УДК 681.31
ББК 32.973
О-23

Авторы-составители: С. М. Мовшович, канд. техн. наук, доцент;
О. А. Кравченко, канд. физ.-мат. наук, доцент

Рецензенты: Е. А. Храбров, канд. техн. наук, доцент кафедры «Промышленная электроника» Гомельского государственного технического университета им. П. О. Сухого;
Т. В. Астапкина, ст. преподаватель кафедры информационно-вычислительных систем Белорусского торгово-экономического университета потребительской кооперации

Рекомендовано к изданию научно-методическим советом учреждения образования «Белорусский торгово-экономический университет потребительской кооперации». Протокол № 2 от 14 декабря 2010 г.

О-23 **Обработка** экономической информации в системе программирования DELPHI : пособие по дисциплине «Алгоритмизация и программирование» для студентов специальности 1-26 03 01 «Управление информационными ресурсами» / авт.-сост. : С. М. Мовшович, О. А. Кравченко. – Гомель : учреждение образования «Белорусский торгово-эко-номический университет потребительской кооперации», 2011. – 128 с.
ISBN 978-985-461-828-9

УДК 681.31
ББК 32.973

ISBN 978-985-461-828-9

торгово-экономический университет
потребительской кооперации», 2011

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Настоящее пособие предназначено для ознакомления студентов с такими функциями системы программирования (СП) Delphi, которые позволяют создавать практически значимые проекты.

В разделе «Теоретические основы курса» рассмотрены следующие темы:

- Программное управление объектами в окне формы.
- Создание многомодульных проектов.
- Использование иерархического меню.
- Создание многостраничных форм с вкладками.
- Методы сортировки информации.
- Обработка упорядоченных массивов.
- Обработка символьной информации.
- Массивы записей.
- Использование типизированных файлов.
- Использование текстовых файлов.
- Подпрограммы пользователя.

Все сведения, методы и приемы, рассмотренные в перечисленных темах, используются в лабораторных работах, изложенных в разделе «Задания лабораторных работ».

Данное пособие предполагает, что студенты освоили основы программирования в системе программирования Delphi и приобрели умения и навыки создания простых проектов, изложенных в пособии Алгоритмизация и программирование.

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ КУРСА

1. ПРОГРАММНОЕ УПРАВЛЕНИЕ ОБЪЕКТАМИ В ОКНЕ ФОРМЫ

1.1. Скрытие и показ объектов

В СП Delphi есть эффективные средства скрытия и показа (восстановления) объектов в окне формы при исполнении проекта. Для этого используются события *Hide* (скрыть) и *Show* (показать), а также свойство *Visible* (визуализация) со значением *True* или *False*.

При установке значения *False* для свойства *Visible* объект скрывается. При установке значения *True* объект показывается в окне формы.

Таким образом, эквивалентные операторы *Image1.Show;* и *Image1.Visible:=True;* приводят к появлению объекта *Image*1 в окне формы, а эквивалентные операторы *Image1.Hide;* и *Image1.Visible:=False;* скрывают объект *Image1*.

1.2. Очистка результирующих полей и полей исходных данных

Если в проекте реализована только одна процедура обработки события (щелчок по командной кнопке), то при выполнении данного проекта при задании новых исходных данных в полях, предназначенных для вывода результатов, остается результат предыдущего исполнения, т. е. старое значение остается в результирующем поле до щелчка по кнопке, с которой связана процедура вычисления искомых значений.

Кроме того, в задачах обработки массивов присутствует дополнительная неприятность, обусловленная тем, что исходные данные связаны между собой, т. е. после задания количества элементов массива в таблицу строк надо ввести ровно столько же значений. В проектах, реализованных традиционными способами, при вводе нового количества элементов в таблице строк продолжают находиться данные предыдущего исполнения.

Рассмотрим приемы, позволяющие избавиться от указанных недостатков.

Очистка результирующих полей может быть проведена следующим образом:

- Выделите одно из полей исходных данных (например, *Edit1*).

• На вкладке *Events* (события) инспектора объектов сделайте двойной щелчок справа от события *OnClick*, в результате чего откроется заготовка процедуры-обработчика щелчка по рассматриваемому полю (в заголовок процедуры будут входить слова *procedure TForm1.Edit1Click*).

• В заготовку процедуры обработки щелчка по полю *Edit1* следует записать операторы, заполняющие результирующие поля пустыми строками.

• Для остальных полей исходных данных надо указать для события *OnClick* процедуру *Edit1Click* путем выбора этого имени в раскрывающемся списке справа от имени события *OnClick*.

Аналогичным образом создается процедура обработки события *OnChange*, возникающего при изменении значения поля *Edit*.

В качестве примера рассмотрим задачу, интерфейс которой приведен на рисунке 1.

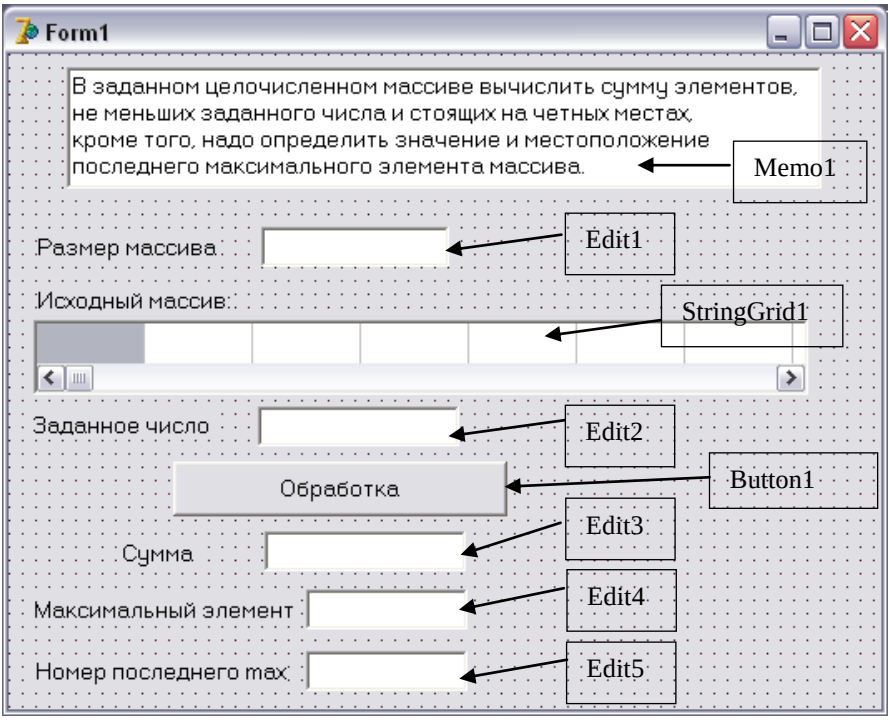


Рисунок 1 – **Интерфейс задачи**

В поля *Edit1* и *Edit2* и в таблицу *StringGrid1* вводят исходные данные, а результаты выводятся в поля *Edit3, Edit4* и *Edit5*. Основная процедура решения задачи выполняется при щелчке по кнопке *Button1*. Если при повторном исполнении проекта выполняется щелчок по полю *Edit1*, то это является сигналом для очистки полей результатов. Изменение значения в поле *Edit1* (событие *OnChange*) должно инициировать не только чистку полей *Edit3, Edit4* и *Edit5*, но и чистку ячеек таблицы *StringGrid1*.

На рисунке 2 приведен текст процедуры обработки щелчка по полю *Edit1*, в которой происходит чистка полей *Edit3, Edit4* и *Edit5*. На этом же рисунке приведен текст процедуры, вызываемой при изменении значения в поле *Edit1* и выполняющей чистку полей *Edit3, Edit4* и *Edit5*, а также ячеек таблицы *StringGrid1*.

```
procedure TForm1.Edit1Click(Sender: TObject);
begin
  Edit3.Text:='';
  Edit4.Text:='';
  Edit5.Text:='';
end;

procedure TForm1.Edit1Change(Sender: TObject);
var
  i:integer;
begin
  Edit3.Text:='';
  Edit4.Text:='';
  Edit5.Text:='';
  for i:=1 to StringGrid1.ColCount do
    StringGrid1.Cells[i-1,0]:='';
end;
```

Рисунок 2 – **Тексты процедур чистки полей формы**

Повторное исполнение проекта может происходить при неизменных значениях элементов массива (поле *Edit1*, ячейки *StringGrid1*) и новом значении поля *Edit2*. Следовательно, при щелчке по полю *Edit2* или при

изменении значения в этом поле надо также очищать поля результатов. Для этого не надо создавать еще одну процедуру, а можно сослаться на процедуру *procedure TForm1.Edit1Click*. Если при повторном исполнении проекта изменяются только элементы массива (ячейки *StringGrid1*), то необходимо очистить поля результатов. Значит, и в этом случае надо сослаться на имеющуюся процедуру *procedure TForm1.Edit1Click*.

Таким образом, для событий *Edit2.Click*, *Edit2.Change* и *StringGrid1.Click* (события *OnChange* для *StringGrid* не существует) необходимо сослаться на ранее созданную процедуру *procedure TForm1.Edit1Click*.

Указанная ссылка реализуется следующим образом: в окне формы выделяется объект *Edit2*, а в инспекторе объектов на вкладке *Events* справа от наименования события *OnClick* раскрывается список и выбирается наименование процедуры *Edit1Click* (рисунок 3). Обратите внимание, что в списке отображаются все процедуры модуля. Аналогичные действия производятся для событий *Edit2.Change* и *StringGrid1.Click*.

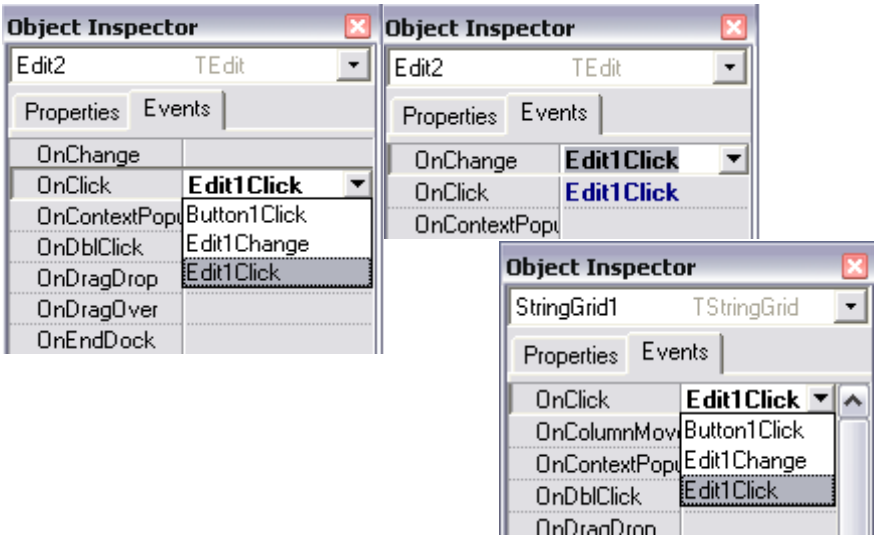


Рисунок 3 – **Выбор процедуры обработки события**

Очистка результирующих полей и полей исходных данных может быть связана со специальной кнопкой. В качестве примера добавим в окно формы, приведенной на рисунке 1, кнопку *Button2* с заголовком *Новая задача*, как это показано на рисунке 4. Процедура обработки щелчка по этой кнопке должна очищать содержимое всех объектов

формы, приводя окно приложения в первоначальное состояние. Текст данной процедуры приведен на рисунке 5.

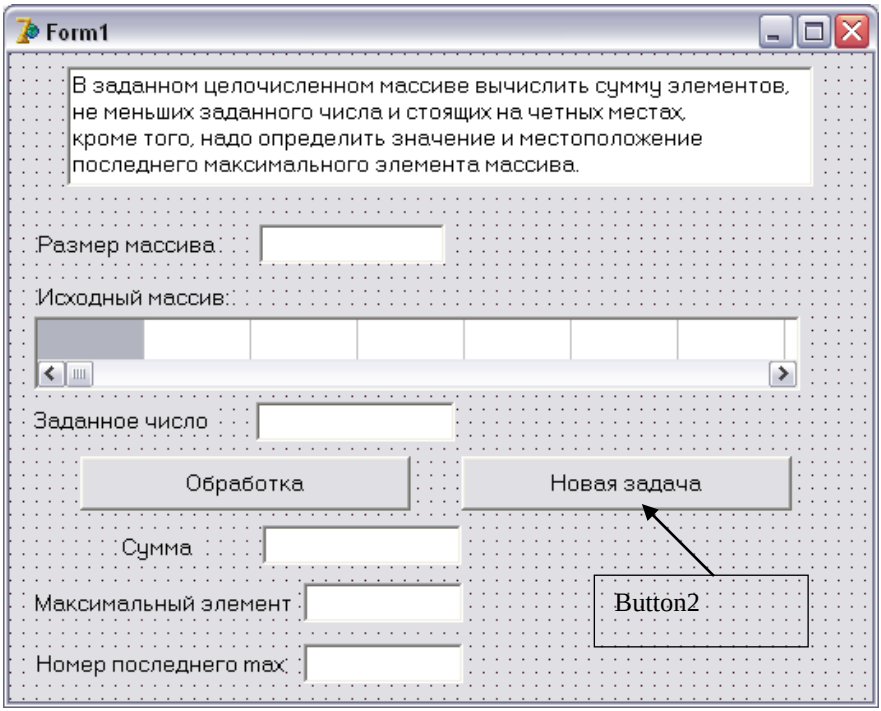


Рисунок 4 – **Интерфейс задачи с дополнительной кнопкой**

```
procedure TForm1.Button2Click(Sender: TObject);
var
  i:integer;
begin
  Edit1.Text:='';
  Edit2.Text:='';
  Edit3.Text:='';
  Edit4.Text:='';
  Edit5.Text:='';
  for i:=1 to StringGrid1.ColCount do
    StringGrid1.Cells[i-1,0]:='';
end;
```

Рисунок 5 – **Процедура очистки всех полей формы**

1.3. Управление объектами при активации формы

С объектом *Form* связано событие *onActivate*, наступающее при запуске проекта на выполнение. Как правило, в проект включают процедуру обработки этого события, которая выполняет следующие функции:

- установка начальных значений для свойств объектов формы;
- скрытие и (или) показ объектов формы;
- размещение в некотором поле мигающего курсора, приглашающего пользователя начинать ввод исходных данных именно с данного поля.

Размещение курсора в нужном объекте в режиме выполнения производится с помощью метода *SetFocus*. Например, оператор *Edit1.SetFocus* приводит к установке курсора в поле *Edit1*. Заметим, что в режиме проектирования для установки курсора в поле *Edit1* можно либо задать значение 0 для свойства *TabOrder* объекта *Edit1*, либо в свойстве *ActiveControl* для формы *Form1* раскрыть список справа от названия свойства и выбрать объект *Edit1*.

Пример. Определить размер платы за гостиницу при условии, что за одноместный номер оплата составляет 100% от базовой величины, двухместный – 75, трехместный – 50%. При этом для ветеранов войны делается скидка в размере 15%, а в случае предоплаты – дополнительная скидка 10%. Интерфейс данной задачи представлен на рисунке 6.

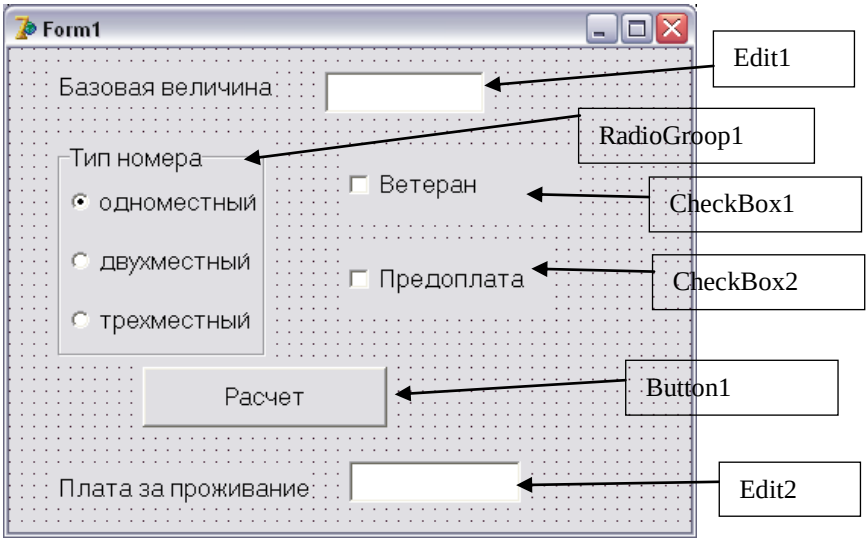


Рисунок 6 – **Интерфейс задачи**

После запуска проекта на выполнение целесообразно, чтобы курсор оказался в поле *Edit1*, флажки *CheckBox1* и *CheckBox2* должны быть неактивны, а в группе переключателей *RadioGroup1* один из перелючателей (например первый) должен быть обязательно активен. Можно также очистить поля *Edit1* и *Edit2*, в качестве страховки от недоработок режима проектирования. Текст данной процедуры приведен на рисунке 7.

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  Edit1.Text:='';
  Edit2.Text:='';
  Edit1.SetFocus;
  CheckBox1.Checked:=False;
  CheckBox2.Checked:=False;
  RadioGroup1.ItemIndex:=0;
end;
```

Рисунок 7 – **Процедура, вызываемая при активации *Form1***

2. СОЗДАНИЕ МНОГОМОДУЛЬНЫХ ПРОЕКТОВ

2.1. Включение в проект новой формы

Все рассмотренные до сих пор проекты использовали только одну форму. В данном разделе приведена методика создания второй формы проекта, а, следовательно, и включения в проект второго модуля, содержащего процедуры обработки событий для объектов второй формы.

Действия по включению в существующий проект второй формы можно разделить на 6 шагов.

1. В форме *Form1* должна находиться кнопка, при щелчке по которой будет открываться окно второй формы *Form2*. Для определенности назовем эту кнопку *Button3*.

2. В процедуру-обработчика щелчка по кнопке *Button3* вставим единственный оператор *Form2.Show*, выполнение которого приведет к выводу на экран второй формы. Текст данной процедуры приведен на рисунке 8.

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  Form2.Show
end;
```

Рисунок 8 – **Процедура вызова формы *Form2***

3. Теперь создаем саму форму *Form2*. Для этого выполняем команду *File / New Form*. На экране появится пустое окно формы, которому по умолчанию присвоится имя *Form2*.

4. Заполняем новую форму необходимыми по условию задачи визуальными компонентами. Необходимым компонентом формы *Form2* является кнопка (для определенности назовем ее *Button1*), с которой связывается процедура возврата к форме *Form1*.

5. В процедуру обработки щелчка по кнопке *Button1* из формы *Form2* надо вставить оператор *Close*, который обеспечивает закрытие окна формы *Form2* и автоматическое появление окна формы *Form1*. Таким образом был обеспечен переход от формы *Form1* к форме *Form2* и возврат назад. Текст указанной процедуры приведен на рисунке 9.

```
procedure TForm2.Button1Click(Sender: TObject);
begin
  Close
end;
```

Рисунок 9 – **Процедура закрытия формы *Form2***

6. Следует подключить форму *Form2* к существующему проекту, что можно выполнить несколькими способами:

• *Первый способ* предполагает запуск проекта на исполнение (клавиша F9). В результате появится окно системных сообщений, представленное на рисунке 10. Смысл сообщения в этом окне можно перевести следующим образом: «в форме *Form1* имеется ссылка на форму *Form2*, описанную в модуле *Unit2*, который не использовался ранее. Добавить его сейчас?».

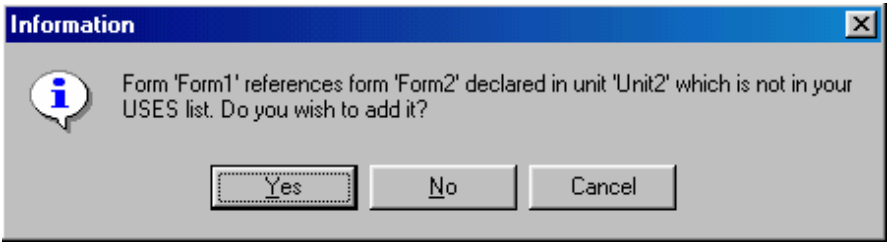


Рисунок 10 – **Сообщение о возможности подключения к проекту нового модуля**

После щелчка по кнопке *Yes* в тексте модуля *Unit1* после строки *implementation* вставится строка *uses Unit2;*.

• *Второй способ* подключения формы *Form2* к существующему проекту заключается в самостоятельном написании строки *uses Unit2;* после строки *implementation.*

• *Третий способ* – выполнение команды *File/Use Unit…* . В открывшемся окне со списком неподключенных модулей следует выбрать модуль *Unit2* и щелкнуть по кнопке *OK*. Результат данного способа тот же – появление строки *uses Unit2;* в тексте модуля *Unit1* после строки *implementation*.

После окончания шестого шага проект с двумя формами готов к исполнению.

Аналогичным образом в проект можно включить произвольное число форм и модулей.

2.2. Структура модулей

Для организации обмена информацией между модулями надо знать структуру модуля.

Любой модуль имеет следующую структуру:
Unit *имя модуля;*
interface
интерфейсная часть
implementation
исполняемая часть
initialization
инициализирующая часть
finalization
завершающая часть
end.

Каждая из четырех составных частей модуля может быть пустой и при этом опущена.

По умолчанию модули получают имена *Unit1, Unit2* и т. д. Текст соответствующего модуля располагается в файле *Unit1.pas, Unit2.pas* и т. д.

Имя модуля служит для его связи с другими модулями, которая устанавливается специальной инструкцией:

uses *список модулей*;

В модуле эта инструкция должна следовать либо сразу за словом *interface*, либо сразу за словом *implementation*, либо располагаться в обоих случаях, т. е. в модуле допускаются две инструкции *uses*.

В *интерфейсной части* модуля находятся описания типов и переменных, которые должны стать доступными для основной программы и возможно для других модулей. Если, например, в модуле *Unit1* находится описание *var Kol:integer*, то для использования этой переменной в модуле *Unit2* необходимо написать в модуле *Unit2* инструкцию *uses Unit1*.

Исполняемая часть модуля содержит описания процедур, объявленных в интерфейсной части. Все объекты, описанные после заголовка процедуры, можно использовать только в теле этой процедуры.

В *инициализирующей части* модуля размещаются операторы, которые исполняются до передачи управления основной программе и обычно используются для задания начальных значений глобальных переменных, т. е. общих для нескольких модулей. Если несколько модулей имеют инициализирующие части, то эти части выполняются последовательно друг за другом в порядке перечисления модулей в инструкции *uses* главной программы.

В *завершающей части* модуля указываются операторы, выполняющиеся после завершения работы основной программы. Если несколько модулей имеют завершающие части, то эти части выполняются последовательно друг за другом в порядке, обратном перечислению модулей в инструкции *uses* главной программы.

3. ИСПОЛЬЗОВАНИЕ ИЕРАРХИЧЕСКОГО МЕНЮ

Визуальный компонент Главное меню (*MainMenu*) находится на странице *Standart* и предназначен для создания в окне формы иерархического меню, с помощью команд которого можно вызывать процедуры проекта. В проектах, выполняющих много функций, использование иерархического меню более удобно для пользователя, чем множество командных кнопок.

После установки компонента в окно формы необходимо создать пункты меню. Для этого надо либо дважды щелкнуть по компоненте левой кнопкой мыши, либо щелкнуть справа от свойства *Items* в окне инспектора объектов. В результате откроется окно конструктора меню. Ввод пунктов меню производится следующим образом: в строке свойства *Caption* инспектора объектов следует набрать название команды и нажать *Enter*, после чего справа возникнет область (пустой прямоугольник) для следующего элемента меню. Затем процесс следует повторить.

Элементы главного меню автоматически получают имена *N1, N2* и т. д.

Каждый пункт главного меню может раскрываться в подменю или являться конечной командой. Для создания пунктов подменю надо

щелкнуть кнопкой мыши ниже пункта меню и ввести имя первой команды подменю. После нажатия *Enter* возникает прямоугольник для ввода следующего пункта подменю. При щелчке справа в верхней строке можно продолжать ввод пунктов главного меню.

На рисунке 11 представлены фрагменты окон, иллюстрирующих процесс создания иерархического меню.

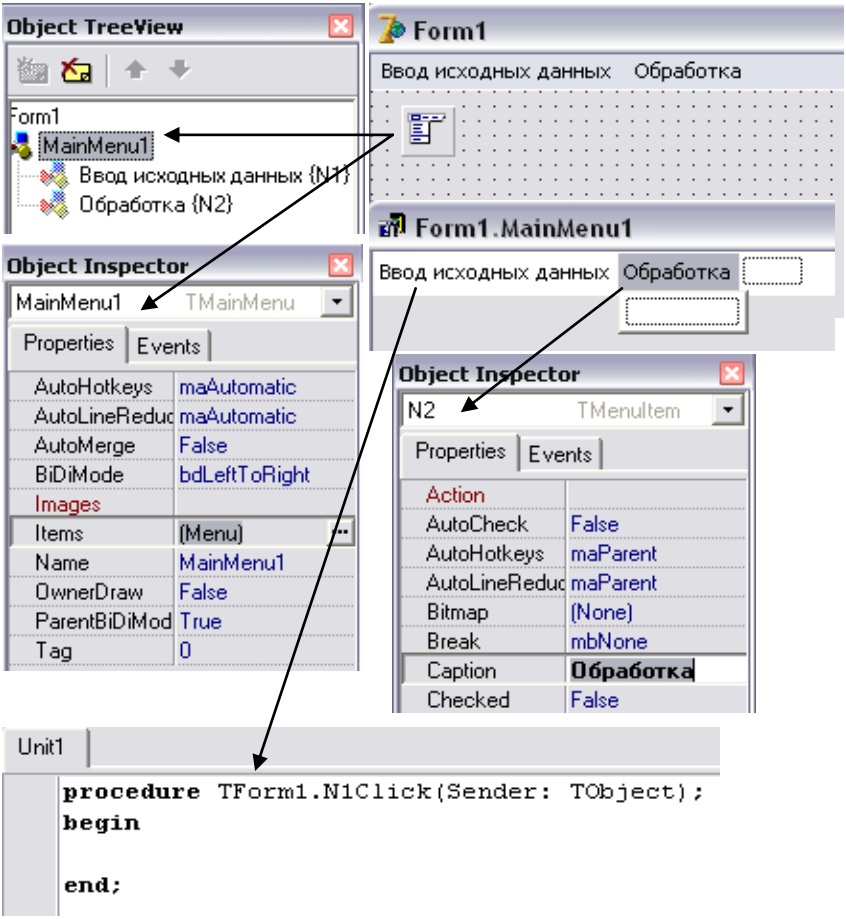


Рисунок 11 – **Технология создания иерархического меню**

В режиме проектирования при щелчке по команде меню в окне редактора кода создается заготовка процедуры, пример которой приведен на рисунке 11. В режиме исполнения проекта щелчок по команде меню приводит к выполнению соответствующей процедуры.

4. СОЗДАНИЕ МНОГОСТРАНИЧНЫХ ФОРМ С ВКЛАДКАМИ

Визуальный компонент *PageControl* позволяет создавать несколько перекрывающих друг друга страниц, каждая из которых относится к одной вкладке. Все страницы относятся к одной форме. Совокупность вкладок располагается в той же форме. Компонент находится на странице Win32 палитры компонентов.

После размещения компонента вызывается его контекстное меню и в нем выбирается команда *New Page*. В свойстве *Caption* набирается название вкладки. Указателем мыши раздвигается граница рабочей зоны страницы до нужных размеров и в нее помещаются необходимые визуальные компоненты, как в окно формы. Затем вновь из контекстного меню выбирается команда *New Page* и создается страница для второй вкладки.

Созданные страницы получают имена *TabSheet1*, *TabSheet2* и т. д. Заметим, что размеры рабочих зон всех страниц совпадают. Визуальные компоненты всех страниц относятся к одной форме. Технология создания вкладок приведена на рисунке 12.

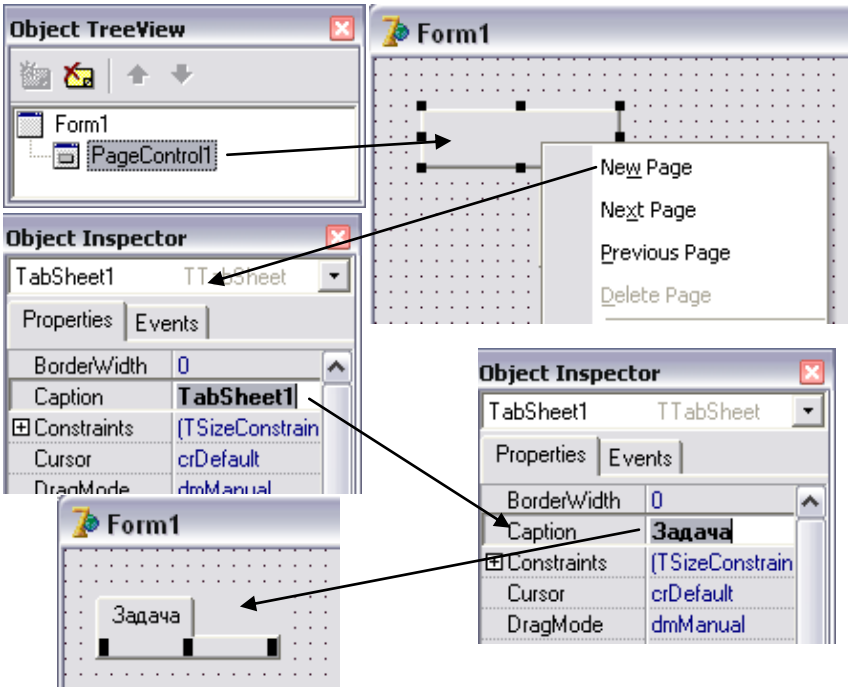


Рисунок 12 – **Технология создания вкладок**

Визуальный компонент *PageControl* обладает следующими свойствами:

- *HotTrack* (при значении *True* название вкладки автоматически выделяется цветом, когда на ней находится указатель мыши).
- *Multiline* (при значении *True* вкладки можно располагать в несколько рядов).
- *TabHeight* (определяется высота каждой вкладки, если значение равно 0, то высота выбирается автоматически).
- *TabWidth* (определяется ширина каждой вкладки, если значение равно 0, то ширина выбирается автоматически).
- *TabPosition* (положение зоны вкладок относительно рабочей зоны компонента: *tpTop* – вверху, *tpBottom* – внизу).
- *ActivePage* (ссылка на активную страницу: *TabSheet1, TabSheet2,…*).
- *TabIndex* (номер выбранной страницы, нумерация начинается с 0).

