

Министерство образования и науки Российской Федерации
КАЗАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. А.Н. ТУПОЛЕВА

Кафедра прикладной математики и информатики им. Ю.В. Кожевникова

Урахчинский И.Н.

ОПЕРАЦИОННЫЕ СИСТЕМЫ

*Методические указания
по лабораторным работам*

Казань 2007

Оглавление

<u>Введение</u>	2
<u>Лабораторная работа № 1</u>	3
<u>Лабораторная работа № 2</u>	21
<u>Лабораторная работа № 3</u>	34
<u>Лабораторная работа № 4</u>	43
<u>Лабораторная работа № 5</u>	54
<u>Лабораторная работа № 6,7</u>	66
<u>Лабораторная работа № 8</u>	84
<u>Литература</u>	96

Введение

До Windows 98 в Windows практически не было средств автоматизации резервного копирования файлов и выполнения рутинных операций по администрированию системы. Конечно, определенные задачи, такие как, копирование файлов, можно было выполнить и с помощью старых пакетных файлов MS-DOS (bat) в окне командой строки. Однако в bat-файлы можно помещать лишь простые последовательности команд MS-DOS, которые не поддерживают диалоговые окна и сообщения (среди их недостатков можно отметить поддержку только простого ветвления и отсутствие настоящей функциональности циклов).

Поскольку в Microsoft Office входит язык VBA (Visual Basic for Applications), а Web-разработчикам известны языки сценариев Microsoft Visual Basic, Scripting Edition (VBScript) и JavaScript от Netscape, создание Microsoft средства работы со сценариями для Windows было лишь вопросом времени. Windows Script Host (WSH) и является таким средством, которое попадает в категорию технологий Microsoft Windows Script.

WSH — это автономный сервер сценариев, позволяющий исполнять сценарии на уровне ОС. Например, можно вызвать сценарий из командной строки или из Windows Explorer, дважды щелкнув файл сценария. WSH удобен для решения многих задач администрирования, для которых пользовательский интерфейс практически не нужен. Он намного гибче в сравнении со старыми пакетными файлами MS-DOS, так как JScript и VBScript являются мощными языками сценариев с полным доступом к объектам WSH и любым другим объектам автоматизации.

Лабораторная работа № 1
Введение в Windows Script Host и JScript

Цель работы:

- 1) Изучение основ Windows Scripting Host.
- 2) Изучение языка JScript.
- 3) Получение первичных навыков разработки простейших сценариев.

Содержание работы:

- 1) Изучение структуры и синтаксиса языка JScript.
- 2) Изучение основных конструкций JScript.
- 3) Изучение и выполнение всех рассмотренных в руководстве примеров.
- 4) Разработка процедуры решения поставленной задачи.
- 5) Отладка и выполнение процедуры.
- 6) Составление отчета.

Содержание отчета:

- 1) Краткие теоретические сведения.
- 2) Постановка задачи.
- 3) Описание процедуры.
- 4) Руководство пользователю процедуры.
- 5) Листинг процедуры.
- 6) Результаты выполнения процедуры.

Краткие теоретические сведения

Windows Scripting Host (WSH) – языконезависимая платформа для Win32. Microsoft предоставляет его как вместе со своими продуктами (Windows

98, IIS 4.0, Windows 2000 и выше), так и как отдельный инсталлятор (для **Windows 95** и **Windows NT 4**). В базовую поставку входит поддержка скриптовых языков **Java Script** и **Visual Basic Script**, но возможна установка расширений для поддержки других скриптовых языков (разработаны расширения для **Perl**, **TCL**, **REXX**, **Python** и многих других).

Раньше единственным поддерживаемым **Windows** скриптовым языком был командный язык **DOS**. Но его возможности довольно бедны по сравнению с **VBScript** и **JScript**. Хотя командный язык **DOS** по-прежнему поддерживается, современные **ActiveX**-скрипты позволяют решать более сложные задачи.

Скрипт может принимать решения на основе использования полноценных операторов **If/Else**. Сценарий может выполнять один набор команд, если данное условие истинно, или другой набор, если условие ложно. Кроме того, **JScript** и **VBScript** хороши при выполнении математических операций, включая общие тригонометрические функции.

Другое свойство **WSH** заключается в том, что скрипты могут исполняться вне браузера. Достаточно кликнуть по файлу с текстом скрипта или ввести его название в командной строке, чтобы запустить его на исполнение. **WSH** не требователен к памяти и прекрасно подходит для автоматизации **Windows**.

Как ни хороши скриптовые языки, многого с ними не сотворишь. Они неспособны повлиять на что-либо за пределами родного скрипта. Эти языки не имеют прямых инструкций, позволяющих читать или записывать файлы на диске, выводить информацию в командную строку, изменять записи в Реестре **Windows** и так далее.

Чтобы справиться с такими задачами, можно воспользоваться дополнительными **COM**-объектами. Ряд таких объектов входит в поставку **WSH**, а один из них, **WScript**, даже уже имеет созданный экземпляр, и им можно пользоваться непосредственно, без предварительного создания. Остальные создаются с помощью синтаксиса, соответствующего конкретному языку или функции **WScript.CreateObject**.

