Департамент образования Вологодской области
бюджетное профессиональное образовательное учреждение
Вологодской области
«ВОЛОГОДСКИЙ СТРОИТЕЛЬНЫЙ КОЛЛЕДЖ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к практическим работам

ПО ДИСЦИПЛИНЕ

ОП.06. ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

2017г.

Рассмотрено на заседании предметной цикловой комиссии общепрофессиональных, специальных дисциплин и дипломного проектирования по специальностям 08.02.01 Строительство и эксплуатация зданий и сооружений, 08.02.07 Монтаж и эксплуатация внутренних сантехнических устройств, кондиционирования воздуха и вентиляции, 43.02.08 Сервис домашнего и коммунального хозяйства.

Данные методические указания предназначены для студентов специальности 09.02.04 Информационные системы (по отраслям) БПОУ ВО «Вологодский строительный колледж» при выполнении практических работ по дисциплине ОП.06. Основы алгоритмизации и программирования.

Настоящие методические указания включают в себя краткий теоретический материал, практические задачи, указания к их выполнению.

Методические указания могут быть рекомендованы к использованию студентами специальности **09.02.04 Информационные системы (по отраслям).**

Авторы:

Попова И.В. - преподаватель

Содержание

Практическая работа №30. Добавление в программы художественного оформления и специальных эффектов
Практическая работа №31. Использование событий и методов мыши.
Тема 3.4. Элементы разработки классов
Практическая работа №32. Объявление класса
Практическая работа №33.Создание экземпляров класса.
Практическая работа №34. Создание проекта с использованием класса
Практическая работа №35. Создание проекта с использованием класса.

Введение

Практическое занятие – это одна из форм систематических учебных занятий, на которых обучающиеся приобретают необходимые умения и навыки по определенному разделу дисциплины, входящей в состав учебного плана.

Общие цели практического занятия сводятся к закреплению теоретических знаний, более глубокому освоению уже имеющихся у обучающихся умений и навыков и приобретению новых умений и навыков, необходимых им для осуществления своей профессиональной деятельности и составляющих квалификационные требования к специалисту исходя из специальности и занимаемой им должности.

Основными задачами практических занятий являются:

- ✓ углубление теоретической и практической подготовки обучающихся;
- ✓ отражение в учебном процессе требований научно-технического прогресса, современных достижений науки и техники;
- ✓ развитие инициативы и самостоятельности обучающихся во время выполнения ими практических занятий.

Дидактические цели практических занятий сводятся:

- ✓ к закреплению теоретических знаний;
- ✓ овладению умениями и навыками, предусмотренными целями конкретной темы или раздела программы;
- ✓ изучению на практике методов исследований, обучению приемам исследовательской работы в условиях аудитории;
- ✓ развитие инициативы и самостоятельности в работе, оценке научно-теоретических положений.

Практические занятия могут быть:

- ✓ работами по образцу, цель которых решение типовых задач в строгом соответствии с содержанием методических указаний и рекомендациями преподавателя;
- ✓ реконструктивными работами, при которых обучающийся получает методические указания лишь в схематическом виде, а для выполнения работы требуется элемент творчества на основе ранее приобретенных умений и навыков;
- ✓ вариантными работами, в ходе которых изыскиваются новые варианты выполнения работы, не предусмотренные указаниями преподавателя и методическими рекомендациями;
- ✓ творческими работами, цель которых поиск решения новой задачи.

Объем практических занятий в часах и по тематике определяется учебным планом.

ТЕМАТИЧЕСКИЙ ПЛАН ПРОВЕДЕНИЯ ПРАКТИЧЕСКИХ РАБОТ

Цели проведения практических занятий

Цели и задачи выполнения практических заданий по дисциплине – требования к результатам освоения дисциплины.

В результате выполнения практических работ по дисциплине ОП. 06. Основы алгоритмизации и программирования обучающийся должен

уметь:

- использовать языки программирования, строить логически правильные и эффективные программы.

знать:

- общие принципы построения алгоритмов, основные алгоритмические конструкции;
- понятие системы программирования;
- основные элементы процедурного языка программирования, структуру программы, операции, управляющие структуры, структуры данных, файлы, кассы памяти;
- подпрограммы, составление библиотек подпрограмм;
- объектно-ориентированную модель программирования, понятие классов и объектов, их свойств и методов.

обладать общими компетенциями:

ОК 1. Понимать сущность и социальную значимость своей будущей профессии, проявлять к ней устойчивый интерес.

ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.

ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность

ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.

ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.

ОК 6. Работать в коллективе и в команде, эффективно общаться с коллегами, руководством, потребителями.

ОК 7. Брать на себя ответственность за работу членов команды (подчиненных), результат выполнения заданий.

ОК 8. Самостоятельно определять задачи профессионального и личностного развития, заниматься самообразованием, осознанно планировать повышение квалификации.

ОК 9. Ориентироваться в условиях частой смены технологий и профессиональной деятельности

обладать профессиональными компетенциями:

ПК 1.2 Взаимодействовать со специалистами смежного профиля при разработке методов, средств и технологий применения объектов профессиональной деятельности.

ПК 1.3 Производить модификацию отдельных модулей информационной системы в соответствии с рабочим заданием, документировать произведенные изменения.

ПК 2.2 Программировать в соответствии с требованиями технического задания

ПК 2.3 Применять методики тестирования разрабатываемых приложений

ТЕМАТИЧЕСКИЙ ПЛАН ПРАКТИЧЕСКИХ ЗАНЯТИЙ

№ п/п	Практические занятия	Объем часов
1	Составление блок-схем алгоритмов	2
2	Состав среды программирования. Состав окна, меню программы. Ввод текста программы в окне редактора	2
3	Составление программ линейной структуры	2
4	Составление программ разветвляющейся структуры.	2
5	Составление программ усложненной разветвляющейся структуры.	2
6	Составление программ циклической структуры.	2
7	Составление программ усложненной циклической структуры.	2
8	Обработка одномерных массивов.	2
9	Операции с элементами массивов, обмен элементами.	2
10	Обработка двумерных массивов.	2
11	Работа со строковыми переменными.	2
12	Использование стандартных функций и процедур для работы со строками.	2
13	Работа с данными типа множество.	2
14	Работа с файлом последовательного доступа.	2
15	Работа с файлом произвольного доступа.	2
16	Организация процедур и функций.	2
17	Программирование модуля.	2
18	Использование библиотеки подпрограмм.	2
19	Изучение интегрированной среды разработчика. Создание простого проекта.	2
20	Создание проекта с использованием кнопочных компонентов.	2
21	Создание проекта с использованием компонентов для работы с текстом.	2
22	Создание проекта с использованием компонентов ввода и отображения чисел, дат и времени.	2
23	Создание проекта с использованием группы зависимых переключателей.	2
24	Создание проекта с использованием полос прокрутки для ввода информации.	2
25	Создание проекта с использованием компонентов стандартных диалогов и системы меню.	2
26	Разработка оконного приложения.	2
27	Разработка оконного приложения с несколькими экранами.	2
28	Разработка многооконного приложения.	2
29	Разработка многооконного приложения.	2
30	Добавление в программы художественного оформления и специальных эффектов	2
31	Использование событий и методов мыши.	2
32	Объявление класса	2
33	Создание экземпляров класса.	2
34	Создание проекта с использованием класса	2
35	Создание проекта с использованием класса.	2
	И Т О Г О :	**70**

Практическая работа №1.
Составление блок-схем алгоритмов.

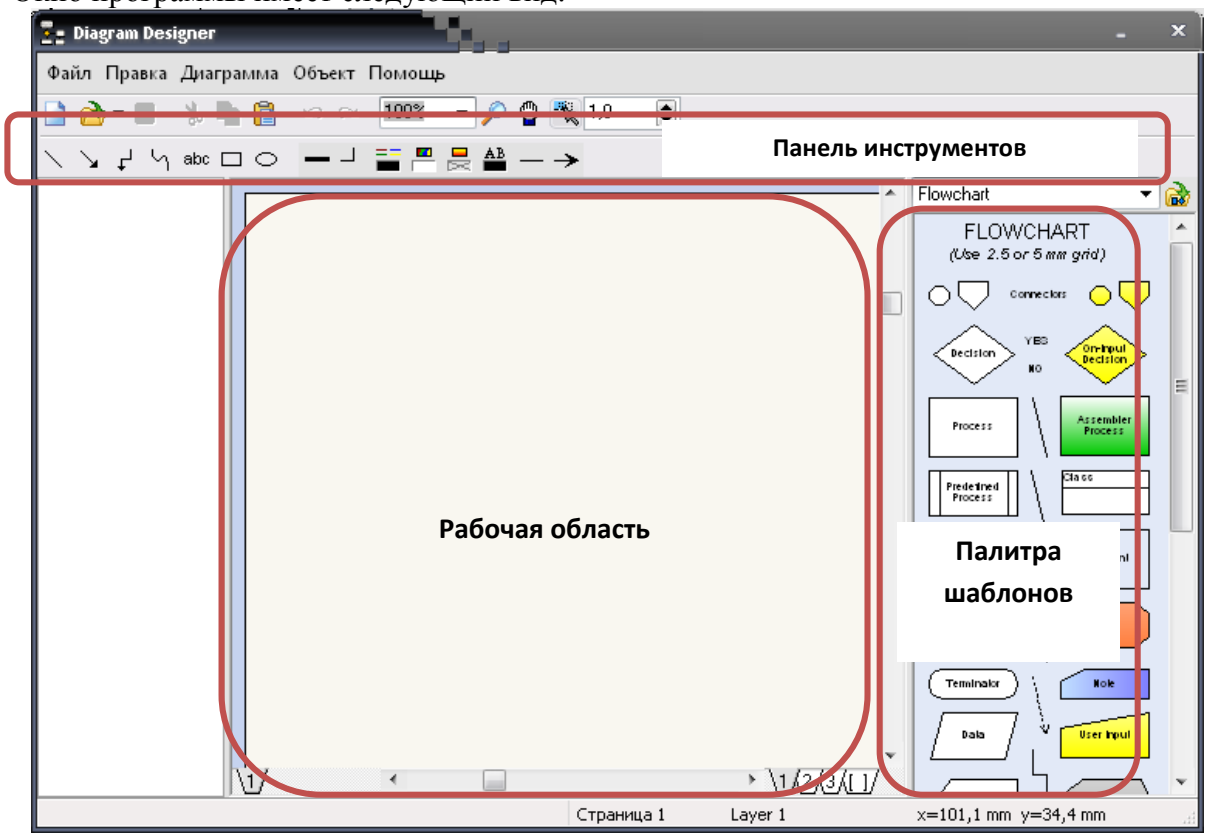
Цель работы:сформировать умения по составлению алгоритма программы; научить записывать его с помощью блок-схем.

Оборудование, технические и программные средства: персональный компьютер, приложение **DiagramDesigner**.

Задание 1. Создание простейшей блок-схемы

Методические указания по выполнению задания:

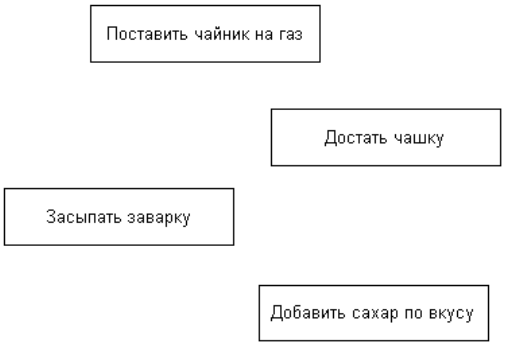
1. Запустите приложение **DiagramDesigner**. **DiagramDesigner** является приложением для создания блок-схем. В целом приложение способно соединять любые «блоки» поэтому в качестве блоков может выступать даже элементы электрической принципиальной схемы. Элементы-блоки можно составлять пользователю. Так же заявлены возможность построения графиков, слайд-шоу и встроенный калькулятор.
2. Окно программы имеет следующий вид:



3. Выберем шаблон, который был специально создан для данной практической работы, для этого скопируйте файл **simple** в папку своей группы. Далее в окне программы щелкните правой кнопкой мыши в палитре шаблонов и в контекстном меню выполните команду **Загрузите палитру шаблонов** и выберите файл **simple.**
4. Разместите в рабочей области прямоугольный блок, являющийся блоком **Действия**. Для этого нажмите левую кнопку мыши на блоке **Действия** и перетащите на рабочую область и отпустите левую кнопку мыши. Теперь вы умеете добавлять блоки на блок-схему – в основном вы будете добавлять блоки перемещением с палитры шаблонов и расположении их в рабочей области.
5. Далее необходимо уточнить, что же за действие выполняется, ведь иначе блок не имеет смысла, для этого необходимо выполнить двойной щелчок левой кнопкой мыши на блоке, в

открывшемся окне **Редактировать текст** введите текст **Поставить чайник на газ** и нажмите кнопку **Хорошо**. В результате текст внутри блока изменится на то, что мы ввели в окне редактора.

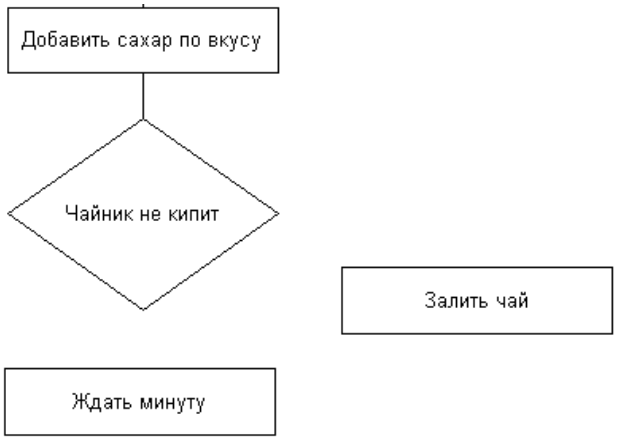
6. Добавьте блок действия и измените его описание на **Достать чашку**. После, добавьте ещё один блок действия с описанием **Засыпать заварку**. И наконец, добавьте последний блок действия **Добавить сахар по вкусу**. Блок-схема примет следующий вид:



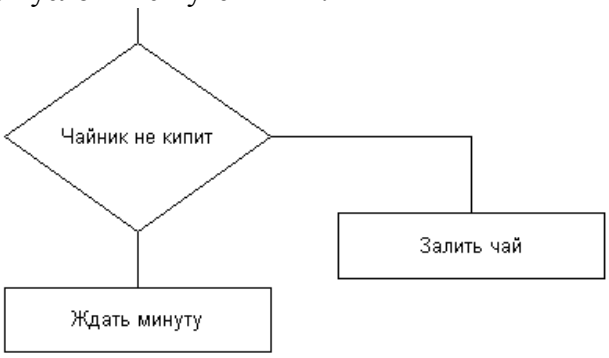
7. Блоки раскиданы в рабочей области произвольно, и это мало напоминает блок-схему. Конечно, можно расставить их более-менее в «столбец», но это тяжело и нудно. Поэтому необходимо выполнить соединение блоков. Для этого щелкните левой кнопкой мыши на элементе **Линия**  на панели инструментов. Нажмите левую кнопку мыши на центре нижней рамки блока **Поставить чайник на газ**, там изображено красное перекрестие. Далее наведите на центр верхней рамки блока **Достать чашку**, при наведении крестик меняет цвет с красного на зеленый, отпустите левую кнопку мыши. Теперь эти два блока соединены. Чтобы убедиться, что блоки были действительно соединены, выделите то пространство где хотите сделать такую проверку. В этой области все соединения будут подсвечены «зеленым», т.е. крестики будут вместо красных – зеленые. Так соединяются не только блоки действий, а все блоки на блок-схеме. Важно лишь, что можно соединять их только за те места (обозначены красными крестами) которые предусмотрены блоком.
8. Выполним выравнивание блоков на блок-схеме. Для этого выделите блок **Достать чашку** левой кнопкой мыши и не отпуская её передвигайте блок, пока стрелка соединяющая блоки не выровняется.
9. Соедините и выровняйте все оставшиеся блоки.

10. Добавьте на блок-схему блок нового типа – блок **Условия**. Для этого нажмите левую кнопку мыши на блоке **Условия** и перетащите его на рабочую область. Как видите, добавление блока **Условия** ничем не отличается от добавления блока Действия.

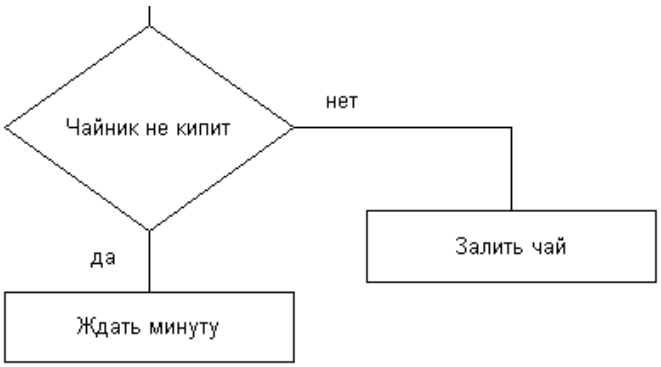
11. Соедините блок действия **Добавить сахар по вкусу** и новоиспеченный блок условия.
12. Измените описание блока условия со стандартного **Условный(ые) оператор(ы)** на **Чайник не кипит**.
13. Условие **Чайник не кипит** может быть ДА или НЕТ. Все, что внутри блока условия преобразуется кДА и НЕТ, чтобы получить разветвление пути программы. Принято путь выполнения ДА изображать под блоком условия, а путь выполнения НЕТ справа от него, но не на том же уровне, а ниже. Чтобы провести связи от блока условия создадим те блоки, что будут идти в пути «ДА» и в пути «НЕТ». Добавьте блоки действий **Ждать минуту** - под блоком условия, а блок **Залить чай** справа от него. Вот что должно получиться:



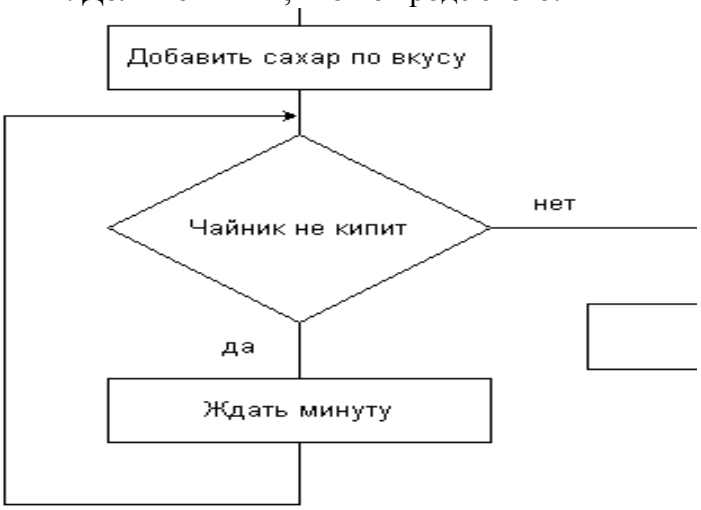
14. Для пути ДА легко будет соединить обычной линией, в то время как для соединения блока действия **Залить чай** с блоком условия **Чайник не кипит** необходимо либо две линии, либо коннектор. В блок-схемах не допускаются линии «наискосок», поэтому одной линией не обойтись. Коннектор есть на палитре шаблонов, и более того – для лучшего понимания он как раз и соединяет блок условия с путем НЕТ.



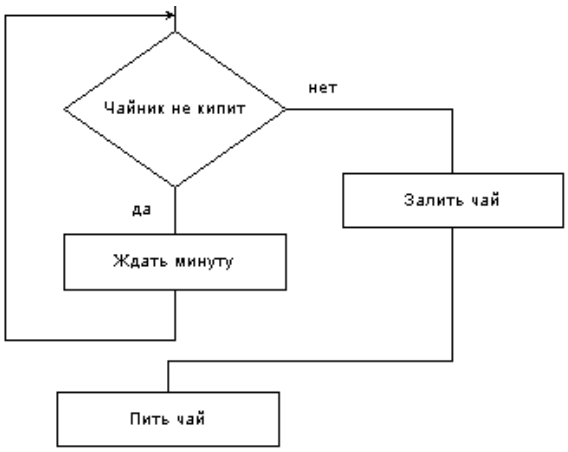
15. В результате мы имеем разветвление работы алгоритма (так же будет и с блок-схемами программ), но все же стоит указать где ДА и НЕТ тем более вначале это не совсем очевидно. Сделайте это так как показано на рисунке.



16. Теперь прекрасно видно, что когда происходит. Осталось только подумать что будет после действия **Ждать минуту** и после действия **Залить чай** и добавить это. После **Ждать минуту** очевидно, нужно снова проверить чайник. Т.е. мы возвращаемся в место перед блоком условия **Чайник не кипит**. Сделайте небольшую линию вниз от блока **Ждать минуту**, к ней присоедините ещё одну линию, идущую на несколько сантиметров влево, потом – линия вверх, на уровень середины линии между блоками **Добавить сахар по вкусу** и **Чайник не кипит**. Должно выйти, что-то вроде этого:



17. То, что мы сейчас организовали, в программировании называется цикл. На самом деле есть специальные блоки для циклов, но нагляднее изобразить их так. Важно же в блок-схеме, то, что мы можем создать принцип работы нашей программы, и лучше осознать как сделать её.
18. Не забывая и про блок **Залить чай** продолжим его путь. Для этого стрелочками образуем дорогу вниз, а потом влево, так чтобы оказаться снова на «осевой линии» блок-схемы. Таковы правила, они делают блок-схемы более наглядными. После этого добавим блок **Пить чай** и, соединив его с последней линией, закончим блок-схему:

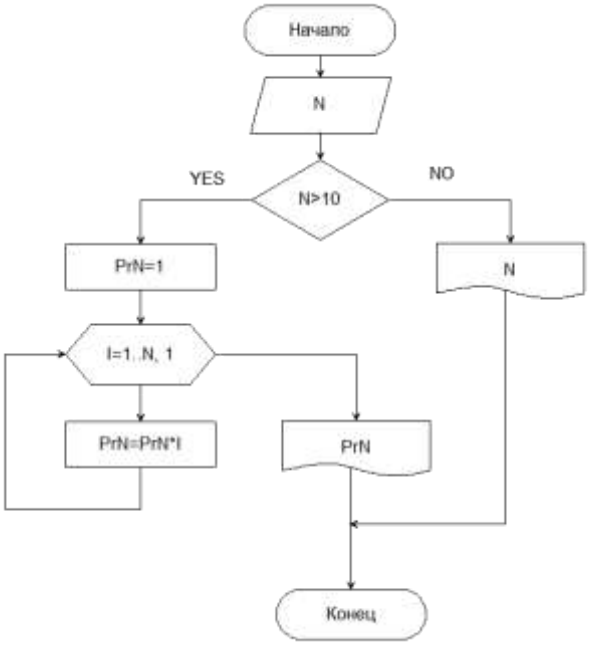


19. Сохраните блок-схему в папке своей группы под именем **схема1.ddd**

Задание 2. Создание блок-схемы алгоритма программы

Методические указания по выполнению задания:

1. Создайте блок-схему алгоритма программы, которая запрашивает с клавиатуры целое число N и если это число больше 10, то вычисляет и выводит на экран произведение всех целых чисел от 1 до N. Составим алгоритм решения данной задачи на естественном языке:
   - Ввод исходных данных N.
   - Проверка условия N>10.
   - В случае выполнения условия вычисляется произведение всех целых чисел от 1 до N.
   - Вывод на экран полученного произведения и завершение выполнения программы.
   - В противном случае вывод на экран значения N и завершения программы.
2. Запишите данный алгоритм на языке блок-схем. Для записи алгоритма воспользуйтесь программным средством **DiagramDesigner**.
3. Создайте новую блок-схему. Затем в левой части рабочего окна выберите соответствующий шаблон и нанесите его на рабочую область. Для задания надписей на блоках, выполните двойной щелчок по соответствующему блоку.
4. Оформите блок схему в соответствии с рисунком:



5. Сохраните построенную блок-схему в папке своей группы под именем **схема2.ddd**

Индивидуальные задания:

Каждый студент должен выполнить все задания.

- Напишите блок-схему программы, которая запрашивает у пользователя номер одного из весенних месяцев, и выводит количество дней в этом месяце. Программа должна проверять является ли введенный месяц весенним.
- Напишите блок-схему программы, которая запрашивает с клавиатуры число X. Если X меньше 10, то вычисляет и выводит на экран квадрат числа X, а если больше или равно 10, то вводит новое число Y, а затем вычисляет и выводит на экран значение суммы X и Y.

Практическая работа №2

«Состав среды программирования. Состав окна, меню программы. Ввод текста программы в окне редактора»

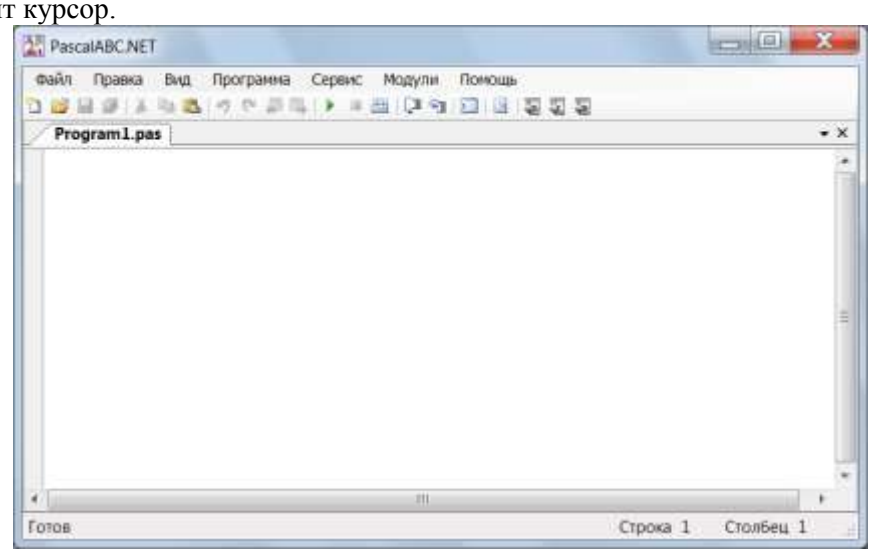
Цель работы: сформировать практические навыки работы с системой PascalABC.Net; научиться создавать, вводить в компьютер, выполнять и исправлять простейшие программы на языке PascalABC.Net.
Оборудование, технические и программные средства: персональный компьютер, система программирования **PascalABC.NET**.

Задание 1.

Познакомьтесь со способами загрузки среды программирования **PascalABC.Net**, её интерфейсом. Изучите назначение команд меню системы программирования **PascalABC.Net**.

Методические указания по выполнению задания:

1. Система **PascalABC** основана на языке **Delphi Pascal** и призвана осуществить постепенный переход от простейших программ к объектно-ориентированному программированию. Составление последовательности команд для решения конкретных задач на языке программирования называется разработкой программ, либо программированием. Для вызова среды программирования **PascalABC** необходимо запустить на выполнение файл **PascalABC.exe** или загрузить среду посредством ярлыка, если он существует на рабочем столе. Запустите систему программирования **PascalABC.Net**.
2. Рабочее окно **PascalABC** содержит уже знакомые нам элементы: это **Строка заголовка окна**, кнопки: **Свернуть, Развернуть, Закрыть**. Ниже находится **Строка меню**, затем **Панель инструментов**.
3. Под панелью инструментов находится **Вкладка**, т.е. та программа, которая сейчас открыта и **Рабочее окно программы**, т.е. окно в котором непосредственно будем набирать текст программы. По обе стороны от окна находятся **Полосы прокрутки**, которыми пользуются, если текст программы не вмешается в рабочее окно. Внизу экрана находится **Строка состояния**, показывающая на какой позиции стоит курсор.



4. Для того чтобы лучше ориентироваться в среде программирования **PascalABC**, разберем основные пункты меню. Первый пункт меню **Файл.** Как и других приложениях **Windows** мы видим пункт меню **Новый** (создаем новую программу), **Открыть** (открываем ранее сохраненную программу),

Сохранить (можем сохранить программу с расширением **pas**), **Сохранить все** (используется, если нужно сохранить несколько открытых программ), **Выход** (выйти из программы).

5. Следующий пункт меню **Правка**. Здесь находятся команды для работы с текстом программы. Можно отменить действие, восстановить действие, вырезать, копировать, вставить, найти, заменить, найти далее
6. Следующий пункт меню **Вид.** В этом пункте можно включить/выключить окна выполнения программы, окна отладки и др. Для этого нажимаем на соответствующую команду и видим, что появилось окно выполнения программы. Эти понятия для Вас являются новыми, и в процессе дальнейшего изучения **PascalABC** Вы познакомитесь с ними более подробно.
7. В пункте меню **Программа** можно начать выполнение программы. Обратите внимание на комбинации горячих клавиш. Запишите в тетрадях основные команды для выполнения и завершения программы:
   - выполнение программы: **Программа – Выполнить**, или **F9** или на Панели инструментов нажать  .
   - завершение выполнения программы: Программа – Завершить, или Ctrl+F2 или на Панели инструментов нажать  .
   - выполнение программы по шагам. Если допущена ошибка в программе или необходимо проверить часть программы, вы выполняете её по шагам, т.е. нажимаете **F7**, и каждое нажатие этой клавиши соответствует выполнению одной конкретной команды.
8. Следующий пункт **Сервис**. В программе **PascalABC** есть встроенные задачи, чтобы просмотреть их содержимое необходимо выбрать пункт **Просмотреть задание**. Выбираем тему, задание, нажимаем просмотр и по условию мы можем составить программу, а программа PascalABC проверит правильность выполнения задания.
9. В пункте меню **Помощь** находится встроенный электронный учебник.
10. Поочередно войдите в указанные ниже разделы главного меню и найдите следующие команды:
    - в меню Файл: Новый; Открыть; Сохранить; Сохранить как…; Выход;
    - в меню Правка: Отменить – отменить изменение; Восстановить – вернуть изменение;
    - В меню Программа: Выполнить – выполнить программу; Остановить – остановить программу.

Задание 2.

Изучите основные возможности работы в текстовом редакторе системы программирования **PascalABC**. Составьте простейшие программы.

Методические указания по выполнению задания:

1. Наберите простейшую программу, соответствующую следующему условию задачи. Требуется ввести с клавиатуры два целых числа, найти их сумму, результат вывести на экран с поясняющим текстом.

```
Program Raschet;                        //Название программы
Uses CRT;                               //Подключаемые модули
Var x, y, s:Integer;                    // объявление имен переменных и их типа
Begin                                   // начало исполнительной части
  Writeln('Введите два целых числа');   // написать на экране текст
  Readln(x,y);                          //  прочитать данные с клавиатуры и запомнить их в переменных
  s:=x+y;                               //  выполнить расчет и запомнить его в переменной
  Writeln('Сумма чисел =',s);           // написать на экраме текст и значение переменной
End.                                    // конец программы
```

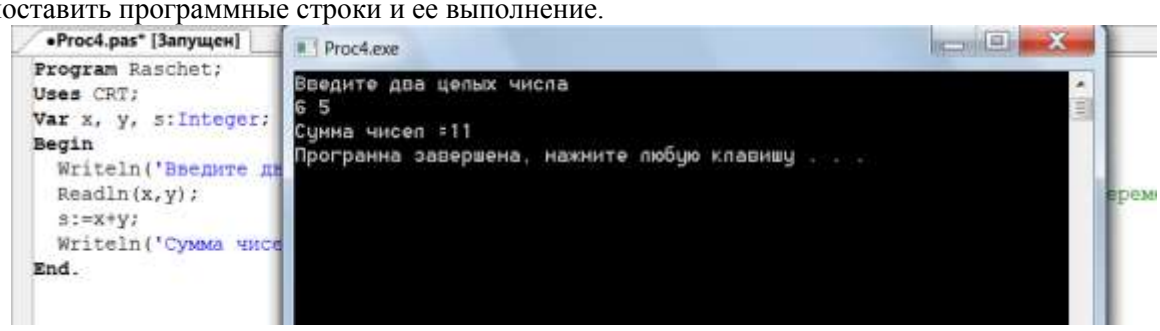
2. Просмотрите текст файла, обратите внимание на структуру программы. Структура простейших программ выглядит следующим образом:

```
Program    ...; заголовок программы и ее имя
  Var      ...; блок объявления переменных и их типа
    Begin    начало исполнительной части программы
     ...;      предложения, обеспечивающие
     ...;      выполнение
     ...;      программы
    End.     конец программы (точка обязательна)
```

3. Программа на **PascalABC** составляется из отдельных законченных элементов, называемых предложениями. В **PascalABC** текст программы обычно начинается особым предложением –

заголовком. Заголовок необязателен. В качестве имени программы можно применять комбинацию английских букв и цифр, следует писать в одно слово и нельзя применять служебные слова языка.

4. Каждое предложение языка должно отделяться от следующего за ним точкой с запятой (;). Исключение составляют комментарии. Они не отделяются точкой с запятой.
5. Обычно каждое предложение записывается с новой строки для наглядности и более легкого понимания текста. Для этих же целей используют отступы и выравнивания.
6. Комментарии предназначены для пояснения задачи и для временного исключения из текста программы некоторых операторов. В тексте они выделяются фигурными скобками { } или отделяются двумя косыми чертами //. Комментарии игнорируются компьютером при выполнении, однако при выводе текста программы – печатаются.
7. В **PascalABC** имеется особая группа слов, таких как, например: begin, for, end, program и другие, за которыми закреплены специальные смысловые значения. Такие слова называются служебными (зарезервированными) и должны употребляться в строгом соответствии с заложенным в них смыслом.
8. Существует и другая группа имен, имеющих стандартно определенный смысл, например, integer, writeln и другие. Их так и называют – стандартные или предопределенные имена.
9. Под именем программы располагается ее декларативная часть, здесь компьютеру сообщается обо всех именах констант и переменных, определяемых программистом, и о той роли, которую эти имена должны исполнять в программе.
10. За декларативной частью следует исполнительная часть программы, обрамляемая словами-ограничителями (логическими скобками): **Begin** и **End**. Между указанной парой слов и размещаются предложения, выполняющие в программе те или иные действия. Исполнительную часть программы называют телом программы.
11. Запустите набранную программу на выполнение (поскольку в программе выполняется подключение модуля **CRT**, то запускать программу нужно сочетанием клавиш **Shift+F9**). Если после запуска программы внизу окна появляется красная строчка с сообщением, то в строке, где находится курсор или в предыдущей находится ошибка. Внимательно просмотрите всю строчку, найдите и исправьте ошибку. После исправления всех ошибок и появления в новом окне начала работы программы, введите нужные данные (если в программе подразумевается ввод нескольких переменных, то это следует делать через **Enter** или пробел), получите результат работы и проверьте его на правильность. Так как текст программы и ее работа показываются в разных окнах (если подключен модуль **CRT**), можно сопоставить программные строки и ее выполнение.

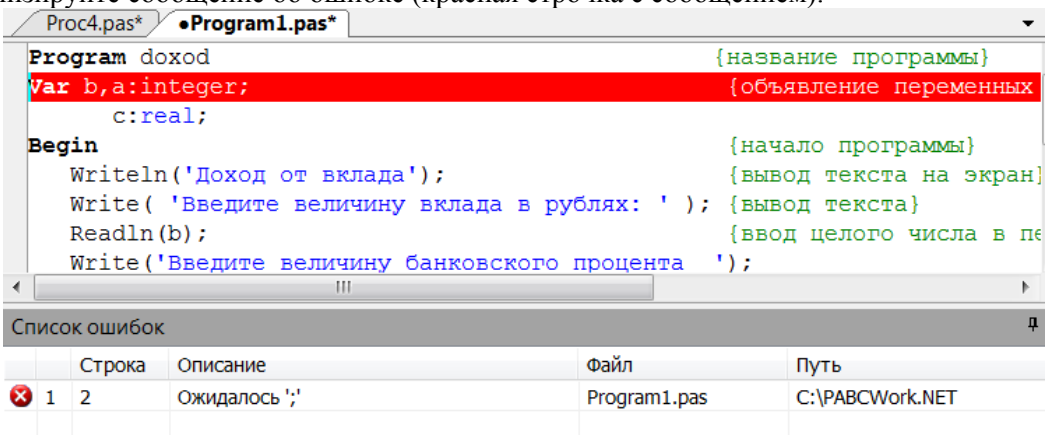


12. Создайте в папке своей группы папку с именем **Begin**. Сохраните набранную программу в папку **Begin** своей группы под именем **Begin1.pas**.
13. Разберитесь с работой программы и измените ее так, чтобы она вычисляла не сумму, а разность чисел. Проверьте правильность работы измененной программы. Сохраните программу под именем **Begin2.pas** в папке **Begin**.
14. Выполните команду **Файл – Новый**. Наберите текст программы. При наборе текста программы соблюдайте позиционирование (отступы) строк. Это не влияет на работу программы, но делает ее читабельной и облегчает поиск ошибок. В программе подсчитывается доход клиента за 1 год в зависимости от банковского процента и от величины денежного вклада.

Proc4.pas* [Запущен] | •Program1.pas*

```
Program doxod;                                  {название программы}
Var b,a:integer;                                {объявление переменных и их типа}
      c:real;
Begin                                           {начало программы}
   Writeln('Доход от вклада');                  {вывод текста на экран}
   Write( 'Введите величину вклада в рублях: ' ); {вывод текста}
   Readln(b);                                   {ввод целого числа в переменную b}
   Write('Введите величину банковского процента  ');
   Readln(a);
   c:=a*b/100;                                  {расчет значения переменной c}
   Writeln('Ваш доход =',c,' рублей');          {вывод текста, значения переменной и текста}
 end.
```

15. Запустите программу на выполнение. Введите следующие данные: введите величину вклада в рублях: 1000; введите величину банковского процента: 10. В результате должен получиться ответ: ваш доход = 100 рублей
16. Снова запустите программу и введите другие разумные исходные данные.
17. Вернитесь в текст, удалите символ «**;**» первой строке программы и запустите ее на исполнение. Проанализируйте сообщение об ошибке (красная строчка с сообщением).



18. Исправьте ошибку, затем удалите точку после последнего **End** в программе. Эта ошибка часто встречается у начинающих. Запустите программу и посмотрите, как реагирует **PascalABC** на подобную ошибку.
19. Удалите любую букву, например, в слове **Writeln**. Посмотрите, как реагирует **PascalABC** на подобную ошибку.
20. Удалите в блоке **Var** объявленную переменную и посмотрите, как отреагирует **PascalABC** на запуск программы с такой ошибкой. Запоминайте сообщения компьютера. Исправьте ошибки и сохраните программу под именем **Begin3.pas** в папке **Begin**.

Индивидуальные задания:

Составьте программу соответствующую следующей задаче:

Программа запрашивает имя пользователя и его возраст, по введенным данным определяет год рождения (текущий год запрашивается с клавиатуры), выводит его на экран и прощается с пользователем по имени.

Практическая работа №3

«Составление программ линейной структуры»

Цель работы: сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с линейной структурой.
Оборудование, технические и программные средства: персональный компьютер, система программирования **PascalABC.NET**.

Задание 1.
В системе программирования **PascalABC.NET** составьте программу по заданию. Протестируйте работу программы. Выполните задания на модификацию созданной программы.
Методические указания по выполнению задания:

1. Запустите систему программирования **PascalABC.NET**.
2. В окне редактора наберите текст программы, которая вычисляет сторону и периметр квадрата, если известна его площадь.

```
Program Primer1;
Var a,P,S:Real;
Begin
  Write('S='); Readln(S);
  a:=Sqrt(S);
  P:=4*a;
  Writeln('a=',a:5:2,'ед.');
  Writeln('P=',P:5:2,'ед.');
End.
```

В **PascalABC** используется несколько типов представления числовых значений, на начальном этапе будут рассмотрены лишь некоторые из них: **Integer** – целые числа в интервале от -2147483648 до 2147483647; **Real** – вещественные – целые и дробные положительные и отрицательные числа

Описания констант в декларативной части производится перед переменными, и предусматривают определенную форму записи чисел (дополнительно тип константы не оговаривается): если константа записана с точкой, тип константы считается Real. При записи значения константы используется знак равенства.

Переменная – это вид объектов в программе, предназначенный для хранения информации во время выполнения программы. По правилам **PascalABC** каждая переменная должна быть объявлена, т.е. описана в декларативной части программы. Переменная не имеет какого-либо конкретного значения до тех пор, пока компьютеру не будет дано точное предписание, поместить что-либо определенное в соответствующую ячейку памяти.

Описание переменных следует за описанием констант. В описании переменных после двоеточия указывается тип переменной/

В **PascalABC** возможны следующие действия (группы операций записаны в порядке приоритета): операция возведения в степень; умножение, деление, деление целочисленное, получение остатка от целочисленного деления; сложение, вычитание.

В пределах одной группы приоритета порядок выполнения операций, если нет скобок, определяется последовательностью записи.