Далее рассмотрим пример использования вкладок.

Пример. С помощью визуального компонента *PageControl* создадим три вкладки, которые соответствуют страницам *TabSheet1* (Задача), *TabSheet2* (Об авторе) и *TabSheet3* (Справка) соответственно.

На рисунках 13 и 14 приведены страницы формы с тремя вкладками и окно дерева объектов.

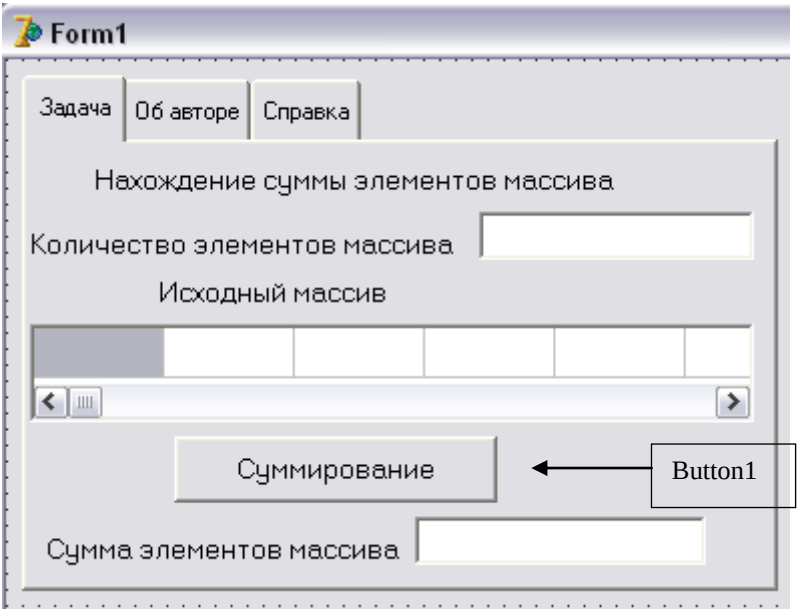


Рисунок 13 – **Страница вкладки *Задача* (*TabSheet1*)**

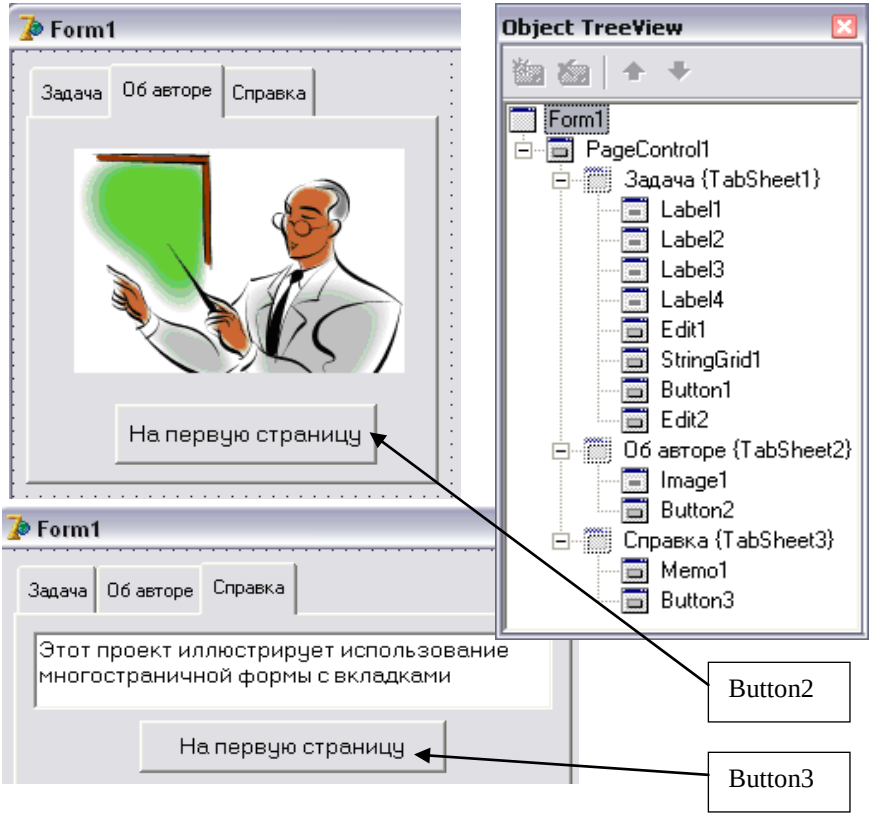


Рисунок 14 – **Страницы вкладок *Об авторе* (*TabSheet2*) и *Справка* (*TabSheet3*) и окно дерева объектов**

В режиме проектирования щелчок по вкладке отображает соответствующую страницу. Таким же образом пользователь активизирует страницы и во время выполнения проекта. Программным путем активизация страницы производится оператором *TabSheet1.Show*.

После запуска проекта для появления первой страницы можно в процедуру обработки события *FormActivate* включить оператор *TabSheet1.Show*.

Для возврата на первую страницу после работы на второй и третьей страницах в рабочие зоны этих страниц помещены командные кнопки *Button2* и *Button3*, а в процедуры обработки щелчка по этим кнопкам включен оператор *TabSheet1.Show*.

Тексты процедур проекта приведены на рисунке 15.

```
procedure TForm1.Button1Click(Sender: TObject);
{Решение задачи на странице TabSheet1}
 var
  i,n:integer;
  a:array[1..10] of integer;
  S:integer;
begin
   n:=StrToInt(Edit1.Text);
   for i:=1 to n do
     a[i]:=StrToInt(StringGrid1.Cells[i-1,0]);
   S:=0;
   for i:=1 to n do
     S:=S+a[i];
   Edit2.Text:=IntToStr(S)
end;

procedure TForm1.Button2Click(Sender: TObject);
{Переход со страницы TabSheet2 на страницу
 TabSheet1}
begin
  TabSheet1.Show
end;

procedure TForm1.Button3Click(Sender: TObject);
{Переход со страницы TabSheet3 на страницу
 TabSheet1}
begin
  TabSheet1.Show
end;

procedure TForm1.FormActivate(Sender: TObject);
{При активизации формы активизируется страница
 TabSheet1, соответствующая вкладке Задача}
begin
  TabSheet1.Show
end;
```

Рисунок 15 – **Тексты процедур проекта**

5. МЕТОДЫ СОРТИРОВКИ ИНФОРМАЦИИ

5.1. Общие сведения о сортировке

Сортировка – это операция, упорядочивающая последовательность (массив) элементов по ключам.

Ключ – это некоторое числовое свойство элемента массива, т. е. каждому элементу массива ставится в соответствие некоторое число, называемое ключом элемента.

Если есть числовой массив, то ключ элемента – это сам элемент. В этом случае сортировка массива – это упорядочивание массива (перестановка его элементов) таким образом, чтобы получилась неубывающая или невозрастающая последовательность.

Если каждый элемент исходного массива представляет собой слова, составленные из букв русского алфавита, то ключ, по которому может быть упорядочен массив, связывается с порядковым номером буквы в алфавите. По этому принципу упорядочиваются слова в словарях – из двух слов первым помещается то слово, ключ которого меньше. Следует учесть, что отсутствие буквы (т. е. пустая строка) имеет меньший ключ, чем любая другая буква. Так, слово «студент» помещается в словаре перед словом «студентка».

Если слова состоят из букв разных алфавитов и цифр, как, например, имена файлов и папок, то любая цифра имеет меньший ключ, чем любая буква, а любая латинская буква имеет меньший ключ по сравнению с любой русской буквой. При сортировке таких имен в качестве ключа используется код символа в некоторой кодовой таблице. По этому принципу, например, отсортированы имена встроенных функций MS Excel в категории «*Полный алфавитный перечень*». Этот принцип упорядочивания еще называют лексикографическим порядком, или расширенным алфавитом.

Разработкой различных алгоритмов сортировки информации, хранящейся в оперативной памяти компьютера или на его жестком диске, программисты занимаются уже давно. Интерес к этой проблеме обусловлен тем, что по мнению специалистов 25% всего времени обработки информации расходуется на сортировку данных, так как с отсортированными данными работать легче, чем с произвольно расположенными. Когда элементы отсортированы, то проще найти нужный элемент или установить, что его нет.

Если элементы массива связаны отношениями $a_1 < a_2 < \ldots < a_{n-1} < a_n$, то говорят, что массив упорядочен по *возрастанию*. Такая упорядоченность предполагает, что в массиве нет одинаковых элементов.

Если элементы массива связаны отношениями $a_1 \le a_2 \le \ldots \le a_{n-1} \le a_n$, то говорят, что массив упорядочен по *неубыванию*. Такая упорядоченность не исключает наличия в массиве одинаковых элементов.

Если элементы массива связаны отношениями $a_1 > a_2 > \ldots > a_{n-1} > a_n$, то говорят, что массив упорядочен по *убыванию*. Такая упорядоченность предполагает, что в массиве нет одинаковых элементов.

Если элементы массива связаны отношениями $a_1 \ge a_2 \ge \ldots \ge a_{n-1} \ge a_n$, то говорят, что массив упорядочен по *невозрастанию*. Такая упорядоченность не исключает наличия в массиве одинаковых элементов.

5.2. Классификация методов сортировки

Все методы сортировки можно разделить на пять групп:

- методы извлечения;
- методы включения;
- методы обменов;
- методы слияния;
- методы распределения.

Общая концепция *методов извлечения* заключается в следующем: из исходного массива извлекается минимальный элемент и меняется местами с первым элементом массива, затем извлекается минимальный элемент из части массива, начиная со второго элемента, и меняется местами со вторым элементом и т. д. Последний раз минимальный элемент выбирается из двух последних элементов массива. В результате получается массив, упорядоченный по неубыванию.

Различные методы извлечения отличаются объектом извлечения (минимальный или максимальный элемент) и, соответственно, объектами перестановки (первый или последний элемент), а также условием окончания процесса сортировки.

Идея *методов включения* заключается в том, что сначала первый элемент массива рассматривается как упорядоченный массив и в этот массив включается следующий элемент исходного массива так, чтобы получился упорядоченный по неубыванию массив из двух элементов. Затем в полученный упорядоченный массив включается третий элемент массива так, чтобы опять-таки получился упорядоченный массив. Процесс продолжается до тех пор, пока не будет включен последний элемент.

Различные алгоритмы включения отличаются способами выбора элемента для включения, определения места включения, а также методами организации самого процесса включения.

Идея *методов обменов* заключается в том, что в исходном массиве выбирается пара элементов, которые сравниваются между собой. Если их положение не удовлетворяет требованию упорядоченности, то элементы переставляются. Затем выбирается следующая пара элементов и так до тех пор, пока не получится упорядоченный массив.

Различные алгоритмы метода обменов отличаются способами выбора пары элементов для сравнения и перестановки, а также условиями окончания процесса сортировки.

Метод слияния применяется в том случае, когда имеются два (или больше) упорядоченных массивов и требуется соединить исходные массивы в один общий упорядоченный массив.

Метод распределения употребляется в тех случаях, когда в исходном массиве имеется известное заранее количество различных ключей (значений), например, список студентов с оценками по десятибалльной системе, полученными на экзамене. Заранее известно, что оценки могут быть 10, 9, … 1. Поэтому для упорядочения массива по невозрастанию можно сначала выбрать все записи с оценками 10, затем с оценками 9, 8 и, в последнюю очередь, 1.

5.3. Создание проекта на основе сортировки методом извлечения

В данном пункте представлен пример упорядочения заданного целочисленного массива по *неубыванию* на основе алгоритма *извлечения минимального* элемента.

Интерфейс искомого проекта приведен на рисунке 16.

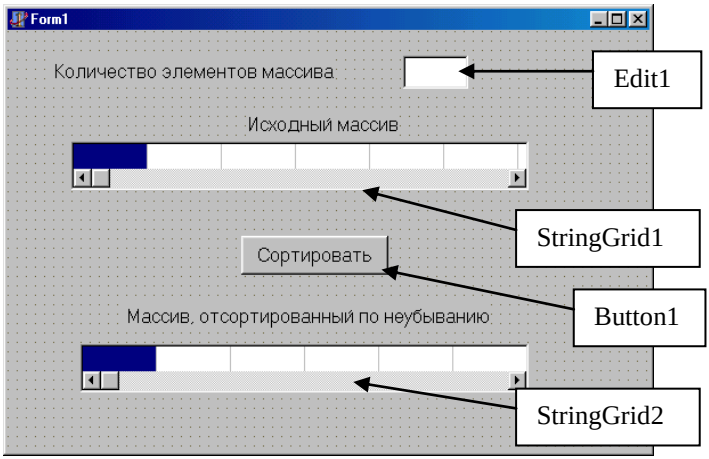


Рисунок 16 – **Интерфейс проекта сортировки массива по неубыванию**

Для создания процедуры обработки щелчка по кнопке *Сортировать* следует разработать алгоритм сортировки на основе извлечения минимального элемента. Сначала нужно записать необходимые действия в словесной форме следующим образом:

1. Найдем минимальный элемент среди всех элементов массива $a_1, a_2, \ldots, a_{n-1}, a_n$ и определим его номер. Пусть это будет элемент a_m.

2. Поменяем местами элементы a_1 и a_m. Таким образом, минимальный элемент массива окажется на своем окончательном месте.

3. Найдем минимальный элемент среди элементов массива, начиная со второго $a_2, a_3, \ldots, a_{n-1}, a_n$, и определим его номер. Опять обозначим этот элемент как a_m.

4. Поменяем местами элементы a_2 и a_m. В результате два первых элемента массива окажутся на своих окончательных местах.

5. На заключительном этапе процесса сортировки надо выбрать минимальный элемент из двух последних элементов массива a_{n-1} и a_n. После чего этот элемент должен быть поставлен на предпоследнее место.

Теперь обобщим представленные шаги алгоритма следующим образом: нахождение минимального элемента a_m среди элементов $a_k, a_{k+1}, \ldots, a_{n-1}, a_n$ и последующая перестановка элементов a_m и a_k, при этом указанный процесс должен повторяться при изменении k от 1 до n–1.

Сформулированному алгоритму соответствует графическая схема алгоритма, приведенная на рисунке 17.

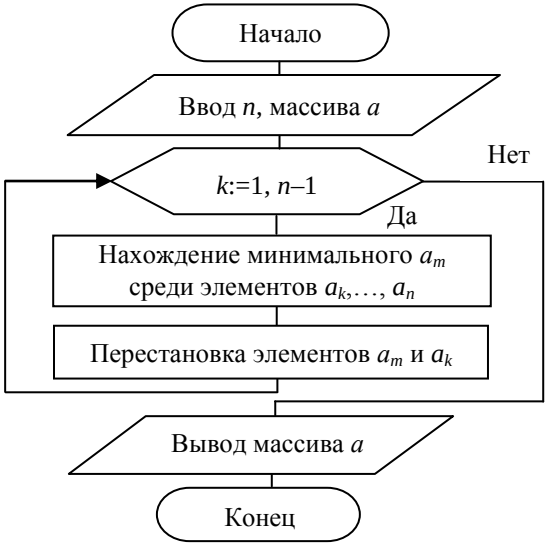


Рисунок 17 – **Схема алгоритма сортировки массива по неубыванию методом извлечения**

На основании приведенной схемы алгоритма запишем текст процедуры обработки щелчка по кнопке *Сортировать* (рисунок 18).

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    a:array [1..20] of integer;
    i,k,n,m,b,c:integer;
begin
  n:=StrToInt(Edit1.Text);
  for i:=1 to n do
    a[i]:= StrToInt(StringGrid1.Cells[i-1,0]);
  //окончание ввода исходных данных
  for k:=1 to n-1 do
  //k-номер элемента, с которого начнется поиск min
    begin
      c:=a[k]; m:=k;//элемент a[k] принимается за min
      for i:=k+1 to n do //цикл поиска min
        if a[i]<c
          then
            begin
              c:=a[i]; m:=i
            end;//конец цикла поиска min
      a[m]:=a[k]; a[k]:=c;//перестановка a[k] и a[m]
    end;//конец цикла по k
   // вывод отсортированного массива
   for i:=1 to n do
     StringGrid2.Cells[i-1,0]:=IntToStr(a[i]);
end;
```

Рисунок 18 – **Текст процедуры сортировки методом извлечения**

Отладку процедуры следует провести для следующих тестов:

- в массиве все элементы одинаковые;
- исходный массив упорядочен по неубыванию;
- исходный массив упорядочен по невозрастанию;
- элементы массива неупорядочены, причем в массиве несколько минимальных элементов.

5.4. Создание проекта на основе сортировки методом обменов

Упорядочим заданный целочисленный массив по *неубыванию* на основе алгоритма «*пузырек*», относящегося к алгоритмам *обменов*.

Интерфейс искомого проекта ничем не отличается от интерфейса предыдущего проекта, приведенного на рисунке 16.

Для создания процедуры обработки щелчка по кнопке *Сортировать* сначала запишем необходимые действия в словесной форме следующим образом:

1. Сравним элементы a_1 и a_2. Если не выполняется условие $a_1 \le a_2$, то меняем местами эти элементы и сравниваем элементы a_2 и a_3. Таким образом сравниваем и при необходимости меняем местами все пары исходного массива. Последней рассматривается пара a_{n-1} и a_n. Таким образом, заканчивается первый просмотр массива, при котором максимальный элемент окажется на последнем месте. Это действие напоминает процесс вскипания воды, когда первым всплывает самый большой «пузырек». Именно поэтому рассматриваемый способ сортировки называется методом «пузырька».

2. Второй просмотр массива также начинается со сравнения элементов a_1 и a_2. Последней рассматривается пара a_{n-2} и a_{n-1}. В результате на месте элемента a_{n-1} окажется второй по величине элемент массива («всплыл второй по величине пузырек»).

3. Третий просмотр массива начинается с проверки пары a_1 и a_2, заканчивается проверкой пары a_{n-3} и a_{n-2}, на месте элемента a_{n-2} окажется третий по величине элемент массива.

4. При последнем просмотре будут сравниваться только элементы a_1 и a_2.

Далее обобщим представленные шаги алгоритма следующим образом: просмотр массива заключается в проверке условия $a_i \le a_{i+1}$ и перестановке этих элементов при невыполнении условия неубывания. При этом значение переменной i изменяется от 1 до некоторого k, т. е. последнее проверяемое условие будет $a_k \le a_{k+1}$. Первый просмотр происходит при $k = n-1$, следующий при $k = n-2$, затем при $k = n-3$ и так до $k=1$.

Сформулированному алгоритму соответствует графическая схема алгоритма, приведенная на рисунке 19.

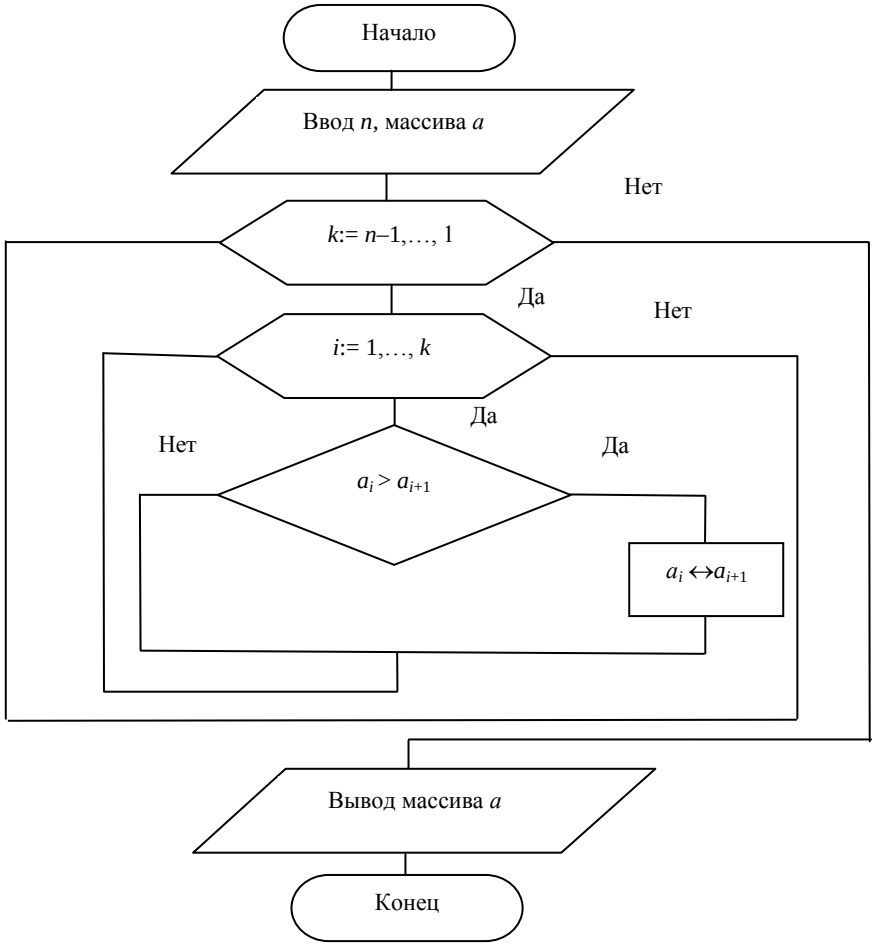


Рисунок 19 – **Схема алгоритма сортировки массива по неубыванию методом «пузырька»**

На основании приведенной схемы алгоритма запишем текст процедуры обработки щелчка по кнопке *Сортировать* (рисунок 20) с использованием оператора цикла *for-downto-do*. Причем отладку процедуры следует провести для тех же тестов, что и в предыдущем проекте.

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    a:array [1..20] of integer;
    i,k,n,m,b:integer;
begin
  n:=StrToInt(Edit1.Text);
  for i:=1 to n do
    a[i]:= StrToInt(StringGrid1.Cells[i-1,0]);
  //конец ввода исходных данных
  for k:=n-1 downto 1 do
  {цикл задания номера последней пары
  при просмотре массива}
    for i:=1 to k do
    {цикл проверки упорядоченности
    элементов a[i],...,a[k+1]}
        if a[i]>a[i+1]
          then
            begin
              //перестановка a[i] и a[i+1]
              b:=a[i]; a[i]:=a[i+1]; a[i+1]:=b;
            end;
   //вывод отсортированного массива
   for i:=1 to n do
     StringGrid2.Cells[i-1,0]:=IntToStr(a[i]);
end;
```

Рисунок 20 – **Текст процедуры сортировки методом «пузырька»**

5.5. Минимизация числа просмотров при сортировке методом «пузырька»

При сортировке методом «пузырька» часто встречается ситуация, когда массив уже отсортирован, а просмотры массива продолжаются. Чтобы вовремя прекратить процесс сортировки, следует фиксировать факт перестановки в какой-нибудь переменной.

Для этой цели лучше всего подходит переменная логического типа. Она принимает одно из двух значений – *True* или *False*. Переменной *W* присваивается значение *True* всякий раз, когда после проверки очередной пары происходила перестановка значений сравниваемых

элементов. Очередной оборот цикла для организации нового просмотра (цикл по переменной *k*) будем выполнять только в том случае, когда при предыдущем просмотре была сделана хотя бы одна перестановка. Перед началом цикла проверки упорядоченности (цикл по переменной *i*) надо переменной *W* присвоить значение *False* (признак того, что пока перестановок не было).

Схема алгоритма приведена на рисунке 21.

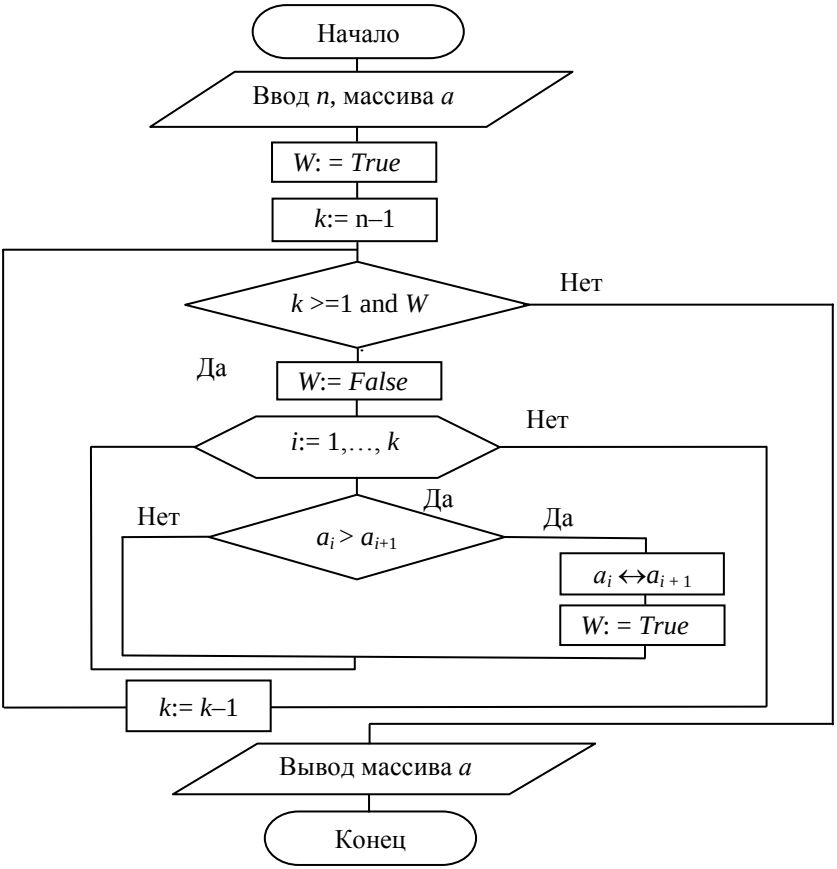


Рисунок 21 – **Схема алгоритма сортировки массива по неубыванию методом «пузырька» с минимизацией числа просмотров**

Обратите внимание на блок *W:= True* перед циклом по переменной *k*. Такой блок обеспечивает выполнение цикла первый раз.

Текст процедуры, соответствующей схеме алгоритма, представленной на рисунке 21, приведен на рисунке 22.

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    a:array [1..20] of integer;
    i,k,n,b:integer;
    W:Boolean;
begin
  n:=StrToInt(Edit1.Text);
  for i:=1 to n do
    a[i]:= StrToInt(StringGrid1.Cells[i-1,0]);
  //конец ввода исходных данных
  W:=True;//признак необходимости просмотра массива
  k:=n-1;//номер последней пары при просмотре
  while (k>=1)and W do
  {цикл прекратит выполнение либо после n-1 просмотра,
  либо при отсутствии перестановок на предыдущем просмотре}
    begin
      W:=False; //признак отсутствия перестановок
      for i:=1 to k do
      //цикл проверки упорядоченности элементов
        if a[i]>a[i+1]
          then
            begin
              b:=a[i]; a[i]:=a[i+1]; a[i+1]:=b;
              W:=True;//установлен признак перестановки
            end;//конец цикла проверки упорядоченности
      k:=k-1;
    end;
   //вывод отсортированного массива
   for i:=1 to n do
     StringGrid2.Cells[i-1,0]:=IntToStr(a[i]);
end;
```

Рисунок 22 – **Текст процедуры сортировки методом «пузырька» с минимизацией числа просмотров**

5.6. Сортировка методом обменов за один просмотр «с возвращением»

Рассмотрим алгоритм сортировки, который, как и «метод рузырька», основан на последовательной проверке упорядоченности по неубыванию пар элементов, начиная с $a_1 \le a_2$ до $a_{n-1} \le a_n$. Однако, в отличие от метода «пузырька», после перестановки элементов, например a_k и a_{k+1}, мы не будем сразу продолжать просмотр слева направо,

а будем устанавливать правильное местоположение элемента a_{k+1}, проверяя пары элементов справа налево. Таким образом, идея рассматриваемого алгоритма заключается в том, что после нахождения пары, не удовлетворяющей условию неубывания, т. е. пары $a_k > a_{k+1}$, в упорядоченной части массива a_1, …, a_k отыскиваем такое место для элемента a_{k+1}, чтобы отсортированной оказалась часть массива a_1, …, a_{k+1}. После этого можно продолжать просмотр массива слева направо, т. е. можно переходить к проверке условия $a_{k+1} > a_{k+2}$.

Схема описанного алгоритма приведена на рисунке 23.

Текст соответствующей процедуры приведен на рисунке 24.

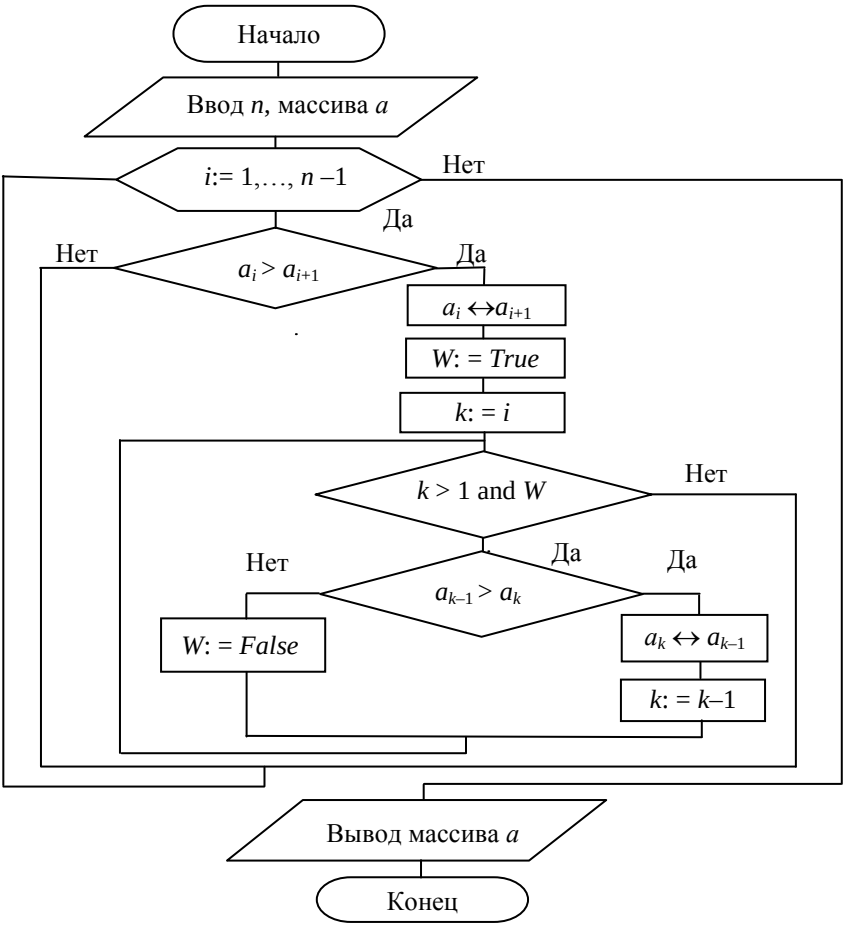


Рисунок 23 – **Схема алгоритма сортировки массива по неубыванию методом обменов за один просмотр «с возвращением»**

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    a:array [1..20] of integer;
    i,k,n,b:integer;
    W:Boolean;
begin
  n:=StrToInt(Edit1.Text);
  for i:=1 to n do
    a[i]:= StrToInt(StringGrid1.Cells[i-1,0]);
  //конец ввода исходных данных
  for i:=1 to n-1 do
  {цикл проверки упорядоченности массива слева направо}
    if a[i]>a[i+1]
      then
        begin
          b:=a[i]; a[i]:=a[i+1]; a[i+1]:=b;
          W:=True;
          k:=i;
          while (k>1)and W do
          {цикл просмотра справа налево для определения
           местоположения элемента a[i]}
            if a[k-1]>a[k]
              then
                begin
                  b:=a[k]; a[k]:=a[k-1]; a[k-1]:=b;
                  k:=k-1
                end
              else
                W:=False;//найдено место для a[i]
        end;
  //вывод отсортированного массива в таблицу строк
  for i:=1 to n do
    StringGrid2.Cells[i-1,0]:=IntToStr(a[i]);
end;
```

Рисунок 24 – **Текст процедуры сортировки массива по неубыванию методом обменов за один просмотр «с возвращением»**

5.7. Сортировка включением

Рассмотрим теперь сортировку по неубыванию целочисленного массива методом включения. В качестве интерфейса проекта можно использовать окно формы, приведенное на рисунке 16.