Создание и запуск скриптов

По сравнению с различными программистскими изысками писать скрипты для **WSH** относительно просто. В **Notepad** вы пишете свой скрипт с использованием упомянутых объектов. Потом сохраняете файл с расширением **.vbs** для **VBScript** или **.js** для **JScript**. Мы будем пользоваться **JScript**.

Запуск скриптов - тоже несложная задача. Есть несколько способов. Самый простой - командная строка **DOS** и аналогичная версия **WSH**, **CSCRIPT.EXE**. Эта версия позволяет контролировать исполнение скрипта с помощью параметров командной строки.

Параметры хоста включают или отключают различные опции **WSH** и всегда предваряются двумя слэшами (**//**). Имя скрипта - это всегда имя файла, а параметры скрипта - параметры, передаваемые скрипту. Эти параметры всегда предваряются одним слэшем (**/**). Таблица 1.1. содержит некоторые параметры **CSCRIPT.EXE**.

Таблица 1.1.

Параметр	Описание
//B	Включает пакетный режим
//T	Задаёт таймаут в секундах
//I	Выключает пакетный режим (используется по умолчанию)
//logo	Включает надпись (увы, используется по умолчанию) <i>Microsoft (R) WSH Version 5.0 for Windows Copyright (C) Microsoft Corporation 1996-1997. All rights reserved.</i>
//nologo	Выключает эту надпись.
//H:Cscript или WScript	Делает CSCRIPT.EXE или WSCRIPT.EXE хостом по умолчанию, т.е. ассоциирует их с расширениями скриптов.
//S	Сохраняет установки командной строки для текущего поль-

Если вы не желаете видеть окошек с кнопкой **ОК**, используйте *cscript.exe*, а для вывода сообщений пользуйтесь методом **WScript.Echo**.

Для запуска скриптов с помощью *WScript.exe* имеются три пути:

- просто дважды щелкнуть по файлу или иконке.
- выбрать **Выполнить...** из меню **Пуск** и написать полное имя файла в текстовом поле **Открыть**.
- запустить сам *WSCRIPT.EXE* из того же **Выполнить...**, добавив к нему имя скрипта и любой из возможных параметров.

Попробуем написать какой-нибудь скрипт на **JScript**. По традиции, первая программа на изучаемом языке должна приветствовать мир.

- с помощью текстового редактора создайте файл **Hello.js**
- набейте в него следующий текст:

```
// hello.js
WScript.Echo("Hello, world!");
```

Не забывайте про различие строчных и ПРОПИСНЫХ букв!

- запустите командную строку (или консольный файловый менеджер типа **Far**) и перейдите в директорию, где у вас находится скрипт
- запустите скрипт на выполнение:

```
cscript hello.js
```

- посмотрите вывод программы. Должно получиться что-то вроде

```
Microsoft (R) Windows Script Host Version 5.1 for Windows
Copyright (C) Microsoft Corporation 1996-1999. All rights reserved.

Hello, world!
```

- если версия **WSH** меньше **5.1**, то установите обновление.
- запустите

- пронаблюдайте отличия **wscript** от **cscript**

Синтаксис JScript

Каждый оператор должен отделяться от последующего точкой с запятой (;). Имена объектов и переменных должны набираться с учётом регистра. Текст JS-сценария состоит из операторов программы (не путайте с арифметическими, логическими и т.п. операторами). Каждый оператор, в свою очередь, состоит из ключевых слов языка, операторов (арифметических, логических и т.п.) , идентификаторов, литералов и специальных символов.

Типы данных

JS язык с мягкой типизацией. Тип переменной определяется ее значением, и все необходимые преобразования происходят автоматически.

//Неявное преобразование типов в JS.

```
var a = "Строка";
```

```
var b = 3;
```

```
WScript.Echo(a + b);
```

Тем не менее типы данных в JS разумеется есть, и они состоят из скалярных и комбинированных типов.

Скалярные типы бывают целочисленными (десятичные, восьмеричные и шестнадцатеричные), числа с плавающей точкой, строковые и булевские.

Целочисленные значения, не содержащие префикса 0, считаются десятичными, содержащие префикс 0 и состоящие только из цифр от 0 до 7 – восьмеричные, и префикс 0x – шестнадцатеричные. Числа с плавающей точкой могут быть представлены в обычной и математической (с указанием мантииссы и порядка) нотации. Строковые переменные обрамляются обычно двойными (но допустимо и одинарными) кавычками. Для включения в строковую переменную спецсимволов, используются escape – последовательности, начинающиеся символом “\” – обратный слеш. Это, например, символы двойной кавычки (бу-

дет выглядеть `\"\"`) или управляющие символы (например, `\n` – перевод на новую строку, `\t` – табуляция, `\r` – возврат каретки). Булевы переменные могут содержать значения `true` (истина), `false` (ложь), `null` (значение неопределенно).

К комбинированным типам данных можно отнести данные типа массив, дата/время и др.