3. Сохраните созданную программу в папке **Begin**, созданную ранее под именем **Zad1.pas**.
4. Выполните команду файл – Новый и наберите следующую программу. Разберитесь в её работе.

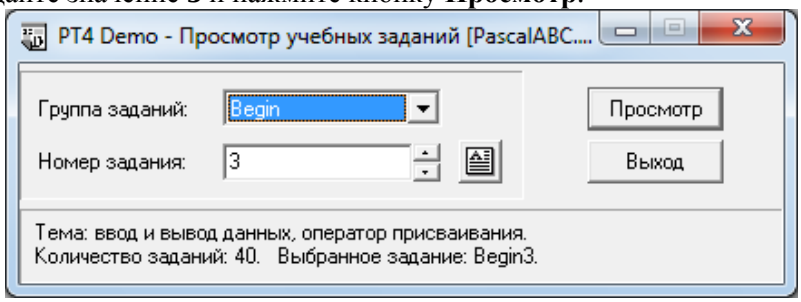
```
Program Primer2;
Uses crt;
   Var  a, s, d, e : Integer;
Begin
  Writeln('Сумма цифр трехзначного числа');
  Write('Введите целое трехзначное число  ');
  Readln(a);
  Clrscr;
  s:= trunc(a/100);
  d:= trunc((a-s*100)/10);
  e:=a-s*100-d*10;
  writeln('Сумма цифр трехзначного числа=', s+d+e);
End.
```

5. Определите с помощью каких стандартных операций вычисляется количество сотен и количество десятков. Запишите в тетрадь синтаксис, используемой для этого функции.
6. Измените программу так, чтобы для определения количества сотен и количества десятков использовались операции целочисленного деления и получения остатка от целочисленного деления.
7. Сохраните созданную программу в папке **Begin** под именем **Zad2.pas**.

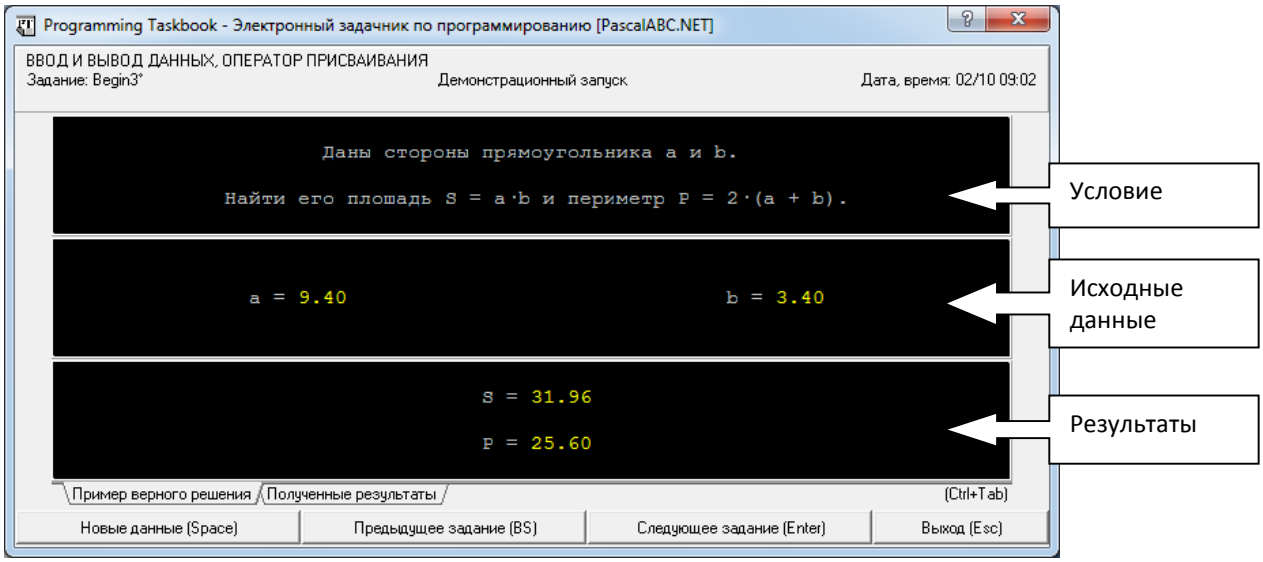
Задание 2. Используя встроенный задачник системы программирования **PascalABC.NET**, выполните решение задач **Begin3**, **Begin4, Begin5.**

Методические указания по выполнению задания:

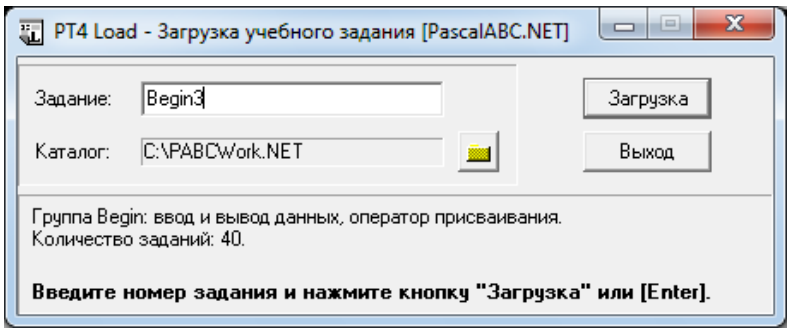
1. Откройте встроенный задачник системы программирования **PascalABC.NET**, для этого выполните команду **Модули – Просмотреть задания**. В диалоговом окне **Просмотр учебных заданий** в поле номер задания задайте значение **3** и нажмите кнопку **Просмотр**.



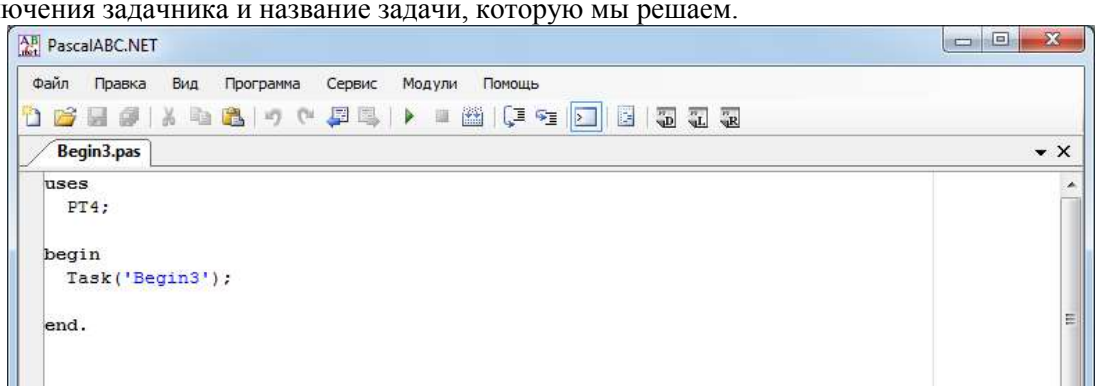
2. Окно просмотра текста задачи состоит из трех разделов. В первом записано условие задачи, во втором представлены значения исходных данных, а в третьем разделе окна указаны результаты решения задачи.



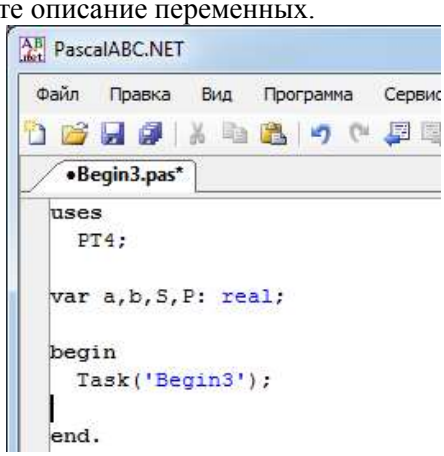
3. Решим данную задачу, используя систему программирования **PascalABC.NET**, для этого перейдите в систему программирования **PascalABC** и выполните команду **Модули – Создать шаблон программы**. В окне **Загрузка учебного задания** в поле **Задание** введите **begin3** и нажмите кнопку **Загрузка**.



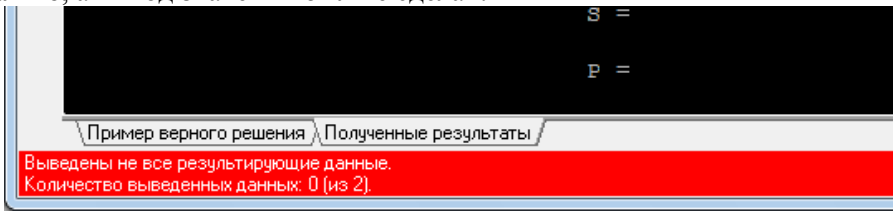
4. После этого в системе программирования **PascalABC** загрузится шаблон программы, где указан модуль подключения задачника и название задачи, которую мы решаем.



5. Просмотрите еще раз решение задачи, для этого выполните команду **Программы – выполнить** или нажмите соответствующую кнопку на панели инструментов.
6. При написании программы для решения задачи нам необходимо описать четыре переменные, две из них этого стороны прямоугольника **a, b**, именно их значения указаны в окне исходных данных, и переменные S и P, для результатов вычислений, их значения указаны в окне результатов. Описание переменных осуществляется в разделе **var**. Проанализировав значения исходных данных и полученных результатов в задачнике, можно сделать вывод, что переменные имеют тип **real**. Перейдите к окну шаблона программы и выполните описание переменных.

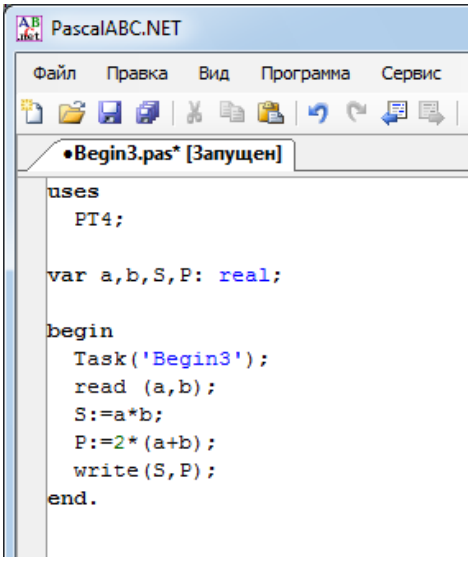


7. Далее перейдем к написанию программы. Нам необходимо ввести значения переменных **a, b**, для этого необходимо считать их значения с помощью оператора ввода **read**. Запустите программу на исполнение, программа выдаст сообщение, что **выведены не все результирующие данные**, т.е. ввод прошел нормально, а вывод значений был не сделан.



8. Для того чтобы выполнить вывод результирующих значений на экран нам необходимо выполнить расчет площади и периметра, формулы для вычисления представлены в условии задачи **S=ab, P=2(a+b)**. Запишите данные формулы, используя оператор присваивания.

9. Выполните запуск программы, изменился ли текст сообщения? Почему не изменился? Какой оператор необходимо добавить в текст программы?
10. Добавьте в тело программы оператор вывода, обратите внимание на последовательность переменных при выводе.



11. Запустите программу, изменился ли текст сообщения?
12. Выполните самостоятельно решение задач **Begin4, Begin5**.

Задание 3.

Составьте программы для решения следующих задач:

- Рассчитайте и выведите на экран количество рабочих часов в месяце, если продолжительность рабочего дня равна 8 часам в день, а число рабочих дней в месяце запрашивается у пользователя вашей программы.
- Скорость передачи данных по локальной сети запрашивается у пользователя и измеряется в битах в секунду. Ученик качал игру Т минут (время запрашивается у пользователя). Рассчитайте и выведите на экран размер файла (в Гбайтах), который скачал ученик и сколько денег придётся заплатить ему за трафик, если первый Гбайт не оплачивается, а всё то, что сверху - по у рублей за Гбайт (запрашивается у пользователя).
- Жёсткий диск имеет объём свободного пространства Х Гбайт – запрашиваемая величина. Сколько книг, каждая из которых состоит из 350 страниц, на каждой странице по 35 строк, в каждой строке по 55 символов, можно записать на жёсткий диск, если для хранения кода одного символа отводится 2 байта?

Задание 4.

Проверьте файлы, созданные в процессе выполнения практической работы. Заархивируйте папку с помощью программы архиватора, установленной на ваш компьютер. Передайте полученный архив преподавателю на проверку, разместив его в общих папках.

Практическая работа №4
«Составление программ разветвляющейся структуры»

Цель работы: сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с разветвляющейся структурой.
Оборудование, технические и программные средства: персональный компьютер, система программирования **PascalABC.NET**.

Задание 1.

В системе программирования **PascalABC.Net** составьте программу по заданию. Протестируйте работу программы. Выполните задания на модификацию созданной программы.

Методические указания по выполнению задания:

1. Запустите систему программирования **PascalABC.Net**.
2. В папке своей группы на диске создайте папку **IF**, в дальнейшем все программы необходимо сохранять в эту папку.

3. В окне редактора наберите текст программы, котораявычисляет квадратный корень из числа. Сохраните файл под именем **Zad1.pas**.

•Zad1.pas

```
program IF1;
  var a:real;
begin
  writeln ('Введите значение числа A');
  readln(a);
  if a>=0 then
          writeln('Квадратный корень числа  ',sqrt(a))
     else
          writeln('Корень из отрицательного числа не извлекается');
  writeln('Программа завершена!');
end.
```

4. Проверьте правильность работы программы, задав в качестве исходных данных **A** следующие значения: **655536; -256; 125,44**.
5. Поставьте «**;**» после оператора **writeln('Квадратный корень числа',sqrt(a))**в строке **7**. Запустите программу на исполнение. Появилось сообщение об ошибке. Почему это произошло? Исправьте ошибку.
6. Внесите изменения в программу в строки **9-10**, добавив составной оператор **begin-end**.

```
          begin writeln('Корень из отрицательного числа не извлекается');
writeln('Программа завершена!'); end;
```

7. Используя программу **DiagramDesigner,** начертите блок-схему, которая соответствует полученной программе. Сохраните файл с блок-схемой в папке **IF** под именем **Zad1.ddd**.
8. Исправьте программу так, чтобы в случае равенства числа нулю программа выводила на экран монитора сообщение **«Значение квадратного корня равняется нулю»**. Сохраните программу.

Задание 2.

Используя встроенный задачник системы программирования **PascalABC.NET**, выполните решение задач **If6**, **If8**.

Методические указания по выполнению задания:

8. Создайте новое окно, выполнив команду **Файл – Новый**.
9. Откройте встроенный задачник системы программирования **PascalABC.NET**, для этого выполните команду **Модули – Просмотреть задания**. В диалоговом окне **Просмотручебных заданий** в поле группа заданий выберите **If**, а в поле номер задания задайте необходимый номер задания и нажмите кнопку **Просмотр**.
10. Решите данную задачу, используя систему программирования **PascalABC.NET**, для этого перейдите в систему программирования **PascalABC** и выполните команду **Модули – Создать шаблон программы**. В окне **Загрузка учебного задания** в поле **Задание** введите **If6 (If8)** и нажмите кнопку **Загрузка**. После этого в системе программирования **PascalABC** загрузится шаблон программы, где указан модуль подключения задачника и название задачи, которую мы решаем.
11. Просмотрите еще раз решение задачи, для этого выполните команду **Программы – Выполнить** или нажмите соответствующую кнопку на панели инструментов.
12. В разделе описания переменных **var** выполните описание переменных в соответствии с условием задачи.
13. Составьте программу для решения задачи и выполните её тестирование. При необходимости исправьте ошибки. Чтобы задание считалось выполненным, запустите программу еще два раза.
9. Сохраните созданные программы в папке **IF**.

Задание 3.

Используя систему программирования **PascalABC.NET**, выполнить индивидуальные задания. Для каждой задачи построить блок-схему алгоритма решения, используя инструменты **DiagramDesigner.** Созданные программу и её блок-схему сохранить в папку **IF**, в названии файла укажите номер задачи.

Варианты заданий:

мер варианта	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
мера заданий	,25	,24	,23	,22	,21	,20	,19	,18	,17	0,16	1,15	2,14	3,1	4,2	5,3

Задания:

1. Даны три действительные числа. Возвести в квадрат те из них, значения которых неотрицательны, и в четвертую степень — отрицательные.
2. Даны две точки $A(x1, y1)$ и $B(x2, y2)$. Составить алгоритм, определяющий, которая из точек находится ближе к началу координат.
3. Даны два угла треугольника (в градусах). Определить, существует ли такой треугольник. Если да, то будет ли он прямоугольным.
4. Даны действительные числа x и y, не равные друг другу. Меньшее из этих двух чисел заменить половиной их суммы, а большее — их удвоенным произведением.
5. На плоскости XOY задана своими координатами точка A. Указать, где она расположена: на какой оси или в каком координатном угле.
6. Даны целые числа m, n. Если числа не равны, то заменить каждое из них одним и тем же числом, равным большему из исходных, а если равны, то заменить числа нулями.
7. Дано двухзначное число N. Проверить, будет ли сумма его цифр четным числом.
8. Дано двухзначное число N. Проверить, будет ли сумма больше 10.
9. Определить, является ли треугольник со сторонами a, b, c равносторонним.
10. Определить, является ли треугольник со сторонами a, b, c равнобедренным.
11. Определить, имеется ли среди чисел a, b, c хотя бы одна пара взаимно противоположных чисел.
12. Подсчитать количество отрицательных чисел среди чисел a, b, c.
13. Подсчитать количество положительных чисел среди чисел a, b, c.
14. Подсчитать количество целых чисел среди чисел a, b, c.
15. Дано целое число. Если оно является положительным, то прибавить к нему1; в противном случае не изменять его. Вывести полученное число.
16. Дано целое число. Если оно является положительным, то прибавить к нему1; в противном случае вычесть из него2. Вывести полученное число.
17. Дано целое число. Если оно является положительным, то прибавить к нему1; если отрицательным, то вычесть из него2; если нулевым, то заменить его на10. Вывести полученное число.
18. Даны два действительных числа. Вывести первое число, если оно больше второго, и оба числа, если это не так.
19. Даны два действительных числа.Заменить первое число нулем, если оно меньше или равно второму, и оставить числа без изменения в противном случае.
20. Даны три действительных числа. Выбрать из них те, которые принадлежат интервалу(1, 3).
21. Есть окружность с центром в начале координат и радиусом 5 единиц. Ввести с клавиатуры два числа, как координаты точки на той же плоскости, что и окружность. Вывести одно из сообщений: точка внутри окружности; точка на окружности; точка вне окружности.
22. Найти действительные корни квадратного уравнения $ax^2 + bx + c = 0$для любых вводимых значений коэффициентов. Рассмотреть ситуации:$a = 0$, $D < 0$ и обычное решение с двумя корнями.
23. Решить уравнение $Ax = B$. Предусмотреть случаи отсутствия решения и бесконечного множества решений.
24. Вычислить площадь треугольника по формуле Герона для любых вводимых длин сторон A, B и C: $p = \frac{a+b+c}{2}$; $S = \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}$. Предусмотреть проверку существования треугольника.
25. Даны три числа. Найти сумму двух наибольших из них.

Задание 4.

Проверьте файлы, которые были сохранены в папку **IF** в процессе выполнения практической работы. Заархивируйте папку с помощью программы архиватора, установленной на ваш компьютер. Передайте полученный архив преподавателю на проверку, разместив его в общих папках.

Практическая работа №5
«Составление программ усложненной разветвляющейся структуры»

Цель работы:сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с усложненной разветвляющейся структурой.
Оборудование, технические и программные средства: персональный компьютер, система программирования **PascalABC.NET**.

Задание 1.

В системе программирования **PascalABC.Net**составьте программу с усложненной разветвляющейся структурой по заданию. Протестируйтеработу программы. Выполните задания на модификацию созданной программы.

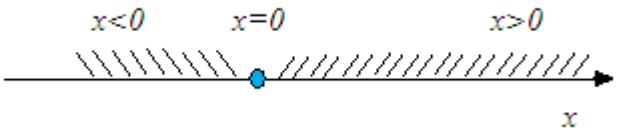
Методические указания по выполнению задания:

1. Запустите систему программирования **PascalABC.Net**.
2. В папке своей группы на диске создайте папку **IF2**, в дальнейшем все программы необходимо сохранять в эту папку.
3. В окне редактора наберите текст программы, которая вычисляет значение функции:

$$f(x)=\begin{cases} x-2, & x>0 \\ 5, & x=0 \\ x^2, & x<0 \end{cases}.$$

При решении задач часто приходится рассматривать не два, а большее количество вариантов. Это можно реализовать, используя несколько условных операторов. В этом случае после служебных слов **Then** и **Else** записывается новый условный оператор.

Для решения этой задачи рассмотрим координатную прямую, на которой отметим промежутки, на которые разбиваются все значения переменной *x*.



Начнем записывать условный оператор:

```
ли x>0
        то
        вычислить y по формуле y=x-12

        иначе
```

Что же должно выполниться в случае иначе? На эту ветку оператора попадают все не положительные значения *x*. Если бы для этих чисел нужно было бы выполнить один и тот же оператор (или группу операторов), то проблемы бы не стояло. Но нам нужно этот промежуток разделить еще на две части (отрицательные и ноль), и для части выполнить свой оператор. Поэтому ветка **Иначе** будет содержать еще один условный оператор и наш вложенный условный оператор будет иметь вид:

ли x>0

то

вычислить y по формуле y=x-12

иначе

если x=0

то

y вычислить по формуле y=5

иначе

y вычислить по формуле y=sqr(x);

Тогда фрагмент программы для решения этой задачи на языке программирования **PascalABC** будет выглядеть так:

```
If x>0
      Then
         y := x-12
      Else
          If x=0
      Then
          y := 5
      Else
          y := sqr(x);
```

4. Дополните фрагмент описанием необходимых переменных, операторами ввода-вывода. Сохраните программу в папку **IF2** под именем **Zad1.pas**. Протестируйте работу программы при различных значениях аргумента, так, чтобы можно было проверить каждую из ветвей.

Когда оператор **If** появляется внутри другого оператора **If**, они считаются вложенными. Такое вложение используется для уменьшения числа необходимых проверок. Этот метод часто обеспечивает большую эффективность, однако одновременно он уменьшает наглядность программы. Не рекомендуется использовать более одного-двух уровней вложения **If**. За вторым уровнем вложения становится трудно восстановить последовательность проверки условий каждым условным оператором.

5. Создайте новую вкладку. В окне редактора наберите текст программы, которая выполняет случайное предсказание одного из десяти вариантов ближайшего будущего с вероятностью 1/20, в остальных случаях программа выводит–«неудача».

Для написания данной программы воспользуемся оператором выбора. Данный оператор служит для выбора одного из помеченных вариантов действия (операторов), в зависимости от значения «параметра».

В программе будем использовать функцию **Random(x),** которая генерирует случайное число, с равномерной плотностью распределения на заданном интервале. Для инициализации распределения в начале программы необходимо вызвать процедуру **Randomize.**

Program2.pas*

```
Program FUTURE;
Var N : Word;
Begin
  Writeln('ПРЕДСКАЗАНИЕ БУДУЩЕГО');
  Randomize;
  N:=Random(20)+1;       { N - случайное число от 1 до 20 }
  Writeln;   write('Вас ожидает ');
  Case N of
     1 : writeln('счастье');
     2 : writeln('пятерка');
     3 : writeln('дорога');
     4 : writeln('двойка');
     5 : writeln('болезнь');
     6 : writeln('здоровье');
     7 : writeln('деньги');
     8 : writeln('любовь');
     9 : writeln('встреча');
     10 : writeln('дети')
      Else writeln('неудача')
  End;
  Writeln('Нажми Enter');
  Readln;
End.
```

6. Сохраните программу в папку **IF2** под именем **Zad2.pas**.

Задание 2.

Используя встроенный задачник системы программирования **PascalABC.NET**, выполните решение задач **If26**, **Case10**.

Методические указания по выполнению задания:

1. Создайте новое окно, выполнив команду **Файл – Новый**.
2. Откройте встроенный задачник системы программирования **PascalABC.NET**, для этого выполните команду **Модули – Просмотреть задания**. В диалоговом окне **Просмотручебных заданий** в поле группа заданий выберите **If**, а в поле номер задания задайте необходимый номер задания и нажмите кнопку **Просмотр**.
3. Решите данную задачу, используя систему программирования **PascalABC.NET**, для этого перейдите в систему программирования **PascalABC** и выполните команду **Модули – Создать шаблон программы**. В окне **Загрузка учебного задания** в поле **Задание** введите **If26 (Case10)** и нажмите кнопку **Загрузка**. После этого в системе программирования **PascalABC** загрузится шаблон программы, где указан модуль подключения задачника и название задачи, которую мы решаем.
4. Просмотрите еще раз решение задачи, для этого выполните команду **Программы – Выполнить** или нажмите соответствующую кнопку на панели инструментов.
5. В разделе описания переменных **var** выполните описание переменных в соответствии с условием задачи.
6. Составьте программу для решения задачи и выполните её тестирование. При необходимости исправьте ошибки. Чтобы задание считалось выполненным, запустите программу еще несколько раз.
7. Сохраните созданные программы в папке **IF2**.

Задание 3.

Используя систему программирования **PascalABC.NET**, выполнить индивидуальные задания. Для каждой задачи построить блок-схему алгоритма решения, используя инструменты **Diagram Designer.** Созданные программу и её блок-схему сохранить в папку **IF**, в названии файла укажите номер задачи.

Варианты заданий:

Номер варианта	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**

Номера заданий	,16	,17	,18	,19	,20	,21	,22	,23	,24	0,25	1,26	2,27	3,28	4,29	5,30

Задания:

1. Напишите программу, которая определяет, попадает ли точка *A* с координатами *(x,y)* внутрь круга радиуса *R*. Центр круга совпадает с началом координат.
2. Напишите программу «Угадай число». Компьютер «загадывает» число, а пользователь его отгадывает.
3. Напишите программу, которая находит сумму положительных чисел, больших 20, меньших 100 и кратных 3.
4. Составьте программу для упорядочения трёх чисел a, b, c по возрастанию таким образом, чтобы имени a соответствовало наименьшее число, имени b- среднее, имени c - наибольшее.
5. Составьте программу для упорядочения трех чисел a, b,c по возрастанию таким образом, чтобы имени a соответствовало наименьшее число, имени b - среднее, имени c - наибольшее.
6. Напишите программу, которая преобразовывает римские числа в натуральные числа.
7. Дано действительное число *a*. Вычислить *f(a)*, если

$$f(a)=\begin{cases} x^2, & -2 \le x \le 2 \\ 4, & \textit{в противном случае} \end{cases}$$

8. Дано действительное число *a*. Вычислить *f(a)*, если

$$f(a)=\begin{cases} x^2+4x+5, & x \le 2 \\ \dfrac{1}{x^2+4x+5}, & \textit{в противном случае} \end{cases}$$

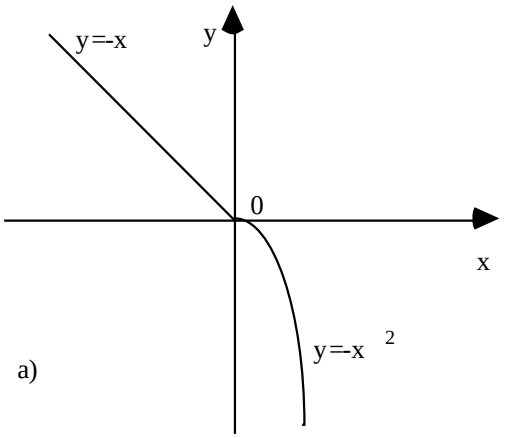
9. Дано действительное число *a*. Вычислить *f(a)*, если

$$f(a)=\begin{cases} 0, & x \le 0 \\ x, & 0 < x \le 1 \\ x^4, & \textit{в остальных случаях} \end{cases}$$

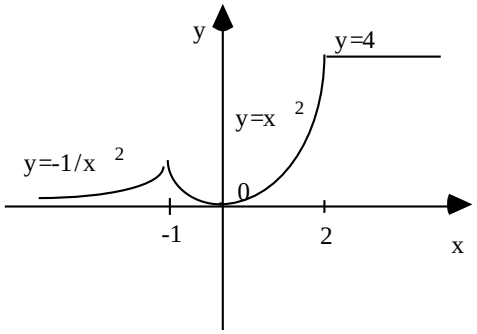
10. Дано действительное число *a*. Вычислить *f(a)*, если

$$f(a)=\begin{cases} 0, & x \le 0 \\ x^2-x, & 0 < x \le 1 \\ x^2-\sin x^2, & \textit{в остальных случаях} \end{cases}$$

11. Даны действительные положительные числа a, b, c, d. Выяснить, можно ли прямоугольник со сторонами a и b уместить внутри прямоугольника со сторонами c и d так, чтобы каждая из сторон одного прямоугольника была параллельна каждой стороне второго прямоугольника;
12. Даны действительные положительные числа a, b, c, x, y. Выяснить, пройдет ли кирпич с ребрами a, b, c в прямоугольное отверстие со сторонами x, y. Просовывать кирпич в отверстие разрешается только так, чтобы каждое из его ребер было параллельно или перпендикулярно каждой из сторон отверстия.
13. Дано действительное число *a*. Для функций *f(x)*, графики которых представлены на рисунках, вычислить *f(a)*



14. Дано действительное число *a*. Для функций *f(x)*, графики которых представлены на рисунках, вычислить *f(a)*



15. Даны целые числа a, b, c. Если a≤ b≤ c, то все числа заменить их квадратами, если a>b>c, то каждое число заменить наибольшим из них, в противном случае сменить знак каждого числа.
16. Составить программу, позволяющую получить словесное описание школьных отметок (1 — «плохо», 2 — «неудовлетворительно», 3 — «удовлетворительно», 4 — «хорошо», 5 — «отлично»).
17. Написать программу, классифицирующую треугольники (остроугольные, прямоугольные, тупоугольные), если даны углы.
18. Написать программу, которая по номеру дня недели - целому числу от 1 до 7 выдает в качестве результата количество пар в соответствующий день.
19. Написать программу нахождения числа дней в месяце, если даны: Номер месяца n - целое число a, равное 1 для високосного года и равное 0 в противном случае.
20. Написать программу, которая по номеру дня недели выводит его название.
21. В зависимости от того введена ли открытая скобка или закрытая, напечатать «открытая круглая скобка» или «закрытая фигурная скобка».
22. В зависимости от введённого символа L, S, V программа должна вычислять длину окружности; площадь круга; объём цилиндра.
23. Написать программу, которая по введенному числу от 0 до 15 выводит название цвета, соответствующего этому коду.
24. Определить, является ли введенная буква русского алфавита гласной.
25. Напишите программу, которая по введенному числу из промежутка 0..24, определяет время суток.
26. Напишите программу, которая по введенному номеру месяца високосного или невисокосного года, выводит количество дней в месяце.
27. Написать программу, которая по номеру месяца выдает название следующего за ним месяца (при т = 1 получаем февраль, 4 — май и т.д.).
28. Для каждой введенной цифры (0 — 9) вывести соответствующее ей название на английском языке (0 — zero, 1 — one, 2 — two,...).
29. Написать программу, позволяющую по последней цифре числа определить последнюю цифру его квадрата.
30. Написать программу, котораясообщает сдал студент экзамен или нет. Если оценка 3, 4, 5 - то экзамен сдан; если оценка 2, то не сдан.

Задание 4.

Проверьте файлы, которые были сохранены в папку **IF** в процессе выполнения практической работы. Заархивируйте папку с помощью программы архиватора, установленной на ваш компьютер. Передайте полученный архив преподавателю на проверку, разместив его в общих папках.

Практическая работа №6

«Составление программ циклической структуры»

Цель работы: сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с циклической структурой.
Оборудование, технические и программные средства: персональный компьютер, система программирования **PascalABC.NET**.

Задание 1.

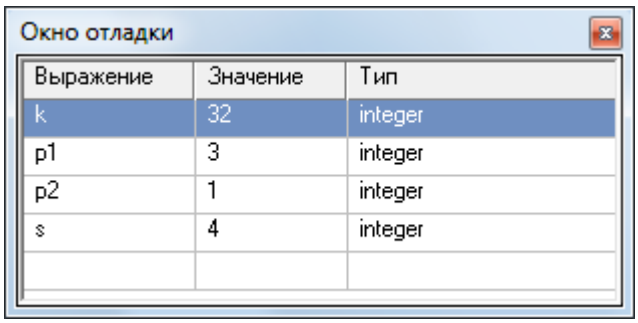
В системе программирования **PascalABC.Net** составьте программу по заданию. Протестируйте работу программы. Выполните задания на модификацию созданной программы.

Методические указания по выполнению задания:

1. Запустите систему программирования **PascalABC.Net**.
2. В папке своей группы на диске создайте папку **Циклы**, в дальнейшем все программы необходимо сохранять в эту папку.
3. В окне редактора наберите текст программы, которая из чисел от 10 до 99 выводит на экран те, сумма цифр которых равна n ($0 < n \leq 18$). Сохраните файл под именем **Zad1.pas**.

```
Program  PR_1;
Var   s, k, n, p1, p2: Integer;
Begin
        Writeln ('Введите целое число ');
        Readln (n);
        For k:= 10 to 99  do
                    Begin
                       P1:= k div 10;
                       P2:= k mod 10;
                       S:= p1+p2;
                       If s=n Then writeln (k);
                    End;
End.
```

4. Проанализируйте текст программы и ответьте на вопросы:
   - Каким образом выделили последнюю (младшую) цифру числа?
   - Каким действием в программе выделили первую (старшую) цифру числа?
5. Объясните обозначение переменных.
6. Рассмотрим выполнение программы в пошаговом режиме для $n = 2$. Режим пошагового выполнения предназначен для отладки программы. Для выполнения одного шага (одной строки) программы следует нажать клавишу **F8** (шаг без входа в подпрограмму), либо клавишу **F7** (шаг со входом в подпрограмму). Окно отладки позволяет просматривать во время пошагового исполнения программы значения переменных. По умолчанию оно располагается в правом верхнем углу окна редактора. Для добавления переменной или выражения в окно отладки следует нажать комбинацию клавиш **Ctrl-F5** или выполнить команду **Программа – Добавить переменную**. Добавьте в окно отладки переменные **k, p1, p2** и **s** и запустите программу в пошаговом режиме.



7. В результате работы программы на экране появится 2 числа: 11, 20.
8. В данной программе используется цикл со счетчиком, изменим программу так чтобы в программе использовался цикл с предусловием. Для этого необходимо внести изменения в строки программы **6-11**.

```
Readln (n);
k:=10;
while k<>99  do
            Begin
               P1:= k div 10;
               P2:= k mod 10;
               S:= p1+p2;
               If s=n Then writeln (k);
               k:=k+1;
            End;
```

9. Запустите программу на исполнение при $n = 2$. Проанализируйте полученные результаты.
10. Самостоятельно выполните изменение программы так, чтобы в ней использовался цикл с постусловием.

Задание 2.

Используя встроенный задачник системы программирования **PascalABC.NET**, выполните решение задач **While1, While4, While11, For5, For12**.

Методические указания по выполнению задания:

1. Создайте новое окно, выполнив команду **Файл – Новый**.
2. Откройте встроенный задачник системы программирования **PascalABC.NET**, для этого выполните команду **Модули – Просмотреть задания**. В диалоговом окне **Просмотр учебных заданий** в поле группа заданий выберите **While (For)**, а в поле номер задания задайте необходимый номер задания и нажмите кнопку **Просмотр**.
3. Решите данную задачу, используя систему программирования **PascalABC.NET**, для этого перейдите в систему программирования **PascalABC** и выполните команду **Модули – Создать шаблон программы**. В окне **Загрузка учебного задания** в поле **Задание** введите **While1 (While4, While11, For5, For12)** и нажмите кнопку **Загрузка**. После этого в системе программирования **PascalABC** загрузится шаблон программы, где указан модуль подключения задачника и название задачи, которую мы решаем.
4. Просмотрите еще раз решение задачи, для этого выполните команду **Программы – Выполнить** или нажмите соответствующую кнопку на панели инструментов.
5. В разделе описания переменных **var** выполните описание переменных в соответствии с условием задачи.
6. Составьте программу для решения задачи и выполните её тестирование. При необходимости исправьте ошибки. Чтобы задание считалось выполненным, запустите программу необходимое количество раз.
7. Сохраните созданные программы в папке **Циклы**.

Задание 3.

Используя систему программирования **PascalABC.NET**, выполнить индивидуальные задания. Для каждой задачи построить блок-схему алгоритма решения, используя инструменты **Diagram Designer.** Созданные программу и её блок-схему сохранить в папку **Циклы**, в названии файла указать номер задачи. Каждый студент должен выполнить все задания.

- С помощью оператора while напишите программу определения суммы всех нечетных чисел в диапазоне от 1 до 99 включительно.
- Найти нечетные и кратные трем числа в диапазоне от 30 до 60 включительно. Распечатать их в порядке убывания.

- В сберкассу на трехпроцентный вклад положили S рублей. Какой станет сумма вклада через N лет?

Задание 4.

Проверьте файлы, которые были сохранены в папку **Циклы** в процессе выполнения практической работы. Заархивируйте папку с помощью программы архиватора, установленной на ваш компьютер. Передайте полученный архив преподавателю на проверку, разместив его в общих папках.

Практическая работа №7

Составление программ усложненной циклической структуры

Цель работы: сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с усложненной циклической структурой.
Оборудование, технические и программные средства: персональный компьютер, среда программирования **Visual Studio**.

Задание 1.

В среде программирования **Visual Studio** составьте программу с вложенными циклами. Протестируйте работу программы. Выполните задания на модификацию созданных программ.

Методические указания по выполнению задания:

1. Запустите среду программирования **Visual Studio**.
2. В папке своей группы на диске создайте папку **Циклы2**, в дальнейшем все программы необходимо сохранять в эту папку.
3. Создайте новое консольное приложение с именем **Zad1**. В окне редактора наберите текст программы, которая позволит вывести на экран числа следующим образом:

2 2 2 2 2

2 2 2 2 2

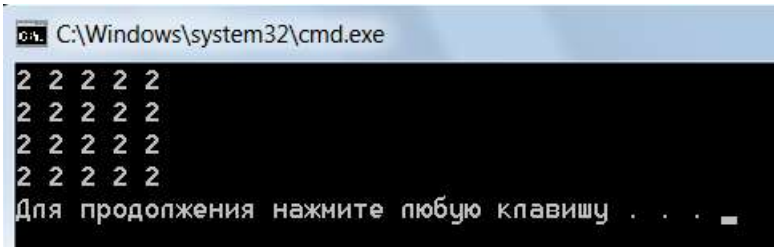
2 2 2 2 2

2 2 2 2 2

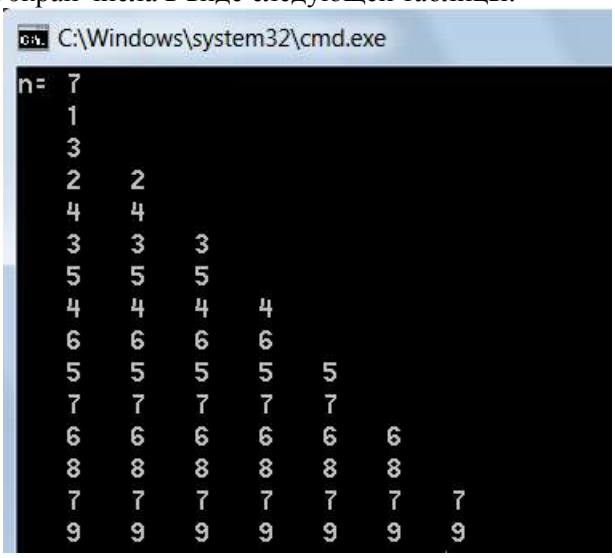
Циклы могут быть простые или вложенные (кратные, циклы в цикле). Вложенными могут быть циклы любых типов: **while**, **do while**, **for**. Каждый внутренний цикл должен быть полностью вложен во все внешние циклы. «Пересечения» циклов не допускаются.

```
static void Main()
{
  for (int i = 1; i <= 4; i++)
  {
    for (int j = 1; j <= 5; j++)
    {
      Console.Write("2 ");
    }
    Console.WriteLine();
  }
}
```

4. Проверьте правильность работы программы, задав в качестве исходных данных следующие значения:



5. Измените программу так, чтобы таблица содержала *n* строк и *m* столбцов (значения *n* и *m* вводятся с клавиатуры).
6. Создайте новое консольное приложение **Zad2**. В окне редактора текста напишите программу, которая выводит на экран числа в виде следующей таблицы:



7. Протестируйте работу программы. Сохраните программу.

Задание 2.

В среде программирования **Visual Studio** составьте программу с оператором цикла и условным оператором. Протестируйте работу программ. Выполните задания на модификацию созданных программ.