Алгоритм сортировки включением заключается в том, что в упорядоченный отрезок массива $a_1 \le a_2 \le ... \le a_i$ надо включить элемент a_{i+1} так, чтобы последовательность из $I+1$ символа также была упорядоченной по неубыванию.

Отметим, что указанный процесс надо повторять при изменении значений переменной i от 1 до n–1.

Включение элемента a_{i+1} в отрезок $a_1 \le a_2 \le ... \le a_i$ будем проводить следующим образом: проверяем условие $a_i > a_{i+1}$ и, в случае его выполнения, переставляем сравниваемые элементы. Затем проверяем условие $a_{i-1} > a_i$ и так до левого края массива или до нахождения пары, упорядоченной по неубыванию.

Схема алгоритма приведена на рисунке 25.

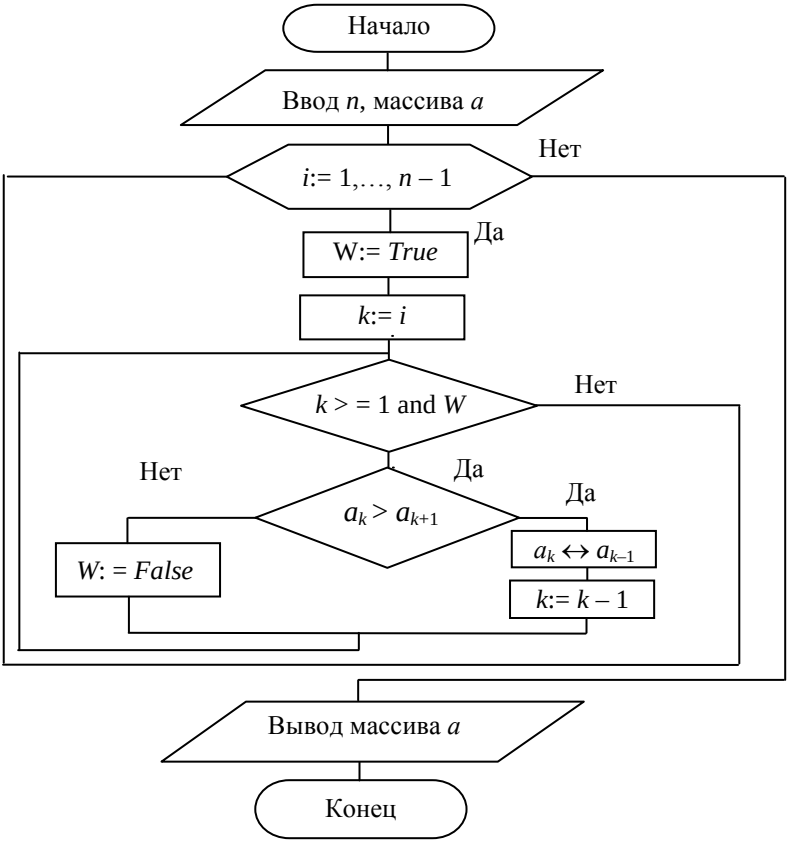


Рисунок 25 – **Схема алгоритма сортировки массива по неубыванию методом включения**

Текст процедуры обработки щелчка по кнопке *Сортировать* приведен на рисунке 26.

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    a:array [1..20] of integer;
    i,k,n,b:integer;
    W:Boolean;
begin
  n:=StrToInt(Edit1.Text);
  for i:=1 to n do
    a[i]:= StrToInt(StringGrid1.Cells[i-1,0]);
  //конец ввода исходных данных
  for i:=1 to n-1 do
  {цикл включения элемента a[i+1]}
    begin
      W:=True;
      k:=i;
      while (k>=1)and W do
      {цикл просмотра справа налево для определения
       места включения элемента a[i+1]}
        if a[k]>a[k+1]
          then
            begin
              b:=a[k]; a[k]:=a[k+1]; a[k+1]:=b;
              k:=k-1
            end
          else
            W:=False;//найдено место для a[i+1]
    end;//конец циклов
  //вывод отсортированного массива в таблицу строк
  for i:=1 to n do
    StringGrid2.Cells[i-1,0]:=IntToStr(a[i]);
end;
```

Рисунок 26 – **Текст процедуры сортировки массива по неубыванию методом включения**

Отладку процедуры следует провести для тех же тестов, что и в проекте сортировки методом извлечения.

5.8. Сортировка слиянием

Алгоритм сортировки слиянием рассмотрим на примере решения следующей задачи.

Пусть дан массив $a_1 \le a_2 \le \ldots \le a_n$, упорядоченный по неубыванию, и массив $b_1 \ge b_2 \ge \ldots \ge b_m$, упорядоченный по невозрастанию.

Требуется объединить эти два массива в один массив $c_1 \le c_2 \le \ldots \le c_{n+m}$, упорядоченный по неубыванию.

Интерфейс проекта приведен на рисунке 27.

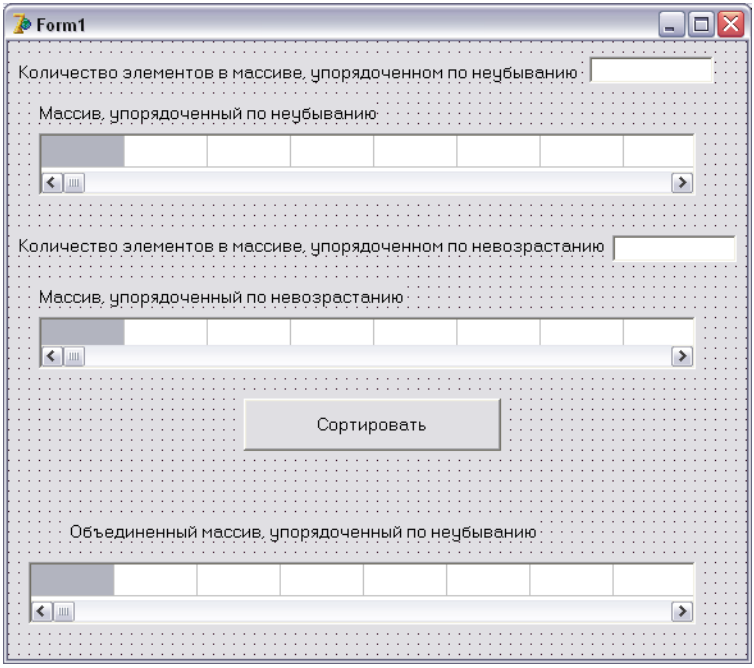


Рисунок 27 – **Интерфейс проекта сортировки слиянием**

Алгоритм сортировки слиянием основан на цикле поэлементного формирования массива $c_1, c_2, \ldots, c_{n+m}$. Очень важную роль в этом процессе играют переменные k и p. Переменная k указывает на номер элемента массива a, который еще не переписан в массив c. Так как элементы из массива a переписываются в массив c слева направо, то в начальный момент k=1. Аналогично, переменная p указывает на номер элемента массива b, который еще не переписан в массив c. Так как элементы из массива b переписываются в массив c справа налево, то в начальный момент p=m. Процесс формирования искомого массива иллюстрирует рисунок 28.

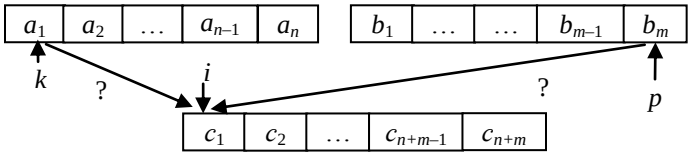


Рисунок 28 – **Схема слияния упорядоченных массивов**

Итак, на место элемента c_i надо поместить меньший из элементов a_k и b_p. Если меньшим оказался элемент a_k, то после его размещения в массиве c надо счетчик k увеличить на 1. Если меньшим оказался элемент b_p, то после его размещения в массиве c надо счетчик p уменьшить на 1.

Прежде чем сравнивать элементы a_k и b_p надо проверить их существование. Выполнение условия $k > n$ означает, что все элементы из массива a уже переписаны в массив c_N , а, следовательно, надо переписать в c_i элемент b_p. Аналогично, выполнение условия $p < 1$ означает, что все элементы из массива b уже переписаны в массив c, а, следовательно, надо переписать в c_i элемент a_k.

На рисунке 29 приведен фрагмент процедуры, реализующей алгоритм сортировки слиянием (без операторов ввода исходных и вывода полученного массивов).

```
//k - номер анализируемого элемента из массива a
//p - номер анализируемого элемента из массива b
k:=1; p:=m;
for i:=1 to n+m do //цикл заполнения искомого массива
  if k>n
    then // уже переписаны все элементы из массива a
      begin
        //перепись очередного элемента из массива b
        c[i]:=b[p]; p:=p-1
      end
    else
      if p<1
        then // уже переписаны все элементы из массива b
          begin
            //перепись очередного элемента из массива a
            c[i]:=a[k]; k:=k+1
          end
        else
          if a[k]<b[p]
            then
              begin
                //перепись очередного элемента из массива a
                c[i]:=a[k]; k:=k+1
              end
            else
              begin
                //перепись очередного элемента из массива b
                c[i]:=b[p]; p:=p-1
              end;//конец цикла
```

Рисунок 29 – **Фрагмент процедуры сортировки массива слиянием**

Отладку проекта следует провести для следующих тестов:

- все элементы первого массива больше элементов второго массива;
- все элементы второго массива больше элементов первого массива;
- все элементы первого массива меньше максимального элемента и больше минимального элемента второго массива;
- все элементы второго массива меньше максимального элемента и больше минимального элемента первого массива;
- элементы заданных массивов связаны соотношением $b_m < a_1 < b_1 < a_n$, при этом по крайней мере одно и тоже число содержится в обоих массивах;
- элементы заданных массивов связаны соотношением $a_1 < b_m < a_n < b_1$, при этом по крайней мере одно и тоже число содержится в обоих массивах.

5.9. Сортировка распределением

Алгоритм сортировки распределением рассмотрим на примере решения следующей задачи.

Задан список студентов группы с экзаменационными оценками, задается также балльность системы оценок. Требуется упорядочить список в порядке невозрастания экзаменационной оценки.

Интерфейс задачи представлен на рисунке 30.

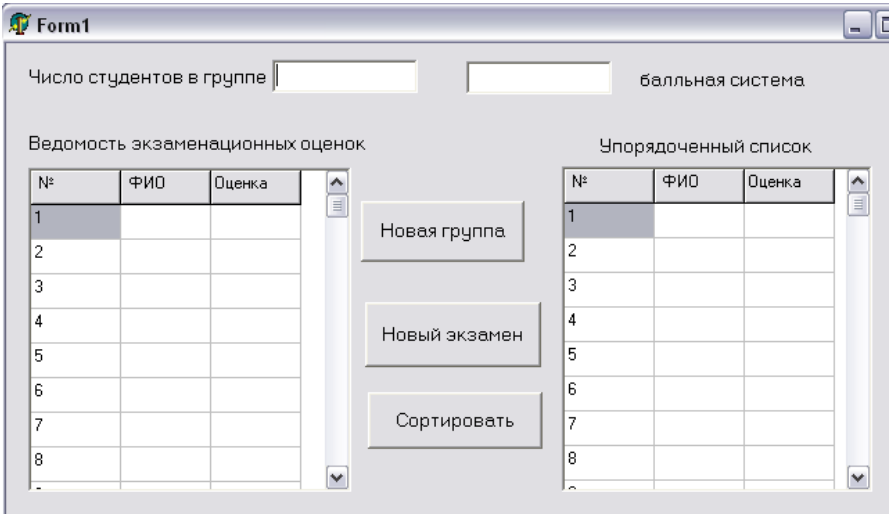


Рисунок 30 – **Интерфейс задачи сортировки распределением**

Разработаем алгоритм данной задачи. Пусть *n* – число студентов в группе, а *m* – балльность системы оценок. После ввода фамилий и оценок в первую таблицу оценки из третьего столбца переписывается в массив *a* и организуется цикл просмотра этого массива с целью поиска оценок, равных *m*. Как только такая оценка найдена, фамилия и оценка переписываются в очередную строку второй таблицы. По завершении просмотра первой таблицы для поиска всех строк с оценкой *m* начинается поиск и перепись во вторую таблицу всех строк с оценкой *m*–1. Далее процесс повторяется для поиска оценок *m*–2 и так далее до поиска оценок, равных 1.

Текст процедуры обработки щелчка по кнопке *Сортировать* приведен на рисунке 31.

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    a:array[1..25] of integer;
    i,j,n,m,k:integer;
begin
  n:=StrToInt(Edit1.Text); //количество оценок
  m:=StrToInt(Edit2.Text); //балльность
  for i:=1 to n do //цикл ввода оценок
    a[i]:=StrToInt(StringGrid1.Cells[2,i]);
  k:=0;//номер строки в формируемом списке
  for j:=m downto 1 do //цикл перебора всех оценок
    for i:=1 to n do
      if a[i]=j
        then //найдена рассматриваемая оценка j
          begin // запись ФИО и оценки в StringGrid2
            k:=k+1;
            StringGrid2.Cells[1,k]:=StringGrid1.Cells[1,i];
            StringGrid2.Cells[2,k]:=StringGrid1.Cells[2,i];
          end;
end;
```

Рисунок 31 – **Процедура сортировки распределением**

Отладку процедуры следует провести для тех же тестов, что и в проекте сортировки методом извлечения.

6. ОБРАБОТКА УПОРЯДОЧЕННЫХ МАССИВОВ

Особенность обработки упорядоченных массивов состоит в том, что при последовательном анализе элементов массива часто нет необходимости просматривать весь массив. Так, например, при нахождении суммы отрицательных значений в упорядоченном по неубыванию массиве суммируются элементы до тех пор, пока не встретится неотрицательное значение.

Кроме того, в таких задачах надо проверять упорядоченность заданного массива, т. е. корректность исходных данных. Ниже рассматриваются два проекта по обработке упорядоченных массивов.

Рассмотрим два примера обработки упорядоченных массивов.

Пример 1. Требуется разработать проект, определяющий количество отрицательных чисел в упорядоченном по невозрастанию массиве. Проект должен включать в себя проверку упорядоченности заданного массива и выдавать сообщение об ошибке, если массив не удовлетворяет условию упорядоченности по невозрастанию.

Отладку проекта следует провести для следующих тестов:

- Массив упорядочен по неубыванию и в нем есть неравные между собой элементы, например, массив (–3, –3, 0, 0, 2). Ответом в данном случае должно быть сообщение, что исходный массив не упорядочен по невозрастанию.
- Первая пара элементов массива упорядочена по невозрастанию, а остальные элементы составляют неубывающую последовательность, например, массив (–3, –4, 0, 0, 2). Ответом должно быть сообщение, что исходный массив не упорядочен по невозрастанию.
- Последняя пара элементов массива упорядочена по невозрастанию, а остальные элементы составляют неубывающую последовательность, например, массив (0, 0, 2, –3, –4). Ответом должно быть сообщение, что исходный массив не упорядочен по невозрастанию.
- Массив упорядочен по невозрастанию и все его элементы – отрицательные числа, например, для массива (–3,–3, –7, –7) – ответ – 4.
- Массив упорядочен по невозрастанию и в нем нет отрицательных элементов, например, для массива (7, 7, 3, 3) – ответ – 0.
- Все элементы массива положительны и равны между собой, например, для массива (4, 4, 4) – ответ будет 0.
- В упорядоченном по невозрастанию массиве есть положительные, отрицательные и нулевые элементы, например, для массива (7, 5, 5, 0, 0, –3, –5, –5) ответ – 3.

Интерфейс проекта представлен на рисунке 32.

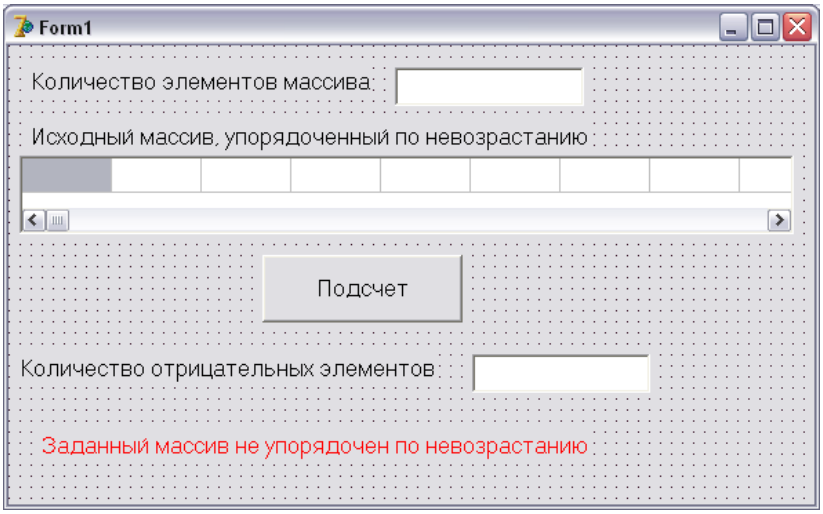


Рисунок 32 – **Интерфейс проекта подсчета отрицательных элементов в упорядоченном массиве**

Алгоритм проверки упорядоченности массива по невозрастанию состоит в том, что последовательно проверяется условие $a_i \geq a_{i+1}$ при $i = 1, 2, \ldots$. Проверка прекращается в одном из двух случаев. Во-первых, если найдется пара, неудовлетворяющая условию невозрастания. В этом случае надо выдать сообщение о неупорядоченности исходного массива. Во-вторых, если последняя пара элементов упорядочена по невозрастанию $a_{n-1} \geq a_n$, а это означает, что исходный массив упорядочен и начинается подсчет отрицательных элементов.

Идея алгоритма подсчета отрицательных элементов такова: следует проверять на отрицательность элементы массива слева направо, процесс просмотра заканчивается или при обнаружении первого отрицательного элемента, или при достижении конца массива. Если на i-ом месте обнаружится отрицательный элемент, то, следовательно, все остальные элементы массива являются отрицательными и их число равно $n - i + 1$. Если же мы дойдем до конца массива, то отрицательных элементов в массиве нет.

Текст процедуры проверки упорядоченности и подсчета отрицательных элементов приведен на рисунке 33, а тексты процедур чистки полей при активации формы и изменении значений исходных данных приведены на рисунке 34.

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    a:array[1..20] of integer;
    i,n,k:integer;
    W:Boolean;
begin
   n:=StrToInt(Edit1.Text);//"длина" массива
   for i:=1 to n do //цикл ввода массива
     a[i]:= StrToInt(StringGrid1.Cells[i-1,0]);
   //проверка упорядоченности массива по невозрастанию
   i:=1;
   W:=true; {признак того, что все проверенные пары
             упорядочены по невозрастанию}
   while (i<n) and W do
     if a[i]>=a[i+1]
       then i:=i+1 //переход к проверке следующей пары
       else W:=false; // нашлась возрастающая пара
   {конец цикла проверки упорядоченности
   массива по невозрастанию}
   if W
     then //массив упорядочен по невозрастанию
       begin
         i:=1;
         W:=true;//пока не найден отрицательный элемент
         while (i<=n) and W do
           if a[i]<0
             then W:=false //найден отрицательный элемент
             else i:=i+1;
         //конец цикла подсчета отрицательных элементов
         if W //определение числа отрицательных
           then k:=0
           else k:=n-i+1;
         Edit2.Text:=IntToStr(k);
       end
     else Label4.Show;{вывод сообщения о неупорядоченности
                       исходного массива по невозрастанию}
end;
```

Рисунок 33 – **Процедура подсчета отрицательных элементов в упорядоченном по невозрастанию массиве с проверкой упорядоченности**

```
//процедура активации формы
procedure TForm1.FormActivate(Sender: TObject);
var
  i:integer;
begin
  Edit1.Text:='';
  Edit1.SetFocus;
  Edit2.Text:='';
  Label4.Hide;
  for i:=1 to StringGrid1.ColCount do
     StringGrid1.Cells[i-1,0]:='';
end;
{процедура обработки щелчка по полю Edit1
 или по ячейке таблицы StringGrid1}
procedure TForm1.Edit1Click(Sender: TObject);
begin
  Edit2.Text:='';
  Label4.Hide;
end;
//процедура изменения значения поля Edit1
procedure TForm1.Edit1Change(Sender: TObject);
  var
  i:integer;
begin
  Edit2.Text:='';
  Label4.Hide;
  for i:=1 to StringGrid1.ColCount do
     StringGrid1.Cells[i-1,0]:='';
end;
```

Рисунок 34 – **Тексты вспомогательных процедур проекта**

Пример 2. Требуется разработать проект, вычисляющий сумму положительных элементов в упорядоченном по неубыванию целочисленном массиве и определяющий, есть ли в массиве отрицательные элементы. Проект должен включать в себя проверку упорядоченности заданного массива и выдавать сообщение об ошибке, если массив не удовлетворяет условию упорядоченности по неубыванию.

Отладку проекта следует провести для следующих тестов:

• Массив упорядочен по невозрастанию и в нем есть неравные между собой элементы, например, массив (2, 0, 0, –3, –3). Ответом должно быть сообщение, что исходный массив не упорядочен по неубыванию.

• Первая пара элементов массива упорядочена по неубыванию, а остальные элементы составляют невозрастающую последовательность, например, массив (0, 2, –1, –1, –4). Ответом должно быть сообщение, что исходный массив не упорядочен по неубыванию.

• Последняя пара элементов массива упорядочена по неубыванию, а остальные элементы составляют невозрастающую последовательность, например, массив (6, 4, 4, 3, 4). Ответом должно быть сообщение, что исходный массив не упорядочен по неубыванию.

• Массив упорядочен по неубыванию и все его элементы отрицательные числа, например, массива (–3, –3, –2, –2). В ответе должно значится, что сумма равна 0, отрицательный элемент в массиве имеется.

• Массив упорядочен по неубыванию и в нем нет отрицательных элементов, например, для массива (3, 3, 7, 7) сумма положительных элементов равна 20.

• Все элементы массива равны 0, например, для массива (0, 0, 0) в ответе должно быть указано, что сумма равна 0 и в массиве нет отрицательных элементов.

• В упорядоченном по неубыванию массиве есть положительные, отрицательные и нулевые элементы, например, для массива (–5, –5, –3, 0, 0, 5, 5, 7) сумма получится равной 17.

Интерфейс проекта представлен на рисунке 35.

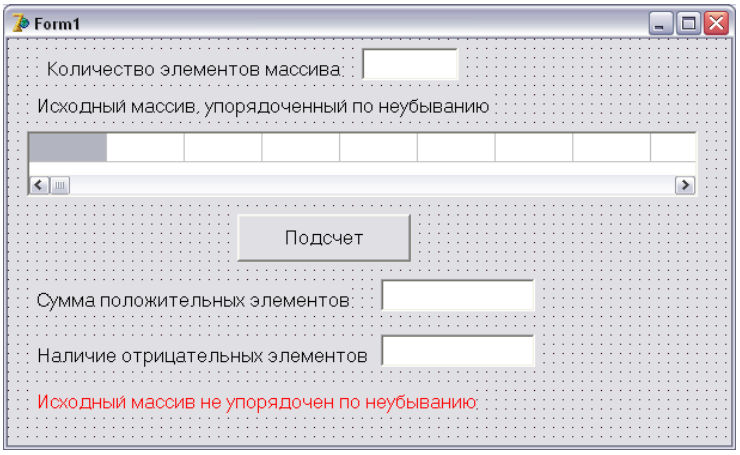


Рисунок 35 – **Интерфейс проекта подсчета суммы положительных элементов в упорядоченном массиве**

Идея данного алгоритма заключается в следующем: просматривается массив с конца до встречи с отрицательным элементом, при просмотре суммируются положительные элементы. Текст процедуры проверки упорядоченности и подсчета отрицательных элементов приведен на рисунках 36 и 37. При этом, на рисунке 36 представлена первая часть процедуры, содержащая раздел описаний, операторы ввода исходных данных и проверки упорядоченности исходного массива.

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    a:array[1..20] of integer;
    i,n,S:integer;
    W:Boolean;
    k:string;
begin
   n:=StrToInt(Edit1.Text);//"длина" массива
   for i:=1 to n do //цикл ввода массива
     a[i]:= StrToInt(StringGrid1.Cells[i-1,0]);
   //проверка упорядоченности массива по неубыванию
   i:=1;
   W:=true; {признак того, что все проверенные пары
            упорядочены по неубыванию}
   while (i<n) and W do
     if a[i]<=a[i+1]
       then i:=i+1 //переход к проверке следующей пары
       else W:=false; // нашлась убывающая пара
   {конец цикла проверки упорядоченности
   массива по неубыванию}
```

Рисунок 36 – **Начало основной процедуры проекта (проверка упорядоченности массива по неубыванию)**

Заключительная часть текста процедуры обработки щелчка по кнопке *Подсчет*, суммирующая положительные элементы и определяющая наличие в массиве отрицательного элемента, приведена на рисунке 37.

```
    if W
      then
        begin
          i:=n; //просмотр с последнего элемента
          W:=true;//признак отсутствия отрицательного
          S:=0; // для накопления суммы положительных
          while (i>=1) and W do
            if a[i]<0
              then W:=false //найден отрицательный элемент
              else
                  begin
                    S:=S+a[i];//накопление суммы положительных
                    i:=i-1; //переход к следующему элементу
                  end;
          //конец цикла вычисления суммы положительных
          if W //отрицательного элемента не было?
            then k:='нет'
            else k:='есть';
          Edit2.Text:=IntToStr(S);
          Edit3.Text:=k;
        end
      else Label5.Show;//массив не упорядочен по неубыванию
end;
```

Рисунок 37 – **Заключительная часть основной процедуры проекта (суммирование положительных элементов и определение наличия отрицательного элемента)**

Процедуры чистки полей при активации формы и изменении значений исходных данных данного проекта аналогичны соответствующим процедурам предыдущего проекта, приведенным на рисунке 34.

7. ОБРАБОТКА СИМВОЛЬНОЙ ИНФОРМАЦИИ

7.1. Представление символов в памяти компьютера

Каждому символу клавиатуры поставлен в соответствие код, т. е. целое число в шестнадцатеричной системе исчисления, которым символ представлен в памяти компьютера. Все коды сводятся в кодовую таблицу.

Примерами кодов некоторых символов в соответствии с широко используемой кодовой таблицей могут быть следующие:

• пробел→20_{16}; 0→30_{16}; 9→39_{16};
• латинские буквы: A→41_{16};…Z→$5A_{16}$; a→61_{16};…z→$7A_{16}$;
• русские буквы: A→80_{16};…Я→$9F_{16}$;… a→$A0_{16}$.

Для составления программ не надо знать конкретные коды символов. Однако полезно знать соотношения между кодами символов при сортировке символьной информации, в которой используются цифры, латинские и русские буквы. Так, наименьший код имеет пробел, затем цифры, прописные и строчные латинские буквы, а самые большие коды имеют прописные и строчные буквы кириллицы.

Для хранения кода символа необходим один байт памяти.

7.2. Описание переменных для символьной информации

Для представления символьной информации в программе используются переменные двух типов – литерные и строковые.

Переменные литерного типа можно описать следующим образом:

```
var
    s, s1: char;
    w: char;
```

Каждая из переменных литерного типа может принимать в качестве значения один символ, т. е. одно значение из кодовой таблицы. При использовании в выражениях строка и литера заключаются в апострофы (*s:='a'*).

Переменные строкового типа могут принимать в качестве значения последовательность символов, т. е. строки. Количество символов в строке (длина строки) может быть от 0 до 255.