//Переменные разных типов данных.

```
var x = 123; // десятичная целочисленная переменная
var y = 034; // восьмеричная целочисленная переменная
var z = 0x1AF; // шестнадцатеричная целочисленная переменная
var x1 = 123.45; // переменная с плавающей точкой в обычной нотации
var y1 = -123.45E-12; // переменная с плавающей точкой в математической нотации
var str1 = "Это строка"; // строковая переменная
var str2 = "А это \n две строки" // тоже строковая переменная
var flg = (3 > 2); // булева переменная
var flg1 = false; // тоже булева переменная
```

Типов констант несколько больше: строка (обрамлена кавычками), число (начинается с цифры) и регулярное выражение (обрамлено в слэши).

В JS допустимо использование как однострочных, так и многострочных строк - комментариев.

Однострочные комментарии начинаются символами `«//»`, весь последующий текст до конца строки в этом случае воспринимается интерпретатором как комментарии. Многострочные комментарии обрамляются символами `«/*»` и `«*/»`.

// однострочный комментарий

```
var a = 123; // снова однострочный комментарий
/* А это уже комментарий на
несколько строк*/
```

Переменные

Так же как и в других языках программирования, переменные в JS – это поименованные области памяти, предназначенные для хранения значений определенного типа. Переменные могут быть как глобальные, так и локальные. Глобальные переменные доступны из любого места сценария. Область действия локальных переменных ограничивается контекстом (обычно функцией), внутри которого эти переменные объявлены.

Переменные в JS можно использовать без их предварительного объявления, но рекомендуется все же их объявлять. Исключение составляют локальные переменные, определенные в функциях.

Переменная в JS объявляется с помощью ключевого слова `var`.

```
// varDemo.js – демонстрация использования переменных
var str="Вы запустили файл скрипта '"+ WScript.FullName+"'";// сложение
строк
WScript.Echo(str);
```

При присвоении переменной константы создается объект, представляющий константу. Например, переменная **str** в примере выше является ссылкой на объект **String**.

При объявлении тип переменной не указывается. Тип переменной определяется только тогда, когда ей присваивается какое-либо значение.

Имена переменных (а также пользовательских функций) должны начинаться с буквы ("A"-"Z", "a"-"z") или с символов "_", "\$" и может состоять только из букв, цифр, а также символов "_", "\$".

Имена переменных и функций чувствительны к регистру. Например, `myvar` и `MyVar` – две разные переменные.

Имена переменных и функций не должны совпадать с зарезервированными ключевыми словами JS, а также с именами встроенных объектов, методов и функций.

Операторы JS

Наличие в JS типов данных предполагает механизмы их обработки. К простейшим из них можно отнести операторы выполнения операций, такие как: присваивание, математические операторы, операторы сравнения, булевские операторы, битовые операторы (табл. 1.2.-1.5.).

Таблица 1.2.

Оператор	Выполняемое действие	Пример	Аналог
операнд1=операнд2	Присваивает операнду1 значение операнда2	i=i;	
операнд1+=операнд2	Присваивает операнду1 значение суммы операнда1 и операнда2	i+=j;	i=i+j
операнд1-=операнд2	Присваивает операнду1 значение разности операнда1 и операнда2	i-=j;	i=i-j
операнд1*=операнд2	Присваивает операнду1 значение произведения операнда1 и операнда2	i*=j;	i=i*j
операнд1/=операнд2	Присваивает операнду1 значение частного деления операнда1 на операнда2	i/=j;	i=i/j;
операнд1%=операнд2	Присваивает операнду1 значение остатка деления операнда1 на операнда2	i%=j;	i=i%j

// Применения операторов присваивания.

```
var x = 121; //десятичная целочисленная переменная
var y = 22;  // десятичная целочисленная переменная
var z = 0;
z = x;
```

```

WScript.Echo(z); //выводит 121
z+=y;
WScript.Echo(z); //выводит 143
z-=y;
WScript.Echo(z); //выводит 121
z *= 2;
WScript.Echo(z); //выводит 242
z /= 4;
WScript.Echo(z); // выводит 60,5
z %= 5;
WScript.Echo(z); // выводит 0,5

```

Таблица 1.3.

Оператор	Выполняемое действие	Пример
-операнд1	Меняет знак операнда1 на противоположный	$k = - i;$
операнд1-операнд2	Вычисляется разность операнда1 и операнда2	$k = j - i;$
операнд1+операнд2	Вычисляется сумма операнда1 и операнда2	$k = j + i;$
операнд1*операнд2	Вычисляется произведение операнда1 и операнда2	$k = j * i;$
операнд1/операнд2	Вычисляется деление операнда1 и операнда2	$k = j / i;$
операнд1%операнд2	Вычисляется остаток от деления операнда1 на операнда2	$k = j \% i;$
операнд1++	Увеличивает значение операнда1 на 1	$j ++;$

операнд1--	Уменьшает значение операнда1 на 1	j --;
-------------------	--------------------------------------	-------

// Применения математических операторов.

var x = - 121; //десятичная целочисленная переменная

var y = 22; // десятичная целочисленная переменная

var z = 0;

z = - x;

WScript.Echo(z); //выводит 121

z = x + y;

WScript.Echo(z); //выводит -99

z = x - y;

WScript.Echo(z); //выводит -143

z = x * y;

WScript.Echo(z); //выводит -2662

z = x / y;

WScript.Echo(z); // выводит -5,5

z = x % y;

WScript.Echo(z); // выводит -11

z ++;

WScript.Echo(z); // выводит -10

z --;

WScript.Echo(z); // выводит -11

Таблица 1.4.