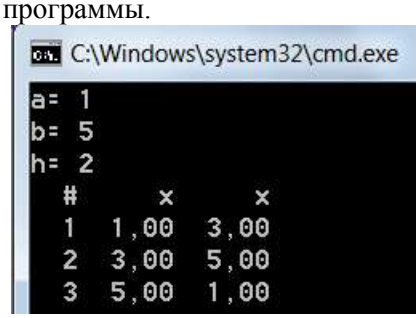
Методические указания по выполнению задания:

1. Создайте новое консольное приложение с именем **Zad3**. В окне редактора наберите текст программы, которая выводит на экран таблицу значений функции $y(x)$ для $x \in [a,b]$ с шагом h.

$$y(x)=\begin{cases}(x^3+1)^2, & \text{ïðè } x<0\\ 0, & \text{ïðè } 0\le x<1\\ |x^2-5x+1|, & \text{ïðè } x\ge 1\end{cases}$$

```
static void Main(string[] args)
{
    Console.Write("a= ");
    double a = double.Parse(Console.ReadLine());
    Console.Write("b= ");
    double b = double.Parse(Console.ReadLine());
    Console.Write("h= ");
    double h = double.Parse(Console.ReadLine());
    double y;
    int i = 1;
    Console.WriteLine("{0,3} {1,5} {1,5}", "#", "x", "f(x)");
    for (double x = a; x <= b; x += h, i++)
    {
        if (x < 0)
        {
            y = Math.Pow(Math.Pow(x, 3) + 1, 2);
        }
        else
        {
            if (x < 1)
            {
                y = 0;
            }
            else
            {
                y = Math.Abs(x * x - 5 * x + 1);
            }
        }
        Console.WriteLine("{0,3} {1,5:f2} {2,5:f2}", i, x, y);
    }
}
```

2. Проверьте правильность работы программы.



Задание 3.

В среде программирования **Visual Studio** составьте программу реализующую алгоритм расчета простейшей рекуррентной последовательности. Протестируйте работу программы. Выполните задания на модификацию созданных программ.

Методические указания по выполнению задания:

1. Создайте новое консольное приложение с именем **Zad4**. В окне редактора наберите текст программы, в которой вычисляются первые n членов арифметической прогрессии при условии, что $a_1 = \frac{1}{2}$ и $d = \frac{1}{4}$.

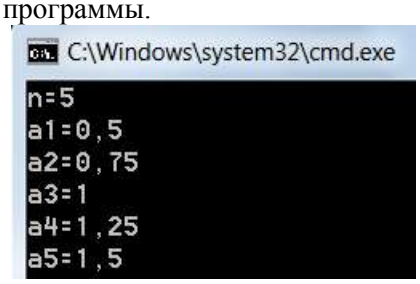
Пусть $a_1, a_2, ..., a_n$ – произвольная числовая последовательность. Рекуррентным соотношением называется такое соотношение между членами последовательности, в котором каждый следующий член выражается через несколько предыдущих, т.е.:

$a_k = f(a_{k-1}, a_{k-2}, ..., a_{k-l}),\ k > l$

Самым простым примером рекуррентной последовательности является арифметическая прогрессия. Рекуррентное соотношение для нее записывается в виде $a_k = a_{k-1} + d$, где d – разность прогрессии. Зная первый элемент и разность прогрессии, и, используя данное рекуррентное соотношение, можно последовательно вычислить все остальные члены прогрессии.

```
static void Main(string[] args)
{
    double a = 0.5;
    const double d = 0.25;
    Console.Write("n=");
    int n = int.Parse(Console.ReadLine());

    //вывели первый член последовательности
    Console.WriteLine("a1={0}", a);

    //организуем вычисление 2, 3, … ,n члена последовательности
    for (int i = 2; i <= n; i++)
    {
        //для этого прибавляем к предыдущему члену значение d
        a += d;
        //и выводим новое значение a на экран
        Console.WriteLine("a{0}={1}", i, a);
    }
}
```

2. Проверьте правильность работы программы.



Задание 4.

Используя среду программирования **Visual Studio**, выполнить индивидуальные задания. Созданную программу сохранить в папку **Циклы2**, в названии файла укажите номер задачи.

Варианты заданий:

Номер компьютера	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Номера заданий	,15	,14	,13	,12	,11	,10	7,9	8,8	9,7	10,6	11,5	12,4	3,3	4,2	5,1

Задание 1:

Вывести на экран числа следующим образом:

1.
1 1 1 1 1 1
2 2 2 2 2 2
3 3 3 3 3 3
4 4 4 4 4 4

2.
1 2 3 ... 10
1 2 3 ... 10
1 2 3 ... 10
1 2 3 ... 10

3.
-10 -9 -8 ... 12
-10 -9 -8 ... 12
-10 -9 -8 ... 12
-10 -9 -8 ... 12
-10 -9 -8 ... 12

4.
41 42 43 ... 50
51 52 53 ... 60
61 62 63 ... 70
...
71 72 73 ... 80

5.
5
5 5
5 5 5
5 5 5 5
5 5 5 5 5

6.
1 1 1 1 1
1 1 1 1
1 1 1
1 1
1

7.
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5

8.
6 6 6 6 6
7 7 7 7
8 8 8
9 9
10

9.
7
6 6
5 5 5
4 4 4 4
3 3 3 3 3

10.
8 8 8 8 8
7 7 7 7
6 6 6
5 5
4

11.
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

12.
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1

13.
0 1 2 3 4
0 1 2 3
0 1 2
0 1
0

14.
4 3 2 1 0
3 2 1 0
2 1 0
1 0
0

15.
1
0
2 2
0 0
3 3 3
0 0 0
4 4 4 4
0 0 0 0
5 5 5 5 5
0 0 0 0 0

Задание 2:

Постройте таблицу значений функции $y(x)$ для $x \in [a,b]$ с шагом h:

1. $y(x) = \begin{cases} \dfrac{1}{(0{,}1+x)^2}, \ при \ x \geq 0{,}9 \\ 0{,}2x + 0{,}1, \ при \ 0 \leq x < 0{,}9 \\ x^2 + 0{,}2, \ при \ x < 0 \end{cases}$

2. $y(x) = \begin{cases} 0, \ при \ x < a \\ \dfrac{x-a}{x+a}, \ при \ x > a \\ 1, \ при \ x = a \end{cases}$

3. $y(x) = \begin{cases} -4, \ при \ x < 0 \\ x^2 + 3x + 4, \ при \ 0 \leq x < 1 \\ 2, \ при \ x \geq 1 \end{cases}$

4. $y(x) = \begin{cases} (x^2 - 1)^2, \ при \ x < 1 \\ \dfrac{1}{(1+x)^2}, \ при \ x > 1 \\ 0, \ при \ x = 1 \end{cases}$

5. $y(x) = \begin{cases} x^2 + 5, \ при \ x \leq 5 \\ 0, \ при \ 5 < x < 20 \\ 1, \ при \ x \geq 20 \end{cases}$

6. $y(x) = \begin{cases} x, \ при \ x > 0 \\ 0, \ при \ -1 \leq x \leq 0 \\ x^2, \ при \ x < -1 \end{cases}$

7. $y(x) = \begin{cases} 0, \ при \ x < 5 \\ \dfrac{x-5}{x+5}, \ при \ x > 5 \\ 1, \ при \ x = 5 \end{cases}$

8. $y(x) = \begin{cases} (x^2 - b)^2, \ при \ x < b \\ \dfrac{1}{(b+x)^2}, \ при \ x > b \\ 0, \ при \ x = b \end{cases}$

9. $y(x)=\begin{cases}0, \ при \ x<0\\ x^2+1, \ при \ 0\le x<1\\ 1, \ при \ x=1\end{cases}$

10. $y(x)=\begin{cases}x^2-0{,}3, \ при \ x<3\\ 0, \ при \ 3\le x\le 5\\ x^2+1, \ при \ x>5\end{cases}$

11. $y(x)=\begin{cases}x^2-1, \ при \ x\le 1\\ 2x-1, \ при \ 1<x\le 2\\ x^5-1, \ при \ x>2\end{cases}$

12. $y(x)=\begin{cases}x^2, \ при \ x<1\\ \dfrac{1}{x^2-1}, \ при \ x>1\\ 0, \ при \ x=1\end{cases}$

13. $y(x)=\begin{cases}x^2, \ при \ x<1\\ \dfrac{1}{x+2}, \ при \ x>1\\ x+2, \ при \ x=1\end{cases}$

14. $y(x)=\begin{cases}x^3-1, \ при \ x<0{,}1\\ 2x-1, \ при \ x>0{,}1\\ 0, \ при \ x=0{,}1\end{cases}$

15. $y(x)=\begin{cases}x^3-0{,}1, \ при \ x\le 5\\ 0{,}2x-0{,}1, \ при \ 5<x<20\\ x^3+0{,}1, \ при \ x\ge 20\end{cases}$

Практическая работа №8
«Обработка одномерных массивов»

Цель работы: сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с использованием одномерных массивов.
Оборудование, технические и программные средства: персональный компьютер, система программирования **PascalABC.NET**.

Задание 1.

В системе программирования **PascalABC.Net** составьте программу по заданию. Протестируйте работу программы. Выполните задания на модификацию созданной программы.

Методические указания по выполнению задания:

1. Запустите систему программирования **PascalABC.Net**.
2. В папке своей группы на диске создайте папку **Array1**, в дальнейшем все программы необходимо сохранять в эту папку.
3. На языке программирования **PascalABC.Net** составьте программу, которая находит сумму 30 целых чисел. Сохраните файл под именем **Zad1.pas**.

Одномерный массив – это фиксированное количество элементов одного и того же типа, объединенных одним именем, где каждый элемент имеет свой номер. Обращение к элементам массива осуществляется с помощью указания имени массива и номеров элементов. Нам для работы требуется массив из 30 целых чисел.

Опишем в разделе описания типов свой тип – одномерный массив, состоящий из 30 целых чисел.

```
Type MyArray = Array [1..30] Of Integer;
```

Напомним, что раздел типов начинается со служебного слова **Type**, после этого идет имя нового типа и его описание. Между именем типа и его описанием ставится знак «равно». В нашем случае **MyArray** – имя нового типа данных; **Array** – служебное слово (в переводе с английского означает «массив», «набор»); **[1..30]** – в квадратных скобках указывается номер первого элемента, затем, после двух точек, номер последнего элемента массива; **Of** – служебное слово (в переводе с английского «из»); **Integer** – тип элементов массива.

Далее необходимо заполнить наш массив числами, для этого необходимо воспользоваться циклической конструкцией следующего вида:

```
WriteLn ('Введите ' ,n, ' чисел');
For i:=1 To n Do ReadLn(A[i]);
```

В данном случае числа вводятся пользователем с клавиатуры.

Далее необходимо воспользоваться циклом для определения суммы чисел:

```
s:=0;
For i:=1 To n Do s:=s+A[i];
WriteLn ('Их сумма равна ' ,s) ;
```

Тогда решение данной задачи с использованием массива имеет вид:

•Zad1.pas

```
Program Zad1;
Const n=30 ;
Type MyArray=Array[1..n] Of Integer;
Var
  A: MyArray;
  s,i: Integer;
Begin
  WriteLn ('Введите ' ,n, ' чисел');
  For i:=1 To n Do ReadLn(A[i]);
  s:=0;
  For i:=1 To n Do s:=s+A[i];
  WriteLn ('Их сумма равна ' ,s) ;
End.
```

4. Проверьте правильность работы программы.
5. Измените программу так, чтобы определялась сумма элементов массива, кратных заданному числу. Сохраните полученную программу в папке **Array1** под именем **Zad2.pas**.

Решение задачи изменится незначительным образом. Добавляется описание еще одной переменной для хранения значения числа, на кратность которому проверяются значения элементов массива.

Появляются операторы:

```
WriteLn ('Введите число'); ReadLn (k);
```

и изменяется оператор из тела цикла:

```
For i:=1 To n Do
If A[i] Mod k = 0 Then s:=s+A[i];
```

6. Измените программу так, чтобы определялось количество положительных и отрицательных элементов в данном массиве. Сохраните полученную программу в папке **Array1** под именем **Zad3.pas**.

Суть основного изменения программы заключается во введении двух переменных (счетчиков – **pos**, **neg**) для хранения значений количества положительных и отрицательных элементов в массиве.

```
Var
  A: MyArray;
  s,i,k: Integer;
  pos, neg: Integer;
```

Добавьте в программу следующие строки программного кода:

```
s:=0;
pos:=0;
neg:=0;
For i:=1 To n Do
  If A[i]>0 Then Inc(pos) Else
    If A[i]<0 Then Inc (neg);
WriteLn (pos:4, neg:4);
```

7. Измените программу так, чтобы в массив **B** записывались номера четных элементов массива **A**. Сохраните полученную программу в папке **Array1** под именем **Zad4.pas**.

Введение массива **B** и работа с ним требует введения переменной **j** для обращения к элементам **B**. Суть решения заключается в просмотре элементов массива **A**, выявлении четных элементов и записи их номеров по текущему значению переменной **j** в массив **B**.

Решение данной задачи имеет следующий вид:

Zad1.pas* •Program1.pas*

```
Program Zad4;
Const n=10;
Type MyArray=Array[1..n] Of Integer;
Var A, B: MyArray;
    i, j: Integer;
Begin
 WriteLn ('Введите ' ,n, ' чисел');
 For i:=1 To n Do ReadLn(A[i]);
 j:=0;
 For i:=1 To n Do
   If A[i] Mod 2 = 0 Then
     Begin
       Inc(j);
       B[j]:=i;
     End;
 For i:=1 To j Do
 Write(B[i]:3);
 WriteLn;
End.
```

8. Выполните команду **Файл – Новый** и составьте программу нахождения значения и номера всех отрицательных элементов. Сохраните полученную программу в папке **Array1** под именем **Zad5.pas**.

Zad1.pas* Program1.pas* •Program2.pas*

```
Program Zad5;
Const n=7;
Type MyArray=Array[1..n] Of Integer;
Var
  A:MyArray; i:Integer;
Begin
  WriteLn('Ввод элементов массива. Не забудьте об отрицательных элементах.');
  For i:=1 To n Do Read(A[i]) ;
    WriteLn('Вывод отрицательных элементов массива и их номеров (индексов)');
  For i:=1 To n do
    If A[i]<0 Then WriteLn (A[i], ' ',i,' ');
End.
```

9. Текст программы поиска номеров всех отрицательных элементов в массиве содержит строки: **For i:=1 To n do** Измените во второй строке **n** на **n+1** и поставьте перед текстом программы директиву компилятора **{$R+}**. Директивы компилятора управляют режимом компиляции. Директива начинается с символа **$** после открывающей скобки комментария, за которым следует имя директивы (состоящее из одной или нескольких букв), определяющее ее назначение. Контроль вхождения в диапазон осуществляется с помощью директивы **{$R+}** (**{$R-}**), при этом устанавливается или отменяется генерация кода контроля вхождения в диапазон. В режиме **{$R+}** все индексы массивов и строк контролируются на выход за границы, а все присваивания значений скалярных переменных и ограниченных типов проверяются на вхождение в диапазон. При выходе за границы выполнение программы прекращается и выдается сообщение об ошибке.
10. Запустите программу на исполнение. При запуске программы появится ошибка времени выполнения: **Индекс выходит за границы массива**. Она является сообщением о том, что значение переменной **i** (индекс массива **A**) вышло за допустимый предел, т.е. за значение константы **n**.
11. Формирование значений элементов массива путем ввода их с клавиатуры – достаточно утомительное занятие. Используйте для этих целей генератор случайных чисел - **Random**. Данная функция возвращает случайное число. Выполните команду **Файл – Новый** и составьте программу для заполнения массива случайными числами. Сохраните полученную программу в папке **Array1** под именем **Zad6.pas**.

```
Program Zad6;
Const n=10;
Type MyArray=Array[1..n] Of Integer;
Var
  A:MyArray;
  i:Integer;
Begin
  {Randomize;}
  WriteLn('Формирование значений элементов массива A');
  For i:=1 To n Do A[i]:=Random(21) {-10};
  WriteLn('Вывод');
  For i:=1 To n Do Write(A[i]:5);
End.
```

12. Запустите несколько раз программу. На экране мы видим одну и ту же последовательность чисел в диапазоне от 0 до 20.
13. Уберите фигурные скобки у -10 и снова запустите несколько раз программу.

```
A[i]:=Random(21)-10;
```

Последовательность чисел на экране одна и та же, но числа из интервала от -10 до 10. Объясните этот факт. Получите числа из интервала от -17 до 25.

14. Уберите фигурные скобки у процедуры **Randomize**. Повторите многократный запуск программы. Последовательности чисел на экране разные. В чем дело? Наш черный ящик, функция **Random**, начинает генерировать в первом случае числа (каким-то неизвестным нам образом) от фиксированного начального числа. Во втором случае эти начальные числа меняются от запуска к запуску (процедурой **Randomize**) и последовательности получаются разные.

Задание 2.

Используя встроенный задачник системы программирования **PascalABC.NET**, выполните решение задач **Array4, Array7**.

Методические указания по выполнению задания:

1. Создайте новое окно, выполнив команду **Файл – Новый**.
2. Откройте встроенный задачник системы программирования **PascalABC.NET**, для этого выполните команду **Модули – Просмотреть задания**. В диалоговом окне **Просмотр учебных заданий** в поле группа заданий выберите **Array**, а в поле номер задания задайте необходимый номер задания и нажмите кнопку **Просмотр**.
3. Решите данную задачу, используя систему программирования **PascalABC.NET**, для этого перейдите в систему программирования **PascalABC** и выполните команду **Модули – Создать шаблон программы**. В окне **Загрузка учебного задания** в поле **Задание** введите **Array4 (Array7)** и нажмите кнопку **Загрузка**. После этого в системе программирования **PascalABC** загрузится шаблон программы, где указан модуль подключения задачника и название задачи, которую мы решаем.
4. Просмотрите еще раз решение задачи, для этого выполните команду **Программы – Выполнить** или нажмите соответствующую кнопку на панели инструментов.
5. В разделе описания переменных **var** выполните описание переменных в соответствии с условием задачи.
6. Составьте программу для решения задачи и выполните её тестирование. При необходимости исправьте ошибки. Чтобы задание считалось выполненным, запустите программу необходимое количество раз.
7. Сохраните созданные программы в папке **Array1.**

Задание 3.

Используя систему программирования **PascalABC.NET**, выполнить индивидуальные задания. Для каждой задачи построить блок-схему алгоритма решения, используя инструменты **DiagramDesigner.** Созданные программу и её блок-схему сохранить в папку **Array1**, в названии файла укажите номер задачи.

Задания:

1. Вывести элементы массива на экран в обратном порядке.
2. Дан массив. Все его элементы: увеличить в 2 раза; уменьшить на число A; разделить на первый элемент.
3. Дан массив целых чисел. напечатать все четные элементы; все элементы, оканчивающиеся нулем.

Задание 4.

Проверьте файлы, которые были сохранены в папку **Array1** в процессе выполнения практической работы. Заархивируйте папку с помощью программы архиватора, установленной на ваш компьютер. Передайте полученный архив преподавателю на проверку, разместив его в общих папках.

Практическая работа №9.

«Операции с элементами массивов, обмен элементами»

Цель работы: сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с использованием одномерных массивов.
Оборудование, технические и программные средства: персональный компьютер, среда программирования **Visual Studio**.

Задание 1.

В системе программирования **Visual Studio** составьте программу по заданию. Протестируйте работу программы. Выполните задания на модификацию созданной программы.

Методические указания по выполнению задания:

1. Запустите среду программирования **Visual Studio**.
2. В папке своей группы на диске создайте папку **Массивы1**, в дальнейшем все программы необходимо сохранять в эту папку.
3. Создайте новое консольное приложение с именем **Zad1**. В окне редактора наберите текст программы, которая находит сумму 30 целых чисел.

```
static int[] Input()
{
    int[] a = new int[30];
    for (int i = 0; i < 30; ++i)
    {
        Console.Write("a[{0}]= ", i);
        a[i] = int.Parse(Console.ReadLine());
    }
    return a;
}

static int Sum(int[] a)
{
    int sum = 0;
    foreach (int elem in a)
    {
        sum += elem;
    }
    return sum;
}

static void Main()
{
    int[] a = Input();
    Console.WriteLine("Сумма элементов массива={0}", Sum(a));
}
```

4. Проверьте правильность работы программы.
5. Создайте новое консольное приложение с именем **Zad2**. Измените ранее созданную программу так, чтобы определялась сумма элементов массива, кратных заданному числу.

Решение задачи изменится незначительным образом. Добавляется описание еще одной переменной для хранения значения числа, на кратность которому проверяются значения элементов массива.

Появляется оператор ввода заданного числа и изменяется оператор из тела цикла метода **Sum**:

```
static int Sum(int[] a)
{
    int sum = 0;
    Console.Write("Введите число k=");
    int k = int.Parse(Console.ReadLine());
    foreach (int elem in a)
    {
        if (elem%k==0)
        {
            sum += elem;
        }
    }
    return sum;
}
```

6. Создайте новое консольное приложение с именем **Zad3**. Измените программу так, чтобы определялось количество положительных и отрицательных элементов в данном массиве.

Суть основного изменения программы заключается во введении двух переменных (счетчиков – **pos**, **neg**) для хранения значений количества положительных и отрицательных элементов в массиве.

Внесите в программу следующие изменения:

```
static void Sum(int[] a)
{
    int pos = 0;
    int neg = 0;
    foreach (int elem in a)
    {
        if (elem>0)
        {
            pos++;
        }
        else
        {
            if (elem<0)
            {
                neg++;
            }
        }
    }
    Console.WriteLine("Кол-во положительных={0}", pos);
    Console.WriteLine("Кол-во отрицательных={0}", neg);
}

static void Main()
{
    int[] a = Input();
    Sum(a);
}
```

7. Создайте новое консольное приложение с именем **Zad4**. Измените программу так, чтобы в массив **B** записывались номера четных элементов массива **A.**

Введение массива **B** и работа с ним требует введения переменной **j** для обращения к элементам **B**. Суть решения заключается в просмотре элементов массива **A**, выявлении четных элементов и записи их номеров по текущему значению переменной **j** в массив **B.**

Текст программы будет иметь следующий вид:

```
static int[] Input()
{
    int[] a = new int[10];
    for (int i = 0; i < 10; ++i)
    {
        Console.Write("a[{0}]= ", i);
        a[i] = int.Parse(Console.ReadLine());
    }
    return a;
}
static void Print(int[] a, int j)
{
    for (int i = 0; i < j; ++i)
    {
        Console.Write("{0} ", a[i]);
    }
}

static void Main()
{
    int[] a = Input();
    int[] b;
    b = new int[10];
    int j = 0;
    for (int i = 0; i < 10; ++i)
    {
        if (a[i] % 2 == 0)
        {
            b[j] = i;
            j++;
        }
    }
    Print(b,j);
}
```

8. Создайте новое консольное приложение с именем **Zad5**. Составьте программу нахождения значения и номера всех отрицательных элементов массива.

Формирование значений элементов массива путем ввода их с клавиатуры – достаточно утомительное занятие. Используйте для этих целей генератор случайных чисел - **Random.**

```
static int[] Input()
{
    Random rnd = new Random();
    int[] a = new int[10];
    for (int i = 0; i < 10; ++i)
    {
        a[i] = 7-rnd.Next(10);
    }
    return a;
}

static void Main()
{
    int[] a = Input();
    Console.WriteLine("Вывод отрицательных элементов массива и их номеров");
    for (int i = 0; i < 10; ++i)
    {
        if (a[i]<0)
        {
            Console.WriteLine("A[{0}]={1}, i={0}", i, a[i]);
        }
    }
}
```

Задание 2.

Используя систему программирования **Visual Studio**, выполнить индивидуальные задания. Для каждой задачи построить блок-схему алгоритма решения, используя инструменты **DiagramDesigner.** Созданные программу и её блок-схему сохранить в папку **Массивы1**, в названии файла укажите номер задачи.

Задания:

1. Вывести элементы массива на экран в обратном порядке.
2. Дан массив. Все его элементы: увеличить в 2 раза; уменьшить на число А; разделить на первый элемент.
3. Дан массив целых чисел, напечатать все элементы массива, оканчивающиеся нулем.

Практическая работа №10.

«Обработка двумерных массивов»

Цель работы: сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с использованием двумерных массивов.
Оборудование, технические и программные средства: персональный компьютер, среда программирования **Visual Studio**

Задание. В двумерном массиве **B**, состоящем из **N** строк и **M** столбцов, находятся размеры начисленной заработной платы **N** работников предприятия за **M** месяцев, т. е. **B[I, K]** – это заработная плата работника предприятия с номером **I** за месяц с номером **K**.

Составить программу, позволяющую выполнить:

Ÿ ввести с клавиатуры количество работников и количество месяцев, а также размеры заработной платы, для ввода исходных размеров заработной платы использовать подпрограмму-процедуру;

Ÿ определить наименьшую заработную плату за месяц с номером 1;

Исходные данные:

N – количество работников предприятия (целое число);

M – количество месяцев (целое число);

B[I, K] (I = 1, 2, 3,…, N; K = 1, 2, 3,…, M) – массив значений заработной платы (значения вещественного типа);

Требуется определить:

MIN – наименьшую заработную плату за месяц с номером 1 (число с дробной частью);

Значения заработной платы представляют собой конечный набор (**N´ M** значений) однотипных данных. Для их хранения можно использовать двумерный массив **B**, состоящий из **N** строк и **M** столбцов (**B [1, 1]** – заработная плата работника номер 1 за первый месяц, **B [1, 2]** – заработная плата работника номер 1 за второй месяц, **B [2, 1]** – заработная плата работника номер 2 за первый месяц и т.д.

Абстрагируясь от физического смысла значений элементов массива, сформулируем задачу обработки абстрактного двумерного массива **B**.

Для заданного двумерного массива B, состоящего из N строк и M столбцов, выполнить:

1. ввести с клавиатуры количество строк и столбцов массива, а также значения элементов массива;
2. определить наименьшее значение среди элементов столбца с номером 1 (MIN)

Тестовый вариант расчетов

Исходные данные	**Результаты**
N = 4; M = 3 Массив B:	MIN = 4000;

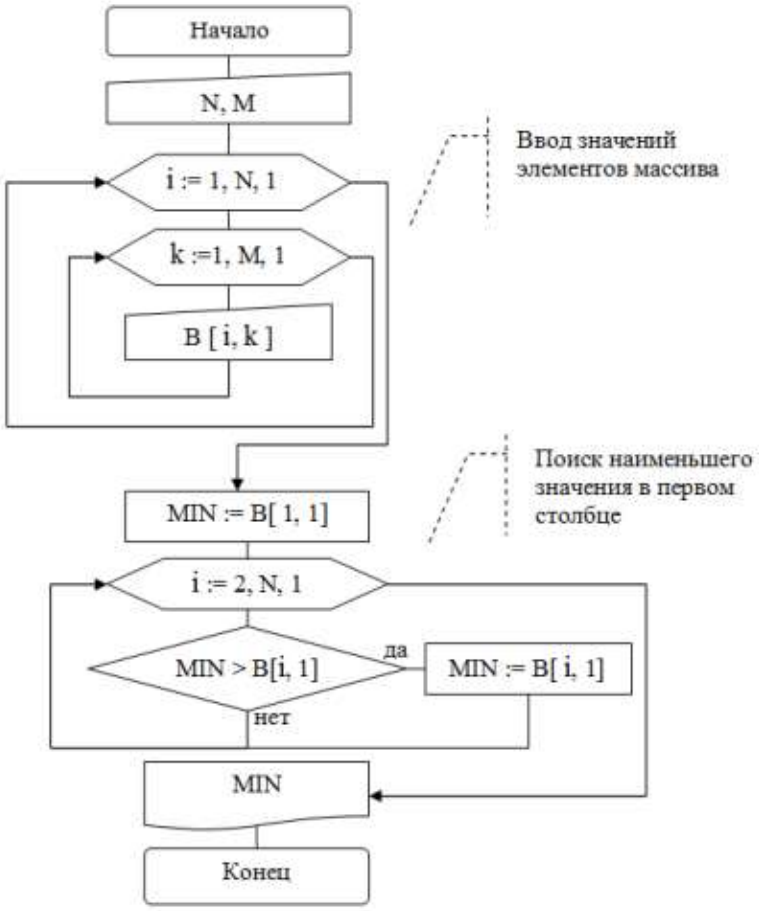
5000	7000	11000
8000	12000	13000
11000	4000	12000
4000	5000	5000

Алгоритм решения задачи

Определение наименьшего значения элементов первого столбца (**MIN**) выполняется попарным сравнением элементов, при этом второй индекс элементов массива будет неизменным и равным 1. Блок-схема алгоритма

Программа

```
program Lab_4;
var n, m, i, k : byte;
min : real;
b : array [ 1 .. 10, 1 .. 10 ] of real;
begin
  Writeln ( ' Введите количество строк и столбцов в массиве B ' );
  Readln ( n , m );
  Writeln ( ' Вводите значения зарплат из массива B построчно ' );
  for i := 1 to n do
  for k := 1 to m do
    Readln ( b[ i , k ] );
  {поиск минимального значения в первом столбце массива B}
  min := b [ 1 , 1 ];
  for i  := 1 to n do
    if min  > b [ i , 1 ] then min := b [ i , 1 ];
  Writeln ( ' Минимальная зарплата за первый месяц= ' , min : 8 : 2 );
end.
```



Блок-схема алгоритма решения задачи

Задание 2: составить программу, позволяющую выполнить следующее:

1. Ввести с клавиатуры количество работников и количество месяцев, а также размеры заработной платы;
2. определить наибольшее значение заработной платы работника №2;
3. провести расчеты для N=4, M=3 если

	1 месяц	2 месяц	3 месяц
1 работник	5000	7000	11000
2 работник	8000	12000	13000
3 работник	11000	4000	12000
4 работник	4000	5000	5000

План выполнения работы:

1. Указать исходные данные и результаты, которые должны получиться.
2. Выполнить формализацию задачи.
3. Подготовить тестовый вариант расчетов (с помощью калькулятора)
4. Составить алгоритм и программу решения задачи.
5. Выполнить ввод и отладку программы в среде программирования.
6. Предъявить результаты расчетов преподавателю.

Практическая работа №11.

«Работа со строковыми переменными»

Цель работы: сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с использованием строковых переменных.
Оборудование, технические и программные средства: персональный компьютер, среда программирования **Visual Studio**.

Задание 1.

В системе программирования **Visual Studio** составьте программы, демонстрирующие работу с символьным типом данных в **C#**. Протестируйте работу программ. Выполните задания на модификацию созданных программ.

Методические указания по выполнению задания:

1. Запустите среду программирования **Visual Studio**.
2. В папке своей группы на диске создайте папку **Строки1**, в дальнейшем все программы необходимо сохранять в эту папку.
3. Тип **char** предназначен для хранения символа в кодировке **Unicode**. Символьный тип относится к встроенным типам данных **C#** и соответствует стандартному классу **Char** библиотеки .**Net** из пространства имен **System**. В этом классе определены статические методы, позволяющие задавать вид и категорию символа, а также преобразовывать символ в верхний, или нижний регистр, а также в число.
4. Создайте новое консольное приложение с именем **Zad1**. Рассмотрим применение методов для работы с символьным типом. Составьте программу по образцу и протестируйте её. Выпишите в тетрадь основные методы для работы с символьным типом.

```
static void Main(string[] args)
{
    Console.WriteLine("{0,5} {1,8} {2,15}");
    for (ushort i=0; i<255; i++)
    {
        char a = (char)i;
        Console.Write("\n{0,5} {1,8}", i, a);
        if (char.IsLetter(a))
            Console.Write("{0,20}", "Буква");
        if (char.IsUpper(a))
            Console.Write("{0,20}", "Верхний регистр");
        if (char.IsLower(a))
            Console.Write("{0,20}", "Нижний регистр");
        if (char.IsControl(a))
            Console.Write("{0,20}", "Управляющий символ");
        if (char.IsNumber(a))
            Console.Write("{0,20}", "Число");
        if (char.IsPunctuation(a))
            Console.Write("{0,20}", "Знак препинания");
        if (char.IsDigit(a))
            Console.Write("{0,20}", "Цифра");
        if (char.IsSeparator(a))
            Console.Write("{0,20}", "Разделитель");
        if (char.IsWhiteSpace(a))
            Console.Write("{0,20}", "Пробельный символ");
    }
}
```

5. Используя символьный тип можно организовать массив символов и работать с ним используя класс **Array**. Создайте новое консольное приложение с именем **Zad2**. Составим программу, которая строчные символы меняет на прописные и меняет символы в строке местами. Протестируйте работу программы.

```
namespace Zad1
{
    class Program
    {
        static void Print(char[] a)
        {
            foreach (char elem in a)
            {
                Console.Write(elem);
            }
            Console.WriteLine();
        }
        static void Main(string[] args)
        {
            char[] a = "кол около колокола".ToCharArray();
            Console.WriteLine("Исходный массив a:");
            Print(a);
            for (int x=0; x<a.Length; x++)
            {
                a[x] = char.ToUpper(a[x]);
            }
            Array.Reverse(a);
            Console.WriteLine("Измененный массив a:");
            Print(a);
        }
    }
}
```

6. Измените программу так, чтобы в массиве А подсчитывалось количество знаков пунктуации.

Практическая работа №12.

«Использование стандартных функций и процедур для работы со строками»

Цель работы: сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с использованием стандартных функций и процедур для работы со строками.

Оборудование, технические и программные средства: персональный компьютер, среда программирования **Visual Studio**.

Задание.

В системе программирования **Visual Studio** составьте программы, демонстрирующие работу со строковым типом данных в **C#**. Протестируйте работу программ. Выполните задания на модификацию созданных программ.

Методические указания по выполнению задания:

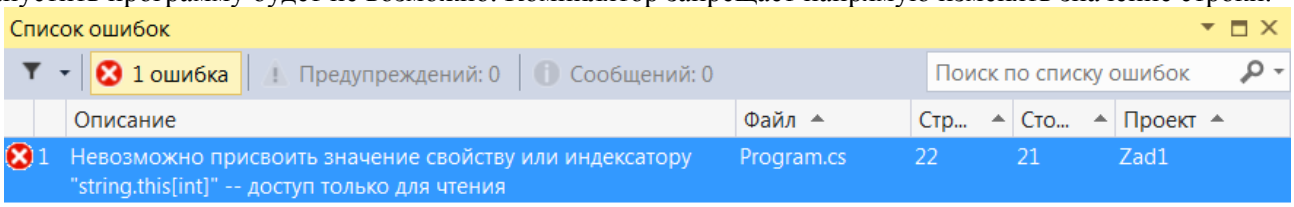
1. Тип **string**, предназначенный для работы со строками символов в кодировке **Unicode**, является встроенным типом **C#**. Ему соответствует базовый класс **System.String** библиотеки **.Net**. Тип **string** относится к ссылочным типам, хотя работа с ним во многом напоминает работу с размерными типами. С объектом типа **string** можно работать посимвольно, т.е. поэлементно. Класс **string** обладает богатым набором методов для сравнения строк, поиска в строке и других действий со строками. Все методы возвращают ссылку на новую строку, созданную в результате преобразования копии исходной строки. Для того чтобы сохранить данное преобразование, нужно установить на него новую ссылку. Область применения типа **string** – это поиск, сравнение, извлечение информации из строки.
2. Создайте новое консольное приложение с именем **Zad3**. Составьте программу для нахождения количества вхождений символа в строке и протестируйте её.

```
static void Main(string[] args)
{
    string a = "кол около колокола";
    Console.WriteLine("Дана строка: {0}", a);
    char b = 'o';
    int k = 0;
    for (int x=0; x<a.Length; x++)
    {
        if (a[x]==b)
        {
            k++;
        }
    }
    Console.WriteLine("Символ {0} содержится в ней {1} раз", b, k);
}
```

3. Попробуем заменить все вхождения буквы о на букву а.

```
static void Main(string[] args)
{
    string a = "кол около колокола";
    Console.WriteLine("Дана строка: {0}", a);
    char b = 'o';
    char c = 'a';
    int k = 0;
    for (int x=0; x<a.Length; x++)
    {
        if (a[x]==b)
        {
            a[x]=c;
        }
    }
    Console.WriteLine("Символ {0} содержится в ней {1} раз", b, k);
}
```

Запустить программу будет не возможно. Компилятор запрещает напрямую изменять значение строки.



4. Строковый тип **StringBuilder** определен в пространстве имен **System.Text** и предназначен для создания строк, значение которых можно изменять. Объекты данного класса всегда создаются с помощью явного вызова конструктора класса, т.е. через операцию **new**. С объектами класса **StringBuilder** можно работать посимвольно. Внесите изменения в созданную ранее программу следующим образом:

```
static void Main(string[] args)
{
    StringBuilder a = new StringBuilder ("кол около колокола");
    Console.WriteLine("Дана строка: {0}", a);
    char b = 'o';
    char c = 'a';
    int k = 0;
    for (int x=0; x<a.Length; x++)
    {
        if (a[x]==b)
        {
            k++;
        }
    }
    Console.WriteLine("Символ {0} содержится в ней {1} раз", b, k);
}
```

5. Протестируйте работу программы. Она ничем не отличается от аналогичной программы с типом **string**.
6. Измените программу так, чтобы все вхождения буквы о были заменены на букву а. Возникает ли ошибка при работе данной программы.

```
static void Main(string[] args)
{
    StringBuilder a = new StringBuilder ("кол около колокола");
    Console.WriteLine("Дана строка: {0}", a);
    char b = 'o';
    char c = 'a';
    int k = 0;
    for (int x=0; x<a.Length; x++)
    {
        if (a[x]==b)
        {
            a[x]=c;
        }
    }
    Console.WriteLine("Преобразованная строка: {0}", a);
}
```

7. Завершите работу с программой.

Практическая работа №13. Работа с данными типа множество.

Цель работы: Овладеть техникой составления программ для работы с множествами, её компиляции и записи на диск под заданным именем.

Задачи работы : Научиться применять основные принципы алгоритмизации и программирования в решении задач, использующих в качестве исходных данных множество символов.

Задание Составить программы для предложенных заданий, отладить их и записать на диск D.

Требования к отчету :
Итоги практической работы представить в виде блок-схемы алгоритма и текста программы, привести значения исходных данных и значения полученных результатов (при необходимости вывести на печать).
Методические указания по выполнению задания

1. Ознакомьтесь с теоретическим материалом, необходимым для выполнения работы:

Множество – совокупность однотипных элементов, рассматриваемых как единое целое.
Элементы множества не пронумерованы и неупорядочены.
Каждый отдельный элемент множества не идентифицируется и с ним нельзя выполнять какие-либо действия. Действия могут выполняться только над множеством в целом.
Описание множеств и операции над ними Описывают множества, используя ключевое слово set of: Var M: set of Char;
Присвоить значение переменной множественного типа можно так: M:= [‘a’..’y’]
Для того, чтобы определить, принадлежит ли элемент множеству, используют зарезервированное слово in: If ‘s’ in M then…
Над множествами можно выполнять операции:
- объединения (+);
- пересечения (*);
- разности (-);
- операции отношений (=, <>, , <=, >=);
- проверка принадлежности элемента множеству.

Например, используя операцию объединения, к множеству M можно добавить символ ‘z’, представив его как множество из одного элемента: M:= M+[‘z’]. Либо эту же операцию выполняет процедура Include(M, ‘z’). Процедура Exclude предназначена для исключения элемента из множества. Например, Exclude(M, ‘s’) исключит символ ‘s’ из множества M. Константы множественного типа записывают с помощью квадратных скобок и списка эле- ментов: Abc=[‘A’, ‘B’, ‘C’]; {Множество из трех букв} Cifr=[0..9]; {Множество цифр} P=[] – пустое множество Особенности множества: 1) В нем могут содержаться элементы только одного

базового типа. 2) Порядок элементов множества не фиксируется : {7, 3,1}. 3) В множестве не может быть одинаковых элементов. 18 2. Выполните следующие упражнения: 1. Заполнить множество NB случайными значениями в пределах от 0 до 49. Количество элементов множества запрашивается с клавиатуры: uses crt; Var nb : set of 1..200; k, n, i: integer; Begin clrscr; nb :=[]; {Создаем пустое множество} randomize; {Подключаем процедуру случайных чисел} Writeln('Введите количество элементов множества'); Readln(k); For i:=1 to k do begin n:=Random(50); nb:=nb+[n]; Write(n:3); end; end. Сохраните программу в свою папку на диск D: P7PR1 2. Дано множество A, содержащее знаки препинания английского алфавита. Вывести множество A на экран: var A: set of char; ch: char; Begin clrscr; A:=['.',',',';',':','!','?']; Writeln('A='); For ch:=#0 to #127 do if ch in A then Write(ch:3); Readln; end. Сохраните программу в свою папку на диск D: P7PR2 3. Выполните самостоятельно 1. Составить программу, которая моделирует «лототрон (5 из 36)» т.е. случайную выборку 5 шаров из контейнера, содержащего 36 шаров, пронумерованных от единицы до 36. Сохранить программу под именем P7PR3. 2. Дано множество A – множество гласных букв английского алфавита и множество B, состоящее из символов английского алфавита: a, o, !, ?. Вывести на экран исходные множества. Найти множество C, являющимся объединением A и B. Сохранить программу под именем P7PR4.

