Практическое занятие №2
Поиск экстремума, накопление суммы, количества, произведения. Текстовые файлы

Задание: обсудить 1) создание функциональных тестов, описание метода и алгоритма на примере двух задач (Лабораторная работа №3 и №5); 2) использование нестандартных текстовых файлов; 3) необходимость использования параметров программы; 4) возможность сравнения символов и строк, конкатенацию строк.

2.1. Схема решения задачи из Лабораторной работы №5 «Поиск экстремума (максимума, минимума) в одномерном массиве»

По структуре данных и алгоритма это задача того же класса, что Задача №1 из Лабораторных работ №2-4, так что образец полной разработки есть: Пример-отчета-для-лабораторной-работы-2.*pdf*.

1. Задача Extremum. Постановка задачи (ПЗ)
Задание: Написать программу обработки одномерного массива(ов) в соответствии с условием.
Условие: Найти номер и значение максимального элемента в одномерном массиве ***a***(***n***).

2.Уточненная ПЗ (*уточнить структуры, типы, имена, добавить альтернативные («отрицательные» и особые) решения)*.
Задан одномерный целочисленный массив ***a***, состоящий из ***n*** элементов.
Найти значение ***Amax*** и номер ***Kmax*** максимального элемента заданного массива ***a***.
Если элементов с максимальным значением в массиве будет несколько, то найти номер *первого* из них.

3. Пример, на базе которого затем построим тесты. Возьмем ***n***=6.

a

-4	0	-5	**9**	**9**	2
1	2	3	4	5	6

Максимальное значение: ***Amax*** = 9,
Номер первой 9-ки: ***Kmax*** = 4

4. Таблица данных

Класс	Имя	Описание (смысл), диапазон, точность	Тип	Структура	Формат
Входные данные	***a***	заданный массив, $\|a_i\|<100$	цел	одномерный массив (20)	+XX (:4)
	n	число элементов массива a, $0 < n \le 20$	цел	простая переменная	XX (:2)
Выходные данные	***kmax***	номер максимального элемента, .* . .	*	*	*
	amax	его значение, . . .*	*	*	*
Промежу-точные	***i***	индекс текущего элемента, . . .*	*	*	---
	dat	входной файл *Extr_dat*<№теста>.*txt*	файл	текстовый	---
	res	выходной файл *Extr_res*<№теста>.*txt*	файл	текстовый	---
					---

**Диапазоны, типы, точность, структуру, формат для выходных и промежуточных параметров заполнить самим.*

5. Форма ввода (*Extr_dat*<№теста>*.txt***)**

```
<n>
<a[1]>
<a[2]>
. . . .
<a[n]>
```

Данные вводить из текстового файла.

Форма ввода элементарна, поэтому образцы можно не отмечать.

Программировать ввод-вывод в точном соответствии с входной и выходной формами!

6. Форма вывода (*Extr_res*<№теста>*.txt*)

```
обр1                    Поиск максимума в массиве
обр2    Число элементов  = <n>
        Значения элементов:
        <a[1]>
обр3    <a[2]>
        ......
        <a[n]>
обр4    Максимальный элемент = <Amax>
обр5    Его номер = <Kmax>
```

7. Аномалии не рассматриваем (но желательно успевающим сделать хоть что-то: $n<1$, $n>20$, $|a_i|>=100$, отсутствует входной файл, неправильный формат входного файла (ошибка при чтении), невозможно создать выходной файл, ошибка при чтении/записи в файл).

8. Функциональные тесты составить самостоятельно.

В том числе будут полезными тесты, в которых:

1) один экстремум в середине/конце массива;
2) один экстремум, и он первый в массиве;
3) более одного экстремума (несколько равных максимумов/минимумов: см пример);
4) все равные элементы.

При этом опять же нужно покрыть все возможные части диапазонов исходных данных и результатов: заполните полностью таблицу, указав номера подходящих тестов в пустых ячейках:

:**Исходные данные**								**Результаты**		Тест№
	аном	*граница*	*сред*		*сред*	*граница*	*аном*	***Kmax***	*макс* =20	
N	<1	1	(2	,	19)	20	>20		*мин* = 1	
Тест№	---		*1*	←	←	*2*	---		*сред* = (0,20)	1
	аном	*граница*	*сред*	*0*	*сред*	*граница*	*аном*		*не сущ* = не возм-но	---
***a*[*i*]**	<-99	-99	(-99,0)	0	(0,99)	99	>99		*0* = не возможно	---
Тест№	---	*2*	*1*	*1*	*1*		---		*Макс.вычислит. нагрузка* = при ***n*** =20	2
								Amax	*макс* = 99	
									мин = -99	2
									сред = (-99, 99)	1
									не сущ = не возм-но	---
									0 = 0	---
									Макс. выч. нагрузка = при ***n*** =20	2

