробелами.

Оператор readln (элемент 1, элемент 2, ...) осуществляет ввод данных, а затем обеспечивает переход к началу новой строки.

Для вывода данных используются операторы:

***write (элемент 1, элемент 2, ...);***

# writeln (элемент 1, элемент 2, ...);

где элемент – это переменная или строка символов, заключеннаяв апострофы.

Например, оператор write („ Значение В= „, В);

выводит на экран дисплея текст

Значение В=

а затем числовое значение переменной В.

Для вывода целых и действительных чисел можно указыватьформаты в операторе WRITE. Формат указывается через двоеточие после переменной. Для действительных чисел формат со- стоит из двух величин. Первая величина указывает на общее количество позиций для вывода числового значения переменной: знак числа, количество цифр в целой части, точку и количество цифр в дробной части, второе – количество позиций для вывода дробной части. Например, опе- ратор WRITE (Y:5:2) осуществит вывод значения Y на экран и при этом для отображения значе- ния Y будет отведено пять позиций, из них две – на дробную часть.

Для вывода целых чисел количество позиций для дробной части не указывается. Напри- мер, если необходимо вывести значение целого

числа N=125, то оператор вывода будет иметь вид: WRITE ('N=', N:3),т.е. для вывода чис- лового значения N отведено 3 позиции.

Допускается использование оператора вывода writeln (элемент1, элемент2 ...), который сначала выводит значения переменных, а затемосуществляет переход на новую строку.

Оператор writeln обеспечивает пропуск одной строки и переход к началу новой строки. Текст программы решения задачи, схема которой приведена в примере имеет вид:

Program rabota;

Var a,b,c,p,t,ha,hb,hc:real;Begin

Writeln(„vvod а,b,c‟);

Readln(a,b,c);

p:=(a+b+c)/2;

t:=2\*sqrt(p\*(p-a)\*(p-b)\*(p-c));ha:=t/a; hb:=t/b;hc:=t/c;

writeln („ha=‟,ha:6:2,‟ hb=‟,hb:6:2, „ hc=‟, hc:6:2);end.

Исходными данными для решения задачи являются значениядлин сторон треугольника: a, b, c. Для ввода этих значений используется оператор Readln. В программе используется перемен- наяp для вычисления полупериметра и вспомогательная переменная t дляисключения повторе- ний.

Вычисленные значения высот ha, hb, hc выводятся с соответствующими именами пере- менных. Когда выводятся элементы списка вывода, пробелы между ними не устанавливаются автоматически, поэтому необходимо непосредственно их указать в списке вывода. Для улучше- ния внешнего вида данных, которые выводятся на экран дисплея, используется форматирован- ный вывод.

# Практические задания

**Задание 1.**

1. Изучить лекционный материал и методические рекомендациидля выполнения задания.
2. Разработать алгоритм для расчета заданных переменных.
3. Составить программу на алгоритмическом языке Pascal.
4. Выполнить отладку и тестирование программы накомпьютере.
5. Подготовить ответы на контрольные вопросы.

**Задание 2.** Составить блок-схему и программу вычисления значений функции при задан- ных значениях аргумента.

|  |  |
| --- | --- |
| 1) | 2) |
| 3) https://studfile.net/html/7651/383/html_R1ObxNOmue.yXfG/img-TMs1kO.png | 4) |
| 5) https://studfile.net/html/7651/383/html_R1ObxNOmue.yXfG/img-A1tNfL.png | 6) https://studfile.net/html/7651/383/html_R1ObxNOmue.yXfG/img-8RCtpJ.png |
| 7) | 8) |
| 9) | 10) |
| 11) | 12) |
| 13) https://studfile.net/html/7651/383/html_R1ObxNOmue.yXfG/img-gHmxkK.png | 14) https://studfile.net/html/7651/383/html_R1ObxNOmue.yXfG/img-BP0kGb.png |
| 15) | 16) |

|  |  |
| --- | --- |
| 17) https://studfile.net/html/7651/383/html_R1ObxNOmue.yXfG/img-aTTN1X.png | 18) |
| 19) | 20) |
| 21) | 22) |
| 23) | 24) |
| 25) | 26) |
| 27) | 28) |
| 29) | 30) https://studfile.net/html/7651/383/html_R1ObxNOmue.yXfG/img-YiN9X0.png |

# Контрольные вопросы:

1. Какими свойствами обладает алгоритм?
2. Какие способы используются для описания алгоритма?
3. Какие геометрические фигуры могут входить в блок-схемулинейного алго- ритма?
4. Каковы типовые структуры алгоритма?
5. Какова структура программы?
6. Как обозначается оператор присваивания?
7. Какой оператор позволяет осуществить ввод значенияпеременной во время выпол- нении программы?
8. Какой оператор позволяет вывести на экран текстовоесообщение?
9. Каким образом каждый оператор в программе отделяетсядруг от друга?

**Практическая работа №4.** Составление программ циклической структуры. Обработка одно- мерных массивов

Цель:

 закрепление знаний о программах циклической структуры, составление программы и ра- бота с ней.

# Краткие теоретические сведения

Алгоритм циклической структуры - это алгоритм, в котором происходит многократное по- вторение одного и того же участка программы. Такие повторяемые участки вычислительного процесса называются циклами. Программа циклической структуры содержит один или несколько циклов. Различают детерминированные циклы с заранее известным числом повторений и итера- ционные циклы, в которых число повторений заранее неизвестно. Изменяющаяся в цикле пере- менная называется параметром цикла.

Для организации цикла необходимо выполнить следующие действия:

1. задать перед циклом начальное значение параметра цикла;
2. изменять параметр перед каждым новым повторением цикла;
3. проверять условие повторения цикла;
4. управлять циклом, т.е. переходить к его началу, если он не закончен, или выходить из него по окончании.

В языке Паскаль существует 3 вида циклов: 1) цикл с параметром или цикл типа for, 2) цикл с предусловием или цикл типа while, 3) цикл с постусловием или цикл типа repeat ... until. В цикле типа for число повторений известно заранее, в циклах типа while и repeat ... until число повторений цикла заранее неизвестно, производится проверка условия повторения цикла: в цикле типа while

- перед циклом, в цикле типа repeat ... until - после его окончания.

В циклах типов for и while повторяющяся часть (тело цикла) состоит из одного оператора, если требуется выполнить в цикле несколько операторов, они заключаются в операторные скобки begin ... end, образуя составной оператор. В цикле типа repeat ... until тело цикла помещается между зарезервированными словами языка (лексемами) repeat и until, операторные скобки не тре- буются, в названии цикла его тело условно обозначается тремя точками.

С помощью цикла типа for удобно находить суммы, произведения, искать максимальные и минимальные значения и т.п. При нахождении суммы некоторой переменной, например S при- сваивается значение 0, затем в цикле к этой переменной прибавляется соответствующий член заданной последовательности. При нахождении произведения переменной присваивается значе- ние 1, затем в цикле эта переменная умножается на общий член последовательности.

Пример цикла типа for. Вычисление n чисел Фибоначчи: F1=1; F2=1;…; Fn=Fn-1+Fn-2 ,

например F3=F2+F1=1 + 1 = 2; F4 = 2 + 1 = 3 и т.д. program fib; {Нахождение чисел Фибоначчи}

var x, y, z, i, n : integer; begin

writeln (‘Введите n’); read (n); x:=1; y:=0;

for i:=1 to n do begin

z:=x ; x:=x+y ; y:=z ;

write (' ', x ) ; end;

end.

Пример цикла типа while. Cоставление таблицы функции y = a3/(a2+x2) для х, принадлежа- щих отрезку [-1; 1] с шагом 0.1. Так как параметр цикла типа for должен быть целочисленным, удобнее использовать цикл while, в котором значение х можно изменять при каждом шаге на х = 0.1

program cycl\_while;

uses crt; {вызов модуля Crt для управления режимом экрана} var

a, x, y : real ; i : integer; begin

clrscr ; {процедура очистки экрана из модуля Crt} writeln ('Введите а ' ); readln (a);

x:= -1.; writeln (‘ x ‘, ‘ y ’); while do x<1.05 begin

y:= sqr(a)\*a/ (a\*a+x\*x); writeln (x:6:2, y:8:4); x:=x+0.1

end end .

Условие х<1.05 соответствует каждому значению х плюс половина шага .

Пример цикла типа repeat ... until. Определить число n, при котором сумма квадратов нату- рального ряда чисел от 1 до n не превысит величину K, введенную с клавиатуры. Т.е.

S ≤ K, где S= = 12 + 2 2 + . . . + n 2

program sum\_sq; {Сумма квадратов натурального ряда } uses crt;

var k, s, n : integer; begin

clrscr ; writeln( 'Введите K' ); readln (k); s:=0; n:=1;

repeat

s :=s+n\*n; n := n+1; until s > k;

writeln ('N= ', n : 3, ' s= ' , s : 5 ); end.

Цикл повторяется до тех пор, пока условие записанное после ключевого слова until, будет ложным (не выполняется). Как только это условие выполнится, происходит выход из цикла. По- сле окончания цикла производится печать результата (оператор writeln).

Отметим, что цикл с предусловием (типа while) может не выполниться ни разу, цикл с по- стусловием repeat ... until выполнится по крайней мере 1 раз.

Когда число повторений цикла неизвестно заранее, применяются циклы с предусловием или с постусловием. Когда число повторений цикла известно заранее, как правило, применяется цикл типа for. Но любой цикл типа for можно заменить циклом с предусловием или постусло- вием.

# Практические задания

**Задания**

1. Разобрать и проанализировать приведенные выше программы с циклами типа for, while и repeat ... until
2. Составить алгоритм задачи: Вычислить сумму ряда, указанного в варианте задания для лю- бого значения N, введенного с клавиатуры.
3. Составить 3 варианта программ циклической структуры типа for, while и repeat, откомпили- ровать их, ввести исходные данные, сравнить полученные результаты.

# https://studfile.net/html/2706/304/html_w92_0iF4oi.rjWK/htmlconvd-eKH7np_html_54700e91ca617a6f.gifhttps://studfile.net/html/2706/304/html_w92_0iF4oi.rjWK/htmlconvd-eKH7np_html_c57eb89d39340ae.gifВарианты заданий

|  |  |
| --- | --- |
| 1. | . 2. . |
| 3. | . 4. . |
| 5. | . 6. . |
| 7. | . 8. . |

**Контрольные вопросы:**

1. Какой алгоритм является алгоритмом циклической структуры?
2. Типы циклов в языке Паскаль.
3. Цикл с параметром в языке Паскаль.
4. Циклы с предусловием и постусловием в языке Паскаль.
5. Какой из операторов цикла целесообразнее использовать для вашей задачи?
6. Какой цикл выполнится по крайней мере один

**Практическая работа №5**. Составление программ обработки одномерных массивов.

Цель:

* + приобретение практических навыков в составлении программ с массивами.

# Краткие теоретические сведения

При выполнении работ использоваться структурированный тип данных *массив*, который представляет собой фиксированное количество упорядоченных однотипных компонент, снаб- женных индексами. Он может быть *однономерным* и *многомерным*.

Тип-массив описывается в разделе описания типов следующим образом:

**type** <имя типа> = **аrrау**[<тип индекса(индексов)>] **of** <тип компонент>;

Размерность массива может быть любой, компоненты массива могут быть любого (в том числе и структурированного) типа, индекс (индексы) может (могут) быть только интервального или перечисляемого типа*.* При употреблении в качестве типа индекса типа *Integer* или *Word* можно исполь- зовать только его диапазон.

При описании типа индекса (индексов) можно использовать константы, которые должны быть определены до определения типа.

Определенный в разделе описания типов тип-массив можно использовать для описания переменных и типизированных констант. Тип-массив можно вводить непосредственно и при определении соответствующих переменных или типизированных констант.

При задании значений константе-массиву компоненты указываются в круглых скобках и разделяются запятыми, причем, если массив многомерный, внешние круглые скобки соответ- ствуют левому индексу, вложенные в них круглые скобки – следующему индексу и т. д. При этом все компоненты массива должны быть заполнены.

Доступ к компонентам массива осуществляется указанием имени массива, за которым в квадратных скобках помещается значение индекса (индексов) компоненты. В многомерных мас- сивах значения индексов перечисляются через запятую.

Для работы с массивом как с единым целым используется идентификатор массива без ука- зания индексов в квадратных скобках. Массив может участво- вать только в операциях отноше- ния («=», «<>») и в операторе присваивания.

Массивы, участвующие в этих действиях, должны быть идентичны по струк- туре, то есть иметь одинаковые базовые типы и типы индексов.

Задание значений переменной типа массив может осуществляться с кла- виатуры, путем генерирования случайным образом либо иным способом. Когда значения элементов задаются с клавиатуры или генерируются случайным обра- зом, необходима организация цикла, в котором последовательно происходит обращение к каждому элементу массива. Вывод значений элемен- тов массива на экран или в файл также происходит в цикле.

При написании программ необходимо следить за тем, чтобы значения ин- дексов не превы- шали границ, указанных при объявлении массива, так как вы- ход индекса за границы массива приводит к сбою в работе программы. Кон- троль значений индексов массивов можно организо- вать при помощи директивы компилятора {$R+}, которая приводит к проверке всех индексных выражений на соответствие их значений диапазону индекса.

Пример выполнения лабораторной работы

Для примера рассмотрим следующую задачу. Дан одномерный массив це- лых чисел. Уда- лить все отрицательные элементы, расположенные справа от максимального.

Прежде всего, определим алгоритм решения данной задачи. Вначале надо найти макси- мальный элемент среди элементов массива и запомнить его индекс. Затем, начиная со следую- щего за максимальным элементом, проверять элемен- ты массива на знак, и если число оказыва- ется отрицательным, то удаляем этот элемент из массива путем «сдвига» элементов массива влево и уменьшаем дли- ну массива. При этом следует заметить, что в памяти переменная типа массив будет занимать столько же места, как и до удаления элементов.

В графическом виде этот алгоритм будет блок-схему, представленную на рис. 5.1. Теперь пишем код программы.

Поскольку в программе требуется использование генератора случайных чисел, то необхо-

димо подключить стандартный модуль CRT, в котором описа- на функция генерации случайных чисел.

Uses crt;

В разделе описания типов определим тип-массив:

type arr=array[1..15] of integer;

В разделе описания переменных описываем необходимые переменные: var a:arr; {массив}

i,j,n: byte; {индексы, длина массива}

i\_max: byte; {индекс максимального элемента массива} max:integer; {максимальный эле- мент}

|  |  |  |
| --- | --- | --- |
|  | Информацио  нначнаылйо  блок |  |

i от 1 до n шаг 1 ai = случ. число Вывод ai

Генерация случайным образом элементов массива

нет

i от 2 до n шаг 1 ai >max

Нахождение максимального эле-

i = i\_max+1

max=a1; i\_max=1

да i<= n

max=ai; i\_max=i

да ai <0

да

нет

нет

Удаление от- рицательных элементов справа от мак-

aj = aj+1

n=n-1 i=i+;

i от 1 до n шаг 1

Рис. 5.1. Блок-схема алгоритма удаления из массива отрицателВьынвыохдэaлiементов, располо- женных справа от максимального элемента массива

25 конец

Далее описываем основное тело программы. Как отмечено на рисунке, алгоритм состоит из нескольких частей. Первая часть – генерация исходного массива:

begin writeln;

writeln(’ Автор – Иванов И.П., студент гр. ИСЭд-11’); writeln(’ Вариант № 100’);

writeln(’ Дан одномерный массив целых чисел.’);

writeln(’ Удалить все отрицательные элементы, расположенные справа от максимального

’);

writeln;

{$R+} {Включаем контроль значений индексов} clrscr; n:=15;

randomize;

writeln('Исходный массив:');

for i:=1 to n do {генерация элементов массива} begin a[i]:=-20+random(41); {в диапазоне [-20; 20]} write(a[i]:4);

end; writeln;

Генератор случайных чисел активизируется командой randomize. Далее в цикле случайным

образом задаются элементы массива. Функция random(41) возвращает случайное число в диапа- зоне от 0 до 40. Получает- ся, что минимально возможное значение элементов массива будет равно -20 (к -20 прибавить 0, сгенерированный функцией random), максимально воз- можное – 20 (-20 плюс число 40, сгенерированное функцией random). Сразу же выводим значение эле- мента массива на экран. В цикле все элементы будут выводиться в строчку, т. к. используется команда вывода write. После генера- ции и вывода всех элементов массива осуществляется пере- ход на новую строку.

Вторая часть алгоритма – поиск максимального элемента. Предположим, что первый эле- мент – максимальный, тогда запомним значение первого эле- мента в переменной max, а в пере- менную i\_max, где будет запоминаться поло- жение максимального элемента, запишем 1. Далее организуется цикл, в кото- ром ищется элемент, больший чем max.

max:=a[1]; i\_max:=1; for i:=2 to n do

if a[i]>max then begin

max:=a[i]; i\_max:=i; end;

Далее, начиная с элемента, следующим за максимальным, организуется цикл проверки эле-

ментов массива на знак (важно заметить, что в данном случае нужно использовать цикл while, т. к. длина массива будет изменяться, и, соот- ветственно, число итераций цикла также будет ме- няться). Если число оказыва- ется отрицательным, то удаляем этот элемент из массива, для чего организуется еще один цикл, в котором на место удаляемого записывается стоящий следом эле- мент.

Внутренний цикл сдвига можно проиллюстрировать на примере удаления седьмого эле- мента из массива, содержащего 10 элементов (рис. 5.2).

Исходный массив

a1

a2

a3

a4

a5

a6

a7

a9

a10

a8

Удаляемый элемент

1. я итерация цикла

a1

a2

a3

a4

a5

a6



a7 a8

a9

a10

результат

a1

a2

a3

a4

a5

a6

a8

a8

a9

a10

1. я итерация цикла

a8 a9

a1

a2

a3

a4

a5

a6

a8

a10

результат

a1

a2

a3

a4

a5

a6

a8

a9

a9

a10

1. я итерация цикла

a9 a10

a1

a2

a3

a4

a5

a6

a8

a9

результат

a1

a2

a3

a4

a5

a6

a8

a9

a10

a10

Рис. 5.2. Схема удаления элемента из массива

В программе удаление элементов, находящихся справа от максимального, будет выгля- деть следующим образом:

i:=i\_max; while i<=n do begin if a[i]<0 then begin

for j:=i to n-1 do a[j]:=a[j+1]; n:=n-1;

end else

i:=i+1;

end;

Еще одна тонкость данного фрагмента состоит в том, что переход к следую- щему элементу (i:=i+1) для проверки осуществляется только тогда, когда эле- мент не удовлетворяет условию. Если же i-й элемент окажется отрицательным, то после удаления на его месте окажется i+1-й эле- мент, который и нужно прове- рять на следующей итерации цикла.

Остается вывести результирующий массив на экран. writeln('Результат обработки:'); for i:=1 to n do

write(a[i]:4); readln;

{$R-} {Выключаем контроль значений индексов} end.

Ниже представлен полный текст программы. programlab5\_var100; Uses crt;

type arr=array[1..15] of integer; var a:arr; {массив} i,j,n: byte; {индексы, длина массива}

i\_max: byte; {индекс максимального элемента массива} max:integer; {максимальный эле- мент}

begin writeln;

writeln(’ Автор – Иванов И.П., студент гр. ИСЭд-11’); writeln(’ Вариант № 100’);

writeln(’ Дан одномерный массив целых чисел.’);

writeln(’ Удалить все отрицательные элементы, расположенные справа от максимального

’);

writeln;

{$R+} {Включение контроля значений индексов} clrscr; {Очистка экрана} n:=15;

randomize;

writeln('Исходный массив:');

for i:=1 to n do {генерация элементов массива} begin a[i]:=-20+random(41); {в диапазоне [-20; 20]} write(a[i]:4);

end; writeln; max:=a[1]; i\_max:=1;

for i:=2 to n do {поиск максимального элемента массива} if a[i]>max then begin

max:=a[i]; i\_max:=i;

end; i:=i\_max+1;

while i<=n do {цикл для проверки элементов на знак } begin

if a[i]<0 then {если элемент отрицательный, то} begin for j:=i to n-1 do {цикл удаления i-го элемента} a[j]:=a[j+1];

n:=n-1; {уменьшение длины массива} end else

i:=i+1; {иначе переход к следующему элементу} end;

writeln('Результат обработки:'); for i:=1 to n do write(a[i]:4); readln;

{$R-} {Выключение контроля значений индексов} end.

# Варианты заданий

1. Скорректировать массив A=(a1, а2, ..., аn), переписав в начало массива

группу, содержащую наибольшее число подряд идущих положительных элементов. Эле- менты массива вводить с клавиатуры.

1. В массиве A=(a1, а2, ..., аn) все элементы, равные нулю, поставить сразу по- сле мак- симального элемента данного массива. Элементы массива вводить с клавиатуры.
2. В массиве A=(a1, а2, ..., аn) все отрицательные элементы отправить в

«хвост» массива.

1. В массиве A=(a1, а2, ..., аn) удалить последнюю группу положительных эле- ментов. Группой называется подряд идущие элементы одного знака, число которых больше или равно 2.
2. В массиве A=(a1, а2, ..., аn) все положительные элементы, стоящие перед минималь- ным положительным элементом, переслать в «хвост» массива.
3. В массиве A=(a1, а2, ..., аn) удалить все подряд идущие отрицательные эле- менты, идущие вслед за минимальным элементом массива.
4. В массиве A=(a1, а2, ..., аn) удалить все отрицательные элементы, стоящие перед минимальным элементом массива.
5. В массиве A=(a1, а2, ..., аn) удалить все элементы, меньшие, чем элемент массива, расположенный слева от максимального.
6. В массиве A=(a1, а2, ..., аn) вставить новый элемент со значением Р вслед за наибольшим из отрицательных элементов этого массива.
7. В массиве A=(a1, а2, ..., аn) удалить все элементы, стоящие между мини- мальным положительным и максимальным отрицательным элементами.
8. В массиве A=(a1, а2, ..., аn) удалить все положительные элементы, имеющие четный порядковый номер, идущие после минимального элемента массива.
9. В массиве A=(a1, а2, ..., аn) все положительные элементы, начиная со второ- го по- ложительного, отправить в «хвост» массива.
10. В одномерном массиве A=(a1, а2, ..., аn) группу, содержащую наибольшее число рав- ных элементов, заменить на максимальный элемент этого масси- ва. После корректировки массив может содержать меньше элементов, чем прежде. Элементы массива вводить с клавиатуры.
11. В одномерном массиве A=(a1, а2, ..., аn) группу элементов, содержащую наибольшее число подряд идущих отрицательных элементов, переписать в

«хвост» массива. Элементы массива вводить с клавиатуры.

1. В одномерном массиве A=(a1, а2, ..., аn) все отрицательные элементы, имеющие не- четный порядковый номер, отправить в «хвост» массива, т. е. поместить на место последних элементов.
2. В одномерном массиве A=(a1, а2, ..., аn) все группы элементов, содержащие более 3-х подряд идущих отрицательных элементов, заменить на макси- мальный элемент. Элементы массива вводить с клавиатуры.
3. В одномерном массиве A=(a1, а2, ..., аn) все положительные элементы, имеющие четный порядковый номер, переписать в начало массива.
4. В одномерном массиве A=(a1, а2, ..., аn) группу, содержащую наибольшее число рав- ных элементов, заменить на максимальный элемент этого масси- ва. Элементы массива вводить с клавиатуры.
5. В одномерном массиве A=(a1, а2, ..., аn) удалить все отрицательные элемен- ты, рас- положенные между положительными.
6. В одномерном массиве A=(a1, а2, ..., аn) исключить из массива группу с наи- боль- шим числом подряд идущих положительных элементов. Элементы массива вводить с клавиа- туры.
7. В одномерном массиве A=(a1, а2, ..., аn) отрицательные элементы, имеющие четный порядковый номер, переписать в начало массива.
8. В одномерном массиве A=(a1, а2, ..., аn) удалить все равные элементы, оста- вив только один из данных групп равных. Элементы массива вводить с клавиатуры.
9. В одномерном массиве A=(a1, а2, ..., аn) группу из наибольшего числа под- ряд иду- щих нулей заменить на максимальный элемент массива. Элементы массива вводить с клавиа- туры.
10. В одномерном массиве A=(a1, а2, ..., аn) группу, содержащую наибольшее чис- ло под- ряд идущих положительных элементов, переписать в «хвост» массива.
11. В одномерном массиве A=(a1, а2, ..., аn) все положительные элементы, рас- поло- женные между отрицательными, поставить после минимального эле- мента массива.

**Практическая работа №6.** Составление программ обработки двумерных массивов

**Цель работы**: овладеть основными приемами работы с одномерными и двумерными мас- сивами, уметь различать в двумерном массиве обработку строк и столбцов, а также отличать нахождение первых и последних элементов последовательности, обладающих некоторым свой- ством.

# Общие сведения

*Скалярный тип* – простой тип данных. Скалярное данное неделимо. *Массив* – это струк- турированный тип данных. Массив состоит из нескольких элементов. Ко всему массиву можно обращаться по его имени. Можно обращаться к его элементу, но для этого надо задать индекс (индексы). Массивы бывают одномерные и многомерные. Для объявления массива необходимо задать типы его индексов и компонент:

# ARRAY [Тип индексов] OF <Тип компонент>;

*Тип компонент массива* – это просто тип данных, ассоциированный с каждой компонен- той массива. Тип компонент может быть любым REAL, INTEGER, CHAR, BOOLEAN, перечис- лимым, интервальным. В качестве компоненты массива может быть взят и тип массив.

Тип индекса должен быть одним из упорядоченных типов, т.е. любым скалярным типом, кроме REAL: INTEGER, CHAR, интервальный, перечислимый. Тип индекса определяет границы изменения индекса. Если сделана попытка использовать несуществующую компоненту, то воз- никает ошибка (ошибка неверного индекса).

# Одномерные массивы

Одномерный массив можно задать (объявить) двумя способами:

* 1. C помощью служебного слова TYPE описывается тип массива, а затем с помощью VAR вводится переменная этого типа.

# Общая форма записи

TYPE <тип массива> = ARRAY [тип индекса] OF <тип компонент>; VAR <переменная>: <тип массива>;

* 1. С помощью слова VAR сразу описывается переменная типа массив.

# Общая форма записи

VAR <переменная>: ARRAY [тип индекса] OF <тип компонент>;

Например, объявление массива из 100 элементов типа REAL можно осуществить двумя способами:

1. type R100 = array [1..100] of real; var A: R100;
2. var A: array [1..100] of real.

Здесь задан массив с именем А, и его элементы имеют имена: А[1],…,A[100]. Чаще всего для типа индекса используют интервальный тип на основе типов INTEGER и CHAR. Однако можно в качестве индексов брать перечислимый тип.

П р и м е р 1. Подсчет числа вхождений букв в текст определенной длины. program COUNTER;

var COUNT: array ['a'..'z'] of integer; CH: char; N: integer;

begin

for CH := 'a' to 'z' do COUNT [CH] := 0; N := 0;

repeat

read (CH); N := N + 1;

if (CH >= 'a') and (CH <= 'z') then COUNT [CH] := COUNT [CH] + 1;

until CH = '.';

for CH := 'a' to 'z' do

writeln (CH, COUNT [CH]:5);

end.

*Пояснение*. В этом примере тип индекса есть интервальный тип на базе типа CHAR, а тип компонент есть целое число. Таким образом, элементы массива – числа, а их индексы – буквы, т.е. число элементов массива равно 26 (по числу букв латинского алфавита).

Рассмотрим теперь случай, когда тип индекса задан перечислимым типом, а компоненты массива представлены компонентами интервального типа на базе типа INTEGER.

П р и м е р 2. Присваивание переменной с именем месяца числа дней этого месяца. program NUMBRDAY;

type MONAT = (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OKT, NOV, DEC);

var DAY : array [MONAT] of 28..31; T : MONAT;

begin

for T := JAN to DEC do case T of

JAN, MAR, MAY, JUL, AUG, OKT, DEC: DAY [T] := 31; APR, JUN, SEP, NOV: DAY [T] := 30;

FEB : DAY [T] := 28;

end; end.

# Многомерные массивы

Для определения позиции элемента в двумерном массиве необходимы два индекса. Любой двумерный массив есть матрица, а матрица есть таблица. Поэтому удобно описывать двумерные массивы путем указания границ изменения индексов (номеров) строк и столбцов.