Оператор	Выполняемое действие
операнд1 == операнд2	Возвращает true (истина) если значение операнда1 равно значению операнда2, и false (ложь) в противном случае
операнд1 != операнд2	Возвращает true (истина) если значение операнда1 не равно значению операнда2, и false (ложь) в противном случае

	ном случае
операнд1 > операнд2	Возвращает true (истина) если значение операнда1 больше значению операнда2, и false (ложь) в противном случае
операнд1 >= операнд2	Возвращает true (истина) если значение операнда1 больше или равно значению операнда2, и false (ложь) в противном случае
операнд1 < операнд2	Возвращает true (истина) если значение операнда1 меньше значению операнда2, и false (ложь) в противном случае
операнд1 <= операнд2	Возвращает true (истина) если значение операнда1 меньше или равно значению операнда2, и false (ложь) в противном случае

// Применения операторов сравнения.

var x = - 121; //десятичная целочисленная переменная

var y = 22; // десятичная целочисленная переменная

var z = 0;

z = x == y;

WScript.Echo(z); //выводит false

z = x != y;

WScript.Echo(z); //выводит true

z = x > y;

WScript.Echo(z); //выводит false

z = x >= y;

WScript.Echo(z); //выводит false

z = x < y;

WScript.Echo(z); // выводит true

z = x <= y;

WScript.Echo(z); // выводит true

Таблица 1.5.

Оператор	Выполняемое действие
!операнд1	Выполняет булевскую операцию НЕ над операндом1
операнд1 операнд2	Выполняет булевскую операцию ИЛИ над операндом1 и операндом2
операнд1 && операнд2	Выполняет булевскую операцию И над операндом1 и операндом2

// Применения булевских операторов.

```

var x = false;
var y = true;
var z = 0;
z != x;
WScript.Echo(z); //выводит true
z = x || y;
WScript.Echo(z); //выводит true
z = x && y;
WScript.Echo(z); //выводит false

```

Условные операторы

В JS для изменения порядка вычисления выражений в зависимости от определенных условий применяются три условных оператора. Первый из них однострочный **if** имеет следующий синтаксис:

выражение1 ? выражение2 : выражение3

```

//Однострочный оператор if
var x=38; //десятичная целочисленная переменная
var y=42; // десятичная целочисленная переменная

b=y<x?"да":"нет"; //выводит нет

```

```
WScript.Echo(b)

// другой вариант написания
if (y<x)
    b="да";
else
    b="нет";
WScript.Echo(b) //выводит нет
```

Данный оператор в случае, если значение выражения1 есть true (истина), вычисляет и соответственно возвращает выражение2, в противном случае выражение3.

Однострочный **if** применяется когда выражение2 и выражение3 состоят из одного оператора языка. В случае, когда в зависимости от условия надо выполнять группу действий, необходимо использовать оператор **if...else**. Его синтаксис следующий:

if (условие) группа_операторов1 [else группа_операторов2]

Результат действия данного оператора есть выполнение группы_операторов1, если значение условия есть true (истина), или выполнение группы_операторов2 в противном случае. Если по смыслу не требуется выполнение никаких действий в случае невыполнения условия, то группа **else** может быть опущена. Так же любая из групп_операторов может быть следующим оператором **if...else**, т.е. этот оператор может быть вложенным.

```
//if ... else
if (условие)
{
    .....
    if (условие)
    {.....}
    .....
}
```



```

}
else
{
.....
if (условие)
{.....}
else {.....}
.....
}

```

Также имеется оператор выбор **switch**, который имеет следующий синтаксис:

switch (выражение) {case значение1: группа_операторов1 ... case значениеN : группа_операторовN} [default: группа_операторовN+1]}

Результат действия данного оператора есть выполнение группы_операторовK, если значение выражения совпадает со значениемK, или выполнение группы_операторовN+1, если ни одно из значений от 1 до N не совпадут со значением выражения. Если любая из групп_операторов состоит более чем из одного оператора языка, то необходимо заключение этих операторов в фигурные скобки. Если по смыслу не требуется выполнять никаких действий в случае, когда значение выражения не совпадает ни с одним из значенийK, то группа **default** может быть опущена.

```

// cond demo.js
var weekday=new Date().getDay();// получение дня недели
var str;
switch(weekday)
{
case 0:
    str="седьмой";
    break;
case 1:

```

```

        str="первый";
        break;
    case 2:
        str="второй";
        break;
    case 3:
        str="третий";
        break;
    case 4:
        str="четвёртый";
        break;
    case 5:
        str="пятый";
        break;
    case 6:
        str="шестой";
        break;
    default:
        str="неизвестный";
}
WScript.Echo("Сегодня "+str+" день недели ("+
    ( (weekday==0||weekday==6)?"выходной":"рабочий")+")");

```

Операторы цикла

В JS допускается организация циклов **for(;;){}**, **while() {}**, **do {} while()**.