№ теста	Входные данные	Ожидаемый результат	Смысл теста
1	***n*** = 6 **a** | -4 | 0 | -5 | 9 | 9 | 2	***Amax*** =9 ***Kmax*** = 4	Несколько максимумов в середине массива (см.Пример)
2	***n*** = 20 **a** | -99 | -99 | … | -99	***Amax*** = -99 ***Kmax*** = 1	Все значения в массиве одинаковые, минимально возможные
…	*		

*Далее заполните пробелы в этой и предыдущей таблице самостоятельно

9. Метод
(Методы, алгоритмы и код базовых алгоритмов ищите в файле Базовые-алгоритмы.*pdf*)

Рассмотрим процесс поиска максимума и его номера на приведенном выше примере:

a

-4	0	-5	**9**	**9**	2
1	2	3	4	5	6

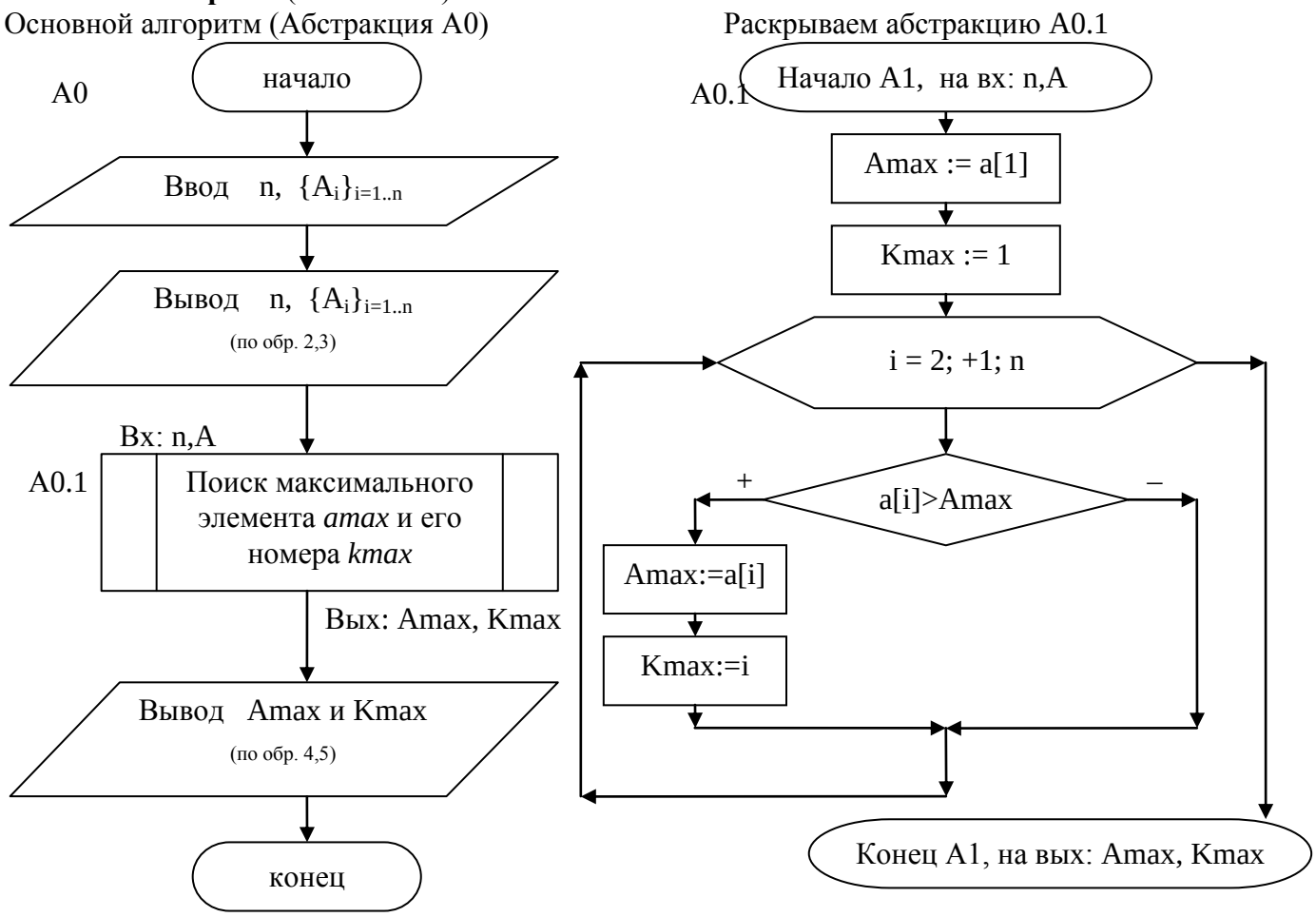
	Начало	0 > -4	-5 < 0	9 > 0	9 ≤ 9	2 < 9	Результат
Amax	**-4**	**0**	0	**9**	9	9	9
Kmax	**1**	**2**	2	**4**	4	4	4