Например, таблица символов M × N, где M – число строк и N – число столбцов, может быть описана:

var TAB : array [1..M, 1..N] of char;

|  |
| --- |
|  |
| **Общая форма записи**  VAR <имя> : ARRAY [тип индекса строки, тип индекса столбца] OF <тип компонент>; |
|  |

Однако двумерный массив можно интерпретировать как вектор-столбец, каждый элемент которого, в свою очередь, является одномерным массивом (вектор-строка). Этот подход к опре- делению двумерного массива влечет его описание с помощью двух строк, где первая содержит описание строки, а вторая – описание столбца:

type LINE = array [1..N] of char; STOLB = array [1..M] of LINE;

var TAB : STOLB.

Здесь TAB [I] – переменная типа LINE, а TAB [I][J] – переменная типа CHAR.

|  |
| --- |
|  |
| **Общая форма записи**  TYPE <тип строки> = ARRAY [тип индекса] OF <тип компонент>;  <тип столбца> = ARRAY [тип индекса] OF <тип строки>; VAR <переменная массива> : <тип столбца>; |
|  |

Эти два вида определения массивов задают и два способа обращения к элементам мас- сива: TAB [I, J] – в первом случае и TAB [I][J] – во втором. Вполне очевидно, что сказанное выше для двумерного массива распространяется и на массивы большей размерности. Например, опи- сание VAR CUBE : ARRAY [1..M, 1..N, 1..K] OF INTEGER определяет задание трехмерного мас- сива целых чисел.

# Примеры работы с массивами

Обработка массивов включает в себя, как правило, следующие компоненты: ввод массива (с клавиатуры или с помощью датчика случайных чисел), вывод полученного массива на экран и собственно его обработка. Все эти компоненты рекомендуется оформлять в виде отдельных процедур. При этом надо учитывать следующий фактор: если процедуре (или функции) будет передаваться массив, то надо объявить в ней этот массив как параметр с атрибутом VAR даже в том случае, если значение массива внутри процедуры не изменяется. Это нужно для того, чтобы не тратить времени и памяти на размещение внутри процедуры копии массива. Заметим, что па- раметр обязательно должен относиться к типу, имеющему имя.

П р и м е р. Сумма элементов таблицы над верхней диагональю. program SUMMA;

const M = 10; {число строк таблицы} N = 10; {число столбцов таблицы} type LINE = array [1..n] of integer; TAB = array [1..m] of LINE;

var s, i, j:integer; MAS: TAB;

procedure VVODMASSIV (var MAS: TAB); begin

for i := 1 to M do for j := 1 to N do

readln (MAS [i][j]); end;

procedure VIVODMASSIV (var MAS:TAB); begin

for i := 1 to M do begin

for j := 1 to N do

write (MAS [i][j]:4,' '); writeln; end;

end;

begin SUM := 0;

procedure OBRABOTKA (MAS: TAB; var SUM: integer);

for i := 1 to M do for j := 1 to N do

if j > i then SUM := SUM + MAS [i][j]; end;

begin

VVODMASSIV (MAS);

writeln ('исходный массив'); VIVODMASSIV (MAS); OBRABOTKA (MAS, s); writeln;

writeln ('сумма элементов = ',s); end.

Табличное представление информации одно из самых распространенных, поэтому мас- сивы широко применяются в прикладных программах.

Перед выполнением работы необходимо ознакомиться с теоретическим материалом по теме «Обработка массивов».

**Пример 1.** Составить программу, позволяющую в одномерном массиве, состоящем из *N* вещественных элементов, вычислить сумму положительных элементов.

Решение. При написании процедур ввода и вывода следует обратить внимание, что эле- менты – вещественные числа, поэтому необходимо позаботиться о верной обработке дробной части. Вычисление суммы оформим в виде функции с одним аргументом - массивом. Локаль- ными переменными функции будут индексная переменная *i* и дополнительная переменная *s* для хранения текущей суммы элементов. В начале тела функции обязательно обнуление *s*. Каждый элемент массива сравним с нулем, и, если значение положительно, добавим его к искомой сумме *s*. В конце функции запишем значение переменной *s* в результирующую переменную*.*

program massiv\_1; const N=10;

type mas=array [1..N] of Real; procedure Vvodmas(var A:mas); var i:Integer;

begin

for i:=1 to N do A[i]:=-50+Random(101)+random; end;

procedure Vivodmas(A:mas); var i:Integer;

begin

for i:=1 to N do Write(A[i]:8:2);

Writeln end;

function Summa(A:mas):real; var i: Integer; s:real;

begin s:=0;

for i:=1 to N do if A[i]>0 then s:=s+A[i]; Summa:=s;

end;

var A: mas; begin

Randomize; Vvodmas(A); Writeln('Исходный массив:'); Vivodmas(A); Writeln('Ответ: ', Summa(A):0:2);

Readln end.

**Пример 2.** В двумерном массиве, состоящем из целочисленных элементов, в каждом столбце поменять местами наибольший по модулю и последний не принадлежащий интер- валу *(a, b)* элементы массива.

Решение. Преобразования необходимо провести в каждом столбце массива, поэтому па- раметр внешнего цикла в процедуре обработки - номер столбца *j*, а вложенного – номер строки *i*.

Для перестановки двух элементов в столбце массива необходимо найти номера их строк *n1* и *n2*, а затем поменять местами значения элементов с использованием промежуточной переменной *p*.

Чтобы найти наибольший по модулю элемент столбца, введем дополнительную перемен- ную *max*, которая будет хранить максимальное по модулю значение в текущем столбце массива на данный момент. (Можно решить задачу без использования переменной *max.* Подумайте, как это сделать.)

Программа должна корректно работать с любыми входными данными, а значит и в тех случаях, когда некоторые или даже все столбцы массива содержат только элементы из интер- вала *(a, b)*, и обмен значений в некоторых столбцах или во всем массиве не нужен.

const n=10; m=7;

type mas=array [1..n,1..m] of Integer; procedure Vvodmas(var D:mas);

var i,j:Integer; begin

for i:=1 to n do for j:=1 to m do

D[i,j]:=-50+Random(101);

end;

procedure Vivodmas(D:mas); var i,j:Integer;

begin

for i:=1 to n do begin

for j:=1 to m do Write(D[i,j]:4); Writeln;

end; end;

procedure Obmen(a,b: real; var D:mas); var i,j,p,n1,n2,max: Integer;

begin

for j:=1 to m do begin

n1:=1; max:=abs(D[1,j]);{считаем первый элемент столбца наибольшим по модулю} for i:=2 to n do

if abs(D[i,j])>max then {обнаружен больший элемент} begin n1:=i; max:=abs(D[i,j]) end;

i:=n; {перебираем элементы столбца, начиная с последнего} while (i>=1)and (D[i,j]>a)and(D[i,j]<b) do i:=i-1;

n2:=i;

if n2<>0 then {если элемент, не принадлежащий интервалу (a,b), был найден} begin

p:=D[n1,j]; D[n1,j]:=D[n2,j]; D[n2,j]:=p; {обмен значений} end;

end; end;

var D: mas; a,b:Real; begin

Randomize; Vvodmas(D); Writeln('Исходный массив:'); Vivodmas(D);

Write('Введите через пробел концы интервала (a,b): '); Readln(a,b); Obmen(a,b,D);

Writeln('Ответ:'); Vivodmas(D);

Readln end.

# Варианты заданий

**Задание 1.** Составить программу, позволяющую в одномерном массиве, состоящем из *N*

вещественных элементов, вычислить:

1. сумму модулей отрицательных элементов массива;
2. количество элементов массива, не принадлежащих интервалу *(a, b)*;
3. наименьший из элементов массива, принадлежащих отрезку [*a*, *b*];
4. количество элементов массива, равных первому элементу;

**Задание 2.** В двумерном массиве, состоящем из целочисленных элементов, поменять ме-

стами:

1. в каждом столбце наибольший по модулю и последний положительный элементы;
2. в каждом столбце первый и последний отрицательные элементы;
3. в каждой строке наибольший и наименьший элементы;
4. в каждом столбце первый принадлежащий отрезку [*a*, *b*] и первый отрицательный эле-

менты;

# Дополнительные задания

* 1. Определить в одномерном массиве число соседств из двух чисел разного знака.
  2. Дан двумерный массив целых чисел. Поменять местами строку, содержащую максимум массива, со строкой, содержащей его минимум.

сива?

# Контрольные вопросы

1. Как описываются в языке Паскаль одномерный и двумерные массивы?
2. Может ли массив содержать разнотипные данные?
3. В каком порядке указываются индексы при обращении к элементам двумерного мас-
4. Привести пример массива, описание которого выглядит следующим образом: *var A:*

*array [1..3, 20..24] of real.*

1. Можно ли при обработке двумерных массивов использовать однократные циклы? Если да, то приведите примеры.
2. Если в одномерном массиве проверяется «похожесть» его первой и второй части, то в каких границах надо писать оператор *for* для «прохождения» по этому массиву?
3. Каким образом надо находить первый и последний элементы одномерного массива, обладающие некоторым свойством (отрицательный, наибольший, входящий в интервал и *пр.)?*

**Практическая работа №7**. Обработка двумерных массивов

Цель:

 приобретение практических навыков в составлении программ с массивами.

# Краткие теоретические сведения

Массивы - структурированный тип данных с элементами одного типа. Количество элемен- тов определяет размер массива. Например, массив составляют заработные платы сотрудников подразделения предприятия, здесь число элементов равно числу сотрудников; массив образуют набор чисел, их количество равно числу элементов массива. Номер элемента массива называется его индексом. Массив может иметь не один, а большее число индексов. Число индексов называ- ется размерностью массива, например, массив с двумя индексами называется двумерным масси- вом. Таким двумерным массивом является, в частности, матрица системы n линейных алгебраи- ческих уравнений с n неизвестными. В то же время столбец свободных членов этой системы яв- ляется одномерным массивом.

Массив должен быть описан либо в разделе переменных VAR, либо в разделе типов TYPE следующим образом:

TYPE <имя\_типа> = ARRAY [ti,t2. ,tn] OF <тип\_элементов\_массива>;

VAR <имя\_массива> : <имя\_типа> ; или

VAR<имя\_массива>: ARRAY [ti,t2,. ,tn]OF<тип\_элементов\_массива>;

где tl,. ,tn - тип индекса (перечислимый или интервальный);

Например:

1. VAR a: ARRAY [1..5] OF real;

Описан массив а действительных чисел, который состоит из 5 элементов.

1. TYPE t = ARRAY[1..3,1..4] OF integer; VAR b : t;

Описана матрица b из 3 строк и 4 столбцов, элементы которой являются целыми числами.

1. TYPE

tl = ARRAY [1..4] OF integer; t = ARRAY [1..5] OF t1; VAR d : t;

к : t1;

В начале описан тип одной строки t1, затем тип всей матрицы t через тип строки t1. В разделе переменных указано ,что d-двумерный массив размером (5,4), a k - одномерный массив k(4).

* + 1. Пример обработки одномерного массива

Дан одномерный массив MAS(12) из вещественных чисел. Найти наибольший элемент массива и его индекс.

program pr4\_1 ;

const n = 12; (\*константа n определяет размер массива в описании\*) type

m = array [1..n] of real ; (\* m – тип массива mas \*) var mas : m ;

i , num : integer; max : real; begin

for i:=l to n do (\*ввод элементов массива mas по 1 в строке\*) begin

writeln(‘введите элемент массива’, i) ; read (mas[i]); end;

num :=1; max := mas [1]; for i: =2 to n do

if mas [ i ] > max then begin max := mas [i ]; num := i ; end;

writeln; (\* вывод массива в строку\*)

for i:=l to n do write (mas [ i ]:5:1 ); writeln;

writeln (‘максимальный элемент = ‘, max:4:1,‘ его индекс=’ ,num ); end.

Переменная max сравнивается с элементами массива, и если элемент массива больше max, то переменной max присваивается значение элемента массива, а переменной num - индекс этого элемента. По окончании цикла переменная max будет иметь значение, равное максимальному элементу массива, а переменная num - значение индекса этого элемента.

Как правило, при обработке многомерных массивов используются вложенные циклы, т.е. цикл по столбцам располагается внутри цикла по строкам.

2.1. Пример обработки многомерного массива

Дана матрица A(3,4), и вектор B (4), состоящие из целых чисел . Умножить матрицу А на вектор В .

program pr4-2 ; const m=3; n=4; var

a : array [ l .. m, 1 .. n ] of integer; (\* описание матрицы \*)

b : array [ 1 .. n ] of integer; (\* описание вектора \*)

c : array [ 1 .. m ] of integer; ( \* описание С \*) i, j: integer;

begin

for i:=l to m do (\* ввод матрицы \*) begin

writeln (‘введите элементы ‘, i , ‘-той строки’); for j:=1 to n do read (a [i, j] ); writeln;

end;

writeln (‘введите элементы вектора’); for j:=1 to n do (\* ввод вектора \*) read (b[ j]); writeln;

for i:=l to m do

begin

c [ i ]:=0; for j:=l to n do c[i] := c[ i ]+ a[i , j]\* b[j]; end;

for i:=l to m do (\*форматный вывод матрицы \*)

begin

for j:=1 to n do write (a [i, j]: 4 ); writeln; end;

for j:=l to n do write (b [ j ] :4); (\* вывод массива B \*) writeln ;

for i:=l to m do write (c [ i ] :4); (\* вывод массива С \*) readln;

end.

В программе элементы матрицы вводятся по строкам по одному с подтверждением клави- шей Enter. А выводятся в общепринятом виде: каждая строка матрицы с новой строки экрана (цикл i по строкам внешний, а цикл j – внутренний).

# Практические задания

**Задание 1.**

1. Набрать и откомпилировать приведенные выше программы, исправить выявленные ошибки. Ввести элементы массива, убедиться в правильности выполнения программ.
2. Составить и выполнить программы с применением массивов согласно вариантам зада-

ний .

# Задание 2.

Варианты задания 1. Ввести массив А из 10 элементов

1. Найти наибольший элемент и переставить его с первым элементом. Преобразованный массив вывести.
2. Найти наименьший элемент и переставить его с последним элементом. Преобразован- ный массив вывести.
3. Найти произведение положительных элементов и вывести его на экран.
4. Найти произведение отрицательных элементов и вывести его на экран.
5. Найти сумму положительных элементов и вывести ее на экран.
6. Найти сумму отрицательных элементов и вывести ее на экран.
7. Найти сумму элементов, больших 3 и меньших 8 и вывести ее на экран.
8. Найти сумму элементов, меньших по модулю 5 и вывести ее на экран.

# Задание 3.

Варианты задания 2

1. Даны матрица A размером m\*n и вектор В размером m. Записать на главную диагональ элементы вектора, а в вектор - элементы главной диагонали.
2. Выбрать максимальный элемент матрицы С (размер m\*n), элементы четных строк раз- делить на максимальный элемент, а к элементам нечетных прибавить максимальный элемент .
3. Найти минимальный элемент матрицы С (размер m\*n), и поменять его местами с пер- вым элементом.
4. Дана матрица Е размером m\*n. Вычислить суммы элементов каждого столбца. Опреде- лить наибольшее значение этих сумм и номер соответствующего столбца.
5. В матрице К размером m\*n найти в каждом столбце произведение отрицательных эле- ментов и количество нулевых элементов в матрице.
6. Даны две матрицы А и В одинаковой размерности m\*n. Получить матрицу C = max (a i j, b i j ), и матрицу D = min (a i j, b i j).
7. Дана матрица Р размером m\*n. Найти сумму минимальных элементов каждого столбца матрицы.
8. Даны две матрицы: А размером m\*k и В размером k\*n. Получить матрицу С=A\*В.

# Контрольные вопросы:

1. Что такое массив? индекс элемента массива?
2. В каких пределах изменяются индексы элементов массива?
3. Как ввести и вывести элементы вектора в строку и в столбец?
4. Способы описания массивов.
5. Вывод матрицы по строкам и по столбцам

**Практическая работа №8.** Работа со строками. Цель:

 изучение символьных типов данных CHAR и STRING и операций над ними.

# Краткие теоретические сведения

Наряду с числовой информацией в Паскале используется алфавитно-цифровая или сим- вольная информация, которая включает в себя заглавные, строчные буквы, цифры от 0 до 9 и вспомогательные символы. Для описания символьных переменных используется тип данных CHAR или STRING.

# Тип данных CHAR

Каждая переменная символьного типа может принимать значение только одного символа. Все символы упорядочены в соответствии с принятым в ЭВМ коде (например ASCII). При этом порядковый номер символов называется кодом (например, код латинского символа 'А ' равен 65; символа '3' равен 51).

Для символьных данных не определены никакие арифметические операции, но они могут сравниваться по своим кодам, участвовать в чтении, печати, операторах присваивания. Суще- ствуют две стандартные функции преобразования :

* + 1. ORD (C) принимает значение кода символа С;
    2. значение функции CHR(I) является символ с кодом Например:ORD('А ')=65 CHR(ORD(C))=C ; CHR (65) = A ;

Строка - это последовательность символов. Строку можно представить как массив, эле- менты которого имеют тип CHАR. Например:

BUK: array[1..17] of char;

Массив BUK-массив символов, который содержит 17 символов. Если символов меньше, то строка дополняется пробелами справа. В противном случае возникает ошибка несоответствия типов. Так как массивы символов являются обычными массивами с элементами типа CHAR, они обладают всеми свойствами массивов.

**Пример**: Из набора 10 любых символов напечатать только заглавные английские буквы и их коды.

program lr2;

type sl =array [1..10] of char;

var s: sl; {описание массива символов} i: integer;

begin

writeln ('введите 10 символов'):

for i:=l to 10 do readln (s[i]); {ввод массива} for i:=l to 10 do

if (s[i]>=’A’) and (s[i]<= ‘Z’) then

writeln (‘Символ :’, s[i], ‘ его код =’, ord (s[i]); readln;

end.

# Тип данных STRING

В Турбо Паскале предусмотрен тип данных STRING. Переменная типа STRING может принимать значения переменной длины. Максимально возможная длина переменной 255 симво- лов.

Например:

str: STRING[200];

ow: STRING[10];

В скобках указывается максимальная длина для данной переменной. Для ввода значений типа STRING необходимо использовать READLN, а не READ. За один раз может быть введена только одна строка. Две строки можно сравнивать, используя операции отношения (сначала сравниваются самые левые символы, если они равны, то сравниваются следующие). Для работы

с переменными типа STRING используют следующие стандартные процедуры и функции:

* + 1. Функция LENGTH

C:=LENGTH(str); Переменной С будет присвоено целое значение, показывающее коли- чествo символов в строковой переменной str .

* + 1. Функция СОNCAТ - сцепление строк в порядке их перечисления. str:=CONCAT(st1,st2,...,stN);str-переменная типа STRING, состоящая из строк st1,...,stN.
    2. Функция POS

P:=POS (st1, st2); Р-целое число, показывающее номер позиции, с которой начинается строка st1 в строкe st2.

* + 1. Функция COPY

S1:=COPY( str, I, J); Sl-символьная подстрока, выделенная из строки str с позиции I, дли- ной J символов.

* + 1. Процедура DELETE(Str, I, J);

Из строки str удаляется J символов, начиная с I позиции.

* + 1. Процедура INSERT(Str1, Str2, I);

Строка Str1 вставляется с I позиции в строку Str2.

* + 1. Процедура STR (V, S1);

Числовое значение переменной V преобразуется в строку символов и записывается в строку S1.

7) Процедура VAL (S1, V, C);

Строковое выражение S1 преобразуется в величину целочисленного или вещественного типа и записывается в переменной V . Если при этом ошибок не обнаруживается, то С будет равно 0 . В противном случае значение С будет равно номеру позиции первого ошибочного сим- вола и V будет неопределено. Строка S1 не должна содержать незначащих пробелов, переменная V может быть целой или вещественной, а переменная С - только целой .

**Пример**: Подсчитать количество слов во введенной с клавиатуры строке. program lr2;

var

s: string[30];

kol, i, n: integer;

begin

writeln ('введите строку'); readln (s); kol:=0; {счетчик количества слов}

n:= length(s); {определяем длину введенного текста} s:= concat(' ',s); {добавляем пробел к первому слову}

for i:=1 to n do

if (copy (s,i,1)=' ') and (copy (s,i+1,1)<>' ')

then kol := kol+1;{подсчет количества слов} writeln (s,' количество слов= ', kol);

readln end.

# Практические задания

**Задание**

1. Набрать и откомпилировать приведенные выше программы, исправить выявленные ошибки. Ввести разные исходные данные, убедиться в правильности выполнения программ.
2. Составить программы с использованием символьных данных согласно вариантам зада- ний, откомпилировать их, проверить полученные результаты.

# Задание 2.

Варианты заданий. Текст вводится с клавиатуры в символьную переменную. Исходный

текст и результаты распечатать.

1. Вывести на печать список слов, имеющих приставку (несколько букв), задаваемую с терминала.
2. Раздвинуть заданный текст, вставив введенную с клавиатуры последовательность символов после 1-го символа каждого слова.
3. В заданном тексте слова разделены запятыми. Напечатать список слов, начинаю- щихся с символа, введенного с клавиатуры.
4. Определить наличие слов в заданном тексте, содержащих сочетание символов, за- даваемое с экрана.
5. Из заданной последовательности слов удалить слова, содержащие числа.
6. Каждое слово текста преобразовать таким образом, чтобы оно читалось слева направо.
7. Подсчитать количество слов, разделенных запятыми, содержащих k гласных букв (k-задается с экрана).
8. Из введенного текста сначала распечатать слова, заканчивающиеся на согласную букву, а потом на гласную букву.

# Контрольные вопросы:

* 1. Описание переменных символьного типа.
  2. Функции преобразования переменных символьного типа.
  3. Отличие типов данных STRING и CHAR.
  4. Приведите конкретные примеры использования каждой функции и процедуры.

**Практическая работа №9.** Работа с данными типа множество

Цель:

 познакомить с понятием "множество" в языке программирования Pascal; выработать навыки работы со структурой данных множество.

# Практические задания

Перед выполнением работы необходимо ознакомится с правилами описания и использова- ния переменных типа множество, типизированных констант типа множество, переменных, задан- ных перечислением, изучить допустимые операции над переменными этих типов.

Понятие множества в языке ПАСКАЛЬ основывается на математическом представлении о множествах: это ограниченная совокупность различных элементов. Для построения конкретного множественного типа используется перечисляемый или интервальный тип данных. Тип элемен- тов, составляющих множество, называется базовым типом.

Множественный тип описывается с помощью служебных слов Set of, например: type M= Set of B;

Здесь М - множественный тип, В - базовый тип. Пример описания переменной множественного типа:

type

M= Set of 'A'..'D';

var

MS : M;

Принадлежность переменных к множественному типу может быть определена прямо в раз- деле описания переменных:

var

C : Set of 0..7;

Константы множественного типа записываются в виде заключенной в квадратные скобки последовательности элементов или интервалов базового типа, разделенных запятыми, например:

['A', 'C'] [0, 2, 7] [3, 7, 11..14].

Константа вида [] означает пустое подмножество.

Множество включает в себя набор элементов базового типа, все подмножества данного множества, а также пустое подмножество. Если базовый тип, на котором строится множество, имеет К элементов, то число подмножеств, входящих в это множество, равно 2 в степени К. Пусть имеется переменная Р интервального типа:

var

P : 1..3;

Эта переменная может принимать три различных значения - либо 1, либо 2, либо 3. Пере- менная Т множественного типа

var

T : Set of 1..3;

может принимать восемь различных значений: [ ] [1,2]

[1] [1,3]

[2] [2,3]

[3] [1,2,3]

Порядок перечисления элементов базового типа в константах безразличен.

Значение переменной множественного типа может быть задано конструкцией вида [T], где T - переменная базового типа.

К переменным и константам множественного типа применимы операции присваивания (:=), объединения(+), пересечения(\*) и вычитания(-):

['A','B'] + ['A','D'] даст ['A','B','D']

['A'] \* ['A','B','C'] даст ['A']

['A','B','C'] - ['A','B'] даст ['C'].

Результат выполнения этих операций есть величина множественного типа.

К множественным величинам применимы операции: тождественность (=), нетождествен- ность (<>), содержится в (<=), содержит (>=). Результат выполнения этих операций имеет логи- ческий тип, например:

['A','B'] = ['A','C'] даст FALSE

['A','B'] <> ['A','C'] даст TRUE

['B'] <= ['B','C'] даст TRUE

['C','D'] >= ['A'] даст FALSE.

Кроме этих операций для работы с величинами множественного типа в языке ПАСКАЛЬ используется операция in проверяющая принадлежность элемента базового типа, стоящего слева от знака операции, множеству, стоящему справа от знака операции. Результат выполнения этой операции - булевский. Операция проверки принадлежности элемента множеству часто ис- пользуется вместо операций отношения, например:

A in ['A', 'B'] даст TRUE, 2 in [1, 3, 6] даст FALSE.

При использовании в программах данных множественного типа выполнение операций про- исходит над битовыми строками данных. Каждому значению множественного типа в памяти ЭВМ соответствует один двоичный разряд. Например, множество

['A','B','C','D']

представлено в памяти ЭВМ битовой строкой 1 1 1 1.

Подмножества этого множества представлены строками: ['A','B','D'] 1 1 0 1

['B','C'] 0 1 1 0

['D'] 0 0 0 1

Величины множественного типа не могут быть элементами списка ввода - вывода.

В каждой конкретной реализации транслятора с языка ПАСКАЛЬ количество элементов базового типа, на котором строится множество, ограничено. В TURBO PASCAL количество ба- зовых элементов не должно превышать 256.

Инициализация величин множественного типа производится с помощью типизированных констант:

const seLit: Set of 'A'..'D'= [];

Проиллюстрируем применение данных множественного типа на примере.

Пример. Составить программу, которая вырабатывает и выводит на экран дисплея наборы случайных чисел для игры в "Спортлото 5 из 36".

Для заполнения каждой карточки спортлото необходимо получить набор из пяти псевдо- случайных чисел. К этим числам предъявляются два требования:

числа должны находиться в диапазоне 1..36; числа не должны повторяться.

Program Lotto; var

nb, k: Set of 1..36; kol, l, i, n: Integer;

begin

Randomize; WriteLn('ВВЕДИ kol'); ReadLn(kol); nb:=[1..36];

for i:=1 to kol do begin

k:=[];

for l:=1 to 5 do begin

repeat n:=Random(36)

until (n in nb) and not (n in k); k:=k+[n];

Write(n:4) end;

WriteLn end

end.

Пример

Пример1: Дан текст. Определить каких букв больше - гласных или согласных.

Этапы решения задачи:

1. Составим блок схему программы

Опишем подробнее блок "Подсчитываем количество гласных и согласных букв" Рассмотрим блок "Печатаем соответствующее сообщение"

Запишем блок-схему целиком

1. Переведем алгоритм на язык Паскаль program example1;

const glasn=['а','е','и','о','у','ы','э','ю','я'];

soglas=['б','в','г','д','ж','з','й','л','м',

'н','р','к','п','с','т','ф','х','ц','ч','ш','щ'];

var

st: string; g,s,i:integer;

begin

write('Введите строку> '); readln(st); g:=0; s:=0;

for i:= 1 to length(st) do

if st[i] in glasn then inc(g) else if st[i] in soglas then inc(s); if g> s then writeln('Гласных больше')

else if g< s then writeln('Согласных больше')

else writeln('Согласных и гласных букв поровну'); readln;

end.

# Практические задания:

Примечание: Гласные буквы - а,е,и,о,у,ы,э,ю,я (ё обычно не входит в литерный тип); согласные

* все остальные буквы, кроме ь, ъ; звонкие согласные - б,в,г,д,ж,з,й,л,м,н,р; глухие согласные - к,п,с,т,ф,х,ц,ч,ш,щ.

# Задание 1.

Дан текст из строчных латинских букв, за которым следует точка. Напечатать:

* + первые вхождения букв в текст, сохраняя их взаимный исходный порядок;
  + все буквы, входящие в текст не менее двух раз;
  + все буквы, входящие в текст по одному разу.

# Задание 2.

Дана непустая последовательность слов из строчных русских букв; между соседними словами - запятая, за последним словом - точка. Напечатать в алфавитном порядке:

все гласные буквы, которые входят в каждое слово; все согласные буквы, которые не входят ни в одно слово;

все звонкие согласные буквы, которые входят хотя бы в одно слово; все глухие соглас- ные буквы, которые не входят хотя бы в одно слово;

все согласные буквы, которые входят только в одно слово; все глухие согласные буквы, которые не входят только в одно слово;

все звонкие согласные буквы, которые входят более чем в одно слово; все гласные буквы, которые не входят более чем в одно слово;

все звонкие согласные буквы, которые входят в каждое нечетное слово и не входят ни в одно четное слово; все глухие согласные буквы, которые входят в каждое нечетное слово и не входят хотя бы в одно четное слово.