Описать строковые переменные можно следующим образом:

```
var
    a, b: string[20];
    c, cod: string[10];
    d: string;
```

То же самое описание может быть записано следующим образом:

```
Type
    st1=string[20];
    st2=string[10];
var
    a, b: st1;
    c, cod: st2;
    d: string;
```

Если в программе есть оператор *a:='x+y=z'*, то значение переменной *a* займет в памяти 6 байт. В нулевом байте отрезка памяти, отведенном для хранения строки, находится длина строки. Таблица 1 иллюстрирует распределение памяти для переменной *a*.

Таблица 1 – **Побайтовое распределение памяти для символьной переменной *a: = 'x + y = z'***

Номера байта	0	1	2	3	4	5
Содержимое (символы)	5	x	+	y	=	z

7.3. Операции над строковыми выражениями

Выражения, операндами которых являются символьные значения (литеры, строки, переменные и функции литерного и строкового типа) называются строковыми выражениями.

Над строковыми выражениями определены операции сцепления (конкатенации), отношения и доступа к отдельным символам.

Операция сцепления (+) применяется для соединения нескольких строк в одну. Пример, приведенный ниже, показывает формирование значения переменной *b* как сцепление нескольких строковых выражений.

```
var
  a,b: string[10];
  d: char;
begin
  a:='кафедра'; d:='И'; b:=a + ' ' + d + 'ВС';
….
end
```

С помощью *операций отношения* (=, <, >, <=, >=, <>) проводят сравнение двух строковых выражений.

Сравнение строк выполняется слева направо до первого несовпадающего символа. При этом та строка считается большей, в которой первый несовпадающий символ имеет больший код в кодовой таблице. Результат операции сравнения имеет булевский тип и принимает значение *True* или *False*.

Следует отметить, что строки равны, если они имеют одинаковую длину и содержат на одних и тех же местах одинаковые символы. Если первые *n* символов сравниваемых строк совпадают и при этом в первой строке только *n* символов, то первая строка (более короткая)

будет меньше, чем вторая (более длинная). Таблица 2 иллюстрирует использование операций отношения для строковых выражений.

Таблица 2 – **Применение операций отношения для строковых выражений**

Выражение	Результат
'cos(x)'<'cos(y)'	*True*
'cos(x)'<'cos(X)'	*False*
'a+b'='b+a'	*False*
'a+b'='a+b'	*True*
'a+'>='a+b'	*False*

Доступ к отдельным символам строки осуществляется по номеру символа, записываемого в квадратных скобках после идентификатора строковой переменной. Пусть строковые переменные *a* и *b* имеют следующие значения:

a:='кафедра'; b:= 'кафедра ИВС';

Тогда переменная *a*[2] имеет значение *'a'*, переменная *a*[1+5] – *'p'*, *b*[8] имеет значение пробел.

В нулевом байте содержится значение текущей длины строки. Таким образом, выражение *a*[0] позволяет получить значение текущей длины строки *a*. Однако *a*[0] не является числом. Для его преобразования в числовой формат надо воспользоваться функцией *Ord*. Для рассмотренного примера функция *Ord*(*a*[0]) имеет значение 7, а *Ord*(*b*[0]) – 11.

7.4. Оператор присваивания

С помощью оператора присваивания строковой или литерной переменной можно присвоить в качестве значения строковое выражение. Если текущая длина строки правой части оператора больше максимально допустимой (в соответствии с описанием) длины левой части, то лишние символы справа отбрасываются. Литерной переменной можно присвоить значение строкового типа с текущей длиной 1.

Пусть имеется следующая совокупность описаний:

```
var
    s10:string[10];
    s3:string[3];
    c:char;
```

Таблица 3 иллюстрирует результаты работы различных операторов присваивания для приведенных переменных.

Таблица 3 – **Применение операторов присваивания**

Операторы	Результат последнего присваивания	Текущая длина результата
s10:='кафедра'+' '+'ИВС';	*'кафедра ИВ'*	10
s3:='И-'; c:='4'; s10:=s3+c;	*'И–4'*	3
s3:='И'; c:=s3;	*'И'*	–
s3:='И-4'; c:=s3[3];	*'4'*	–
s3:='И-4'; s10:=s3[1]+s3[3];	*'И4'*	2

Существуют две особенности использования оператора присваивания для изменения отдельных символов строки:

- если строковая переменная не имела значения, а затем отдельным ее символам присвоены какие-то литеры, то данная строковая переменная остается неопределенной;
- если значение строковой переменной имело текущую длину *n* и с помощью оператора присваивания задается значение ее *(n*+1*)*-ого символа, то значение строковой переменной не изменяется, а текущая длина остается равной *n*.

Таблица 4 иллюстрирует особенности использования оператора присваивания для переменной *s5:string*[*5*].

Таблица 4 – **Особенности применения операторов присваивания**

Исходное состояние переменной *s5*		Оператор	Конечное состояние переменной *s5*	
значение	длина		значение	длина
–	–	*s5[1]:='a';* *s5[2]:='b';*	–	–
'' (пустая строка)	0	*s5[1]:='a';*	'' (пустая строка)	0
'a+b'	3	*s5[3]:='c';*	*'a+c'*	3
'a+b'	3	*s5[4]:='=';* *s5[5]:='c';*	*'a+b'*	3

В качестве примера создадим проект, который определяет максимальную цифру в заданном натуральном числе.

Интерфейс проекта приведен на рисунке 38. Идея алгоритма заключается в том, что заданное число рассматривается как строка из последовательности цифр и сравнение цифр выполняется так же, как и сравнение элементов массива. Тексты процедур проекта приведены на рисунке 39.

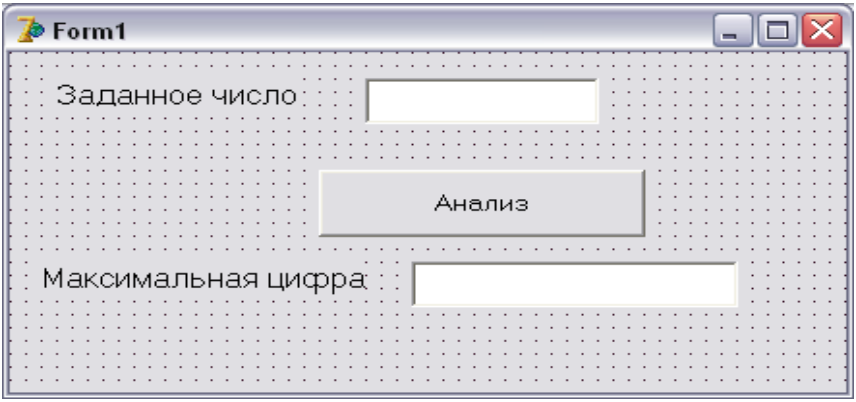


Рисунок 38 – **Интерфейс проекта для определения максимальной цифры числа**

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    a:string[10];
    b:char;
    n,i:integer;
begin
  a:=Edit1.Text; //заданное число
  n:=Ord(a[0]);//количество цифр в заданном числе
  if n>=1
    then
      begin
        b:=a[1];
        for i:=2 to n do
          if a[i]>b
            then b:=a[i];
        Edit2.Text:=b //вывод искомой цифры
      end
    else
      Edit2.Text:='число не задано'
end;
procedure TForm1.Edit1Click(Sender: TObject);
begin
 Edit2.Text:='';
end;
```

Рисунок 39 – **Процедуры проекта для определения максимальной цифры числа**

7.5. Стандартные процедуры

Удаление части строки (подстроки) выполняется с помощью процедуры *Delete(s,m,n)*, которая в строке *s* удаляет *n* символов, начиная с позиции *m*, где *m* и *n* – целые арифметические выражения.

Особенности применения данной процедуры следующие:

- если количество удаляемых символов равно 0, то строка не меняется;
- если конечная позиция удаляемой подстроки выходит за правую границу строки, то удаляется подстрока с указанной позиции до конца строки;
- если начальная позиция удаляемой подстроки или количество удаляемых символов представлены отрицательными числами, то при выполнении оператора *Delete* может быть выдано сообщение об ошибке или строка может дополняться какими-то символами.

В таблице 5 приведены примеры применения процедуры *Delete* для строки *s5:string[5]*.

Таблица 5 – **Применение процедуры удаления части строки**

Исходное состояние переменной *s5*		Оператор	Конечное состояние переменной *s5*	
значение	длина		значение	длина
'12345'	5	*Delete(s5,3,2);*	*'125'*	3
'12345'	5	*Delete(s5,1,5);*	*''* (пустая строка)	0
'123'	3	*Delete(s5,2,4);*	*'1'*	1
'123'	3	*Delete(s5,2,0);*	*'123'*	3
'1234'	4	*Delete(s5,0,5);*	*''* (пустая строка)	0
'123'	3	*Delete(s5,0,3);*	*'3'*	1
'123'	3	*Delete(s5,2,-2);*	сообщение об ошибке	

Вставка подстроки в строку выполняется с помощью процедуры *Insert(s1,s2,m)*, которая осуществляет вставку строки *s1* в строку *s2*, начиная с позиции *m*.

Особенности применения данной процедуры следующие:

- нельзя отмечать место вставки несуществующих символов;
- после вставки текущая длина «принимающей» строки увеличивается, но не может превысить границ, указанных в описании;
- «вставляемая» строка после вставки не изменяется.

В таблице 6 приведены примеры применения процедуры *Insert* для переменных *s5:string[5] и s3:string[3]*.

Таблица 6 – **Применение процедуры вставки подстроки в строку**

Исходное состояние		Оператор	Результат	
*s*5	*s*3		*s*5	*s*3
'123'	*'45'*	*Insert(s3,s5,2);*	*'14523'*	*'45'*
'123'	*'45'*	*Insert(s3,s5,4);*	*'12345'*	*'45'*
'123'	*'45'*	*Insert(s3,s5,0);*	сообщение об ошибке	
'12'	*'4'*	*Insert(s5,s3,1);*	*'12'*	*'124'*
'1234'	*'56'*	*Insert(s3,s5,4);*	*'12356'*	*'56'*
'1234'	*'567'*	*Insert(s3,s5,2);*	*'15672'*	*'567'*

Преобразование числового значения в строковое выполняется с помощью процедуры *Str(c:n,s)* или *Str(c:n:m,s)*, где *n* и *m* – целые числа.

В таблице 7 приведены примеры применения процедуры *Str* для следующих переменных:

```
s5:string[5];
s3:string[3];
x,y:real;
l,k:integer;
x:=12.3; y:= –0.567E4; k:=34; l:= –590;
```

Таблица 7 – **Применение процедуры преобразования числового значения в строковое**

Оператор	Значение результата	Длина
Str(k,s5);	*'34'*	*2*
Str(k,s3);	*'34'*	*2*
Str(l,s5);	*'-590'*	*4*
Str(l,s3);	*'–59'*	*3*
Str(k:4,s5);	*' 34'*	*4*
Str(k:4,s3);	*' 3'*	*3*
Str(x,s5);	*' 1.2'*	*5*
Str(y,s5);	*'–5.6'*	*5*
Str(x:4,s5);	*'1.2E+'*	*5*
Str(y:5,s5);	*'–5.7E'*	*5*
Str(x:4:1,s5);	*'12.3'*	*4*
Str(x:3:1,s5);	*'12.3'*	*4*
Str(x:3:1,s3);	*'12.'*	*3*

Преобразование строкового выражения в числовое выполняется с помощью процедуры *Val(s,c,j)*, где *s* – строковое выражение, *c* – величина целого или вещественного типа, *j* – переменная целого типа, принимающая значение, равное 0, если преобразование прошло без ошибок. Если же преобразование не может быть проведено, то значение *j* – номер ошибочной позиции, а значение *c* в этом случае не определено.

В таблице 8 приведены примеры применения процедуры *Val* для переменных, описание и определение которых приведены перед таблицей 7.

Таблица 8 – **Применение процедуры преобразования строкового выражения в числовое**

Исходное состояние		Оператор	Результат	
значение	длина		число (*x, k*)	*j*
'1.2E+'	5	*Val(s5,x,j);*	–	6
'–5.7E+03'	8	*Val(s20,x,j);*	–5700.0	0
'12'	2	*Val(s3,x,j);*	12.0	0
'+1_A3'	5	*Val(s5,x,j);*	–	4
'12'	2	*Val(s3,k,j);*	12	0
' 12'	3	*Val(s3,k,j);*	–	1
'12.3'	4	*Val(s5,k,j);*	–	3
'+5760'	5	*Val(s5,k,j);*	5760	0
'57E2'	4	*Val(s20,k,j);*	–	3

7.6. Стандартные функции

Выделение подстроки может быть выполнено с помощью функции *Copy(s,m,n)*, которая выделяет из строки *s* подстроку длины *n*, начиная с позиции *m*.

В таблице 9 приведены примеры применения функции *Copy* для переменных *s5:string[5] и s3:string[3]*.

Таблица 9 – **Применение функции выделения подстроки**

Исходное состояние строки *s5*		Оператор	Формируемая строка *s3*	
значение	длина		значение	длина
'12345'	5	*s3:=Copy(s5,3,2);*	*'34'*	2
'12345'	5	*s3:=Copy(s5,1,5);*	*'123'*	3
'12345'	5	*s3:=Copy(s5,4,4);*	*'45'*	2
'12345'	5	*s3:=Copy(s5,6,2);*	'' (пустая строка)	0
'123'	3	*s3:=Copy(s5,0,2);*	сообщение об ошибке	
'123'	3	*s3:=Copy(s5,-1,2);*	сообщение об ошибке	
'123'	3	*s3:=Copy(s5,2,-1);*	'' (пустая строка)	0
'123'	3	*s3:=Copy(s5,2,0);*	'' (пустая строка)	0

Функция *Concat(s1,s2,…,sn)* выполняет *сцепление* перечисленных строк *s1, s2, …, sn.* Ранее указывалось, что сцепление может быть вы-

полнено операторами '+'. Таким образом, следующие операторы эквивалентны

s5:=Concat(s3,'23');

s5:=s3 + '23';

Поиск подстроки в строке осуществляет функция *Pos(s1,s2)*, которая выдает номер позиции первого появления строки *s1* в строке *s2*. Если *s1* не содержится в строке *s2*, то ответ – 0.

В таблице 10 приведены примеры применения функции *Pos* для переменных *s5:string[5] и s3:string[3]*.

Таблица 10 – **Применение функции поиска подстроки в строке**

Значение		Оператор	Результат (*k*)
s3	*s5*		
'2'	*'2322'*	*k:=Pos(s3,s5);*	1
–	*'2322'*	*k:=Pos('4',s5);*	0
'12'	*'31212'*	*k:=Pos(s3,s5);*	2
'123'	*'123'*	*k:=Pos(s3,s5);*	1
'123'	*'12'*	*k:=Pos(s3,s5);*	0

Длина строки s может быть определена с помощью функции *Length(s)*.

В таблице 11 приведены примеры применения функций *Length* и *Ord* для переменной *s5:string[5]*.

Таблица 11 – **Определение длины строки**

Значение *s5*	Оператор	Результат (*k*)
'123'	*k:=Length(s5);*	3
'123'	*k:=Ord(s5[0]);*	3
''	*k:=Length(s5);*	0
''	*k:=Ord(s5[0]);*	0

Преобразование строчной латинской буквы в прописную выполняется функцией *Upcase(c)*, в которой аргумент *c* и результат имеют литерный тип.

Получение кода литеры может быть получено с помощью функции *Ord(c)*, в которой аргумент *c* имеет литерный тип, а результат – целый.

Для *получения литеры по коду* используется функция *Chr(k)*, в которой аргумент имеет целочисленный тип, а результат – литерный.

7.7. Примеры проектов обработки символьной информации

Пример 1. Требуется разработать проект, преобразующий регистр введенной буквы русского алфавита. При вводе символа, не являющегося буквой русского алфавита, проект должен выдавать сообщение об ошибке.

Примеры окон приложений проекта приведены на рисунке 40.

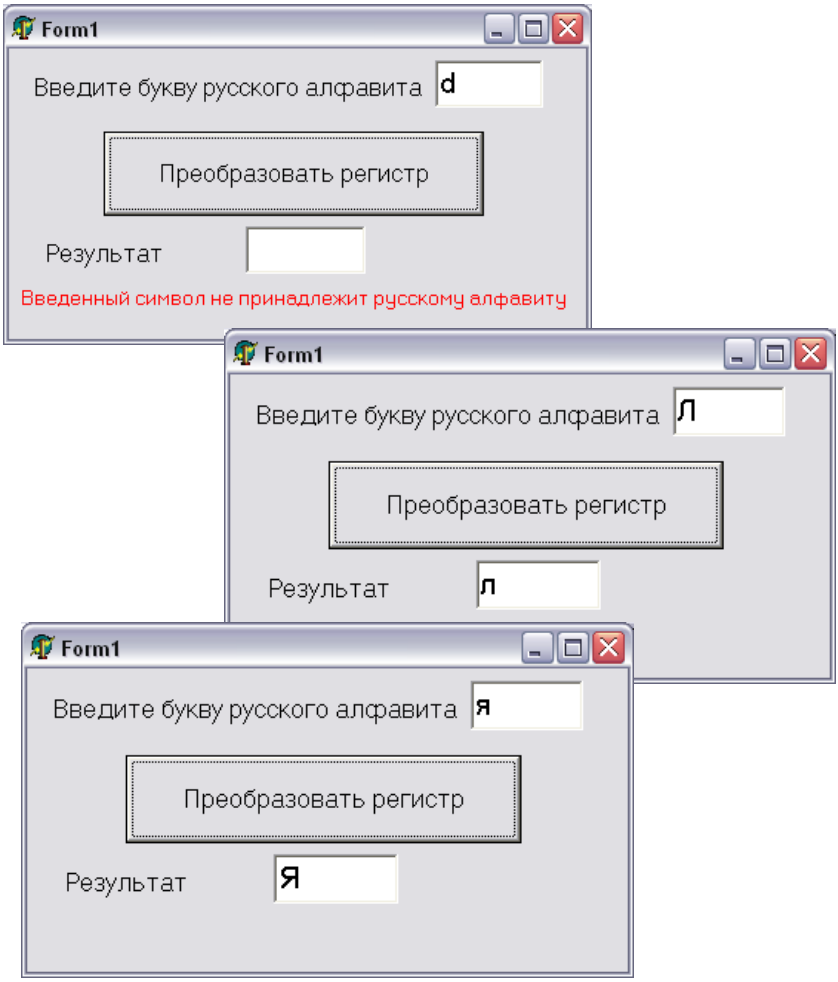


Рисунок 40 – **Примеры выполнения проекта по преобразованию регистра введенной буквы**

Тексты процедур проекта приведены на рисунках 41 и 42.

```
procedure TForm1.Button1Click(Sender: TObject);
//Преобразование строчной буквы в прописную и наоборот
  var
    s:string[1];
    c,d:char;
    k:integer;
begin
    s:=Edit1.Text; c:=s[1];
    if (c>='а') and (c<='я')
      then //введена строчная русская буква
        begin
         //преобразование в код верхнего регистра
          k:=Ord(c)-Ord('а');
          d:=Chr(k+Ord('А'));
          Edit2.Text:=d
        end
      else
        if (c>='А') and (c<='Я')
          then //введена прописная русская буква
            begin
              //преобразование в код нижнего регистра
              k:=Ord(c)-Ord('А');
              d:=Chr(k+Ord('а'));
              Edit2.Text:=d
            end
          else
            Label3.Show;//сообщение об ошибке
end;
```

Рисунок 41 – **Текст процедуры обработки щелчка по кнопке *Преобразовать***

```
procedure TForm1.Edit1Click(Sender: TObject);
//обработка событий Click и Change поля Edit1
begin
  Edit2.Text:='';
  Label3.Hide
end;
procedure TForm1.FormActivate(Sender: TObject);
//активация формы
begin
  Edit1.Text:='';
  Edit1.SetFocus;
  Edit2.Text:='';
  Label3.Hide
end;
```

Рисунок 42 – **Тексты вспомогательных процедур проекта преобразования регистра для введенной буквы**

Пример 2. Требуется разработать проект, который в заданной строке дублирует каждый символ «Э», а если перед этим символом стоит запятая, то удаляет ее.

Интерфейс проекта приведен на рисунке 43.

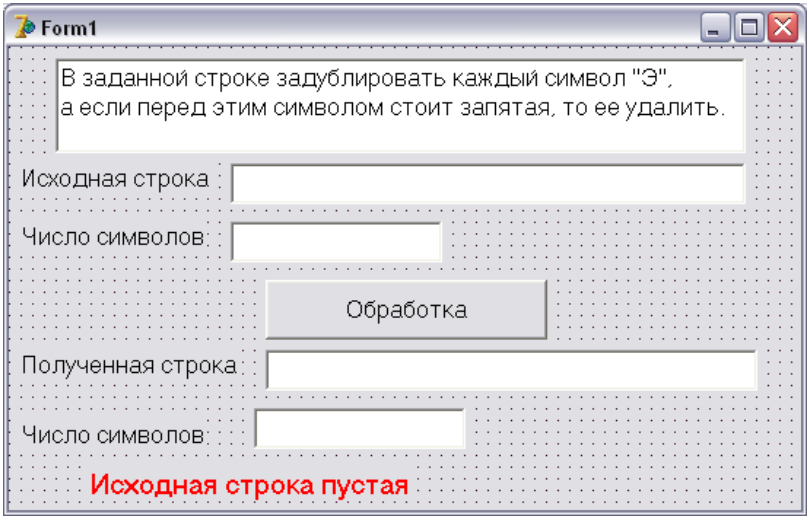


Рисунок 43 – **Интерфейс проекта**

Текст процедуры обработки щелчка по кнопке *Обработка* приведен на рисунке 44.

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    s:string;
    i,n:integer;
begin
    s:=Edit1.Text;//ввод исходной строки
    n:=Length(s);//длина исходной строки
    Edit3.Text:=IntToStr(n);
    if n=0
      then //задана пустая строка
        Label5.Show
      else
        begin
          Label5.Hide;
          i:=1;
          while i<=n do //цикл посимвольного просмотра строки
              if s[i]='Э'
                then //найден символ 'Э'
                  begin
                    Insert(s[i],s,i+1);//дублирование 'Э'
                    if (i>1) and (s[i-1]=',')
                      then //перед 'Э' стоит запятая
                        begin
                          Delete(s,i-1,1);//удаление запятой
                          i:=i+1 //переход к следующему символу
                        end
                      else //перед 'Э' нет запятой
                        begin
                          i:=i+2; //переход к символу после 'Э'
                          n:=n+1 //увеличение длины строки
                        end
                  end
                else //переход к следующему символу
                    i:=i+1;
          Edit2.Text:=s; //вывод полученной строки
          Edit4.Text:=IntToStr(n)//вывод ее длины
        end;
end;
```

Рисунок 44 – **Текст процедуры**

Пример 3. Требуется разработать проект, который в заданной строке, состоящей из слов, разделенных произвольным числом пробелов, подсчитывает количество слов в строке и определяет длину самого длинного слова.

Окно приложения проекта приведено на рисунке 45.

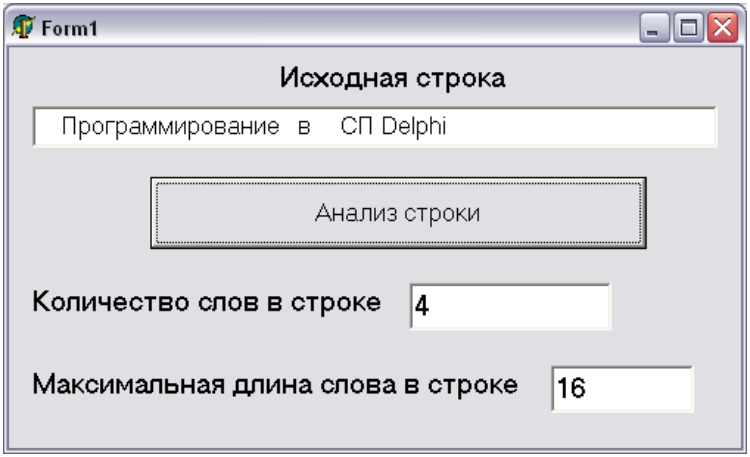


Рисунок 45 – **Окно приложения проекта, анализирующего слова в строке**

Алгоритм решения задачи заключается в следующем: организуется цикл (внешний цикл) посимвольного просмотра строки с целью поиска начала очередного слова, т. е. пропускаются пробелы. Как только находится первая буква слова, то счетчик слов увеличивается на 1 и начинается цикл (внутренний цикл) поиска конца слова с одновременным подсчетом числа букв в анализируемом слове. Признаком конца слова является пробел или конец строки (для последнего слова). После нахождения конца слова его длина сравнивается с найденным ранее максимальным значением. Если новое значение больше, то оно принимается за максимальное.

Текст процедуры обработки щелчка по кнопке *Анализ строки* приведен на рисунках 46 и 47. На рисунке 46 приведена часть текста, включающая раздел описаний, ввод исходной строки, организацию внешнего цикла (поиск начала очередного слова) и заключительную часть процедуры с выводом результатов. На рисунке 47 приведена часть текста процедуры, включающая организацию внутреннего цикла – поиск конца слова и определение максимальной на данный момент длины слова.

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    s:string[80];
    n,i,k,l,m:integer;
    W:Boolean;
begin
    s:=Edit1.Text;
    n:=Length(s);
    if n=0
      then Edit2.Text:='Строка не задана'
      else
        begin
          k:=0;//количество слов в строке
          m:=0;//длина самого длинного слова
          W:=false;//не найдено начало слова
          i:=1;// номер анализируемого символа
          while i<=n do //цикл поиска начала слова
            if s[i]=' ' //пробел?
              then i:=i+1
              else // найден символ, отличный от пробела
                //начало обработки очередного слова
```

Здесь должен находиться текст программы, приведенный на рисунке 47.

```
          if W {в конце исходной строки не было пробела,
          т.е. еще не была оценена длина последнего слова}
            then
              if l>m
                then m:=l;
          if k=0
            then Edit2.Text:='В строке нет слов'
            else //вывод результатов
              begin
                Edit2.Text:=IntToStr(k);
                Edit3.Text:=IntToStr(m)
              end;
        end;
end;
```

Рисунок 46 – **Организация поиска слова и вывод результатов**

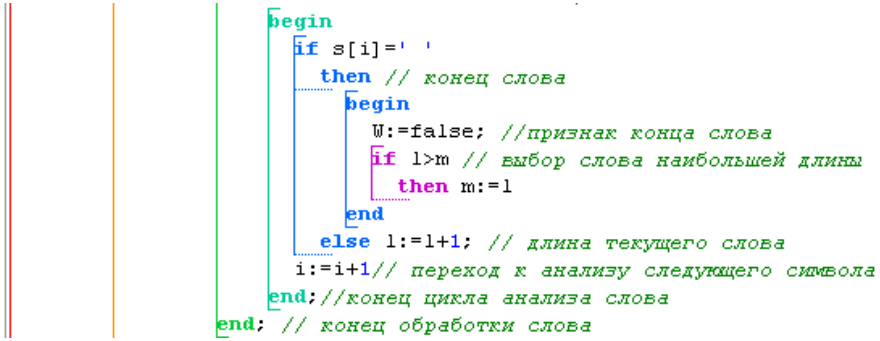


Рисунок 47 – **Обработка очередного слова строки**

8. ИСПОЛЬЗОВАНИЕ ЗАПИСЕЙ

8.1. Описание типа для записи

Часто требуется группировать данные разного типа, логически относящиеся к одному объекту. Например, целесообразно объединить в одну структуру данные о товаре (наименование товара, единица измерения, цена за единицу, процент торговой надбавки). Все эти данные имеют разный тип и (или) длину.

Для реализации таких структур в языке Паскаль имеется комбинированный тип, называемый записью.

Запись – это структурированный тип данных, состоящий из фиксированного числа компонентов разного типа. Определение типа записи начинается со слова *record* и заканчивается словом *end*. Между этими словами заключен список компонентов, называемых полями, с указанием идентификаторов полей и типа каждого поля.

В качестве примера опишем тип записи для упоминавшейся ранее совокупности данных о товаре:

```
Type
    TTovar = record
                NaimTov: string[20]; //наименование товара
                EdIzm: string[8]; //единица измерения
                Cena: integer; // цена за единицу
                PrNadb: real //торговая надбавка
             end;
  Var
    Tovar1, Tovar2: TTovar; //две записи типа TTovar
```

Значения полей записи могут быть использованы в выражениях так же, как и простые переменные. Но в отличие от простых переменных имя поля записывается как *составное имя* – идентификатор переменной, имеющей тип записи, а затем после точки – идентификатор поля. Например: *Tovar1.NaimTov, Tovar1.EdIzm, Tovar1.Cena, Tovar1.PrNadb.*

Tovar1.NaimTov:='Сок апельсиновый';

Допускается применение оператора присваивания и к записям в целом, если они имеют один и тот же тип. Например:

Tovar2:= Tovar1;

8.2. Массивы записей

Во многих задачах удобно использовать массивы записей. С помощью описанного выше типа *TTovar* массив записей *Tovary* можно описать следующим образом:

```
Var
   Tovary: array[1..30] of TTovar;
```

Существует и второй способ описания массива *Tovary*, при котором сначала описывается тип массива записей, а потом уже сам массив:

```
Type
    TTovar = record
                  NaimTov: string[20];
                  EdIzm: string[8];
                  Cena: integer;
                  PrNadb: real
                end;
    TMasTovary= array[1..30] of TTovar;
 Var
    Tovary: TMasTovary;
```

Обращение к полям массива записей выполняется с помощью составного имени следующим образом:

Tovary[i].NaimTov:= 'Сок апельсиновый';

Документ «Товарно-транспортная накладная» (ТТН), пример которого приведен на рисунке 48, можно представить в программе массивом записей, каждый элемент которого содержит информацию из одного документа. В свою очередь, для представления в программе одной накладной нужен еще один массив записей, соответствующий табличной части документа.

Товарно-транспортная накладная
№ 375 от 02.03.2010 г.
о поступлении товаров от ОАО «Гомельские баранки»

Наименование товара	Единица измерения	Количество	Цена за единицу, р.
Желе вишневое	кг	100	2800
Сок ананасовый	л	150	3000
Бублик	шт.	50	1200

Рисунок 48 – **Пример документа для представления в программе массивом записей**

Описание соответствующих типов и переменных для отображения реквизитов ТТН приведено на рисунке 49. Обратите внимание на последовательность описания этой сложной структуры. Сначала описан тип *TStrNakl*, отражающий структуру одной строки накладной, содержащей четыре поля (наименование товара, единица измерения, количество, цена). Затем описан тип *TMasStrNakl*, определяющий массив строк накладной. В соответствии с этим описанием мы установили следующее ограничение: в накладной не может быть более 20 строк. Далее описан тип *TNakladnaja*, определяющий заголовочную часть документа. К полям, соответствующим реквизитам документа (номер ТТН, дата, приход или расход, контрагент) добавлено поле *KolStr*, значение которого определяет количество строк в накладной. Это значение надо определять при вводе реквизитов накладной. В качестве поля записи типа *TNakladnaja* включен массив строк *MasStrNakl*.

Переменная *MasNakl* определяет сам массив накладных. Переменная *KolNakl* используется для хранения количества введенных накладных.

Для использования поля некоторой строки очередной накладной необходимо ввести составное имя, примером которого может служить следующая запись:

MasNakl[i].MasStrNakl[j].Tovar:='Сок апельсиновый';