Практическая работа №14.

«Работа с файлом последовательного доступа»

Цель работы: Овладеть техникой составления программы с использованием файлов последовательного доступа, ее компиляции и записи на диск под заданным именем.

Задание

Составить программы для предложенных заданий, используя файлы, отладить их и записать на диск D в свою папку.

Требования к отчету

Итоги практической работы представить в виде блок-схемы алгоритма и текста программы, привести значения исходных данных и значения полученных результатов (при необходимости вывести на печать).

Технология работы

1. Ознакомьтесь с теоретическим материалом, необходимым для выполнения работы:

1.1 Операции перемещения по файлу

- seek (f, n) – устанавливает указатель в открытом файле f на компонент с номером n. Затем значение компонента может быть считано. Нумерация компонентов идет от 0.
- filepos (f) –определяет текущую позицию указателя (номер элемента) в открытом файле.
- filesize (f) –определяет размер (кол-во элементов) файла f
- eof (f) –возвращает логическое значение true, если достигнут конец файла.

Специальные операции:

- erase (f) – удаляет файл, связанный с файловой переменной f.
- rename (f, NewName) – присваивает файлу, связанному с файловой переменной f, новое имя NewName.

1.2 Текстовые файлы

Текстовый файл является файлом последовательного доступа, и его можно представить как набор строк произвольной длины, разделенных специальными символами, которые называются «конец строки» (eoln). Символ конца строки при просмотре файла не виден, так же как и символ конца файла. В программе файловая переменная текстового типа описывается следующим образом:

Var < имя файловой переменной >: text;

Каждый символ представлен во внутреннем коде (ASCII) и занимает 1 байт.

Главная особенность текстовых файлов в том, что можно считывать из него и записывать в них элементы типа char, integer, real, string, boolean. Если информация несимвольная, то в процессе чтения или записи происходит ее преобразование из символьной формы во внутреннюю и обратно.

Для работы с текстовыми файлами можно использовать все процедуры и функции, используемые при работе с файлами любых типов, за исключением процедуры seek, функций filepos, filesize. Дело в том, что заранее неизвестно, элементы какого типа находятся в файле.

Для работы с текстовыми файлами определены дополнительные операции.

Функция:

eoln (f) –возвращает логическое значение true, если достигнут конец строки, и false в противном случае.

Процедуры:

1. append (f) – открытие уже существующего текстового файла f для добавления данных в конец файла;

2. writeln (f) – завершение текущей строки текстового файла при его записи (записывает символ конца строки);

3. writeln (f, x1, x2, …, xn) – запись в файл f значений переменных x1, x2, …, xn; после выполнения операции записи осуществляется переход к новой строке файла;

4. readln (f) – переход к началу следующей строки файла f при его чтении;

5. readln(f, x1, x2, …, xn) – чтение значений n элементов из файла f в переменные x1, x2, …, xn; после выполнения операции чтения осуществляется переход к новой строке файла.

2. **Выполните следующие упражнения:**

Упражнение 1. Программа, в результате выполнения которой выводятся все четные числа из данного файла int с целочисленными компонентами.

1. Наберите текст программы:

```
Program rem (input, output);
type v=file of integer;
var int: v; i: integer;
begin reset (int);
while not eof (int) do
   begin
     read (int, i);
     if i mod 2=0 then writeln (i)
  end end.
```

2. Сохраните программу в свою папку.

Упражнение 2. Программа, которая формирует типизированный файл из целых чисел, вводимый с клавиатуры. Их количество заранее не известно. Признаком конца ввода является 0. Программа находит: сумму и произведение чисел из файла, разность между предпоследним и вторым по счету числами, наибольшее из чисел.

1. Наберите текст программы:

```
Program rabf
type file_type=file of integer;
var f: file_type;
    sum, mult, r, k1, k2, max: integer;
begin
    writeln ('Введите элементы файла, окончание - 0');
    {Запись элементов в файл}
    assign (f, 'data.dat'); rewrite (f);
    repeat
       readln (r); if r<>0 then write (f, r);
    until r=0;
    {Вычисление результатов}
    seek (f, 0); sum:=0; mult:=1;
    read (f, r); max:=r; seek (f, filepos(f)-1);
    while not eof (f) do
       begin
         read (f, r); sum:=sum+r; mult:=mult*r;
         {Поиск максимального элемента}
         if max<r then max:=r;
       end;
    seek (f, 1); read (f, k1); {Чтение второго компонента}
    seek (f, filesize (f)-2); read (f, k2); { Чтение предпоследнего компонента}
    close (f);
```

```
{Вывод результатов}
writeln ('Сумма = ', sum, #10#13'Произведение = ', mult);
writeln ('Разность = ', k2-k1, #10#13'Максимум = ', max);
end.
```
Сохранить программу.

Контрольные вопросы:

1. Перечислите установочные и завершающие операции работы с файлами.
2. Какие операции ввода-вывода существуют для работы с файлами?
3. Перечислите специальные операции для работы с файлами.
4. Особенности работы с текстовыми файлами

Практическая работа №15.
«Работа с файлом произвольного доступа»

Цель работы: Овладеть техникой составления программы с использованием файлов произвольного доступа, ее компиляции и записи на диск под заданным именем.

Задание

Составить программы для предложенных заданий, используя записи, отладить их и записать на диск D: в свою папку.

Требования к отчету

Итоги практической работы представить в виде блок-схемы алгоритма и текста программы, привести значения исходных данных и значения полученных результатов (при необходимости вывести на печать).

Технология работы

1. Ознакомьтесь с теоретическим материалом, необходимым для выполнения работы:

- Файл произвольного (прямого) доступа – это файл, доступ к элементам которого, осуществляется по их номеру в любом порядке. Для таких файлов разрешается одновременная запись и считывание данных. Файл с произвольным доступом обладает заранее заданной структурой и состоит из записей. Примерами файла произвольного доступа являются базы данных.
- Запись – структурированный комбинированный тип данных, состоящий из определенного числа компонентов разного типа, которые называются полями записи. Описание переменной типа запись начинается ключевым словом record, за которым следует список полей с указанием их типов. Заканчивается описание служебным словом end.

```
Var
  Name: record
   N1: byte;
   N2: string;
   N3: real;
  End;
```

ИЛИ

```
Type anketa=record
            fio: string[45];
            pol: char;
            dat_r: string[8];
            adres: string[50];
            curs: 1..5;
            grupp: string[3];
          end;
Var student: anketa;
```

Для того, чтобы обратиться к полю записи, необходимо указать имя переменной и через точку – имя поля. Например, обращение к полю curs переменной student:

student.curs :=3;

Использование полей записи в выражениях и условиях идентично использованию обычных переменных. Для работы с полями записи пользуются оператором:

with <имя_записи> **do** <действие с полем записи>;

Внутри оператора *with* с полями записи можно работать, как с обычными переменными: без указания составного имени. **Пример: Заполнить сведения о студенте**:

```
for I:=1 to 100 do
  with student[I] do
  begin
    writeln ('введите сведения о', I , '-м
студенте');
    writeln ('введите фамилию, имя и отчество');
    readln (fio);
    writeln ('введите дату рождения');
    readln (dat_r);
    writeln ('введите адрес');
    readln(adres);
    writeln ('введите курс');
    readln(curs);
    writeln ('введите группу');
    readln (grupp);
  end;
```

Записи часто используются при работе с таблицами, где каждая запись – это одна строка таблицы. По этому для обработки всей таблицы необходимо использовать массивы записей.

Пример:

```
type
  ved = record
      fio: string[35];
      date: integer;
      zarplata: real;
  end;
var
  a: array [1..30] of ved;
```

При обращении к полю *fio* в пятой строке таблицы достаточно указать соответствующий элемент массива: *a[5].fio*.

2. Выполните следующие упражнения:

Упражнение 1. ***О каждом студенте известна следующая информация:***

- ***фамилия, инициалы;***
- ***год рождения;***
- ***группа;***
- ***отметка по математике;***
- ***отметка по истории;***
- ***отметка по ВТ;***
- ***отметка по статистике.***

Сформировать таблицу, записав в нее известную информацию о каждом студенте и его средний балл. Подсчитать средний балл по каждому предмету, вывести таблицу на экран в алфавитном порядке.

1. Наберите текст программы:

```
Program stud;
Type   tablica=record  {Описание записи о каждом студенте}
         name: string[15];
         group: string[8];
         god: integer;
         vt, history, stat, math: byte;
         sr_bal: real;
  end;
var i, j, n :integer; a: tablica;
    mas:array [1..30] of tablica; {Таблица - массив записей}
    s_vt, s_history, s_stat, s_math: real; {Переменные для хранения средних значений по предметам}
```

```
begin
  write('n='); readln(n); {Ввод количества записей}
  for i=1 to n do{Ввод элементов массив записей}
    with mas[i] do
    begin
      writeln('i=',i:4);
      writeln('FIO');
      readln(name);
      write('Group');
      readln(group);
      write('Year');
      readln(god);
      write('Otsenki');
      readln(vt, history, stat, math);
      sr_bal:=(vt+history+stat+math)/4;
    end;
  s_vt:=0; s_history:=0; s_stat:=0; s_math:=0;
  for i=1 to n do {Вычисление среднего балла по каждому предмету}
  begin
    s_vt:=s_vt+mas[i].vt;
    s_history:=s_history+mas[i].history;
    s_stat:=s_stat+mas[i].stat;
    s_math:=s_math+mas[i].math;
  end;
  for i=1 to n do {Упорядочение записей массива в алфавитном порядке фамилий}
    for j:=1 to n-1 do
      if  mas[j].name> mas[j+1].name then
      begin
        a:=mas[j];
        mas[j]:=mas[j+1];
        mas[j+1]:=a;
      end;
  clrscr;
  write('  ';4, 'FIO ',' ';4); {Вывод результатов}
  write('  ';2, ' GROUP ',' ';2);
  write('  ';5, ' OTSENKI ',' ';5);
  writeln('Sr/ Bal  ');
  for i=1 to n do
    with mas[i] do
    begin
      write(name:15);
      write('  ',group:8);
      write((' ',god:4);
      writeln('  ',vt:3, '  ',history:3, '  ',stat:3, '  ',math:3, '  ',sr_bal:5:2);
    end;
  writeln('  Sr. Bal: ', '  ', s_vt:3:1, '  ', s_history:3:1, '  ', s_stat:3:1, '  ', s_math:3:1);
end.
```

2. Сохраните программу в свою папку: **<F2>** **D:\Familiya\P12PR1**

3. Выполнить самостоятельно:

1. Известна информация о сотрудниках некоторого предприятия:

- фамилия, инициалы;
- год рождения;
- должность;
- стаж;
- оклад.

Сформировать таблицу, записав в нее известную информацию о каждом сотруднике. Создать поле «Зарплата», добавляя 10% к окладу, если стаж работы более 10 лет, и 15%, если более 20. Определить количество пенсионеров (60 лет и старше) и количество молодых специалистов (моложе 25 лет). Сохранить программу под именем **P12PR2.**

Контрольные вопросы:

1. Понятие записи в Паскале.
2. Как описывается запись?
3. Назовите основные типы операций работы с записями.
4. С помощью какого оператора осуществляется доступ к полям записи?

Практическая работа №16.

«Организация процедур и функций»

Цель работы: Овладеть техникой составления программы с использованием файловых переменных.

Задание

Составить программы для предложенных заданий, используя файлы, отладить их и записать на диск D: в свою папку.

Требования к отчету

Итоги практической работы представить в виде блок-схемы алгоритма и текста программы, привести значения исходных данных и значения полученных результатов (при необходимости вывести на печать).

Технология работы

1. Ознакомьтесь с теоретическим материалом, необходимым для выполнения работы:

понятие файла употребляется в двух смыслах:

a) как поименованная информация на внешнем устройстве (внешний файл);

b) как переменная файлового типа в программе (внутренний файл).

В программе между этими объектами устанавливается связь. Вследствие этого все, что происходит в процессе выполнения программы с внутренним файлом, дублируется во внешнем файле. С элементами файла можно выполнять только две операции: читать из файла и записывать в файл.

Для использования файлов в программе должна быть объявлена переменная файлового типа. В зависимости от способа объявления этой переменной различают три типа файлов:

<имя>:file of<mun> ; - типизированный файл. Тип может быть любой кроме файла;

<имя>:text; - текстовый файл;

<имя>:file; - нетипизированный файл.

Доступ к файлу в программе происходит с помощью переменных файлового типа. Переменную файлового типа описывают в разделе описаний одним из трех способов:

Примеры описания файловых переменных:

```
var
f1: file of char; {типизированный файл }
f3: file {нетипизированный файл };
t: text {текстовый файл};
```

или

```
type
  c=text;
var
  f, f1:c;
```

Процедуры и функции, применимые для файлов любых типов

Для начала работы с файлами необходимо связать файловую переменную в программе с файлом на диске:

Assign(f, FileName) - Связывает файловую переменную f с физическим файлом, полное имя которого задано в строке FileName.

Файл должен находиться в текущем каталоге, либо к нему прописывается путь. Эту процедуру нельзя использовать для открытого файла.

После установления связи между файловой переменной и именем файла на диске нужно открыть файл с помощью процедур reset или rewrite.

Reset(f) - Открывает для чтения существующий файл, с которым связана файловая переменная f, и ставит указатель на начало файла, т.е. на компонент с номером 0. Процедура завершается с сообщением об ошибке, если указанный файл не найден.

Если f - типизированный файл, то процедурой reset он открывается для чтения и записи одновременно.

Rewrite(f) - Открывает файл для записи, устанавливая указатель на компонент с номером 0. Если указанный файл уже существовал, то его содержимое будет уничтожено. Добавление информации в уже существующий файл для различных видов файлов производится различными способами.

По завершении работы с файлом его необходимо закрыть:

Close(f) - При создании нового или изменении имеющегося файла эта процедура обеспечивает сохранение в файле всех новых записей и регистрацию файла в каталоге. Для определения конца файла используется функция:

EOF(f): boolean - Значение этой функции равно TRUE, если файловый указатель стоит в конце файла, т.е. встретился признак конца файла, и FALSE - в противном случае. При записи это означает, что очередной компонент будет добавлен в конец файла, а при чтении - что файл исчерпан.

Физический файл может быть переименован в процессе работы программы:

Rename(f, NewName) - Переименование возможно только после закрытия файла.

Физический файл может быть удален:

Erase(f) - Перед использованием процедуры удаляемый файл должен быть закрыт.

Read(f,a) - Считывает из открытого файла элемент a

Write(f, a) - Записывает в файл f элемент a.

Filesize(f) - Возвращает число реальных компонентов в открытом файле f.

2. Выполните следующее упражнение: Записать n вещественных чисел в файл и вывести содержимое этого файла.

```
Var
f:file of real;
a: real;
i, n: Integer;
Begin
clrscr;
Assign(f,'D:\Number.dat');                {Связываем файловую переменную с файлом на диске
Rewrite(f);                               {Открываем пустой файл для записи}
Writeln('Введите количество чисел'); {Определяем количество элементов в файле}
Readln(n);
For i:=1 to n do
 begin
    Write('a=');
     read(a);                             {Считываем элемент}
    write(f,a);                           {Записываем в файл f элемент a}
end;
Close (f);                                {Закрываем файл}
Writeln;
Assign(f,'D:\Number.dat');
Reset(f);                                 {Открываем пустой файл для чтения}
repeat
    Read (f,a);                           {Считываем из файла f элемент a}
    Writeln(a);                           {Выводим элемент}
     Until Eof(f);                        {Достигнут конец файла}
  Close (f);                              {Закрываем файл}
end.
```

Сохраните программу на своем диске.

Контрольные вопросы:

1. Понятие файла в Паскале.
2. Что называют файловым типом переменной?
3. Какова организация доступа к файлам?
4. Назовите основные типы операций работы с файлами

Практическая работа №17.

«Программирование модуля»

Цель работы: Овладеть техникой составления программы с программированием собственных модулей, ее компиляции и записи на диск под заданным именем.

Задание

Составить программы для предложенных заданий с использованием динамических массивов, отладить их и записать в свою папку на диск D.

Требования к отчету

Итоги практической работы представить в виде блок-схемы алгоритма и текста программы, привести значения исходных данных и значения полученных результатов (при необходимости вывести на печать).

Технология работы

1. Ознакомьтесь с теоретическим материалом, необходимым для выполнения работы:

Модуль – это автономная программная единица, включающая в себя различные компоненты: константы, переменные, типы, процедуры и функции.

Модуль имеет следующую структуру:

UNIT – имя модуля;
INTERFACE – интерфейсная часть;
IMPLEMENTATION – исполняемая часть;
BEGIN – инициирующая часть;
END.

Заголовок модуля состоит из служебного слова UNIT и следующего за ним имени.

Имя модуля должно совпадать с именем файла, в котором он хранится (модуль EDIT, файл – edit.pas).

Интерфейсная часть начинается служебным словом INTERFACE, за которым находятся объявления всех глобальных объектов модуля: типов, констант, переменных и подпрограмм. Эти объекты будут доступны всем модулям и программам, вызывающим данный модуль.

Исполняемая часть начинается служебным словом IMPLEMENTATION и содержит описания подпрограмм, объявленных в интерфейсной части. Здесь могут объявляться локальные объекты, которые используются только в интерфейсной части и остаются недоступными программам и модулям, вызывающим данный модуль.

В инициирующей части размещаются исполняемые операторы, содержащие некоторый фрагмент программы. Инициирующая часть может отсутствовать вместе с начинающим ее словом begin.

После написания модуля его следует откомпилировать: отметить пункт Destination Disk в меню Compile. В результате будет создан файл с расширением .tpu. Затем этот модуль можно использовать так же, как любой стандартный.

Для подключения модуля используется служебное слово USES, после чего указывается имя модуля и происходит это сразу же после заголовка программы.

2. Выполните следующие упражнения:

Создать модуль Maximum, содержащий функцию max для поиска максимального из двух целых чисел.

```
unit Maximum;
interface
function max(a, b : LongInt) : LongInt;
implementation
function max(a, b : LongInt) : LongInt;
  begin
    if a < b then
    max := b
    else
    max := a;
  end;
end.
```

Программа, использующая созданный модуль:

```
uses Maximum;
var a,b,y: integer;
begin
   writeln('Введите число a');
   read(a);
   writeln('Введите число b');
   read(b);
   y:=max(a,b);writeln('Максимальное ',y);
end.
```

4. Окончание работы:
1. Сохранить созданные программы.
2. Подготовить ответы на контрольные вопросы.
3. Показать работу преподавателю.
4. Завершить работу TURBO PASCAL.

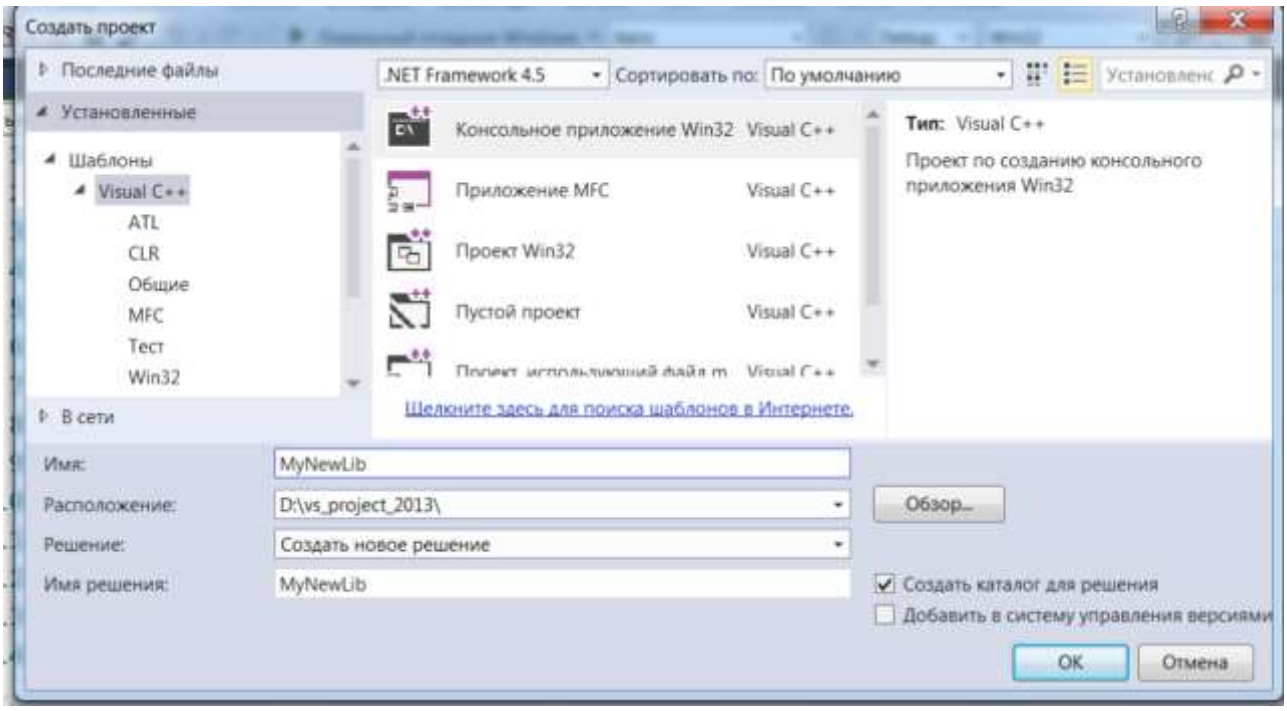
Контрольные вопросы:
1. Какие существуют стандартные модули языка Турбо Паскаль?
2. Какое требование существует к имени модуля?
3. Где должен храниться созданный модуль?

Практическая работа №18. «Использование библиотеки подпрограмм»

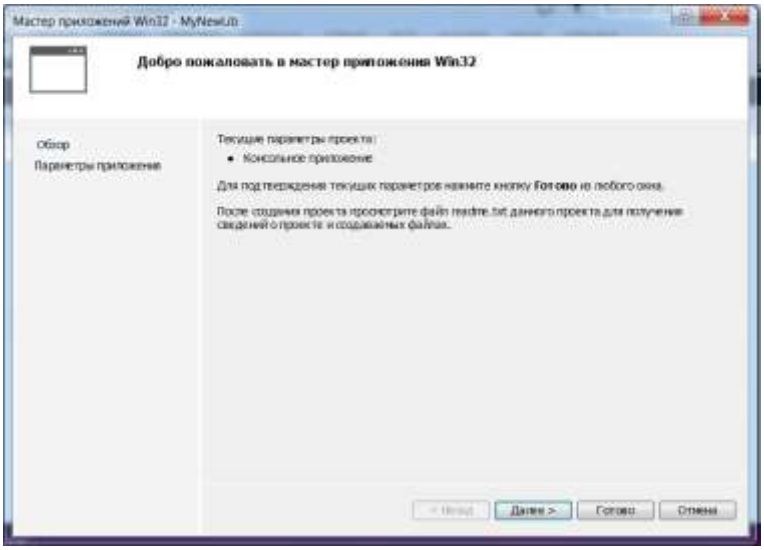
Цель работы: сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с использованием библиотеки подпрограмм **Оборудование, технические и программные средства:** персональный компьютер, среда программирования **Visual Studio**.

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

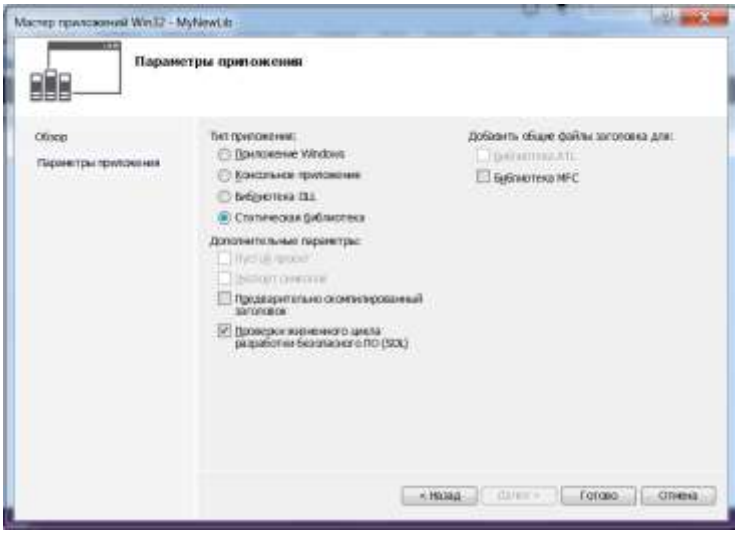
1. Открываем visual studio у меня visual studio 2013
2. Выбираем: “Создать новый проект”, в открывшемся окне выбираем “консольное приложение виндовс 32”, также вводим название проекта MyNewLib и указываем путь где мы сохраним проект.



Дальше нажимаем ок и мы переходим к следующему окошку.



3. В новом окне выбираем “Статическая библиотека” и убираем галочку “предварительно откомпилированные заголовок”, далее нажимаем «готово»



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

4. Дальше в этот проект MyNewLib добавляем два файла один заголовочный function.h:

```
#ifndef FUNCTION_H
#define FUNCTION_H

void func_hellow();
void func_privet();

#endif
```

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 файл библиотеки MyNewLib.lib

Для того, чтобы создавать качественные библиотеки нам нужно каждую функцию записать в отдельный файл, это для того, чтобы в MyNewLib.lib было создано несколько объектных файлов .obj в данном случае мы могли бы файл function.cpp разбить на два файла допустим файл f1.cpp в котором записать определение функции 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 объектный 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 которой каждой функции принадлежит отдельный объектный код, только включаются те функции которые используются, а если все функции в одном .obj то естественно включатся 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:

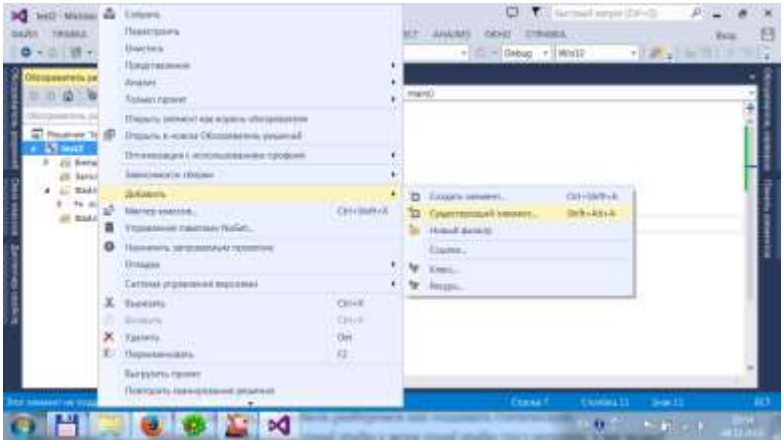
```
#include <iostream>
using std::cout;
using std::endl;

#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

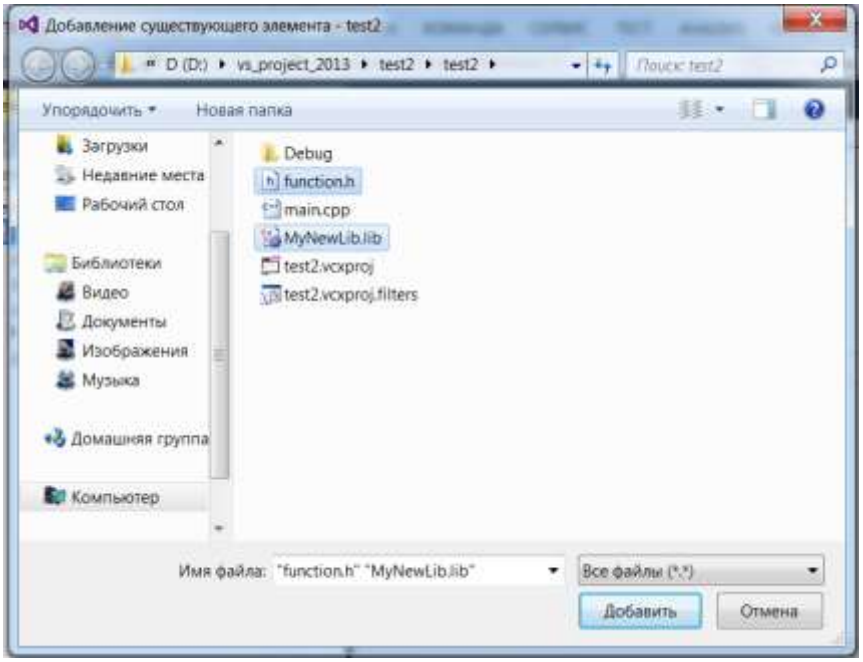
- вызываем функции, (если мы сейчас это все скомпилируем, то у нас вылезет ошибка так как мы реально не добавили файл function.h и не добавили саму библиотеку в проект)

- копируем файлы MyNewLib.lib и файл function.h в папку с файлами нашего проекта – это та папка где находится 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 у вас они должны быть видны из него.

Безымянный

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.

Практическая работа №19.

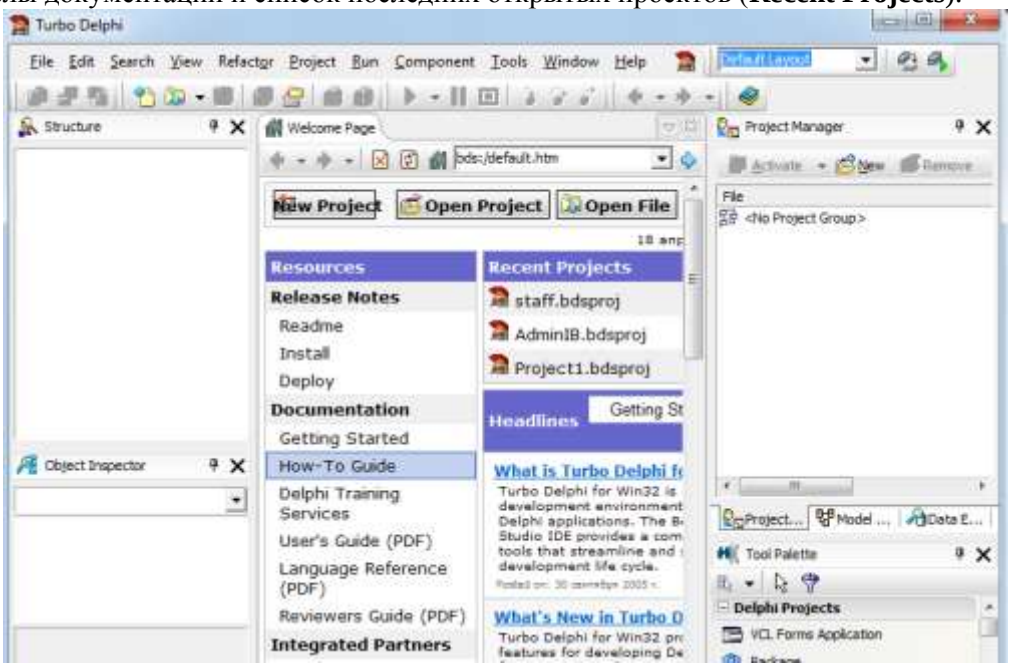
«Изучение интегрированной среды разработчика. Создание простого проекта»

Цель работы: сформировать умения по использованию и настройке интегрированной среды разработчика Delphi, сформировать умения по созданию простейших приложений
Оборудование, технические и программные средства: персональный компьютер, интегрированная среда разработчика **Turbo Delphi**.

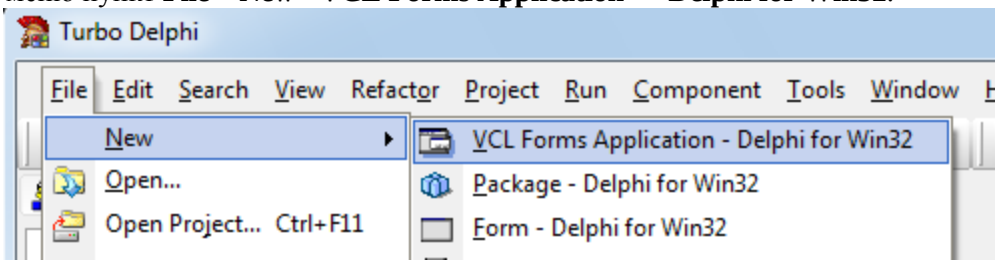
Задание 1. Изучение рабочей среды Turbo Delphi
Методические указания по выполнению задания:

1. Запустите интегрированную среду разработчика **Turbo Delphi**. Для запуска **Turbo Delphi** существует несколько 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.

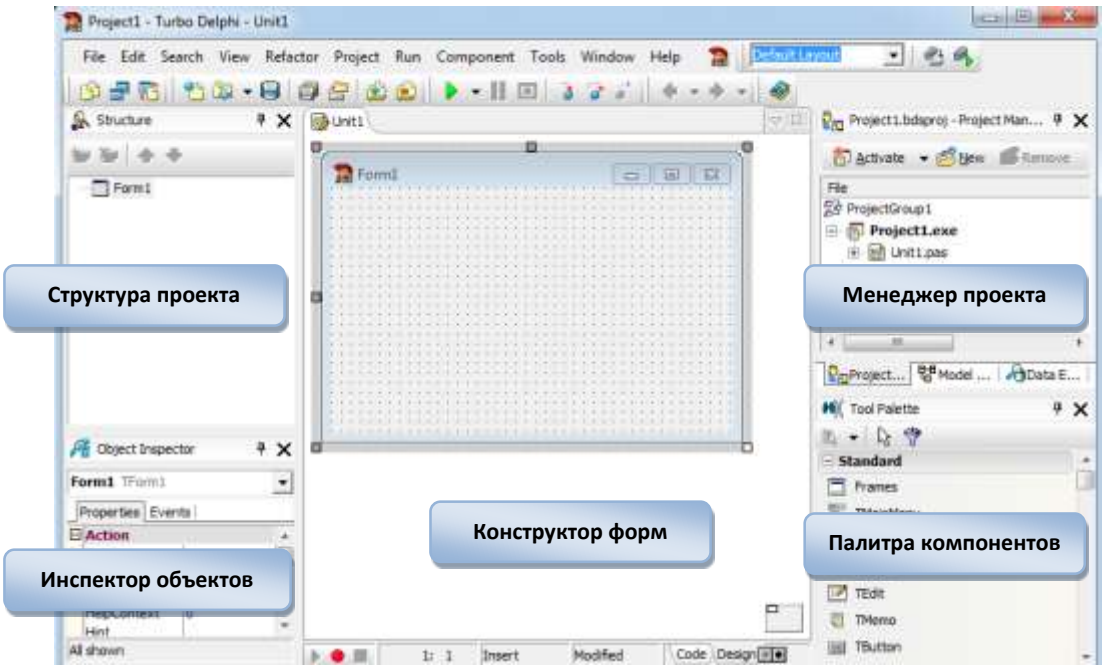
   Turbo Delphi — это среда быстрой разработки от корпорации **Borland**, в которой в качестве языка программирования используется одноименный язык программирования **Delphi**, ранее известный как **Object Pascal**.
2. После запуска появится главное окно среды разработки в центре которого по умолчанию будет открыта HTML-страница **Welcome Page** во встроенном Интернет - браузере. На ней находятся ссылки на разделы документации и список последних открытых проектов (**Recent Projects**).



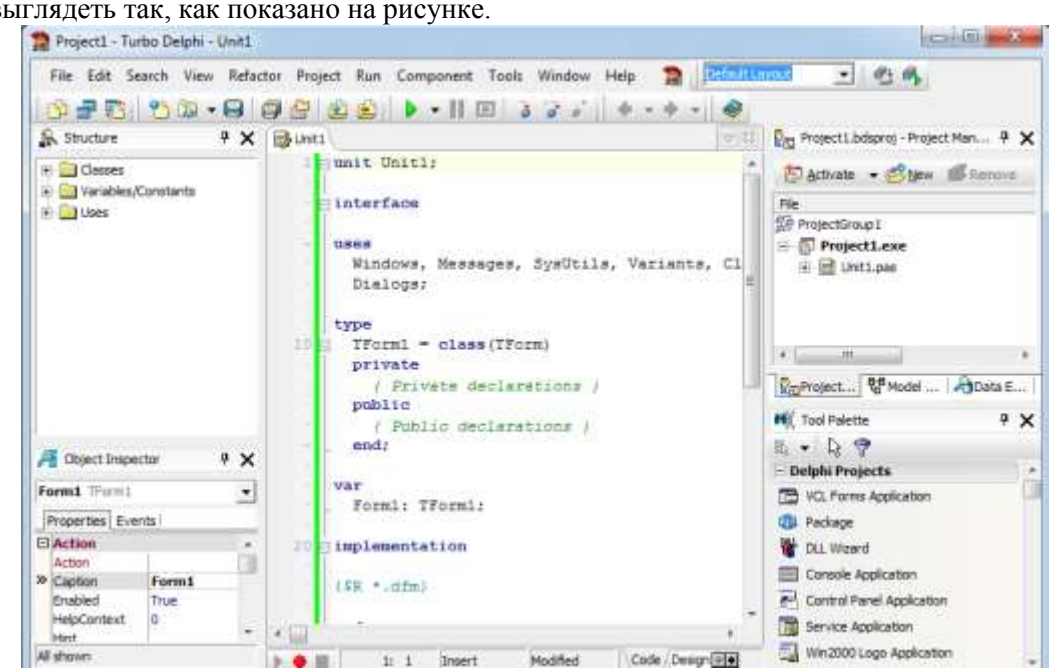
3. Закройте страницу приветствия и создайте новый проект, выбрав в главном меню пункт **File – New - VCL Forms Application — Delphi for Win32**.



4. Среда разработки примет вид, представленный на рисунке. Данное окно содержит несколько частей, с содержимым которых мы сейчас и познакомимся.



5. Центральную часть окна занимает окно **Конструктора форм (Form designer)** приложения. **Формой (Form)** приложения на этапе разработки принято называть окно программы (во время разработки это форма, на этапе выполнения — окно). В окне конструктора форм вы будете формировать внешний вид своего 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).
6. 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** сводится к работе с конструктором форм или редактором кода, при этом посредством конструктора создается внешнее окно приложения, а в редакторе прописывается код программы.

При закрытии конструктора форм или редактора кода автоматически закрывается и приложение.

Вернитесь обратно к окну **Конструктора формы**.

7. Слева от окна конструктора находится окно **Инспектора объектов (Object Inspector)**. Это окно заполнено информацией о выделенном объекте проекта (в данный момент — формы). Окно **Object Inspector** имеет две вкладки — **Properties (Свойства)** и **Events (События)**.

8. 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 клавишу **Enter**, сразу увидите, что ваша форма изменилась. Во время выполнения окно вашего приложения будет иметь введенный вами заголовок.
9. Поначалу группировка свойств может вызвать затруднения (свойств довольно много у любого объекта и 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 выбрать пункт **Arrange - by Name.**
10. Каждый компонент обладает своим собственным набором свойств и 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 компонента вниз (изменять размеры и положение компонентов можно и с помощью мыши).

Логическое свойство **Visible** определяет, будет ли виден данный компонент (результат виден при запуске приложения, а не в режиме 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 запрещает получение управления данному компоненту. Каждый элемент управления может содержать подсказку, появляющуюся в том случае, если указатель мыши находится над 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 приложение. В этом случае при запуске нельзя будет свернуть приложение или изменить его размер.

11. Вторая вкладка окна **Object Inspector — Events** используется для описания событий, на которые будет реагировать выделенный объект (в данный момент им является ваша форма). Именно при выборе необходимого вам события в редакторе кода появится заготовка процедуры обработки, где надо записывать код программы.
12. 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 файлах проекта.
13. **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 вы обычно видите на окнах в операционной системы **Windows** (такие компоненты называют визуальными — **Visual**). Кроме того, это могут быть также и невидимые (не визуальные) компоненты, т. е. те, которые не отображаются в момент выполнения программы, но выполняют различные функции. Типичный пример такого компонента — **таймер (Timer).**

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

14. Окно **Структуры проекта** после создания проекта отображается в левой верхней части экрана. Оно содержит информацию о структуре исходного кода программы (именно поэтому не содержит информации, если активно окно дизайнера формы). Для того чтобы посмотреть данную информацию, следует переключиться в окно редактора кода. После активизации окна редактора кода окно **Structure** заполнится информацией.

Задание 2. Создание простого проекта

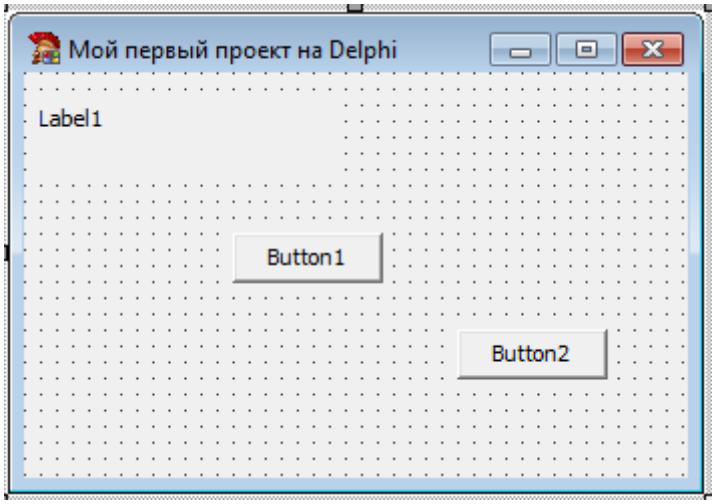
Разработать программу **Hello**, которая при нажатии на кнопку **«Hello»** выводит сообщение **«Hello, мир Windows and Delphi!»**, при нажатии на кнопку **«Выход»** программа завершит работу.

Методические указания по выполнению задания:

1. **Проект (Project)** — совокупность файлов, используемые средой разработки (точнее говоря, компилятором **Turbo Delphi**) для итоговой генерации программы. Проект в простейшем случае представляет собой совокупность следующих файлов:
   - **файл описания проекта (bdsproj-файл)** — файл специального формата, в котором записана общая информация 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 выполнении 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, рекомендуется сохранять их в заранее подготовленном каталоге. Создайте в папке своей группы новую папку и назовите её **Hello**.

2. Сохранение всех файлов проекта осуществляется через пункт главного меню **File - Save all (сохранить все)**. Первый раз при сохранении потребуется сохранить два файла: модуль формы и файл проекта. Сохраните проект, выполнив команду **File - Save Project as...** В открывшемся окне найдите свою новую папку (**Hello**). В этом же окне **Delphi** предложит сохранить (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**
3. 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 **.pas** (в нашем примере **Unit1.pas**)
4. Разместите на форме компоненты в соответствии с рисунком: **Label1, Button1, Button2** вкладки палитры компонентов **Standard**.



5. Выделите кнопку **Button2**, перейдите в **Object Inspector** на вкладку **Properties**, найдите свойство **Caption (Заголовок)** и измените его на **Выход**.
6. Сохраните проект, нажав на соответствующую кнопку на панели инструментов.
7. Перейдите на правую страницу **Object Inspector**, на страницу **Events**, и найдите событие **OnClick**, справа от него дважды щелкните мышью.
8. Из листинга видно, что в рамках формы класса **TForm1** описываются все присутствующие 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**.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Button1: TButton;
    Button2: TButton;
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button2Click(Sender: TObject);
begin
Close;
end;
```

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 **OnCreate** происходит в момент 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 перерисована на экране.