# Задание 3.

Имеются три множества символьного типа, которые заданы своими конструкторами: Y1=['A','B','D','R','H']

Y2=['R','A','H','D'] Y3=['A','R'].

Сформировать новое множество .

Предусмотреть формирование исходных множеств с клавиатуры.

**Задание 4.** Подсчитать общее количество цифр и знаков '+', '-', и '\*', входящих в строку s.

# Контрольные вопросы:

1. Что такое множество, как оно описывается в языке Pascal?
2. Как определить новый тип данных с использованием перечисления?
3. Как описываются типизированные константы типа множество?
4. Как осуществляется ввод-вывод значений переменных типа множество?
5. Какие типы данных используются в качестве базовых при объявлении типа мно- жество?
6. Какие операции определены над множествами?
7. Какие операции допустимы над переменными, заданными перечислением?
8. Чем похожи и чем отличаются множества и массивы?
9. Какое значение у выражений: а) x in [x]; б) [ ] <= [x,y,z]; в) [x]<>[x,x,x]

**Практическая работа №10.** Файлы последовательного доступа

Цель:

 изучение файловых типов данных, приобретение практических навыков создания и обра- ботки файлов.

# Краткие теоретические сведения

**Работа с файлами**

Информация, вводимая с клавиатуры или обрабатываемая с помощью программных средств Бейсика, размещается в оперативной памяти компьютера.

Алгоритм, набранный в Бейсике, может быть сохранен на диске в виде файла.

**Файл** — это поименованная область на магнитном или лазерном диске. В файлах могут содержаться тексты, графические и видеоизображения, звуки и музыка, таблицы и базы, данные программы, данные для этих программ.

**Требования к имени файла**

* имя не должно быть больше чем 8 символов;
* имя может состоять из букв латинского алфавита, цифр и символов, например, \_, -, (,),

$ и некоторых других.

* в имени файла **запрещены** символы <Пробел>, \*, точка, запятая, кавычки, двоеточие. Впрочем, злоупотреблять специальными символами не стоит — букв и цифр вполне хва-

тает.

**Расширение файла**

Файл имеет расширение.

**Расширение имени файла** (англ. *filename extension*, часто говорят просто расширение файла или расширение) — последовательность символов, добавляемых к имени файла и предназначен- ных для идентификации типа (формата) файла. Расширение имеет длину не более трех символов, указывается через точку после имени.

|  |  |
| --- | --- |
| **Расширение файла** | **Описание формата файла** |
| \*.aif, \*.aifc,  \*.aiff | Файлы аудиоданных |
| \*.asm | Исходный текст программы на Ассемблере. |
| \*.avi | Основной формат видеоизображений |
| \*.bas | Текст программы на языке алгоритмическом языке Basic и различных его вариантах (GWBasic, TurboBasic, QuickBasic) |
| \*.bmp | Формат графических файлов (растровая графика). |
| \*.com | Исполняемый файл в двоичном коде |
| \*.cpp | Текст программы на языке C++ |
| \*.doc | Файл с документами или продукт работы текстового процессора Microsoft Word for Windows |
| \*.dot | Шаблон документа текстового процессора MS Word |
| \*.exe | Это всегда исполняемый бинарный файл |
| \*.gif | (Graphics Interchange Format). Растровый графический формат фирмы CompuServe |
| \*.htm, \*.html | Специальный файл текстового типа, написанный на Hyper Text Markup Language |
| \*.mdb | Файл баз данных Microsoft Access |

|  |  |
| --- | --- |
| \*.mov - | Формат хранения видео и аудио |
| \*.pas - | Текст программы на языке Pascal |
| \*.ppt | Файл с презентацией Microsoft PowerPoint |
| \*.sys - | Системные файлы ядра DOS IO.sys и MSDOS.sys. |
| \*.txt | Текстовой файл, созданный в блокноте |
| \*.xls | Файл работы табличного процессора Microsoft Excel |
| \*.zip - | Файл архива сжатого архиваторами |

В файлах вы можете хранить как исходные данные для обработки, так и результаты работы программы.

Для работы в Бейсике необходимы файлы, хранящие однородные по типу или структуре све- дения, о каких-либо объектах. Набор данных о каком-либо одном объекте называется **записью.**

Файл может быть пустым, т. е. содержать 0 байт информации, но имя файла и символ конца файла будут присутствовать. (**Байт** - единица измерения количества информации, объема памяти и емкости запоминающего устройства. По умолчанию байт считается равным 8 битам).

Записи могут содержать данные разных типов, но должны быть обязательно одинаковы по структуре, например:

"Запорожец", "4067 ЛДЕ", "1972", "100$"

"ГАЗ-34", "6666 ЛАА", 1989, "3500$"

# В соответствии со способом доступа к файлам они делятся на два вида.

1. Файл с последовательным доступом;
2. Файл с прямым доступом.

**Файлы последовательного доступа** наиболее просты как в организации, так и в работе с ними. Записи обрабатываются последовательно одна за другой.

Информация в таких файлах хранится в виде текста в кодах ASCII. Такие файлы легко про- смотреть на экране, используя любой простейший редактор, или в самом Бейсике.

Простота — хорошо, а последовательность в данном случае — плохо. Если информация об интересующем объекте упорядочена в файле по алфавиту, то придется перебирать практически весь файл, чтобы добраться до нужной записи. Отсюда, при большом информационном объеме файла обработка его замедляется.

**Файлы прямого доступа** хранят информацию в специальном формате, в котором каждая за- пись занимает строго фиксированную одинаковую с остальными длину. Такие файлы занимают на диске больше места, чем файлы последовательного доступа, но скорость работы с ними зна- чительно выше.

**Операции над файлами**

Независимо от того, какие действия происходят с информацией, хранящейся в файле, про- изводятся следующие обязательные операции:

* 1. открытие файла;
  2. чтение и запись обрабатываемых данных;

В Паскале определены текстовые файлы, типизированные и нетипизированные. Файл, не содержащий ни одного элемента, называется пустым. Создается файл путем добавления новых записей в конец первоначально пустого файла. Длина файла, т.е. количество элементов, не зада- ется при определении файла.

Все файлы должны быть описаны в программе либо в разделе переменных VAR, либо в разделе типов TYPE. Под чтением файла понимают ввод данных из внешнего файла, находяще- гося на диске, в оперативную память машины. Запись в файл - вывод результатов работы про- граммы из оперативной памяти на диск в файл.

Работа с файлами выполняется следующими процедурами:

Assign – устанавливает связь между именем файла в программе (файловой переменной) и физическим именем файла, принятым в ОС.

Reset - открывает существующий файл для чтения.

Rewrite – создает и открывает новый файл для записи на внешнем устройстве (если файл ранее существовал, вся предыдущая информация из него стирается).

Close - закрывает открытый файл.

Для определения конца файла используется стандартная встроенная функция EOF (фай- ловая переменная), которая принимает значение True, если достигнут конец файла, и значение False в противном случае.

# Текстовые файлы

Текстовые файлы – файлы на диске, состоящие из символов ASCII. Для разделения строк используются символы «конец строки». Текстовые файлы являются файлами с последователь- ным доступом. В любой момент времени доступна только одна запись файла. Другие записи ста- новятся доступными лишь в результате последовательного продвижения по файлу. Текстовые файлы внутренне разделены на строки, длины которых различны. Для разделения строк исполь- зуется специальный маркер конца строки. Объявляются текстовые файлы переменной типа text. Обрабатывать их можно только последовательно и с помощью процедур и функций:

Readln (f , st )- чтение строки st из файла f и переход на начало следующей ; Writeln (f, st )- запись строки st в файл f и маркера конца строки ;

Append (f ) - процедура, открывающая файл f для добавления строк в конец файла;

Eoln (st )- логическая функция, результат выполнения которой равен TRUE, если достигнут маркер конца строки st.

**Пример 1.** Создать текстовый файл, в который записать 3 предложения. Прочитать этот файл, вывести его содержимое на экран. Определить длину каждого предложения.

Program File\_text; var

f1 : text; st : string; n: byte;

begin

assign (f1, 'file1.txt'); {связать с файлом file1.txt файловую переменную f1 } rewrite (f1); { создать новый файл с именем file1.txt }

writeln ( f1, 'Очень полезно изучать'); { записать предложения в файл} writeln ( f1, ' всем студентам ');

writeln (f1, ' язык Pascal ');

close (f1); { закрыть файл для записи }

reset (f1); { открыть файл для чтения }

while not eof (f1) do { пока не конец файла f1} begin

readln (f1, st); {читаем строку из файла f1 } writeln(st); { выводим на экран }

n:= length (st); {определяем длину строки } writeln (' длина =',n);

end;

close (f1); { закрыть файл для чтения} end .

# Практические задания

**Задание**

1. Разобрать и проанализировать приведенные программы.
2. Используя подпрограммы, создать внешний файл из 7 записей, прочитать создан- ный файл и, применяя режим прямого доступа, выполнить задания по своему варианту.

# Контрольные вопросы:

1. Чем отличается файл от массива?
2. Особенности организации текстовых файлов.
3. Что понимается под чтением, и что под записью в файл?
4. Типы доступа к файлам.
5. Назначение процедуры SEEK.
6. Привести пример корректировки К-той записи.
7. Как определить размер файла?

**Практическая работа №11.** Типизированные файлы

Цель:

 изучение файловых типов данных, приобретение практических навыков создания и обра- ботки файлов.

# Краткие теоретические сведения Типизированные файлы

Типизированные файлы – это файлы, состоящие из нумерованной последовательности объектов (записей) любого типа. С такими файлами можно работать в режиме прямого доступа, при котором выполняется непосредственное обращение к любой записи файла. Каждая запись файла имеет свой номер, начиная с 0 и т.д.

Процедуры и функции обработки файлов:

1. Write и Read- записывают и читают информацию из указанного файла и переме- щают указатель файла к следующей записи.
2. Seek (файловая переменная, номер записи); процедура перемещения указателя на запись файла с заданным номером .
3. Truncate (файловая переменная); процедура, усекающая файл по текущей позиции указателя файла, т.е. все записи, находящиеся после указателя файла, удаляются.
4. Функция Filesize (файловая переменная); имеет тип Integer и определяет размер файла, т.е. число записей.
5. Функция Filepos (файловая переменная); имеет тип Integer и возвращает текущую позицию указателя файла.

Для добавления записей в конец файла используются процедуры: Readln (a );

Seek (f, filesize (f));

Write (f, a);

При этом указатель устанавливается за конец файла, т.к. нумерация записей начинается с нуля. После чего с помощью Write можно добавлять записи. Открывать файл можно только про- цедурой Reset (f).

Для того, чтобы в режиме произвольного доступа считать, а затем изменить значение за- писи, следует выполнить два вызова процедуры Seek.Один вызов перед операцией Read, а другой

- перед операцией Write (т.к. Read после чтения записи переместит указатель к следующей за- писи).

**Пример:** Cоздать файл из списка 10 студентов с их оценками ( номер, Ф.И.О. и три оценки). Вывести его содержимое на экран, изменить фамилию студента с номером, введенным с клавиатуры, заново прочитать файл.

Program lab6;

Type

wed = record {Тип wed включает 3 поля: n, fio, bal} n : byte ; fio : string[15] ;

bal : array [1..3] of byte; {Поле bal – массив из 3 оценок } end;

Var spisok : wed ; {Запись spicok типа wed} sp : file of wed; {Файл записей типа wed}

procedure vvod; { процедура создания файла} var i,j:byte;

begin

{ оператор assing находится в основной прграмме } rewrite ( sp); {открытие файла для записи} with spisok do

For i:=1 to 10 do begin n:=i;

writeln (' Введите фамилию - ', i ); readln (fio);

writeln (' Введите 3 оценки ', fio ); For j:= 1 to 3 do readln ( bal [j] ); write (sp , spisok); { запись в файл информации о студенте}

end;

close (sp); { закрытие файла для записи } end;

procedure print; { процедура чтения и печати всего файла } var j : byte;

begin

reset ( sp); {открытие файла для чтения} writeln (‘ Список студентов: ‘);

while not eof (sp) do with spisok do

begin

Read (sp, spisok); {чтение данных из файла} write (n,' ',fio); {вывод записи на экран} For j:= 1 to 3 do write (' ', bal [j] );

writeln ; end;

readln; close (sp) ; end;

procedure work; var num: integer; begin

reset ( sp); {открытие файла для чтения} writeln ('номер= '); readln (num);

seek (sp, num-1); {поиск записи с указанным номером (нумерация записей с 0)} read (sp,spisok);{чтение и перемещение указателя к след. записи}

write ('fio='); writeln (spisok.fio);

seek (sp,filepos(sp)-1); {возвращение к изменяемой записи } writeln (‘ Введите новую фамилию’ ); readln (spisok.fio); write (sp, spisok); {запись в файл измененной записи}

close (sp); end;

begin {начало основной программы}

assign (sp,'Vedom.DAT'); {связать файловую перем-ю sp с файлом Vedom.dat} vvod; print; {процедуры создания и чтения файла}

work; print; {корректировка и чтение измененного файла} readln

end.

# Практические задания

**Задание**

1. Запись имеет вид: фамилия, пол, год рождения и рост. Вывести данные о самом высоком спортсмене.
2. Запись имеет вид: название вуза, число студентов, количество факультетов. Добавить в конец файла информацию о трех новых вузах.
3. Запись имеет вид: название издания, газеты или журнала, стоимость одного экземпляра, количество экземпляров в год . Вывести на печать информацию о самом дешевом издании.
4. Запись имеет вид: фамилия студента, номер зачетной книжки, 4 оценки за экзамен. Вы- водить информацию о всех двоечниках и корректировать ее.
5. Запись имеет вид: фамилия спортсмена, его номер, количество набранных очков. По- менять местами в файле записи о первых двух спортсменах. \*
6. Запись имеет вид: фамилия, номер телефона, дата рождения. Внести в начало списка информацию о четырех новых знакомых.

?

лами?

# Контрольные вопросы:

1. Как можно описать файлы?
2. Какие типы файлов существуют в языке Pascal?
3. Как организовать прямой доступ к типизированным файлам?
4. Особенности работы с типизированными файлами?
5. Особенности работы с текстовыми файлами?
6. Особенности работы с нетипизированными файлами?
7. Основные стандартные процедуры и функции для работы с типизированными файлами
8. Основные стандартные процедуры и функции для работы с нетипизированными фай-
9. Основные стандартные процедуры и функции для работы с текстовыми файлами ?
10. Общий алгоритм создания файла.
11. Общий алгоритм обработки файла.
12. Что называют физическим файлом?
13. Что называют логическим файлом?
14. Как связывается логический файл с физическим файлом?
15. Для чего и как открываются файлы?
16. Как закрыть файл?
17. Что называют текущим указателем?
18. Опишите операции ввода-вывода для файлов.
19. Опишите перемещение в среде типизированных файлов.
20. Назовите основные отличия текстовых файлов от типизированных файлов.
21. Назовите основные процедуры и функции, предназначенные для работы с текстовыми

файлами.

**Практическая работа №12.** Применение рекурсивных функций

Цель:

* научиться решать задачи с использованием рекурсивных функций и процедур

# Краткие теоретические сведения

Если поставить два зеркала напротив друг друга и между ними поместить предмет, то по- лучится бесконечное множество изображении, причем каждое из них содержит свое собственное. Любое из этих изображений можно рассматривать как рекурсивный объект, который частично состоит или определяется с помощью самого себя. Рекурсивные объекты обладают несколькими свойствами: — простотой построения;

* + несхожестью конечного результата с начальными данными;
  + внутренним самоподобием.

В математике встречаются рекурсивные определения, позволяющие описать объекты че- рез самих себя. К таким определениям относится, например, определение натурального числа:

* 1. единица есть натуральное число;
  2. число, следующее за натуральным (т. е. больше его на единицу), есть натуральное

число.

Определение, которое задает некоторый объект в терминах более простого случая этого

же объекта, называется рекурсивным определением.

Мощность рекурсивного определения заключается в том, что оно позволяет с помощью конечного высказывания определить бесконечное множество объектов. Как и цикл, рекурсивное определение содержит повторения, но каждый раз при этом используются новые данные, т. е. повторения не являются явными.

Рекурсия — это способ описания функций или процессов через самих себя.

Процесс может быть описан некоторым алгоритмом, называемым в данном случае рекур- сивным. В нем выделяется два этапа выполнения:

1. «погружение» алгоритма в себя, т. е. применение определения «в обратную сторону», пока не будет найдено начальное определение, не являющееся рекурсивным;
2. последовательное построение от начального определения до определения с введенным в алгоритм значением.

Примеры рекурсивных алгоритмов, часто оформляемых в виде процедур и функций.

1. Наиболее распространенным рекурсивным определением является определение факто- риала (нерекурсивное вычисление факториала приведено в примере Р9): (*a*) 1! *=* 1, *(b) n*

*>* 1, *n: = n\*(n -* 1)!

На основе этого определения можно записать программу вычисления факториала, исполь- зующую рекурсивную функцию.

**program** Р25;

**var** n, у: **integer;**

**function** F (x: **integer**): **integer**; *{описание рекурсивной функции}*

# begin

**if** x=1

**then** F: = 1 *{вызов для начального определения}*

**else** F: = х \* F (х - 1) *{вызов для предыдущего определения}*

**end;** *{конец описания функции}*

**begin** *{начало главной программы}*

**readln** (n);

Y: = F (n); *{вызов функции в главной программе}*

**write** (n, ‘! = ‘, Y)

**end.** *{конец главной программы}*

Выполним программу Р25 для *n* = 4. Рекурсивная функция будет работать следующим об- разом (при вызове функции значение n присваивается переменной *х).* Сначала осуществляется

«погружение», работает оператор ветви **else** условного оператора:

1. й шаг: *х =* 4, *х-*1 = 3, выполняется промежуточное вычисление 4! = 4 \* З!
2. й шаг: *х* = 3, *х-*1 = 2, выполняется промежуточное вычисление З! *=* 3 \* 2!
3. й шаг: *х =* 2, *х-*1 = 1, выполняется промежуточное вычисление 2! = 2 \* 1!
4. й шаг (последний): 1! = 1 по начальному определению, работает оператор F: *=* 1 ветви

**then** условного оператора.

Следующий этап выполнения рекурсивного алгоритма — построение «прямого» определе- ния, от начального до получения результата с исходными для алгоритма данными (числом 4). При этом осуществляется подстановка предыдущих вычислений (более поздних шагов) в более ранние:

1. й шаг: 2! = 2 \* 1 = 2
2. й шаг: З! = 3 \* 2 = 6
3. й шаг: 4! = 4 \* 6 = 24 — получен результат, он возвращается в главную программу и присваивается переменной Y.
4. **Вычисление степени с натуральным показателем** можно определить рекурсивно:
   1. *x* 0 = 1
   2. *k* > 0: *xk = x\* xk -* 1

Этому определению соответствует рекурсивная функция **power (k, x).** Программа имеет

вид:

**program** Р26;

**var** у: **real;** n: **Integer;**

**function** power (k: **integer;** x: **real): real;** *{описание рекурсивной функции}* **begin if** k=0

**then** power: = 1 *{начальное определение}*

**else** power: = x \* power (k - 1, x) *{рекурсивное определение}*

**end;** *{конец описания функции}*

**begin** *{начало главной программы}* **write** (‘ основание степени х = ‘); **readln** (у);

**write** (‘показатель степени k = ‘);

**readln** (n);

**write** (‘х в степени k’, power(n, у)) *{вызов функции и печать результата}* **end.** *{конец главной программы}*

* 1. **Вычисление чисел Фибоначчи**. Итальянский математик Фибоначчи придумал последо-

вательность натуральных чисел: 1, 1, 2, 3, 5, 8, 13, ... . Первые два члена последовательности равны единице, а каждый, начиная с третьего, равен сумме двух предыдущих. Для чисел Фибо- наччи верно соотношение:

Fk = Fk + Fk-2

Рекурсивная функция получения значения n-го числа Фибоначчи имеет вид:

# function Fib (n: integer): integer; begin

**if** k<3

**then** Fib: *=* 1

**else** Fib: = Fib (n - 1) + Fib (n - 2)

# end;

Для чисел Фибоначчи используется следующее рекурсивное определение: (a)*n=* 1, *n=*2: fib(*n*) *=* 1

*(b) n >* 2: fib *(n) =* fib (*n* - 2) + fib *(n - 1)*

Для того чтобы определить fib (6), применяя данное рекурсивное определение, надо вычис-

лить:

fib (6) = fib(4) + fib(5) = fib(2) + fib(3) + fib (5) = 1 + fib(3) + fib(5) =

= 1 + fib(l) + fib(2) + fib(5) = 1 + 1 + 1 + fib(5) = 3 + fib(3) + fib(4) =

*=* 3 + fib(l) + fib (2) + fib(4) = 3 + 1 + 1 + fib(4) = 5 + fib(2) + fib(3) =

= 5 + 1 + fib(l) + fib(2) = = 6+1+1=8

Количество действий в данных вычислениях с использованием рекурсивного определения

чисел Фибоначчи резко возрастает, потому что это определение ссылается само на себя два- жды. При вычислении факториала количество действий при выполнении программы с рекур- сивной функцией и примера Е9 одинаково.

* 1. **Рекурсивные алгоритмы** могут быть оформлены и **в виде процедур**. Примером такой

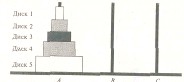
процедуры является решение задачи о Ханойских башнях.

Эта задача связана с легендой о том, что в одном из восточных храмов находится бронзовая плита с тремя алмазными стержнями. На один из них при сотворении мира нанизали 64 диска из чистого золота так, как показано на рисунке 7. Жрецы должны переносить диски с одного стержня на другой, следуя следующим законам:

1. диски можно перемещать только по одному;
2. нельзя класть больший диск на меньший.

Согласно легенде, когда все диски будут перенесены с одного стержня на другой, наступит конец света.

Решение этой задачи реализовано в виде рекурсивного алгоритма, который представляет собой инструкцию по перемещению дисков. Сформулируем задачу, присвоив имена стержням *(А, В, С)* и номера дискам (от 1 до *n*). Надо перенести диски со стержня *А* на стержень С, исполь- зуя *В* как вспомогательный и следуя приведенным выше правилам переноса дисков.



***Рис. 7.*** *Ханойские башни*

Алгоритм на естественном языке имеет вид:

1. если *n* = 0, остановиться;
2. переместить верхние *п -* 1 дисков со стержня *А* на стержень *В,* используя стержень С как вспомогательный;
3. переместить оставшийся диск со стержня *А* на стержень С;
4. переместить *п -* 1 дисков со стержня *В* на стержень С, используя стержень *А* как вспомо- гательный.

В процедуре появляется новый тип данных — **char,** значение этого типа — один символ, заключенный в апострофы.

**program** Р27;

# var k: integer;

**procedure** Hanoy (n: **integer;** One, Two, Three: **char); begin**

**if** n > 0 **then begin**

Hanoy (n - 1, One, Three, Two);

**write** (‘ переместить диск’, n, ‘со стержня ‘, One, ‘на стержень ‘, Two); Hanoy (n - 1, Two, One, Three)

# end; end; begin

**write** *(‘введите количество дисков’);*

**readln** (k):

Hanoy (k,’А’,’В’,’С’)

# end.

Результат работы программы для *n* = 3 — это инструкция из 7 пунктов *(п =* 4 — инструкция из 15 пунктов):

переместить диск 1 со стержня *А* на стержень С переместить диск 2 со стержня *А* на стержень *В* переместить диск 1 со стержня С на стержень *В* переместить диск 3 со стержня *А* на стержень С переместить диск 1 со стержня *В* на стержень *А* переместить диск 2 со стержня *В* на стержень С переместить диск 1 со стержня *А* на стержень С

**Практические задания**

**Задание.**

|  |  |
| --- | --- |
| № | Условие задачи |
| 1 | Написать функцию, которая находит цифровой корень целого числа. |
| 2 | Найти сумму цифр заданного натурального числа. |
| 3 | Найти количество цифр в заданном натуральном числе. |
| 4 | Составить программу вычисления суммы четных факториалов. (n-четное, n10) |
| 5 | Описать рекурсивную логическую функцию Simm(S,I,J), проверяющую, является ли симмет- ричной часть строки S, начинающаяся i-м и заканчивающаяся j-м ее элементом. |
| 6 | Составить программу вычисления суммы нечетных факториалов. (n-четное, n10) |
| 7 | Составить программу сортировки массива целых чисел. |
| 8 | Составить программу вычисления НОД двух натуральных чисел. |
| 9 | Составить программу нахождения числа, которое образуется из данного натурального числа при записи его цифр в обратном порядке.(173371) |
| 10 | Составить программу перевода данного натурального числа в р-ичную систему счисления( 2р9) |
| 11 | Дан прямоугольник со сторонами A и B, где А,В- натуральные числа. Начнем отсекать от него квадраты. Сколько квадратов можно отсечь, если каждый раз отсекается самый большой квадрат. |
| 12 | Поиска значений в упорядоченном списке. |
| 13 | Найти сумму 1/1+1/2+1/3+1/4+…+1/n,основываясь на рекурсии.(сумма k слагаемых равна сумме (k-1) слагаемых плюс k-е слагаемое). |
| 14 | Напишите главную программу для вычисления *n*-го числа Фибоначчи. Почему использовать рекурсивный алгоритм вычисления *n*-го числа Фибоначчи невыгодно? |
| 15 | Определите рекурсивно умножение как сложение и деление как вычитание и оформите алго- ритмы в виде рекурсивных функций с вызовом из главных программ. |
| 16 | Составить программу сортировки массива целых чисел. |
| 17 | Составить программу вычисления НОД двух натуральных чисел. |
| 18 | Составить программу нахождения числа, которое образуется из данного натурального числа при записи его цифр в обратном порядке.(173371) |
| 19 | Составить программу перевода данного натурального числа в р-ичную систему счисления( 2р9) |
| 20 | Дан прямоугольник со сторонами A и B, где А,В- натуральные числа. Начнем отсекать от него квадраты. Сколько квадратов можно отсечь, если каждый раз отсекается самый большой квадрат. |
| 21 | Поиска значений в упорядоченном списке. |

|  |  |
| --- | --- |
| 22 | Найти сумму 1/1+1/2+1/3+1/4+…+1/n,основываясь на рекурсии.(сумма k слагаемых равна сумме (k-1) слагаемых плюс k-е слагаемое). |

**Контрольные вопросы:**

* 1. Что такое рекурсивный объект и каковы его свойства?
  2. Приведите примеры рекурсивного определения в математике.
  3. Что такое рекурсия?
  4. Как выполняется рекурсивный алгоритм?
  5. Поясните выполнения рекурсивной функции вычисления степени с натуральным пока- зателем.

**Практическая работа №13**. Программирование модуля

Цель:

 усвоить знание основ модульного программирования; освоить способы создания и приме- нения модулей.

# Краткие теоретические сведения

Модульное программирование основано на понятии **модуля** – **программы** или функцио- нально завершенного **фрагмента** программы.

Модуль характеризуют:

1. один вход и один выход. На входе программный модуль получает определенный набор исходных данных, выполняет их обработку и возвращает один набор выходных данных;
2. функциональная завершенность. Модуль выполняет набор определенных операций для реализации каждой отдельной функции, достаточных для завершения начатой обработки дан- ных;
3. логическая независимость. Результат работы данного фрагмента программы не зависит от работы других модулей;
4. слабые информационные связи с другими программными модулями. Обмен информа- цией между отдельными модулями должен быть минимален;
5. размер и сложность программного элемента в разумных рамках.

С помощью модулей решаются различные профессиональные задачи обработки данных разного типа.

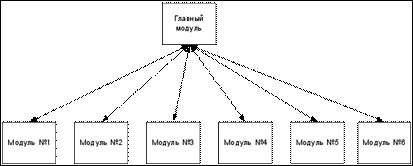


Рис. 1. Схема подключения модулей

Предлагается использовать следующие **характеристики программного модуля** для оценки его приемлемости: **размер** модуля, **прочность** модуля, **сцепление** с другими модулями и **рутинность** модуля.