```
type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    procedure N1Click(Sender: TObject);
    procedure N2Click(Sender: TObject);
    procedure N3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
   TStrNakl = record //тип для строки табличной части ТТН
               Tovar:string[20];//наименование товара
               EdIzm:string[10];//единица измерения
               Cena:real; //цена за единицу
               Kolich:real //количество товара
             end;
  TMasStrNakl = array[1..20] of TStrNakl;//тип для массива строк
  TNakladnaja = record //тип для ТТН
                  NomerNakl:string[6];//номер ТТН
                  DataNakl:string[10];//дата
                  PrichRasch:Boolean;//приходная или расходная
                  Kontragent:string[20];//контрагент
                  KolStr:integer;//количество строк в ТТН
                  MasStrNakl:TMasStrNakl
                end;
   TMasNakl = array[1..30] of TNakladnaja;//тип для массива ТТН
var
  Form1: TForm1;
  MasNakl:TMasNakl; //массив ТТН
  KolNakl:integer; //количество ТТН в массиве
```

Рисунок 49 – **Описание типов и переменных для представления набора ТТН**

Схематично структура записи одной накладной (тип *TNakladnaja*) и массива записей о накладных (тип *TMasNakl*) представлен на рисунке 50.

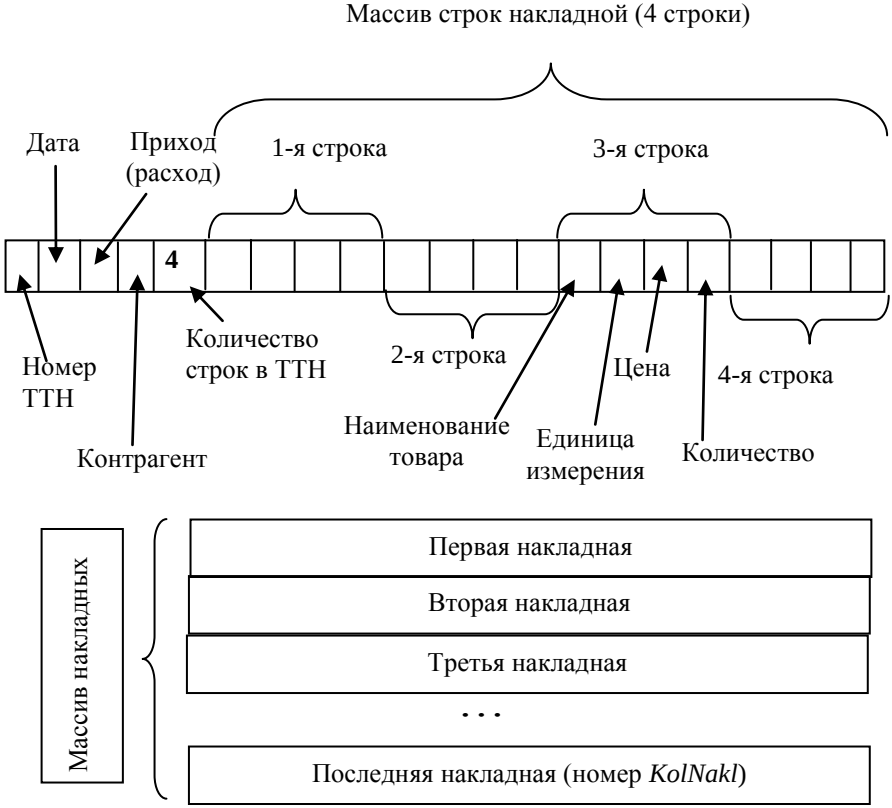


Рисунок 50 – **Структура записи для одной накладной и структура записи массива накладных**

8.3. Оператор присоединения

Приведенный выше пример оператора присваивания для поля массива записей показывает, что обращение к полю массива записей выглядит довольно громоздко. Оператор присоединения упрощает написание доступа к полю записи. Оператор в данном случае имеет следующий общий вид:

with *переменная типа запись* ***do***
оператор;

С помощью оператора присоединения оператор
MasNakl[i].MasStrNakl[j].Tovar:='Сок апельсиновый'; можно записать следующим образом:

```
with MasNakl[i].MasStrNakl[j] do
    Tovar:='Сок апельсиновый';
```

То же самое присваивание можно записать с помощью вложенных операторов присоединения следующим образом:

```
with MasNakl[i] do
    with MasStrNakl[j] do
        Tovar:='Сок апельсиновый';
```

8.4. Пример использования записей в многомодульном проекте

В качестве примера рассмотрим задачу ввода ТТН и их обработку. Для этого создадим проект, который выполняет следующие функции:

- ввод информации из набора ТТН в массив записей;
- вычисление общей стоимости товаров по каждой ТТН;
- вычисление общей стоимости приобретенных товаров по всем приходным накладным;
- вычисление общей стоимости проданных товаров по всем расходным накладным;
- просмотр введенных ТТН.

Распределение функций проекта по его модулям приведено на рисунке 51.

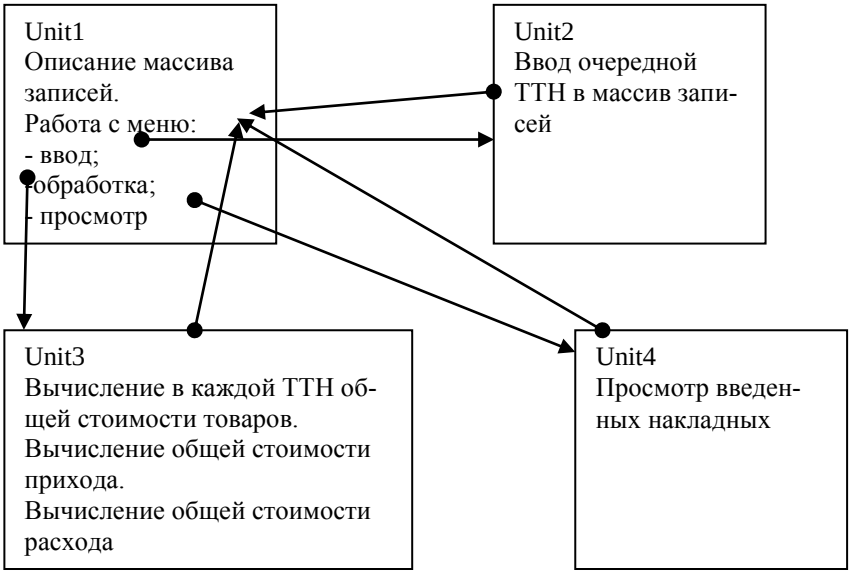
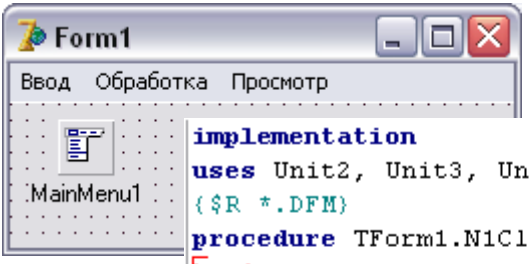


Рисунок 51 – **Функционально-модульная структура проекта обработки ТТН на основе массива записей**

Окно формы модуля *Unit1* (рисунок 52) содержит только один визуальный компонент *MainMenu1*, позволяющий пользователю выбрать одну из трех команд. Текст интерфейсной части модуля с описанием типов и переменных для обработки массива накладных приведен на рисунке 49. В исполняемой части модуля (рисунок 52) имеется инструкция *uses Unit2, Unit3, Unit4;*, благодаря которой типы и переменные из интерфейсной части модуля *Unit1* будут доступны из трех других модулей. Таким образом, все эти описания являются глобальными для всего проекта.




```
implementation
uses Unit2, Unit3, Unit4;
{$R *.DFM}
procedure TForm1.N1Click(Sender: TObject);
begin
  Form2.Show //Ввод ТТН
end;
procedure TForm1.N2Click(Sender: TObject);
begin
  Form3.Show //Обработка ТТН
end;
procedure TForm1.N3Click(Sender: TObject);
begin
  Form4.Show //Просмотр ТТН
end;
initialization
  KolNakl:=0 //количество введенных ТТН
end.
```

Рисунок 52 – **Окно формы, исполняемая и инициализирующая части модуля *Unit1***

В исполняемой части модуля *Unit1* каждая из трех процедур обработки команд меню вызывает соответствующее окно формы. В инициализирующей части этого модуля происходит обнуление глобальной переменной *KolNakl*, фиксирующей количество введенных ТТН (рисунок 52).

Окно формы *Form2* в режиме выполнения (окно приложения) приведено на рисунке 53.

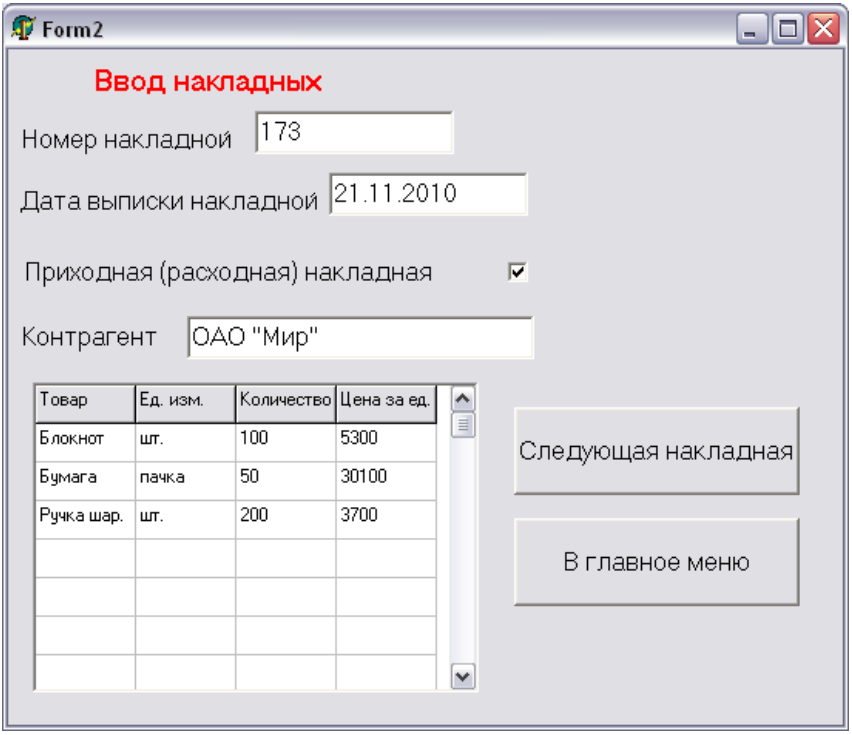


Рисунок 53 – **Окно формы *Form2* в режиме выполнения**

Процедура обработки щелчка по кнопке *Button1* (*Следующая накладная*) должна ввести все заполненные реквизиты ТТН в поля массива записей, а затем очистить поля для набора пользователем следующей ТТН. Если пользователь не заполнил поле *Номер накладной*, то чтение всех других значений не производится. Реквизиты содержательной части накладной, введенные пользователем в ячейки таблицы *StringGrid1*, вводятся до тех пор, пока не встретится пустое поле *Товар*.

Текст процедуры обработки щелчка по кнопке *Button1* (*Следующая накладная*) приведен на рисунке 54. На рисунке 55 приведен начальный фрагмент исполняемой части модуля *Unit2* с инструкцией *uses Unit1;*, обеспечивающей использование в этом модуле глобальных переменных, описанных в модуле *Unit1*, и текст процедуры активации формы *Form2*, которая заполняет названия граф таблицы *StringGrid1*.

```
procedure TForm2.Button1Click(Sender: TObject);
  var
    W:Boolean;
    s:string[20];
    i:integer;
begin
  s:=Edit1.Text;
  if s<>''// номер накладной задан
    then
      begin
        KolNakl:=KolNakl+1;
        with MasNakl[KolNakl] do
          begin
            //ввод реквизитов заголовочной части ТТН
            NomerNakl:=Edit1.Text;
            DataNakl:=Edit2.Text;
            PrichRasch:=CheckBox1.Checked;
            Kontragent:=Edit3.Text;
            KolStr:=0;//число строк в табличной части ТТН
            W:=True;
            while W do // цикл считывания всех строк табличной части ТТН
              begin
                s:=StringGrid1.Cells[0,KolStr+1];
                // наименование товара из очередной строки ТТН
                if s=''
                  then W:=false
                  else
                    begin
                      KolStr:=KolStr+1;
                      with MasStrNakl[KolStr] do
                      //ввод реквизитов очередной строки ТТН
                        begin
                          Tovar:=s;
                          EdIzm:=StringGrid1.Cells[1,KolStr];
                          Cena:= StrToFloat(StringGrid1.Cells[2,KolStr]);
                          Kolich:=StrToFloat(StringGrid1.Cells[3,KolStr]);
                        end;
                    end;
              end;
          end;
      end;
  // чистка всех полей формы кроме заголовков граф таблицы
  Edit1.Text:='';
  Edit2.Text:='';
  Edit3.Text:='';
  CheckBox1.Checked:=False;
  for i:=1 to StringGrid1.RowCount-1 do
    begin
      StringGrid1.Cells[0,i]:='';
      StringGrid1.Cells[1,i]:='';
      StringGrid1.Cells[2,i]:='';
      StringGrid1.Cells[3,i]:='';
    end;
end;
```

Рисунок 54 – **Текст процедуры ввода очередной ТТН**

```
implementation
  uses Unit1;
{$R *.DFM}
procedure TForm2.FormActivate(Sender: TObject);
begin
  StringGrid1.Cells[0,0]:='Товар';
  StringGrid1.Cells[1,0]:='Ед. изм.';
  StringGrid1.Cells[2,0]:='Количество';
  StringGrid1.Cells[3,0]:='Цена за ед.';
end;
```

Рисунок 55 – **Начало исполняемой части модуля *Unit*2**

При щелчке по кнопке *Button2* (*В главное меню*) необходимо выполнить те же действия, что и при щелчке по кнопке *Button1* (*Следующая накладная*), и дополнительно закрыть окно формы *Form2* с помощью оператора *Close*. Таким образом, для создания текста процедуры *TForm2.Button2Click* надо в текст процедуры *TForm2.Button1Click* добавить оператор *Close.*

Окно формы *Form3* в режиме выполнения (окно приложения) приведено на рисунке 56.

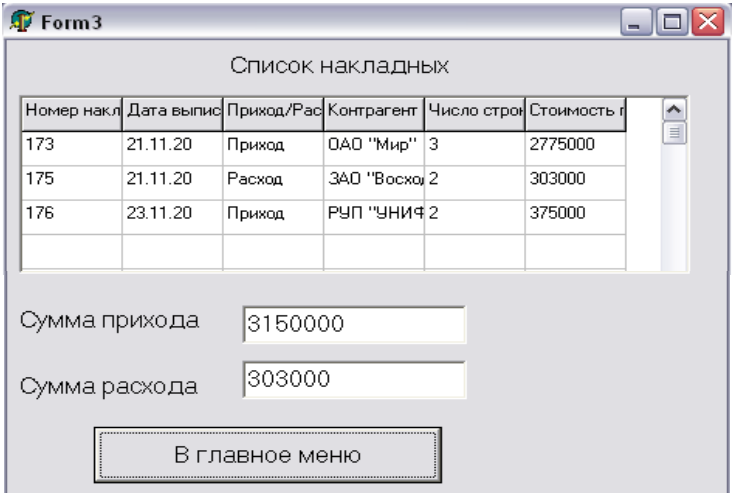


Рисунок 56 – **Окно формы *Form3* в режиме выполнения**

Все вычисления и заполнение полей формы выполняются при активации формы. Текст процедуры приведен на рисунке 57. В испол-

няемой части модуля помещена инструкция *uses Unit1;* для использования глобальных переменных проекта.

```
procedure TForm3.FormActivate(Sender: TObject);
  var
    i,j:integer;
    Stoim,StoimPr,StoimRs:real;
begin
  StringGrid1.Cells[0,0]:='Номер накладной';
  StringGrid1.Cells[1,0]:='Дата выписки';
  StringGrid1.Cells[2,0]:='Приход/Расход';
  StringGrid1.Cells[3,0]:='Контрагент';
  StringGrid1.Cells[4,0]:='Число строк';
  StringGrid1.Cells[5,0]:='Стоимость по накладной';
  StoimPr:=0;
  StoimRs:=0;
  for i:=1 to KolNakl do
  //цикл по всем накладным
    with MasNakl[i] do
      begin
        StringGrid1.Cells[0,i]:=NomerNakl;
        StringGrid1.Cells[1,i]:=DataNakl;
        if PrichRasch //приход или расход
          then StringGrid1.Cells[2,i]:='Приход'
          else StringGrid1.Cells[2,i]:='Расход';
        StringGrid1.Cells[3,i]:=Kontragent;
        StringGrid1.Cells[4,i]:=IntToStr(KolStr);
        Stoim:=0;
        for j:=1 to KolStr do
        //цикл по всем строка накладной для вычисления
        //общей стоимости по накладной
          Stoim:= Stoim+MasStrNakl[j].Cena*MasStrNakl[j].Kolich;
        StringGrid1.Cells[5,i]:=FloatToStr(Stoim);
        if PrichRasch //приход или расход
          then StoimPr:=StoimPr+Stoim
          else StoimRs:=StoimRs+Stoim;
      end;
  Edit1.Text:=FloatToStr(StoimPr);
  Edit2.Text:=FloatToStr(StoimRs);
end;
```

Рисунок 57 – **Основная процедура модуля *Unit3***

Текст процедуры обработки щелчка по кнопке *Button1* (*В главное меню*) состоит из одной команды – *Close*.

Окно формы *Form4* в режиме проектирования приведено на рисунке 58.

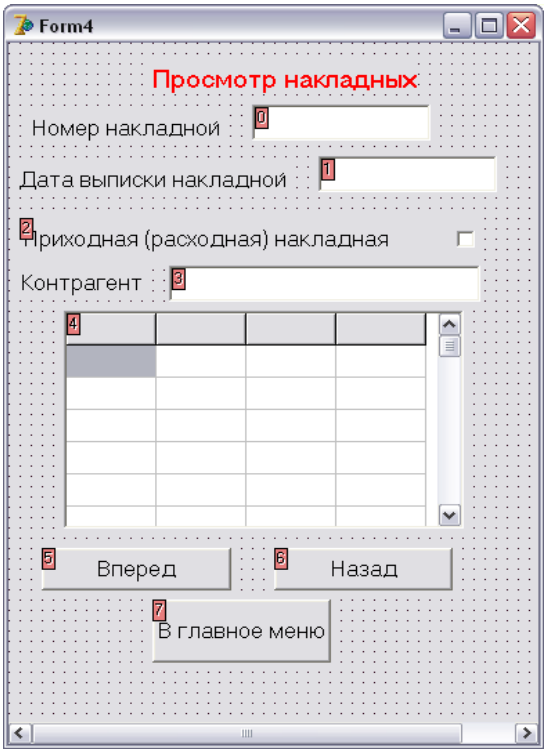


Рисунок 58 – **Окно формы *Form4***

Исполняемая часть модуля содержит инструкцию *uses Unit1;* для использования глобальных переменных проекта и четыре процедуры обработки событий:

- активация формы – вывод в поля формы реквизитов первой накладной из массива записей;
- обработка щелчка по кнопке *Вперед* – вывод реквизитов следующей накладной (если в поля формы выведена последняя накладная из массива, то выдается сообщение об этом и содержимое полей формы не меняется);
- обработка щелчка по кнопке *Назад* – вывод реквизитов предыдующей накладной (если в поля формы выведена первая накладная из массива, то выдается сообщение об этом и содержимое полей формы не меняется);

• обработка щелчка по кнопке *Главное меню* – закрытие окна формы (оператор *Close*).

Для номера накладной, отображенной в окне формы, используется переменная *TecNakl*, которой в инициализирующей части модуля присвоено значение 1.

Текст процедуры активации формы приведен на рисунке 59.

Текст процедуры отображения следующей накладной приведен на рисунках 60 и 61.

```
procedure TForm4.FormActivate(Sender: TObject);
var
  j:integer;
begin
 if KolNakl=0
   then
     Label5.Caption:='Накладных нет'
   else
    with MasNakl[TecNakl] do //TecNakl=1
      begin
        Edit1.Text:=NomerNakl;
        Edit2.Text:=DataNakl;
        if PrichRasch
          then CheckBox1.Checked:=True
          else CheckBox1.Checked:=False;
        Edit3.Text:=Kontragent;
        StringGrid1.Cells[0,0]:='Товар';
        StringGrid1.Cells[1,0]:='Ед. изм.';
        StringGrid1.Cells[2,0]:='Количество';
        StringGrid1.Cells[3,0]:='Цена за ед.';
        for j:=1 to KolStr do
        //цикл по всем строкам накладной
          with MasStrNakl[j] do
            begin
              StringGrid1.Cells[0,j]:=Tovar;
              StringGrid1.Cells[1,j]:=EdIzm;
              StringGrid1.Cells[2,j]:=FloatToStr(Cena);
              StringGrid1.Cells[3,j]:=FloatToStr(Kolich);
            end;
      end;
end;
```

Рисунок 59 – **Процедура активации формы *Form4***

```
procedure TForm4.Button1Click(Sender: TObject);
//отображение следующей накладной
var
  j:integer;
begin
 if TecNak1=KolNak1
   then
     Label5.Caption:='Последняя накладная'
   else
     begin
       Label5.Caption:='';
       TecNak1:=TecNak1+1;//номер следующей ТТН
       with MasNak1[TecNak1] do
         begin
           Edit1.Text:=NomerNak1;
           Edit2.Text:=DataNak1;
           if PrichRasch
             then CheckBox1.Checked:=True
             else CheckBox1.Checked:=False;
           Edit3.Text:=Kontragent;
           StringGrid1.Cells[0,0]:='Товар';
           StringGrid1.Cells[1,0]:='Ед. изм.';
           StringGrid1.Cells[2,0]:='Количество';
           StringGrid1.Cells[3,0]:='Цена за ед.';
```

Рисунок 60 – **Начальная часть процедуры отображения следующей ТТН**

```
           for j:=1 to KolStr do
           //цикл вывода строк ТТН
             with MasStrNak1[j] do
               begin
                 StringGrid1.Cells[0,j]:=Tovar;
                 StringGrid1.Cells[1,j]:=EdIzm;
                 StringGrid1.Cells[2,j]:=FloatToStr(Cena);
                 StringGrid1.Cells[3,j]:=FloatToStr(Kolich);
               end;
            for j:=KolStr+1 to StringGrid1.RowCount-1 do
            //цикл очистки незанятых строк таблицы
             with MasStrNak1[j] do
               begin
                 StringGrid1.Cells[0,j]:='';
                 StringGrid1.Cells[1,j]:='';
                 StringGrid1.Cells[2,j]:='';
                 StringGrid1.Cells[3,j]:='';
               end;
         end;
     end;
end;
```

Рисунок 61 – **Заключительная часть процедуры отображения следующей ТТН**

Текст процедуры отображения предыдущей накладной отличается от текста, приведенного на рисунках 60 и 61, только двумя операторами – текстом выводимого сообщения (*Первая накладная*) и изменением номера текущей ТТН (уменьшается на 1). Соответствующий фрагмент рассматриваемой процедуры приведен на рисунке 62.

```
procedure TForm4.Button2Click(Sender: TObject);
//отображение предыдущей накладной
var
  j:integer;
begin
 if TecNakl=1
   then
     Label5.Caption:='Первая накладная'
   else
     begin
       Label5.Caption:='';
       TecNakl:=TecNakl-1; //номер предыдущей ТТН
```

Рисунок 62 – **Фрагмент процедуры отображения предыдущей ТТН**

9. РАБОТА С ТИПИЗИРОВАННЫМИ ФАЙЛАМИ

9.1. Описание файлового типа

Файл с точки зрения языка Паскаль – это структурированный тип данных, состоящий из последовательности компонентов одного типа и одной длины. Если компонентами файлов являются записи определенного типа, то такие файлы называются *типизированными файлами*.

Описание типизированного файла задается следующим образом:

```
Type
   <имя записи> = record
            ;
            ;
            ;

   end;
Var
   <файловая переменная>: file of <имя записи>;
```

Заметим, что число компонентов файла, называемое длиной файла, не указывается в описании типа файла, а также и при описании

массива записей необходимо указывать максимально возможное значение числа записей.

Следует отметить, что переменные файлового типа нельзя использовать в выражениях. Файл хранится во внешней памяти компьютера, и по мере необходимости данные из компонента файла вызываются в оперативную память для обработки. Этот процесс называется *чтением из файла*. В оперативной памяти данные из файла представлены полями записи. Обратным по отношению к процессу чтения из файла является процесс *записи в файл*, когда значения полей записи, находящейся в оперативной памяти, переписываются в компонент файла.

9.2. Доступ к компонентам файла

Доступ к компонентам файла осуществляется через *указатель файла*. Это специальная переменная целого типа, которая указывает на определенный компонент файла. При чтении или записи указатель перемещается к следующему компоненту и делает его доступным для обработки. Такая переменная не может участвовать в обычных выражениях и операторах присваивания.

Есть два способа доступа к компонентам файла – последовательный и произвольный (прямой).

При *последовательном доступе* поиск начинается с начала файла и проверяется по очереди каждый компонент, пока не будет найден нужный.

Произвольный способ доступа позволяет обращаться к компоненту по его порядковому номеру в файле.

9.3. Стандартные процедуры обработки файлов

В данном пункте будем использовать следующие обозначения:

- *fv* – файловая переменная;
- *str* – строка;
- *p* – переменная типа «запись»; этот же тип имеют компоненты переменной *fv*;
- *n* – целочисленное выражение.

При работе с файлами используются следующие стандартные процедуры:

- *AssignFile (fv, str)* – присвоение имени, заданного строкой *str* файлу, с которым связана файловая переменная *fv*.

• *Rewrite (fv)* – создание (открытие) нового файла, с которым связана файловая переменная *fv*. Если на диске имеется файл с таким именем, то он уничтожается. При этом указатель файла устанавливается в положение 0 (начало файла).

• *Reset (fv)* – открытие файла, с которым связана файловая переменная *fv*, с установкой указателя файла в начальное положение.

• *Read (fv, p)* – чтение из файла, с которым связана файловая переменная *fv*, полей текущего компонента в запись *p*. После завершения процедуры указатель перемещается на следующий компонент.

• *Write (fv, p)* – запись в файл, с которым связана файловая переменная *fv*, полей записи *p* в текущий компонент файла. После выполнения процедуры указатель перемещается к следующему компоненту.

• *Seek (fv, n)* – установка указателя файла, с которым связана файловая переменная *fv*, на компонент с порядковым номером *n*. Нумерация компонентов файла начинается с 0.

• *CloseFile (fv)* – закрытие файла, с которым связана файловая переменная *fv* (нельзя выходить из программы без закрытия файла).

• *Erase (fv)* – уничтожение файла, с которым связана файловая переменная *fv* (нельзя уничтожить открытый файл).

• *Rename (fv, str)* – переименование файла, с которым связана файловая переменная *fv*, новая имя определяется значением *str*.

• *Truncate (fv)* – уничтожение всех компонентов файла, начиная с места текущего положения указателя.

9.4. Стандартные функции обработки файлов

При работе с файлами используются следующие стандартные функции:

• *Eof (fv)* – возвращает значение *True*, если указатель файла, с которым связана файловая переменная *fv*, находится сразу за последним компонентом файла и *False* – в любом другом случае.

• *FilePos (fv)* – определяет текущий номер компонента файла, с которым связана файловая переменная *fv*. Функция возвращает целочисленное значение, равное номеру компонента, на котором установлен в данный момент указатель файла. Отсчет номера компонента начинается с 0.

• *FileSize (fv)* – определяет длину файла (количество компонентов), с которым связана файловая переменная *fv*. Если ответ равен 0, то файл пуст.

• *FileExists (str)* – возвращает значение *True*, если файл с именем *str* существует, и *False* – в противном случае.

9.5. Пример обработки файла «Товарооборот райпо»

В качестве примера решим задачу ввода и обработку ТТН, рассмотренную в пункте 8.4, с использованием типизированного файла вместо массива записей. Итак, создадим проект, который выполняет следующие функции:

- ввод информации из набора ТТН и запись их в файл *Накладные.dat*;
- вычисление общей стоимости товаров по каждой ТТН, содержащейся в файле *Накладные.dat*;
- вычисление общей стоимости приобретенных товаров по всем приходным накладным, содержащихся в файле *Накладные.dat*;
- вычисление общей стоимости проданных товаров по всем расходным накладным, содержащихся в файле *Накладные.dat*;
- просмотр ТТН, содержащихся в файле *Накладные.dat*.

Распределение функций проекта по его модулям приведено на рисунке 63.

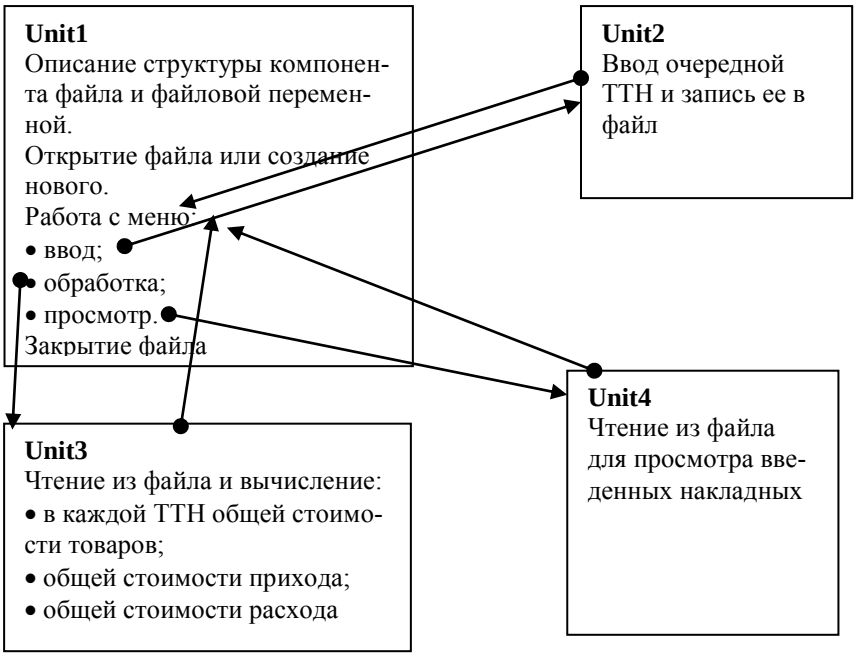


Рисунок 63 – **Функционально-модульная структура проекта обработки ТТН на основе типизированного файла**

Окно формы модуля *Unit1* (рисунок 64) содержит только один визуальный компонент *MainMenu1*, позволяющий пользователю выбрать одну из четырех команд. Текст интерфейсной части модуля с описанием типов и переменных для обработки файла накладных приведен на рисунке 65.

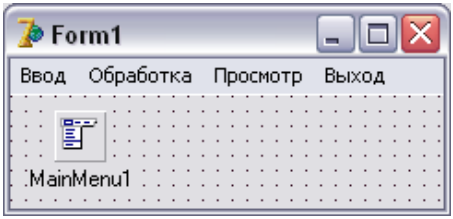


Рисунок 64 – **Окно формы модуля *Unit1***

```
   TStrNakl = record //структура строки ТТН
                Tovar:string[20];
                EdIzm:string[10];
                Cena:real;
                Kolich:real
              end;
  TMasStrNakl = array[1..20] of TStrNakl;
  TNakladnaja = record //структура ТТН
                  NomerNakl:string[6];
                  DataNakl:string[8];
                  PrichRasch:Boolean;
                  Kontragent:string[20];
                  KolStr:integer;
                  MasStrNakl:TMasStrNakl
                end;
var
  Form1: TForm1;
  FNakl:file of TNakladnaja;//файловая переменная
  KolNakl:integer; //количество ТТН
  RNakl:TNakladnaja; //запись со структурой ТТН
```

Рисунок 65 – **Фрагмент интерфейсной части модуля *Unit1***

В исполняемой части модуля (рисунок 66) имеется инструкция *uses Unit2;, Unit3; Unit4;*, благодаря которой типы и переменные из интерфейсной части модуля *Unit1* будут доступны из трех других

модулей. Таким образом, все эти описания являются глобальными для всего проекта. В исполняемой части модуля *Unit1* каждая из процедур обработки первых трех команд меню вызывает соответствующее окно формы, а процедура обработки команды *Выход* содержит только один оператор *Close*, закрывающий окно *Form1*.

В инициализирующей части этого модуля (рисунок 66) происходит создание нового файла *Накладные.dat* или его открытие, если он был уже создан при предыдущих запусках проекта.

Завершающая часть модуля *Unit1* содержит один оператор закрытия файла (рисунок 66).

Окно формы *Form2* в режиме выполнения (окно приложения) приведено на рисунке 53.