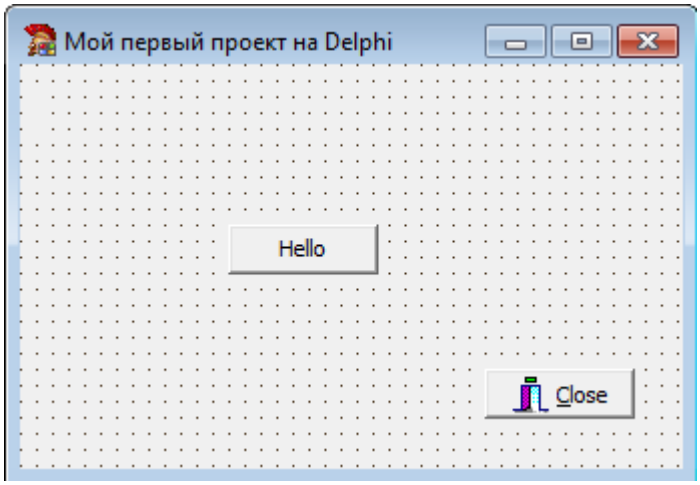
Существуют события, которые формируют сами элементы управления, например: **OnEnter** событие, которое появляется, когда элемент управления становится активным, и **OnExit** событие, возникающее, когда элемент управления перестает быть активным. Данные два события существуют только у элементов управления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

9. Сохраните проект.
10. Запустите проект. Запустить проект на исполнение можно либо нажатием кнопки **F9**, либо через главное меню: **Run - Run**, либо нажатием соответствующей кнопки на панели инструментов:  . При запуске формируется стандартное окно **Windows**, которое отображается соответствующей кнопкой на панели задач. Для дальнейшей доработки исходного файла необходимо 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.

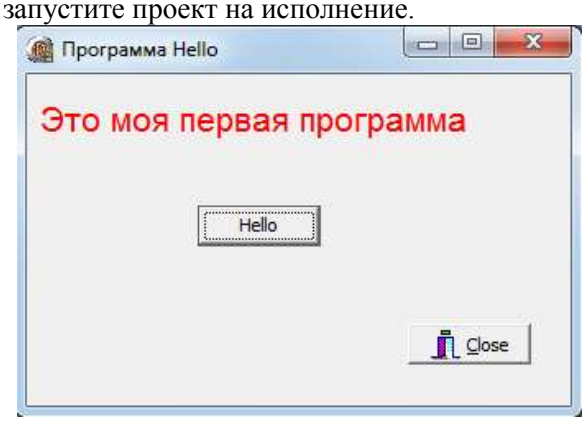
11. Выделите кнопку **Button1**, найдите в **Object Inspector** свойство **Caption** и вставьте новое название кнопки **Hello**.
12. Перейдите в **Object Inspector** на страницу **Events**, найдите событие **OnClick** и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре кнопки **Button1**, напишите следующее выражение:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Label1.Caption := 'Hello, мир Windows and Delphi';
end;
```

13. 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 закроется.
14. Измените содержание выводимой на экран реплики на «**Это моя первая программа**». Для этого дважды щелкните по кнопке **Hello** и измените соответствующий оператор.
15. Задайте шрифт и размер выводимой реплики следующим образом: размер – **16**, цвет – **красный**, гарнитура шрифта - **Arial**. Для этого измените свойство **Font** в **Object Inspector**.
16. Замените простую кнопку **Выход** на **Close**. Для замены кнопки надо удалить существующую, 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**) автоматически выполняет сою функцию по закрытию окна.



17. 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 по этому файлу – файл, как и положено файлу с расширением **.exe**, запустится. Вы увидите, что ваш проект работает в **Windows** без запуска среды **Delphi**.
18. Озаглавьте окно проекта «**Программа Hello**». Для этого выделите форму **Form1**, перейдите в **Object Inspector**, найдите свойство **Caption** и замените надпись на **Программа Hello**.
19. Сохраните изменения и запустите проект на исполнение.



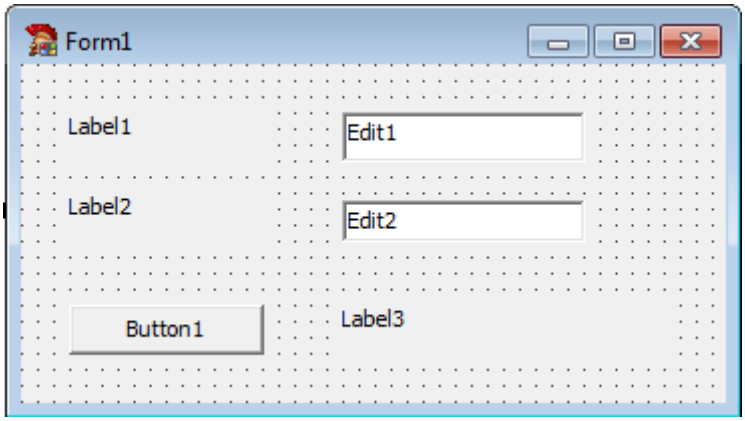
Задание 3. Создание простого проекта

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.

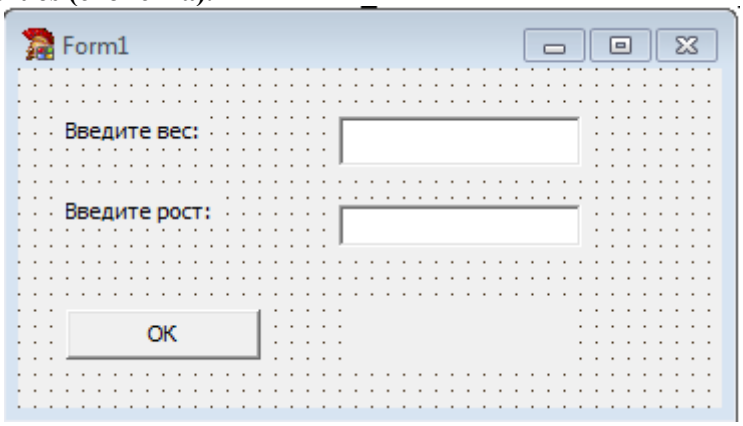
Если фактический вес человека меньше оптимального, то человек худой, и наоборот, если больше, то нужно похудеть.

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.



4. Задайте для элементов управления значение свойства **Caption** в соответствии с рисунком. 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).**



5. 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 разницы между оптимальным весом и фактическим (**Delta**). Пусть значения этих переменных будут округленными, в этом случае тип этих переменных можно объявить как **Integer**.

```
var
  Form1: TForm1;
  faktW, optW, rost, Delta: integer;
```

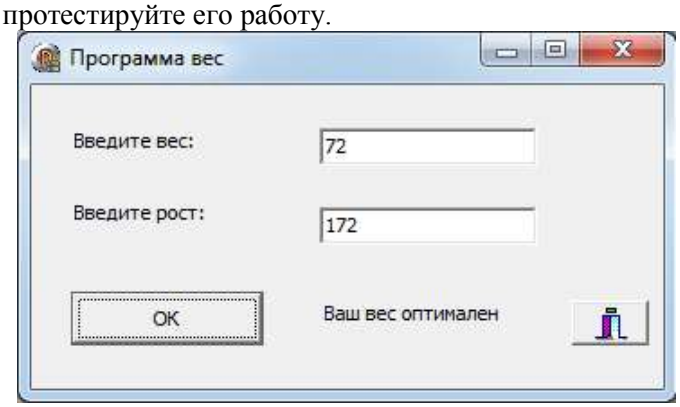
6. Выделите кнопку **Button1**. Перейдите в **Object Inspector**, на страницу **Events (события)**, и найдите событие **OnClick**, справа от него дважды щелкните мышью. Оказавшись в коде программы, а точнее, в заготовке процедуры кнопки **Button1** заполните её следующим кодом:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
faktW:= StrToInt(Edit1.text);
Rost:= StrToInt(Edit2.Text);
OptW:=Rost - 100;
Delta:= abs (faktW - OptW);
if OptW = faktW then Label3.caption := 'Ваш вес оптимален'
else
if OptW > faktW then Label3.caption := 'Вам надо поправиться на'
+IntToStr(Delta)+' кг.'
else
Label3.caption := 'Вам надо похудеть на '+IntToStr(Delta)+' кг.'

end;
```

7. Сохраните проект и протестируйте работу приложения.
8. Усовершенствуем программу так, чтобы можно было бы вводить десятичные величины. Для этого в разделе **Var** зададим тип переменных не **Integer**, а **Real**.
9. Функция **FloatToStr( )** преобразует помещенную в скобки переменную типа **Real** в переменную строкового типа, а функция **StrToFloat()** превращает строку в дробное или действительное число, число с плавающей точкой. Замените в тексте программного кода **IntToStr** на **FloatToStr**, а **StrToInt** на **StrToFloat**.
10. Сохраните проект и протестируйте работу приложения.
11. Для форматирования и преобразования действительных чисел вместо функции **FloatToStr()**. Применим функцию **Format(‘ %f ’,[переменная])**: **Format (‘%f ‘, [Delta]).**

```
procedure TForm1.Button1Click(Sender: TObject);
begin
faktW:= StrToFloat(Edit1.text);
Rost:= StrToFloat(Edit2.Text);
OptW:=Rost - 100;
Delta:= abs (faktW - OptW);
if OptW = faktW then Label3.caption := 'Ваш вес оптимален'
else
if OptW > faktW then Label3.caption := 'Вам надо поправиться на'
+Format ('%f ', [Delta])+' кг.'
else
Label3.caption := 'Вам надо похудеть на '+Format ('%f ', [Delta])+' кг.'
```

12. Сохраните проект и протестируйте работу приложения.
13. Озаглавьте окно проекта **Программа Вес**. Создайте кнопку **Close**.
14. Сохраните проект и протестируйте его работу.



Практическая работа №20.

«Создание проекта с использованием кнопочных компонентов»

Цель работы: сформировать навыки разработки приложений с использованием кнопочных компонентов в среде визуального программирования Delphi, изучить особенности их использования

Оборудование, технические и программные средства: персональный компьютер, интегрированная среда разработчика **Turbo Delphi**.

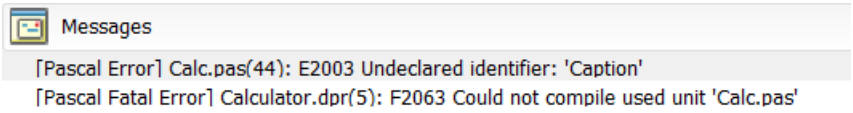
Задание 1.

Используя кнопочные компоненты **TSpeedButton,** разработать программу – калькулятор, выполняющий простейшие действия.

Методические указания по выполнению задания:

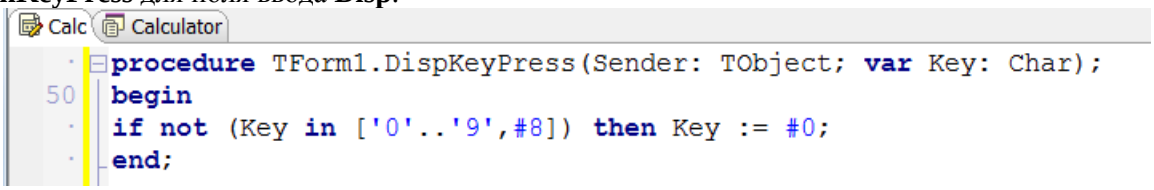
1. Запустите интегрированную среду разработчика **Turbo Delphi**. Создайте новый проект.
2. Создайте в папке своей группы новую папку и назовите её **Калькулятор**. Сохраните проект в созданную ранее папку, выполнив команду **File - Save Project as...** под именем **Calc.bdsproj.**
3. Задайте для формы 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 вообще их можно привести в действие только мышкой. Таким образом, на форме будет всего один компонент, принимающий сигналы с клавиатуры — это поле ввода.
6. 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.

7. Дайте кнопкам-действиям имена **btnPlus**, **btnMinus**, **btnMul**, **btnDiv** (сложение, вычитание, умножение и деление), а кнопке «равно» — имя **btnCalc**.
8. Для действий и кнопки **C** установите жирный шрифт, а для кнопки **C** дополнительно — красный цвет шрифта.
9. Кнопка **C** должна просто стирать содержимое поля ввода **Disp**. То есть, нужно вызвать метод **Disp.Clear**. Добавьте обработчик события **OnClick** для кнопки **C**.
10. Понятно, что когда пользователь щелкнул по кнопке-цифре, нужно добавить эту цифру в конец текста поля 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 единственный 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 них общий обработчик события **onClick**: **Disp.Text := Disp.Text + Sender.Caption;**
11. 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 процедуры как 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, а как **TSpeedButton**: **Disp.Text := Disp.Text + TSpeedButton(Sender).Caption;**

12. Исправьте код и запустите программу. Проверьте, как она работает при вводе текста и букв с клавиатуры и с помощью мыши. Что вам не понравилось?
13. 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 обработчика 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.

14. 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** есть свойства **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 установить это свойство равным длине текста, курсор встанет сразу за последним символом.
15. Добавьте в конец обработчика события **OnClick** код: **Disp.SelStart := Length(Disp.Text);** Проверьте работу программы.
16. Далее необходимо организовать вычисления. Нам нужны две переменных для хранения чисел и одна символьная переменная, в которую будем записывать тип операции. Поскольку 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 вещественных переменные **x1** и **x2** типа **Double** и одну символьную строку **oper**.
17. 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-операций один обработчик. Выделите все кнопки-операции и создайте для них один обработчик события **OnClick**:

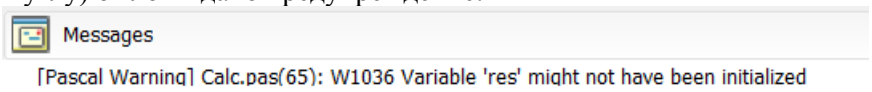
```
x1 := StrToFloat(Disp.Text);
oper := TSpeedButton(Sender).Caption;
```

18. При нажатии на кнопку «**Равно**» нужно прочитать из поля ввода второе число и выполнить операцию. При этом первое число и тип операции уже должны находиться в переменных **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**»:

```
procedure TForm1.btnCalcClick(Sender: TObject);
var res: double;
begin
 x2 := StrToFloat(Disp.Text);
 if oper = '+' then res := x1 + x2;
 if oper = '-' then res := x1 - x2;
 if oper = '*' then res := x1 * x2;
 if oper = '/' then res := x1 / x2;
 Disp.Text := FloatToStr(res);
end;
```

19. Запустите программу и проверьте ее работу. Учтите, что для ввода второго числа нужно сначала очистить экран кнопкой «**C**» (позже мы исправим это неудобство).
20. Закройте окно программы и затем запустите ее заново. Введите какое-нибудь число, а затем сразу щелкните по кнопке «равно». Что получилось? Попытайтесь объяснить этот эффект, учитывая, что при создании глобальной символьной переменной **oper** в нее записана пустая строка.
21. Наверняка вы догадались, что проблема вызвана тем, что мы еще не задали операцию, а уже попытались что-то вычислить. При этом ни один условный оператор не сработал, и значение переменной **res** не изменялось. А что же в ней было? Откуда взялось это число? Переменная **res** объявлена в процедуре, то есть, она локальная. Память для локальных переменных выделяется в стеке при каждом новом вызове процедуры, 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»**. Формально 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, мы просто выйдем из процедуры с помощью оператора **Exit.**

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

```
if oper = '' then Exit;
```

Проверьте работу программы. Что вам не нравится?

23. Конечно, очень неудобно, что перед вводом второго числа нужно очищать поле ввода, нажимая на кнопку «**C**». Хотелось бы делать это автоматически.
24. Запустите стандартную программу **Калькулятор** и посмотрите, в какой момент стирается 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 начинать вводить новое число. Объявите глобальную логическую переменную **newNumber.**
25. В конце обработчиков события **OnClick** для кнопок-действий и кнопки «равно» добавьте строку: **newNumber := True;**
26. Очистку экрана будем делать перед вводом нового числа. Дополните обработчики кнопок-цифр. В начало обработчика события **OnClick** для кнопок-цифр добавьте код:

```
if newNumber then begin
    Disp.Clear;
    newNumber := False;
  end;
```

27. Учтем в нашей программе, что пользователь может набирать число на клавиатуре. Для этого нужно дополнить аналогичным кодом обработчик события **OnKeyPress** для поля ввода. Измените обработчик события **OnKeyPress** так, как показано ниже.

28. Запустите программу и проверьте ее. Можно ли ввести знаки сложения, вычитания, умножения и деления с клавиатуры? Выполняется ли действие при нажатии на клавишу **Enter**?

```
if not (Key in ['0'..'9',#8]) then Key := #0
else
  if newNumber then begin
    Disp.Clear;
    newNumber := False;
  end;
```

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

Обработка нажатия других клавишей строится аналогично. При нажатии клавиши **Enter**, имеющей код 13 (он обозначается в программе как #13), нужно вызвать обработчик события **OnClick** для кнопки «равно» (она должна называться **btnCalc**).

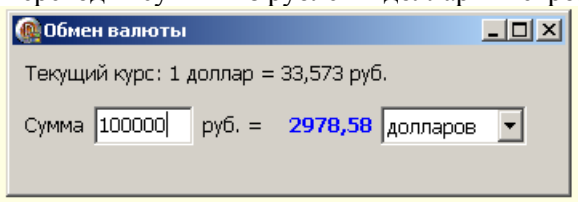
31. Измените код обработчика **OnKeyPress** следующим образом:

```
procedure TForm1.DispKeyPress(Sender: TObject; var Key: Char);
begin
if not (Key in ['0'..'9',#8]) then begin
    case Key of
     '+': btnMinusClick(btnPlus);
     '-': btnMinusClick(btnMinus);
     '*': btnMinusClick(btnMul);
     '/': btnMinusClick(btnDiv);
     #13: btnCalcClick(btnCalc);
    end;
    Key := #0;
  end
  else
  if newNumber then begin
      Disp.Clear;
      newNumber := False;
    end;
Disp.SelStart := Length(Disp.Text);
newNumber := True;
end;
```

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

Контрольное задание:

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



Практическая работа №21.

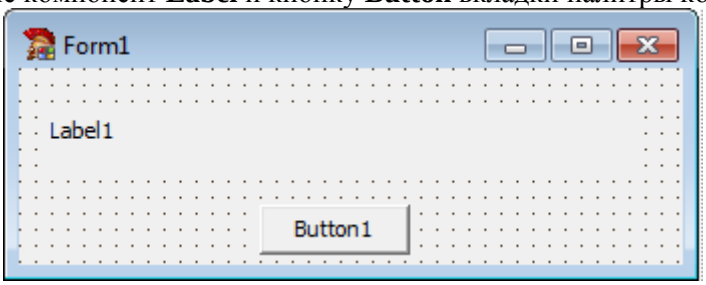
«Создание проекта с использованием компонентов для работы с текстом»

Цель работы: сформировать умения по использованию компонентов для работы с текстом в Delphi, сформировать умения по созданию приложений с компонентами для работы с текстом в Delphi.
Оборудование, технические и программные средства: персональный компьютер, интегрированная среда разработчика **Turbo Delphi**.

Задание 1.
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 сообщение исчезает. При повторном выводе размер надписи должен увеличиваться.

Методические указания по выполнению задания:

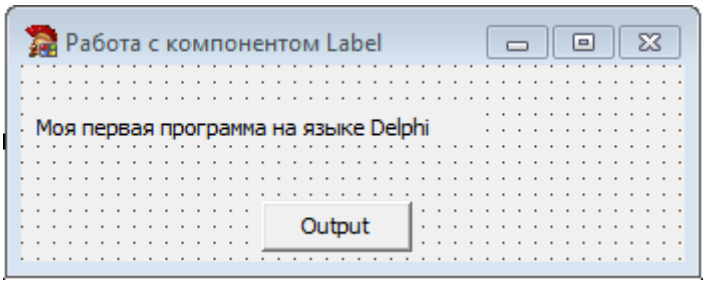
1. Запустите интегрированную среду разработчика **Turbo Delphi**. Создайте новый проект.
2. Создайте в папке своей группы новую папку и назовите её **Label**. Сохраните проект в созданную ранее папку, выполнив команду **File - Save Project as...**
3. Разместите на форме компонент **Label** и кнопку **Button**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



Компоненты ввода — вывода данных можно условно разделить на несколько различных блоков: 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 и другие компоненты. Текст, который будет отображен, можно задавать как на этапе разработки формы, так и в процессе выполнения программы, присвоив значение свойству **Caption**.

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



7. 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:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Label1.Visible:=not label1.Visible;
if Label1.Visible then
 Label1.Font.Size:=Label1.Font.Size+1;
end;
```

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

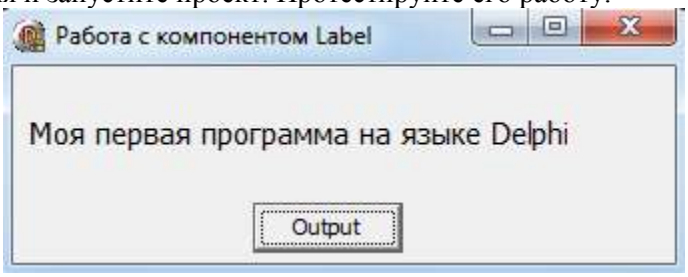
Для вывода определенной информации на экран, кроме уже описанного компонента **Label**, может быть также использован компонент **Panel** с той же самой вкладки или **StaticText** со страницы **Additional**. Они имеют незначительные отличия от компонента **Label.**

Остальные компоненты позволяют вводить и редактировать информацию, включая возможность выделения, 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 выделенный фрагмент, **CutToClipboard** — удалить из текста выделенный фрагмент и поместить его в буфер, **PasteFromClipboard** — копировать текст из буфера в то место редактора, где в данный момент находится курсор.

Для всех редакторов определено дополнительное событие **OnChange**, возникающее, когда изменяется 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 либо вводится, 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.

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

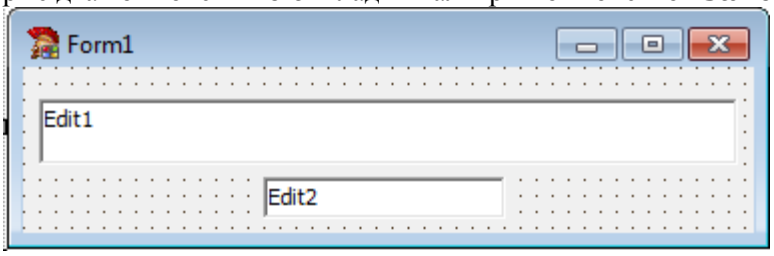


Задание 2.

Разработать программу, которая при вводе текста в первый компонент **Edit1**, во втором компоненте **Edit2** отображает реальную длину вводимой строки. Кроме этого, при выходе из компонента **Edit1** его содержимое копируется в буфер обмена и удаляется, а при возвращении 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 ранее папку, выполнив команду **File - Save Project as...**
2. Разместите на форме два компонент **Edit** вкладки палитры компонентов **Standard**.



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

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
Edit2.Text:= IntToStr(Edit1.GetTextLen);
end;
```

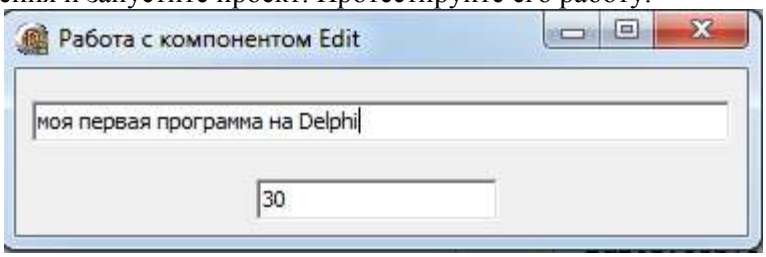
6. Выделите компонент **Edit1**, в **Object Inspector** перейдите на вкладку **Events** и найдите событие **OnEnter** и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре текстового поля **Edit1**, напишите следующий программный код:

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
Edit1.PasteFromClipboard;
end;
```

7. 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:

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
 Edit1.SelectAll;
 Edit1.CopyToClipboard;
 Edit1.Clear;

end;
```

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

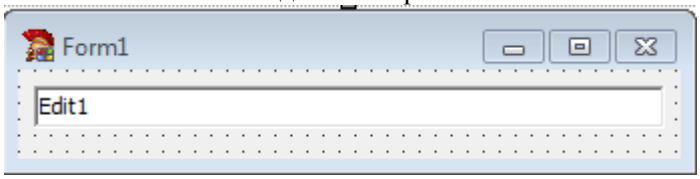


Задание 3.

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 ранее папку, выполнив команду **File - Save Project as...**
2. Разместите на форме компонент **Edit** вкладки палитры компонентов **Standard**.



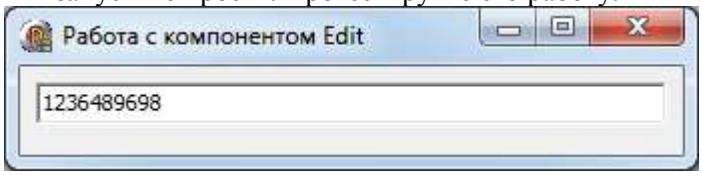
3. Перейдите в код программы (на клавиатуре нажмите клавишу **F12**). Введите в раздел **VAR** глобальную переменную **ch** типа **char**, в которой будет храниться последний нажатый символ.

```
var
  Form1: TForm1;
  ch: char;
```

4. Задайте для формы заголовок **«Работа с компонентом Edit»**.
5. Выделите текстовое поле **Edit1**, найдите в **Object Inspector** свойство **Text** и оставьте его пустым.
6. 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**.

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
if ch=key then key:=#0
 else ch:=key;
end;
```

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

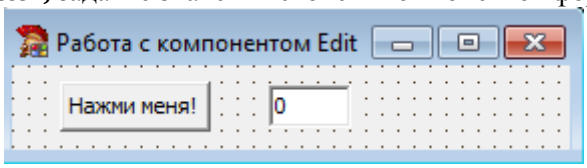


Задание 4.

Разработать программу, которая считает количество нажатий на кнопку и выдает это значение в компоненте **Edit**.

Методические указания по выполнению задания:

1. Создайте в папке своей группы новую папку и назовите её **Edit3**. Сохраните проект в созданную ранее папку, выполнив команду **File - Save Project as...**
2. Разместите на форме компонент **Edit** и кнопку **Button** вкладки палитры компонентов **Standard**.
3. Используя **Object Inspector,** задайте значения свойств компонентов формы в соответствии с рисунком.



4. Если в целочисленной переменной **i** будем считать количество нажатий, то процедура обработки события **OnClick** кнопки может быть записана в виде:

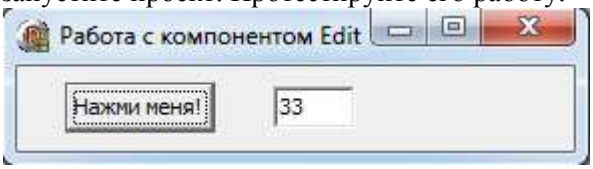
```
procedure TForm1.Button1Click(Sender: TObject);
begin
i:=i+1;
 Edit1.Text:=IntToStr(i);
end;
```

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

5. Перейдите в код программы (на клавиатуре нажмите клавишу **F12**). Введите в раздел **VAR** глобальную переменную **i** типа **integer**, в которой будет храниться последний нажатый символ.
6. Выделите форму, в **Object Inspector** перейдите на вкладку **Events** и найдите событие **OnCreate** и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре формы **Form1**, напишите следующий программный код:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
i:=0;
end;
```

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



9. 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 остальные описываются в виде полей. В этом случае описание формы будет иметь вид:

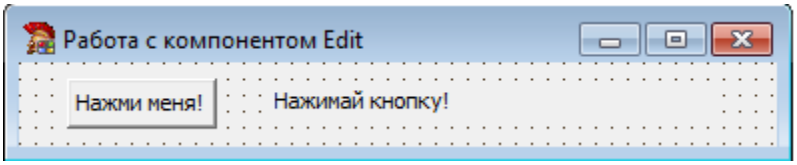
```
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    i: integer;
  end;

var
  Form1: TForm1;
```

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 непосредственно.

10. Внесите изменения в программный код в соответствии с рисунком. Сохраните проект и протестируйте работу приложения.
11. Данную программу можно легко модифицировать так, чтобы после определенного количества нажатий появлялось некоторое сообщение или кнопка блокировалась, или приложение автоматически закрывалось. Результат можно отображать не только посредством компонента **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 отражаться 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 следующем 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.

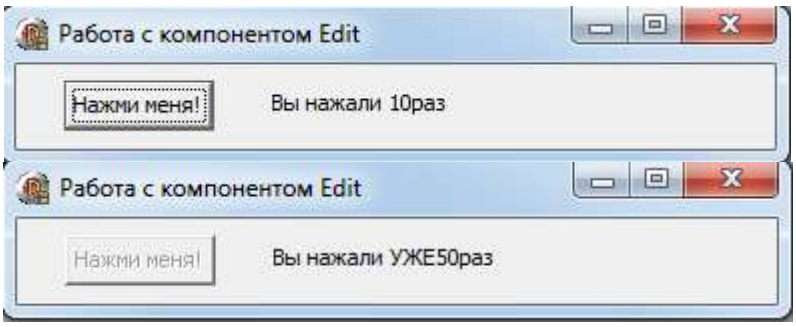
12. Разместите на форме компонент **Label**. Задайте свойство **Caption** равным «**Нажимай кнопку!**». Удалите с формы компонент **Edit1**.



13. Перейдите в окно редактирования кода и внесите изменения в код процедуры обработки события **OnClick** кнопки.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Label1.Visible:=False;
 i:=i+1;
if (i=10)or(i=20)or(i=30)or(i=40) then
 Begin
 Label1.Visible:=True;
 Label1.Caption:='Вы нажали '+intToStr(i)+'раз';
 end;
if i=50 then
 Begin
 Label1.Visible:=True;
 Label1.Caption:='Вы нажали УЖЕ'+intToStr(i)+'раз';
 Button1.Enabled:=False;
 end;
end;
```

14. Сохраните проект и протестируйте работу приложения.



Задание 5.

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

Методические указания по выполнению задания:

1. Создайте в папке своей группы новую папку и назовите её **Memo1**. Сохраните проект в созданную ранее папку, 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**, однако 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 свойство, определяющее количество элементов 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 нумерация 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 указывают на четвертую строку в компоненте **Memo1**.

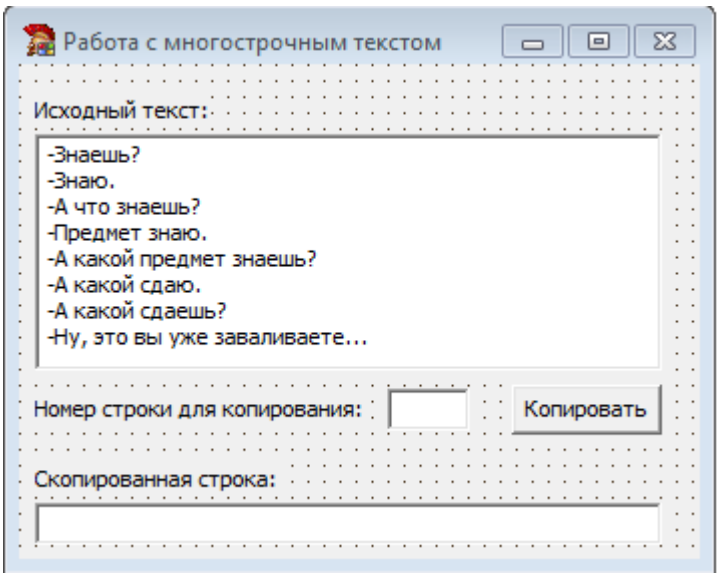
Класс **TString** обладает также рядом методов, среди которых отметим следующие: **Add(St:String):integer** добавляет строку **St** и возвращает номер этой строки; **Delete(Index:Integer)** удаляет строку с номером **Index; Insert (Index:Integer, St:String)** вставляет строку с номером **Index, Clear**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

Все содержимое компонента можно записать в файл с помощью метода **SaveToFile** или прочитать из файла посредством методом **LoadFromFile**. Аналогичным образом можно поступить и с потоком, направив в него весь файл, или прочитать файл из потока.

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

Компонент **RichEdit** обладает всеми характеристиками, присущими компоненту **Memo**, однако 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.

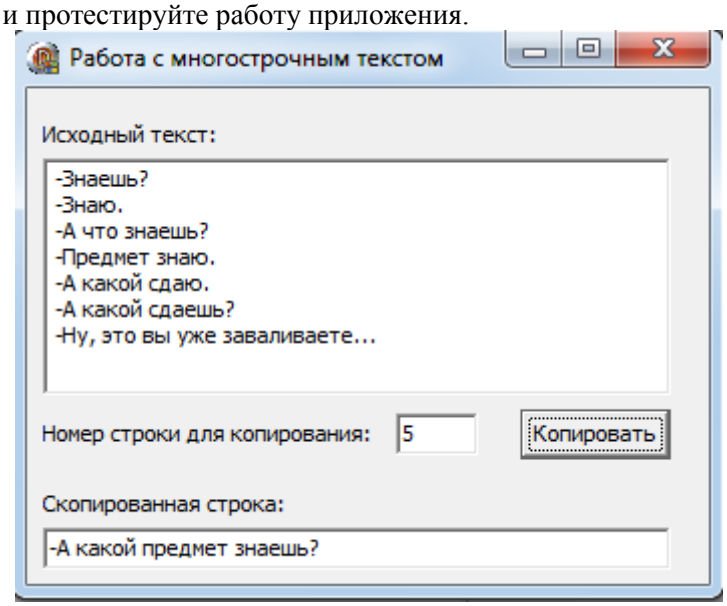
3. Разместите на форме компоненты **Edit,** компонент **Memo,** компонент **Label**, кнопку **Button** вкладки палитры компонентов **Standard**.
4. Используя **Object Inspector,** задайте значения свойств компонентов формы в соответствии с рисунком.



5. Для реализации решения задачи процедура обработки события **OnClick** кнопки может быть записана в виде:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Edit2.Text:=memo1.Lines[strToInt(edit1.Text)-1];
Memo1.Lines.Delete(strToInt(edit1.Text)-1);
end;
```

6. Сохраните проект и протестируйте работу приложения.

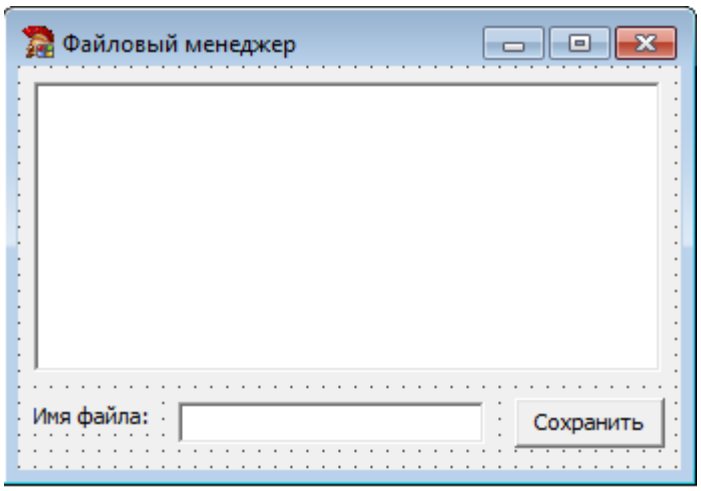


Задание 6.

Разработать программу, которая сохраняет текст, набранный в поле **Memo** в файл, имя которого задано в текстовом поле **Edit**.

Методические указания по выполнению задания:

1. Создайте в папке своей группы новую папку и назовите её **Memo2**. Сохраните проект в созданную ранее папку, выполнив команду **File - Save Project as...**
2. Разместите на форме компонент **Edit,** компонент **Memo,** компонент **Label**, кнопку **Button** вкладки палитры компонентов **Standard**.
3. Используя **Object Inspector,** задайте значения свойств компонентов формы в соответствии с рисунком.