Размер модуля измеряется числом содержащихся в нем операторов. Модуль не должен быть слишком маленьким или слишком большим.

Прочность модуля – это мера его внутренних связей. Чем выше прочность модуля, тем больше связей скрыто от внешней по отношению к нему части программы и, следовательно, тем проще сама программа.

Сцепление модуля – это мера его зависимости по способу передачи данных от других модулей. Чем слабее сцепление модуля с другими модулями, тем сильнее его независимость от других модулей.

Рутинность модуля — это его независимость от предыстории обращений к нему. Модуль будем называть рутинным, если результат обращения к нему зависит только от значений его параметров и не зависит от результатов предыдущих обращений к нему.

С помощью модулей решаются различные профессиональные задачи обработки данных разного типа.

# Практические задания

**Задача 1.** Разработать и реализовать программу ввода данных, вычисления функций при помощи модулей, вывода результатов.

Программа работы

1. Составить список функций и соответствующих им модулей для реализации в про- грамме.
2. Разработать интерфейс программы ввода, обработки и вывода данных.
3. Написать программный код для каждого модуля.
4. Реализовать интерфейс и программный код в среде визуальной разработки про-

грамм.

1. Провести тестирование и отладку программы.
2. Нарисовать интерфейс программы со спецификацией и записать программный код

с комментариями в отчете по работе.

1. Записать несколько вариантов тестирования программы.
2. Провести тестирование исполняемого файла.

**Задача 2.** Разработать и реализовать программу вычисления значения сложного выраже- ния при помощи шагов:

1. Ввода данных, вычисления функций при помощи первого модуля, вывода результа-

тов;

1. Чтения выходных данных первого модуля, вычисления функций при помощи вто-

рого модуля, вывода результатов;

1. Чтения выходных данных второго модуля, вычисления функций при помощи треть- его модуля, вывода результатов.

Программа работы

1. Составить список функций и соответствующих им модулей для реализации в про- грамме.
2. Разработать интерфейс программы ввода, обработки и вывода данных.
3. Написать программный код для каждого модуля.
4. Реализовать интерфейс и программный код в среде визуальной разработки про-

грамм.

1. Провести тестирование и отладку программы.
2. Нарисовать интерфейс программы со спецификацией и записать программный код

с комментариями в отчете по работе.

1. Записать несколько вариантов тестирования программы.
2. Провести тестирование исполняемого файла.

# Контрольные вопросы:

1. Что называют модулем в контексте модульного программирования?
2. Сколько входов и выходов имеет один модуль?
3. Существуют ли типы данных, которые невозможно обработать при помощи модулей?
4. Зависит ли результат работы модуля от работы других модулей?
5. Чем должны быть ограничены размер и сложность модуля?
6. Что используют для оценки приемлемости программного модуля?
7. Чем определяется размер модуля?
8. Какую характеристику называют прочностью модуля?
9. Как зависит независимость модуля от его сцепления?
10. Какой модуль называют рутинным?

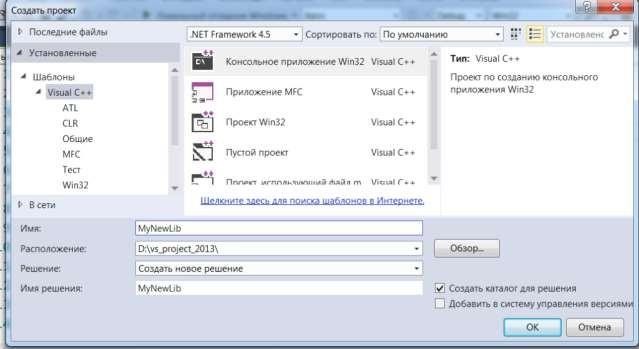
**Практическая работа №14.** Использование указателей для организации связанных списков Цель:

* изучение порядка действий при вычислении выражений;
* приобретение навыков в записи выражений на языке Паскаль ииспользовании стандартных функций.

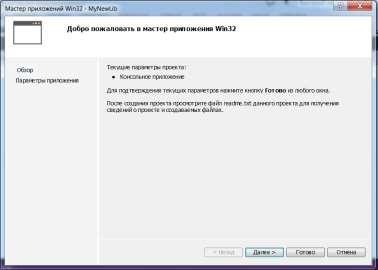
**Цель работы:** сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с использованием библиотеки подпрограмм**Оборудование, технические и программные средства:** персональный компью- тер, среда программирования **Visual Studio**.

Существует два вида библиотек статические .lib и динамические .dll,

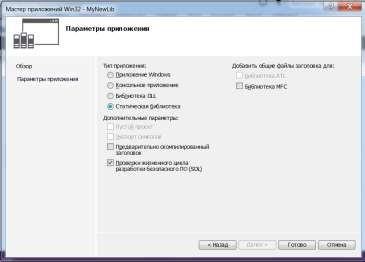
1. Открываем visual studio у меня visual studio
2. Выбираем: “Создать новый проект”, в открывшемся окне выбираем “консольное прило- жениевиндовс 32”, также вводим название проекта MyNewLib и указываем путь где мы сохра- ним проект.



Дальше нажимаем ок и мы переходим к следующему окошку.



1. В новом окне выбираем “Статическая библиотека” и убираем галочку “предвари- тельнооткомпилированные заголовок”, далее нажимаем «готово»



Таким образом, у нас создается новый проект в котором будут создаваться не программы exe астатические библиотеки lib.

1. Дальше в этот проект MyNewLib добавляем два файла один заголовочный

#ifndef FUNCTION\_H#define FUNCTION\_H

void func\_hellow();void func\_privet();

#endif

function.h:

и файл function.cpp:

#include <iostream>using std::cout; using std::endl;

#include "function.h";

void func\_hellow()

{

cout <<"Hellow world gacpada"<<endl;

}

void func\_privet()

{

cout <<"privet eb tu bl9 mazafaka bich"<<endl;

Включаем их в проект, а дальше компилируем нажимаем клавиши ctrl+alt+F7 и у нас создается в папке

Debug или Release смотря какой режим выбрать. (мы выбраем режим Debug) и в папке Debug создался файлбиблиотеки MyNewLib.lib

Для того, чтобы создавать качественные библиотеки нам нужно каждую функцию запи- сать в отдельный файл, это для того, чтобы в MyNewLib.lib было создано несколько объектных файлов .obj в данном случаемы могли бы файл function.сpp разбить на два файла допустим файл f1.cpp в котором записать определениефункции func\_hellow() и на файл f2.cpp в нем записать определение второй функции func\_privet(), тогда у нас будет правильная библиотека и если мы в проекте будем использовать одну функцию, допустим func\_hellow(), а func\_privet() нет, то объ- ектный код .obj с функцией func\_privet() не добавиться , а только

.obj func\_hellow() будет добавляться, я так думаю это фажный момент, возможно ниже попробуем смоделировать два разных варианта создания библиотек. Просто при использовании библиотеки в которойкаждой функции принадлежит отдельный объектный код, только включа- ются те функции которые используются, а если все функции в одном .obj то естественно вклю- чатся все функции и даже те которые мы не используем.

5. Нужно подключить библиотеку к нашему проекту, для этого:

* создаем новый проект пустой, проект будет пусть консольный вин 32 и создаем в нем файл main.cpp со следующим кодом:

#include <iostream>using std::cout; using std::endl;

#include "function.h"

int main()

{

func\_hellow();func\_privet();

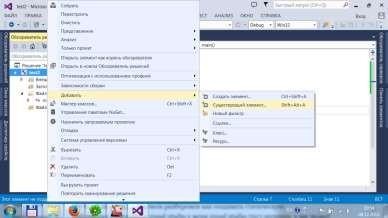
return 0;

}

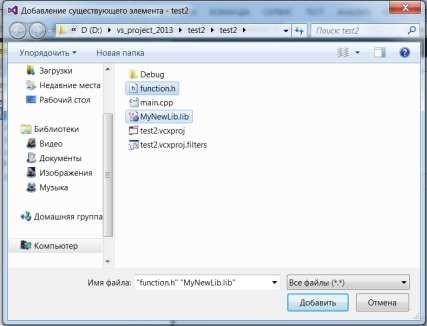
таким образом, мы включили в проект заголовочный файл function.h в котором находятся определения

функций

* вызываем функции, (если мы сейчас это все скомпилируем, то у нас вылезет ошибка так как мы реальноне добавили файл function.h и не добавили саму библиотеку в проект)
* копируем файлы MyNewLib.lib и файл function.h в папку с файлами нашего проекта – это та папка гденаходится наш файл main.cpp.
* с помощью горячих клавиш Shift+Alt+A



Дальше появится окно.



Выбираете файлы и нажимаете добавить и все файлы добавлены, при открытии обозревателя решений увас они должны быть видны из него.

Все из рисунка видно что статическая библиотека MyNewLib.lib у нас добавлена и заголовочный файл с определениями функций из библиотеки MyNewLib.lib включен это файл function.h, теперь нажимаем Ctrl+F5 и все компилируется.

# Контрольные вопросы:

1. Что представляют собой связанные списки и какие виды связанных списков вы зна- ете?
2. К каким структурам данным относятся связанные списки?
3. С помощью какой структуры данных можно наиболее эффективно решить задачу сортировки и почему?
4. Что представляет собой элемент динамической структуры данных?
5. Какие операции можно выполнять над списками?

**Практическая работа №15.** Создание проекта с использованием компонентов для работы с текстом

**Цель работы:** сформировать умения по использованию и настройке интегрированной средыразработчика Delphi, сформировать умения по созданию простейших приложений

**Оборудование, технические и программные средства:** персональный компьютер, инте- грированнаясреда разработчика **Turbo Delphi**.

## Задание 1. Изучение рабочей среды Turbo DelphiМетодические указания по выполнению за- дания:

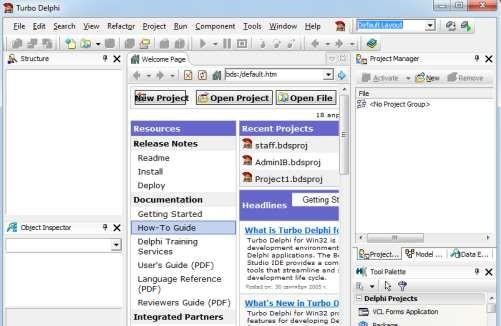
1. Запустите интегрированную среду разработчика **Turbo Delphi**. Для запуска **Turbo Delphi** существуетнесколько способов:

## Пуск – Все Программы – Borland Developer Studio 2006 – Turbo Delphi;

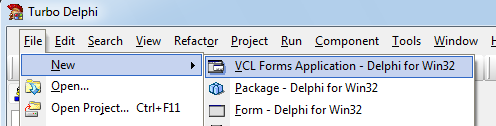
* + Двойной щелчок на пиктограмме программы в окне **Компьютер**;
  + Двойной щелчок на соответствующем ярлыке на рабочем столе.

**Turbo Delphi** — это среда быстрой разработки от корпорации **Borland**, в которой в качестве языка программирования используется одноименный язык программирования **Delphi**, ранее из- вестный как **Object Pascal**.

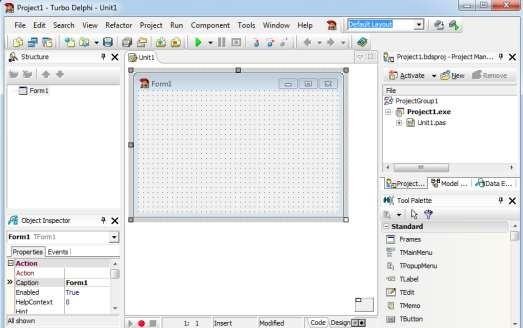
1. После запуска появится главное окно среды разработки в центре которого по умолча- нию будет открыта HTML-страница **Welcome Page** во встроенном Интернет - браузере. На ней находятся ссылки на разделы документации и список последних открытых проектов (**Recent Projects**).



1. Закройте страницу приветствия и создайте новый проект, выбравв главном меню пункт **File – New - VCL Forms Application — Delphi for Win32**.



1. Среда разработки примет вид, представленный на рисунке. Данное окно содержит не- сколько частей, ссодержимым которых мы сейчас и познакомимся.



**Инспектор объектов**

**Палитра компонентов**

**Конструктор форм**

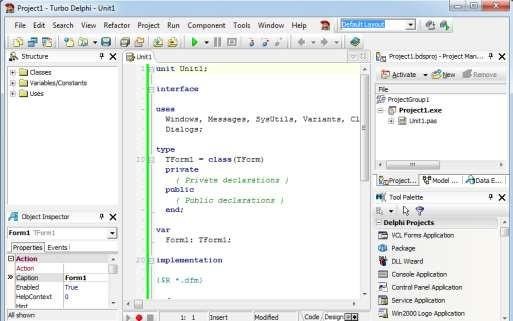
**Менеджер проекта**

**Структура проекта**

1. Центральную часть окна занимает окно **Конструктора форм (Form designer)** при- ложения. **Формой**

**(Form)** приложения на этапе разработки принято называть окно программы (во время разра- ботки это форма, на этапе выполнения — окно). В окне конструктора форм вы будете форми- ровать внешний видсвоего будущего приложения: изменять само окно, а также наполнять его различными элементами. Окно конструктора формы изначально находится на переднем плане и перекрывает окно редактора кода. Переключаться между этими окнами можно нажатием клавиши **F12** (либо нажатием на специальные вкладки **Code** и **Design** в нижней части окна).

1. Активизируйте окно редактора кода, нажав клавишу **F12** на клавиатуре. Содержимое окна программы будет выглядеть так, как показано на рисунке.



Необходимо отметить, что большая часть работы при создании проекта в среде **Delphi** сво- дится к работе сконструктором форм или редактором кода, при этом посредством конструктора создается внешнее окноприложения, а в редакторе прописывается код программы.

При закрытии конструктора форм или редактора кода автоматически закрывается и приложе- ние.Вернитесь обратно к окну **Конструктора формы**.

1. Слева от окна конструктора находится окно **Инспектора объектов (Object Inspector)**. Это окно заполнено информацией о выделенном объекте проекта (в данный мо- мент — формы). Окно **Object Inspector** имеет две вкладки — **Properties (Свойства)** и **Events (События)**.
2. Измените для формы свойство **Caption.** Это свойство отвечает за заголовок формы. Изначально оно равно **Form1**, измените его на **Мой первый проект на Delphi** и нажмите кла- вишу **Enter**, сразу увидите,что ваша форма изменилась. Во время выполнения окно вашего приложения будет иметь введенный вами заголовок.
3. Поначалу группировка свойств может вызвать затруднения (свойств довольно много у любого объектаи для запоминания основных из них нужно некоторое время), поэтому такую группировку можно отключить. Для этого необходимо навести указатель мыши на окно **Object Inspector** и нажать правую кнопку мыши, в появившемся вспомогательном меню вы- брать пункт **Arrange - by Name**.
4. Каждый компонент обладает своим собственным набо-

ром свойстви событиями, на которые он может реагировать. Остановимся на наиболее общих свойствах компонентов. Например, имя компонента задает свойство **Name**, свойства, определяющие размеры и положение компонента на форме: **Width** — ширина, **Height** — высота, **Left** — смещение влево, **Top** —смещение компо-

нента вниз (изменять размеры и положе- ние компонентов можно и с помощью мыши).

Логическое свойство **Visible** определяет, будет ли ви-

ден данныйкомпонент (результат виден при запуске приложения, а не в ре- жиме проектирования), свойство **Color** задает цвет элемента управления, **Cursor** — форму курсора мыши, когда он находится над элементом управления. Логическое свойство **Ctl3D** позволяет выдавать компонент пространственным, а свойство**Enabled** разрешает или запре- щает получение управления данному компоненту. Каждый элемент управления может содер- жать подсказку, появляющуюся в том случае, если указатель мыши находитсянад элементом управления. Если логическое свойство **ShowHint** имеет значение **True**, то при проведении курсора мыши над компонентом будет выдаваться подсказка, текст которой содержится в свойстве **Hint**.

Также стоит отметить такое свойство формы, как **BorderStyle**, позволяющее задавать внешний вид окна. Если это свойство, например, установить в значение **bsDialog**, то при исполнении форма будет содержать на заголовке формы единственную кнопку, которая закрывает прило- жение. В этом случае при запуске нельзя будет свернуть приложение или изменить его размер.

1. Вторая вкладка окна **Object Inspector** — **Events** используется для описания событий, на которые будетреагировать выделенный объект (в данный момент им является ваша форма). Именно при выборе необходимого вам события в редакторе кода появится заготовка проце- дуры обработки, где надозаписывать код программы.
2. В окне **Менеджера проекта (Project Manager)** отображается структура приложения (проекта, над которым вы сейчас работаете). В этом окне содержится общая информация о проекте, информация об используемых внешних модулях (библиотеках), а также обо всех фай- лах проекта.
3. **Палитра компонентов (Tool Palette)** — это один из наиболее часто используемых инструментов **Delphi**. Она состоит из большого числа групп, где располагаются компоненты. **Компонент (Component)** — это элемент пользовательского интерфейса, который может быть перенесен на форму.Это кнопки, метки, поля для ввода всевозможных данных, выпадающие списки, в общем, все то, что выобычно видите на окнах в операционной системы **Windows** (та- кие компоненты называют визуальными

— **Visual**). Кроме того, это могут быть также и невидимые (не визуальные) компоненты, т. е. те, которыене отображаются в момент выполнения программы, но выполняют различные функ- ции. Типичный пример такого компонента — **таймер (Timer)**.

Все компоненты объединяются в группы по функциональному назначению. После создания проектараскрыт список компонентов группы **Standard**, содержащий основные элементы диа- логовых окон **Windows**. В основном мы будем изучать компоненты с этой вкладки, однако нам также понадобятся компоненты с вкладок **Addition, Win32, Dialogs**.

1. Окно **Структуры проекта** после создания проекта отображается в левой верхней ча- сти экрана. Оно содержит информацию о структуре исходного кода программы (именно по- этому не содержит информации, если активно окно дизайнера формы). Для того чтобы по- смотреть данную информацию, следует переключиться в окно редактора кода. После активи- зации окна редактора кода окно **Structure**заполнится информацией.

## Задание 2. Создание простого проекта

Разработать программу **Hello**, которая при нажатии на кнопку **«Hello»** выводит сообщение

**«Hello, мирWindows and Delphi!»**, при нажатии на кнопку **«Выход»** программа завершит работу.

## Методические указания по выполнению задания:

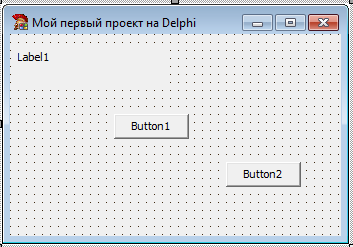
1. **Проект (Project)** — совокупность файлов, используемые средой разработки (точнее говоря, компилятором **Turbo Delphi**) для итоговой генерации программы. Проект в простей- шем случае представляет собой совокупность следующих файлов:
   * **файл описания проекта (bdsproj-файл)** — файл специального формата, в котором записана общаяинформация о проекте;
   * **главный модуль (dpr-файл)** содержит инструкции, обеспечивающие запуск нашей программы;
   * **модуль формы (dfm-файл)** содержит информацию о настройках и компонентах, которые присутствуют на форме. Модуль формы формируется автоматически при выполне- нии настроек формы, перенесении на нее и настройки компонентов;
   * **модуль реализации (pas-файл)** содержит информацию только о присутствующих на форме компонентах и процедурах обработки событий на этих компонентах;
   * ***файл ресурсов (res-файл)****;*

# файл конфигурации (cfg-файл);

* + **исполняемый файл (exe-файл)**, который создается при запуске программы на исполнение.

Поскольку при работе с проектом автоматически создается довольно много файлов, рекомен- дуется сохранять их в заранее подготовленном каталоге. Создайте в папке своей группы новую папку и назовите её **Hello**.

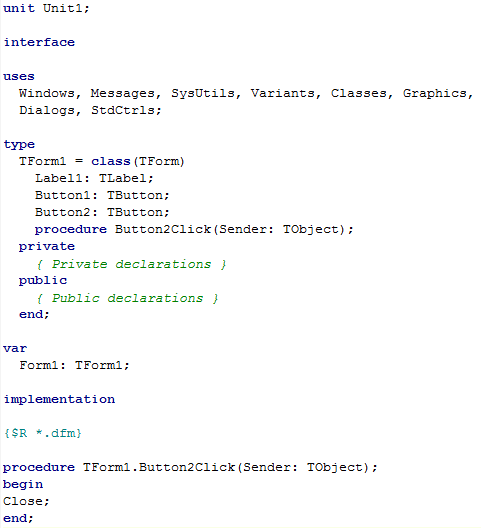
1. Сохранение всех файлов проекта осуществляется через пункт главного меню **File - Save all (сохранитьвсе)**. Первый раз при сохранении потребуется сохранить два файла: мо- дуль формы и файл проекта. Сохраните проект, выполнив команду **File - Save Project as...** В открывшемся окне найдите свою новуюпапку (**Hello**). В этом же окне **Delphi** предложит со- хранить (по умолчанию) первый файл под именем **Unit1.pas**. Можно согласиться, т.е. нажать кнопку **Сохранить**. Сразу же **Delphi** предложит сохранить второй файл под именем **Project1. bdsproj**. Его надо обязательно переименовать в **hello. bdsproj**
2. Откройте папку **Hello**, где вы сохранили файлы. Обратите внимание, что на самом деле вы создали более двух файлов, все они и образуют проект. Из этих файлов основным рабочим файлом является файл с расширением **.рas** (в нашем примере **Unit1.pas**)
3. Разместите на форме компоненты в соответствии с рисунком: **Label1, Button1, Button2** вкладкипалитры компонентов **Standard**.



1. Выделите кнопку **Button2**, перейдите в **Object Inspector** на вкладку **Properties**, найдите свойство

**Caption (Заголовок)** и измените его на **Выход**.

1. Сохраните проект, нажав на соответствующую кнопку на панели инструментов.
2. Перейдите на правую страницу **Object Inspector**, на страницу **Events**, и найдите со- бытие **OnClick**, справа от него дважды щелкните мышью.
3. Из листинга видно, что в рамках формы класса **TForm1** описываются все присутству- ющие на форме компоненты: **Label11, Button1, Button2** и заголовок процедуры **Button2Click**. В реализационной части,после служебного слова **implementation**, создана заготовка для всей процедуры, в которой и будем вписывать необходимый код. Параметр **Sender** передается в любой процедуре обработки события и определяет компонент формы, где произошло событие. Оказавшись в коде программы, а точнее, в заготовке процедуры кнопки **Button2**, напишите лишь одну команду: **Close**.



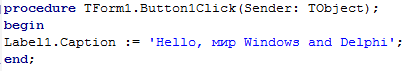
Все события, на которые приложение может реагировать, разделяются на пользовательские и системные. К пользовательским относятся события, связанные с клавиатурой или мышью, например **OnClick** — одинарный щелчок левой клавишей мыши (именно это событие является наиболее распространенным и именно для его обработки нами была создана процедура в предыдущем примере); **OnDblClick** — двойной щелчок левой клавишей мыши; **OnMouseDown** — нажатие клавиши мыши; **OnMouseUp** — отпускание клавиши мыши; **OnMouseMove** — перемещение мыши.

Кроме пользовательских событий, существуют программно управляемые события. Отдельно рассмотрим события, обрабатываемые самой формой: событие **OnCreatе** происходит в мо- мент создания формы; **OnClose** генерируется, когда форма должна быть закрыта. Два данных события происходят с формой всего один раз в отличие от других: **OnShow** — возникает, когда форма должна стать видимой, **OnHade**

— когда форма должна быть убрана с экрана, **OnPaint** — перед тем, как форма будет перери- сована наэкране.

Существуют события, которые формируют сами элементы управления, например: **OnEnter** событие,которое появляется, когда элемент управления становится активным, и **OnExit** собы- тие, возникающее,когда элемент управления перестает быть активным. Данные два события существуют только у элементов управления и только в том случае, если свойство **Enabled** имеет значение **True**.

1. Сохраните проект.
2. Запустите проект. Запустить проект на исполнение можно либо нажатием кнопки **F9**, либо через главное меню: **Run - Run**, либо нажатием соответствующей кнопки на панели ин- струментов: . Призапуске формируется стандартное окно **Windows**, которое отображается соответствующей кнопкой напанели задач. Для дальнейшей доработки исходного файла необ- ходимо сначала остановить приложение, при этом используются стандартные средства Windows. Основное отличие формы при проектировании и после запуска на исполнение — это сетка из точек. Если точки отсутствуют, то приложение активно. Нажмите на кнопку **Выход** и вы вернетесь из режима просмотра программы в режим разработки проекта.
3. Выделите кнопку **Button1**, найдите в **Object Inspector** свойство **Caption** и вставьте новое название кнопки **Hello**.
4. Перейдите в **Object Inspector** на страницу **Events**, найдите событие **OnClick** и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре



кнопки **Button1**, напишите следующее выражение:

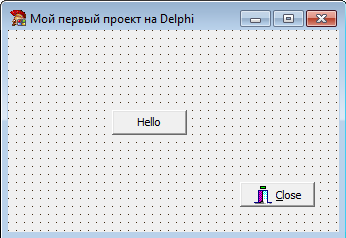
1. Сохраните проект еще раз, запустите и протестируйте его, т.е. запустите проект и нажмите кнопку **Hello**

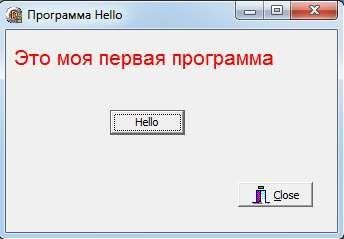
– должна появиться ранее написанная вами фраза, нажмите кнопку **Выход**, и программа за- кроется.

1. Измените содержание выводимой на экран реплики на «**Это моя первая программа**». Для этого дважды щелкните по кнопке **Hello** и измените соответствующий оператор.
2. Задайте шрифт и размер выводимой реплики следующим образом: размер – **16**, цвет –

**красный**, гарнитура шрифта - **Arial**. Для этого измените свойство **Font** в **Object Inspector**.

1. Замените простую кнопку **Выход** на **Close**. Для замены кнопки надо удалить суще- ствующую, найти в палитре компонентов на странице **Additional**, кнопку с названием **BitBtn,**

и разместить её на форме. Затем в свойстве **Kind** измените ее вид на **bkClose**. При этом в процедуре **Button2Click** можно убрать команду **Close**, т.к. кнопка этого типа (**BitBtn**) автома- тически выполняет сою функцию по закрытию окна.

1. Запустите свой проект не в среде **Delphi**, а в среде **Windows**. Для этого сверните или закройте **Turbo Delphi**. Перейдите в папку вашего проекта **Hello** и найдите в ней файл **Hello.exe**. Щелкните мышкой поэтому файлу – файл, как и положено файлу с расширением **.exe**, запустится. Вы увидите, что ваш проектработает в **Windows** без запуска среды **Delphi**.
2. Озаглавьте окно проекта «**Программа Hello**». Для этого выделите форму **Form1**, пе- рейдите в **Object Inspector**, найдите свойство **Caption** и замените надпись на **Программа Hello**.
3. Сохраните изменения и запустите проект на исполнение.

## Задание 3. Создание простого проекта

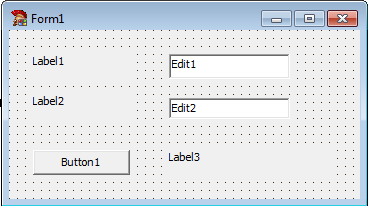
Разработать программу с помощью, которой пользователь, введя свой рост и фактический вес, мог бы определить худой он или полный, и на сколько ему нужно поправиться или похудеть. Для разработки программы воспользуйтесь тем, что оптимальный вес человека определяется так: рост человека минус 100.

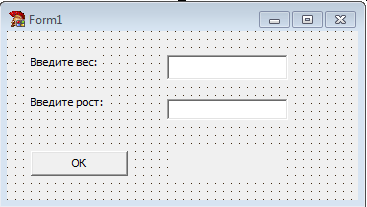
Если фактический вес человека меньше оптимального, то человек худой, и наоборот, если больше, то нужнопохудеть.