Сначала считаем максимальным элементом *первый*.

Затем, просматривая поочередно *все остальные* элементы массива, сравниваем каждый из них с текущим значением максимума (с ***Amax***).

Если текущий элемент больше ***Amax***, то заменяем значение ***Amax*** на значение текущего элемента, а значение ***Kmax*** – на его номер.

10. Алгоритм (блок-схема)



11. Программный код.

Написать самостоятельно, используя цикл for и ветвление if внутри него.

Подсказки в файле Базовые-алгоритмы.pdf и Кодирование-алгоритмов.pdf, а также Пример-отчета-для-лабораторной-работы-2.pdf

Замечание 1. При решении данной задачи можно обойтись без переменной ***Amax.*** При этом для сравнения с текущим элементом и при выводе результата указать на значение максимального элемента можно по его индексу в массиве: ***a***[***Kmax***].

Замечание 2. В вашей задаче может быть экстремального значения не просто самого элемента, а заданного *арифметического выражения*. Тогда за начальное значение *Amax* берется значение первого выражения (а не просто элемента) и вводится дополнительная переменная (*Zmax*, например) для хранения текущего значения выражения, чтобы затем сравнить его с *Amax.*

Замечание 3. Для поиска *минимума* заменяем знак при сравнении текущего и экстремального значения с ">" на "<". Желательно также сменить и названия переменных на *Amin*, *Kmin*, *Zmin* для улучшения читабельности кода.

Замечание 4. В массиве элементы могут *совпадать*, и поэтому при поиске номера требуется уточнение: найти номер *первого* (*последнего*) максимума (минимума), что и было сделано в уточненной постановке рассмотренной задачи. Например, в массиве *a* = (1, 9, 2, 3, 9, 7) *Amax* = 9, *Kmax* = 2 при поиске *первого* максимума и *Kmax* = 5 при поиске *последнего* элемента с максимальным значением. Если при поиске *первого* для проверки используется знак > (<); то при поиске *последнего*: он заменяется на >= (<=); или, не меняя знака, элементы массива можно перебирать с конца («последний с начала» = «первый с конца»), тогда за начальное значение *Amax* берется значение *последнего* элемента и используется цикл с уменьшающимся параметром (*for i*:= *n*-1 ***downto*** 1 *do*)