Оператор цикла по счетчику имеет следующий синтаксис:

for ([инициализация_цикла]; [условие_цикла]; [изменение_счетчика]) группа_операторов

В инициализации_цикла присваиваются начальные значения счетчиков, в условии_цикла проверяется, когда цикл надо прекратить, в измене-

нии_счетчика происходит изменение переменных счетчиков цикла. Если группа_операторов состоит более чем из одного оператора языка, то необходимо заключение этих операторов в фигурные скобки.

Значение счетчика цикла после выхода из цикла сохраняется. В теле цикла можно самостоятельно менять значение счетчика цикла.

Цикл с предусловием имеет синтаксис:

while (условие_цикла) группа_операторов

Условие_цикла проверяется перед каждым проходом цикла. Если условие_цикла истинно, то выполняется группа_операторов. Если условие_цикла изначально ложно, то цикл не выполнится ни разу. Если группа_операторов состоит более чем из одного оператора языка, то необходимо заключение этих операторов в фигурные скобки.

Цикл с постусловием имеет синтаксис:

do группа_операторов while (условие_цикла)

Условие_цикла проверяется после каждого прохода цикла. Если условие_цикла истинно, то опять выполняется группа_операторов. Если условие_цикла изначально ложно, то цикл выполнится один раз. Если группа_операторов состоит более чем из одного оператора языка, то необходимо заключение этих операторов в фигурные скобки.

Операторы break, continue и метки

Данные операторы предназначены для изменения последовательного выполнения тела циклов. Их синтаксис следующий:

break [метка]

continue [метка]

Оператор **break** прерывает выполнение операторов цикла и передает управление либо на первый за циклом оператор языка – в случае, если метка опущена, либо на оператор, следующий за меткой. Оператор **continue** так же прерывает выполнение операторов цикла, но передает управление на заголовок цикла (т.е. начинает следующую итерацию) в случае, если метка опущена, или

на оператор, следующий за меткой. В JS не существует специального оператора безусловного перехода на метку, и поэтому метки используются только в рамках операторов **break** или **continue**.

Контрольные вопросы

1. Что такое WHS?
2. Создание и запуск процедур JScript.
3. Режимы запуска процедур.
4. Параметры запуска процедур.
5. Типы данных JScript.
6. Константы и управляющие символы.
7. Переменные и правила их именования.
8. Типы операторов JScript.
9. Унарные операторы.
10. Бинарные операторы.
11. Управляющие структуры JScript.
12. Операторы цикла.
13. Условные операторы.
14. Оператор выбора.

Варианты заданий

Разработать процедуру расчета суммы n первых членов ряда, в соответствии с индивидуальным заданием. Вычисление факториалов произвести в циклах.

- 1) $(1!)/(1) + (2!)/(2^2) + (3!)/(3^3) + \dots + (n!)/(n^n)$
- 2) $(1!)^2/(2!) + (2!)^2/(4!) + \dots + (n!)^2/(2n)!$
- 3) $A_n = (((n-1)!)^2 / ((2*n)!)) * (2*x)^{2*n}$

$$4) A_n = ((n!)^2 / (2^n n!)) x^n$$

$$5) A_n = (n^2 x^n) / (2^n n + 1)!$$

$$6) A_n = ((-1)^n (2^n n^2 + 1) / (2^n n!)) x^{2^n}$$

$$7) A_n = (1! + 2! + \dots + n!) / (2^n n!)$$

$$8) A_n = (2^n) / (n+1)!$$

$$9) A_n = ((n+1)!)^n / (2! \cdot 4! \cdot \dots \cdot (2^n n!))$$

$$10) (3 \cdot 1!) / 1 + (3^2 \cdot 2!) / 2^2 + \dots + (3^n \cdot n!) / n^n$$

Лабораторная работа № 2
Разработка сценариев на языке JScript

Цель работы:

- 1) Изучение языка JScript.
- 2) Изучение приемов программирования на языке JScript.
- 3) Получение навыков разработки простейших сценариев.

Содержание работы:

- 1) Изучение функций и встроенных объектов JScript.
- 2) Изучение и выполнение всех рассмотренных в руководстве примеров.
- 3) Разработка процедуры решения поставленной задачи.
- 4) Отладка и выполнение процедуры.
- 5) Составление отчета.

Содержание отчета:

- 1) Краткие теоретические сведения.
- 2) Постановка задачи.
- 3) Описание процедуры.
- 4) Руководство пользователю процедуры.
- 5) Листинг процедуры.
- 6) Результаты выполнения процедуры.

Краткие теоретические сведения

Для разработки более сложных процедур язык JScript предоставляет пользователям возможность создания собственных функций и использования встроенных объектов.

Функции

Функции объявляются с помощью ключевого слова **function**. Аргументы и возвращаемое значение функций могут быть ссылками на объекты.