4. Процедура обработки события **OnClick** кнопки **Button** будет состоять из одной строки, и соответственно, листинг будет иметь следующий вид:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Memo1.Lines.SaveToFile(Edit1.Text);
end;
```

5. Сохраните проект и протестируйте работу приложения.

Практическая работа №22. Создание проекта с использованием компонентов ввода и отображения чисел, 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 особенности их использования

Оборудование, технические и программные средства: персональный компьютер, интегрированная среда разработчика **Turbo Delphi**.

Задание 1.

Разработать программу, демонстрирующую действие процедур и функций, оперирующих с системными значениями даты и времени.

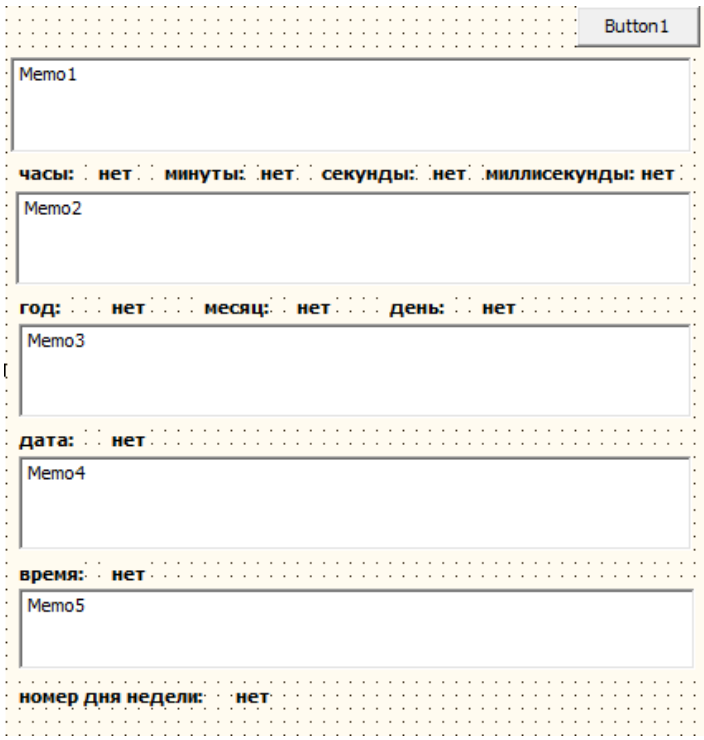
Методические указания по выполнению задания:

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 созданную ранее папку, 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

4. Разместить на форме компоненты в соответствии с рисунком 2.

5. Задайте для элементов управления **Label** значение свойства **Caption** в соответствии с рисунком. Для этого выделите элемент управления и перейдите в **Object Inspector** на страницу **Properties** (свойства).



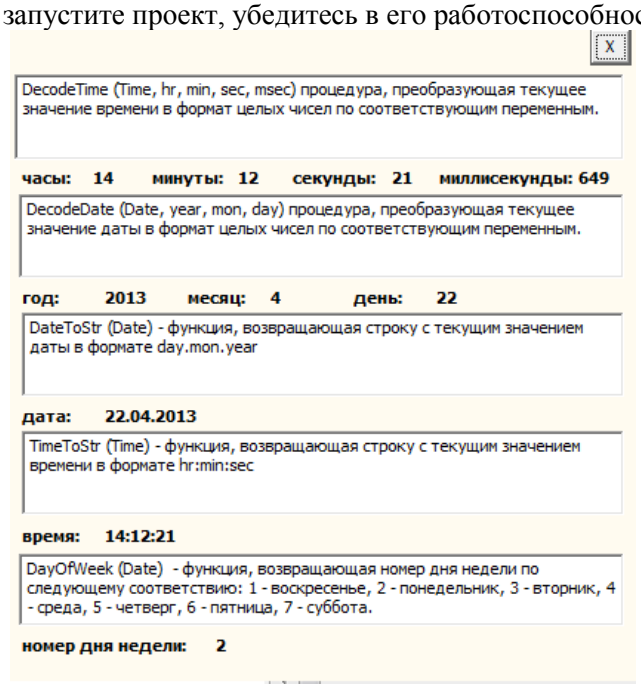
6. Задайте для элементов управления **Memo** значение свойства **Lines** в соответствии с таблицей. Для этого выделите элемент управления и перейдите в **Object Inspector** на страницу **Properties** (свойства).

	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 значение 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 следующему соответствию: 1 – воскресенье, 2 – понедельник, 3 – вторник, 4 – среда, 5 – четверг, 6 – пятница, 7 – суббота.

7. Задайте для элемента управления **Button1** значение свойства **Caption** равным **X.**
8. Введите в окне кода процедуру для отображения даты и времени календаря компьютера.

```
procedure TForm1.FormCreate(Sender: TObject);
var hr,min,sec,msec:word;
    year,mon,day:word;

begin
   DecodeTime(Time, hr, min, sec, msec);
      Label2.Caption:=IntToStr(hr);
      Label4.Caption:=IntToStr(min);
      Label6.Caption:=IntToStr(sec);
      Label8.Caption:=IntToStr(msec);

   DecodeDate(Date, year, mon, day);
      Label10.Caption:=IntToStr(year);
      Label12.Caption:=IntToStr(mon);
      Label14.Caption:=IntToStr(day);

   Label16.Caption:=DateToStr(Date);
   Label18.Caption:=TimeToStr(Time);
   Label20.Caption:=IntToStr(DayOfWeek(Date));

end;
```

9. Запишите для командной кнопки **Button1** процедуру, которая по щелчку на эту кнопку закрывает форму.
10. Сохраните изменения и запустите проект, убедитесь в его работоспособности.



Задание 2.

Разработать приложение, с помощью которого пользователь в одностраничном блокноте выбирает одну из представленных закладок. На рабочем поле блокнота высвечивается соответствующая 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:

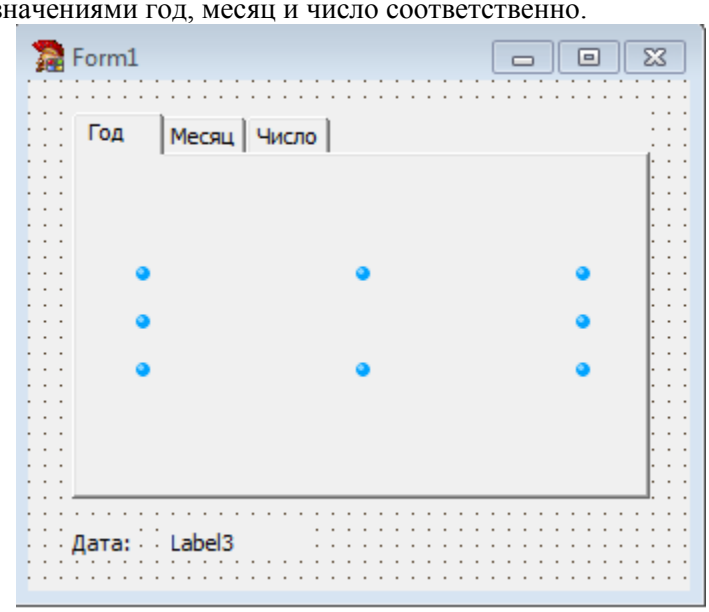
1. В **Delphi** на странице **Win32** Палитры компонентов расположены две составляющие, работающие на программирование элементов с закладками: **TabControl** – одностраничный блокнот; и **PageControl** – многостраничный блокнот. Данные компоненты являются контейнерами и могут содержать в себе другие 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 названия закладок блокнота.
Свойство **TabIndex** указывает текущую закладку в массиве **Tabs**. Программист может использовать данное свойство для переключения на нужную закладку блокнота. Нумерация начинается 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.

2. Создайте новый проект.
3. Создайте в папке своей группы новую папку и назовите её **Date2**. Сохраните проект в созданную ранее папку, выполнив команду **File - Save Project as...**
4. Разместите на форме компоненты **Label**, компонент **TabControl**. В **Label1** будет размещаться надпись блокнота, в **Label2**, **Label3** отображаются текущая дата календаря компьютера.
5. Задайте для элементов управления значение свойства **Caption** в соответствии с рисунком. Для этого выделите элемент управления и перейдите в **Object Inspector** на страницу **Properties** (свойства).
6. 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** введите 3 строки со значениями год, месяц и число соответственно.



7. Перейдите в окно редактирования кода и опишите в разделе **VAR** переменные **y, m, d** типа **word** как глобальные переменные модуля для хранения значений даты календаря компьютера.

```
var
  Form1: TForm1;
  y, m, d: word;
```

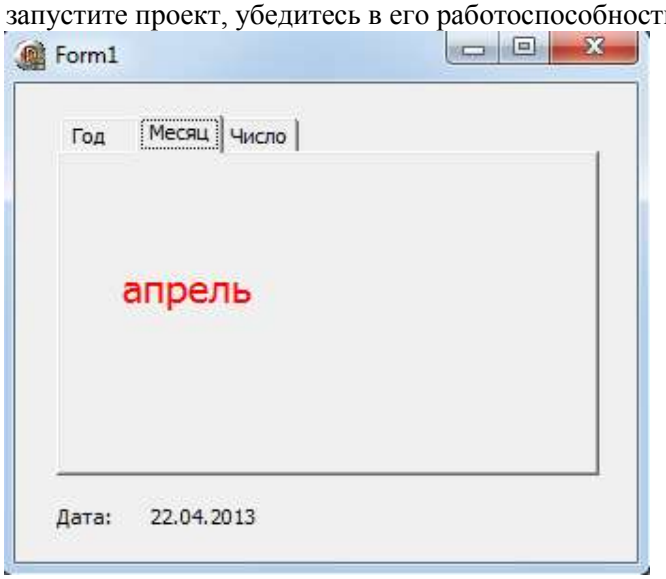
8. Введите в окне редактора кода процедуру для отображения значений даты календаря компьютера:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Label3.Caption:= DateToStr(Date); //отображение текущей даты
 //календаря компьютера
decodeDate(Date,y,m,d);
Label1.Caption:=intToStr(y);    //надпись блокнота
end;
```

9. Выделите элемент управления **TabControl1,** в **Object Inspector** перейдите на страницу **Events** (события) и найти событие **OnChange.**
10. Введите в окне редактора кода процедуру для работы с элементом управления **TabControl1**.

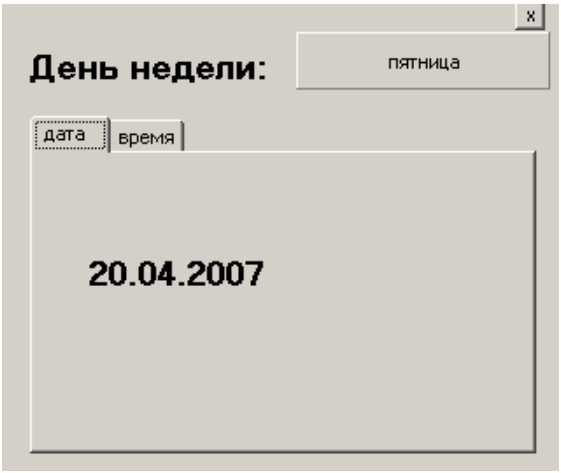
```
procedure TForm1.TabControl1Change(Sender: TObject);
begin
case tabcontrol1.TabIndex of
 0:
 begin
 label1.Caption:='';label1.Caption:=IntToStr(y);
 end;
 1:
 begin
 label1.Caption:='';
  case m of
  1: label1.Caption:='январь';
  2: label1.Caption:='февраль';
  3: label1.Caption:='март';
  4: label1.Caption:='апрель';
  5: label1.Caption:='май';
  6: label1.Caption:='июнь';
    7: label1.Caption:='июль';
  8: label1.Caption:='август';
  9: label1.Caption:='сентябрь';
  10: label1.Caption:='октябрь';
  11: label1.Caption:='ноябрь';
  12: label1.Caption:='декабрь';
  end;
end;
 2:
 begin
 label1.Caption:='';
 label1.Caption:=IntToStr(d);
 end;
end;
end;
```

11. Сохраните изменения и запустите проект, убедитесь в его работоспособности.



Контрольное задание 1.

1. Разработайте приложение, с помощью которого пользователь в одностраничном блокноте выбирает одну из представленных вкладок. На рабочем поле блокнота высвечивается соответствующая надпись: дата, время календаря компьютера. На панели отображается день недели.



2. Измените, проект так, чтобы каждый день недели был раскрашен в разный цвет.

Практическая работа №23.

«Создание проекта с использованием группы зависимых переключателей»

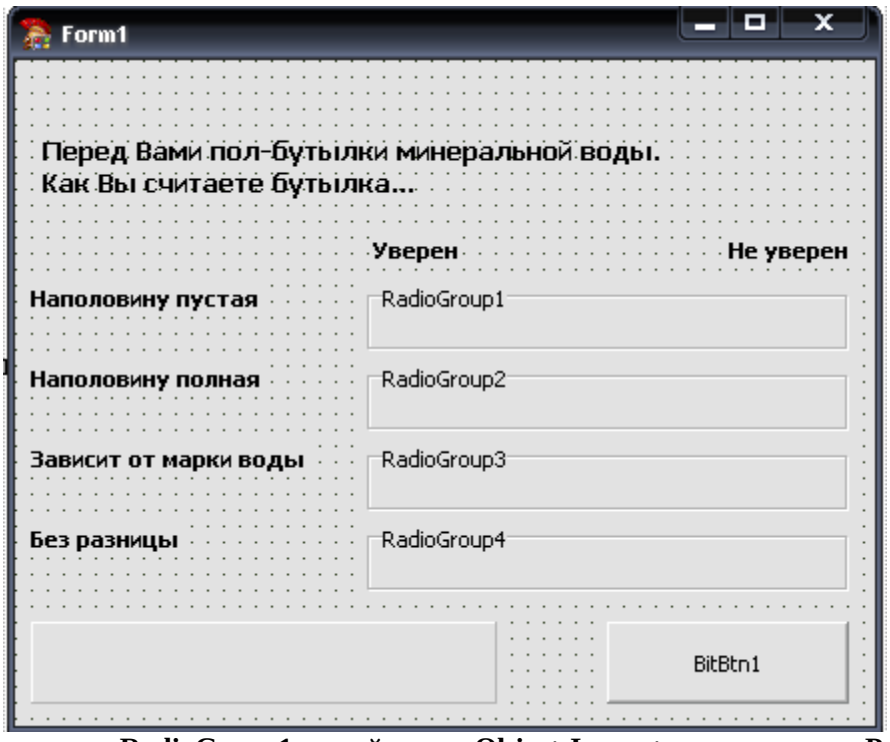
Цель работы: сформировать умения использования переключателей при создании проекта Delphi, изучить их основные свойства; сформировать умения по созданию процедур на основе событий компонентов.
Оборудование, технические и программные средства: персональный компьютер, интегрированная среда разработчика **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 созданную ранее 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. Для этого выделите элемент управления и перейдите в **Object Inspector** на страницу **Properties** (свойства).
5. Оставьте свойство **Caption** панели **Panel1** пустым.



6. 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.**
7. Вызовите **String List Editor**, дважды щелкнув мышкой рядом со свойством **Items**, введите **5** строк со значениями 1, 2, 3, 4 и 5 соответственно.
8. Аналогичные действия проделайте с остальными компонентами **RadioGroup**. Можно выделить элементы управления с нажатой клавишей **Shift** и задать свойства, которые будут заданы для всех выделенных элементов управления.
9. Для того чтобы суммировать набираемые пользователем баллы, в процедуру обработки события **RadioGroup1.OnClick** вставьте следующий код:

```
sum:=0;
with RadioGroup1 do
if ItemIndex>=0 then sum:=sum+ItemIndex+1;
```

Единицу необходимо прибавлять, т.к. индекс первого переключателя равен 0, но соответствует 1 баллу. Целочисленную переменную **sum** необходимо описать в разделе **var** модуля.

10. Вставьте в процедуру обработки событий **RadioGroup2.OnClick**, **RadioGroup3.OnClick** и **RadioGroup4.OnClick** аналогичные коды, но без обнуления переменной **sum**, т.к. обнуление необходимо лишь один раз перед началом суммирования.
11. Выведем на контрольную панель итоговое сообщение в зависимости от набранной суммы баллов. Для этого в процедуру обработки события **RadioGroup4.OnClick** добавим код:

```
case sum of
4..9:   Panel1.Caption:='Вы пессимист';
10..15: Panel1.Caption:='Вы реалист';
16..20: Panel1.Caption:='Вы оптимист';
end;
```

12. 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**.

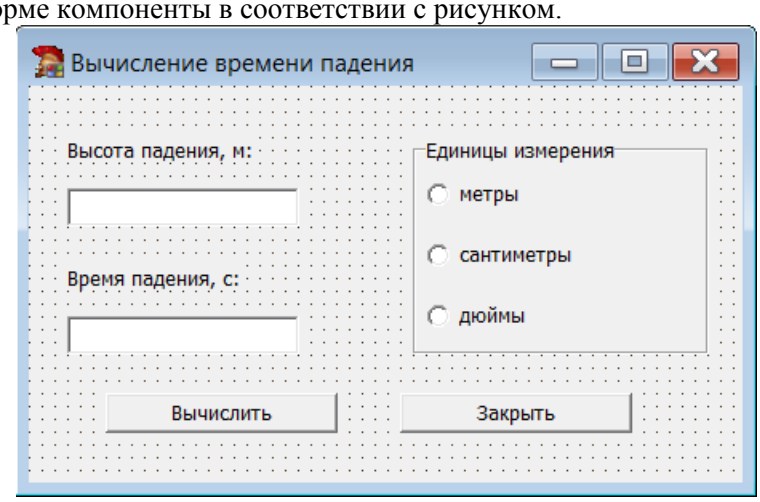
После нажатия кнопки **Ok** в информационном окне сделайте недоступными все **RadioGroup**, а все переключатели в них - невыбранными. Для этого свойствам **Enabled** всех **RadioGroup** присвоить значение **False**, а свойствам **ItemIndex** - значение **-1**.

13. Для контроля правильности работы программы выведите на панель набранную пользователем сумму баллов. Для этого заголовку соответствующей панели присвоить значение **IntToStr(sum)**.

14. Запустите программу и убедитесь, что верная сумма баллов получается лишь при последовательном выборе переключателей 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.
15. 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**.
16. Озаглавьте окно проекта **Проверь себя**.
17. Сохраните изменения и запустите проект, убедитесь в его работоспособности.

Задание 2.
Разработать приложение, с помощью которого можно вычислить время падения тела с некоторой высоты при условии, что высота может задаваться в метрах, сантиметрах и дюймах.

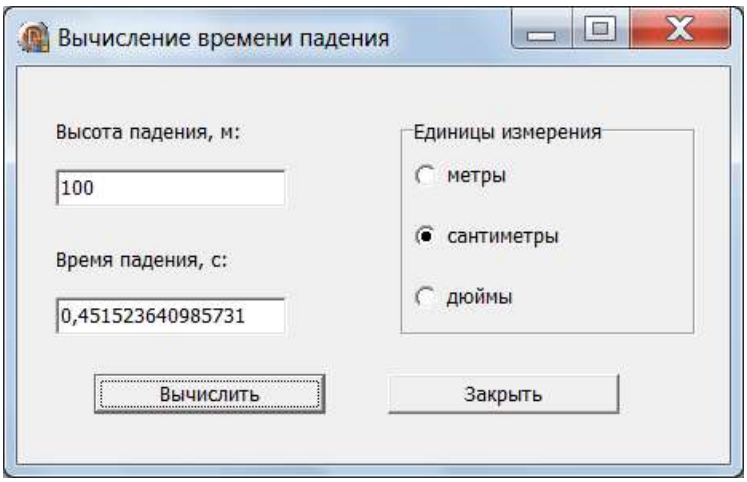
Методические указания по выполнению задания:
1. Запустите интегрированную среду разработчика **Turbo Delphi**. Создайте новый проект.
2. Создайте в папке своей группы новую папку и назовите её **Radio2**. Сохраните проект в созданную ранее папку, выполнив команду **File - Save Project as...**
3. Разместить на форме компоненты в соответствии с рисунком.



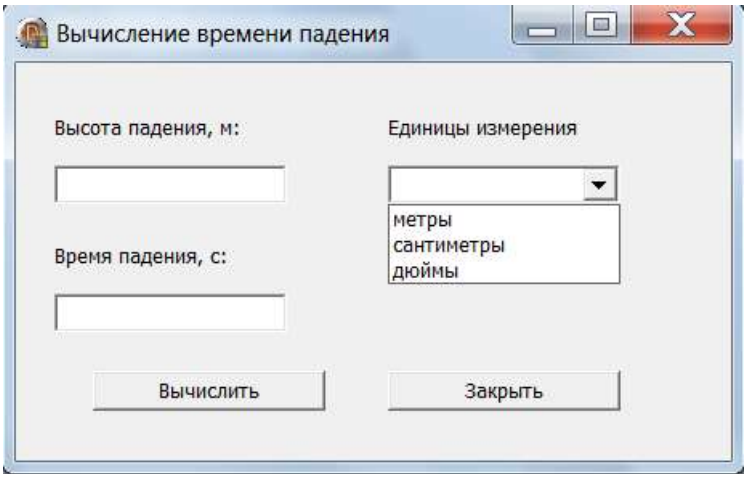
4. Создадим процедуру обработки события **Click** для кнопки **Вычислить**:

```
procedure TForm1.Button1Click(Sender: TObject);
 Const g=9.81;
 Var h,t : Real;
Begin
 h:=StrToFloat(Edit1.Text);
 Case radioGroup1.ItemIndex of
 0: t:=sqrt(2*h/g);          {Высота задается в метрах}
 1: t:=sqrt(2*h/100/g);      {Высота задается в санти-метрах}
 2: t:=sqrt(2*h*2.54/100/g); {Высота задается в дюймах}
 End;
 Edit2.Text:=FloatToStr(t);
end;
```

5. Запустите программу и проверьте её работоспособность.



6. Самостоятельно напишите обработчик события кнопки **Закрыть**.
7. 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**. Комбинированная 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 различать 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.
8. Измените проект так, чтобы выбор единицы измерения осуществлялся с помощью компонента **ComboBox.**

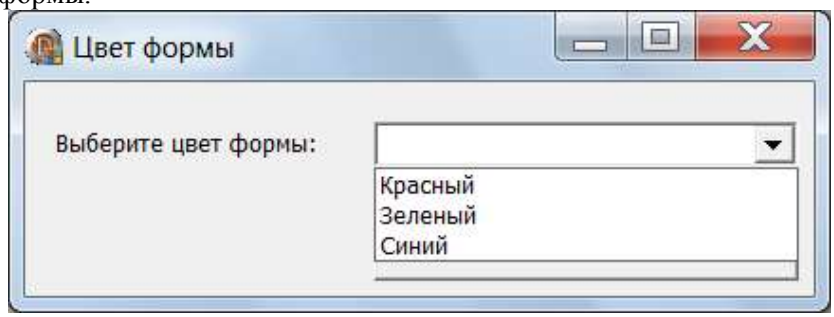



```
procedure TForm1.Button1Click(Sender: TObject);
 Const g=9.81;
 Var h,t : Real;
Begin
 h:=StrToFloat(Edit1.Text);
 if ComboBox1.Text='метры' then t:=sqrt(2*h/g);
 {Высота задается в метрах}
 if ComboBox1.Text='сантиметры' then t:=sqrt(2*h/100/g);
 {Высота задается в сантиметрах}
 if ComboBox1.Text='дюймы' then t:=sqrt(2*h*2.54/100/g);
 {Высота задается в дюймах}
 Edit2.Text:=FloatToStr(t);
end;
```

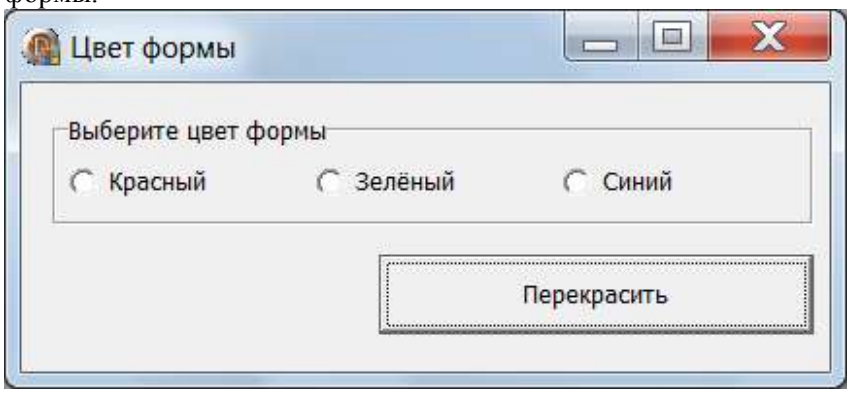
9. Выбор между компонентами **RadioGroup** и **ComboBox** во многом зависит от вкусов программистов, поскольку они выполняют одинаковую роль, а имеют различные внешний вид и применение. Необходимо отметить, что в языке **Delphi** есть большое количество компонентов, имеющих одинаковую область применения, однако отличающихся некоторыми частными внешними особенностями и доступными для применения методами.
10. Сохраните изменения в проекте.

Контрольное задание 1.

1. Разработайте приложение, которое при выборе определенного цвета в компоненте **ComboBox**, изменяет цвет формы.



2. Разработайте приложение, которое при выборе определенного цвета в компоненте **RadioGroup**, изменяет цвет формы.



Практическая работа №24.

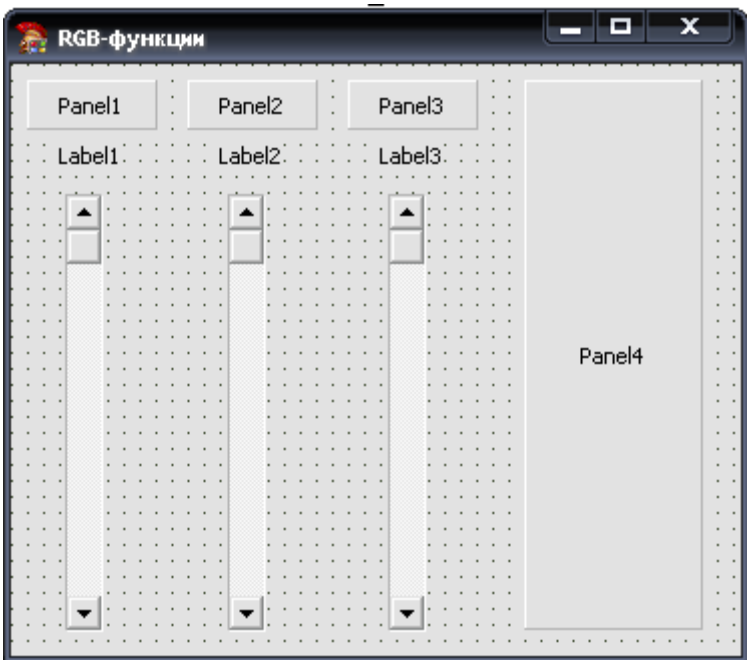
«Создание проекта с использованием полос прокрутки для ввода информации»

Цель работы: сформировать умения использования полос прокрутки для ввода информации **Delphi**, изучить основные свойства; сформировать умения по созданию процедур на основе событий компонента.
Оборудование, технические и программные средства: персональный компьютер, интегрированная среда разработчика **Turbo Delphi**.

Задание 1.
Разработать проект демонстрации работы RGB – функций (установок цвета по трем составляющим) с помощью полос прокрутки. Каждый бегунок полос прокрутки должен будет менять вклад RGB – компонента, отображающийся на панели как цвет, а на метке как число. Результирующий цвет должен отображаться 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 созданную ранее 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 редактирования **Windows** этот элемент появляется при необходимости автоматически.
В палитре VCL-компонент для создания полос прокрутки как самостоятельных элементов введен отдельный компонент: **ScrollBar** – самостоятельная полоса прокрутки, может быть горизонтальной 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 на стрелках 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.**

4. Задайте для элемента управления **ScrollBar1** значение свойства **Name**=**RedBar** и установить значение свойств **Max=255**, **Position=122**.
5. Задайте для элемента управления **Scrollbar2** значение свойства **Name=GreenBar** и установите значение свойств **Max=255**, **Position=122**.
6. Задайте для элемента управления **Scrollbar3** значение свойства **Name=BlueBar** и установите значение свойств **Max=255**, **Position=122**.
7. 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:

```
procedure TForm1.RedBarChange(Sender: TObject);
begin
Panel1.Color:=TColorRef(RGB(RedBar.Position,0,0));
Label1.Caption:=IntToStr(RedBar.Position);
Panel4.Color:=TColorRef(RGB(RedBar.Position,GreenBar.Position,BlueBar.Position))
end;
```

Функция **TColorRef** преобразует значение цветовых составляющих в число типа **LongInt**.

8. Аналогично запишите процедуры обработки события **OnChange** для элементов **GreenBar** и **BlueBar**.
9. Задайте имя формы проекта **RGB-функции**. Создайте кнопку **Close**.
10. Вызов процедур обработки событий **OnChange** поместите в событие при создании формы **OnCreat** для **Form1**.

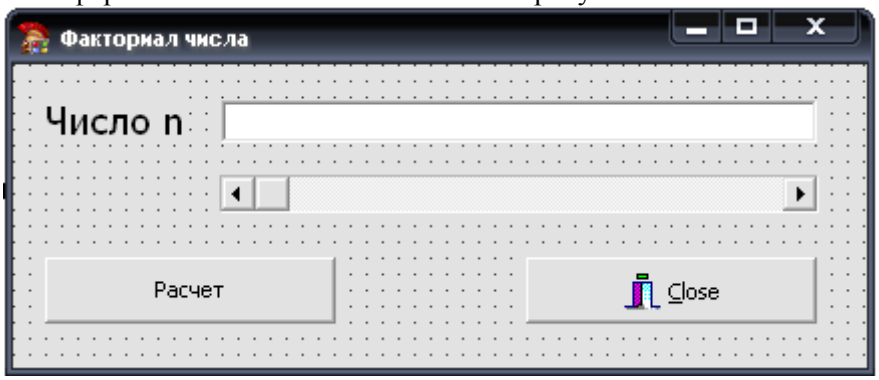
```
procedure TForm1.FormCreate(Sender: TObject);
begin
BlueBar.OnChange(Sender);
RedBar.OnChange(Sender);
GreenBar.OnChange(Sender);
end;
```

11. Сохраните изменения и запустите проект, убедитесь в его работоспособности.

Задание 2.
Разработать проект, который позволяет пользователю вычислить факториал числа. Число, для которого рассчитывается факториал, выбирается с помощью вертикальной полосы прокрутки. При щелчке по кнопке «Расчет», меняется надпись «Число n» на «N!» и в строке ввода выводится значение факториала числа.

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 в созданную 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:

```
procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
Edit1.Text:=IntToStr(ScrollBar1.Position);
end;
```

6. Запишите процедуру обработки события **OnClick** кнопки «**Расчет**». При расчете факториала числа воспользоваться конструкцией цикл.
7. Задайте имя формы проекта **Факториал числа**. Создайте кнопку **Close.**
8. Сохраните изменения в проекте.

Практическая работа №25.
«Создание проекта с использованием компонентов стандартных диалогов и системы меню»

Цель работы: сформировать умения использования компонентов стандартных диалогов и системы меню, изучить основные свойства; сформировать умения по созданию процедур на основе событий компонентов.

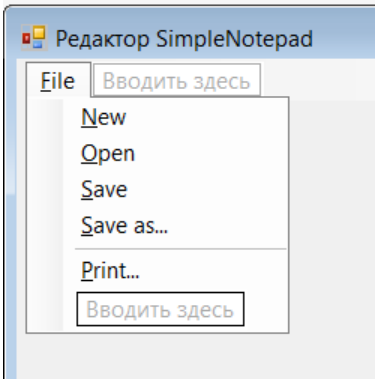
Оборудование, технические и программные средства: персональный компьютер, среда программирования **Visual Studio**.

Задание:

Разработать приложение **SimpleNotepad**, представляющее собой простейший текстовый редактор. Использовать в приложении компонент для работы с меню и стандартные окна диалога.

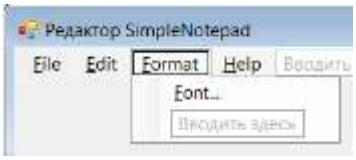
Методические указания по выполнению задания:

1. Запустите среду программирования **Visual Studio**. Создайте новое **Приложение Windows Forms**. Имя проекта и приложения **SimpleNotepad**.
2. Измените свойство **Name** формы на **SimpleNotepadForm.**
3. Перейдите в окно **Обозреватель решений** и переименуйте файл **Form1.cs** на **SimpleNotepadForm.cs.**
4. Измените размеры окно формы. Измените заголовок окна на **Редактор SimpleNotepad**.
5. Добавьте в окно приложения элемент управления **MenuStrip**.
6. 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 потому, что перед ней стоит префикс &. Этим префиксом отмечается буква, предназначенная для ускоренного выбора меню при помощи клавиатуры.
7. Создайте меню **File** в соответствии с рисунком. Для вставки разделительной черты необходимо вызвать контекстное меню и выбрать команду **Separator**.

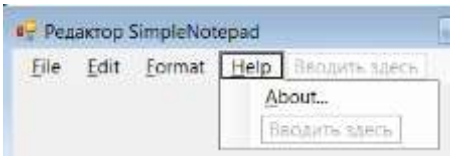


8. Создайте меню **Edit** в соответствии с рисунком. В меню **Edit** реализованы стандартные функции приложения **Блокнот**.

9. Меню **Format** состоит только из одной строки **Font**, с помощью которой пользователь может изменить шрифт текста.



10. Меню **Help** состоит только из одной строк **About**.



11. Переименуйте идентификаторы меню и строк меню таким образом, чтобы с ними было удобнее 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.**
12. Запустите приложение. Убедитесь, что меню корректно отображается, и вы можете выбирать его строки.
13. 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.
14. Создайте обработчики событий, необходимые для выполнения таких функций как создание нового документа, открытие и сохранение документа.
15. Добавьте на форму компонент **OpenFileDialog** с вкладки **Диалоговые окна** панели элементов. 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 отображения диалогового окна открытия файла.

```
private void menuFileOpen_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog()==System.Windows.Forms.DialogResult.OK && openFileDialog1.FileName.Length>0)
    {
        try
        {
            richTextBox1.LoadFile(openFileDialog1.FileName, RichTextBoxStreamType.RichText);
        }
        catch (System.ArgumentException ex)
        {
            richTextBox1.LoadFile(openFileDialog1.FileName, RichTextBoxStreamType.PlainText);
        }
        this.Text = "Файл[" + openFileDialog1.FileName + "]";
    }
}
```

16. Выделите компонент **openFileDialog1**, на панели **Свойства** найдите свойство **Filter** и введите следующую строку **RTF files|*.rtf|Text files|*.txt|All files|*.***
17. 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 отображения диалогового окна **Сохранение файла**. Этот код сохраняет текст введенный в элемент управления **richTextBox**, в текстовый файл в указанной папке. Метод **ShowDialog** отображает диалоговое окно, а затем с помощью поля **DialogResult**.**OK** осуществляется проверка нажата ли пользователем кнопка **ОК.**

```
private void menuFileSave_Click(object sender, EventArgs e)
{
    if (saveFileDialog1.ShowDialog()==System.Windows.Forms.DialogResult.OK && saveFileDialog1.FileName.Length>0)
    {
        richTextBox1.SaveFile(saveFileDialog1.FileName);
        this.Text = "Файл [" + saveFileDialog1 + "]";
    }
}
```

18. Выделите компонент **saveFileDialog1**, на панели **Свойства** найдите свойство **Filter** и введите следующую строку **RTF files|*.rtf**, найдите свойство **FileName** и задайте значение **doc1.rtf**
19. Добавьте в окно дизайнера форм компоненты **PrintDocumet**, **PrintDialog** вкладки **Печать** панели элементов.

20. Компонент **PrintDocumet** предназначен для вывода данных документа на принтер. Свойства компонента **PrintDocumet** описывают, как именно нужно распечатывать документ. В свойство **DocumentName** данного компонента запишите строку **SimpleNotepad Document**. Эта строка будет идентифицировать документ при отображении состояния очереди печати. Значения остальных свойств оставьте без изменения.
21. С помощью компонента **PrintDialog** приложение выведет на экран стандартное диалоговое окно печати документа. Задайте для данного компонента значение свойства **Document** равным **printDocument1**. Этим обеспечивается связь компонента **printDialog** с компонентом **PrintDocumet**.
22. Для работы с классами, предназначенными для выполнения операций с потоками и печати, добавьте в начало программы следующие строки:

```
using System.IO;
using System.Drawing.Printing;
```

23. Добавьте в класса **SimpleNotepadForm** поля **m_myReader** (для печати содержимого редактора текста) и **m_PrintPageNumber** (номер текущей распечатываемой страницы документа).

```
public partial class SimpleNotepadForm : Form
{
    private StringReader m_myReader;
    private uint m_PrintPageNumber;

    public SimpleNotepadForm()
    {
```

24. Создайте обработчик события печати документа.

```
private void menuFilePrint_Click(object sender, EventArgs e)
{
    m_PrintPageNumber = 1;
    string strText = this.richTextBox1.Text;
    m_myReader = new StringReader(strText);
    Margins margins = new Margins(100, 50, 50, 50);
    printDocument1.DefaultPageSettings.Margins = margins;
    if (printDialog1.ShowDialog()==DialogResult.OK)
    {
        this.printDocument1.Print();
    }
    m_myReader.Close();
}
```

Печать документа будет начинаться с первой страницы, поэтому в поле **m_PrintPageNumber** записывается значение 1. Далее выполняется чтение текущего содержимого окна редактирования текста в поток **m_myReader** класса **StringReader**. Далее задаются границы отступов на распечатываемой странице и отображается диалоговое окно печати документа. Если пользователь щелкает в этом окне кнопку **OK**, документ **printDocument1** отправляется на печать методом **Print**. Далее ненужный более поток **m_myReader** закрывается методом **Close.**

25. 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.
26. Чтобы в нашем приложении заработала функция печати, необходимо создать обработчик события **PrintPage**. Для этого нужно дважды щелкнуть левой клавишей мыши значок компонента **printDocument1**. В тело обработчика событий вставьте программный код из текстового файла **Print.txt.** Комментарии в тексте обработчика событий **PrintPage** поясняют назначение отдельных программных строк. Заметим, что для полного понимания действий, выполняемых нашим обработчиком событий, требуется предварительное знакомство с графической подсистемой **Graphics Device Interface Plus (GDI+)**, реализованной компанией **Microsoft** в рамках библиотеки классов **.NET Framework**. Пока же нужно отметить, что приложение распечатывает текст построчно в цикле. После завершения печати всех строк текущей страницы обработчик событий **PrintPage** печатает верхний и нижний колонтитулы, а также рисует горизонтальные линии, отделяющие текст колонтитулов от текста документа.
27. Измените меню **File**, добавив пункт **Exit,** при выборе которого окно редактора текста должно быть закрыто. Добавьте обработчик события для данного пункта меню:

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.Close();
}
```

28. 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, нам нужно каким-то образом отслеживать наличие изменений в окне редактирования текста. Определите в классе **SimpleNotepadForm** поле **m_DocumentChanged**, в котором будет храниться флаг, отмечающий изменения, сделанные пользователем в документе. В новом или только что загруженном документе изменений нет, поэтому начальное значение этого флага равно **false.**

```
private bool m_DocumentChanged = false;
```

29. Создайте обработчик события **richTextBox1_TextChanged**, выполнив двойной щелчок по компоненту **richTextBox**. Этот обработчик получит управление, как только пользователь внесет любые изменения в содержимое редактируемого документа.

```
private void richTextBox1_TextChanged(object sender, EventArgs e)
{
    m_DocumentChanged = true;
}
```

30. Если пользователь редактировал документ, а потом решил создать новый, выбрав из меню **File** строку **New**, изменения, внесенные в старый документ, могут быть потеряны.
31. Чтобы избежать этого, проверьте флаг **m_DocumentChanged** перед тем как очищать содержимое редактора текста. Если в редакторе есть не сохраненные изменения, необходимо выполнить сохранение документа. Отредактируйте код обработчика события пункта меню **New** следующим образом:

```
private void menuFileNew_Click(object sender, EventArgs e)
{
    if (m_DocumentChanged)
    {
        if (saveFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK && saveFileDialog1.FileName.Length > 0)
        {
            richTextBox1.SaveFile(saveFileDialog1.FileName);
            this.Text = "Файл [" + saveFileDialog1 + "]";
            m_DocumentChanged = false;
        }
    }
    richTextBox1.Clear();
}
```

32. Отредактируйте код обработчика события пунктов меню **Save** и **Save As**, добавив строку кода **m_DocumentChanged=false**
33. Измените обработчик события пункта меню **Exit** следующим образом:

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (saveFileDialog1.ShowDialog()==System.Windows.Forms.DialogResult.OK && saveFileDialog1.FileName.Length>0)
    {
        richTextBox1.SaveFile(saveFileDialog1.FileName);
        this.Text = "Файл [" + saveFileDialog1 + "]";
        m_DocumentChanged = false;
    }
    this.Close();
}
```

34. Надо выполнить еще одну проверку флага **m_DocumentChanged** в методе **Dispose**, который вызывается при закрытии окна приложения. Для этого на панели Обозреватель решений выделите **SimpleNotepadForm.Designer.cs**, найдите метод **Dispose** и внесите необходимые изменения.

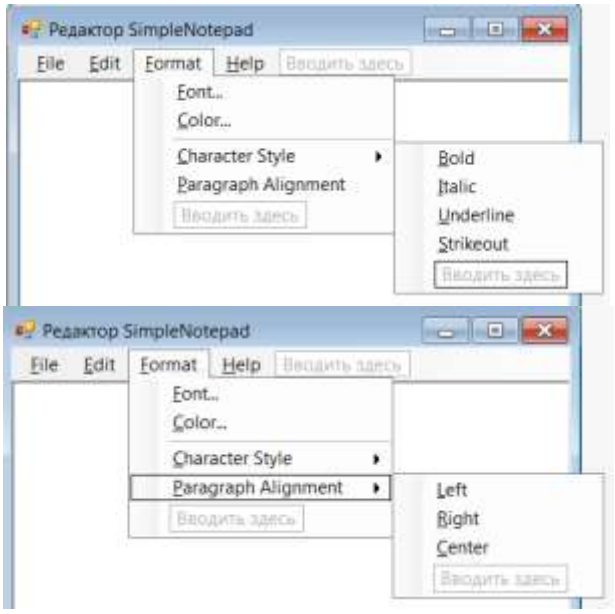
```
protected override void Dispose(bool disposing)
{
    if (saveFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK && saveFileDialog1.FileName.Length > 0)
    {
        richTextBox1.SaveFile(saveFileDialog1.FileName);
        this.Text = "Файл [" + saveFileDialog1 + "]";
        m_DocumentChanged = false;
    }
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}
```

Теперь, когда пользователь попытается закрыть программу с помощью соответствующей кнопки заголовка окна, не сохранив сделанные изменения, на экране появится стандартное диалоговое окно, предлагающее ему сохранить документ.