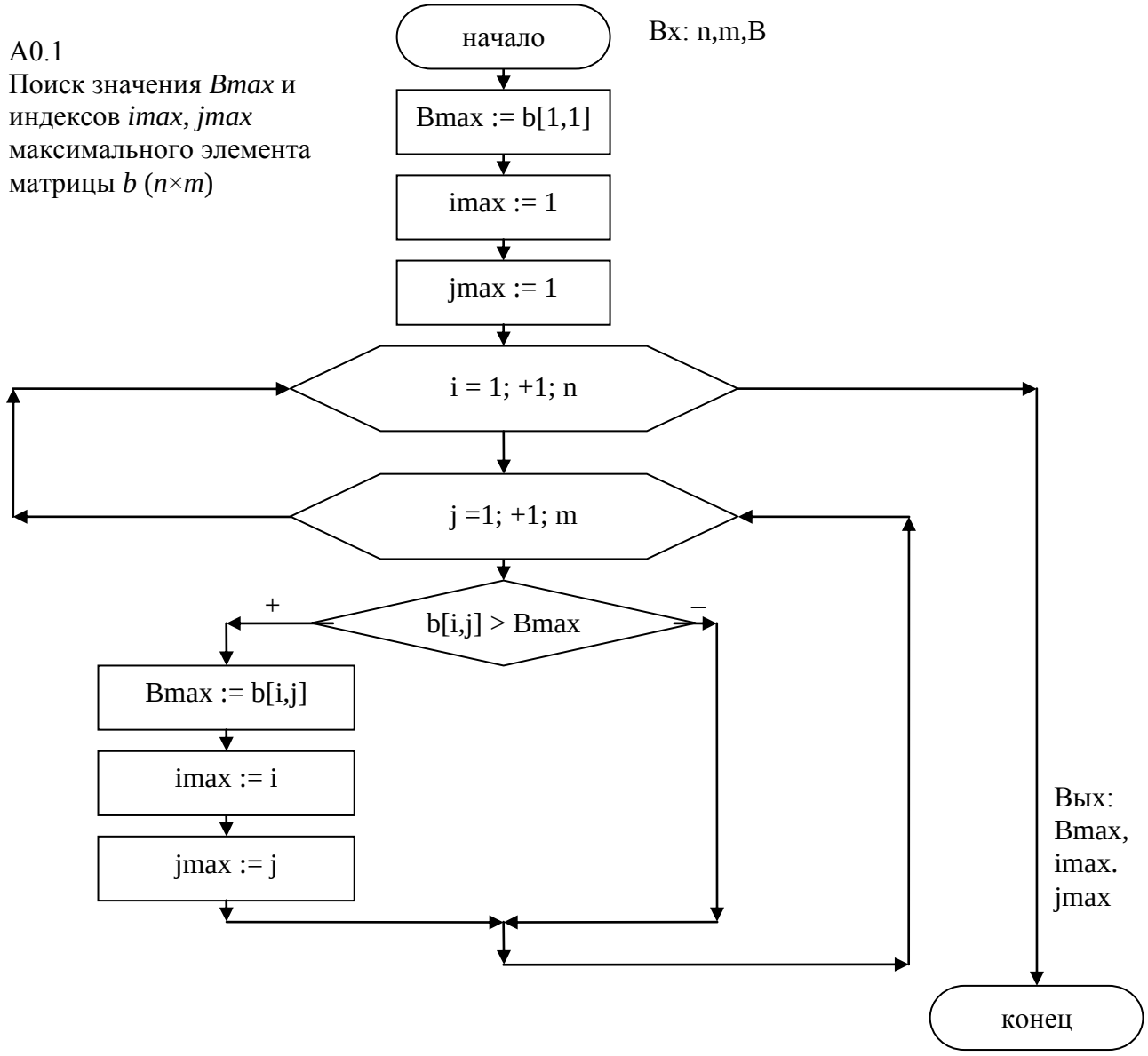


Рис. Раскрытие абстракции A0.1 для поиска максимума в матрице

Замечание 5. Метод поиска экстремума *Bmax* в *матрице b*(*n*×*m*) из *n* строк и *m* столбцов и двух(!) его индексов *imax, jmax* аналогичен, но надо учесть, что все элементы матрицы перебираются с помощью двух вложенных один в другой циклов: один – по строкам, другой – по столбцам (элементам строки), и что начинаем оба цикла с первого элемента (*i* = 1, *j* = 1), а не со второго, иначе потеряется целый столбец или строка:

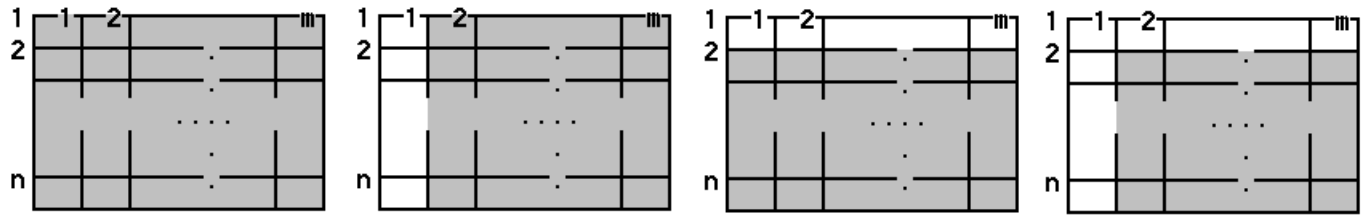


Рис. Потеря строки или столбца ри поиске не с первого элемента

Замечание 6. Если экстремальные элементы надо найти для всех (каждой в отдельности) строк или столбцов матрицы, то результатом будет не *простая переменная Amax*, а *одномерный массив* из *n* или *m* элементов соответственно.

Задание 1 для выполнения на занятии: В заданном одномерном массиве *X* из *m* элементов найти номер и значение элемента, для которого минимально значение произведения $X_i \cdot sin(X_i)$. Только блок-схему алгоритма. Если в массиве окажется несколько элементов с минимальным значением, найти

номер *последнего* из них двумя способами: при обходе массива от начала к концу и при обходе с конца в начало.