Для определения функций используется следующий синтаксис:

```
function   имя_функции([параметр1],[параметр2],[...,   параметрN])  
{группа_операторов}
```

При вызове функции, входящая в нее группа_операторов выполняются как одно целое. Чтобы функция могла вернуть в вызывающий JS-код какое-либо значение, необходимо включить в группу_операторов оператор **return** со следующим синтаксисом:

```
return возвращаемое_значение
```

Параметр1, ..., параметрN представляют собой список формальных параметров функции. Даже если конкретной функции по смыслу не требуется передача параметров, все равно необходимо указывать пустой список, обрамленный круглыми скобками, как при описании функции, так и при ее вызове. С точки зрения синтаксиса передача реальных параметров в функцию при ее вызове осуществляется по значению. Однако, если в качестве реального параметра функции выступает ссылка на объект, то ее изменение в теле функции ведет к реальному изменению значения объекта.

Для переменных, объявленных в теле функции, область их видимости ограничивается самим телом функции. Если в теле функции встречаются одноименные переменные, объявленные в вызывающем коде, то значения таких переменных внутри тела функции будет определяться соответствующими операторами из тела функции. По выходу из функции значения переменных вернутся к значениям, определенным в вызывающем коде.

```
// funcdemo.js – вычисление расстояния между двумя точками на  
плоскости  
function hypotenuse(x,y)  
{
```

```
        return Math.sqrt(x*x+y*y);
    }
    var x1=2;
    var y1=3;
    var x2=5;
    var y2=6;
    WScript.Echo("Расстояние между (" +x1+", "+y1+") и (" +x2+y2+") равно
    "+hypotenuse(x2-x1,y2-y1));
```

Встроенные объекты

Поддерживается динамическое создание только встроенных объектов. Для этого служит оператор **new**. С помощью его можно создавать объекты, перечисленные в таблице 2.1.

Таблица 2.1.

Объект	Назначение
ActiveXObject	Позволяет создавать объекты OLE и COM
Array	Позволяет создавать массивы (аналогично Java)
Boolean	Объект для типа bool
Date	Объект для представления дат
Enumerator	Объект для обработки OLE -коллекций
Error	Объект для представления ошибки
Function	Объект для указателя на функцию
Number	Объект для типа double
Math	Объект для типа math
String	Объект для типа string
VBAArray	Объект для динамических Visual-Basic массивов

Объект Math

В JS для работы с математическими константами и функциями существует встроенный объект Math. Все свойства и методы объекта Math статичные, поэтому при вызове методов и получении свойств в качестве префикса используется Math, а не конкретный экземпляр объекта (таблица 2.2.).

Таблица 2.2.

Свойства	Описание
Math.E	Возвращает основание натурального логарифма.
Math.LN10	Натуральный логарифм от 10.
Math.LN2	Натуральный логарифм от 2.
Math.LOG10E	Десятичный логарифм от E.
Math.LOG2E	Логарифм по основанию 2 от E.
Math.PI	Число Пи.
Math.SQRT1_2	Квадратный корень от $\frac{1}{2}$.
Math.SQRT2	Квадратный корень от 2.
Методы	Описание
Math.abs(аргумент)	Возвращает модуль аргумента.
Math.acos(аргумент)	Возвращает арккосинус аргумента в радианах.
Math.asin(аргумент)	Возвращает арксинус аргумента в радианах.
Math.atan(аргумент)	Возвращает арктангенс аргумента в радианах.
Math.atan2(аргумент1, аргумент2)	Возвращает арктангенс частного аргумента1 и аргумента2 в радианах.
Math.ceil(аргумент)	Возвращает наименьшее целое, больше или равное чем аргумент.
Math.cos(аргумент)	Возвращает косинус аргумента.
Math.exp(аргумент)	Возвращает E в степени аргумента.
Math.floor(аргумент)	Возвращает наибольшее целое, меньше или равное чем аргумент.

Math.log(аргумент)	Возвращает натуральный логарифм от аргумента. Если аргумент отрицательный возвращается значение NaN.
Math.max(аргумент1, аргумент2)	Возвращает максимальное значение из аргумента1 и аргумента2.
Math.min(аргумент1, аргумент2)	Возвращает минимальное значение из аргумента1 и аргумента2.
Math.pow(аргумент1, аргумент2)	Возвращает значение степени аргумента2 по основанию аргумента1.
Math.random()	Возвращает псевдослучайное число в интервале от 0 до 1.
Math.round(аргумент)	Возвращает аргумент, округленный до ближайшего целого.
Math.sin(аргумент)	Возвращает синус аргумента.
Math.sqrt(аргумент)	Возвращает квадратный корень от аргумента. Если аргумент отрицательный возвращается значение NaN.
Math.tan(аргумент)	Возвращает тангенс аргумента.

Объект String

В JS для того, чтобы оперировать со строковыми переменными и составляющими их символами используется объект String. Для создания экземпляра объекта String используется конструктор со следующим синтаксисом:

имя_переменной = new String([начальное_значение])

JS различает строковые объекты и литералы. Тем не менее, к строковым литералам можно применять все методы класса String, а так же получать свойство length. В этом случае строковый литерал автоматически конвертируется к

объекту String, к нему применяется соответствующий метод, а затем объект String удаляется из памяти.