```
implementation
uses Unit2, Unit3, Unit4;
{$R *.DFM}
procedure TForm1.N1Click(Sender: TObject);
begin  //Вызов модуля "Ввод"
  Form2.Show
end;
procedure TForm1.N2Click(Sender: TObject);
begin  //Вызов модуля "Обработка"
  Form3.Show
end;
procedure TForm1.N3Click(Sender: TObject);
begin  //Вызов модуля "Просмотр"
  Form4.Show
end;
procedure TForm1.N4Click(Sender: TObject);
begin  //Окончание работы
  Close
end;
initialization
  //Связывание имени файла с файловой переменной
  Assign(FNakl,'Накладные.dat');
  if FileExists('Накладные.dat')
    then //открытие существующего файла
      Reset(FNakl)
    else //создание нового файла
      Rewrite(FNakl);
finalization
  CloseFile(FNakl)
end.
```

Рисунок 66 – **Исполняемая, инициализирующая и завершающая части модуля *Unit1***

Процедура обработки щелчка по кнопке *Button1* (*Следующая накладная*) должна ввести все заполненные реквизиты ТТН в поля записи *RNakl*, а затем записать *RNakl* в первый свободный компонет файла *FNakl*. После записи в файл надо очистить поля формы для набора пользователем следующей ТТН. Если пользователь не заполнил поле *Номер накладной*, то чтение всех других значений из полей формы не производится. Реквизиты содержательной части накладной, введенные пользователем в ячейки таблицы *StringGrid1*, вводятся до тех пор, пока не встретится пустое поле *Товар*. Текст процедуры обработки щелчка по кнопке *Button1* (*Следующая накладная*) приведен на рисунках 67 и 68.

```
procedure TForm2.Button1Click(Sender: TObject);
  var
    W:Boolean; s:string[20]; i:integer;
begin
  s:=Edit1.Text;
  if s<>'' //Задан ли номер накладной?
    then
      begin
        //Указатель файла устанавливается на конец файла
        Seek(FNakl,FileSize(FNakl));
        with RNakl do
          begin
            //Перепись содержимого полей в переменную RNakl
            NomerNakl:=Edit1.Text;
            DataNakl:=Edit2.Text;
            PrichRasch:=CheckBox1.Checked;
            Kontragent:=Edit3.Text;
            KolStr:=0;
            W:=True;
            while W do //Цикл ввода информации из строк таблицы
              begin
                s:=StringGrid1.Cells[0,KolStr+1];
                if s='' //Задано ли имя товара?
                  then W:=false
                  else
                    begin
                      KolStr:=KolStr+1;
                      with MasStrNakl[KolStr] do
                        begin
                          //Перепись полей из таблицы в переменную RNakl
                          Tovar:=s;
                          EdIzm:=StringGrid1.Cells[1,KolStr];
                          Cena:= StrToFloat(StringGrid1.Cells[2,KolStr]);
                          Kolich:=StrToFloat(StringGrid1.Cells[3,KolStr]);
                        end;
                    end;
              end;
          end;
```

Рисунок 67 – **Начальная часть процедуры обработки щелчка по кнопке *Следующая накладная***

```
      //Запись в очередной компонет файла всей информации Формы 2
      Write(FNak1,RNak1);
    end;
  //Чистка полей формы
  Edit1.Text:='';
  Edit2.Text:='';
  Edit3.Text:='';
  CheckBox1.Checked:=False;
  for i:=1 to StringGrid1.RowCount-1 do
    begin
      StringGrid1.Cells[0,i]:='';
      StringGrid1.Cells[1,i]:='';
      StringGrid1.Cells[2,i]:='';
      StringGrid1.Cells[3,i]:='';
    end;
end;
```

Рисунок 68 – **Заключительная часть процедуры обработки щелчка по кнопке *Следующая накладная***

При щелчке по кнопке *Button2* (*В главное меню*) необходимо выполнить те же действия, что и при щелчке по кнопке *Button*1 (*Следующая накладная*), и дополнительно закрыть окно формы *Form2* с помощью оператора *Close*. Таким образом, для получения текста процедуры *TForm2.Button2Click* надо в текст процедуры *TForm2.Button2Click* добавить оператор *Close.*

На рисунке 55 приведен начальный фрагмент исполняемой части модуля *Unit2* с инструкцией *uses Unit1;*, обеспечивающей использование в этом модуле глобальных переменных, описанных в модуле *Unit1*, и текст процедуры активации формы *Form2*, которая заполняет названия граф таблицы *StringGrid1*.

Окно формы *Form3* в режиме выполнения (окно приложения) приведено на рисунке 56.

Все вычисления и заполнение полей формы выполняются при активации формы. Текст процедуры приведен на рисунке 69. В исполняемой части модуля помещена инструкция *uses Unit1;* для использования глобальных переменных проекта.

```
procedure TForm3.FormActivate(Sender: TObject);
  var
    i,j:integer;
    Stoim,StoimPr,StoimRs:real;
begin
  StringGrid1.Cells[0,0]:='Номер накладной';
  StringGrid1.Cells[1,0]:='Дата выписки';
  StringGrid1.Cells[2,0]:='Приход/Расход';
  StringGrid1.Cells[3,0]:='Контрагент';
  StringGrid1.Cells[4,0]:='Число строк';
  StringGrid1.Cells[5,0]:='Стоимость по накладной';
  StoimPr:=0;
  StoimRs:=0;
  Seek(FNakl,0); //Установка указателя в начало файла
  KolNakl:=FileSize(FNakl); //Количество накладных
  for i:=1 to KolNakl do  //цикл чтения всех компонентов файла
    begin
      Read(Fnakl,RNakl);//Чтение из файла очередной ТТН
      with RNakl do
        begin
          StringGrid1.Cells[0,i]:=NomerNakl;
          StringGrid1.Cells[1,i]:=DataNakl;
          if PrichRasch
            then StringGrid1.Cells[2,i]:='Приход'
            else StringGrid1.Cells[2,i]:='Расход';
          StringGrid1.Cells[3,i]:=Kontragent;
          StringGrid1.Cells[4,i]:=IntToStr(KolStr);
          Stoim:=0;
          for j:=1 to KolStr do//цикл перебора всех строк текущей ТТН
            Stoim:= Stoim+MasStrNakl[j].Cena*MasStrNakl[j].Kolich;
          StringGrid1.Cells[5,i]:=FloatToStr(Stoim);
          if PrichRasch
            then StoimPr:=StoimPr+Stoim //накопление прихода
            else StoimRs:=StoimRs+Stoim; //накопление расхода
          //вывод результатов в поля формы 3
          Edit1.Text:=FloatToStr(StoimPr);
          Edit2.Text:=FloatToStr(StoimRs);
        end;
    end;
end;
```

Рисунок 69 – **Текст процедуры активации формы *Form3***

Текст процедуры обработки щелчка по кнопке *Button1* (*В главное меню*) состоит из одной команды – *Close*.

Окно формы *Form4* в режиме проектирования приведено на рисунке 58.

Исполняемая часть модуля содержит инструкцию *uses Unit1;* для использования глобальных переменных проекта и четыре процедуры обработки событий:

- Активация формы – вывод в поля формы реквизитов первой накладной из файла *FNakl*.
- Обработка щелчка по кнопке *Вперед* – вывод реквизитов следующей накладной (если в поля формы выведена последняя накладная из файла *FNakl*, то выдается сообщение об этом и содержимое полей формы не меняется).
- Обработка щелчка по кнопке *Назад* – вывод реквизитов предыдущей накладной (если в поля формы выведена первая накладная из файла *FNakl*, то выдается сообщение об этом и содержимое полей формы не меняется).
- Обработка щелчка по кнопке *Главное меню* – закрытие окна формы (оператор *Close*).

Для номера накладной (номер компонента файла), отображенной в окне формы, используется переменная *TecNakl*, которой в инициализирующей части модуля присвоено значение 0.

Текст процедуры активации формы *Form4* отличается от текста, приведенного на рисунке 59, только начальной частью. Соответствующий фрагмент процедуры приведен на рисунке 70.

```
procedure TForm4.FormActivate(Sender: TObject);
var
  j:integer;
begin
 Reset(FNakl);
 if FileSize(FNakl)=0
   then
     Label5.Caption:='Накладных нет'
   else
     begin
       //чтение очередного компонента файла
       Read(FNakl,RNakl);
       with RNakl do
```

Рисунок 70 – **Фрагмент процедуры активации формы *Form4***

Текст процедуры отображения следующей накладной отличается от текста, приведенного на рисунках 60 и 61, только начальной частью. Соответствующий фрагмент процедуры приведен на рисунке 71.

```
procedure TForm4.Button1Click(Sender: TObject);
var
  j:integer;
begin
 if (TecNakl+1)>=FileSize(FNakl)
   then
     Label5.Caption:='Последняя накладная'
   else
     begin
       Label5.Caption:='';
       TecNakl:=TecNakl+1;
       //установка указателя на следующий компонет файла
       Seek(FNakl,TecNakl);
       //чтение из файла очередного компонента
       Read(FNakl,RNakl);
       with RNakl do
```

Рисунок 71 – **Фрагмент процедуры обработки щелчка по команде *Вперед***

```
procedure TForm4.Button2Click(Sender: TObject);
var
  j:integer;
begin
 if TecNakl=0
   then
     Label5.Caption:='Первая накладная'
   else
     begin
       Label5.Caption:='';
       TecNakl:=TecNakl-1;
       //установка указателя на следующий компонет файла
       Seek(FNakl,TecNakl);
       //чтение из файла очередного компонента
       Read(FNakl,RNakl);
       with RNakl do
```

Рисунок 72 – **Фрагмент процедуры обработки щелчка по команде *Назад***

Текст процедуры отображения предыдущей накладной отличается от текста, приведенного на рисунке 71, только двумя операторами – текстом выводимого сообщения (*Первая накладная*) и изменением номера текущей ТТН, который уменьшается на 1. Соответствующий фрагмент рассматриваемой процедуры приведен на рисунке 72.

10. РАБОТА С ТЕКСТОВЫМИ ФАЙЛАМИ

10.1. Стандартные процедуры и функции обработки текстовых файлов

Текстовый файл – это файл, состоящий из компонентов, являющихся строками. Длина строки может изменяться от 0 до 255. Каждая строка завершается кодом символа *Enter*.

Для описания файловых переменных текстового типа используется стандартный идентификатор *TextFile*, который выглядит следующим образом:

```
Var
Fv:TextFile;
```

Для работы с текстовыми файлами можно использовать следующие процедуры и функции, которые использовались при работе с типизированными файлами, приведенными в пунктах 9.3 и 9.4:

- *AssignFile (fv, str)* – присвоение имени файлу.
- *Rewrite (fv)* – создание (открытие) нового файла. Если на диске уже имеется файл с таким именем, то он уничтожается. Указатель файла устанавливается в положение 0.
- *Reset (fv)* – открытие файла и установка указателя файла в начало файла.
- *CloseFile (fv)* – закрытие файла (нельзя выходить из программы без закрытия файла).
- *FileExists (str)* – возвращает значение *True*, если файл с именем *str* существует, и *False* – в противном случае.

Процедуры чтения и записи, специфические для текстовых файлов, имеют следующий вид:

- *Read (fv, Ch)* – считывание из файла очередного символа в литерную переменную *Ch*.
- *Write (fv, Ch)* – запись в файл литерной переменной *Ch*.
- *Readln (fv, str)* – считывание из файла очередной строки в строковую переменную *str* .
- *Writeln (fv, str)* – запись в файл в качестве очередной строки переменной *str*.
- *Append (fv)* – открытие файла и установка указателя на маркер конца файла.

Для текстовых файлов существуют и специфичные функции:

- *SeekEof (fv)* – возвращает значение *True*, если указатель файла находится на маркере конца файла и *False* – в любом другом случае.

• *Eoln (fv)* – возвращает значение *True*, если указатель файла достиг маркера конца строки и *False* – в любом другом случае.

• *SeekEoln (fv)* – аналогична предыдущей, но указатель файла переходит все пробелы и знаки табуляции, а также возвращает значение *True*, если указатель файла достиг маркера конца строки.

10.2. Пример обработки текстовых файлов

Создадим проект, который записывает в текстовый файл сведения о сотрудниках, в каждом компоненте которого записаны фамилия, год рождения и стаж, разделенные символом «*». При задании значения стажа проект должен выбрать из файла сведения о тех сотрудниках, стаж которых превышает заданное значение.

Окно формы приложения приведено на рисунке 73.

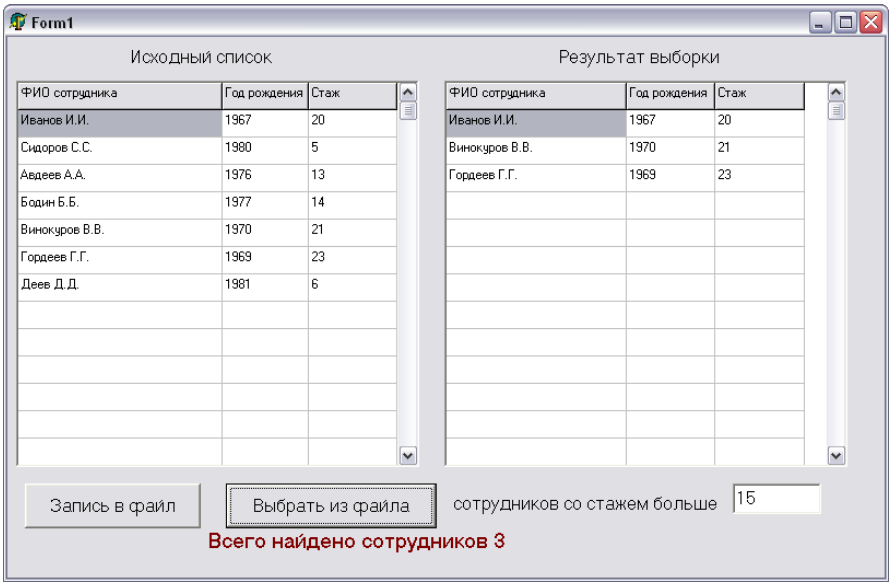


Рисунок 73 – **Окно приложения проекта обработки текстового файла**

Процедура активации формы заполняет заголовки граф таблиц и очищает остальные ячейки таблицы.

Процедура обработки щелчка по кнопке *Запись в файл* дописывает в файл *ФИО.txt* строки таблицы *StringGrid1*, заполненные пользователем. После записи в файл строки таблицы *StringGrid1* очищаются. Текст данной процедуры приведен на рисунке 74.

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    fv:TextFile;
    s1,s2,s3:string[255];
    W:Boolean;
    i,j:integer;
begin
   AssignFile(fv,'ФИО.txt');
   if FileExists('ФИО.txt')
     then //открытие существующего файла
          //с установкой указателя в конец файла
       Append(fv)
     else //создание нового файла
       Rewrite(fv);
   W:=true; i:=1;
   while W do
     begin
       s1:= StringGrid1.Cells[0,i];
       s2:= StringGrid1.Cells[1,i];
       s3:= StringGrid1.Cells[2,i];
       if s1<>''
         then //запись в файл информации о сотруднике
           //ФИО, год рождения и стаж разделены *
           writeln(fv,s1+'*'+s2+'*'+s3)
         else W:=false;
       i:=i+1
     end;
   CloseFile(fv); //закрытие файла
   //очистка ячеек таблиц
   for i:=1 to StringGrid1.RowCount-1 do
     for j:=0 to StringGrid1.ColCount-1 do
       StringGrid1.Cells[j,i]:='';
   for i:=1 to StringGrid2.RowCount-1 do
      for j:=0 to StringGrid2.ColCount-1 do
     StringGrid2.Cells[j,i]:='';
   Label2.Caption:=''
end;
```

Рисунок 74 – **Обработка щелчка по кнопке *Запись в файл***

Просмотреть содержимое текстового файла можно в любом текстовом редакторе. На рисунке 75 файл *ФИО.txt* открыт в Word.

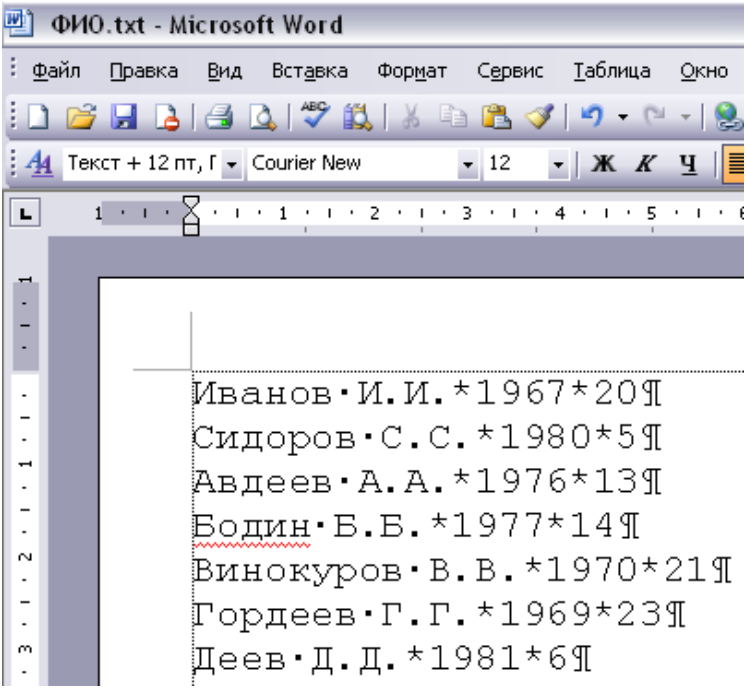


Рисунок 75 – **Содержимое файла *ФИО.txt***

Процедура обработки щелчка по кнопке *Выбрать из файла* выполняет следующие функции: открывает файл *ФИО.txt*, отображает его содержимое в таблицу *StringGrid1*, закрывает файл *ФИО.txt* и переписывает в таблицу *StringGrid2* из таблицы *StringGrid1* сведения о тех сотрудниках, стаж работы которых превышает заданное значение. Текст данной процедуры приведен на рисунках 76 и 77.

```
procedure TForm1.Button2Click(Sender: TObject);
  var
    fv:TextFile;
    str,s1,s2,s3:string[255];
    staz,staz1:real;
    W:Boolean;
    i,k,m,n:integer;
begin
   AssignFile(fv,'ФИО.txt');
   if not(FileExists('ФИО.txt'))
     then
       Label2.Caption:='Файл не существует'
     else
       begin
         Reset(fv); //открытие файла
         staz:=StrToFloat(Edit1.Text);
         //вывод содержимого файла в StringGrid1
         i:=1;
         while not(SeekEof(fv)) do
           begin
             //считывание очередной записи файла
             readln(fv,str);
             //выделение ФИО
             k:=Pos('*',str); s1:=Copy(str,1,k-1);
             //выделение года рождения
             Delete(str,1,k); m:=Pos('*',str);
             s2:=Copy(str,1,m-1);
             //выделение стажа
             Delete(str,1,m);
             //вывод реквизитов записи в таблицу
             StringGrid1.Cells[0,i]:=s1;
             StringGrid1.Cells[1,i]:=s2;
             StringGrid1.Cells[2,i]:=str;
             i:=i+1
           end;
```

Рисунок 76 – **Начальная часть процедуры обработки щелчка по кнопке *Выбрать из файла***

```
        CloseFile(fv); //закрытие файла
        n:=i-1; //количество записей файла
        //цикл перебора значений стажа в таблице для
        //выбора значений больше заданного и переписи
        //соответствующих строк в таблицу StringGrid2
        k:=0; // количество найденных значений
        for i:=1 to n do
          begin
           staz1:= StrToFloat(StringGrid1.Cells[2,i]);
           if staz1>staz
             then
               begin
                 k:=k+1;
                 StringGrid2.Cells[0,k]:=StringGrid1.Cells[0,i];
                 StringGrid2.Cells[1,k]:=StringGrid1.Cells[1,i];
                 StringGrid2.Cells[2,k]:=StringGrid1.Cells[2,i];
               end;
          end;
        Label2.Caption:='Всего найдено сотрудников '+IntToStr(k);
      end;
end;
```

Рисунок 77 – **Заключительная часть процедуры обработки щелчка по кнопке *Выбрать из файла***

11. ПОДПРОГРАММЫ ПОЛЬЗОВАТЕЛЯ

11.1. Общие сведения о подпрограммах

Подпрограмма пользователя – это часть проекта (часть процедуры обработки события), которая записывается один раз, а использоваться может неоднократно путем обращения к подпрограмме. В тексте подпрограммы используются так называемые формальные параметры, т. е. специальные переменные, которые при обращении к подпрограмме заменяются фактическими параметрами – переменными и значениями, определенными в самой программе (вне подпрограммы).

Формальные параметры можно разделить на два класса – входные и выходные. *Входные формальные параметры* – это параметры, которые при обращении к подпрограмме заменяются конкретными значениями или переменными, имеющими значения. При работе подпрограммы происходит обработка входных параметров и в результате получаются значения, которые присваиваются *выходным параметрам* (переменным).

Некоторые формальные параметры могут одновременно быть и входными, и выходными. Подпрограмма может иметь только один класс параметров – или только входные, или только выходные. Также допустима подпрограмма, не имеющая никаких параметров.

Входные формальные параметры еще называют *параметрами, заменяемыми по значению* (*параметры-значения*), а выходные формальные параметры называют *параметрами, заменяемыми по ссылке* (*параметры-ссылки*).

Подпрограммы пользователя бывают двух видов – *процедуры пользователя* и *функции пользователя*.

Функция – это частный случай подпрограммы, когда в результате ее выполнения получается одно единственное значение. В процедуре нет ограничений на количество выходных параметров.

Ранее уже описывались встроенные (стандартные) процедуры, например, *Str(v:n:m, s)*. При обращении к этой процедуре, например, с помощью оператора *Str(a:7:2, w)* формальные параметры *v, n, m* и *s* заменяются соответственно фактическими *a*, 7, 2, *w*. При этом параметры *v, n, m* заменяются по значению, а параметр *s* – по ссылке.

Также ранее использовать встроенные функции, например, функция *sin(x)*, где *x* – входной формальный параметр, заменяемый по значению. Результату выполнения функции не поставлен в соответствие какой-либо формальный параметр. Для использования результата необходимо, например написать какой-нибудь оператор присваивания: *y:=sin(x)*.

При использовании встроенных процедур и функций программист не знает текст самой подпрограммы, а при использовании процедур и функций пользователя необходимо самому писать текст подпрограммы.

Назначение подпрограмм пользователя следующее:

- сокращение объема всего проекта (подпрограмма записывается один раз, а используется многократно);
- облегчение отладки программы, так как возможна отладка проекта отдельными блоками;
- упрощение внесения изменений в проект, так как изменения внутри подпрограммы могут не вызывать корректировки других подпрограмм;
- организация работы нескольких программистов над одним проектом.

Подпрограммы пользователя могут располагаться внутри процедуры-обработчика события и в разделе описаний модуля.

11.2. Процедуры пользователя

Процедура пользователя – это именованная группа операторов, реализующая определенную часть общей задачи и вызываемая при необходимости для выполнения по имени процедуры из раздела операторов той части проекта, в которой находится описание процедуры.

Общий вид *описания процедуры* следующий:

Procedure *имя процедуры (список формальных параметров с указанием типа);*
раздел описаний локальных параметров процедуры;
begin
раздел операторов процедуры
end;

Первая строка описания называется *заголовком процедуры*, а последующая группа строк – *телом процедуры*.

Обращение к процедуре выполняется с помощью оператора вызова процедуры, который записывается в следующем виде:

имя процедуры (список фактических параметров);

В списке формальных параметров элементы списка разделяются «;». Тип формального параметра должен быть указан с помощью ранее определенного типа. В списке фактических параметров элементы списка разделяются запятыми. Между фактическими параметрами в операторе вызова процедуры и формальными параметрами в заголовке описания процедуры устанавливается взаимнооднозначное соответствие слева направо. Тип фактического параметра должен совпадать с типом соответствующего формального параметра. Количество фактических параметров должно равняться числу формальных.

В списке формальных параметров перед выходными параметрами (параметрами, передаваемыми по ссылке) необходимо указывать слово *var*.

Характерная черта параметров, заменяемых по значению, заключается в том, что при изменении в процедуре такого формального параметра значение соответствующего фактического параметра не изменяется. В отличие от этого при изменении в процедуре параметра-ссылки изменяется соответствующий фактический параметр.

Пример. Процедура, определяющая минимальный элемент *X* вещественного массива *a*, содержащего *n* элементов, и номер *k* самого первого из них, имеет следующий вид:

```
Type
    Mas=array[1..20] of real;
Var
    c:Mas; m:integer; Y:real;
Procedure MinNomMas (a:Mas; n:integer; var X:real; var
                                    k:integer);
  Var
     i:integer; b:real;
  begin
     b:=a[1];
     k:=1;
     for i:=2 to n do
        if a[i]<b
          then
            begin
               b:=a[i]; k:=i
            end;
        X:=b
  end;
...
    MinNomMas (c, 10, Y, m);  // обращение к процедуре
```

11.3. Функции пользователя

Функция пользователя – это частный случай процедуры пользователя, когда в результате ее выполнения получается одно единственное значение.

Общий вид описания функции следующий:

```
Function имя функции (список формальных параметров с указанием типа):тип функции;
  раздел описаний локальных параметров процедуры;
  begin
     раздел операторов функции
  end;
```

Описание функции отличается от описания процедуры следующими признаками:

- после списка формальных параметров указывается тип полученного результата – это один из скалярных типов, т. е. *real*, *integer*, *char* или *string*;
- в списке формальных параметров нет параметров-ссылок;

• в разделе операторов функции должен быть, по крайней мере, один оператор присваивания, в левой части которого указывается имя функции. При этом имя функции не может использоваться в правой части оператора присваивания.

Обращение к функции выполняется с помощью указателя функции, т. е.

имя функции (список фактических параметров);

Указатель функции не является оператором и должен находиться в каком-либо выражении.

Правило соответствия фактических и формальных параметров то же самое, что и в процедуре пользователя.

Пример. Функция, определяющая минимальный элемент *X* вещественного массива *a*, содержащего *n* элементов, имеет следующий вид:

```
Type
   Mas=array[1..20] of real;
Var
   c:Mas; Y:real;
Function MinMas (a:Mas; n:integer):real;
   Var
      i:integer; b:real;
   begin
      b:=a[1];
      for i:=2 to n do
         if a[i]<b
           then
              b:=a[i];
      MinMas:=b
   end;
...
   Y:=MinNomMas (c, 10);  // обращение к функции
```

11.4. Методика использования процедур и функций

При решении задачи необходимо выполнить следующие действия:

• разделить задачу на отдельные подзадачи и определить, какие из подзадач повторяются, т. е. реализуются одними и теми же операциями только над различными наборами данных;

• для каждой повторяющейся подзадачи надо определить вид подпрограммы (процедура или функция), с помощью которой она может быть реализована;

• для каждой процедуры следует определить множество входных и выходных параметров и их типы;

• для каждой функции следует определить множество входных параметров и их типы, а также тип результата.