Задание 2 для выполнения на занятии: переделать готовую блок-схемы из предыдущего задания для аналогичного поиска в матрице при обходе ее слева направо сверху вниз.

2.2 **Пример использования файлов в задачи из Лабораторной работы №2**

Будем вводить данные из текстового файла на диске и выводить в другой текстовый файл на диске. Сразу зададим имена файлов и соответствующих файловых переменных, добавим их в таблицу данных и укажем их рядом со входной и выходной формами.

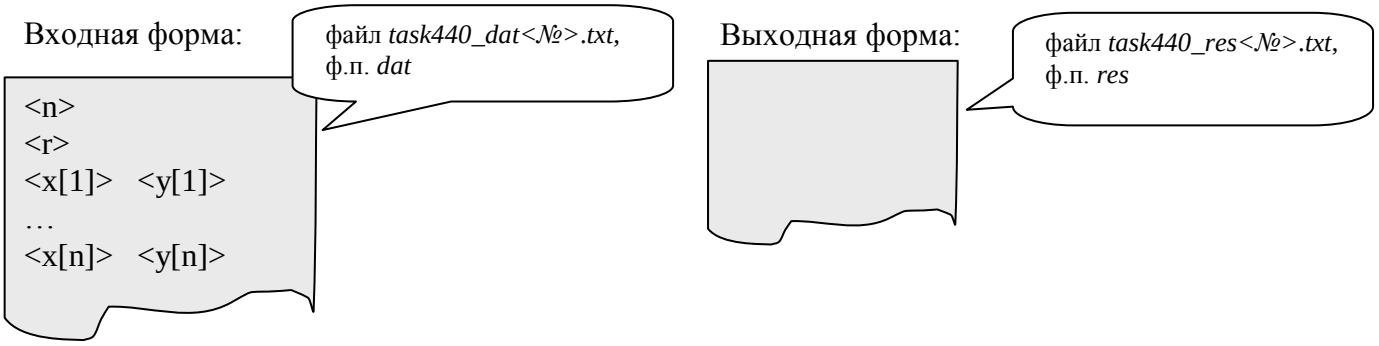
Поскольку входных/выходных файлов обычно несколько (как минимум тестовые примеры), то удобно их именовать по такой, например, схеме:

task440_dat<№>.txt – файлы входных данных, ***dat*** – соответствующая файловая переменная,
task440_res<№>.txt – файлы выходных данных, ***res*** – соответствующая файловая переменная,
где <№> – номер теста, например для первого теста: *task440_dat1.txt* и *task440_res1.txt*

Дополняем таблицу данных:

Класс	Имя	Описание (смысл)	Тип	Структура	Формат
Промежуточные	*Dat*	файловая переменная для входного файла ***task440_dat<№>.txt***	Текстовый	файл	См.форму ввода
	Res	файловая переменная для выходного файла ***task440_res<№>.txt***	Текстовый	файл	См.форму вывода

Указываем рядом с формами ввода-вывода, что это форматы файлов :



Файл *task440_dat<№>.txt* должен быть создан ***согласно входной форме*** до запуска программы с помощью любого редактора текстов, например в среде *Delphi* (*File→New→Other→Other_Files→Text*). В данном случае следует набрать только значения входных данных, *опустив приглашения*. Содержимое файла *task440_dat1.txt* для первого теста:

```
5
3
2  -2
2  1
4  3
-1  -2
-5 -5
```

Файл *task440_res1.txt* будет создан автоматически (Про открытие Ролик №5 в *Read_me.html*).

Изменения и добавления в программе:

1) В раздел описаний следует добавить описания файловых переменных:

```
var
     dat, res: TextFile;
```

2) В начало раздела операторов (сразу после begin) следует добавить операторы назначения и открытия файлов:

```
AssignFile (dat, 'task440_dat1.txt'); Reset (dat);
AssignFile (res, 'task440_res1.txt'); Rewrite (res);
```

3) ***Во все операторы ввода*** перед началом списка ввода в операторах ***read*** и ***readLn*** следует указать ф. п. ***dat***, (имя ф.п. и запятая – синтаксис см. выше),а ***во все операторы вывода write*** и ***writeLn*** перед началом списка вывода – ф.п. ***res***,

4) В конец раздела операторов (перед *end.*) следует добавить операторы закрытия файлов:

```
CloseFile(dat);  CloseFile(res);
```

5) Убрать приглашения для ввода, предусмотренные для режима диалога (образцы 1.1, 2.1 и 3).