## Методические указания по выполнению задания:

1. Создайте в папке своей группы новую папку и назовите её **Weight**.
2. Сохраните оба файла проекта в папке **Weight**, выполнив команду **File - Save Project as...**
3. Разместите на форме компоненты в соответствии с рисунком. В **Edit1** будет вводиться вес в кг, а в **Edit2**

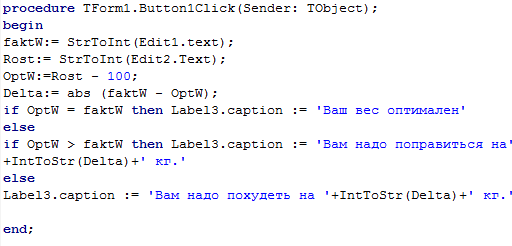
– рост в см.



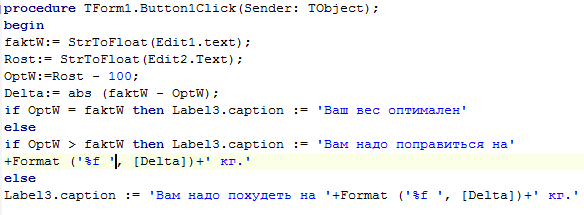
1. ****Задайте для элементов управления значение свойства **Caption** в соответствии с рисун- ком. Для **Label3** значение свойства **Caption** оставьте пустым, а для элементов управления **Edit1** и **Edit2** значение свойства **Text** оставьте пустым. Для этого выделите элемент управления и перейдите в **Object Inspector**на страницу **Properties (свойства).**
2. Перейдите в код программы (на клавиатуре нажмите клавишу **F12**). Введите в раздел **VAR** переменные для сохранения значений фактического веса (**faktW**), оптимального веса (**optW**), роста (**Rost**) и разницымежду оптимальным весом и фактическим (**Delta**). Пусть зна- чения этих переменных будутокругленными, в этом случае тип этих переменных можно объ- явить как **Integer**.

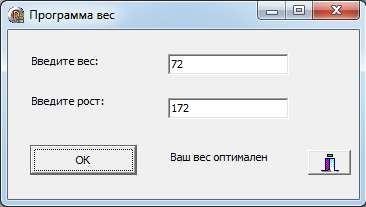


1. Выделите кнопку **Button1**. Перейдите в **Object Inspector**, на страницу **Events (собы- тия)**, и найдите событие **OnClick**, справа от него дважды щелкните мышью. Оказавшись в коде программы, а точнее, взаготовке процедуры кнопки **Button1** заполните её следующим кодом:



1. Сохраните проект и протестируйте работу приложения.
2. Усовершенствуем программу так, чтобы можно было бы вводить десятичные вели- чины. Для этого в разделе **Var** зададим тип переменных не **Integer**, а **Real**.
3. Функция **FloatToStr( )** преобразует помещенную в скобки переменную типа **Real** в переменнуюстрокового типа, а функция **StrToFloat()** превращает строку в дробное или дей- ствительное число, числос плавающей точкой. Замените в тексте программного кода **IntToStr** на **FloatToStr**, а **StrToInt** на **StrToFloat**.
4. Сохраните проект и протестируйте работу приложения.
5. Для форматирования и преобразования действительных чисел вместо функции

**FloatToStr()**. Применимфункцию **Format(‘ %f ’,[переменная])**: **Format (‘%f ‘, [Delta]).**

1. Сохраните проект и протестируйте работу приложения.
2. Озаглавьте окно проекта **Программа Вес**. Создайте кнопку **Close**.
3. Сохраните проект и протестируйте его работу.

## Задание.

Используя кнопочные компоненты **TSpeedButton,** разработать программу – калькулятор, вы- полняющийпростейшие действия.

## Методические указания по выполнению задания:

1. Запустите интегрированную среду разработчика **Turbo Delphi**. Создайте новый проект.
2. Создайте в папке своей группы новую папку и назовите её **Калькулятор**. Со- храните проект всозданную ранее папку, выполнив команду **File - Save Project as...** под име- нем **Calc.bdsproj.**
3. Задайте для формы следующие свойства: **Caption** – Calc, **Font** - **Size** – 10.
4. Разместите на форме одно поле ввода, дайте ему имя **Disp** и очистите свойство **Text**.
5. Для калькулятора мы будем использовать кнопки **TSpeedButton** (группа **Additional**). Их отличие от кнопок **TButton** и **TBitBtn** состоит в том, что они не получают фокуса ввода клавиатуры. Это значит, что их нельзя сделать активными, переходя от элемента к элементу с помощью клавиши **Tab**, и вообщеих можно привести в действие только мышкой. Таким обра- зом, на форме будет всего один компонент,принимающий сигналы с клавиатуры — это поле

ввода.

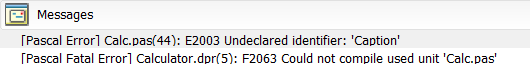
1. Добавьте на форму 12 кнопок **TSpeedButton** для цифр, арифметических действий, знака «равно» и операции «сброс». Установите для всех кнопок размеры 25 на 25 пикселей и разместите их так, как на рисунке.
2. Дайте кнопкам-действиям имена **btnPlus**, **btnMinus**, **btnMul**, **btnDiv** (сложение, вы- читание, умножениеи деление), а кнопке «равно» — имя **btnCalc**.
3. Для действий и кнопки **С** установите жирный шрифт, а для кнопки **С** дополнительно

— красный цветшрифта.

1. Кнопка **С** должна просто стирать содержимое поля ввода **Disp**. То есть, нужно вызвать метод

**Disp.Clear**. Добавьте обработчик события **OnClick** для кнопки **С**.

1. Понятно, что когда пользователь щелкнул по кнопке-цифре, нужно добавить эту цифру в конец текстаполя ввода. Конечно, можно для каждой кнопки написать обработчик типа **Disp.Text := Disp.Text + '1';**Однако можно сделать и более грамотно, определив для всех кнопок общий обработчик. Заметим, что цифра, которую нужно добавить, это и есть текст кнопки, ее свойство **Caption**. Таким образом, когда мы назначим всем кнопкам один един- ственный обработчик, в нем нужно как-то определить, какую именно кнопку мы нажали. А такая возможность есть, ведь обработчик имеет параметр **Sender** — это адрес компонента, от которого пришло сообщение. Выделите все кнопки-цифры и определите для нихобщий обра- ботчик события **onClick**: **Disp.Text := Disp.Text + Sender.Caption;**
2. Запустите программу. Скорее всего, вы получите сообщение об ошибке:

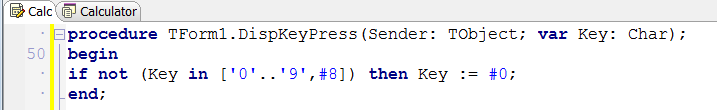


Фраза **Indecrared identifier** означает **Необъявленное имя**, то есть, по мнению **Delphi** объект **Sender** не имеет свойства **Caption**. Обратим внимание на то, что параметр **Sender** объявлен в заголовке процедурыкак переменная типа **TObject**, а объект **TObject** действительно не имеет свойства **Caption** (это можно проверить по справочной системе). Но мы знаем, что в нашей программе все компоненты, которые могут вызвать этот обработчик, относятся к типу **TSpeedButton**, у которого свойство **Caption** есть. Поэтому можно сказать, что программа должна воспринимать параметр **Sender** не как просто объект, акак **TSpeedButton**: **Disp.Text := Disp.Text + TSpeedButton(Sender).Caption;**

1. Исправьте код и запустите программу. Проверьте, как она работает при вводе текста

и букв с клавиатуры и с помощью мыши. Что вам не понравилось?

1. При работе программы вы увидели, что с клавиатуры можно ввести буквы, которые нам совсем не нужны. Когда пользователь нажмет клавишу в поле ввода, возникает событие **OnKeyPress**, которое можно перехватить, установив соответствующий обработчик. В нем все

«ненужные» символы заменяются символом с кодом 0, который не изменяет содержимое поля. Создайте обработчик события**OnKeyPress** для поля ввода **Disp**:

Разберемся, как работает данная процедура. Обратите внимание, что второй параметр обра- ботчика объявлен как **var Key**, то есть, его можно менять в процедуре. В квадратных скобках записывают множество значений, причем '0'..'9' означает интервал: все символы от '0' до '9'. Символ с кодом 8, обозначаемый как **#8** — это возврат каретки (**Backspace**, удаление символа слева от курсора). Если **Key** — не цифра и не возврат каретки, в эту переменную записывается 0 — нажатие клавиши игнорируется.

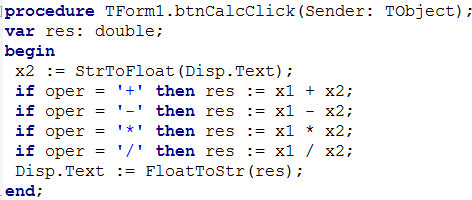
Запустите программу и попробуйте вводит буквы.

1. При работе программы вы могли заметить, что при щелчке по кнопке курсор сразу перемещается в начало числа, а хотелось бы, чтобы он оставался справа. Мы сделаем так, чтобы при вводе очередной цифры с помощью кнопки курсор автоматически переводился в конец числа. У компонента **TEdit** естьсвойства **SelStart** и **SelLength**, которые определяют начало и длину выделенной части (она обычно обозначается синим фоном). Если **SelLength=0** (ничего не выделено), тогда **SelStart** указывает на позицию курсора в поле ввода. Если устано- вить это свойство равным длине текста, курсор встанет сразуза последним символом.
2. Добавьте в конец обработчика события **OnClick** код: **Disp.SelStart := Length(Disp.Text);** Проверьте работу программы.
3. Далее необходимо организовать вычисления. Нам нужны две переменных для хране- ния чисел и одна символьная переменная, в которую будем записывать тип операции. По- скольку при расчетах могут получиться числа с дробной частью, для хранения чисел будем использовать вещественные переменные. В простейшем случае для хранения операции можно использовать переменную типа **Char**(один символ), но мы объявим ее как символьную строку, так как при доработке программы могут понадобиться и многосимвольные названия операций. Объявите в начале программы две вещественныхпеременные **x1** и **x2** типа **Double** и одну сим- вольную строку **oper**.
4. Когда мы нажимаем на одну из кнопок-операций, нужно запомнить введенное число в переменной **x1** и тип операции в переменной **oper**. Тип операции (надпись на кнопке) легко узнать, обратившись к свойству **Caption**. Поэтому можно установить для всех кнопок-опера- ций один обработчик. Выделите все кнопки-операции и создайте для них один обработчик события **OnClick**:



1. При нажатии на кнопку «**Равно**» нужно прочитать из поля ввода второе число и вы- полнить операцию.При этом первое число и тип операции уже должны находиться в перемен- ных **x1** и **oper**. Поскольку в результате деления может получиться число с дробной частью, переменная для хранения результата (назовем ее **res**) тоже должна быть вещественной.

Введите обработчик события **OnClick** для кнопки «**Равно**»:



1. Запустите программу и проверьте ее работу. Учтите, что для ввода второго числа нужно сначала очистить экран кнопкой «**С**» (позже мы исправим это неудобство).
2. Закройте окно программы и затем запустите ее заново. Введите какое-нибудь число, а затем сразу щелкните по кнопке «равно». Что получилось? Попытайтесь объяснить этот эф- фект, учитывая, что присоздании глобальной символьной переменной **oper** в нее записана пу- стая строка.
3. Наверняка вы догадались, что проблема вызвана тем, что мы еще не задали операцию, а уже попыталисьчто-то вычислить. При этом ни один условный оператор не сработал, и зна- чение переменной **res** не изменялось. А что же в ней было? Откуда взялось это число? Пере- менная **res** объявлена в процедуре, то есть, она локальная. Память для локальных переменных выделяется в стеке при каждом новом вызовепроцедуры, причем эти ячейки не обнуляются. Поэтому в переменной **res** в нашем последнем эксперименте остался «мусор» — постороннее значение, которое программа и вывела на экран. Если вы были очень внимательны, можно было заметить, что при трансляции программы в окне **Message** (в нижнем левом углу) было выдано предупреждение:



Это значит **«Переменной res, возможно, не будет присвоено никакого значения»**. Фор- мально это не ошибка, и программа может запуститься, однако к предупреждениям нужно относиться внимательно, потому что они могут указать на скрытые логические ошибки (ошибки в алгоритме), как в нашем случае. Какой же выход? Проще всего сделать так: если в переменной **oper** записана пустая строка, мыпросто выйдем из процедуры с помощью опера- тора **Exit**.

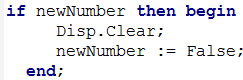
1. Добавьте в начало обработчика события **OnClick** для кнопки «Равно» строчку:

Проверьте работу программы. Что вам не нравится?

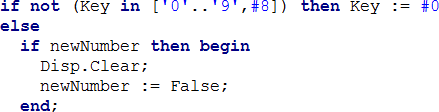
1. Конечно, очень неудобно, что перед вводом второго числа нужно очищать поле ввода, нажимая на кнопку «**С**». Хотелось бы делать это автоматически.
2. Запустите стандартную программу **Калькулятор** и посмотрите, в какой момент сти- рается первое число. Наверное, вы увидели, что число из поля ввода автоматически стирается, когда после нажатия на кнопки-действия или кнопку «**Равно**» пользователь набирает новое число. Мы введем логическую переменную **newNumber**, которой будем присваивать значение **True** в том случае, если нужно начинатьвводить новое число. Объявите глобальную логиче- скую переменную **newNumber**.
3. В конце обработчиков события **OnClick** для кнопок-действий и кнопки «равно» добавьте строку:

## newNumber := True;

1. Очистку экрана будем делать перед вводом нового числа. Дополните обработчики кнопок-цифр. В начало обработчика события **OnClick** для кнопок-цифр добавьте код:



1. Учтем в нашей программе, что пользователь может набирать число на клавиатуре. Для этого нужно дополнить аналогичным кодом обработчик события **OnKeyPress** для поля ввода. Измените обработчиксобытия **OnKeyPress** так, как показано ниже.
2. Запустите программу и проверьте ее. Можно ли ввести знаки сложения, вычитания, умножения и деления с клавиатуры? Выполняется ли действие при нажатии на клавишу **Enter**?

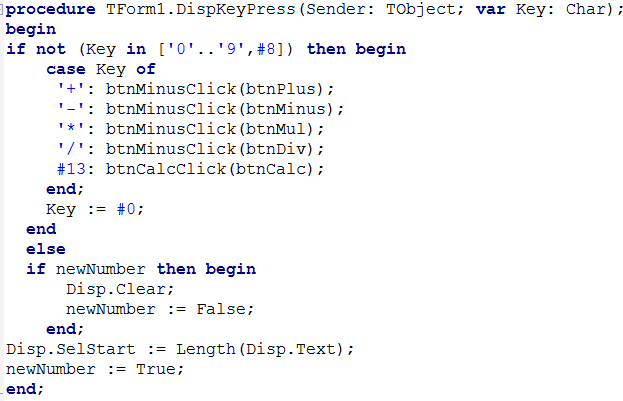


1. При работе программы вы увидели, что пока программа позволяет вводить с клавиа- туры только цифры,но не коды действий. Чтобы исправить этот недостаток, мы дополним об- работчик **OnKeyPress**.
2. Что нужно сделать, когда нажали символ + на клавиатуре? Проще всего имитировать щелчок по кнопке,то есть, вызвать процедуру-обработчик события **OnClick** для кнопок-дей- ствий. Предположим, что этотобработчик называется **btnDivClick** (у вас он может называться по-другому). Но этой функции нужно передать параметр — адрес объекта, пославшего сооб- щение. Этот объект — кнопка **btnPlus**, поэтому при нажатии клавиши + нужно выполнить команду **btnDivClick ( btnPlus );**

Обработка нажатия других клавишей строится аналогично. При нажатии клавиши **Enter**, име- ющей код 13(он обозначается в программе как #13), нужно вызвать обработчик события **OnClick** для кнопки

«равно» (она должна называться **btnCalc**).

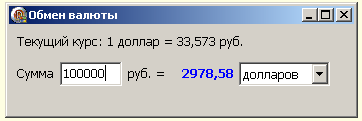
1. Измените код обработчика **OnKeyPress** следующим образом:



1. Запустите программу и проверьте ее работу. После этого закройте проект.

## Контрольное задание:

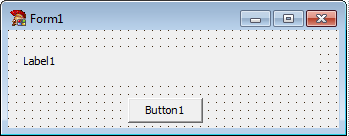
Составьте программу, которая переводит суммы из рублей в доллары и евро.



## Задание 6.

Разработать программу, которая при нажатии на кнопку **«Output»** выводит сообщение **«Моя первая программа на языке Delphi»**, а затем при повторном нажатии на эту же кнопку со- общение исчезает. Приповторном выводе размер надписи должен увеличиваться.

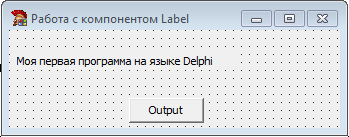
## Методические указания по выполнению задания:

1. Запустите интегрированную среду разработчика **Turbo Delphi**. Создайте новый проект.
2. Создайте в папке своей группы новую папку и назовите её **Label**. Сохраните проект в созданную ранеепапку, выполнив команду **File - Save Project as...**
3. Разместите на форме компонент **Label** и кнопку **Button** вкладки палитры компонен- тов **Standard**.

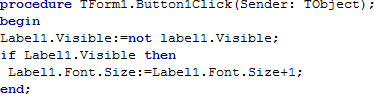
Компоненты ввода — вывода данных можно условно разделить на несколько различных бло- ков: компоненты вывода текстовой информации на экран; однострочные поля ввода текстовой и числовой информации; многострочные поля ввода.

Для вывода определенной информации на экран, кроме уже ранее используемого компонента **Label**, есть идругие компоненты. Текст, который будет отображен, можно задавать как на этапе разработки формы,так и в процессе выполнения программы, присвоив значение свой- ству **Caption**.

1. Задайте для формы заголовок **«Работа с компонентом Label»**.
2. Выделите надпись **Label1**, найдите в **Object Inspector** свойство **Caption** и вставьте новое название надписи **Моя первая программа на языке Delphi.**
3. Выделите кнопку **Button1**, найдите в **Object Inspector** свойство **Caption** и вставьте новое название кнопки **Output**.



1. Перейдите в **Object Inspector** на страницу **Events**, найдите событие **OnClick** и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре



кнопки **Button1**, напишите следующий программный код:

В этой программе при каждом очередном нажатии происходит изменение свойства **Visible**, вследствие чегонадпись то появляется, то исчезает с экрана, а также происходит увеличение свойства **Size**.

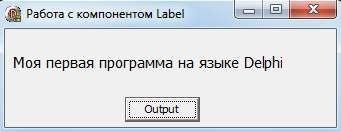
Для вывода определенной информации на экран, кроме уже описанного компонента **Label**, может быть также использован компонент **Panel** с той же самой вкладки или **StaticText** со страницы **Additional**. Они имеют незначительные отличия от компонента **Label**.

Остальные компоненты позволяют вводить и редактировать информацию, включая возмож- ностьвыделения, копирования, удаления и вставки фрагментов текста. Отметим общие для всех редакторов методы: **Clear** — удалить весь текст, помещенный в редакторе; **ClearSelect**

— удалить выделенный фрагмент текста; **CopyToClipboard** — копировать в буфер выделен- ный фрагмент, **CutToClipboard** —удалить из текста выделенный фрагмент и поместить его в буфер, **PasteFromClipboard** — копироватьтекст из буфера в то место редактора, где в данный момент находится курсор.

Для всех редакторов определено дополнительное событие **OnChange**, возникающее, когда из- меняется текст, находящийся в редакторе.

Для ввода или вывода одной строки могут использоваться компоненты **Edit** со страницы **Standart** и **MaskEdit** со страницы **Additional**. Основное свойство данных компонентов — это строка, которая либовводится, либо выводится. Данное свойство имеет имя **Text** и тип **String**, доступное как во время подготовки, так и время выполнения. Логическое свойство **ReadOnly** позволяет запретить изменения, а целочисленное свойство **GetTextLen** выдает текущую длину строки.

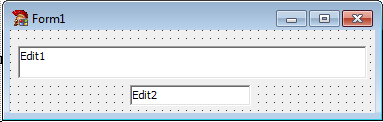
1. Сохраните изменения и запустите проект. Протестируйте его работу.

## Задание 7.

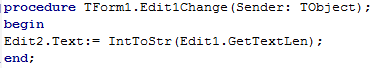
Разработать программу, которая при вводе текста в первый компонент **Edit1**, во втором ком- поненте **Edit2**отображает реальную длину вводимой строки. Кроме этого, при выходе из компонента **Edit1** его содержимое копируется в буфер обмена и удаляется, а при возвраще- нии в программу появляется снова.

## Методические указания по выполнению задания:

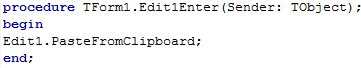
1. Создайте в папке своей группы новую папку и назовите её **Edit1**. Сохраните проект в созданную ранеепапку, выполнив команду **File - Save Project as...**
2. Разместите на форме два компонент **Edit** вкладки палитры компонентов **Standard**.



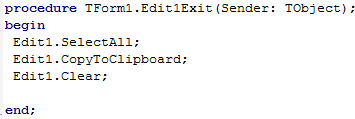
1. Задайте для формы заголовок **«Работа с компонентом Edit»**.
2. Выделите текстовое поле **Edit1**, найдите в **Object Inspector** свойство **Text** и оставьте его пустым. Аналогичные действия выполните со вторым текстовым полем.
3. Выделите компонент **Edit1**, в **Object Inspector** перейдите на вкладку **Events** и найдите событие **OnChange** и справа от него дважды щелкните мышкой. Оказавшись в коде про- граммы, но теперь в процедуре текстового поля **Edit1**, напишите следующий программный код:



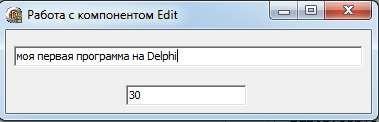
1. Выделите компонент **Edit1**, в **Object Inspector** перейдите на вкладку **Events** и найдите событие**OnEnter** и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре текстового поля **Edit1**, напишите следующий программный код:



1. Выделите компонент **Edit1**, в **Object Inspector** перейдите на вкладку **Events** и найдите событие **OnExit**и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре текстового поля **Edit1**, напишите следующий программный код:



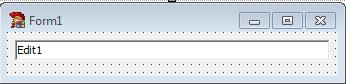
1. Сохраните изменения и запустите проект. Протестируйте его работу.



## Задание 8.

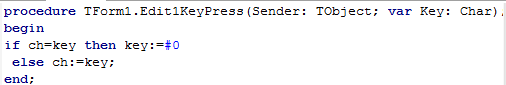
Разработать программу, которая запрещает ввод в компонент **Edit1** подряд двух одинаковых символов.

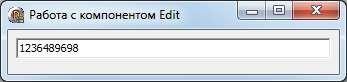
## Методические указания по выполнению задания:

1. Создайте в папке своей группы новую папку и назовите её **Edit2**. Сохраните проект в созданную ранеепапку, выполнив команду **File - Save Project as...**
2. Разместите на форме компонент **Edit** вкладки палитры компонентов **Standard**.
3. Перейдите в код программы (на клавиатуре нажмите клавишу **F12**). Введите в раздел **VAR** глобальнуюпеременную **ch** типа **char**, в которой будет храниться последний нажатый символ.



1. Задайте для формы заголовок **«Работа с компонентом Edit»**.
2. Выделите текстовое поле **Edit1**, найдите в **Object Inspector** свойство **Text** и оставьте его пустым.
3. Создайте процедуру обработки события **KeyPress** текстового поля **Edit1**, параметр **Key** данной процедуры содержит символ нажатой клавиши. Если вновь введенный символ совпадает с только что нажатым символом, то он игнорируется. В противном случае, новый символ запоминается в переменной **ch**.



1. Сохраните изменения и запустите проект. Протестируйте его работу.

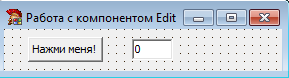
## Задание 9.

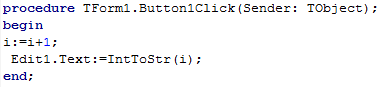
Разработать программу, которая считает количество нажатий на кнопку и выдает это значе- ние вкомпоненте **Edit**.

## Методические указания по выполнению задания:

1. Создайте в папке своей группы новую папку и назовите её **Edit3**. Сохраните проект в созданную ранеепапку, выполнив команду **File - Save Project as...**
2. Разместите на форме компонент **Edit** и кнопку **Button** вкладки палитры компонентов

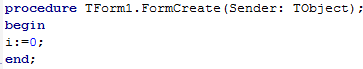
# Standard.

1. Используя **Object Inspector,** задайте значения свойств компонентов формы в соответ- ствии с рисунком.
2. Если в целочисленной переменной **i** будем считать количество нажатий, то про- цедура обработкисобытия **OnClick** кнопки может быть записана в виде:

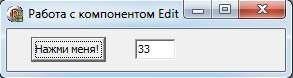


Однако остается вопрос, где описывать данную переменную **i**. Если сделать это внутри данной процедуры,то также необходимо осуществлять обнуление переменной, а это приведет к полу- чению одного и того результата, равного единице. Следовательно, переменная **i** должна быть глобальной переменной в модуле, а ее начальная инициализация должна происходить в про- цедуре, которая выполняется всего один раз, и всего один раз происходит это событие. Таким событием является создание формы **OnCreat**,данное событие произойдет один раз и проце- дура **FormCreate(Sender:TObject)** будет вызвана всего один раз.

1. Перейдите в код программы (на клавиатуре нажмите клавишу **F12**). Введите в раздел **VAR** глобальнуюпеременную **i** типа **integer**, в которой будет храниться последний нажатый символ.
2. Выделите форму, в **Object Inspector** перейдите на вкладку **Events** и найдите событие **OnCreate** и справаот него дважды щелкните мышкой. Оказавшись в коде программы, но те- перь в процедуре формы **Form1**, напишите следующий программный код:

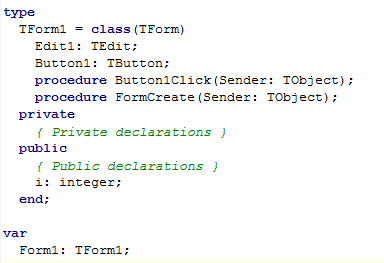


1. Сохраните изменения и запустите проект. Протестируйте его работу.



1. При возникновении необходимости сделать данную переменную **i** общедоступной, можно поместить описание переменной в интерфейсной части модуля после служебного слова **public**. Именно так, как правило, и поступают. В модуле необходима всего одна переменная

— форма, а все остальныеописываются в виде полей. В этом случае описание формы будет иметь вид:



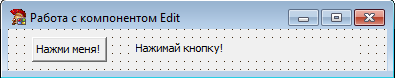
В программе для обращения к переменной **i** необходимо писать ее полное имя **Form1.i**. Однако код процедур обработки событий можно и не переписывать, поскольку процедуры обработки

описаны непосредственно в формы, следовательно, данное числовое поле доступно непосред- ственно.

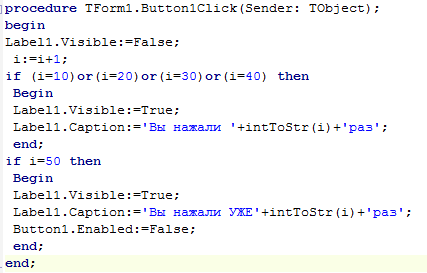
1. Внесите изменения в программный код в соответствии с рисунком. Сохраните проект и протестируйтеработу приложения.
2. Данную программу можно легко модифицировать так, чтобы после определенного ко- личества нажатийпоявлялось некоторое сообщение или кнопка блокировалась, или приложе- ние автоматически закрывалось. Результат можно отображать не только посредством компо- нента **Edit**, но и через не редактируемый текст, т. е. компонент **Label**, что в данном случае является более естественным.

Свойству **Visible** компонента **Label** присваиваем **False**, т. е. при открытии формы надпись от- ражаться не будет. Затем, как и ранее, при нажатии на кнопку переменная **i** увеличивается на

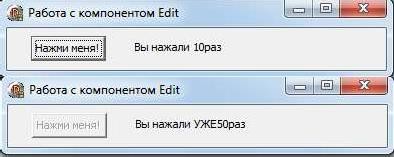
**1**. Когда значение переменной **i** будет равно **10, 20, 30** или **40** компонент **Label** становится видимым, а свойству **Caption** надписи присваиваем значение «**Вы нажали i раз**». При следу- ющем нажатии надпись становится невидима. Когда **i** станет равной **50**, кнопку необходимо сделать неактивной, для чего необходимо изменить значение свойства **Enabled** с **True** — включено на **False** — выключено.