Рассмотрим приведенную методику на примере решения следующей задачи.

В каждом их трех заданных массивов найти минимальный элемент, а затем удвоить значение каждого элемента.

Окно приложения представлено на рисунке 78.

Рисунок 78 – **Окно приложения проекта, использующего подпрограммы пользователя**

Данная задача может быть разделена на следующие подзадачи:

• ввод исходных данных (трех массивов);

• нахождение минимального элемента в каждом из трех массивов;

• удвоение элементов в каждом из трех массивов;

• вывод результатов (трех минимальных значений и трех массивов).

Для нахождения минимального элемента в некотором формальном массиве опишем функцию *MinMas*, в качестве входных формальных параметров которой укажем массив *x* и количество элементов в этом массиве *k*.

Удвоение элементов массива будем выполнять с помощью процедуры *Dva*, у которой будут те же формальные параметры, что и у функции *MinMas*, однако параметр *x* будет в данном случае и входным, и выходным.

Текст описаний процедуры обработки щелчка по кнопке *Обработка* приведен на рисунке 79, а текст тела этой процедуры – на рисунке 80.

```
procedure TForm1.Button1Click(Sender: TObject);
  type mas=array[1..10]of integer;
  var
    a,b,c:mas;
    i,na,nb,nc,mina,minb,minc:integer;
    procedure Dva(Var x:mas; k:integer);
    //Процедура удвоения элементов массива
      var
        i:integer;
      begin
        for i:=1 to k do
          x[i]:=x[i]*2;
      end;
    function MinMas(x:mas; k:integer):integer;
    //Функция определения минимального элемента массива
      var
        i,minx:integer;
      begin
        minx:=x[1];
        for i:=2 to k do
          if x[i]<minx
            then minx:=x[i];
        MinMas:=minx
      end;
```

Рисунок 79 – **Раздел описаний процедуры обработки щелчка по кнопке *Обработка***

```
begin
  //ввод исходных данных
  na:=StrToInt(Edit1.Text);
  for i:=1 to na do
    a[i]:=StrToInt(StringGrid1.Cells[i-1,0]);
  nb:=StrToInt(Edit2.Text);
  for i:=1 to nb do
    b[i]:=StrToInt(StringGrid2.Cells[i-1,0]);
  nc:=StrToInt(Edit3.Text);
  for i:=1 to nc do
    c[i]:=StrToInt(StringGrid3.Cells[i-1,0]);
  //нахождение минимальных значений
  mina:=MinMas(a,na);
  minb:=MinMas(b,nb);
  minc:=MinMas(c,nc);
  //вывод минимальных значений
  Edit4.Text:=IntToStr(mina);
  Edit5.Text:=IntToStr(minb);
  Edit6.Text:=IntToStr(minc);
  //удвоение значений элементов
  Dva(a,na);
  Dva(b,nb);
  Dva(c,nc);
  //вывод полученных массивов
  for i:=1 to na do
    StringGrid4.Cells[i-1,0]:=IntToStr(a[i]);
  for i:=1 to nb do
    StringGrid5.Cells[i-1,0]:=IntToStr(b[i]);
  for i:=1 to nc do
    StringGrid6.Cells[i-1,0]:=IntToStr(c[i]);
end;
```

Рисунок 80 – **Тело процедуры обработки щелчка по кнопке *Обработка***

11.5. Использование в подпрограммах визуальных компонентов

В задаче, рассмотренной в предыдущем пункте, ввод количества и значений трех массивов организован последовательностью однотипных операторов, отличающихся не только именами массива и переменной, обозначающей число элементов, но и именами визуальных компонентов.

Для включения в список фактических параметров имен визуальных компонентов необходимо объявить в качестве типа соответствующего формального параметра стандартный тип визуального компонента (класс компонента).

Запишем новый вариант решения задачи из пункта 11.4 исходя из использования двух новых процедур пользователя. Одна процедура выполняет ввод размера элементов массива из поля *Edit* и элементов массива из текстовой таблицы. Вторая процедура организует вывод новых значений элементов массива в текстовую таблицу. Тексты указанных процедур представлены на рисунке 81.

```
procedure InputMas(Var x:mas; Var k:integer;
                   Ed:TEdit; SG: TStringGrid);
//Процедура ввода количества и значений элементов массива
  var
    i:integer;
  begin
    k:=StrToInt(Ed.Text);
    for i:=1 to k do
      x[i]:=StrToInt(SG.Cells[i-1,0]);
  end;
procedure OutputMas(x:mas; k:integer; Var SG: TStringGrid);
//Процедура вывода значений элементов массива
  var
    i:integer;
  begin
    for i:=1 to k do
      SG.Cells[i-1,0]:=IntToStr(x[i]);
  end;
```

Рисунок 81 – **Процедуры организации ввода и вывода массива**

Текст тела процедуры обработки щелчка по кнопке *Обработка* с использованием процедур ввода и вывода элементов массива приведен на рисунке 82.

```
begin
  //ввод исходных данных
  InputMas(a,na, Edit1,StringGrid1);
  InputMas(b,nb, Edit2,StringGrid2);
  InputMas(c,nc, Edit3,StringGrid3);
  //определение минимальных элементов
  mina:=MinMas(a,na);
  minb:=MinMas(b,nb);
  minc:=MinMas(c,nc);
  //вывод минимальных элементов
  Edit4.Text:=IntToStr(mina);
  Edit5.Text:=IntToStr(minb);
  Edit6.Text:=IntToStr(minc);
  //удвоение значений элементов
  Dva(a,na);
  Dva(b,nb);
  Dva(c,nc);
  //вывод полученных массивов
  OutputMas(a,na,StringGrid4);
  OutputMas(b,nb,StringGrid5);
  OutputMas(c,nc,StringGrid6);
end;
```

Рисунок 82 – **Тело процедуры обработки щелчка по кнопке *Обработка* с использованием четырех подпрограмм пользователя**

ЗАДАНИЯ ЛАБОРАТОРНЫХ РАБОТ

Лабораторная работа 1
Программное управление объектами в окне формы

Цель работы: получение навыков реализации проектов, использующих методы визуализации и скрытия объектов, размещенных в окне формы.

Разработайте проект табулирования одной из четырех заданных функций с выводом результатов в текстовую таблицу. Исходные формулы выберите из таблицы 12. В окно формы проекта включите четыре объекта *Image* для отображения исходных формул. Организуйте выбор номера функции для табулирования с помощью объекта *RadioGroup* (группа переключателей).

Таблица 12 – **Виды функций**

Номер варианта	Вид функции	Номер варианта	Вид функции
1	$y = \frac{arctg\ bx}{1+\sin^2 x}$	16	$y = \frac{arctg\ (a+x)}{\sqrt{a^3+b^3}}$
2	$y = \frac{\sin^2 x + a}{\sqrt{x}+bx}$	17	$y = \frac{1+\sqrt{bx}}{0{,}5+\sin^2 ax}$
3	$y = \sqrt{\frac{a+bx}{\ln^2 x}}$	18	$y = \frac{a-e^{bx}}{\ln\lvert 2x\rvert}$
4	$y = \frac{\ln^2(x-b)}{a\sqrt{x}}$	19	$y = \frac{(a+bx)^2}{1+\cos^3 ax}$
5	$y = \frac{a\ln^2 x}{b+\sqrt{x}}$	20	$y = \frac{b+\sin^2 ax}{e^{-x/2}}$
6	$y = \frac{e^{ax}+b}{1+\cos^2 x}$	21	$y = \frac{\sin^2 x - a}{bx}$
7	$y = \frac{a+\sqrt[3]{x}}{\sin^2 bx}$	22	$y = \frac{arctg^2 ax}{b+0{,}5x}$
8	$y = \frac{a\sqrt{\lvert x\rvert}-bx}{\ln^3 x}$	23	$y = \frac{\ln\ (a^2-x)}{b\ \sin^2 x}$
9	$y = \frac{\sqrt{ax}-b}{tg^2 x}$	24	$y = \frac{a-\sqrt{bx}}{1+\cos\lvert 2x\rvert}$
10	$y = e^{-x}\frac{a+bx}{\ln^2\lvert x+1\rvert}$	25	$y = \frac{\ln^2(a+x)}{(b+x)^2}$
11	$y = \frac{tg^2 x - b}{e^{ax}}$	26	$y = \frac{\sqrt{\lvert a\ \ln\ x\rvert}}{1+tg^2 bx}$
12	$y = \frac{arctg\ bx}{1+\sqrt[3]{ax}}$	27	$y = \frac{1+tg^2 x}{b+e^{x/a}}$
13	$y = \frac{\sin^3 ax}{ax+b}$	28	$y = \frac{\cos^2 2x + b}{\sqrt{1+e^{ax}}}$
14	$y = \frac{e^{-ab}}{b+\cos^3 ax}$	29	$y = \frac{\sqrt{ax+b}}{\ln^2\lvert x\rvert}$
15	$y = \frac{\ln^2\lvert x\rvert + b}{a\sqrt{x}}$	30	$y = \frac{1+\sin^2 ax}{b^2+x^2}$

При размещении переключателей в области группы используйте свойство *Items*. В этот список занесите номера формул.

При разработке процедуры табулирования выбранной функции обеспечьте скрытие формул трех других функций. Кроме основной процедуры, выполняющей требуемые расчеты, создайте процедуру восстановления исходного состояния формы при щелчке в любой области исходных данных.

Включите в проект вторую форму, содержащую сведения об авторе. Произведите отладку проекта с помощью табличного процессора MS Excel.

Лабораторная работа 2
Определение параметров регрессионной зависимости

***Цель работы*:** получение студентами навыков по созданию многомодульного проекта в СП Delphi для определения параметров регрессионной зависимости и его отладки на базе табличного процессора MS Excel.

В процессе выполнения работы студент решает следующие задачи:

- разработка алгоритма вычисления статистических характеристик по заданным формулам;
- запись алгоритма в виде блок-схемы;
- подготовка данных для отладки средствами табличного процессора MS Excel;
- разработка интерфейса приложения в системе программирования Delphi;
- отладка проекта в системе программирования Delphi.

Исходные данные

1. Функции $y_1 = a_{01} + a_{11} \cdot f_1(x)$ и $y_2 = a_{02} + a_{12} \cdot f_2(x)$, используемые как регрессионные зависимости.

2. Набор значений фактора $x_1,x_2,\ldots,x_n$ и результата $y_1,y_2,\ldots,y_n$, где n – количество значений.

3. Формулы, с помощью которых можно рассчитать такие значения параметров a_{01} и a_{11}, что зависимость $y_1 = a_{01} + a_{11} \cdot f_2(x)$ наилучшим образом приближает исходные наборы фактора $x_1,x_2,\ldots,x_n$ и результата $y_1,y_2,\ldots,y_n$, а также формулы для расчета параметров a_{02} и a_{12}.

На основании указанных исходных данных требуется создать проект в СП Delphi, который поможет определить следующее:

1. Существует ли связь между заданными значениями фактора и результата.

2. Если связь существует, то какая из двух заданных регрессионных зависимостей лучше ее описывает.

Наличие связи определяется путем расчета коэффициента детерминации (это значение находится в пределах от 0 до 1). Если его значение получается большим, чем 0,5, то связь между фактором и результатом существует.

Наилучшее приближение к исходным данным обеспечивает та регрессионная зависимость, у которой больше коэффициент Фишера.

Варианты заданий

Варианты зависимостей приведены в таблице 13.

Все регрессионные зависимости имеют вид $y = a_0 + a_1 \cdot f(x)$.

Таблица 13 – **Виды регрессионных зависимостей**

Номер варианта	Зависимость	Номер варианта	Зависимость
1	$y = a_0 + a_1 \cdot x$	16	$y = a_0 + a_1 \cdot e^{1/x} / x^2$
2	$y = a_0 + a_1 \cdot x^2$	17	$y = a_0 + a_1 \cdot \ln^2(x)$
3	$y = a_0 + a_1 \cdot (1/x)$	18	$y = a_0 + a_1 \cdot \sqrt{x} \cdot \ln(x)$
4	$y = a_0 + a_1 \cdot \sqrt{x}$	19	$y = a_0 + a_1 \cdot \sqrt{x} / \ln(x)$
5	$y = a_0 + a_1 \cdot \ln(x)$	20	$y = a_0 + a_1 \cdot \sqrt{x} \cdot e^{1/x}$
6	$y = a_0 + a_1 \cdot e^{1/x}$	21	$y = a_0 + a_1 \cdot \sqrt{x} / e^{1/x}$
7	$y = a_0 + a_1 \cdot x / \ln(x)$	22	$y = a_0 + a_1 \cdot x \cdot \ln^2(x)$
8	$y = a_0 + a_1 \cdot x / e^{1/x}$	23	$y = a_0 + a_1 \cdot x^2 \cdot \sqrt{x}$
9	$y = a_0 + a_1 \cdot x^2 / \ln(x)$	24	$y = a_0 + a_1 \cdot \sqrt{x} \cdot \ln^2(x)$
10	$y = a_0 + a_1 \cdot x^2 / e^{1/x}$	25	$y = a_0 + a_1 \cdot e^{1/x} \cdot \ln(x)$
11	$y = a_0 + a_1 \cdot x^2 \cdot \ln(x)$	26	$y = a_0 + a_1 \cdot \ln(\sqrt{x})$
12	$y = a_0 + a_1 \cdot x^2 \cdot e^{1/x}$	27	$y = a_0 + a_1 \cdot x \cdot \ln(\sqrt{x})$
13	$y = a_0 + a_1 \cdot \ln(x) / x$	28	$y = a_0 + a_1 \cdot \sqrt{x} \cdot \ln(\sqrt{x})$
14	$y = a_0 + a_1 \cdot e^{1/x} / x$	29	$y = a_0 + a_1 \cdot x \cdot e^{1/\sqrt{x}}$
15	$y = a_0 + a_1 \cdot \ln(x) / x^2$	30	$y = a_0 + a_1 \cdot x^3 \cdot \ln(x)$

Варианты наборов исходных данных приведены в файле *Факторы и результаты.xls*. Номер набора соответствует номеру листа. Первый числовой столбец на листе – значения фактора, второй – значения результата.

Подготовка данных для отладки проекта

Первым этапом работы является подготовка данных для отладки проекта. Подготовка данных выполняется с помощью MS Excel и надстройки *Пакет анализа*.

Скопируйте заданный набор данных на лист рабочей книги Excel и вычислите с помощью функции *КОРРЕЛ* коэффициент парной корреляции (коэффициент детерминации), характеризующий связь между заданными значениями x_i и y_i, где $i = \overline{1, n}$. Пример соответствующего фрагмента листа приведен на рисунке 83.

B19 =КОРРЕЛ(B2:B17;C2:C17)

	A	B	C
1	Райпо	Транспортные расходы по общепиту, млн. р.	Валовые доходы по общепиту, млн. р.
2	Барановичское райпо	25	329
3	Березовское райпо	73	896
4	Брестское райпо	32	291
16	Пружанское райпо	32	677
17	Столинское райпо	212	2517
18			
19	Коэффициент детерминации	0,97601781	

Рисунок 83 – **Пример исходного набора данных с вычислением коэффициента детерминации**

На рисунке видно, что в данном примере коэффициент детерминации примерно равен 0,98, т. е. связь между фактором и результатом очень хорошая.

Образуйте дополнительный столбец и заполните его формулами, вычисляющими функцию $f_1(x_i)$. На рисунке 84 приведен пример выполнения задания для функции $y1 = a_{01} + a_{11} \cdot \frac{1}{x} \cdot e^{\frac{1}{x}}$.

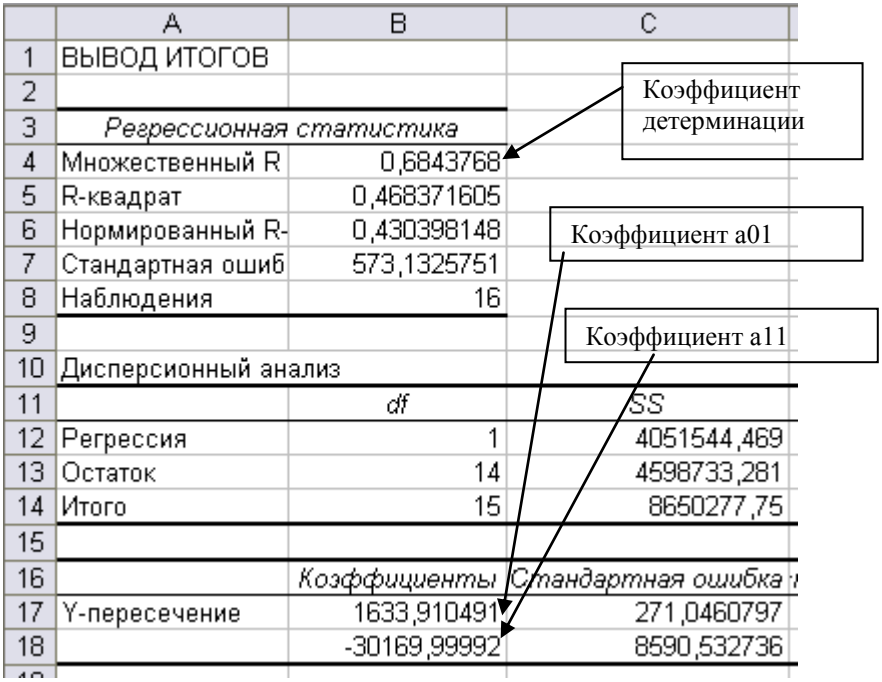
	A	D	E
1	Райпо	$e^{1/x} / x$	Значения первой регрессии (y1)
2	Кобринское райпо	0,062387533	-248,3213758
3	Малоритское райпо	0,055475855	-39,79604612
4	Ганцевичское райпо	0,045410317	263,8812335
16	Столинское райпо	0,004739284	1490,926305
17	Лунинецкое райпо	0,003571406	1526,161175
18			
19		Первая регрессионная зависимость	
20		a01	1633,910491
21		a11	-30169,99992

Рисунок 84 – **Вычисление параметров первой регрессии**

Вызовите надстройку *Пакет анализа* и в диалоговом окне *Анализ данных* выберите элемент *Регрессия*.

В диалоговом окне *Регрессия* укажите в качестве входного интервала *Y* диапазон ячеек, содержащих значения y_i, а в качестве входного интервала *X* – диапазон ячеек, содержащих значения $f_1(x_i)$, так как функциональность *Регрессия* MS Excel рассчитывает параметры только линейной регрессионной зависимости.

На листе результатов проанализируйте полученное значение коэффициента детерминации (*Множественный R*), показывающего качество приближения исходных данных найденной регрессионной зависимостью (рисунок 85).



	A	B	C
1	ВЫВОД ИТОГОВ		
2			
3	*Регрессионная статистика*		
4	Множественный R	0,6843768	
5	R-квадрат	0,468371605	
6	Нормированный R-	0,430398148	
7	Стандартная ошиб	573,1325751	
8	Наблюдения	16	
9			
10	Дисперсионный анализ		
11		*df*	*SS*
12	Регрессия	1	4051544,469
13	Остаток	14	4598733,281
14	Итого	15	8650277,75
15			
16		*Коэффициенты*	*Стандартная ошибка*
17	Y-пересечение	1633,910491	271,0460797
18		-30169,99992	8590,532736

Рисунок 85 – **Пример листа результатов вычисления регрессии**

Скопируйте на лист исходных данных с листа результатов полученные коэффициенты регрессии a_{01} и a_{11} и заполните еще один столбец значениями $y1 = a_{01} + a_{11} \cdot f_1(x)$, как это сделано на рисунке 84.

Добавьте к таблице столбец с формулами, вычисляющими функцию $f_2(x_i)$, и аналогично предыдущим действиям с помощью функциональности *Регрессия* надстройки *Пакет анализа* определите коэффициент детерминации и коэффициенты регрессии a_{02} и a_{12}, заполните новый столбец значениями $y2 = a_{02} + a_{12} \cdot f_2(x)$.

Выполните сортировку таблицы по возрастанию значений фактора и постройте на отдельном листе графики трех функций y, $y1$ и $y2$ с отметкой их значений в точках $x_i, i = \overline{1,n}$ (пример графика приведен на рисунке 86).

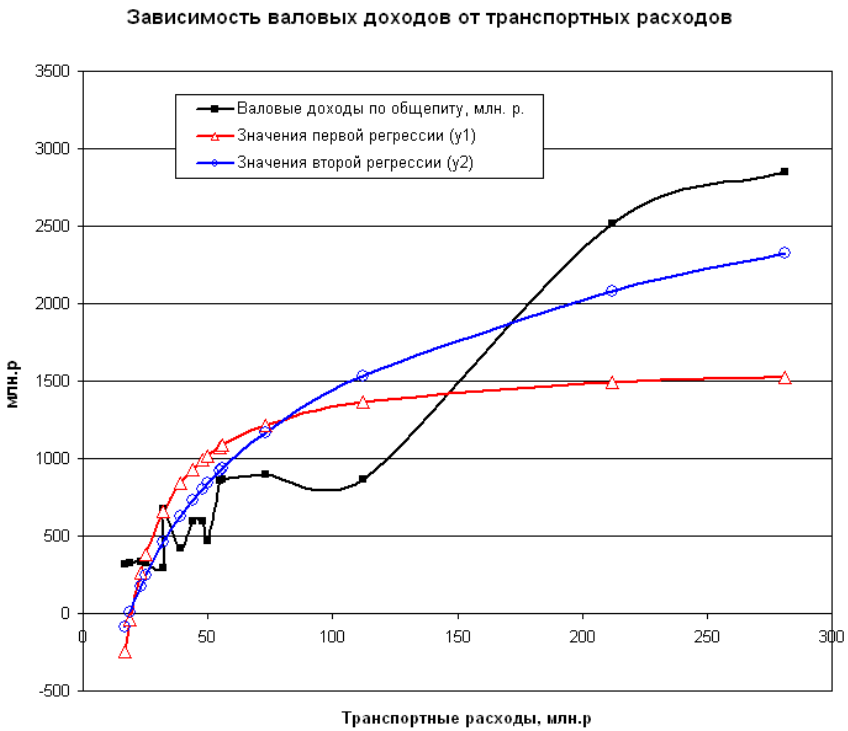


Рисунок 86 – **Пример графика исходной и регрессионных зависимостей**

Для количественной оценки качества приближения исходных данных регрессионными зависимостями вычислим для них значения коэффициента Фишера. С этой целью добавим к нашей таблице три столбца: в первом из них вычислим значения $(y_i - \bar{y})^2$, где $\bar{y}$ – среднее значение результата, во втором столбце вычислим значения $(y_i - y1_i)^2$ и в третьем – значения $(y_i - y2_i)^2$. Пример приведен на рисунке 87.

F	G	H	I	J
$\ln(\sqrt{x})$	Значения второй регрессии (y2)	$(y-\bar{y})^2$	$(y-y1)^2$	$(y-y2)^2$
1,41660667	-86,46785265	254898,7656	324127	166030,0509
1,47221949	9,040911219	246884,7656	136011	102373,8185

Рисунок 87 – **Пример вычисления второй регрессии и вспомогательных вычислений для получения коэффициентов Фишера**

Поместите под таблицей формулы, вычисляющие значения коэффициентов Фишера, как это показано на рисунке 88.

	A	B
19	Среднее по x	=СРЗНАЧ(B2:B17)
20	Среднее по y	=СРЗНАЧ(C2:C17)
21	Дисперсия по y	=СУММ(H2:H17)/15
22	Остаточная дисперсия первой регрессии	=СУММ(I2:I17)/15
23	Остаточная дисперсия второй регрессии	=СУММ(J2:J17)/15
24	Коэфициент Фишера для первой регрессии	=B21/B22
25	Коэфициент Фишера для второй регрессии	=B21/B23

Рисунок 88 – **Формулы для вычислений коэффициентов Фишера**

Подготовка данных для отладки проекта закончена.

Порядок разработки проекта в СП Delphi

Разработайте интерфейс многомодульного проекта, обеспечивающий ввод исходных данных, вывод результатов и графика, построенного в Excel, а также получение сведений об авторе.

Для переключения между модулями используйте иерархическое меню или форму с вкладками.

Разработайте алгоритм и представьте его в виде блок-схемы, который для заданных наборов фактора x_1, x_2,…, x_n и результата y_1, y_2, … y_n рассчитывает следующие статистические характеристики:

- средние значения этих наборов:

$$\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i \ , \ \bar{y} = \frac{1}{n} \cdot \sum_{i=1}^{n} y_i \ ;$$

- дисперсии:

$$\sigma_x^2 = \frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{x})^2 \ , \ \sigma_y^2 = \frac{1}{n-1} \cdot \sum_{i=1}^{n} (y_i - \bar{y})^2 \ ;$$

- среднеквадратические отклонения:

$$\sigma_x = \sqrt{\sigma_x^2} \ , \ \sigma_y = \sqrt{\sigma_y^2} \ ;$$

- коэффициента детерминации (парной корреляции):

$$r = \frac{\sum_{i=1}^{n} (x_i - \bar{x}) \cdot (y_i - \bar{y})}{(n-1) \cdot \sigma_x \cdot \sigma_y} \ .$$

В алгоритме должен быть предусмотрен анализ вычисленного значения коэффициента детерминации *r:* если $|r| < 0.5$, то считают, что между фактором и результатом отсутствует связь, а при $|r| \geq 0{,}5$ – считают, что зависимость результата от фактора существует.

Далее в алгоритме следует предусмотреть вычисление характеристик регрессионных зависимостей $y1 = a_{01} + a_{11} \cdot f_1(x)$ и $y2 = a_{02} + a_{12} \cdot f_2(x)$. Для этого надо вычислить значения параметров a_{01}, a_{11}, a_{02} и a_{12} по следующим формулам:

$$d = n \cdot \sum_{i=1}^{n} f^2(x_i) - (\sum_{i=1}^{n} f(x_i))^2 \ ;$$

$$a_0 = (\sum_{i=1}^{n} y_i \cdot \sum_{i=1}^{n} f^2(x_i) - \sum_{i=1}^{n} f(x_i) \cdot \sum_{i=1}^{n} (y_i \cdot f(x_i))) / d \ ;$$

$$a_1 = (n \cdot \sum_{i=1}^{n} (y_i \cdot f(x_i)) - \sum_{i=1}^{n} f(x_i) \cdot \sum_{i=1}^{n} y_i) / d \ .$$

Указанные формулы выведены из условия минимальности суммы квадратов отклонений значений $y1_i$ и $y2_i$ от y_i в точках x_i, $i = \overline{1,n}$).

Полученные значения параметров a_{01}, a_{11}, a_{02} и a_{12} позволят вычислить значения по регрессионным зависимостям $y1_i = a_{01} + a_{11} \cdot f_1(x_i)$ и $y2_i = a_{02} + a_{12} \cdot f_2(x_i)$.

Далее следует вычислить остаточные дисперсии по формулам

$$\sigma 1_y^2 = \frac{1}{n-1} \cdot \sum_{i=1}^{n} (y_i - y1_i)^2 \ ; \ \sigma 2_y^2 = \frac{1}{n-1} \cdot \sum_{i=1}^{n} (y_i - y2_i)^2 \ ;$$

Коэффициенты Фишера, характеризующие качество приближений функций $y1$ и $y2$ к заданному результату y, вычисляются по формулам

$$F1 = \frac{\sigma_y^2}{\sigma 1_y^2} \ ; \ F2 = \frac{\sigma_y^2}{\sigma 2_y^2} \ .$$

В заключение алгоритм должен сравнить полученные значения $F1$ и $F2$. Следует отметить, что лучшей регрессионной зависимостью является та, для которой коэффициент Фишера имеет большее значение.

Графическая схема алгоритма решения задачи приведена на рисунках 89 и 90. Данная схема является структурной, т. е. представляет общую структуру задачи без детализации отдельных блоков. Для решения задачи необходимо детализировать блоки A–B, B–C и т. д.

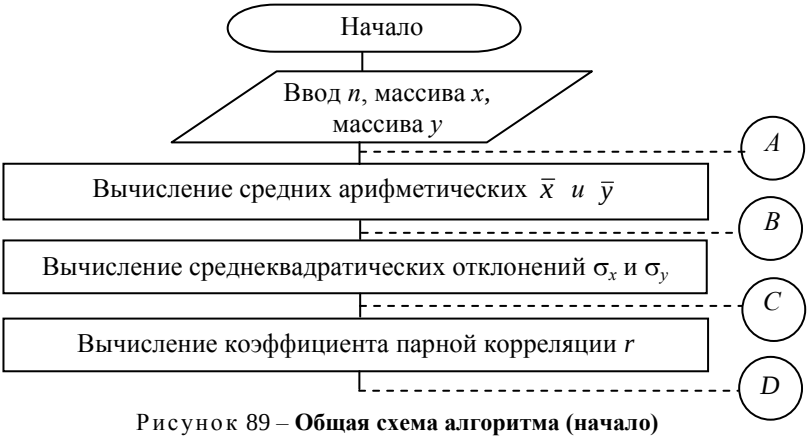


Рисунок 89 – **Общая схема алгоритма (начало)**

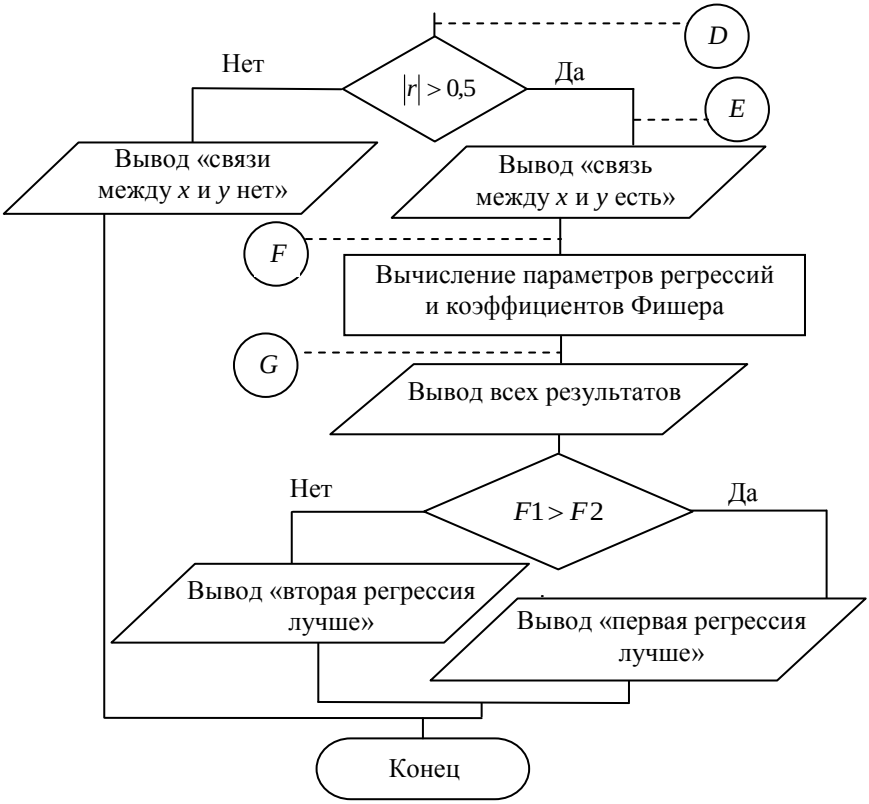


Рисунок 90 – **Общая схема алгоритма (окончание)**

Составьте тексты процедур обработки событий для объектов форм в соответствии с разработанным алгоритмом.