Вопрос: тестов несколько, а файл входных (выходных) данных в программе – один. Как прогнать программу на всех тестовых примерах? Можно ли это сделать, **не меняя текста программы**?

Ответ Можно, например, используя параметры программы.

2.3 Использование параметров в программе на Delphi

2.3.1 Добавление в программу обращения к параметрам

Для проверки правильности программы разрабатываются тестовые примеры (тесты), и в процессе поиска и исправления ошибок необходимо проверять программу на *всех* тестах после каждого изменения программного кода до тех пор, пока не будет достигнут ожидаемый результат для всех тестовых примеров. При этом удобно исходные данные для всех тестов хранить в файлах на диске: *один тест – один текстовый файл*. Чтобы не менять программный код, меняя названия файлов для каждого теста можно использовать возможность передачи параметров программе при ее запуске. Результаты запуска для каждого теста также удобно хранить в отдельных файлах.

Пусть программа тестируется на двух тестах.

Пусть имя входного файла – 1-й параметр, имя выходного файла – 2-й параметр.

Пусть prog.dpr – файл с программным кодом головного модуля проекта,

prog.exe – соответствующий .exe-файл,
prog_dat1.txt – файл входных данных для первого теста,
prog_dat2.txt – файл входных данных для второго теста,
prog_res1.txt – файл выходных данных для первого теста,
prog_res2.txt – файл выходных данных для второго теста.

Создать текстовые файлы с исходными данными можно в Блокноте или в среде *Delphi* (меню *File→New→Other→Other_Files→Text*). Файлы с результатами создадутся автоматически при открытии с помощью *ReWrite*, либо надо создать их самим при открытии для дозаписи с помощью *Append*. Просмотреть результаты можно опять же с помощью простейшего текстового редактора «Блокнот» или прямо из среды разработки программ.

Изменения, которые надо произвести в тексте программы – минимальны: конкретные имена файлов заменяются на обращения к функции *ParamStr*:

```
assignFile(dat, ParamStr(1));
assignFile(res, ParamStr(2));
```

ParamStr – стандартная функция для работы с параметрами в *Delphi*, она возвращает параметр с заданным номером. Ее синтаксис:

```
function ParamStr(<№ параметра>: word): string;
```

Все параметры трактуются как отдельные строки (*string*). Параметры пользователя нумеруются, начиная с *единицы*. В *нулевом* параметре *ParamStr*(0) ОС передает программе полное имя запускаемого приложения (например, *D:\Гречкина\Project1.exe*). Этот (нулевой) параметр не входит в общее число параметров, которое можно узнать с помощью функции *ParamCount*:

```
function ParamCount: word;
```

Для вывода значений всех параметров можно использовать код:

```
Program Params;
{$AppType CONSOLE}
Var i:byte;
Begin
  For i:=0 to ParamCount do writeln('Param',i:2,' ', ParamStr(i));
  Write('Press ENTER'); ReadLN
End.
```

2.3.2 Запуск программы с параметрами

Запускать программу можно в отладочном режиме в среде разработке Delphi, но готовая программа все же представляет собой исполняемый exe-файл, который можно запускать вне среды разработки. Для запуска программ в среде *Windows* принято использовать ***ярлыки***, вынесенные на Рабочий стол или в меню Пуск.

а) Для запуска в среде *Delphi* перед запуском программы надо указать через пробел параметры в секции *Parameters* (меню *Run→Parameters*) (см. Рис.). Затем запустить программу как обычно: меню *Run→Run* или *F9.*