1. Разместите на форме компонент **Label**. Задайте свойство **Caption** равным «**Нажимай кнопку!**». Удалите с формы компонент **Edit1**.
2. Перейдите в окно редактирования кода и внесите изменения в код процедуры обработки события

**OnClick** кнопки.



1. Сохраните проект и протестируйте работу приложения.



## Задание 10.

Разработать программу, которая считывает строку под определенным номером и помещает её в текстовоеполе.

## Методические указания по выполнению задания:

1. Создайте в папке своей группы новую папку и назовите её **Memo1**. Сохраните проект в созданную ранеепапку, выполнив команду **File - Save Project as...**
2. Для ввода или вывода нескольких строк могут использоваться компоненты **Memo** со страницы **Standard** и **RichEdit** со страницы **Win32** (полный текстовый редактор для **RTF**- файлов). Многие свойства у данных компонентов аналогичны свойствам компонента **Edit**, од- нако для возможности доступа к строкам вместо свойства **Text** имеется свойство **Lines**, при выборе которого во время проектирования задается начальное значение строк.

Для доступа к строкам во время выполнения программы также используется свойство **Lines** класса **TString**.Подробнее остановимся на классе **TString**, с которым в последствии мы будем еще встречаться. А именно, этот класс обладает свойствами: **Count** — целочисленное свой- ство, определяющее количествоэлементов в списке (в данном случае это будет количество строк в компоненте **Memo**), **Text** — свойство, содержащее все строки списка, **String[Index:Integer]** — свойство, определяющее строку с номером **Index**. Учитывая, что ну- мерация строк начинается с 0 и свойство **String** является свойством по умолчанию, можно утверждать, что свойства **Memo1.Lines.String[3] и Memo1.Lines[3]** эквивалентны и указы- вают на четвертую строку в компоненте **Memo1**.

Класс **TString** обладает также рядом методов, среди которых отметим следующие: **Add(St:String):integer** добавляет строку **St** и возвращает номер этой строки; **Delete(Index:Integer)** удаляет строку с номером **Index; Insert (Index:Integer, St:String)** встав- ляет строку с номером **Index, Clear** полностью уничтожаетвсе содержимое компонента.

Все содержимое компонента можно записать в файл с помощью метода **SaveToFile** или прочи- тать из файлапосредством методом **LoadFromFile**. Аналогичным образом можно поступить и с потоком, направив внего весь файл, или прочитать файл из потока.

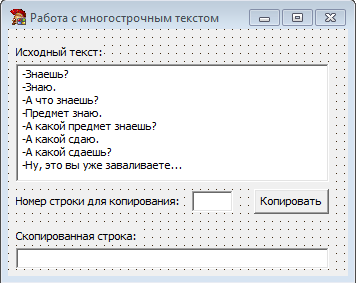
Важным свойством компонентов **Memo** и **RichEdit** является **ScrollBar**, которое определяет, будет ли окносодержать горизонтальные или вертикальные линейки прокрутки.

Компонент **RichEdit** обладает всеми характеристиками, присущими компоненту **Memo**, од- нако имеет богатые возможности для работы с текстовым форматом **RTF**. Данный формат предполагает возможность разбивать текст на параграфы. Для этого существуют специальные свойства: **SelAttributes**определяет атрибуты выделенного фрагмента и **Paragraph** — атрибуты абзаца.

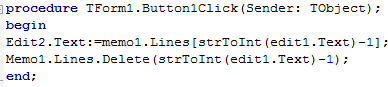
1. Разместите на форме компоненты **Edit,** компонент **Memo,** компонент **Label**, кнопку

**Button** вкладки палитры компонентов **Standard**.

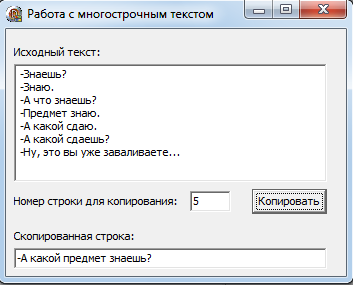
1. Используя **Object Inspector,** задайте значения свойств компонентов формы в соответ- ствии с рисунком.



1. Для реализации решения задачи процедура обработки события **OnClick** кнопки мо- жет быть записана ввиде:



1. Сохраните проект и протестируйте работу приложения.



## Задание 11.

Разработать программу, которая сохраняет текст, набранный в поле **Memo** в файл, имя кото- рого задано втекстовом поле **Edit**.

## Методические указания по выполнению задания:

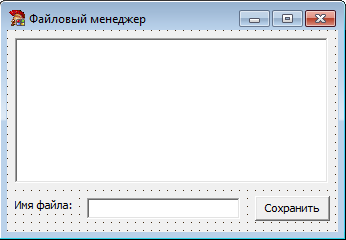
1. Создайте в папке своей группы новую папку и назовите её **Memo2**. Сохраните проект в

созданную ранеепапку, выполнив команду **File - Save Project as...**

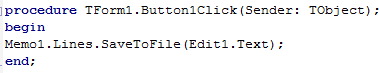
1. Разместите на форме компонент **Edit,** компонент **Memo,** компонент **Label**, кнопку

**Button** вкладкипалитры компонентов **Standard**.

1. Используя **Object Inspector,** задайте значения свойств компонентов формы в соответ- ствии с рисунком.



1. Процедура обработки события **OnClick** кнопки **Button** будет состоять из одной строки, исоответственно, листинг будет иметь следующий вид:



1. Сохраните проект и протестируйте работу приложения.



**Практическая работа №16.** Создание проекта с использованием компонентов ввода и отоб- ражениячисел, дат и времени.

**Цель работы:** сформировать навыки разработки приложений с использованием компонент ввода и отображения чисел, дат и времени в среде визуального программирования Delphi, изучить особенностиих использования

**Оборудование, технические и программные средства:** персональный компьютер, интегри- рованнаясреда разработчика **Turbo Delphi**.

## Задание 1.

Разработать программу, демонстрирующую действие процедур и функций, оперирующих с системнымизначениями даты и времени.

## Методические указания по выполнению задания:

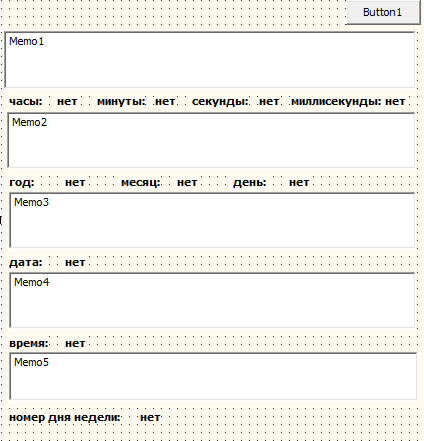
1. Запустите интегрированную среду разработчика **Turbo Delphi**. Создайте новый проект.
2. Создайте в папке своей группы новую папку и назовите её **Date1**. Сохраните проект в создан- ную ранеепапку, выполнив команду **File - Save Project as...**
3. Задайте для формы следующие свойства:

|  |  |
| --- | --- |
| **Свойство** | **Значение** |
| BorderStyle | bsNone |
| ClientHeigth | 441 |
| ClientWidth | 423 |
| Color | clCream |
| Left | 102 |
| Position | poScreenCenter |
| Top | 98 |

1. Разместить на форме компоненты в соответствии с рисунком 2.



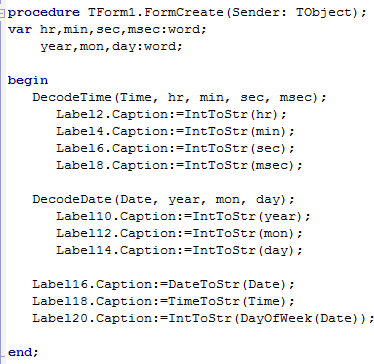
1. Задайте для элементов управления **Label** значение свойства **Caption** в соответствии с ри- сунком. Дляэтого выделите элемент управления и перейдите в **Object Inspector** на страницу **Properties** (свойства).

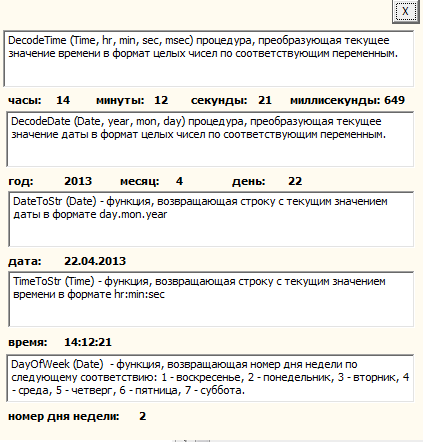


1. Задайте для элементов управления **Memo** значение свойства **Lines** в соответствии с табли- цей. Для этоговыделите элемент управления и перейдите в **Object Inspector** на страницу **Properties** (свойства).

|  |  |
| --- | --- |
|  | **Значение** |
| Memo1 | **DecodeTime (Time, hr, min, sec, msec)** процедура, преобразующая текущее значение  времени в формат целых чисел по соответствующим переменным. |
| Memo2 | **DecodeDate (Date, year, mon, day)** процедура, преобразующая текущее значе- ние даты в  формат целых чисел по соответствующим переменным. |
| Memo3 | **DateToStr (Date)** – функция, возвращающая строку с текущим значением даты в формате  day.mon.year |
| Memo4 | **TimeToStr (Time)** – функция, возвращающая строку с текущим значением времени в  формате hr:min:sec |
| Memo5 | **DayOfWeek (Date)** - функция, возвращающая номер дня недели по следу- ющемусоответствию: 1 – воскресенье, 2 – понедельник, 3 – вторник, 4 – среда,  5 – четверг, 6 – пятница, 7 – суббота. |

1. Задайте для элемента управления **Button1** значение свойства **Caption** равным **X**.
2. Введите в окне кода процедуру для отображения даты и времени календаря компьютера.



1. Запишите для командной кнопки **Button1** процедуру, которая по щелчку на эту кнопку закрывает форму.
2. Сохраните изменения и запустите проект, убедитесь в его работоспособности.

## Задание 2.

Разработать приложение, с помощью которого пользователь в одностраничном блокноте вы- бирает однуиз представленных закладок. На рабочем поле блокнота высвечивается соответ- ствующая надпись: год, месяц или день календаря компьютера.

Компонент приложения, содержащий несколько страниц, каждая из которых имеет ярлычок в виде закладки, называется элементом с закладками. Страницы пользователь может выбирать, щелкая по закладкам (корешкам или ярлычкам).

## Методические указания по выполнению задания:

1. В **Delphi** на странице **Win32** Палитры компонентов расположены две составляющие, работаю- щие на программирование элементов с закладками: **TabControl** – одностраничный блокнот; и

**PageControl** – многостраничный блокнот. Данные компоненты являются контейнерами и могут содер- жать в себе другие элементы или группы.

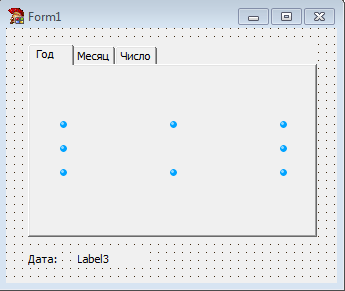
**TabControl** имеет несколько стилей отображения (свойство **Style**):

* + **tsTabs** – стандартные закладки объемного вида;
  + **tsButtons** – закладки в виде кнопок;
  + **tsFlatButtons** – закладки в виде плоских кнопок. Свойство **Tabs** задает число и назва- ния закладок блокнота.

Свойство **TabIndex** указывает текущую закладку в массиве **Tabs**. Программист может исполь- зоватьданное свойство для переключения на нужную закладку блокнота. Нумерация начина- ется с 0. Если значение **TabIndex** равно – 1, то ни одна закладка не выбрана.

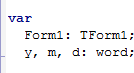
Свойство **HotTrack** задает подсвечивание названия закладки при наведении на неё курсора мыши.

1. Создайте новый проект.
2. Создайте в папке своей группы новую папку и назовите её **Date2**. Сохраните проект в создан- ную ранеепапку, выполнив команду **File - Save Project as...**
3. Разместите на форме компоненты **Label**, компонент **TabControl**. В **Label1** будет размещаться надписьблокнота, в **Label2**, **Label3** отображаются текущая дата календаря компьютера.
4. Задайте для элементов управления значение свойства **Caption** в соответствии с рисунком. Для этоговыделите элемент управления и перейдите в **Object Inspector** на страницу **Properties** (свой- ства).
5. Выделите элемент управления **TabControl1**, в **Object Inspector** перейти на страницу **Properties** (свойства) и найти свойство **Tabs**, щелкнуть на кнопке  . В открывшемся окне **String List Editor** вве- дите 3 строки со значениями год, месяц и число соответственно.

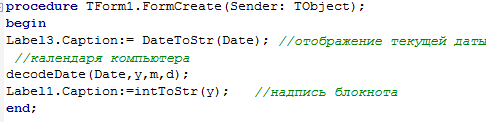


1. Перейдите в окно редактирования кода и опишите в разделе **VAR** переменные **y, m, d** типа

**word** какглобальные переменные модуля для хранения значений даты календаря компьютера.



1. Введите в окне редактора кода процедуру для отображения значений даты календаря компью- тера:

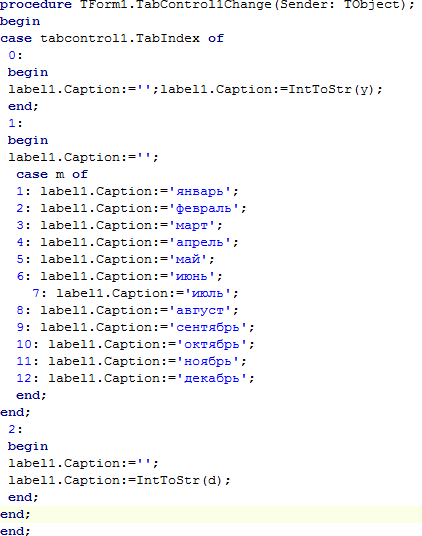


1. Выделите элемент управления **TabControl1,** в **Object Inspector** перейдите на страницу

**Events**

(события) и найти событие **OnChange.**

1. Введите в окне редактора кода процедуру для работы с элементом управления **TabControl1**.

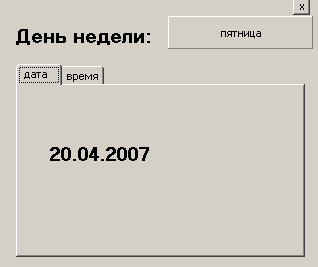


1. Сохраните изменения и запустите проект, убедитесь в его работоспособности.



## Контрольное задание 1.

1. Разработайте приложение, с помощью которого пользователь в одностраничном блокноте выби- рает одну из представленных вкладок. На рабочем поле блокнота высвечивается соответствующая надпись:дата, время календаря компьютера. На панели отображается день недели.



1. Измените, проект так, чтобы каждый день недели был раскрашен в разный цвет.

**Практическое занятие № 17.** Создание проекта с использованием кнопочных компонентов

**Цель работы:** сформировать умения использования переключателей при создании проекта Delphi, изучить их основные свойства; сформировать умения по созданию процедур на основе событий ком- понентов.

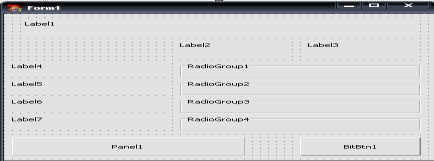
**Оборудование, технические и программные средства:** персональный компьютер, интегрированнаясреда разработчика **Turbo Delphi**.

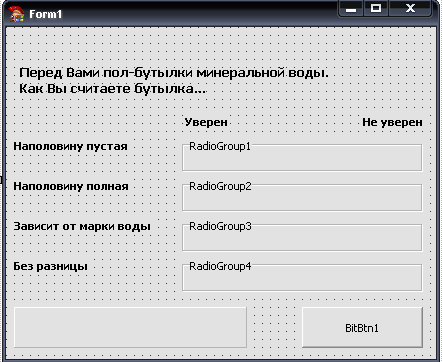
## Задание 1.

Разработать программу, с помощью которой пользователь мог бы выполнить следующее. После запуска программы появляется изображение, аналогичное рисунку. Пользователь по своему усмотрению выбирает один переключатель в группе. Каждому переключателю соответствует определенный балл. В зависимости от суммы набранных баллов появляется одно из сообщений

«Вы пессимист», «Вы реалист»или «Вы оптимист».

## Методические указания по выполнению задания:

1. Запустите интегрированную среду разработчика **Turbo Delphi**. Создайте новый проект.
2. Создайте в папке своей группы новую папку и назовите её **Radio1**. Сохраните проект в создан- нуюранее папку, выполнив команду **File - Save Project as...**
3. Разместить на форме компоненты в соответствии с рисунком.
4. Задайте для элементов управления **Label** значение свойства **Caption** в соответствии с рисунком. Дляэтого выделите элемент управления и перейдите в **Object Inspector** на страницу **Properties** (свой- ства).
5. Оставьте свойство **Caption** панели **Panel1** пустым.



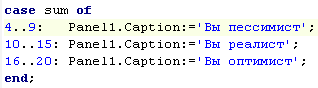
1. Выделите компонент **RadioGroup1**, перейдите в **Object Inspector** на страницу **Properties**, найдите свойство **Caption** и удалите заголовок. Свойству **Columns**, определяющему количество колонок, в которые будут отображаться переключатели, присвойте значение **5**.
2. Вызовите **String List Editor**, дважды щелкнув мышкой рядом со свойством **Items**, введите **5** строк со значениями 1, 2, 3, 4 и 5 соответственно.
3. Аналогичные действия проделайте с остальными компонентами **RadioGroup**. Можно выделить эле- менты управления с нажатой клавишей **Shift** и задать свойства, которые будут заданы для всех выде- ленных элементов управления.
4. Для того чтобы суммировать набираемые пользователем баллы, в процедуру обработки события

**RadioGroup1.OnClick** вставьте следующий код:



Единицу необходимо прибавлять, т.к. индекс первого переключателя равен 0, но соответствует 1баллу. Целочисленную переменную **sum** необходимо описать в разделе **var** модуля.

1. Вставьте в процедуру обработки событий **RadioGroup2.OnClick**, **RadioGroup3.OnClick** и **RadioGroup4.OnClick** аналогичные коды, но без обнуления переменной **sum**, т.к. обнуление необхо- димо лишь один раз перед началом суммирования.
2. Выведем на контрольную панель итоговое сообщение в зависимости от набранной суммы баллов. Для этого в процедуру обработки события **RadioGroup4.OnClick** добавим код:



1. Выведем сообщение об окончании тестирования, добавив в процедуру обработки события

## RadioGroup4.OnClick код: ShowMessage(‘Конец теста’);

В результате выполнения этого оператора появится информационное окно со словами «**Конец теста**» и единственной кнопкой **Ok**.

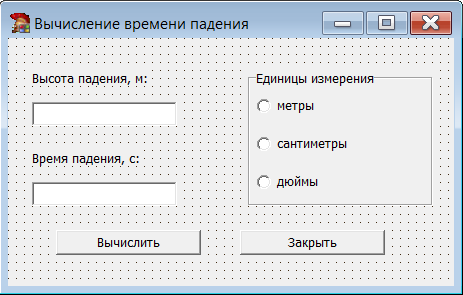
После нажатия кнопки **Ok** в информационном окне сделайте недоступными все **RadioGroup**, а всепереключатели в них - невыбранными. Для этого свойствам **Enabled** всех **RadioGroup** присвоить значение **False**, а свойствам **ItemIndex** - значение **-1**.

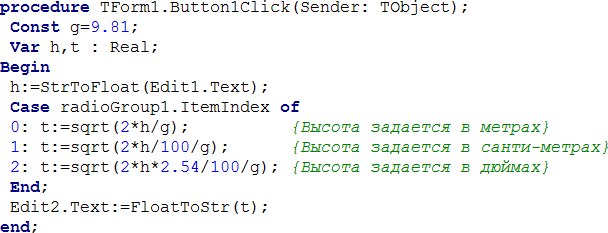
1. Для контроля правильности работы программы выведите на панель набранную пользователем суммубаллов. Для этого заголовку соответствующей панели присвоить значение **IntToStr(sum)**.
2. Запустите программу и убедитесь, что верная сумма баллов получается лишь при последовательном выборе переключате- лей сначала из **RadioGroup1**, затем из **RadioGroup2** и т.д. Если порядок был нарушен, или пользователь, изменив решение, выбрал другой переключатель в одной и той же группе,то результат будет неверным. Чтобы этого не случалось, сделайте доступной только ту группу переключателей, в которой необходимо сделать выбор. Для этого первоначально свойству **Enabled** всех **RadioGroup**, кроме **RadioGroup1**, присвойте значение **False**. В процедуре обработки события **RadioGroup1Click** добавить код, присваивающий свойству **Enabled RadioGroup2** значение **True** и т.д.
3. Сделайте возможным повторный запуск программы. Для этого разместите на форме кнопку **Button1**,свойству **Caption** которой присвойте значение **Retry**, а в процедуре обработки события **Button1Click**вставьте код, делающий доступной **RadioGroup1**.
4. Озаглавьте окно проекта **Проверь себя**.
5. Сохраните изменения и запустите проект, убедитесь в его работоспособности.

## Задание 2.

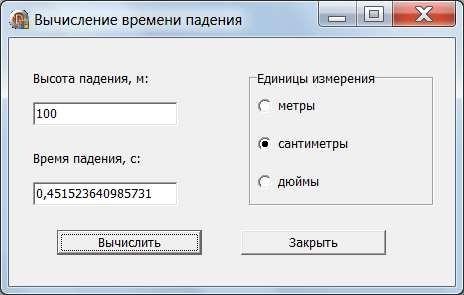
Разработать приложение, с помощью которого можно вычислить время падения тела с некото- рой высотыпри условии, что высота может задаваться в метрах, сантиметрах и дюймах.

## Методические указания по выполнению задания:

* 1. Запустите интегрированную среду разработчика **Turbo Delphi**. Создайте новый проект.
  2. Создайте в папке своей группы новую папку и назовите её **Radio2**. Сохраните проект в созданную ра- неепапку, выполнив команду **File - Save Project as...**
  3. Разместить на форме компоненты в соответствии с рисунком.
  4. Создадим процедуру обработки события **Click** для кнопки **Вычислить**:

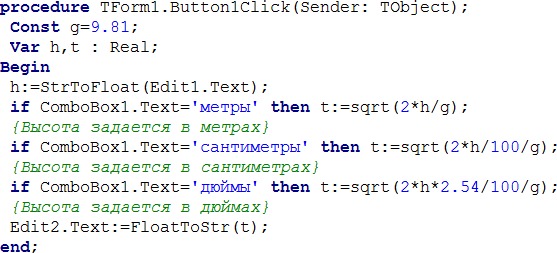
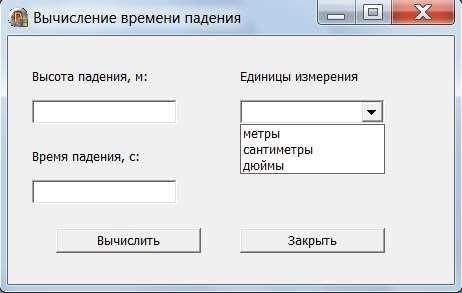


* 1. Запустите программу и проверьте её работоспособность.



* 1. Самостоятельно напишите обработчик события кнопки **Закрыть**.
  2. Недостаток компонента **RadioGroup** заключается в том, что имеется возможность определить только номер выбранной альтернативы, а не ее текстовое содержание. Для того чтобы определить и текстовое содержание альтернативы, можно использовать комбинированную строку, компонент **ComboBox**.Ком- бинированная строка ввода объединяет в себе свойство строки и списка. В обычном состоянии она имеет вид строки **Edit** со стоящей рядом кнопкой с изображением направленной вниз стрелки. Если нажать эту кнопку, то появится список строк, где можно выбрать произвольную. Данный компонент имеет свойство **Items**, поэтому задание альтернатив, которые в данном случае будут раскрывающимся списком, происходит аналогичным образом. Однако на этапе выполнения допустимо свойство **Text**, в котором находится выбранная из списка строка. При работе с данным компонентом необходимо раз- личать свойства **ComboBox.Items**. **Text** и **ComboBox.Text**. Если первое — это свойство, где находятся все строки списка, включая разделители (это свойство имеет подобный смысл и для **RadioGroup**), то второе содержит выбранную из списка строку.
  3. Измените проект так, чтобы выбор единицы измерения осуществлялся с помощью ком- понента

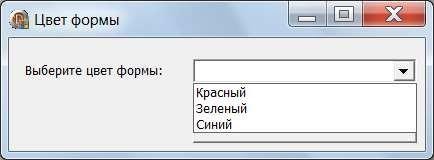
## ComboBox.

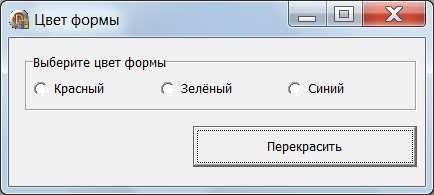


* 1. Выбор между компонентами **RadioGroup** и **ComboBox** во многом зависит от вкусов программистов, поскольку они выполняют одинаковую роль, а имеют различные внешний вид и применение. Необхо- димо отметить, что в языке **Delphi** есть большое количество компонентов, имеющих одинаковую об- ласть применения, однако отличающихся некоторыми частными внешними особенностями и доступ- ными для применения методами.
  2. Сохраните изменения в проекте.

## Контрольное задание 1.

* + 1. Разработайте приложение, которое при выборе определенного цвета в компоненте **ComboBox**,из- меняет цвет формы.



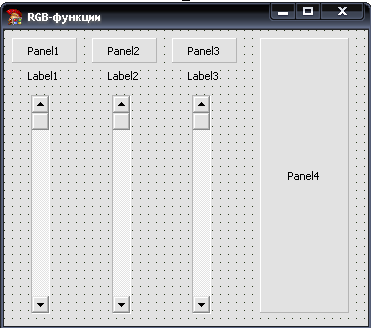
* + 1. Разработайте приложение, которое при выборе определенного цвета в компоненте **RadioGroup**,из- меняет цвет формы.

## Задание 3.

Разработать проект демонстрации работы RGB – функций (установок цвета по трем составляю- щим) с помощью полос прокрутки. Каждый бегунок полос прокрутки должен будет менять вклад RGB – компонента, отображающийся на панели как цвет, а на метке как число. Результи- рующий цвет долженотображаться на панели.

## Методические указания по выполнению задания:

1. Запустите интегрированную среду разработчика **Turbo Delphi**. Создайте новый проект.
2. Создайте в папке своей группы новую папку и назовите её **ScrollBox1**. Сохраните проект в создан- нуюранее папку, выполнив команду **File - Save Project as...**
3. Разместите на форме компоненты в соответствии с рисунком.



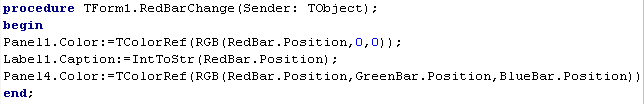
**Полоса прокрутки** – классический элемент оконного интерфейса, использующийся для скроллинга информации, которая не помещается целиком в область вывода. В объектах просмотра и редактиро- вания **Windows** этот элемент появляется при необходимости автоматически.

В палитре VCL-компонент для создания полос прокрутки как самостоятельных элементов введенот- дельный компонент:  **ScrollBar** – самостоятельная полоса прокрутки, может быть горизонталь- ной или вертикальной, что определяется свойством **Kind=sbHorizontal/sbVertital.**

Свойствами **Min** и **Max** можно задать края диапазона целых значений, соответствующих положению бегунка полосы (свойство **Position**). Обычно также выбирают размеры скачков бегунка при щелчке настрелках или на свободной полосе, определяя свойства **SmallChange** и **LargeChange**.

При перемещении бегунка **ScrollBar** генерируется событие **OnChange** (при попытке перемещения – **OnScroll**). При этом можно получать и обрабатывать данные как свойство **Position**. Во время работы программы значения свойств **Position, Min, Max** можно задавать с помощью метода **SetParams**.