Включите график, построенный в Excel, в качестве графического объекта в соответствующее окно формы проекта.

Заполните окно «Сведения об авторе».

Произведите отладку проекта.

Лабораторная работа 3
Методы сортировок

Цель работы: получение навыков реализации проекта сортировки одномерного массива, использующего иерархическое меню для выбора алгоритма сортировки.

Разработайте многомодульный проект, реализующий различные алгоритмы сортировки, по следующей технологии:

1. В окно первой формы поместите визуальный компонент *MainMenu* с четырьмя элементами – *Извлечением*, *Включением*, *Обменами*, *Специальными методами*. Пункт меню *Обменами* должен содержать три подпункта – *Фиксированное число просмотров*, *Минимизация числа просмотров*, *Один просмотр*. Пункт меню *Специальными методами* должен содержать два подпункта – *Слиянием* и *Распределением*.

2. В окно первой формы следует также поместить компоненты для задания исходного массива и отсортированного массива.

3. Для элементов меню *Извлечением*, *Включением*, *Фиксированное число просмотров*, *Минимизация числа просмотров* и *Один просмотр* соответствующие процедуры сортировки реализуйте в модуле *Unit1*.

4. Выбор элемента *Слиянием* должен приводить к активизации второй формы, в которой помещаются таблицы для двух исходных упорядоченных массивов и результата их слияния. Выбор элемента *Распределением* должен приводить к активизации третьей формы, на которой задаются исходный массив, массив ключей и таблица для размещения результата.

5. Для указанного преподавателем варианта разработайте пять процедур сортировки в модуле *Unit1* и по одной в модулях *Unit2* и *Unit3*.

Вариант 1

Отсортировать массив по невозрастанию методом извлечения минимального элемента. Поиск минимального элемента проводить справа налево.

Отсортировать массив по неубыванию методом включения с выбором включаемого элемента слева направо.

Отсортировать массив по невозрастанию методом обменов рядом стоящих элементов с фиксированным числом просмотров, направленных слева направо.

Отсортировать массив по неубыванию методом обменов рядом стоящих элементов с минимально необходимым (переменным) числом просмотров, направленных справа налево.

Отсортировать массив по невозрастанию методом обменов рядом стоящих элементов за один просмотр (с возвратами) слева направо.

Получить упорядоченный по неубыванию массив методом слияния двух упорядоченных по неубыванию массивов.

Отсортировать массив по неубыванию методом распределения по массиву ключей, упорядоченному по возрастанию.

Вариант 2

Отсортировать массив по невозрастанию методом извлечения максимального элемента. Поиск минимального элемента проводить справа налево.

Отсортировать массив по невозрастанию методом включения с выбором включаемого элемента справа налево.

Отсортировать массив по неубыванию методом обменов рядом стоящих элементов с фиксированным числом просмотров, направленных справа налево.

Отсортировать массив по неубыванию методом обменов рядом стоящих элементов с минимально необходимым (переменным) числом просмотров, направленных слева направо.

Отсортировать массив по невозрастанию методом обменов рядом стоящих элементов за один просмотр (с возвратами) справа налево.

Получить упорядоченный по неубыванию массив методом слияния двух массивов, один из которых упорядочен по неубыванию, а другой – по невозрастанию.

Отсортировать массив по неубыванию методом распределения по массиву ключей, упорядоченному по убыванию.

Вариант 3

Отсортировать массив по невозрастанию методом извлечения минимального элемента. Поиск минимального элемента проводить слева направо.

Отсортировать массив по невозрастанию методом включения с выбором включаемого элемента слева направо.

Отсортировать массив по неубыванию методом обменов рядом стоящих элементов с фиксированным числом просмотров, направленных слева направо.

Отсортировать массив по невозрастанию методом обменов рядом стоящих элементов с минимально необходимым (переменным) числом просмотров, направленных справа налево.

Отсортировать массив по неубыванию методом обменов рядом стоящих элементов за один просмотр (с возвратами) слева направо.

Получить упорядоченный по неубыванию массив методом слияния двух массивов, один из которых упорядочен по невозрастанию, а другой – по неубыванию.

Отсортировать массив по невозрастанию методом распределения по массиву ключей, упорядоченному по убыванию.

Вариант 4

Отсортировать массив по невозрастанию методом извлечения максимального элемента. Поиск максимального элемента проводить слева направо.

Отсортировать массив по невозрастанию методом включения с выбором включаемого элемента справа налево.

Отсортировать массив по неубыванию методом обменов рядом стоящих элементов с фиксированным числом просмотров, направленных справа налево.

Отсортировать массив по невозрастанию методом обменов рядом стоящих элементов с минимально необходимым (переменным) числом просмотров, направленных слева направо.

Отсортировать массив по неубыванию методом обменов рядом стоящих элементов за один просмотр (с возвратами) справа налево.

Получить упорядоченный по неубыванию массив методом слияния двух упорядоченных по невозрастанию массивов.

Отсортировать массив по невозрастанию методом распределения по массиву ключей, упорядоченному по возрастанию.

Вариант 5

Отсортировать массив по неубыванию методом извлечения минимального элемента. Поиск минимального элемента проводить слева направо.

Отсортировать массив по неубыванию методом включения с выбором включаемого элемента слева направо.

Отсортировать массив по невозрастанию методом обменов рядом стоящих элементов с фиксированным числом просмотров, направленных слева направо.

Отсортировать массив по неубыванию методом обменов рядом стоящих элементов с минимально необходимым (переменным) числом просмотров, направленных справа налево.

Отсортировать массив по невозрастанию методом обменов рядом стоящих элементов за один просмотр (с возвратами) справа налево.

Получить упорядоченный по невозрастанию массив методом слияния двух упорядоченных по неубыванию массивов.

Отсортировать массив по невозрастанию методом распределения по массиву ключей, упорядоченному по убыванию.

Вариант 6

Отсортировать массив по неубыванию методом извлечения максимального элемента. Поиск максимального элемента проводить слева направо.

Отсортировать массив по неубыванию методом включения с выбором включаемого элемента справа налево.

Отсортировать массив по неубыванию методом обменов рядом стоящих элементов с фиксированным числом просмотров, направленных справа налево.

Отсортировать массив по невозрастанию методом обменов рядом стоящих элементов с минимально необходимым (переменным) числом просмотров, направленных справа налево.

Отсортировать массив по неубыванию методом обменов рядом стоящих элементов за один просмотр (с возвратами) слева направо.

Получить упорядоченный по невозрастанию массив методом слияния двух массивов, один из которых упорядочен по неубыванию, а другой – по невозрастанию.

Отсортировать массив по неубыванию методом распределения по массиву ключей, упорядоченному по убыванию.

Вариант 7

Отсортировать массив по неубыванию методом извлечения минимального элемента. Извлечение минимального элемента проводить справа налево.

Отсортировать массив по невозрастанию методом включения с выбором включаемого элемента слева направо.

Отсортировать массив по неубыванию методом обменов рядом стоящих элементов с фиксированным числом просмотров, направленных слева направо.

Отсортировать массив по неубыванию методом обменов рядом стоящих элементов с минимально необходимым (переменным) числом просмотров, направленных слева направо.

Отсортировать массив по неубыванию методом обменов рядом стоящих элементов за один просмотр (с возвратами) справа налево.

Получить упорядоченный по невозрастанию массив методом слияния двух массивов, один из которых упорядочен по невозрастанию, а другой – по неубыванию.

Отсортировать массив по неубыванию методом распределения по массиву ключей, упорядоченному по возрастанию.

Вариант 8

Отсортировать массив по неубыванию методом извлечения максимального элемента. Поиск максимального элемента проводить справа налево.

Отсортировать массив по невозрастанию методом включения с выбором включаемого элемента справа налево.

Отсортировать массив по неубыванию методом обменов рядом стоящих элементов с фиксированным числом просмотров, направленных справа налево.

Отсортировать массив по невозрастанию методом обменов рядом стоящих элементов с минимально необходимым (переменным) числом просмотров, направленных слева направо.

Отсортировать массив по невозрастанию методом обменов рядом стоящих элементов за один просмотр (с возвратами) слева направо.

Получить упорядоченный по невозрастанию массив методом слияния двух упорядоченных по невозрастанию массивов.

Отсортировать массив по невозрастанию методом распределения по массиву ключей, упорядоченному по возрастанию.

Лабораторная работа 4
Обработка упорядоченных массивов

***Цель работы*:** получение навыков реализации алгоритма проверки упорядоченности массива и методов обработки отсортированного массива.

Для указанного преподавателем варианта разработайте проект, в котором осуществляется проверка упорядоченности исходного массива и, в случае положительного ответа, реализуется указанный алгоритм обработки.

Вариант 1

В упорядоченном по невозрастанию массиве чисел найти произведение отрицательных чисел и количество нулей.

Вариант 2

В упорядоченном по убыванию массиве чисел найти сумму положительных чисел и определить, есть ли в нем отрицательные числа.

Вариант 3

В упорядоченном по неубыванию массиве чисел найти произведение отрицательных чисел и определить, есть ли в массиве положительные числа.

Вариант 4

В упорядоченном по возрастанию массиве чисел найти произведение отрицательных чисел и определить, есть ли в массиве положительные числа.

Вариант 5

В упорядоченном по неубыванию массиве чисел найти количество отрицательных чисел и определить, есть ли в массиве нули.

Вариант 6

В упорядоченном по неубыванию массиве чисел определить, есть ли заданное число A, и найти количество чисел, меньших A

Вариант 7

В упорядоченном по невозрастанию массиве чисел найти количество чисел, меньших A, и определить, есть ли в массиве числа, попадающие в промежуток [X; Y].

Вариант 8

В упорядоченном по невозрастанию массиве чисел найти количество чисел, больших A, и определить, есть ли в массиве отрицательные числа.

Вариант 9

В упорядоченном по убыванию массиве чисел найти сумму номеров нулевых чисел и определить, есть ли в массиве отрицательные числа.

Вариант 10

В упорядоченном по невозрастанию массиве чисел определить, на каком месте должно находиться заданное число B.

Вариант 11

В упорядоченном по возрастанию массиве чисел определить, есть ли заданное число A, если нет, то найти номер места, на котором оно должно находиться.

Вариант 12

В упорядоченном по возрастанию массиве чисел найти сумму номеров положительных чисел и определить, есть ли нули в этом массиве.

Вариант 13

В упорядоченном по невозрастанию массиве чисел найти сумму номеров отрицательных чисел и количество нулей.

Вариант 14

В упорядоченном по неубыванию массиве чисел найти среднее арифметическое чисел из заданного промежутка (X; Y) и определить, есть в массиве неотрицательные числа.

Лабораторная работа 5
Обработка строк. Работа с символами

Цель работы: получение навыков реализации алгоритмов поиска символов в строке.

Для указанного преподавателем варианта разработайте проект, в котором осуществляются вычисления и преобразования исходной строки, сформулированные в задании.

Вариант 1

Подсчитать общее количество символов «+» и «–» и заменить каждый символ «;» на пару символов «,.».

Вариант 2

После каждого символа «,» вставить пробел и подсчитать количество букв «A» и «B» отдельно.

Вариант 3

Заменить символ «*» на «++» и подсчитать общее количество букв «F» и «D».

Вариант 4

Подсчитать количество букв «C» и «D» отдельно и заменить каждую пару символов «**» на пробел.

Вариант 5

После каждого символа «!» вставить символ «I» и подсчитать общее количество цифр в строке.

Вариант 6

Удалить каждую пару символов «PQ» и подсчитать общее количество символов «.» и «,» в строке.

Вариант 7

Подсчитать количество пар символов «+ –» и заменить каждый символ «*» на «/».

Вариант 8

После каждой цифры вставить такую же цифру и подсчитать количество пар «AC» в строке.

Вариант 9

Удалить каждый символ «A», стоящий после «,», и подсчитать количество пар «BC» и «DE» отдельно.

Вариант 10

Подсчитать количество символов «.», стоящих перед пробелом, и заменить каждую пару символов «ST» на символ «P».

Вариант 11

После каждого символа «A» вставить пробел и подсчитать количество символов «B», стоящих между знаками «+» и «–».

Вариант 12

Удалить каждый символ «?», стоящий после «;», и подсчитать общее количество символов «0» и «O».

Вариант 13

Подсчитать количество символов «+», стоящих между «A» и «B», заменить каждый символ «0» на «OO».

Вариант 14

В каждую пару символов «AB» вставить символ «*», подсчитать, сколько раз в строке символ «*I*» стоит перед «2».

Вариант 15

Удалить все «,» из строки и подсчитать количество символов «F», стоящих после «+», и количество символов «F», стоящих после «–» (отдельно).

Вариант 16

Подсчитать количество пар «23» и «45» по отдельности и заменить каждый символ «/» на символ «:» с последующим пробелом.

Лабораторная работа 6
Обработка строк. Работа со словами

Цель работы: получение навыков реализации алгоритмов выделения слов в строке.

Для указанного преподавателем варианта разработайте проект, в котором задается строка текста, состоящая из нескольких слов. Слова отделяются последовательностью пробелов. В процедуре проекта должны производиться вычисления или преобразования, указанные в задании.

Вариант 1

Подсчитать количество слов и после каждого поставить запятую.

Вариант 2

Подсчитать количество букв в третьем слове.

Вариант 3

Перед первой буквой каждого слова вставить символ «*».

Вариант 4

Определить количество слов, начинающихся с буквы «А».

Вариант 5

Определить количество слов, в которых буква «П» встречается хотя бы один раз.

Вариант 6

Подсчитать количество слов, длина которых превышает 5.

Вариант 7

Удалить последнюю букву в каждом слове.

Вариант 8

После последней буквы каждого слова вставить точку.

Лабораторная работа 7
Массивы записей

***Цель работы*:** получение навыков реализации алгоритмов обработки экономической информации, представленной в табличной форме.

Для указанного преподавателем варианта разработайте проект, в котором должны быть описаны исходный и получаемый массивы записей. Структура записей должна быть определена на основе условия задачи. Процедура обработки щелчка по командной кнопке должна обеспечивать вычисление требуемых экономических показателей.

Вариант 1

Задана таблица из трех граф: *Наименование изделия*, *План выпуска*, *Фактически выпущено*. Разработайте алгоритм и программу вычисления процента выполнения плана для каждого изделия.

Результаты расчета представьте в виде таблицы, состоящей из двух граф: *Наименование изделия*, *Выполнение плана, %.*

Вариант 2

Задана таблица, характеризующая потребность в строительных материалах и содержащая 13 граф: *Наименование материала*, *Январь*, *Февраль*, …, *Декабрь*. В каждой графе с названием месяца указана потребность материала в тоннах.

Разработайте алгоритм и программу вычисления годовой потребности каждого материала и получения выходного документа, содержащего две графы: *Наименование материала*, *Годовая потребность, т*.

Вариант 3

Задана таблица, характеризующая стоимость выпущенной предприятием продукции по кварталам и содержащая пять граф: *Наименование предприятия*, *Квартал I*, *Квартал II*, *Квартал III*, *Квартал IV*. В последних четырех графах указана стоимость выпущенной продукции в тысячах рублей в соответствующем квартале.

Разработайте алгоритм и программу вычисления выпуска продукции в денежном выражении каждым предприятием за год. Результаты расчета представьте в виде таблицы, содержащей две графы: *Наименование предприятия*, *Годовой объем выпуска, тыс. р.*

Вариант 4

Имеется документ, содержащий сведения о реализации товаров в отчетном и базисном периодах: *Наименование товара*, *Количество, реализованное в отчетном периоде*, *Цена за единицу в отчетном периоде*, *Количество, реализованное в базисном периоде*, *Цена за единицу в базисном периоде*.

Разработайте алгоритм и программу вычисления стоимости реализации каждого товара. Результаты расчета представьте в виде таблицы, содержащей три графы: *Наименование товара*, *Стоимость реализации в отчетном периоде*, *Стоимость реализации в базисном периоде*.

Вариант 5

Имеется сводка о простоях конвейера с начала месяца по некоторому цеху, содержащая четыре графы: *Дата*, *1 смена*, *2 смена*, *3 смена*. В последних трех графах указано время простоя в часах. Разработайте алгоритм и программу определения общего времени простоя в каждый день месяца за все смены.

Результаты расчета представьте в виде таблицы из двух граф: *Дата*, *Общее время простоя, ч*.

Вариант 6

Технологическая карта изготовления изделия содержит данные о деталях, используемых при выпуске изделия: *Наименование детали*, *Количество деталей в изделии*, *Наименование материала*, *Норма расхода материала на одну деталь, кг*. Каждая деталь упоминается в карте только один раз.

Разработайте алгоритм и программу формирования таблицы с графами: *Наименование детали*, *Наименование материала*, *Расход материала на деталь, кг*.

Вариант 7

Имеется документ, содержащий сведения о реализации товаров в отчетном периоде: *Наименование товара*, *Объем реализации, т*, *Цена за тонну, тыс. р.*

Разработайте алгоритм и программу формирования таблицы с графами: *Наименование товара*, *Стоимость реализации, тыс. р*.

Вариант 8

Имеется документ, содержащий сведения о среднемесячной численность рабочих в отчетном периоде: *Наименование предприятия*, *Численность по плану*, *Фактическая численность*.

Разработайте алгоритм и программу формирования таблицы с графами: *Наименование предприятия*, *Отклонение от плана*.

Вариант 9

Задана таблица из трех граф: *Наименование изделия*, *Наименование детали*, *Количество деталей, шт*. Известно, что одна и та же деталь может входить в разные изделия. Разработайте алгоритм и программу подсчета количества одноименных деталей по всем изделиям.

Результаты расчета представьте в виде таблицы из двух граф: *Наименование детали*, *Общее количество деталей, шт*.

Указание: целесообразно после ввода исходных данных в массив записей отсортировать массив по значениям графы *Наименование детали*.

Вариант 10

Задана таблица, характеризующая стоимость выпущенной продукции по кварталам и содержащая 6 граф: *Наименование предприятия*, *Наименование изделия*, *Квартал I*, *Квартал II*, *Квартал III*, *Квартал IV*. В последних четырех графах указана стоимость выпущенной продукции в тысячах рублей в соответствующем квартале. Известно, что одно и то же изделие может выпускаться разными предприятиями и каждое предприятие может выпускать несколько видов изделий.

Вычислите выпуск продукции в денежном выражении каждым предприятием за год по всем изделиям. Результаты расчета представьте в виде таблицы, содержащей две графы: *Наименование предприятия*, *Годовой объем выпуска, тыс. р.*

Указание: целесообразно после ввода исходных данных в массив записей отсортировать массив по значениям графы *Наименование предприятия.*

Вариант 11

Имеется сводка о простоях конвейера с начала месяца по некоторому цеху, содержащая 5 граф: *Дата*, *Причина простоя*, *1 смена, 2 смена*, *3 смена*. В последних трех графах указано время простоя в часах. Разработайте алгоритм и программу определения времени простоя в каждый день месяца по всем причинам за все смены.

Результаты расчета представьте в виде таблицы из двух граф: *Дата*, *Общее время простоя, ч*.

Указание: целесообразно после ввода исходных данных в массив записей отсортировать массив по значениям графы *Дата*.

Вариант 12

Технологическая карта содержит данные на каждое из выпускаемых изделий: *Наименование изделия*, *Наименование детали*, *Количество деталей в изделии*, *Наименование материала*, *Норма расхода материала на одну деталь, кг*. Известно, что в одно изделие могут входить несколько деталей и на изготовление одной детали могут потребоваться различные материалы.

Разработайте алгоритм и программу формирования таблицы с графами: *Наименование изделия*, *Наименование материала*, *Расход материала на изделие, кг*.

Указание: целесообразно после ввода исходных данных в массив записей отсортировать массив по двум итогам: *Наименование изделия* и *Наименование материала*.

Лабораторная работа 8
Использование типизированных файлов

***Цель работы*:** получение навыков использования в проектах типизированных файлов.

Для указанного преподавателем варианта лабораторной работы № 7 разработайте трехмодульный проект.

В окно первой формы поместите меню из двух пунктов: *Ввод данных* и *Результаты*. Описание типизированного файла расположите в модуле *Unit1*.

В окно второй формы, активизируемом по команде *Ввод данных*, поместите поля для ввода исходных данных и кнопки: *Запись в файл* и *В главное меню*. Щелчок по любой из кнопок должен обеспечить запись в типизированный файл всей введенной информации. При этом после использования первой кнопки поля формы должны очищаться.

В окно третьей формы, активизируемом по команде *Результаты*, поместите визуальные компоненты для вывода результатов обработки исходных данных и командную кнопку *В главное меню*.

Лабораторная работа 9
Использование текстовых файлов

***Цель работы*:** получение навыков использования в проектах текстовых файлов.

Выполните задание лабораторной работы № 8 с использованием текстового файла.

Лабораторная работа 10
Использование подпрограмм пользователя

***Цель работы*:** получение навыков использования в проектах процедур и функций пользователя.

Вариант 1

По данным о ежемесячных материальных затратах двух организаций за отчетный период определите для каждой организации следующее:

- сколько месяцев организация отработала неэффективно, т. е. превысила отраслевой норматив;
- какова доля затрат этих месяцев в общей сумме материальных затрат организации за отчетный период.

Вариант 2

По данным о ежемесячном объеме выпуска товарной продукции двумя предприятиями за отчетный период определите для каждого предприятия следующее:

- сколько месяцев предприятие отработало «хуже» и «лучше» среднего уровня;
- какова доля «лучших» месяцев в общем объеме выпущенной продукции организацией за отчетный период.

Вариант 3

По данным о ежемесячном объеме выплаченной заработной платы работникам двух организаций за отчетный период определите для каждой организации следующее:

- среднемесячный размер фонда заработной платы;
- долю суммы заработной платы, выплаченной в первые два месяца анализируемого периода в общей сумме выплат за весь период.

Вариант 4

По данным о ежемесячно получаемой прибыли в двух организациях в течение отчетного периода определите для каждой организации следующее:

- сколько месяцев организация отработала «лучше» среднего уровня;
- какова доля прибыли этих месяцев в общей сумме прибыли, полученной организацией за анализируемый период.

Вариант 5

По данным о ежемесячных материальных затратах двух организаций за отчетный период определите для каждой организации следующее:

- сколько месяцев организация отработала лучше, чем заложено по нормативу;
- какова доля этих месяцев в общей сумме материальных затрат организации за отчетный период.

Вариант 6

По данным о среднемесячной численности персонала двух фирм за отчетный период определите для каждой фирмы следующее:

- количество месяцев, когда фирма работала в составе, меньшем заданного значения;
- удельный вес этих месяцев в продолжительности отчетного периода.

Вариант 7

По данным о месячных объемах фактически привлеченных средств во вклады филиалами двух банков за отчетный период найдите для каждого банка следующее:

- общую сумму привлеченных средств филиалами банка;
- среднюю сумму привлеченных средств за те месяцы, когда привлекалось средств меньше заданного значения.

Вариант 8

По данным о ежемесячных затратах двух организаций на сырье за отчетный период определите для каждой организации следующее:

- сколько месяцев организация расходовала сырья больше норматива;
- какова доля этих затрат в общей сумме затрат на сырье организации за анализируемый период.

Вариант 9

По данным о цеховых расходах двух организаций за конкретный месяц определите для каждой организации следующее:

- количество цехов, отработавших на уровне среднецеховых расходов;
- долю расходов тех цехов организации, которые отработали «хуже» первого цеха.

Вариант 10

По данным о фонде заработной платы производственных рабочих цехов двух организаций найдите для каждой организации следующее:

- среднецеховую заработную плату производственных рабочих;
- долю заработной платы рабочих тех цехов, которые имеют фонд заработной платы ниже заданного значения.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – Введ. 1992-01-01. – М. : Изд-во стандартов, 1992. – 28 с.

Информатика : учеб. для вузов / под ред. Н. В. Макаровой. – М. : Финансы и статистика, 2003. – 768 с.

Вычислительная техника и программирование : учеб. для техн. вузов / А. В. Петров [и др.] ; под ред. А. В. Петрова. – М. : Высш. шк., 1990. – 478 с.

Попов, В. Б. Паскаль и Дельфи : учеб. курс / В. Б. Попов. – СПб. : Питер, 2005. – 576 с.

Алексеев, В. Е. Вычислительная техника и программирование : практикум по программированию / В. Е. Алексеев, А. С. Ваулин, Г. Б. Петрова ; под ред. А. В. Петрова. – М. : Высш. шк., 1991. – 399 с.

Прикладная информатика : учеб. пособие / А. М. Морозевич [и др.] ; под ред. А. М. Морозевича. – Минск : Выш. шк., 2003. – 335 с.

Фаронов, В. В. Турбо Паскаль 7.0. Начальный курс : учеб. пособие / В. В. Фаронов. – М. : КНОРУС, 2006. – 576 с.

Фаронов, В. В. Delphi. Программирование на языке высокого уровня : учеб. для вузов / В. В. Фаронов. – СПб. : Питер, 2004. – 640 с.

Глинский, Я. В. Turbo Pascal 7.0. Delphi : учеб. пособие для вузов / Я. В. Глинский, В. Е. Анохин, В. А. Ряжская. – СПб. : ДиаСофт-ЮП, 2001. – 208 с.

Могилев, А. В. Информатика : учеб. пособие для вузов / А. В. Могилев, Н. И. Пак, Е. К. Хеннер ; под ред. Е. К. Хеннера. – М. : Академия, 1999. – 816 с.

Меняев, М. Ф. Информатика и основы программирования : учеб. пособие для вузов / М. Ф. Меняев. – М. : Омега-Л, 2005. – 432 с.

Основы информатики и вычислительной техники. Информатика : практикум к лабораторным работам по одноименным курсам для студентов экономических специальностей дневной и заочной форм обучения / авт.-сост. : Н. В. Водополова, В. И. Мисюткин, С. А. Чабуркина. – Гомель : ГГТУ им. П. О. Сухого, 2005. – 33 с.

Основные приемы программирования в объектно-ориентированных языках : практикум по выполнению лабораторных и контрольных работ по курсам «Информатика» и «Основы информатики и вычислительной техники» для студентов экономических специальностей / авт.-сост. : Н. В. Водополова, В. И. Мисюткин, С. А. Чабуркина. – Гомель : ГГТУ им. П. О. Сухого, 2006. – 41 с.

Алгоритмизация и программирование : пособие для студентов специальности 1-25 01 07 «Экономика и управление на предприятии» специализации 1-25 01 07 02 «Экономическая информатика», специальности 1-26 03 01 «Управление информационными ресурсами» / авт.-сост. : С. М. Мовшович, О. А. Кравченко. – Гомель : Бел. торгово-экон. ун-т потребит. кооп., 2008. – 104 с.

СОДЕРЖАНИЕ

Учебное издание

ОБРАБОТКА ЭКОНОМИЧЕСКОЙ ИНФОРМАЦИИ В СИСТЕМЕ ПРОГРАММИРОВАНИЯ DELPHI

Пособие
по дисциплине «Алгоритмизация и программирование»
для студентов специальности 1-26 03 01 «Управление информационными ресурсами»

Авторы-составители:
Мовшович Семен Михайлович
Кравченко Ольга Алексеевна

Редактор О. В. Ивановская
Технический редактор Н. Н. Короедова
Компьютерная верстка И. А. Козлова

Подписано в печать 25.03.11. Бумага типографская №1.
Формат 60×84 $^{1}/_{16}$. Гранитура Таймс. Ризография.
Усл. печ. л. 7,44. Уч.-изд. л. 7,50 Тираж 130 экз.
Заказ № 18-03-11.

Учреждение образования
«Белорусский торгово-экономический
университет потребительской кооперации».
246029, г. Гомель, просп. Октября, 50.
ЛИ № 02330/0494302 от 04.03.2009 г.

Отпечатано в учреждении образования
«Белорусский торгово-экономический
университет потребительской кооперации».
246029, г. Гомель, просп. Октября, 50.

БЕЛКООПСОЮЗ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БЕЛОРУССКИЙ ТОРГОВО-ЭКОНОМИЧЕСКИЙ
УНИВЕРСИТЕТ ПОТРЕБИТЕЛЬСКОЙ КООПЕРАЦИИ»

Кафедра информационно-вычислительных систем

ОБРАБОТКА ЭКОНОМИЧЕСКОЙ ИНФОРМАЦИИ В СИСТЕМЕ ПРОГРАММИРОВАНИЯ DELPHI

Пособие
по дисциплине «Алгоритмизация и программирование»
для студентов специальности 1-26 03 01 «Управление информационными ресурсами»

Гомель 2011