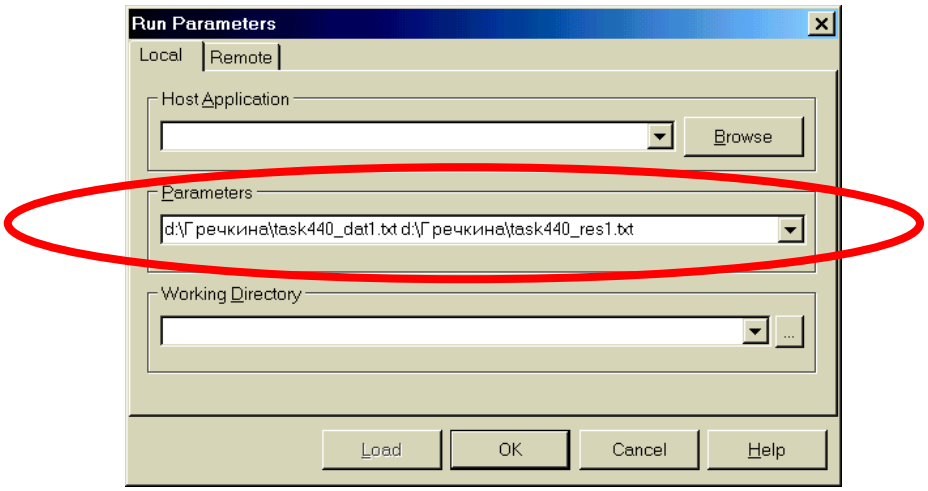


Рис. Указание параметров в среде *Delphi*

б) Вне среды разработки *Delphi* (в операционной среде *Microsoft Windows)* можно запустить готовую программу – *exe*-файл. Для этого надо создать для него ярлыки для каждого из тестов, указав в них нужные параметры после имени программы, и запускать с помощью этих ярлыков.

Для создания такого ярлыка в папке проекта:

1) Откройте папку со своим проектом и выполните один щелчок правой кнопкой мыши на значке исполняемого файла вашего проекта (он создается после компиляции (меню *Project→Compile* или *Ctrl+F9*) или запуска (меню *Run→Run* или *F9*) вашего проекта в среде *Delphi.*

2) После щелчка появится контекстное меню, в котором следует выбрать пункт «Создать ярлык». Должен появиться ярлык для выбранного файла (см. Рис.).

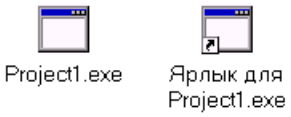


Рис. Ярлык

3) Теперь надо изменить свойства этого ярлыка:

– выполните один щелчок правой кнопкой мыши на значке ярлыка;

– в появившемся контекстном меню выберите последний пункт «Свойства»;

– в строке Объект (см. Рис.) добавьте два параметра (имена входного и выходного файлов через пробел) и нажмите кнопку «*OK*». Например,

D:\Гречкина\task440_dat1.txt D:\Гречкина\task440_res1.txt

Получится

D:\Гречкина\Project1.exe D:\Гречкина\task440_dat1.txt D:\Гречкина\task440_res1.txt

Если файлы находятся в Рабочей папке проекта, полный путь к ним указывать не обязательно, достаточно указать только имена:

D:\Гречкина\Project1.exe task440_dat1.txt task440_res1.txt

4) Для каждого теста надо создать свой ярлык и назвать их (ярлыки) можно соответственно Тест1, Тест2 и т.д.

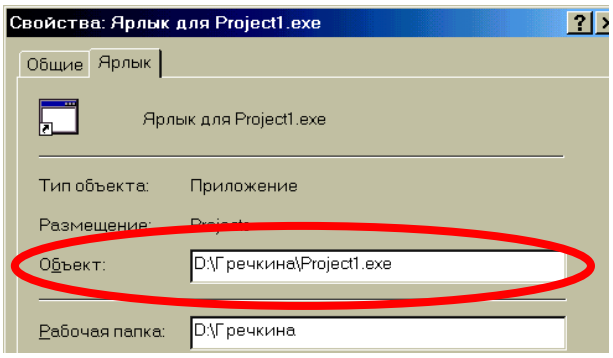


Рис.Свойства ярлыка

При запуске (двойной щелчок на ярлыке) параметры передаются в программу. Смысл параметров и способ их обработки в программе определяется программистом.

К сожалению, в ярлыках используется полный путь к файлу, и при копировании папки проекта в другое место на диске старый путь становится неактуальным. Поэтому можно для запуска программ с параметрами использовать ещё и *bat*-файлы. Чтобы создать *bat*-файл можно воспользоваться Блокнотом, создав обычный текстовый файл с командной строкой(ами) внутри и сохранив его с расширением *bat*. Например, для рассмотренного примера в папке проекта можно создать *bat*-файл со следующими строками:

Project1.exe task440_dat1.txt task440_res1.txt
Project1.exe task440_dat2.txt task440_res2.txt

Поскольку пути указаны относительные, то при смене места расположения папки проекта, обе команды запуска останутся актуальными.