35. Создайте обработчики событий пунктов меню **Edit**. Реализация данных функций является простой, поскольку элемент управления **RichTextBox** имеет все необходимые методы. Добавьте в пункт меню **Edit** строку **Redo**. Подготовьте обработчики событий для всех строк меню **Edit**.

```
private void menuEditUndo_Click(object sender, EventArgs e)
{
    richTextBox1.Undo();
}

private void menuEditRedo_Click(object sender, EventArgs e)
{
    richTextBox1.Redo();
}

private void menuEditCut_Click(object sender, EventArgs e)
{
    richTextBox1.Cut();
}

private void menuEditCopy_Click(object sender, EventArgs e)
{
    richTextBox1.Copy();
}

private void menuEditPaste_Click(object sender, EventArgs e)
{
    richTextBox1.Paste();
}

private void menuEditDelete_Click(object sender, EventArgs e)
{
    richTextBox1.Cut();
}

private void menuEditSelectAll_Click(object sender, EventArgs e)
{
    richTextBox1.SelectAll();
}
```

36. 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.



37. 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:

```
private void menuFormatFont_Click(object sender, EventArgs e)
{
    if (fontDialog1.ShowDialog()==DialogResult.OK)
    {
        richTextBox1.SelectionFont = fontDialog1.Font;
    }
}
```

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.

38. 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:

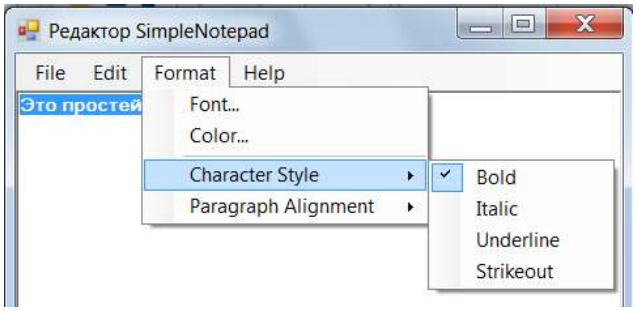
```
private void colorToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (colorDialog1.ShowDialog()==DialogResult.OK)
    {
        richTextBox1.SelectionColor = colorDialog1.Color;
    }
}
```

39. Создайте обработчик события для пункта меню **Bold** и вставьте следующий код:

```
private void menuFormatFontCharacterStyleBold_Click(object sender, EventArgs e)
{
    if (richTextBox1.SelectionFont!=null)
    {
        System.Drawing.Font currentFont = richTextBox1.SelectionFont;
        System.Drawing.FontStyle newFontStyle;
        if (richTextBox1.SelectionFont.Bold==true)
        {
            newFontStyle = FontStyle.Regular;
        }
        else
        {
            newFontStyle = FontStyle.Bold;
        }
        richTextBox1.SelectionFont = new Font(currentFont.FontFamily, currentFont.Size, newFontStyle);
        CheckMenuFontCharacterStyle();
    }
}
```

Получив управление данный метод прежде всего, определяет шрифт фрагмента текста, выделенного пользователем, анализируя свойство **richTextBox1.SelectionFont**. Если шрифт определить не удалось, и это свойство содержит значение **null**, программа не делает никаких изменений. В противном случае программа сохраняет текущий шрифт в переменной **currentFont** класса **System.Drawing.Font**. Далее метод проверяет, был ли выделен фрагмент текста жирным шрифтом, анализируя свойство **richTextBox1.SelectionFont.Bold**. Если это свойство содержит 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.**

40. При выделении фрагмента текста жирным шрифтом в меню одновременно отмечается «галочкой» 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.
41. Запустите программу и протестируйте её работу.



42. Аналогично создайте обработчики для пунктов меню **Italic, Underline, Strikeout**.

43. Подготовьте обработчики событий для строк меню **Paragraph Alignment** следующим образом:

```
private void menuFormatParagraphAlignmentLeft_Click(object sender, EventArgs e)
{
    richTextBox1.SelectionAlignment = HorizontalAlignment.Left;
}

private void menuFormatParagraphAlignmenRight_Click(object sender, EventArgs e)
{
    richTextBox1.SelectionAlignment = HorizontalAlignment.Right;
}

private void menuFormatParagraphAlignmentCenter_Click(object sender, EventArgs e)
{
    richTextBox1.SelectionAlignment = HorizontalAlignment.Center;
}
```

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 установить в него текстовый курсор).

44. Сохраните изменения и протестируйте работу приложения.

Практическая работа №26.

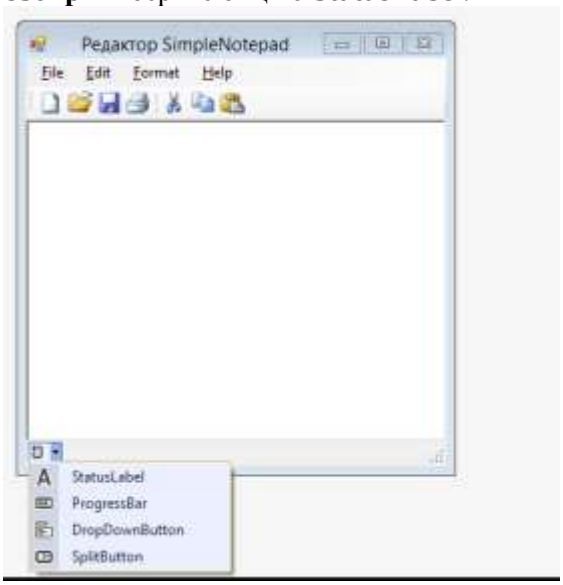
«Разработка оконного приложения»

Цель работы: сформировать умения использования компонентов стандартных диалогов и системы меню, изучить основные свойства.
Оборудование, технические и программные средства: персональный компьютер, среда программирования **Visual Studio**.
Задание
Методические указания по выполнению задания:

1. Во многих приложениях, созданных для операционной системы **Windows**, наиболее часто использующиеся 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 появится в верхней части формы. В только что добавленной панели нет ни одной кнопки.
2. Сразу после добавления окно инструментальной панели будет расположено над окном редактора 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 правильное взаимное расположение.
3. Создайте в папке проекта отдельную папку для хранения изображений инструментальной панели, назвав ее, например, **ImageList**. Далее скопируйте в созданную папку изображения, подготовленные 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 сохранённых ранее в папке **ImageList**.
7. Задайте комментарий-подсказку для кнопки **создатьToolStripButton**, установив свойство **ToolTripText** равным **Создать**.
8. Аналогично создайте кнопки **открытьToolStripButton, сохранитьToolStripButton, печатьToolStripButton.**
9. Далее необходимо создать разделитель между группами кнопок, ответственных за выполнение различных групп операций. Для этого в списке элементов выберите **Separator** и нажмите кнопку **Добавить**.
10. Далее аналогично создайте кнопки **вырезатьToolStripButton, копироватьToolStripButton, вставитьToolStripButton**.
11. Теперь необходимо связать созданные ранее функции-обработчики события команд пункт меню с соответствующими кнопками панели инструментов. Выделите кнопку **создатьToolStripButton,** на панели **Свойства** перейдите на вкладку **События**  , найдите событие **Click** и в списке выберите функцию **menuFileNew_Click**. Аналогично добавьте функции для остальных кнопок панели инструментов.
12. 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** панели элементов.
2. В меню компонента **StatusStrip** выберите опцию **StatusLabel**.



3. Далее на панели **Свойства** найдите свойство **Text** и сделайте его пустым.
4. Для того чтобы отслеживать выбор строк меню необходимо предусмотреть специальный обработчик события **DropDownItemClicked.** Это событие создается строками меню, когда пользователь 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** панели **Свойства**. В списке событий 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:

```
private void menuFile_DropDownItemClicked(object sender, ToolStripItemClickedEventArgs e)
{
    this.UpdateStatus(e.ClickedItem);
}
```

5. Создайте пустую строку перед обработчиком данного события и опишите метод **UpdateStatus** следующим образом:

```
private void UpdateStatus(ToolStripItem item)
{
    if (item != null)
    {
        string msg = String.Format("{0} selected", item.Text);
        this.statusStrip1.Items[0].Text = msg;
    }
}
```

6. 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.
7. Сохраните приложение. Запустите его и протестируйте работу строки состояния.

Практическая работа №27.

«Разработка оконного приложения с несколькими экранами»

Цель работы: сформировать практические умения по созданию простейшего приложения с несколькими экранами в среде **MIT App Inventor**.
Оборудование, технические и программные средства: персональный компьютер, программа браузер.
Задание 1.
Используя инструменты среды **MIT App Inventor**, создать приложение, в котором при щелчке по кнопке один 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 копирования блоков между экранами.

Навигация (переход) между экранами организуется с помощью кнопок или с помощью действий.

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 они открываются и закрываются. Экраны могут использовать для хранения и получения с других экранов данных только с использованием **TinyDB**, к примеру счет игры, количество набранных 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 приложения 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 недоступна для других приложений.

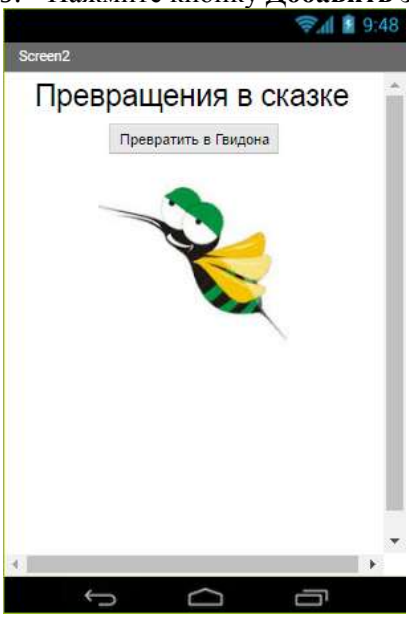
Свойств данный компонент не имеет

Компонент **TinyDB** очень полезен, поскольку позволяет сохранить данные приложения на **Android** устройстве. Обычно такие небольшие данные используются для сохранения настроек приложения.

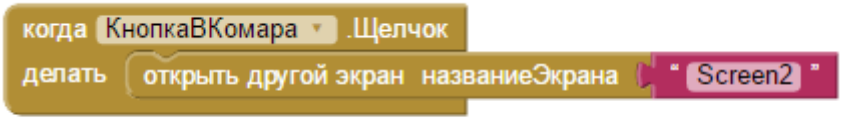
После того как данные сохранены в **TinyDB** они останутся там, пока **TinyDB** не будет очищено.

3. На панели **Свойства** установите выравнивание всех объектов **Screen1** по центру, для этого выберите свойство **ВыровнятьПоГоризонтали** и задайте его значение равным **Центр.**
4. 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**.
5. На панели **Компоненты** выделите **Надпись1** и нажмите внизу панели кнопку **Переименовать**, задайте имя компонента **НадписьЗаголовок.**
6. 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.**
7. На панели **Компоненты** выделите **Кнопка1** и нажмите внизу панели кнопку **Переименовать**, задайте имя компонента **КнопкаВКомара.**
8. Перетащите компонент **Изображение** в окно экрана. Установите в качестве источника изображения файл **gvidon.jpg**. Нажмите кнопку **Переименовать** и дайте компоненту название **ИзображениеГвидон.**

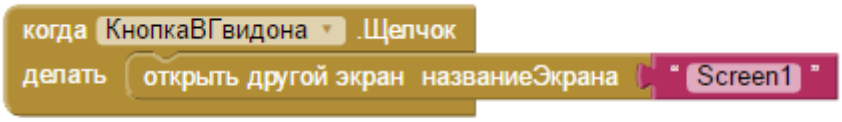
9. Нажмите кнопку **Добавить экран**. Перейдите на экран **Screen2**. Создайте по аналогии дизайн экрана.



10. Перейдите на экран **Screen1**. Перейдите на вкладку **Блоки**. Соберите из блоков следующую конструкцию.



11. Перейдите на экран **Screen2**. Перейдите на вкладку **Блоки**. Соберите из блоков следующую конструкцию.

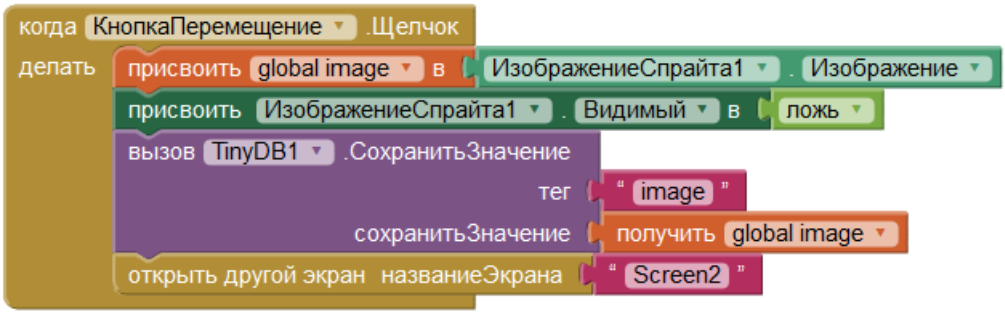


12. Протестируйте работу.

Задание 2.
Используя инструменты среды **MIT App Inventor,** создать приложение, в котором происходит перемещение 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 его в 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 **Перенести в палаты.**

10. В палитре перейдите в раздел **Хранилище**, выберите компонент **TinyDB** и переместите его в окно **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**.

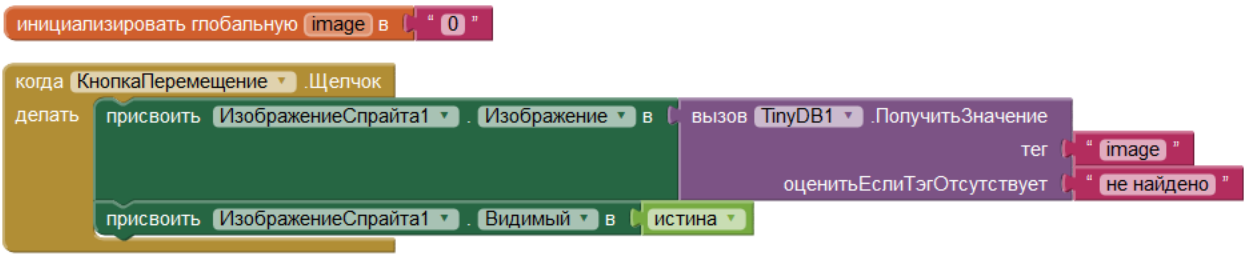
Как только **Кнопка** будет нажата этой переменной необходимо присвоить **ИзображениеСпрайта** изображение Гвидона, который будет перемещаться. После перемещения изображение не должно отображаться на экране, поэтому ему присваивается параметр **Видимый – Ложь.**

Далее вызывается компонент **TinyDB**, задается изображению **ТЕГ** (имя) под которым он будет храниться 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.

13. Перейдите в режим **Блоки** и соберите следующую конструкцию:



Разберем блоки конструкции более подробно. Вначале также осуществляется инициализация глобальной 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 получении значения - задается только **ТЕГ**.

В случае если допущена ошибку и изображение во внешнем хранилище не найдено, выдается сообщение, что оно не найдено.

14. Протестируйте работу приложения.

Задание 3.

Используя инструменты среды **MIT App Inventor,** разработайте контентное приложение с использованием меню. Для создания приложения, используйте материалы папки **SourceOcean**.

Практическая работа №28.

«Разработка многооконного приложения»

Цель работы: сформировать практические умения создания многооконного приложения сс фреймами средствами среды программирования **Visual Studio**.

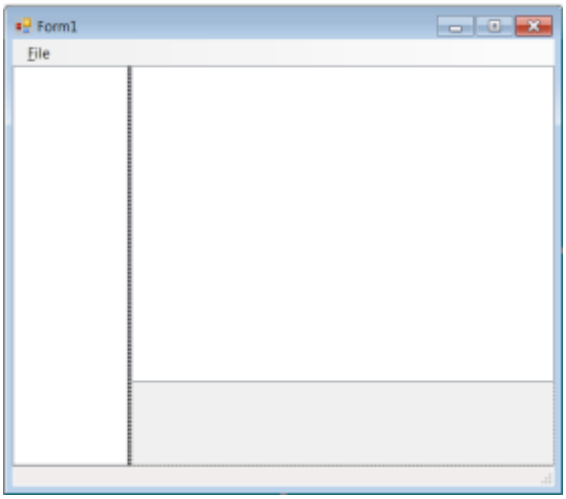
Оборудование, технические и программные средства: персональный компьютер, среда программирования **Visual Studio**.

Задание 1. Создание главного окна приложения

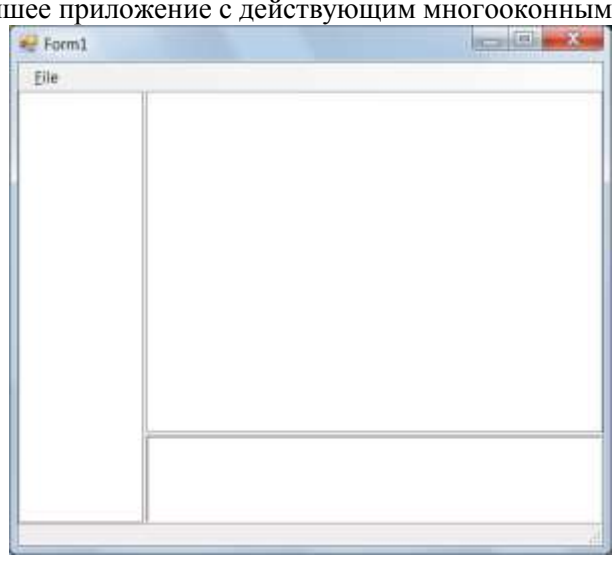
Методические указания по выполнению задания:

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 обработчик данного события.

3. Добавьте в окно нашего приложения элемент управления **TreeView**, перетащив мышью его значок 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.
4. Для элемента управления **TreeView** создайте разделитель. Для этого перетащите значок элемента управления **Splitter** из панели элементов вкладки **Все формы Windows Forms** в окно формы. Разделитель будет автоматически расположен справа от окна элемента управления **TreeView.**
5. Элемент управления **Splitter** имеет свойства, которые можно менять в процессе разработки приложения, 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,** благодаря чему разделитель разместился слева от окна дерева **TreeView**. При необходимости можно изменять расположение разделителя с помощью окна редактирования, аналогичного окну.
6. 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 результате наша панель будет расположена справа от разделителя и займет всю правую часть окна приложения.
7. 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 нашего приложения примет следующий вид.



8. 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 установите значение свойства **Dock** добавленного разделителя, равным **Top**. В результате разделитель будет расположен непосредственно под окном элемента управления **ListView**.
9. В нижней части окна приложения будет находиться элемент управления **RichTextBox**, предназначенный 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.**
10. Протестируйте работу приложения. Сохраните полученный проект. Не написав ни единой строчки кода, вы создали простейшее приложение с действующим многооконным интерфейсом пользователя.



Задание 2. Создание методов для элемента управления TreeView
Методические указания по выполнению задания:

1. Инициализация дерева **TreeView** осуществляется в конструкторе класса **Form1**. Для этого необходимо подготовить метод **DriveTreeInit**.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        DriveTreeInit();
    }
```

2. Для обращения к этому методу, а также к другим методам, работающим с дисками, каталогами и файлами, подключите пространство имен **System.IO**.

```
using System.IO;
```

3. Перейдите в окно редактирования кода и создайте метод **DriveTreeInit**:

```
public void DriveTreeInit()
{
    string[] drivesArray = Directory.GetLogicalDrives();

    treeView1.BeginUpdate();
    treeView1.Nodes.Clear();

    foreach(string s in drivesArray)
    {
        TreeNode drive = new TreeNode(s, 0, 0);
        treeView1.Nodes.Add(drive);

        GetDirs(drive);
    }
    treeView1.EndUpdate();
}
```

В самом начале своей работы этот метод получает список логических дисковых устройств, установленных в 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 используется в том случае, когда нужно добавить, удалить или изменить большое количество элементов дерева. Если не заблокировать перерисовку окна, на обновление дерева уйдет слишком 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 устройства 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 дискового устройства.

4. Метод **GetDirs** получает в качестве параметра ссылку на узел дерева, который требуется наполнить списком имен каталогов и файлов. Создайте метод **GetDirs.**

```
public void GetDirs(TreeNode node)
{
    DirectoryInfo[] diArray;
    node.Nodes.Clear();

    string fullPath = node.FullPath;
    DirectoryInfo di = new DirectoryInfo(fullPath);

    try
    {
        diArray = di.GetDirectories();
    }
    catch
    {
        return;
    }

    foreach(DirectoryInfo dirinfo in diArray)
    {
        TreeNode dir = new TreeNode(dirinfo.Name, 0, 0);
        node.Nodes.Add(dir);
    }
}
```

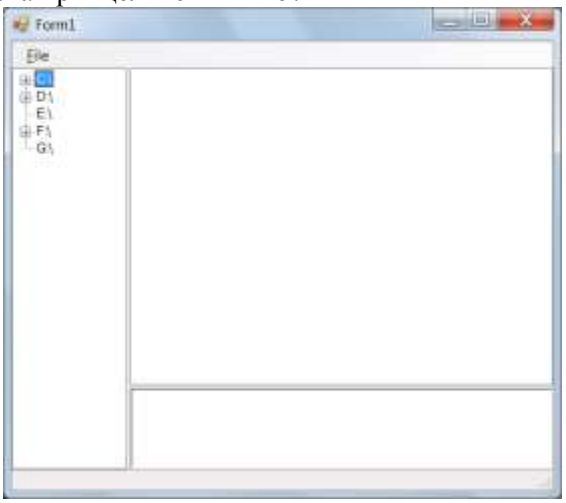
Первым делом метод **GetDirs** удаляет все элементы из текущего узла, вызывая для этого метод **node.Nodes.Clear**.

Далее метод переписывает в переменную **fullPath** типа **string** полный путь **node.FullPath** к узлу дерева. Эта строка получается объединением текстовых строк всех родительских узлов. Если корневой узел хранит текстовую 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**, используется метод **GetDirectories**. При возникновении исключения программа просто возвращает управление, не выполняя над узлом дерева никаких функций.

В том случае если содержимое каталога было успешно получено, оно сохраняется в массиве **diArray**. Далее содержимое массива **diArray** используется для заполнения узла дерева содержимым каталога.

5. Запустите программу на исполнение. В окне дерева появится список дисковых устройств. Раскрываются ли узлы дерева при щелчке мышью?

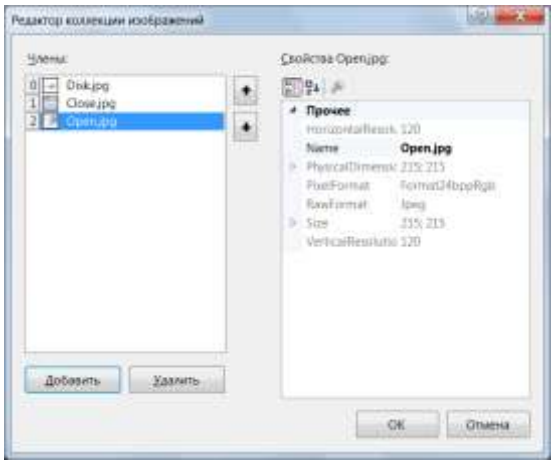


6. Чтобы узлы дерева раскрывались, когда пользователь щелкает их мышью или пытается раскрыть при помощи клавиатуры, нужно обрабатывать событие **BeforeExpand**. Для создания обработчика этого события выделите дерево в окне дизайнера формы, а затем откройте вкладку событий. Далее в поле **BeforeExpand** ведите имя обработчика событий **treeView1_OnBeforeExpand**. После этого добавьте в этот обработчик событий следующий код:

```
private void treeView1_OnBeforeExpand(object sender, TreeViewCancelEventArgs e)
{
    treeView1.BeginUpdate();

    foreach (TreeNode node in e.Node.Nodes)
    {
        GetDirs(node);
    }

    treeView1.EndUpdate();

}
```

Событие **BeforeExpand** возникает при попытке пользователя раскрыть узел дерева. В этом случае наш обработчик заполняет открываемый узел при помощи рассмотренного ранее метода **GetDirs**. Ссылка на узел извлекается из поля **e.Node.Nodes**, передаваемого обработчику событий в качестве параметра.

7. Запустите приложение. Раскрываются ли узлы дерева при щелчке мышью теперь?
8. 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 к проекту, выберите из меню проект системы **Microsoft Visual Studio** строку **Добавить существующий элемент**, а затем добавьте файлы значков при помощи появившегося на экране диалогового окна.
9. 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 **Редактор 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.



10. 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**.
11. 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 список каталогов. 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

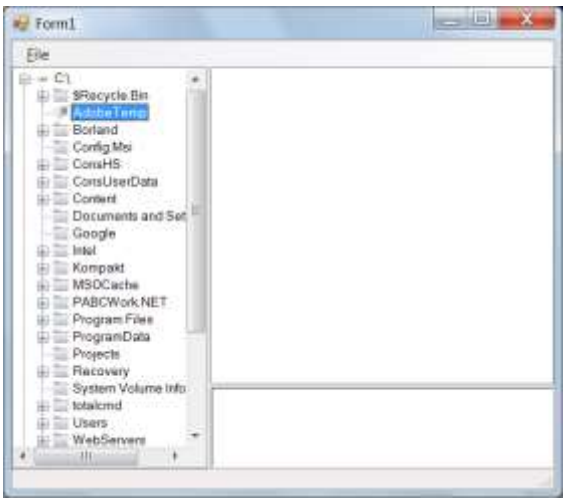
значок дискового устройства, имеющий в нашем случае индекс 0. Поэтому второй и третий параметры конструктора передают нулевые значения.

```
public void DriveTreeInit()
{
    string[] drivesArray = Directory.GetLogicalDrives();

    treeView1.BeginUpdate();
    treeView1.Nodes.Clear();

    foreach(string s in drivesArray)
    {
        TreeNode drive = new TreeNode(s, 0, 0);
        treeView1.Nodes.Add(drive);

        GetDirs(drive);
    }
    treeView1.EndUpdate();
}
```

12. В методе **GetDirs** конструктору класса **TreeNode**, размещенному внутри оператора цикла **foreach**, через второй и третий параметры передаются индексы значков закрытой и открытой папки, соответственно.

```
public void GetDirs(TreeNode node)
{
    DirectoryInfo[] diArray;
    node.Nodes.Clear();

    string fullPath = node.FullPath;
    DirectoryInfo di = new DirectoryInfo(fullPath);

    try
    {
        diArray = di.GetDirectories();
    }
    catch
    {
        return;
    }

    foreach(DirectoryInfo dirinfo in diArray)
    {
        TreeNode dir = new TreeNode(dirinfo.Name, 1, 2);
        node.Nodes.Add(dir);
    }
}
```

13. Протестируйте работу приложения.



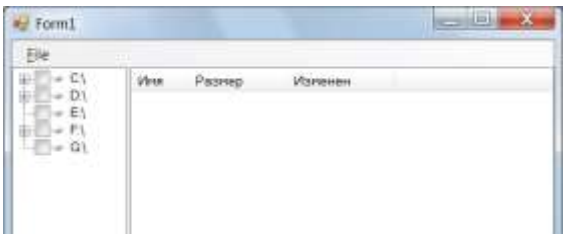
14. Выделите элемент управления **TreeView** на панели Свойства найдите свойство **CheckBoxes** и установите его значение **True**. Теперь узлы дерева будут иметь индивидуальные флажки. Отметив все или некоторые узлы дерева флажками, можно выполнять над соответствующими объектами групповые операции.
15. Выделите элемент управления **TreeView** на панели Свойства найдите свойство **LabelEdit** и установите его значение **True**. Теперь пользователь может редактировать текстовые надписи, расположенные около узлов дерева.

Практическая работа №29.

«Разработка многооконного приложения»

Цель работы: сформировать практические умения создания многооконного приложения сс фреймами средствами среды программирования **Visual Studio**.
Оборудование, технические и программные средства: персональный компьютер, среда программирования **Visual Studio**.
Задание 3. Создание методов для элемента управления ListView
Методические указания по выполнению задания:

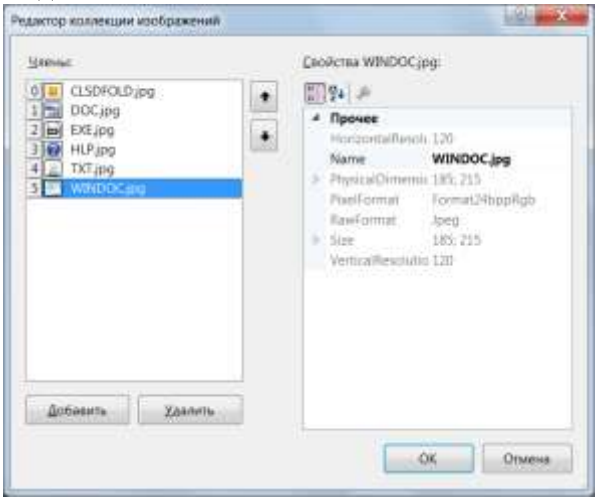
1. Выделите список **ListView** в окне дизайнера форм и установите для свойства **View** значение **Details**. При необходимости программа сможет динамически переключать режимы отображения, изменяя значение свойства **View** во время своей работы.
2. Поскольку в приложении будет отображаться список в 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 верхней 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.



5. Элемент управления **ListView** может отображать содержимое своего окна в четырех различных режимах, задаваемых при помощи свойства **View**. В зависимости от выбранного режима, элементы

списка могут снабжаться значками маленького или большого размера. В приложении **FramesApp** будут использованы 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.

6. Для работы с этими значками потребуется еще один список изображений **ImageList**. Перетащите его значок с панели элементов в окно дизайнера форм. Новый список будет иметь идентификатор **imageList2**. Отредактируйте данный список.



7. 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 список к элементу управления **ListView**. Для этого присвойте свойствам **SmallImageList**, **LargeImageList** значение **imageList2**.
8. В общем случае список **ListView** содержит элементы, каждый из которых имеет дополнительные элементы более низкого уровня, которые называются атрибутами. Сам элемент представляется текстовой строкой в первом столбце списка, отображаемого в виде детализированной таблицы. Значения остальных атрибутов отображаются в остальных столбцах таблицы. Теперь необходимо наполнить список **ListView** функциональностью. Когда пользователь выделяет диск или каталог в дереве просмотра **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 обработчик события **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 корневому каталогу диска, либо к одному из подкаталогов. Далее мы создаем объект класса **DirectoryInfo** и получаем списки всех файлов и каталогов, располагающихся в каталоге, выделенном 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 конструктора 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. Эти имена тоже добавляются в цикле

Размер очередного файла мы получаем с помощью свойства **Length**, а дату последнего изменения — с помощью свойства **LastWriteTime** (как и для каталогов).

Расширение имени файла извлекается из полного имени методом **Path.GetExtension**, а затем все его буквы преобразуется в прописные методом **ToLower**. Непосредственный выбор значка выполняется при помощи оператора **switch**. Подготовленный элемент списка, описывающий текущий файл, добавляется в список методом **listView1.Items.Add.**

9. Протестируйте работу приложения. Приложение может отображать в левом фрейме дерево каталогов, а в правом верхнем фрейме — содержимое выбранных каталогов.
10. Разработайте обработчик события **ItemActivate** к элементу управления **ListView**, для того чтобы было можно отображать в окне текстового редактора содержимое текстовых файлов, отображаемых в правом верхнем 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:

```
private void OnItemActivate(object sender, EventArgs e)
{
    foreach(ListViewItem lvi in listView1.SelectedItems)
    {
        string ext = Path.GetExtension(lvi.Text).ToLower();
        if (ext==".txt"||ext==".htm"||ext==".html")
        {
            try
            {
                richTextBox1.LoadFile(Path.Combine(fullPath, lvi.Text),
                    RichTextBoxStreamType.PlainText);

                StatusLabel1.Text = lvi.Text;
            }
            catch
            {
                return;
            }
        }
        else if (ext==".rtf")
        {
            try
            {
                richTextBox1.LoadFile(Path.Combine(fullPath, lvi.Text),
                    RichTextBoxStreamType.RichText);

                StatusLabel1.Text = lvi.Text;
            }
            catch
            {
                return;
            }
        }
    }
```

В качестве первого параметра методу передается полный путь к файлу, полученный комбинированием полного пути каталога **fullPath**, выделенного в дереве, и имени файла **lvi.Text**, выделенного в списке **ListView**.

Через второй параметр методу **LoadFile** передается тип загружаемого документа, заданный как **RichTextBoxStreamType.PlainText** (т.е. текстовый документ). Дополнительно имя документа отображается в строке состояния главного окна приложения, для чего это имя переписывается из свойства **lvi.Text** в свойство **statusLabel1.Text**.

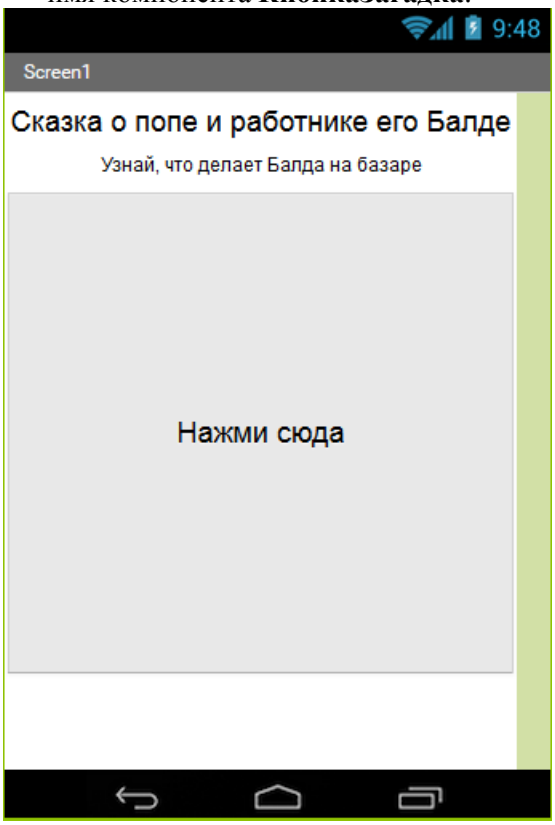
11. Протестируйте работу приложения. Сохраните изменения.

Практическая работа №30.

«Добавление в программы художественного оформления и специальных эффектов»

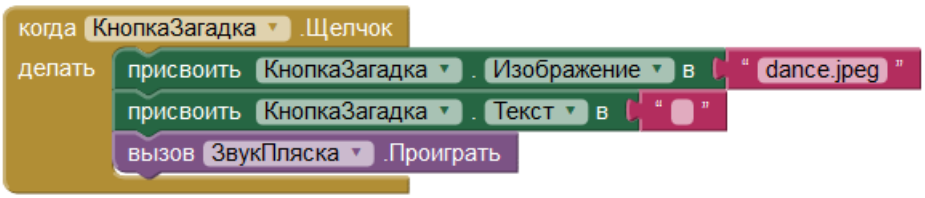
Цель работы: сформировать практические умения по созданию простейшего приложения в среде **MIT App Inventor** содержащих мультимедиа объекты.
Оборудование, технические и программные средства: персональный компьютер, программа браузер.
Задание 1.
Используя инструменты среды **MIT App Inventor,** создать приложение, в котором при нажатии на кнопку меняется изображение на ней и проигрывается звук.
Методические указания по выполнению задания:

1. Запустите программу браузер и перейдите по ссылке **ai2.appinventor.mit.edu**. Создайте новый проект, назовите его **AudioButton**.
2. На панели **Свойства** установите выравнивание всех объектов **Screen1** по центру, для этого выберите свойство **ВыровнятьПоГоризонтали** и задайте его значение равным **Центр**.
3. В палитре выберите компонент **Надпись** и переместите в окно **Screen1**. На панели **Свойства** поменяйте текст надписи на **Сказка о попе и работнике его Балде**.
4. На панели **Компоненты** выделите **Надпись1** и нажмите внизу панели кнопку **Переименовать**, задайте имя компонента **НадписьЗаголовок**.
5. В палитре выберите компонент **Надпись** и переместите в окно **Screen1**. На панели **Свойства** поменяйте текст надписи на **Узнай, что Балда делает на базаре**. Измените размер шрифта надписи на **14 пт**.
6. На панели **Компоненты** выделите **Надпись1** и нажмите внизу панели кнопку **Переименовать**, задайте имя компонента **НадписьКомментарий**.
7. В палитре выберите компонент **Кнопка** и переместите в окно **Screen1**. На панели **Свойства** поменяйте текст кнопки на **Нажми сюда**. Установите ширину и высоту кнопки **300 px**. Измените размер шрифта надписи на **20 пт**.
8. На панели **Компоненты** выделите **Кнопка1** и нажмите внизу панели кнопку **Переименовать**, задайте имя компонента **КнопкаЗагадка**.



9. Загрузите файл с картинкой для установки на кнопку. Для этого на панели **Медиа** нажмите кнопку **Загрузить файл** и выберите файл для загрузки **Dance.jpg**.
10. В палитре перейдите в раздел **Медиа**. Выберите компонент **Звук** и переместите его в окно **Screen1**. Переименуйте компонент **Звук** на **ЗвукПляска**.

11. Загрузите звуковой файл **Dance2.mp3** для компонента **ЗвукПляска**. на панели **Свойства** задайте значение свойства **Источник** для звука файл **Dance2.mp3.**
12. Перейдите на вкладку **Блоки**. Соберите из блоков следующую конструкцию.



Объекты (компоненты) приложения могут реагировать на определенные события: нажатие на какую-либо клавишу, щелчок мышью по объекту, прикосновение к экрану или достижение края экрана и т.п.

После определенного события могут происходить какие-либо действия. Основным средством для обработки события являются так называемые блоки заголовков. Например, Когда **Кнопка Шелчок (событие)** - проигрывается **Звук (действие).**

Некоторые события могут не зависеть от пользователя, к ним относятся:

- События таймера,
- События датчика, например, координаты GPS
- События на устройство, например входящие сообщения.
- События, связанные с анимацией объектов, например, достигнут край или происходит наложение объектов.
- Анимация события, такие как два объекта столкновения
- Веб-события, связанные с данными поступающими из сети

Кнопки являются наиболее часто используемым компонентом и используются для запуска различных действий.

События, которые могут происходить с компонентом **Кнопка** по инициативе пользователя, включают в себя, следующие

- Щелчок
- В фокусе
- Потерян Фокус
- Долгое нажатие
- Провести вниз

Провести вверх

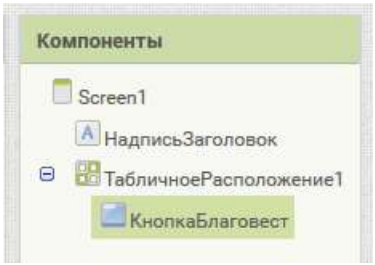
13. Протестируйте работу.

Задание 2.

Используя инструменты среды **MIT App Inventor,** создать приложение, в котором при нажатии на соответствующие изображения, проигрываются соответствующие звуки.

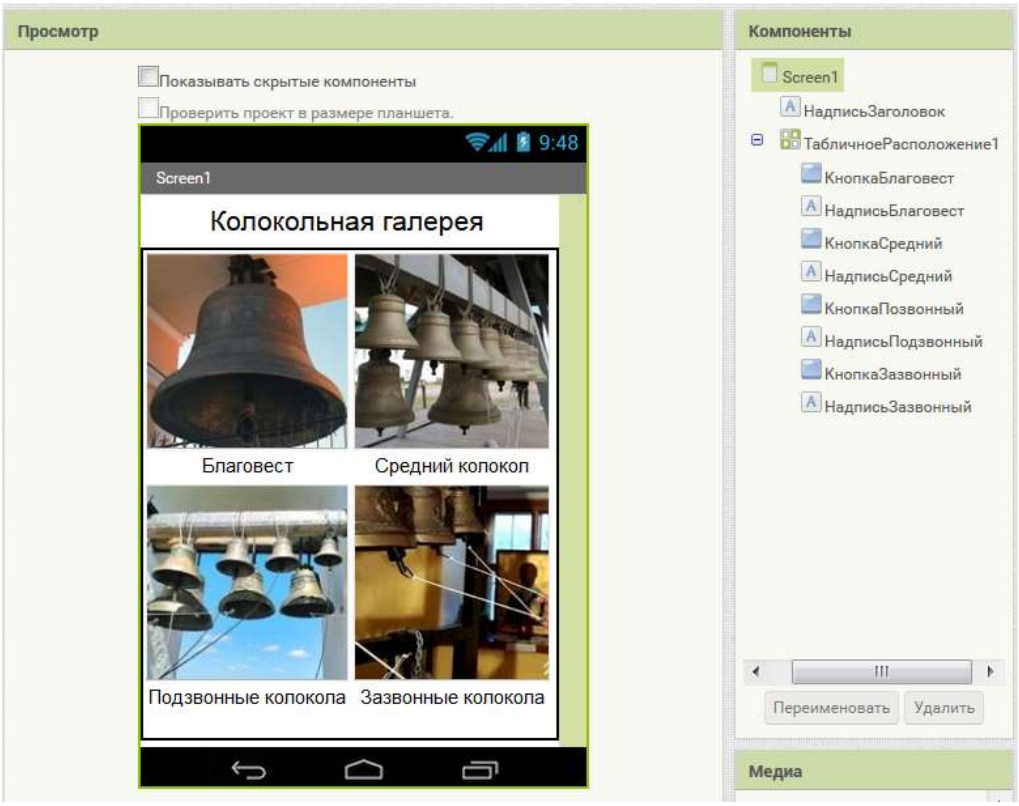
Методические указания по выполнению задания:

1. Создайте новый проект, назовите его **BellGallery**. 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 **Колокольная галерея**. Измените размер шрифта надписи на **24 пт.**
3. На панели **Компоненты** выделите **Надпись1** и нажмите внизу панели кнопку **Переименовать,** задайте имя компонента **НадписьЗаголовок.**
4. В палитре перейдите в раздел **Расположение**. Выберите компонент **Табличное расположение** и переместите его в окно **Screen1.**
5. На панели **Свойства** установите следующие параметры для компонента **Табличное расположение**: высота – **Наполнить родительский**, ширина – **Наполнить родительский**, строки – **8.**
6. В палитре выберите компонент **Кнопка** и переместите в окно **Screen1**. Выделите компонент **Кнопка** и нажмите **Переименовать**. Введите текст **КнопкаБлаговест.**

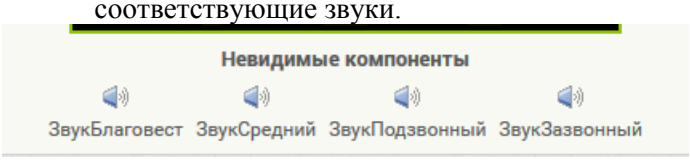


7. Загрузите для данной кнопки изображение. Установите его в качестве источника изображения. Удалите текст кнопки.
8. Перенесите на экран компонент **Надпись** и переименуйте ее в **НадписьБлаговест**.
9. Установите следующие параметры компонента **НадписьБлаговест**: размер шрифта - **18 пт**, текст – **Благовест**, Выравнивание - **по центру**.

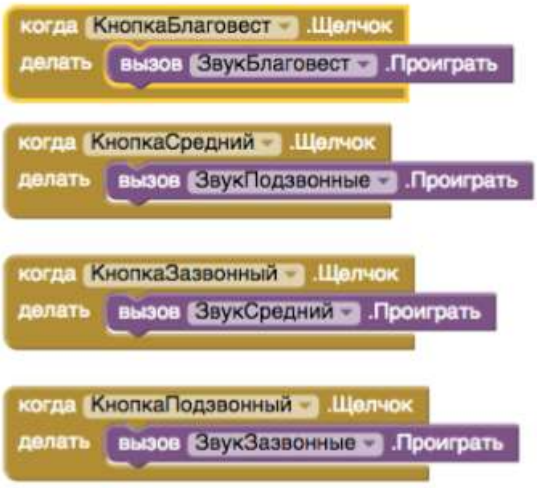
10. Аналогично оформите три оставшиеся кнопки.



11. Перенесите на экран компонент **Звук**. Переименуйте его в **ЗвукБлаговест**. Загрузите аудиофайл в качестве источника звука.
12. Создайте таким же образом три компонента **Звук**, установив в них в качестве источника соответствующие звуки.



13. Перейдите в режим **Блоки**. Соберите из блоков следующую конструкцию.

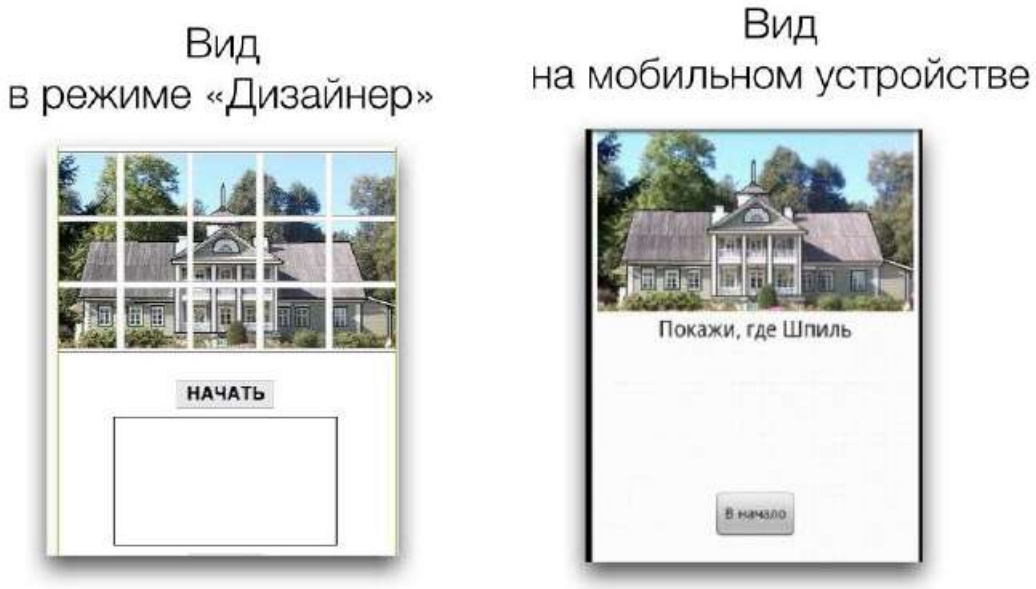


14. Перейдите в режим **Дизайнер**. Установите следующие свойства для **Screen1**: название приложения (**AppName**) - **Колокольная галерея**, иконка приложения - **blagovestnik.jpeg**, заголовок экрана – **Колокола**.
15. Протестируйте работу приложения.

Задание 3.
Используя инструменты среды **MIT App Inventor,** создать приложение, в котором при нажатии на часть изображения выдается сообщение, соответствующее данной части изображения.

Методические указания по выполнению задания:

1. Одним из интересных приемов работы с изображением служит скрытое разделение изображения на части. Изображение, части которого необходимо выделить разрезается на нужное количество частей с помощью **http://imagesplitter.net/** Полное изображение, создается из кнопок, каждая их которых содержит отдельный элемент. На основе таких приемов, можно строить тестовые задания с использованием изображений и кнопок.
2. Создайте данное приложение самостоятельно.



Практическая работа №31.

«Использование событий и методов мыши»

Цель работы: сформировать практические умения по созданию простейшего приложения с использованием событий и методов мыши.
Оборудование, технические и программные средства: персональный компьютер, программа браузер.
Событие Closed

Событие sf::Event::Closed срабатывает, когда пользователь хочет закрыть окно, используя любой из методов, предоставляемых менеджером окон (кнопка закрытия, макросы клавиатуры и так далее). Это событие предоставляет информацию о том, что пользователь попытался закрыть окно; оно еще не закрыто.

Обычно, в ответ на это событие, код вызывает window.close(), чтобы закрыть окно. Однако, вы можете сделать еще что-то, например, сохранить текущее состояние приложения или спросить пользователя, что надо сделать. Если вы ничего не сделаете в ответ на это событие, окно будет оставаться открытым.

Член класса, ассоциируемый с этим событием, не существует.