1. Задайте для элемента управления **ScrollBar1** значение свойства **Name**=**RedBar** и установить значение свойств **Max=255**, **Position=122**.
2. Задайте для элемента управления **Scrollbar2** значение свойства **Name=GreenBar** и установите значе- ние свойств **Max=255**, **Position=122**.
3. Задайте для элемента управления **Scrollbar3** значение свойства **Name=BlueBar** и установите значение свойств **Max=255**, **Position=122**.
4. Выделите элемент **RedBar,** в **Object Inspector** перейдите на вкладку **Events**. Найдите событие **OnChange,** справа от него в поле сделайте двойной щелчок левой кнопкой мыши. Оказавшись в коде программы, введите следующий код:



Функция **TColorRef** преобразует значение цветовых составляющих в число типа **LongInt**.

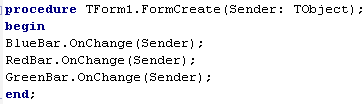
1. Аналогично запишите процедуры обработки события **OnChange** для элементов **GreenBar** и

**BlueBar**.

1. Задайте имя формы проекта **RGB-функции**. Создайте кнопку **Close**.
2. Вызов процедур обработки событий **OnChange** поместите в событие при создании формы

**OnCreat**

для **Form1**.

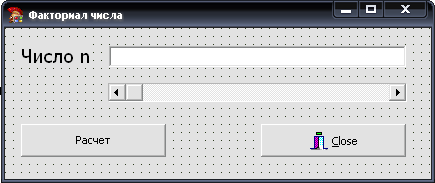
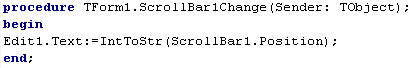


1. Сохраните изменения и запустите проект, убедитесь в его работоспособности.

## Задание 4.

Разработать проект, который позволяет пользователю вычислить факториал числа. Число, для которогорассчитывается факториал, выбирается с помощью вертикальной полосы прокрутки. При щелчке по кнопке «Расчет», меняется надпись «Число n» на «N!» и в строке ввода выво- дится значение факториалачисла.

## Методические указания по выполнению задания:

* 1. Запустите интегрированную среду разработчика **Turbo Delphi**. Создайте новый проект.
  2. Создайте в папке своей группы новую папку и назовите её **ScrollBox2**. Сохраните проект всо- зданную ранее папку, выполнив команду **File - Save Project as...**
  3. Разместите на форме компоненты в соответствии с рисунком.
  4. Для элемента управления **ScrollBar1** установите значение свойств **Max=10**, **Position=5**.
  5. Выделите элемент **ScrollBar1**, в **Object Inspector** перейдите на вкладку **Events**. Найдите событие **OnChange**, справа от него в поле сделать двойной щелчок левой кнопкой мыши. Оказавшись в коде программы, ввести следующий код:
  6. Запишите процедуру обработки события **OnCliсk** кнопки «**Расчет**». При расчете факториала числавоспользоваться конструкцией цикл.
  7. Задайте имя формы проекта **Факториал числа**. Создайте кнопку **Close**.
  8. Сохраните изменения в проекте.

**Практическое занятие № 18.** Создание проекта с использованием компонентов стандарт- ных диалогов и системы меню

**Цель работы:** сформировать умения использования компонентов стандартных диалогов и системы меню,изучить основные свойства; сформировать умения по созданию проце- дур на основе событий компонентов.

**Оборудование, технические и программные средства:** персональный компьютер, средапро- граммирования **Visual Studio**.

## Задание:

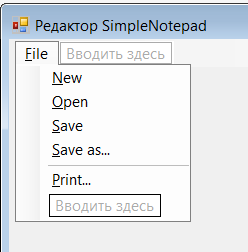
Разработать приложение **SimpleNotepad**, представляющее собой простейший текстовый редактор.Использовать в приложении компонент для работы с меню и стандартные окна диалога.

## Методические указания по выполнению задания:

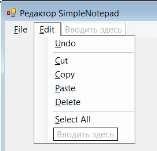
1. Запустите среду программирования **Visual Studio**. Создайте новое **Приложение Windows Forms**. Имяпроекта и приложения **SimpleNotepad**.
2. Измените свойство **Name** формы на **SimpleNotepadForm.**
3. Перейдите в окно **Обозреватель решений** и переименуйте файл **Form1.cs** на

**SimpleNotepadForm.cs.**

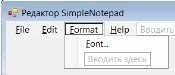
1. Измените размеры окно формы. Измените заголовок окна на **Редактор SimpleNotepad**.
2. Добавьте в окно приложения элемент управления **MenuStrip**.
3. В поле **Вводить здесь** меню введите строку **&File**. В результате в окне приложения появится меню **File**.Обратите внимание, что первая буква в названии подчеркнута. Это получилось по- тому, что перед ней стоит префикс &. Этим префиксом отмечается буква, предназначенная для ускоренного выбора меню при помощи клавиатуры.
4. Создайте меню **File** в соответствии с рисунком. Для вставки разделительной черты необходимо вызватьконтекстное меню и выбрать команду **Separator**.



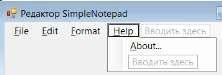
1. Создайте меню **Edit** в соответствии с рисунком. В меню **Edit** реализованы стандартные функ- ции приложения **Блокнот**.



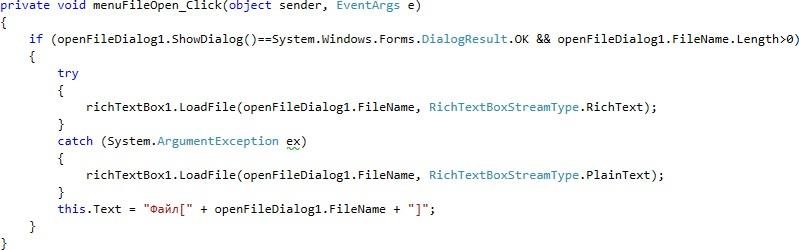
1. Меню **Format** состоит только из одной строки **Font**, с помощью которой пользователь может изменитьшрифт текста.



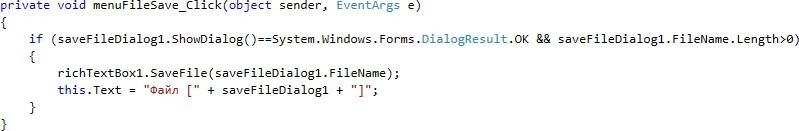
1. Меню **Help** состоит только из одной строк **About**.



1. Переименуйте идентификаторы меню и строк меню таким образом, чтобы с ними было удоб- нее работать в программе. Для этого в окне дизайнера формы щелкните правой кнопкой мыши главное меню приложения и затем выберите из контекстного меню строку **Правка DropDownItems**. В окне **Редактор коллекций элементов** отредактируйте имена меню и строк меню, изменив значение свойства **Name** соответствующего элемента. При этом меню верхнего уровня должны называться **menuFile**, **menuEdit**, **menuFormat**, **menuHelp**. Имена строк формируются путем добавления к имени меню текста, отображаемого в строке меню. Например, строка **New** называется **menuFileNew**.
2. Запустите приложение. Убедитесь, что меню корректно отображается, и вы можете выбирать его строки.
3. Редактирование текста в приложении будет выполнять компонент **RichTextBox**. Перетащите компонент **RichTextBox** с панели элементов в окно приложения. Задайте значение свойства **Dock** данного компонента равным **Fill**, для того чтобы он занимал все окно приложения.
4. Создайте обработчики событий, необходимые для выполнения таких функций как создание нового документа, открытие и сохранение документа.
5. Добавьте на форму компонент **OpenFileDialog** с вкладки **Диалоговые окна** панели элемен- тов. Выделите пункт меню **Open...** и дважды щелкните по нему левой кнопкой мыши, для того чтобы добавить обработчик события. В обработчик события добавьте следующий код для отображениядиалогового окна открытия файла.



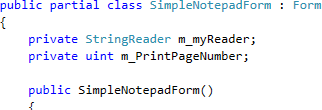
1. Выделите компонент **openFileDialog1**, на панели **Свойства** найдите свойство **Filter** и введите следующую строку **RTF files|\*.rtf|Text files|\*.txt|All files|\*.\***
2. Добавьте на форму компонент **SaveFileDialog** с вкладки **Диалоговые окна** панели элементов. Выделите пункт меню **Save As…** и дважды щелкните по нему левой кнопкой мыши, для того чтобы добавить обработчик события. В обработчик события добавьте следующий код для отображениядиалогового окна **Сохранение файла**. Этот код сохраняет текст введенный в эле- мент управления **richTextBox**, в текстовый файл в указанной папке. Метод **ShowDialog** отоб- ражает диалоговое окно, а затем с помощью поля **DialogResult**.**OK** осуществляется проверка нажата ли пользователем кнопка **ОК**.



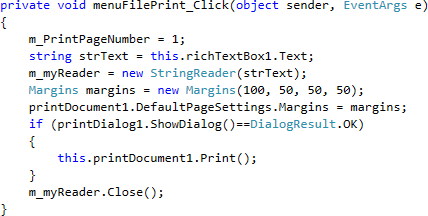
1. Выделите компонент **saveFileDialog1**, на панели **Свойства** найдите свойство **Filter** и введите следующую строку **RTF files|\*.rtf**, найдите свойство **FileName** и задайте значение **doc1.rtf**
2. Добавьте в окно дизайнера форм компоненты **PrintDocumet**, **PrintDialog** вкладки **Печать** па- нели элементов.



1. Компонент **PrintDocumet** предназначен для вывода данных документа на принтер. Свойства компонента **PrintDocumet** описывают, как именно нужно распечатывать документ. В свой- ство **DocumentName** данного компонента запишите строку **SimpleNotepad Document**. Эта строка будет идентифицировать документ при отображении состояния очереди печати. Значе- ния остальных свойствоставьте без изменения.
2. С помощью компонента **PrintDialog** приложение выведет на экран стандартное диалоговое окно печатидокумента. Задайте для данного компонента значение свойства **Document** равным **printDocument1**. Этим обеспечивается связь компонента **printDialog** с компонентом **PrintDocumet**.
3. Для работы с классами, предназначенными для выполнения операций с потоками и печати, добавьте вначало программы следующие строки:
4. Добавьте в класса **SimpleNotepadForm** поля **m\_myReader** (для печати содержимого редак- тора текста)и **m\_PrintPageNumber** (номер текущей распечатываемой страницы документа).



1. Создайте обработчик события печати документа.



Печать документа будет начинаться с первой страницы, поэтому в поле **m\_PrintPageNumber** за- писываетсязначение 1. Далее выполняется чтение текущего содержимого окна редактирова- ния текста в поток **m\_myReader** класса **StringReader**. Далее задаются границы отступов на

распечатываемой странице и отображается диалоговое окно печати документа. Если пользо- ватель щелкает в этом окне кнопку **OK**, документ **printDocument1** отправляется на печать методом **Print**. Далее ненужный более поток **m\_myReader** закрывается методом **Close**.

1. На данном этапе приложение еще не в состоянии распечатать документ. Причина этого в том, что приложение пока еще не знает, каким именно образом нужно печатать документ.
2. Чтобы в нашем приложении заработала функция печати, необходимо создать обработчик со- бытия **PrintPage**. Для этого нужно дважды щелкнуть левой клавишей мыши значок компо- нента **printDocument1**. В тело обработчика событий вставьте программный код из текстового файла **Print.txt**.Комментарии в тексте обработчика событий **PrintPage** поясняют назначение отдельных программныхстрок. Заметим, что для полного понимания действий, выполняемых нашим обработчиком событий, требуется предварительное знакомство с графической подси- стемой **Graphics Device Interface Plus (GDI+)**, реализованной компанией **Microsoft** в рамках библиотеки классов **.NET Framework**. Пока женужно отметить, что приложение распечаты- вает текст построчно в цикле. После завершения печати всех строк текущей страницы обра- ботчик событий **PrintPage** печатает верхний и нижний колонтитулы,а также рисует горизон- тальные линии, отделяющие текст колонтитулов от текста документа.
3. Измените меню **File**, добавив пункт **Exit,** при выборе которого окно редактора текста должно быть закрыто. Добавьте обработчик события для данного пункта меню:



Однако при закрытии окно приложения возникает проблема: окно редактора текста будет закрыто и в том случае, если пользователь не сохранил сделанные им изменения. Чтобы решить эту проблему, намнужно каким-то образом отслеживать наличие изменений в окне редактирования текста. Опреде- лите вклассе **SimpleNotepadForm** поле **m\_DocumentChanged**, в котором будет храниться флаг, отме- чающийизменения, сделанные пользователем в документе. В новом или только что загруженном до- кументе изменений нет, поэтому начальное значение этого флага равно **false**.



1. Создайте обработчик события **richTextBox1\_TextChanged**, выполнив двойной щелчок по компоненту**richTextBox**. Этот обработчик получит управление, как только пользователь вне- сет любые изменения в содержимое редактируемого документа.

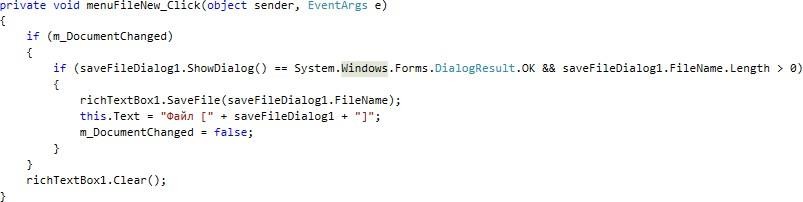


1. Если пользователь редактировал документ, а потом решил создать новый, выбрав из меню

**File** строку

**New**, изменения, внесенные в старый документ, могут быть потеряны.

1. Чтобы избежать этого, проверьте флаг **m\_DocumentChanged** перед тем как очищать содер- жимое редактора текста. Если в редакторе есть не сохраненные изменения, необходимо выпол- нить сохранениедокумента. Отредактируйте код обработчика события пункта меню **New** сле-

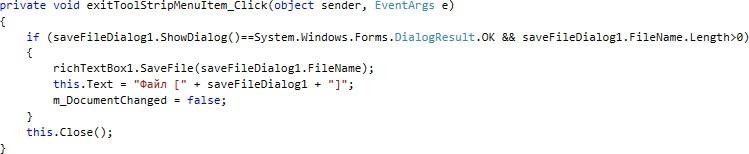


дующим образом:

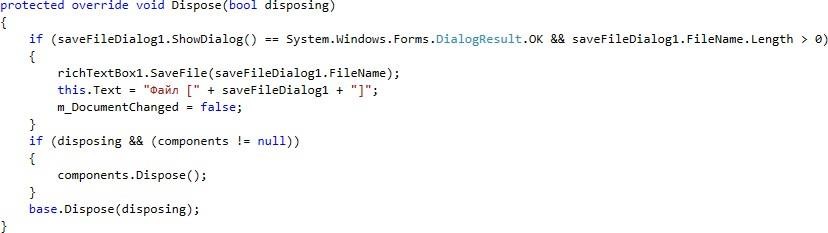
1. Отредактируйте код обработчика события пунктов меню **Save** и **Save As**, добавив строку кода

## m\_DocumentChanged=false

1. Измените обработчик события пункта меню **Exit** следующим образом:

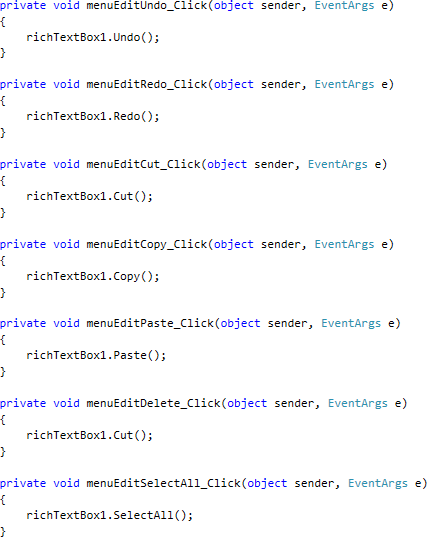


1. Надо выполнить еще одну проверку флага **m\_DocumentChanged** в методе **Dispose**, который вызывается при закрытии окна приложения. Для этого на панели Обозреватель решений вы- делите **SimpleNotepadForm.Designer.cs**, найдите метод **Dispose** и внесите необходимые изме- нения.



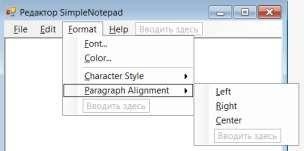
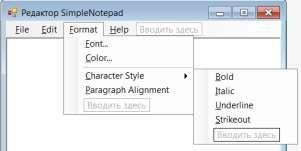
Теперь, когда пользователь попытается закрыть программу с помощью соответствующей кнопки заголовкаокна, не сохранив сделанные изменения, на экране появится стандартное диалоговое окно,предлагающее ему сохранить документ.

1. Создайте обработчики событий пунктов меню **Edit**. Реализация данных функций является

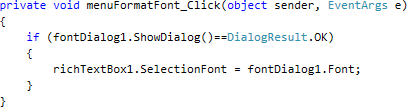


простой, поскольку элемент управления **RichTextBox** имеет все необходимые методы. До- бавьте в пункт меню **Edit** строку **Redo**. Подготовьте обработчики событий для всех строк меню **Edit**.

1. Измените меню **Format**, добавив в него строку **Color**, а также два меню второго уровня – **Character Style** и **Paragraph Alignment**. Измените имена строк меню таким образом, чтобы с ними было легче работать в программе.

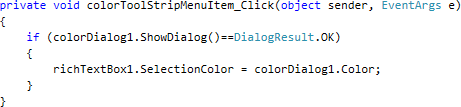


1. Добавьте на форму компонент **FontDialog** с вкладки **Диалоговые окна** панели элементов. Этот компонент отображает на экране стандартное диалоговое окно выбора шрифта. Создайте обработчик события пункта меню **Font** и вставьте следующий код:



После того как пользователь выбрал нужный ему шрифт, обработчик события переписывает этот шрифт из свойства **fontDialog1.Font** в свойство **richTextBox1.SelectionFont**. Свойство **SelectionFont** позволяет изменить шрифт фрагмента текста, выделенного пользователем (или программой) в окне редактирования.

1. Добавьте на форму компонент **ColorDialog** с вкладки **Диалоговые окна** панели элементов. Этот компонент отображает на экране стандартное диалоговое окно выбора шрифта. Создайте обработчик события пункта меню **Color** и вставьте следующий код:

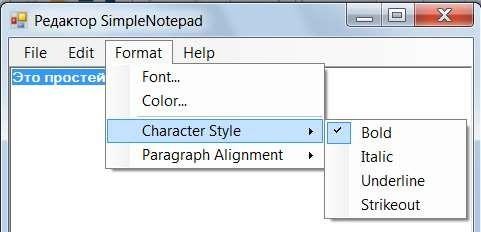


1. Создайте обработчик события для пункта меню **Bold** и вставьте следующий код:

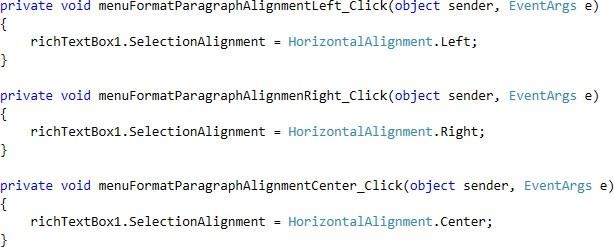


Получив управление данный метод прежде всего, определяет шрифт фрагмента текста, выделен- ного пользователем, анализируя свойство **richTextBox1.SelectionFont**. Если шрифт опреде- лить не удалось, и это свойство содержит значение **null**, программа не делает никаких измене- ний. В противном случае программа сохраняет текущий шрифт в переменной **currentFont** класса **System.Drawing.Font**. Далее метод проверяет, был ли выделен фрагмент текста жир- ным шрифтом, анализируя свойство**richTextBox1.SelectionFont.Bold**. Если это свойство со- держит значение **true**, то метод снимает выделение, если нет, то устанавливает его. Для снятия выделения программа записывает в переменную**newFontStyle** значение **FontStyle.Regular**, а для установки — значение **FontStyle.Bold**.

1. При выделении фрагмента текста жирным шрифтом в меню одновременно отмечается «га- лочкой» строка **Bold**. Для этого необходимо создать **CheckMenuFontCharacterStyle**. Для этого перейдите в окно кода и вставьте программный код процедуры из текстового документа **CheckMenu.txt.** Метод **CheckMenuFontCharacterStyle** по очереди проверяет стилевое оформление выделенных фрагментов, отмечая «галочками» соответствующие строки меню **CharacterStyle** или снимая со строк этого меню отметки.
2. Запустите программу и протестируйте её работу.



1. Аналогично создайте обработчики для пунктов меню **Italic, Underline, Strikeout**.
2. Подготовьте обработчики событий для строк меню **Paragraph Alignment** следующим обра-



зом:

Все эти обработчики событий строк меню **Paragraph Alignment** изменяют значение свойства **richTextBox1.SelectionAlignment**, задающего выравнивание параграфа текста, выделенного пользователем (для выделения параграфа с целью изменения выравнивания достаточно уста- новить в него текстовый курсор).

1. Сохраните изменения и протестируйте работу приложения.

**Практическая работа №19.** Создание процедур на основе событий

**Цель работы:** сформировать умения использования компонентов стандартных диалогов и си- стемы меню,изучить основные свойства.

**Оборудование, технические и программные средства:** персональный компьютер, средапрограммиро- вания **Visual Studio**.

## Задание

**Методические указания по выполнению задания:**

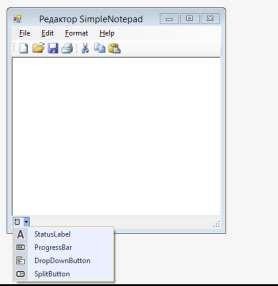
1. Во многих приложениях, созданных для операционной системы **Windows**, наиболее часто ис- пользующиеся строки меню дублируются кнопками, расположенными на инструментальных панелях. Чтобы добавить инструментальную панель в окно приложения, перетащите мышью элемент управления **ToolStrip** вкладки **Все формы Windows Forms** панели элементов в окно проектирования формы нашего приложения. По умолчанию окно инструментальной панели по- явится в верхней части формы. В только что добавленной панели нет ни одной кнопки.
2. Сразу после добавления окно инструментальной панели будет расположено над окном редак- тора текста. Чтобы исправить это положение, щелкните правой кнопкой мыши окно редактора текста, а затем выберите из контекстного меню строку **На передний план**. В результате окна примут правильноевзаимное расположение.
3. Создайте в папке проекта отдельную папку для хранения изображений инструментальной па- нели, назвав ее, например, **ImageList**. Далее скопируйте в созданную папку изображения, под- готовленные для кнопок панели.
4. Выделите компонент **ToolStrip** и на панели свойств найдите свойство **Items,** щелкните по кнопке .Откроется окно редактора коллекций.
5. В диалоговом окне **Редактор коллекции элементов** в списке элементов выберите **Button** и нажмите кнопку **Добавить**. На панели свойств измените значение свойства **Name** данной кнопки на **создатьToolStripButton**, а размер кнопки установите равным **24**.
6. В окне свойств найдите свойство **Image**. Нажмите на кнопку , в диалоговом окре **Выбор ресурса** нажмите на кнопку **Импорт** и выберите соответствующее изображение из сохранён- ных ранее в папке **ImageList**.
7. Задайте комментарий-подсказку для кнопки **создатьToolStripButton**, установив свойство

**ToolTripText** равным **Создать**.

1. Аналогично создайте кнопки **открытьToolStripButton, сохранитьToolStripButton, пе- чатьToolStripButton.**
2. Далее необходимо создать разделитель между группами кнопок, ответственных за выполнение различных групп операций. Для этого в списке элементов выберите **Separator** и нажмите кнопку **Добавить**.
3. Далее аналогично создайте кнопки **вырезатьToolStripButton, копироватьToolStripButton, вставитьToolStripButton**.
4. Теперь необходимо связать созданные ранее функции-обработчики события команд пункт меню с соответствующими кнопками панели инструментов. Выделите кнопку **со- здатьToolStripButton,** на панели **Свойства** перейдите на вкладку **События **, найдите со- бытие **Click** и в списке выберите функцию **menuFileNew\_Click**. Аналогично добавьте функции для остальных кнопок панели инструментов.
5. Сохраните приложение. Запустите его и протестируйте работу кнопок панели инструментов.

## Задание 2. Создание строки состояния Методические указания по выполнению задания:

1. Разместите на форме компонент **StatusStrip** со вкладки **Меню и панели инструментов** па- нелиэлементов.
2. В меню компонента **StatusStrip** выберите опцию **StatusLabel**.

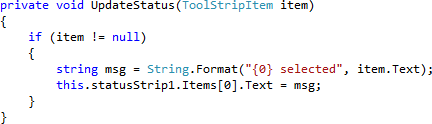


1. Далее на панели **Свойства** найдите свойство **Text** и сделайте его пустым.
2. Для того чтобы отслеживать выбор строк меню необходимо предусмотреть специальный обра- ботчик события **DropDownItemClicked**. Это событие создается строками меню, когда пользова- тель выбирает их при помощи мыши или клавиатуры. Создайте обработчик событий. Для этого выделите мышью меню **File** в окне дизайнера формы, а затем перейдите на вкладку **События** па- нели **Свойства**. В спискесобытий найдите событие **DropDownItemClicked** и дважды щелкните по строке рядом с ним. После этого будет создано пустое тело обработчика события. Вставьте следующий код в обработчик события:



1. Создайте пустую строку перед обработчиком данного события и опишите метод

**UpdateStatus**

следующим образом:

1. Выделите мышью меню **Edit** в окне дизайнера формы, а затем перейдите на вкладку **События** панели **Свойства**. В списке событий найдите событие **DropDownItemClicked** и в списке рядом с ним выберите событие **menuFile\_DropDownItemClicked**. Аналогично выполните для всех остальных пунктов меню.
2. Сохраните приложение. Запустите его и протестируйте работу строки состояния.

**Практическая работа №20.** Разработка оконного приложения с несколькими формами

**Цель работы:** сформировать практические умения по созданию простейшего приложения с несколькимиэкранами в среде **MIT App Inventor**

**Оборудование, технические и программные средства:** персональный компьютер, программа брау- зер.

## Задание 1.

Используя инструменты среды **MIT App Inventor**, создать приложение, в котором при щелчке по кнопкеодин объект превращается в другой.

## Методические указания по выполнению задания:

1. Запустите программу браузер и перейдите по ссылке **ai2.appinventor.mit.edu**. Создайте новый проект, назовите его **Transformation**.
2. Создание приложения с несколькими экранами аналогично созданию нескольких отдельных приложений. Количество создаваемых в приложении экранов определяется разработчиком, но не может быть более 10. По умолчанию в момент создания нового проекта в нем всего один экран.

Компоненты каждого экрана создаются в режиме **Дизайнер** для этого экрана. В режиме

# Блоки

отображаются блоки, только для компонент текущего экрана.

При создании приложений с несколькими идентичными экранами, используйте функцию копи- рования блоков между экранами.

Навигация (переход) между экранами организуется с помощью кнопок или с помощью дей- ствий.

Каждый экран, закрывается в случае перехода на другой или возвращении на тот, с которого он был открыт.

Экраны могут обмениваться информацией путем принятия и возвращения значений, когда ониоткрываются и закрываются. Экраны могут использовать для хранения и получения с дру- гих экранов данных только с использованием **TinyDB**, к примеру счет игры, количество набран- ных баллов в тесте. **TinyDB** невидимый компонент **App Inventor**, который хранит данные непосредственно в устройстве.

Компонент **TinyDB** используется внутри приложения для передачи данных между экранами, В этом их отличие от глобальных переменных, которые сохраняются в пределах одного экрана, пока приложение работает.

Приложения, созданные с помощью **App Inventor** инициализируются каждый раз заново, когда они выполняются. Это означает, что если приложение устанавливает значение переменной, а затем пользователь выходит из приложения, значение этой переменной теряется. В отличие от этого, **TinyDB**является стойким хранилище данных для приложений.

Данные, хранящиеся в **TinyDB** будут доступны каждый раз, когда приложение запускается. Каждая переменная сохраняется под собственным именем.

Каждое приложение имеет собственное хранилище данных. С помощью **TinyDB** вы не можете получить доступ к данным другого приложения. Но отсюда же следует. Что для каждого при- ложения имеется только одно хранилище. То есть, если сохранить переменную под именем «X», то в данном приложении будет только одна переменная под этим именем, но она будет недо- ступна для других приложений.

Свойств данный компонент не имеет

Компонент **TinyDB** очень полезен, поскольку позволяет сохранить данные приложения на

# Android

устройстве. Обычно такие небольшие данные используются для сохранения настроек прило- жения.

После того как данные сохранены в **TinyDB** они останутся там, пока **TinyDB** не будет очи- щено.