2.4 Символьные и строковые данные

(опережающие, но полезные сведения; так, сведения о символах пригодятся в ***задаче упорядочения***)

Тип/ Структура	Константа	Возможное значение переменной	Описание переменной
Символьный/ Простая переменная	‘<символ>’	Символы согласно таблице символов ‘.’, ‘<’, ‘5’, ‘a’	Var <имя>: char;
Символьный/ Строка (<длина>)	‘<последовательность символов в апострофах>’	‘’ {пустая строка} ‘Несколько слов’	Var <имя>: **string**[<длина>]; {<длина> не должна превышать 255} <имя>: **ShortString;** //255 символов (это статические строки; динамические – в следующем семестре)

Символы упорядочены согласно *таблице кодов* и *порядок* почти совпадает с алфавитным. Таблицы кодов, используемые в операционных сиситемах *Windows* и *DOS*, не совпадают, поэтому *на экран символы кириллицы выводятся не корректно*. Сначала идут цифры (48..57), символы английского алфавита (ЗАГЛАВНЫЕ – 65..90, прописные – 97..122), затем символы русского алфавита (ЗАГЛАВНЫЕ – 192..223 и прописные – 224..255, Ё – 168, ё – 184, а в DOS ЗАГЛАВНЫЕ – 128..159, прописные – 160..175 и 224..239, Ё – 240, ё – 241). Символьные данные можно сравнивать обычным образом – при этом сравниваются *коды символов*. Таким образом, ‘0’<’1’<…<‘F’<’G’<…<’f’<’g’<…<‘Г’<’Д’<…<’я’. Строки также можно сравнивать: ‘01’<’1’<’10’<’A’<’ABC’<’ABD’<’z’<’Антон’<’Тоня’<’Я’<’Яблоко’.

Некоторые операции над *символами*:
Ord(<символ>) – функция, результатом которой является код символа в таблице кодов.
Chr(<код_символа>) – функция, результатом которой является символ с указанным кодом.
А также операции над порядковыми типами ***Succ***(<символ>) и ***Pred***(<символ>)
Некоторые операции над *строками*:
Length(<строка>) – функция, результатом которой является длина указанной строки.
+ (конкатенация) – строки можно складывать простым сложением их в одну строку:
‘Слово’+ ‘Слово’ → ‘СловоСлово’; ‘Слово’+ ‘ ‘+ ‘Слово’ → ‘Слово Слово’; ‘1’+ ‘20’ → ‘120’
Другие символьные и строковые типы и операции рассмотрим в следующем семестре.

Например:

```
program CharStr;          {скопируйте текст в новый проект, сохраните и запустите}
{$APPTYPE CONSOLE}

var
   a, b :char;          {простые символы}
   c: string[6];        {максимальную длину строки установим в 6 символов}
   s: ShortString;      {максимальная длина строки 255 символов}
   d, e, f: boolean;    {логическая переменная, принимающая два значения
                              - true (Истина) или false(Ложь)}
   len1, len2: integer;
   code: byte;
begin
    {задание значений переменным}
    a:='s';
    b:=' '; {b присваивается значение пробела, заключенного в кавычки}
     writeln('a=', a);
     writeln('b=', b);

    c:='';
    len1:=length(c); {длина строки c}
    len2:=ord(c[0]); {длина строки с другим способом (из нулевого символа)}
     writeln(#13#10'c=', c,'');   {вывод с пропуском строки}
     writeln('len1=', len1);  {вывод с пропуском строки #10 - eoln; #13 - перевод каретки}
     writeln('len2=', len2);
```

```
    c:='pascal Delphi';  {при присвоении строки длиннее 6, она будет обрезана до  6 символов}
    len1:=length(c); {длина строки c}
    len2:=ord(c[0]); {длина статической строки с другим способом}
      writeln(#13#10+'c=', c);   {вывод с пропуском строки}
      writeln('len1=', len1);
      writeln('len2=', len2);

    s:=c+' Delphi'; // конкатенация строк
    len1:=length(s); {длина строки s}
    len2:=ord(s[0]); {длина строки s другим способом}
      writeln(#13#10's=', s);  {вывод с пропуском строки}
      writeln('len1=', len1, '  len2=', len2);

       {сравнение символов}
    d:= a<b; {d=false}
       {сравнение символов строки. Обращение - как к элементам одномерного массива}
    e:= c[1]<c[length(c)]; {сравнение первого и последнего символа строки e=false}
    f:= c[3] = a; {сравнение 3-го символа строки и символа, хранящегося в переменной a f =
true}
      writeln(#13#10'd=', d);      {вывод с пропуском строки}
      writeln('e=', e, '  f=', f);

        {вывод символов кириллицы по их кодам}
    writeln;
    for code:=128 to 159 do write(Chr(code));
    writeln;
    for code:=160 to 175 do write(Chr(code));
    writeln;
    for code:=224 to 239 do write(Chr(code));
    writeln;
    for code:=240 to 241 do write(Chr(code));
    writeln;

    write(#13#10'Press ENTER');  {вывод с пропуском строки}
    readln;
end.
```

Про символьный тип также можно узнать из лекции «Простые типы данных»