Ниже в таблице 2.3. приведены свойства и методы объекта String.

Таблица 2.3.

Свойства	Описание
строка.constructor	Определяет функцию, с помощью которой создается прототип строки.
строка.length	Возвращает длину строки в символах.
строка.prototype	Используется для добавления новых свойств в строку.
Методы	Описание
строка.charAt(индекс)	Возвращает символ, стоящий на месте индекса в строке. Если индекс больше длины строки – то возвращается пустая строка.
строка.concat(строка1[, строка2, ... строкаN])	Возвращает строку, объединенную из нескольких строк.
строка.indexOf(подстрока [, индекс])	Ищет слева направо первое вхождение подстроки в строку, и возвращает индекс первого элемента вхождения. Если присутствует параметр индекс – поиск осуществляется, начиная с этого индекса. Если вхождение не обнаружено, то возвращается –1. Метод чувствителен к регистру.
строка.lastIndexOf(подстрока [, индекс])	Ищет слева направо последнего

	вхождение подстроки в строку, и возвращает индекс первого элемента вхождения. Если присутствует параметр индекс – поиск осуществляется, начиная с этого индекса. Если вхождение не обнаружено, то возвращается –1. Метод чувствителен к регистру.
строка.match(регулярное_выражение)	Осуществляет поиск в строке по регулярному_выражению.
строка.search(регулярное_выражение)	Возвращает индекс вхождения при поиске в строке по регулярному_выражению или –1, если нет вхождений.
строка.substr(индекс[,число_символов])	Выделяет подстроку из строки от индекса (включительно), либо до конца строки (если второй параметр отсутствует), либо с длиной, равной числу_символов.
строка.substring(индекс1[,индекс2])	Выделяет подстроку из строки, начиная от индекса1 (включительно), до индекса2 (исключая), либо до конца строки (если второй параметр отсутствует).
строка.toLowerCase()	Преобразует символы строки к нижнему регистру.
строка.toUpperCase()	Преобразует символы строки к верхнему регистру.

Объект *ActiveXObject*

Этот псевдообъект повторяет метод **WScript.CreateObject** (см. [объект WScript](#)). Он служит в качестве обёртки для **COM**-объектов. Синтаксис использования:

```
obj=new ActiveXObject("ProgID"[,"server"]);
```

Здесь:

- **obj** – переменная, в которую будет помещён объект
- **ProgID** – строка с идентификатором **COM** – класса (см. ниже)
- **server** – строка с именем компьютера, на котором нужно создать объект **DCOM**

ProgID

Каждый зарегистрированный класс в **COM** идентифицируется с помощью уникального идентификатора (**CLSID**), либо с помощью строкового идентификатора (**ProgID**). Формат **ProgID** таков:

```
Application.Object.Version
```

Здесь:

- **Application** – название приложения, поддерживающего объект
- **Object** – название объекта
- **Version** – версия объекта (может отсутствовать)

К примеру, **ProgID** объекта **Application** для **Microsoft Word**:

- **97**

```
Word.Application.8
```

- **2000**

```
Word.Application.9
```

- **2002 (XP)**

```
Word.Application.10
```

Существует так называемый версионно-независимый **ProgID**, т.е. **ProgID**, у которого не указана часть **Version**. Рекомендуется пользоваться именно вер-

сионно-независимым **ProgID**, так как он не изменяется при смене версий компонента.

Пример использования:

```
// objdemo.js – вывод текущей даты в Internet Explorer
var iexplorer=new ActiveXObject("InternetExplorer.Application");
iexplorer.Visible=true;// показ окошка
```

COM – объекты: методы и свойства

В предыдущем примере было показано создание и отображение объекта **InternetExplorer.Application**. Для получения информации о поддерживаемых свойствах и методах объекта необходимо прочесть документацию по объекту, либо просмотреть библиотеку **TLB** с помощью специальной программы **OLEView**.

Объект Enumeration

Объект служит для удобной работы с **COM**-коллекциями. Он аналогичен оператору **for each** в **Visual Basic** (таблица 2.4.).

Таблица 2.4.

Методы	Назначение
attend()	возвращает true , если достигнут конец коллекции
item()	возвращает текущий элемент коллекции
moveFirst()	перемещает указатель на первый элемент коллекции
moveNext()	перемещает указатель на следующий элемент коллекции

Контрольные вопросы

1. Определение функции.
2. Передача параметров.
3. Области действия имен.
4. Возврат значений функции.
5. Встроенные объекты JScript.
6. Объект **Math**.
7. Объект **String**.
8. Методы объекта **String**.
9. Объект **ActiveXObject**.
10. Запуск объектов.
11. Объект **Enumeration**.

Варианты заданий

I) Разработать процедуру расчета суммы n первых членов ряда, в соответствии с индивидуальным заданием. Вычисление факториалов произвести с помощью рекурсивных процедур.