1. На панели **Свойства** установите выравнивание всех объектов **Screen1** по центру, для этого выберите свойство **ВыровнятьПоГоризонтали** и задайте его значение равным **Центр**.
2. В палитре выберите компонент **Надпись** и переместите в окно **Screen1**. На панели **Свойства** поменяйте текст надписи на **Превращение в сказке**. Измените размер надписи на **28 пт**.
3. На панели **Компоненты** выделите **Надпись1** и нажмите внизу панели кнопку **Переименовать**, задайте имя компонента **НадписьЗаголовок**.
4. В палитре выберите компонент **Кнопка** и переместите в окно **Screen1**. На панели **Свойства** поменяйте текст кнопки на **Превратить в комара**. Измените размер шрифта надписи на **16 пт**.
5. На панели **Компоненты** выделите **Кнопка1** и нажмите внизу панели кнопку **Переименовать**, задайте имя компонента **КнопкаВКомара**.
6. Перетащите компонент **Изображение** в окно экрана. Установите в качестве источника изображения файл **gvidon.jpg**. Нажмите кнопку **Переименовать** и дайте компоненту название **ИзображениеГвидон**.



1. Нажмите кнопку **Добавить экран**. Перейдите на экран **Screen2**. Создайте по аналогии дизайн экрана.
2. Перейдите на экран **Screen1**. Перейдите на вкладку **Блоки**. Соберите из блоков следующуюкон- струкцию.



1. Перейдите на экран **Screen2**. Перейдите на вкладку **Блоки**. Соберите из блоков следующуюкон- струкцию.



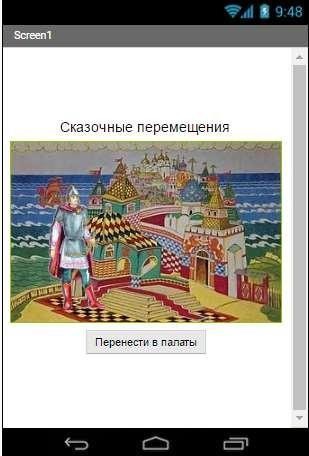
1. Протестируйте работу.

## Задание 2.

Используя инструменты среды **MIT App Inventor,** создать приложение, в котором происхо- дитперемещение объекта с одного экрана на другой.

## Методические указания по выполнению задания:

1. Создайте новый проект, назовите его **Moving**. Установите в параметрах **Screen1** выравнивание по центру.
2. В палитре выберите компонент **Надпись** и переместите в окно **Screen1**. На панели **Свойства** поменяйте текст надписи на **Сказочные перемещения**. Измените размер шрифта надписи на **18 пт**.
3. На панели **Компоненты** выделите **Надпись1** и нажмите внизу панели кнопку **Переименовать**, задайте имя компонента **НадписьЗаголовок**.
4. В палитре перейдите в раздел **Рисование и анимация**. Выберите компонент **Холст** и переместите егов окно **Screen1**.
5. На панели **Свойства** установите размеры холста: ширина - **300 px**, высота - **200 px**.
6. Загрузите файл изображения **ostrov.jpg**. Установите его в качестве фонового рисунка для холста.
7. В палитре в разделе **Рисование и анимация** выберите компонент **ИзображениеСпрайт** и переместите его на **Холст**.
8. Загрузите файл изображения **gvidon.png** и установите загруженное изображение как изображение спрайта.
9. В палитре выберите компонент **Кнопка** и переместите в окно **Screen1**. Выделите компонент **Кнопка** и нажмите **Переименовать**. Введите текст **КнопкаПеремещение**. Измените текст надписи на **Перенестив палаты**.



## Screen1.

11. Перейдите в режим **Блоки** и соберите следующую конструкцию:



Разберем блоки конструкции более подробно. Вначале осуществляется инициализация глобальной переменной с изображением Гвидона=0, которая будет передаваться с экрана на экран, именно она и будет храниться в **TinyDB**.

Как только **Кнопка** будет нажата этой переменной необходимо присвоить **ИзображениеСпрайта** изображение Гвидона, который будет перемещаться. После перемещения изображение не должно отображаться на экране, поэтому ему присваивается параметр **Видимый – Ложь**.

Далее вызывается компонент **TinyDB**, задается изображению **ТЕГ** (имя) под которым он будет хра- ниться на устройстве, и выбирается значение, которое нужно сохранить **GlobalImage**.

Выполняется переход на другой экран.

12. Нажмите кнопку **Добавить экран**. Перейдите на экран **Screen2**. Создайте по аналогии дизайн экрана. Обратите внимание, что изображения Гвидона до нажатия кнопки **Поместить в палаты** нет.



Разберем блоки конструкции более подробно. Вначале также осуществляется инициализация гло- бальной переменной с изображением Гвидона=0, которая будет передаваться с экрана на экран, именно она и будет храниться в **TinyDB**.

Далее присваивается **ИзображениюСпрайта** изображение Гвидона, которое берётся из **TinyDB**. При полу- чении значения - задается только **ТЕГ**.

В случае если допущена ошибку и изображение во внешнем хранилище не найдено, выдается со- общение, что оно не найдено.

14. Протестируйте работу приложения.

## Задание 3.

Используя инструменты среды **MIT App Inventor,** разработайте контентное приложение с ис- пользованиемменю. Для создания приложения, используйте материалы папки **SourceOcean**.



**Практическое занятие № 21.** Разработка игрового приложения.

**Цель работы:** сформировать практические умения создания многооконного приложения cс фреймамисредствами среды программирования **Visual Studio**.

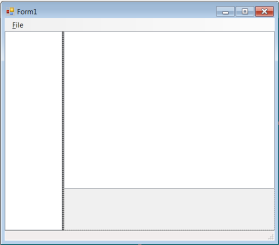
**Оборудование, технические и программные средства:** персональный компьютер, средапрограмми- рования **Visual Studio**.

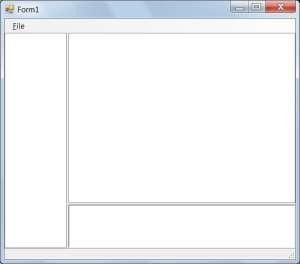
## Задание 1. Создание главного окна приложенияМетодические указания по выполне- нию задания:

1. Запустите среду программирования **Visual Studio**. Создайте новое **Приложение Windows Forms**. Имя проекта и приложения **FramesApp**.
2. Добавьте в форму главного окна приложения меню (**MenuStrip**) и строку состояния (**StatusStrip**). В меню **File** создайте строку **Exit**, предназначенную для завершения работы приложения. Напишите об- работчик данного события.



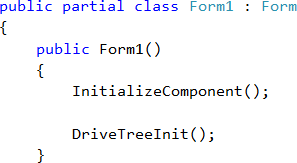
1. Добавьте в окно нашего приложения элемент управления **TreeView**, перетащив мышью его зна- чок из панели элементов вкладки **Все формы Windows Forms** в окно формы. Установите значение свойства **Dock** данного элемента управления равным **Left**. В результате элемент управления **TreeView** будет выровнен по левой границе главного окна приложения. В нашем приложении окно элемента управления**TreeView** будет использовано для отображения списка дисков и каталогов.
2. Для элемента управления **TreeView** создайте разделитель. Для этого перетащите значок элемента управления **Splitter** из панели элементов вкладки **Все формы Windows Forms** в окно формы. Раздели- тель будет автоматически расположен справа от окна элемента управления **TreeView**.
3. Элемент управления **Splitter** имеет свойства, которые можно менять в процессе разработки при- ложения, а также динамически во время его работы. Самое важное свойство разделителя **Splitter** —это свойство **Dock**, задающее его расположение. По умолчанию значение этого свойства равно **Left**, бла- годаря чему разделитель разместился слева от окна дерева **TreeView**. При необходимости можно из- менять расположение разделителя с помощью окна редактирования, аналогичного окну.
4. Перетащите значок элемента управления **Panel** из панели элементов вкладки **Все формы Windows Forms** в окно формы, а затем установите значение свойства **Dock** этой панели равным **Fill**. В результатенаша панель будет расположена справа от разделителя и займет всю правую часть окна при- ложения.
5. В верхнем окне будет находиться элемент управления **ListView**, показывающий список файлов и каталогов. Добавьте в окно приложения элемент управления **ListView**, перетащив его значок из панели элементов вкладки **Все формы Windows Forms** в окно формы. После установки значения свойства **Dock**, равным **Top**, и увеличения высоты окна этого элемента управления, главное окно нашегоприло- жения примет следующий вид.



1. Добавьте в правую часть главного окна горизонтальный разделитель **Splitter**. Перетащите его пиктограмму из панели элементов вкладки **Все формы Windows Forms** в окно формы, а затем устано- вите значение свойства **Dock** добавленного разделителя, равным **Top**. В результате разделительбудет расположен непосредственно под окном элемента управления **ListView**.
2. В нижней части окна приложения будет находиться элемент управления **RichTextBox**,предназна- ченный для отображения дополнительной информации. Перетащите значок элемента управления **RichTextBox** из панели элементов вкладки **Все формы Windows Forms** в окно формы и установите значение свойства **Dock** равным **Fill**.
3. Протестируйте работу приложения. Сохраните полученный проект. Не написав ни единой строчки кода, вы создали простейшее приложение с действующим многооконным интерфейсом поль- зователя.

## Задание 2. Создание методов для элемента управления TreeViewМетодические указания по выполнению задания:

1. Инициализация дерева **TreeView** осуществляется в конструкторе класса **Form1**. Для этого необ- ходимоподготовить метод **DriveTreeInit**.

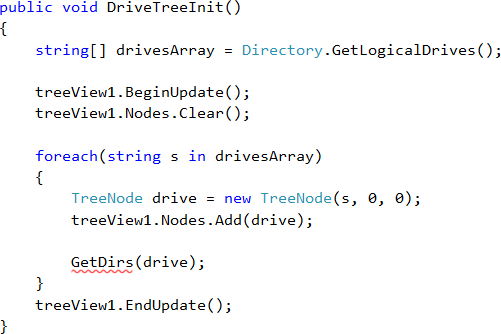


1. Для обращения к этому методу, а также к другим методам, работающим с дисками, каталогами и

файлами, подключите пространство имен **System.IO**.



1. Перейдите в окно редактирования кода и создайте метод **DriveTreeInit**:

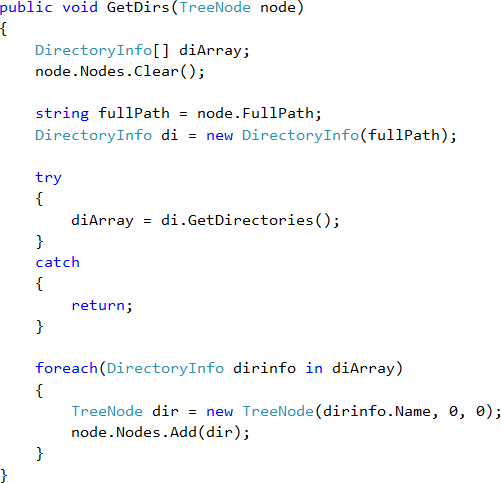


В самом начале своей работы этот метод получает список логических дисковых устройств, установленныхв системе, и сохраняет его в массиве **drivesArray**. Далее метод **DriveTreeInit** вызывает метод **treeView1.BeginUpdate**. Этот метод временно блокирует перерисовку окна дерева до тех пор, пока не будет вызван метод **treeView1.EndUpdate**. Пара этих методов ис- пользуется в том случае, когда нужнодобавить, удалить или изменить большое количество элементов дерева. Если не заблокировать перерисовку окна, на обновление дерева уйдет слиш- ком много времени.

В классе **TreeView** определено свойство **Nodes**, хранящее все узлы дерева. Перед тем как приступить к заполнению дерева, метод **DriveTreeInit** удаляет все узлы, вызывая для этого метод **treeView1.Nodes.Clear**.

Заполнение дерева происходит в цикле. Массив **drivesArray** содержит массив текстовых строк с обозначениями логических устройств, установленных в системе. Для каждого такого устрой- ства в цикле создается объект класса **TreeNode** — узел дерева. В качестве первого параметра конструктору передается текстовая строка названия устройства. Созданный узел добавляется в набор узлов дерева с помощью метода **treeView1.Nodes.Add**. В качестве параметра этому методу передается ссылка на объект **TreeNode**, т.е. на добавляемый узел. Далее вызывается метод **GetDirs**, добавляющий в дерево список содержимого корневого каталога текущего дис- кового устройства.

1. Метод **GetDirs** получает в качестве параметра ссылку на узел дерева, который требуется напол- нить списком имен каталогов и файлов. Создайте метод **GetDirs.**



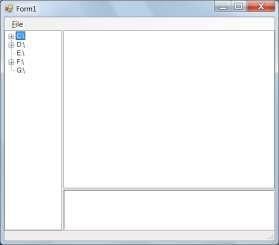
Первым делом метод **GetDirs** удаляет все элементы из текущего узла, вызывая для этого метод

## node.Nodes.Clear.

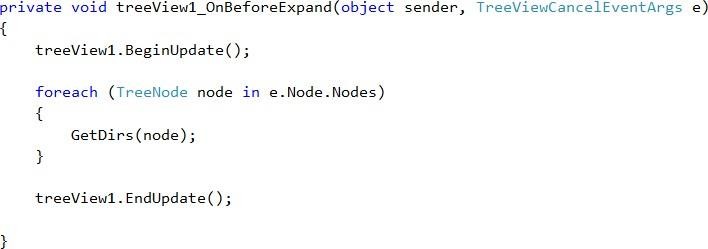
Далее метод переписывает в переменную **fullPath** типа **string** полный путь **node.FullPath** к узлу дерева. Этастрока получается объединением текстовых строк всех родительских узлов. Если корневой узел храниттекстовую строку обозначения логического диска, то после выполнения этой процедуры в переменной**fullPath** будет храниться полный путь к файлу или каталогу.

Далее для получения содержимого каталога, на который ссылается переменная **fullPath**, ис- пользуется метод **GetDirectories**. При возникновении исключения программа просто возвра- щает управление, не выполняя над узлом дерева никаких функций.

В том случае если содержимое каталога было успешно получено, оно сохраняется в массиве **diArray**. Далеесодержимое массива **diArray** используется для заполнения узла дерева содер- жимым каталога.

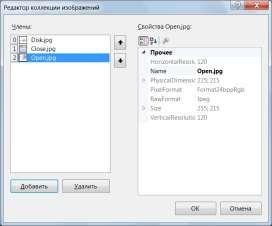
1. Запустите программу на исполнение. В окне дерева появится список дисковых устройств. Рас- крываются ли узлы дерева при щелчке мышью?
2. Чтобы узлы дерева раскрывались, когда пользователь щелкает их мышью или пытается раскрыть

при помощи клавиатуры, нужно обрабатывать событие **BeforeExpand**. Для создания обработчика этого события выделите дерево в окне дизайнера формы, а затем откройте вкладку событий. Далее в поле **BeforeExpand** ведите имя обработчика событий **treeView1\_OnBeforeExpand**. После этого добавьте в этот обработчик событий следующий код:



Событие **BeforeExpand** возникает при попытке пользователя раскрыть узел дерева. В этом случае наш обработчик заполняет открываемый узел при помощи рассмотренного ранее ме- тода **GetDirs**. Ссылка на узел извлекается из поля **e.Node.Nodes**, передаваемого обработчику событий в качестве параметра.

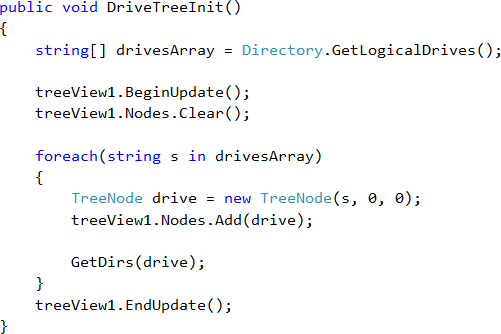
1. Запустите приложение. Раскрываются ли узлы дерева при щелчке мышью теперь?
2. Вы можете улучшить внешний вид дерева, если добавите к его узлам значки дисковых устройств и каталогов (папок). В папке проекта создайте папку **ImageList**. Подготовьте эти значки, используя любой графический редактор. Скопируйте эти значки в папку **ImageList**. Чтобы подключить флажки кпроекту, выберите из меню проект системы **Microsoft Visual Studio** строку **Добавить существую- щийэлемент**, а затем добавьте файлы значков при помощи появившегося на экране диалогового окна.
3. Чтобы использовать изображения в дереве, их необходимо объединить в список класса **ImageList**. Добавьте этот список в приложение, перетащив значок компонента **ImageList** в окно проектирования формы из панели элементов вкладки **Все формы Windows Forms**. Выделив элемент **imageList1** левой клавишей мыши, отредактируйте его свойство **Images**. Для редактирования будет открыто окно **Ре- дактор коллекций изображений**. Воспользуйтесь кнопкой **Добавить** для добавления в список файлов изображений, скопированных ранее в каталог нашего приложения. Изображения следует разместить в следующем порядке: первым (с индексом 0) должно идти изображение для диска, вторым(с индексом 1) — изображение закрытой папки, и третьим (с индексом 2) — изображение открытой папки.



1. Создав и заполнив список изображений, подключите его к дереву просмотра дисков и каталогов. Для этого отредактируйте свойство **ImageList** элемента управления **treeView1**, присвоив ему ссылку на список изображений **imageList1**.
2. Для отображения значков в узлах дерева необходимо изменить исходный текст методов **DriveTreeInit**и **GetDirs**. Первый из этих методов инициализирует дерево, а второй — добавляет к узлу дерева списоккаталогов. Обратите внимание на конструктор класса **TreeNode**, создающий узлы дерева внутри тела цикла **foreach**. Первый параметр задает текст надписи для узла дерева. Если к элементу управления **TreeView** подключен список изображений, то второй и третий параметры конструктора класса **TreeNode** задают индексы изображений для узла дерева. При этом второй параметр определяет

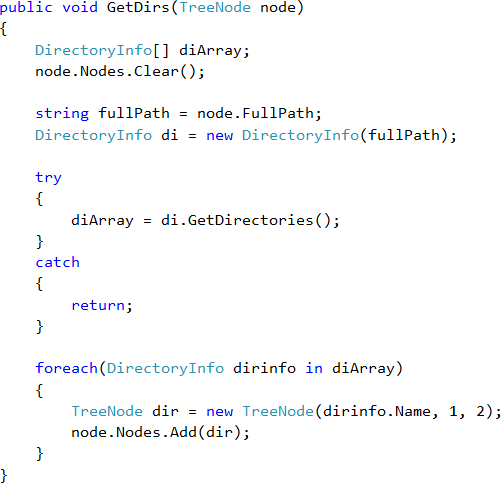
изображение невыделенного узла дерева, а третий — выделенного. Что касается метода **DriveTreeInit**, то расположенный в нем конструктор создает узлы, отображающий только дисковые устройства. В любом состоянии (как выделенном, так и невыделенном) нам необходимо отображать один и тот же

значок дискового устройства, имеющий в нашем случае индекс 0. Поэтому второй и третий

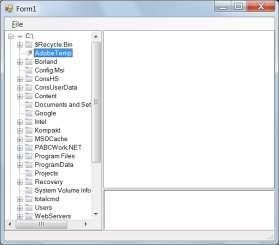


параметрыконструктора передают нулевые значения.

1. В методе **GetDirs** конструктору класса **TreeNode**, размещенному внутри оператора цикла **foreach**, черезвторой и третий параметры передаются индексы значков закрытой и открытой папки, соответ- ственно.



1. Протестируйте работу приложения.



1. Выделите элемент управления **TreeView** на панели Свойства найдите свойство **CheckBoxes** и установите его значение **True**. Теперь узлы дерева будут иметь индивидуальные флажки. Отметив все или некоторые узлы дерева флажками, можно выполнять над соответствующими объектами групповые операции.
2. Выделите элемент управления **TreeView** на панели Свойства найдите свойство **LabelEdit** и уста- новитеего значение **True**. Теперь пользователь может редактировать текстовые надписи, расположен- ные около узлов дерева.

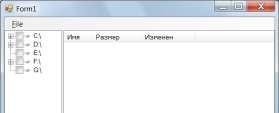
**Практическая работа №22.** Тестирование, отладка приложения

**Цель работы:** сформировать практические умения создания многооконного приложения cс фреймамисредствами среды программирования **Visual Studio**.

**Оборудование, технические и программные средства:** персональный компьютер, средапрограмми- рования **Visual Studio**.

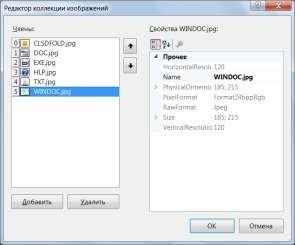
## Задание 3. Создание методов для элемента управления ListViewМетодические указания по выполнению задания:

1. Выделите список **ListView** в окне дизайнера форм и установите для свойства **View** значение **Details**. При необходимости программа сможет динамически переключать режимы отображения, из- меняязначение свойства **View** во время своей работы.
2. Поскольку в приложении будет отображаться список в виде таблицы, то необходимо создать столбцы таблицы и определить их атрибуты. Для этого необходимо отредактировать свойство **Columns**. Это делается при помощи **Редактор коллекции ColumnHeader**.
3. С помощью кнопки **Добавить** добавьте в таблицу три столбца, а затем настройте свойства **Text**, **TextAlign** и **Width** этих столбцов. Свойство **Text** задает название столбца, отображаемого в верх- ней части списка **ListView**. Первый столбец (с индексом 0) должен называться **Имя**, второй — **Размер**, а третий — **Изменен**. Свойство **TextAlign** задает выравнивание заголовка столбца. По умолчанию это свойство имеет значение **Left**, задающее выравнивание по левой границе. С помощью свойства **Width** можно указать начальную ширину столбца в пикселях. После отображения списка пользователь сможет изменять ширину столбцов при помощи мыши.
4. Запустите приложение на исполнение. Теперь в правом верхнем фрейме приложения появится пустой список с заголовками.



1. Элемент управления **ListView** может отображать содержимое своего окна в четырех различ- ных режимах, задаваемых при помощи свойства **View**. В зависимости от выбранного режима, эле- менты

списка могут снабжаться значками маленького или большого размера. В приложении **FramesApp** будутиспользованы шесть значков для отображения папок и файлов различных типов (файлы исполнимых программ, файлов справочной системы, текстовых файлов, файлов документов и пр.). Скопируйте файлы значков в папку **ImageList** созданную ранее.

1. Для работы с этими значками потребуется еще один список изображений **ImageList**. Перета- щите его значок с панели элементов в окно дизайнера форм. Новый список будет иметь идентификатор **imageList2**. Отредактируйте данный список.
2. В нашем приложении список **ListView** отображается только в одном режиме, а именно, как детализированная таблица. Такая таблица снабжается значками только маленького размера. Если же Ваше приложение будет использовать режим **LargeIcon**, необходимо подготовить дополнительный список **ImageList**, добавив в него файлы значков большого размера. Подключите подготовленный спи- сок к элементу управления **ListView**. Для этого присвойте свойствам **SmallImageList**, **LargeImageList** значение **imageList2**.
3. В общем случае список **ListView** содержит элементы, каждый из которых имеет дополнитель- ные элементы более низкого уровня, которые называются атрибутами. Сам элемент представляется текстовой строкой в первом столбце списка, отображаемого в виде детализированной таблицы. Значе- ния остальных атрибутов отображаются в остальных столбцах таблицы. Теперь необходимо наполнить список **ListView** функциональностью. Когда пользователь выделяет диск или каталог в дереве про- смотра **TreeView**, создается событие **AfterSelect**. Обработчик этого события должен определить, какой диск или какой каталог был выделен, а затем наполнить окно списка **ListView** именами каталогов и файлов, расположенных на этом диске или в этом каталоге. Создайте обработчиксобытия **AfterSelect**. Предварительно создайте поле **fullPath** в классе **Form1**, вставив следующую строку кода: public string fullPath;

Код обработчика события скопируйте из файла **AfterSelect.txt**.

Прежде всего, данный метод извлекает ссылку на узел дерева, выделенный пользователем, из свойства **Node** параметра обработчика событий **treeView1\_OnAfterSelect**. Далее, полный путь к выделенному узлу записывается в поле **fullPath** класса **string**.

Так как наше дерево содержит только обозначения дисков и каталогов, то это будет путь либо к корневомукаталогу диска, либо к одному из подкаталогов. Далее мы создаем объект класса **DirectoryInfo** и получаем списки всех файлов и каталогов, располагающихся в каталоге, вы- деленном пользователем в дереве. Для выполнения этих операций применяются методы **GetFiles** и **GetDirectories**. Перечень файлов обработчик события сохраняет в массиве **fiArray**, а перечень каталогов — в массиве **diArray**.

Наполнение списка именами каталогов выполняется в цикле **foreach**. С помощью конструк- тора класса **ListViewItem** мы создаем элемент списка, а затем задаем значения атрибутов этого элемента. Длина каталогов считается равной нулю, а время последнего изменения каталога извлекается при помощи метода **LastWriteTime** и преобразуется в текстовую строку методом **ToString**.

В свойство **lvi.ImageIndex** записывается нулевое значение — индекс значка в списке изображений

**imageList2** с изображением закрытой папки.

После заполнения всех атрибутов элемента списка этот элемент добавляется в список методом

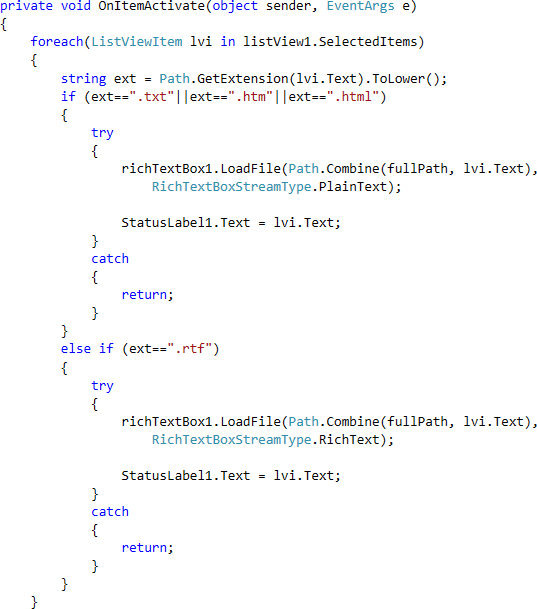
## listView1.Items.Add.

На следующем этапе в список добавляются имена файлов, расположенных в выделенном каталоге. Этиимена тоже добавляются в цикле

Размер очередного файла мы получаем с помощью свойства **Length**, а дату последнего изме- нения — с помощью свойства **LastWriteTime** (как и для каталогов).

Расширение имени файла извлекается из полного имени методом **Path.GetExtension**, а затем все его буквыпреобразуется в прописные методом **ToLower**. Непосредственный выбор значка выполняется при помощи оператора **switch**. Подготовленный элемент списка, описывающий текущий файл, добавляетсяв список методом **listView1.Items.Add.**

1. Протестируйте работу приложения. Приложение может отображать в левом фрейме дерево каталогов, а в правом верхнем фрейме — содержимое выбранных каталогов.
2. Разработайте обработчик события **ItemActivate** к элементу управления **ListView**, для того чтобы быломожно отображать в окне текстового редактора содержимое текстовых файлов, отображае- мых в правомверхнем фрейме. Это событие возникает, когда пользователь дважды щелкает строку в окне списка **ListView**. Для того, чтобы в строке состояния отображалась информация об имени файла добавьте на строку состояния **StatusLabel**. Исходный текст обработчика события **ItemActivate** имеет следующий вид:



В качестве первого параметра методу передается полный путь к файлу, полученный комбини- рованием полного пути каталога **fullPath**, выделенного в дереве, и имени файла **lvi.Text**, вы- деленного в списке **ListView**.

Через второй параметр методу **LoadFile** передается тип загружаемого документа, заданный как **RichTextBoxStreamType.PlainText** (т.е. текстовый документ). Дополнительно имя документа отобра- жается в строке состояния главного окна приложения, для чего это имя переписывается из свойства **lvi.Text** в свойство **statusLabel1.Text**.

1. Протестируйте работу приложения. Сохраните изменения.

**Практическое занятие №23.** Программирование приложений

**Цель работы:** научиться создавать проекты с использованием класса.

**Оборудование, технические и программные средства:** персональный компьютер, программа браузер.