```
if (event.type == sf::Event::Closed)
    window.close();
```

Событие Resized

Событие `sf::Event::Resized` срабатывает, когда происходит изменение размера окна. Либо в результате действия пользователя, либо программно, после вызова `window.setSize`.

Вы можете использовать это событие, чтобы отрегулировать настройки отображения: область просмотра, если вы используете OpenGL напрямую, или текущую точку обзора, если вы используете `sfml-graphics`.

Член класса, ассоциируемый с этим событием, называется event.size и содержит новый размер окна.

```
if (event.type == sf::Event::Resized)
{
    std::cout << "new width: " << event.size.width << std::endl;
    std::cout << "new height: " << event.size.height << std::endl;
}
```

События LostFocus и GainedFocus

События `sf::Event::LostFocus` и `sf::Event::GainedFocus` срабатывают, когда окно потеряло/приобрело фокус; это происходит, когда пользователь меняет текущее активное окно. Когда окно теряет фокус, оно не получает события клавиатуры.

Это событие можно использовать, например, для приостановки игры, когда окно неактивно.

Член класса, ассоциируемый с этим событием, не существует.

```
if (event.type == sf::Event::LostFocus)
    myGame.pause();

if (event.type == sf::Event::GainedFocus)
    myGame.resume();
```

События TextEntered

Событие `sf::Event::TextEntered` срабатывает, когда происходит ввод символа. Это не событие `KeyPressed`: `TextEntered` срабатывает, когда пользователь вводит символ, который может быть выведен. Например, нажатие '^' и 'e' на французской клавиатуре приведет к двум событиям `KeyPressed` и только к одному `TextEntered`, содержащему символ 'ê'. Это работает для всех методов ввода, предоставленным операционной системой.

Это событие обычно используется для отлова пользовательского ввода в текстовое поле.

Член класса, ассоциируемый с этим событием, называется `event.text` и содержит номер символа Unicode введенного символа. Вы можете поместить это значение в sf::String, или привести его к типу `char` после того, как убедитесь, что этот символ лежит в диапазоне ASCII символов (0 — 127).

```
if (event.type == sf::Event::TextEntered)
{
    if (event.text.unicode < 128)
        std::cout << "ASCII character typed: " << static_cast<char>(event.text.unicode)
 << std::endl;
}
```

События KeyPressed и KeyReleased

События `sf::Event::KeyPressed` и `sf::Event::KeyReleased` срабатывают, когда клавиша на клавиатуре нажата/отжата.

Если клавиша зажата, события `KeyPressed` будут генерироваться с определенным интервалов, заданным операционной системой (т.е. та же задержка, что применяется, когда вы вводите текст в редакторе). Чтобы отменить повторение событий `KeyPressed`, вы можете вызвать `window.setKeyRepeatEnabled(false)`. Очевидно, что событие `KeyReleased` никогда не повторяется.

Эти события можно использовать, если вы хотите вызвать какое-нибудь действие ровно один раз, когда клавиша нажата или отжата, например, чтобы заставить персонажа прыгнуть.

Иногда люди пытаются использовать событие `KeyPressed` для реализации плавного движения. Это не приводит к ожидаемому эффекту, потому что это событие генерируется с определенным интервалом. Чтобы реализовать плавное движение с помощью событий, вы должны использовать логическое значение, устанавливаемое `KeyPressed` и обнуляемое `KeyReleased`; движение должно осуществляться, пока это логическое значение установлено.

Другие (более простые) пути реализации плавного движение, с использованием вода с клавиатуры с помощью sf::Keyboard (смотрите посвященную этому статью).

Член класса, ассоциируемый с этим событием, называется `event.key` и содержит код нажатого/отжатого символа, а также текущее состояние клавиш-модификаторов (alt, control, shift, system).

```
if (event.type == sf::Event::KeyPressed)
{
    if (event.key.code == sf::Keyboard::Escape)
    {
        std::cout << "the escape key was pressed" << std::endl;
        std::cout << "control:" << event.key.control << std::endl;
        std::cout << "alt:" << event.key.alt << std::endl;
        std::cout << "shift:" << event.key.shift << std::endl;
        std::cout << "system:" << event.key.system << std::endl;
    }
}
```

Событие MouseWheelMoved

Событие `sf::Event::MouseWheelMoved` устарело и было упразнено в SFML 2.3. Используйте `MouseWheelScrolled`.

Событие MouseWheelScrolled

Событие `sf::Event::MouseWheelScrolled` вызывается, когда колесо мыши двигается вверх, вниз или вбок (если это поддерживается мышью).

Член класса, ассоциируемый с этим событием, называется `event.mouseWheelScroll` и содержит число тиков, на которое сместилось колесо, ориентацию движения колеса и текущую позицию курсора мыши.

```
if (event.type == sf::Event::MouseWheelScrolled)
{
    if (event.mouseWheelScroll.wheel == sf::Mouse::VerticalWheel)
        std::cout << "wheel type: vertical" << std::endl;
    else if (event.mouseWheelScroll.wheel == sf::Mouse::HorizontalWheel)
        std::cout << "wheel type: horizontal" << std::endl;
    else
        std::cout << "wheel type: unknown" << std::endl;
    std::cout << "wheel movement: " << event.mouseWheelScroll.delta << std::endl;
    std::cout << "mouse x: " << event.mouseWheelScroll.x << std::endl;
    std::cout << "mouse y: " << event.mouseWheelScroll.y << std::endl;
}
```

События MouseButtonPressed и MouseButtonReleased

События `sf::Event::MouseButtonPressed` и `sf::Event::MouseButtonReleased` срабатывают, когда кнопка мыши нажата/отпущена.

SFML поддерживает 5 кнопок мыши: левую, правую, среднюю (колесо мыши), экстра #1 и экстра #2 (кнопки на боковой стороне).

Член класса, ассоциируемый с этим событием, называется `event.mouseButton` и содержит код нажатой/отжатой кнопки, а также позицию курсора мыши.

```
if (event.type == sf::Event::MouseButtonPressed)
{
    if (event.mouseButton.button == sf::Mouse::Right)
    {
        std::cout << "the right button was pressed" << std::endl;
        std::cout << "mouse x: " << event.mouseButton.x << std::endl;
        std::cout << "mouse y: " << event.mouseButton.y << std::endl;
    }
}
```

Событие MouseMoved

Событие `sf::Event::MouseMoved` срабатывает, когда курсор мыши движется внутри окна.

Это событие срабатывает, даже если окно не в фокусе. Однако, срабатывание происходит только когда курсор мыши движется только в пределах внутренней области окна (во внутреннюю облась окна не входят заголов и границы),

Член класса, ассоциируемый с этим событием, называется `event.mouseMove` и содержит позицию курсора мыши относительно окна.

```
if (event.type == sf::Event::MouseMoved)
{
    std::cout << "new mouse x: " << event.mouseMove.x << std::endl;
    std::cout << "new mouse y: " << event.mouseMove.y << std::endl;
}
```

События MouseEntered и MouseLeft

События `sf::Event::MouseEntered` и `sf::Event::MouseLeft` срабатывают, когда курсор мыши входит в облась окна или покидает ее.

Член класса, ассоциируемый с этим событием, не существует.

```
if (event.type == sf::Event::MouseEntered)
    std::cout << "the mouse cursor has entered the window" << std::endl;

if (event.type == sf::Event::MouseLeft)
    std::cout << "the mouse cursor has left the window" << std::endl;
```

События JoystickButtonPressed и JoystickButtonReleased

События `sf::Event::JoystickButtonPressed` и `sf::Event::JoystickButtonReleased` срабатывают, когда кнопка геймпада нажата/отжата.

SFML поддерживает 8 джойстиков и 32 кнопки.

Член класса, ассоциируемый с этим событием, называется `event.joystickButton` и содержит идентификатор джойстика и индекс нажатой/отжатой кнопки.

```
if (event.type == sf::Event::JoystickButtonPressed)
{
    std::cout << "joystick button pressed!" << std::endl;
    std::cout << "joystick id: " << event.joystickButton.joystickId << std::endl;
    std::cout << "button: " << event.joystickButton.button << std::endl;
}
```

Практическая работа №32.
«Объявление класса»

Цель работы: научиться создавать классы, определять данные, методы классов, их свойства, используя язык программирования C#.
Оборудование, технические и программные средства: персональный компьютер, программа браузер.
1. **Теоретические сведения**

Классы — это фундамент языка C#, в нем заключена вся суть объектов. Внутри класса определяются данные и действия, выполняемые над этими данными. Класс — это обертка (шаблон), которая определяет форму объектов. Объекты, в свою очередь, — это экземпляры класса. Определяя класс, мы создаем данные. В данном случае данные — это переменные-экземпляры класса. Над данными должен выполняться определенный код. Реализация кода находится в методах класса.

Класс создаётся с помощью ключевого слова **class**. Давайте напишем общую структуру класса, которая содержит только 2 переменные класса и 2 метода (чтобы было проще понимать, методы — это те же самые функции, но реализованные внутри класса):

```
class имя_класса {
   уровень_доступа тип переменная1;
   уровень_доступа тип переменная2;

   уровень_доступа тип_возвращаемого_значени метод1(параметры) {
      //тело метода
   }
   уровень_доступа тип_возвращаемого_значени метод2(параметры) {
      //тело метода
   }
}
```

Перед каждой переменной и методом есть уровень доступа, который определяет, как к этому члену получить доступ. Существует несколько уровней доступа:
1) **private**. Данный уровень выставлен по умолчанию. Если вы ничего не напишите, то данный член класса будет иметь уровень доступа **private**. Это закрытый уровень доступа. Обращаться к данному члену класса, можно только из самого класса.
2) **public**. Это самый открытый уровень доступа. Члены данного класса могут быть доступны из любой части программы, а не только из класса.
3) **protected**. Это нечто среднее между **public** и **private**. К членам с данным уровнем доступа можно обращаться из самого класса, и из классов-наследников (одна из трех парадигм объектно-ориентированного программирования — это наследование, об этом поговорим в одной из следующих статей).
Давайте создадим простейший класс **Room**, в котором будут находиться данные о длине, ширине и высоте комнаты, а также будет содержаться информация о том, есть ли в ней окна:

```
class Room {
   public double length; //длина комнаты
   public double height; //высота комнаты
   public double width; //ширина комнаты
   public bool windows; //есть ли окна?
}
```

Класс — это нечто абстрактное, он не создает реальность. А вот объект — это уже нечто реальное. Давайте создадим объект класса **Room**:

```
Room myRoom = new Room();
```

Каждый объект класса содержит собственные копии всех переменных-членов класса, поэтому к ним можно обращаться и присваивать различные значения. Давайте зададим значения всем четырем переменным-членам класса:

```
myRoom.length = 7;
myRoom.height = 2.5;
myRoom.width = 4;
myRoom.windows = true;
```

Т.е. мы видим, что обращение к переменным-членам класса идёт через оператор ".". Давайте сообщим пользователю объем комнаты и имеет ли данная комната окна:

```
Console.WriteLine("Объем комнаты равен =
 "+ myRoom.length * myRoom.height * myRoom.width);
if (myRoom.windows) Console.WriteLine("В
комнате имеются окна.");
else Console.WriteLine("В комнате нет окон.");
```

Можно создавать сколько угодно объектов от одного класса. И все эти объекты могут иметь различные значения переменных.

Пока в нашем классе содержатся только переменные-экземпляры. Давайте добавим в него пару методов. Сразу отмечу, что в хорошей программе один метод выполняет только одну задачу. Методу можно присвоить любое имя, за исключением ключевых слов C#, а также **Main()**, так как данный метод служит точкой входа в программу.

1. **Постановка задачи**

Реализуем метод, который возвращает площадь комнаты, а также реализуем метод, который возвращает объём комнаты (так сказать рефакторинг кода):

```
public double SRoom(double x, double y) {
   return x * y;
}
public double VRoom(double x, double y, double z) {
   return x * y * z;
}
```

Полный код программы будет иметь следующий вид:

```
class Room
  {
    public double length;
    public double height;
    public double width;
    public bool windows;

    public double SRoom(double x, double y)
    {
       return x * y;
    }
    public double VRoom(double x, double y, double z)
    {
       return x * y * z;
    }
  }
  class Program
  {
    static void Main(string[] args)
    {
       Room myRoom = new Room();
       myRoom.length = 7;
       myRoom.height = 2.5;
       myRoom.width = 4;
       myRoom.windows = true;
       Console.WriteLine("Объем комнаты равен = " +
myRoom.VRoom(myRoom.length,myRoom.height,myRoom.width));
       Console.WriteLine("Площадь комнаты равна = " + myRoom.SRoom(myRoom.length,
myRoom.width));
       if (myRoom.windows) Console.WriteLine("В комнате имеются окна."); else
Console.WriteLine("В комнате нет окон.");
       Console.ReadLine();
    }
  }
```

2. **Варианты**

1. В класс Room добавьте поле price, задания которое будет отвечать за цену одного квадратного метра.
2. Создайте метод, который будет принимать 2 параметра: общую площадь комнаты и цену. Данный метод должен возвращать стоимость данной комнаты.

3. **Содержание отчета**

1. Название и цель работы.
2. Постановка задачи для конкретного варианта.
3. Алгоритм решения каждой задачи в виде блок-схемы.
4. Программы для решения задач на языке C#.
5. Результаты решения.

5. Контрольные вопросы:

1. Что такое класс?
2. Каким образом создается класс?
3. Какие идентификаторы доступа вам известны? Охарактеризуйте каждый из них.
4. Что такое область видимости? Какова она для элементов класса, помеченных метками public, protected?

Практическая работа №33.

«Создание экземпляров класса»

Цель работы: научиться разрабатывать экземпляры класса, используя язык высокого уровня C# и среду разработки Visual Studio

Оборудование, технические и программные средства: персональный компьютер, программа браузер.

1. **Теоретические сведения**

Класс является обобщенным понятием, определяющим характеристики и поведение некоторого множества конкретных объектов этого класса, называемых экземплярами, или объектами, класса. В целях ускорения работы программиста и упрощения применяемых алгоритмов применяют классы. Применение классов позволяет разбить сложную задачу на ряд простых, взаимосвязанных задач. Проделав эту операцию неоднократно применительно к вновь полученным классам, можно получить алгоритм решения задачи в виде набора простых, понятных, легко осуществимых классов, взаимодействующих друг с другом. «Классический» класс содержит данные, задающие свойства объектов класса, и функции, определяющие их поведение. В последнее время в класс часто добавляется третья составляющая - события, на которые может реагировать объект класса. Это оправдано для классов, использующихся в программах, построенных на основе событийно-управляемой модели, например, при программировании для Windows. Процедура объявления и создания экземпляров пользовательского типа не отличается от таковой для предопределенных типов за исключением необходимости использования Спецификатора New (new) для создания экземпляров пользовательских типов значений и ссылочных типов.

Спецификаторы классов:

new - используется для вложенных классов. Задает новое описание класса взамен унаследованного от предка. Применяется в иерархиях объектов.

public - доступ не ограничен

protected - используется для вложенных классов. Доступ только из элементов данного и производных классов

internal - доступ только из данной программы (сборки)

protected internal - доступ только из данного и производных классов или из данной программы (сборки)

private - используется для вложенных классов. Доступ только из элементов класса, внутри которого описан данный класс

abstract - абстрактный класс. Применяется в иерархиях объектов.

sealed - бесплодный класс. Применяется в иерархиях объектов.

static - статический класс. Введен в версию языка 2.0.

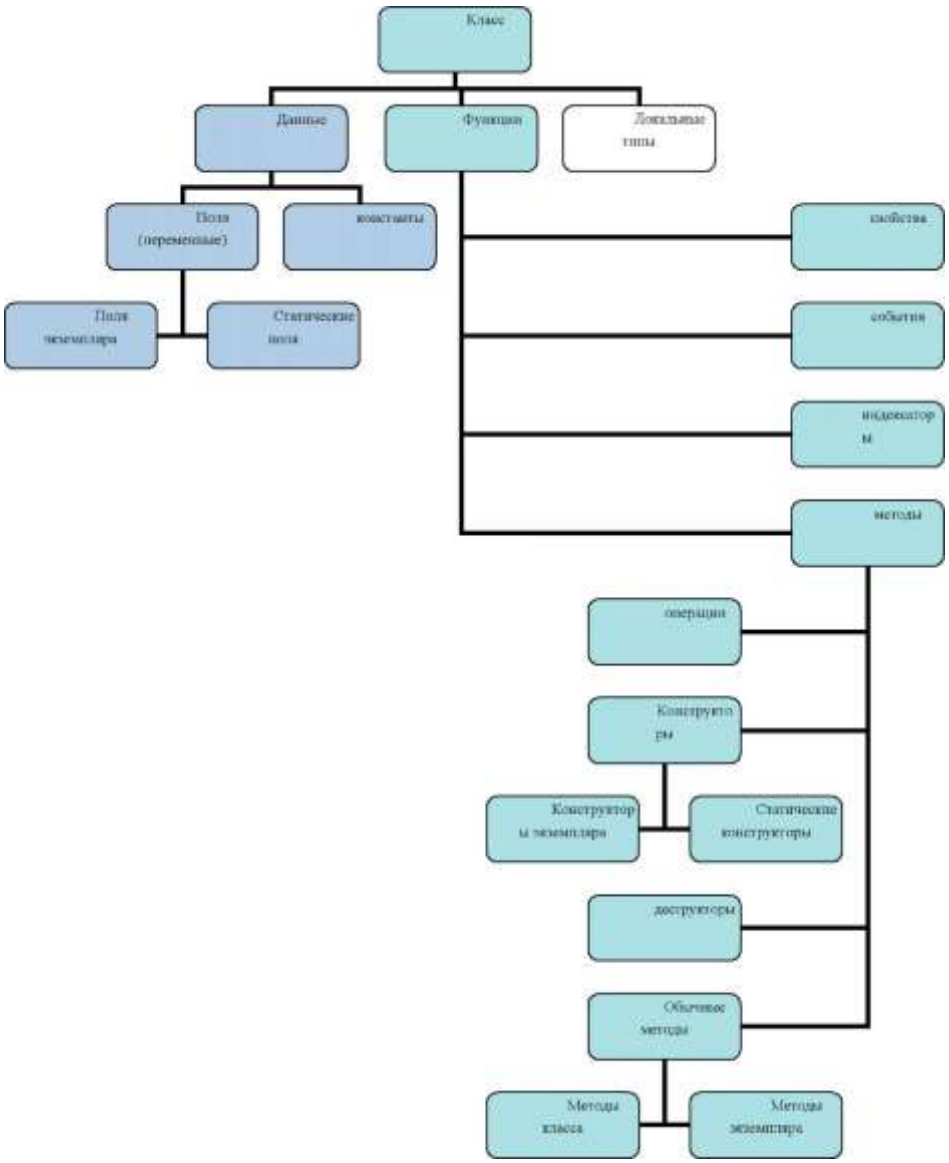
Спецификаторы 2 - 6 называются спецификаторами доступа. Они определяют, откуда можно непосредственно обращаться к данному классу. Спецификаторы доступа могут присутствовать в описании только в вариантах, приведенных в таблице, а также могут комбинироваться с остальными спецификаторами.

Методы делают всю работу, для исполнения которой предназначены классы и структуры: вычисляют значения, изменяют данные, получают вводимую информацию - в общем, выполняют все действия, составляющие поведение объекта

Объявление методов

Один из видов членов, которые добавляются к классам, - это методы, определяющие действия, которые должен выполнять класс. Есть две разновидности методов: те, что возвращают значения, и те, которые никаких значений не возвращают. Следующий пример иллюстрирует обе разновидности методов:

```
// Этот метод возвращает целые (int) значения
public int Add (int first, int second)
{
  int Result;
  Result = first + second;
  Return result;
}
public void myVoidMethod ()
// void - не возвращает значений
{
   MessageBox.Show ("Этот метод не возвращает значения ");
}
```



Вызов методов.

Метод не выполняется, пока его не вызовут. Чтобы вызвать метод, следует указать его имя и задать необходимые ему параметры.

// Так вызывают метод Rotate с двумя параметрами Rotate (45, "Degrees");

Метод Main - особый, он вызывается сразу после начала исполнения программы. К особым методам также относятся деструкторы, которые вызывают во время выполнения непосредственно перед уничтожением объекта. Третий вид особых методов - конструкторы; они исполняются во время инициализации объектов. Метод может иметь один или несколько параметров. Параметр - это аргумент, передаваемый методу вызывающим его методом. При объявлении метода список параметров указывают после его имени в круглых скобках, при этом для каждого параметра необходимо указать его тип.

```
public void DisplayName (string name, byte age)
{
        Console.WriteLine ("Hello " + name + ". You are " +
                age.ToString () +"years old.");
}
```

1. **Постановка задачи**

Написать программу, осуществляющую заданные вычисления с использованием процедур. Вид используемых классов и методов определить самостоятельно.

Пример 1. Задан вектор D(5). Создать класс vect осуществляющий ввод данных с клавиатуры, вывод элементов вектора на консоль, и перестановку первого по порядку элемента со k-тым отрицательным элементом вектора. Ограничение доступа в класс (снаружи можно вызывать только public части класса)

```
using System;
namespace ConsoleApplication1
 {
class Program
 {
  class vect                    // класс векторов vect
   {
     double[ ] a;              // объявление массива int n;
   int n;                      // размер массива
    public vect(int n)           // конструктор создаёт вектор
       {
       this.a = new double[n];
        this.n = n;
     }
      public int vv()           // заполнение вектора целиком,
       // клавиатурный ввод
  {
    for (int i = 0; i < n; i++)
      {
     string buf;               // строка символов
      buf = Console.ReadLine();              // заполнение строки
        a[i] = Convert.ToDouble(buf);              // преобразование строки в число
      }
       return (0);
       }
       public int vyv()                          // вывод на консоль          // элементов вектора
   {
     for (int i = 0; i < n; i++)
     Console.WriteLine("a[" + i + "] = " + a[i]);
   return (0);
      }
        public int perest(int k)             // переставить 1 и k-ое отрицательное число
     {
         // ключ:  0 – если порядок              //          - 1 – если нет
    int k1=0;              // k больше размера вектора
  if  (k >= n)  k1= -1;
      else
      {
        // счетчик отрицательных чисел
    int cnt = 0;                  //  указатель на позицию в массиве
     int pos = -1;
           //  подсчёт отрицательных чисел
          while (cnt < k)
     {
         pos++;
           if (a[pos] < 0) cnt++;
    }
              // подсчёт
            tmp = a[pos] ;               //перестановка
        a[pos] = a[0];
        a[0] = tmp;
 k1=0;
```

```
  }
      return k1;
     }
    //главная программа
   static void Main()
   {
                                    // создали
    vect  d = new vect (5);
      d.vv ();                       //  заполнили
      d.vyv ();                      // вывели на консоль
      d.perest ();                   // переставили
    d.vyv ();                        // вывели на консоль
     }
   }
 } }
```

Пример 2. Задана матрица D(3;3) . Написать метод транспонирования матрицы и метод перемножения квадратных матриц. Получить результат перемножения транспонированной матрицы на соответствующую ей исходную. Отдельный класс для матриц не создаем. Работаем в классе myprog. Поэтому все содержимое класса доступно.

```
using System; // моё пространство имён
namespace ConsoleApplication3
 {
class myprog
 {
 static void Main()
  {
// объявление матриц
    double[,] d = new double[3,3];
   double[,] dt = new double[3, 3];
     double[,] ddt = new double[3, 3];      // заполнение d с клавиатуры
      int i; int j;                // строки, колонки
      for (i = 0; i < 3; i++)
      {
      for (j = 0; j < 3; j++)
        d[i, j] = Convert.ToDouble(Console.ReadLine());
       }
  // траспонирование матрицы
  transp(d, 3, dt);
  // умножение транпонированной       // матрицы dt на исходную d
    mt_q_mult(dt, d,ddt,3);
  }
 // транпонирование
 static void transp(double[,] mtr, int p, double[,] mtrt)
 // double[,] mtr - так определяют матрицу в формальных параметрах
 {
  int i; int j;
   for (i = 0; i < p; i++)
 {
  for (j = 0; j < p; j++)
       mtrt[j, i] = mtr[i, j];
   }
  }
// умножение квадратных матриц
static void mt_q_mult(double[,] m_a, double[,] m_b, double[,] m_r, int p)
  // m_r=m_a * m_b , int p размер матриц
 {
```

```
 int i; int j;
int k; double mel;
  for (i = 0; i < p; i++)                  // строки m_a
    {
      // столбцы m_b
     for (j = 0; j < p; j++)
       {
    mel=0;              // столбцы m_a и строки m_b
   for (k = 0; k < p; k++)
       mel=mel+m_a[i,k]*m_b[k,j];
       m_r[i,j]=mel;
     }
 } }
```

2. **Варианты**

1. Заданы две матрицы A (4, 4) и B (4, 4). Написать программу суммирования двух векторов X и Y, где X - вектор, в котором размещены элементы столбца матрицы A с минимальным средним арифметическим значением, Y - то же для матрицы B.
2. Заданы две матрицы A (4, 4) и B (4, 4). Написать программу вычисления вектора Z = X + Y, где X - строка матрицы A, включающая минимальный элемент ее главной диагонали, Y - то же для матрицы B.
3. Заданы две матрицы B(4,4) и D(3,3). Написать программу транспонирования каждой из заданных матриц с последующим перемножением транспортированной матрицы на соответствующую ей исходную.
4. Заданы два массива C(6) и D(10). Для каждого из них осуществить перестановку первого по порядку элемента со вторым отрицательным элементом массива.
5. Заданы два вектора A(0.052; 0.9; 0.15; 0.84; 0.67) и B(0.948; 0.1; 0.33; 0.16; 0.85). Написать программу, позволяющую расположить элементы одного вектора по возрастанию, а другого - по убыванию. Вычислить сумму полученных векторов.
6. Заданы два двумерных массива A(4, 4) и B(3, 3). Для каждого из них переставить столбцы с максимальным и минимальными элементами.
7. Заданы координаты четырёх точек A, B, C, D на плоскости. Определить наибольший из периметров треугольников ABC, ABD, ACD. Три треугольника заданы координатами своих вершин. Написать программу вычисления площадей треугольников и определения минимальной площади.
8. Заданы матрицы C(4, 4) и D(3, 3). Определить индексы максимального элемента каждой из матриц среди элементов, расположенных выше главной диагонали.
9. Заданы вектора A(5), B(10), D(15). Дли каждого из них определить максимальный и минимальный элементы и их индексы.
10. Заданы массивы Y (4, 4) и Z(8, 8). Для каждого из них вычислить среднее арифметическое значение положительных элементов, расположенных выше главной диагонали.
11. Заданы координаты вершин трех треугольников. Определить треугольник с максимальным периметром.
12. Заданы три матрицы размерами 2x2, 3x3, 4x4. Для каждой из матриц определить среднее арифметическое значение положительных элементов главной диагонали.
13. Заданы одномерные массивы X(5) и Y(7). Для каждого из них определить количество и сумму элементов, которые без остатка делятся на заданное число B.
14. Составить программу заполнения массивов A(5) и B(10) факториалами значений индексов их элементов. Вычисление факториала выполнить в подпрограмме.
15. Заданы матрицы A(4,4), B(6,6). Для каждой матрицы определить среднее арифметическое значение положительных элементов, расположенных не выше главной диагонали.

3. **Содержание отчета**

1. Название и цель работы.
2. Постановка задачи для конкретного варианта.
3. Алгоритм решения каждой задачи в виде блок-схемы.
4. Программы для решения задач на языке C#.
5. Результаты решения.

4. **Контрольные вопросы:**

1. Какие спецификаторы классов вам известны?

2. Что такое метод?
3. Как происходит объявление методов?
4. Как происходит вызов методов?
5. Что относится к особым методам?

Практическая работа №34.

«Создание проекта с использованием класса»

Цель работы: научиться создавать проекты с использованием класса.
Оборудование, технические и программные средства: персональный компьютер, программа браузер.

1. Создайте БД по учету студентов. Используйте механизм наследования. **Сохраните** проект именем **C34PR1.**

```
#include <iostream>
#include <string.h>
using namespace std;
class Subject //класс, описывающий свойства некоторого субъекта
{ protected:
   char name[20]; //имя субъекта
   int age;       //возраст
   char adress[30]; //адрес
  public:
   void Read();     //функция ввода информации о субъекте с клавиатуры
   void Write();    //функция вывода информации о субъекте на экран
};
class Student:public Subject  //класс, описывающий свойства студента
{ char  group[7];   //название группы, в которой учится студент
  char  numb[8];    //номер его зачетной книжки
  int   balls[10];  //оценки, полученные на экзамене
  static int n;     //количество экзаменов в сессию
  protected:
  float rait;      //рейтинг студента (среднее по баллам, полученным на экзаменах)
 public:
```

```
  void Exam();      //функция ввода баллов по предметам
  void CalcRait();  //функция вычисления рейтинга студента
  void ReadSt();    // функция ввода информации о студенте с клавиатуры
  void WriteSt();   // функция вывода информации о студенте на экран
};
class DayStud:public Student //класс, описывающий свойства студента дневной
                             // формы обучения
{int stip;             //стипендия студента
 public:
 void CalcStip();     //функция вычисления стипендии студента
 void WriteSt();      //переопределенная функция вывода информации о студенте
};
// Определение методов классов
void Subject::Read()
{ cout<<" Vvedite informaciu\n  Name";
  cin>>name;
  cout<<"\n Vozrast";
  cin>> age;
  cout<<"\n Adress";
  cin>>adress;
}
void Subject::Write()
{ cout<<" Name  "<<name<<"  Vozrast  "<<age<<"  Adress  "<<adress;}
int Student::n=4;
void Student::ReadSt()
{ Read();
  cout<<"\n Number zachetki";
  cin>>numb;
  cout<<"\n Group";
  cin>>group;
}
void Student::WriteSt()
{ Write();
  cout<<"Number zachetki  "<<numb<<"Group  "<<group<<" Reyting "<<rait<<"\n";
}
void Student::CalcRait()
{ rait=0;
  for(int i=0;i<n;i++)
   rait+=balls[i];
   rait/=n;
}
void Student::Exam()
{ for (int j=0;j<n;j++)
   { cout<<"\n Predmet  N"<<j+1;
     cin>>balls[j];
 }}
```

```
void DayStud::CalcStip()
{if (rait>=90)   stip=300;
 else  if (rait>=76)
   stip=200;
  else   stip=0;
}
void DayStud::WriteSt()
{ Student::WriteSt();
  cout<<"Stipendiya"<<stip;
}
//пример использования определенных выше классов
int main()
{ const int m=5;     //будем работать с 5-ю студентами
  int i;
  DayStud gr[m];
  for(i=0;i<m;i++)
   gr[i].ReadSt();     //вводим информацию о каждом студенте
  for(i=0;i<m;i++)
   { cout<<"Examen"<<i+1<<" student";
     gr[i].Exam();     //проводим экзамены (вводим информацию о баллах, полученных
      //каждым студентом на экзаменах
   }
  for(i=0;i<m;i++)
   gr[i].CalcRait();   //вычисляем рейтинг каждого студента
  for(i=0;i<m;i++)
   gr[i].CalcStip();   //вычисляем стипендию каждого из студентов
  for(i=0;i<m;i++)
   gr[i].WriteSt();    //выводим информацию о каждом студенте на экран
  return 0;
}
```

Контрольные вопросы:

1. Укажите имя базового класса в проекте **C34PR1**.

2. Перечислите свойства базового класса проекта **C34PR1**.

3. Перечислите методы базового класса проекта **C34PR1**.

4. Перечислите классы-потомки и укажите для каждого из них ключ доступа, используемый при наследовании для проекта **C34PR1**.

5. Перечислите свойства классов-потомков проекта **C34PR1**.

6. Перечислите методы, вызываемые от унаследованного класса проекта **C34PR1**.

Практическая работа №35.
«Создание проекта с использованием класса»

Цель работы: научиться создавать проекты с использованием класса.

Оборудование, технические и программные средства: персональный компьютер, программа браузер.

Задание

Создайте базовый класс, классы – наследники (house – дом и office – офис). Используйте механизм publish-наследования. Сохранить проект под именем C34PR2

```
#include <iostream>
using namespace std;
class building
{
protected:
 int floors;
 int rooms;
 double footage;
};
class house: public building
{
 int bedrooms;
 int bathrooms;
public:
 house (int f, int r, double ft, int br, int bth)
 { floors=f; rooms=r; footage=ft; bedrooms=br; bathrooms=bth;}
 void show() {
       cout << "этажей: " << floors << endl;
       cout << "комнат: " << rooms << endl;
       cout << "метраж: " << footage << endl;
       cout << "спален: " << bedrooms << endl;
       cout << "ванн: " << bathrooms << endl;
 }
};
class office: public building
{
 int phones;
 int extinguishers;
public:
 office (int f, int r, double ft, int p, int ext)
 {floors=f; rooms=r; footage=ft; phones=p; extinguishers=ext;}
void show (){
     cout << "этажей: " << floors << endl;
       cout << "комнат: " << rooms << endl;
       cout << "метраж: " << footage << endl;
       cout << "телефонов: " << phones << endl;
       cout << "огнетушителей: " << extinguishers << endl;
```

```
}
};
void main()
{
	setlocale (LC_ALL, "RUS");
	house h_ob(2, 12, 5000, 6, 4);
	office o_ob(4, 25, 12000, 30, 8);
	cout << "Жилой дом " << endl;
	h_ob.show();
	cout << "Офис " << endl;
	o_ob.show();
}
```