1. $(1!)/(1) + (2!)/(2^2) + (3!)/(3^3) + \dots + (n!)/(n^n)$
2. $(1!)^2/(2!) + (2!)^2/(4!) + \dots + (n!)^2/(2n)!$
3. $A_n = (((n-1)!)^2 / ((2*n)!)) * (2*x)^{2*n}$
4. $A_n = (((n!)^2) / (2*n!)) * x^n$
5. $A_n = (n^2 * x^n) / (2*n+1)!$
6. $A_n = ((-1)^n * (2*n^2+1) / (2*n!)) * x^{2*n}$
7. $A_n = (1!+2!+\dots+n!) / (2*n)!$
8. $A_n = (2^n) / (n+1)!$

9. $A_n = ((n+1)!)^n / (2! * 4! * \dots * (2*n)!)$

10. $(3*1!)/1 + (3^2*2!)/2^2 + \dots + (3^n*n!)/n^n$

II) Разработать процедуру расчета суммы n первых членов ряда, в соответствии с индивидуальным заданием. Вычисление математических функций произвести с использованием встроенного объекта **Math**.

1) $\sin(x)/2 + \sin(2*x)/2^2 + \dots + \sin(n*x)/2^n$

2) $(\cos(x) - \cos(2*x))/1 + (\cos(2*x) - \cos(3*x))/2 + \dots + (\cos(n*x) - \cos((n+1)*x))/n$

3) $A_n = 1/(n^p) * \sin(3.1415/n)$

4) $A_n = (1/(n^p)) * (1 - (x * \ln(n))/n)^n$

5) $A_n = (n! * \exp(n))/n^{(n+p)}$

6) $A_n = 1/(\ln^2(\sin(1/n)))$

7) $A_n = (n!)/(n^{\sqrt{n}})$

8) $A_n = (n^{\ln(n)})/(\ln(n))^n$

9) $A_n = \ln(1/n^a) - \ln(\sin(1/n^a))$

10) 2630 $A_n = \ln(n!)/n^a$

III) Разработать процедуру обработки текста, в соответствии с индивидуальным заданием. Все текстовые преобразования произвести с использованием встроенного объекта **String**.

- 1) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте убирает лишние пробелы между словами, оставляя их по одному. В качестве результата вывести исходный и преобразованный тексты.

- 2) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слово максимальной длины. В качестве результата вывести исходный текст, найденное слово и его длину.
- 3) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, оканчиваются заданной буквы. В качестве результата вывести исходный текст, найденные слова и их количество.
- 4) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова заданной длины. В качестве результата вывести исходный текст, найденные слова и их количество.
- 5) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, начинающиеся с гласной буквы. В качестве результата вывести исходный текст, найденные слова и их количество.
- 6) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, оканчивающиеся гласной буквой. В качестве результата вывести исходный текст, найденные слова и их количество.

- 7) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, начинающиеся с заданной буквы. В качестве результата вывести исходный текст, найденные слова и их количество.
- 8) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, в которые входит заданная буква, но она не является первой буквой слова. В качестве результата вывести исходный текст, найденные слова и их количество.
- 9) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, в которые входит заданная буква, но она не является последней буквой слова. В качестве результата вывести исходный текст, найденные слова и их количество.
- 10) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, в которые не входит заданная буква. В качестве результата вывести исходный текст, найденные слова и их количество.

Лабораторная работа № 3

Работа с объектами WSH

Цель работы:

- 1) Изучение объектов **WSH**.
- 2) Изучение приемов программирования с использованием объектов **WSH**.
- 3) Получение навыков разработки сценариев.

Содержание работы:

- 1) Изучение основных свойств и методов объекта **WScript**.
- 2) Выработка навыков использования объекта **WScript**.
- 3) Изучение и выполнение всех рассмотренных в руководстве примеров.
- 4) Разработка процедуры решения поставленной задачи.
- 5) Отладка и выполнение процедуры.
- 6) Составление отчета.

Содержание отчета:

- 1) Краткие теоретические сведения.
- 2) Постановка задачи.
- 3) Описание процедуры.
- 4) Руководство пользователю процедуры.
- 5) Листинг процедуры.
- 6) Результаты выполнения процедуры.

Краткие теоретические сведения

В поставку **WSH** входят 4 объекта: **WScript**, **WSHShell**, **WSHNetwork**, и **FileSystemObject**. Каждый объект, естественно, имеет набор собственных методов и свойств.

Объект WScript

Объект **WScript** позволяет скриптам получать аргументы из командной строки, создавать **COM**-объекты и управлять ими. Основные свойства и методы этого объекта перечислены в таблице 3.1.

Условные обозначения:

- **progid** – строка, программного идентификатора **OLE** – объекта
- **prefix** – приставка, добавленная к именам методов, обрабатывающих события.

Таблица 3.1.

Свойство	Описание
Arguments	Возвращает указатель на список аргументов командной строки
FullName	Возвращает имя исполняемого файла хоста и полный путь к нему (например, C:\Windows\WScript.exe)
Name	Выводит замечательную надпись WSH
Path	Определяет каталог и путь, содержащие Wscript.exe или cscript.exe
ScriptFullName	Возвращает полный путь и имя исполняемого в данный момент скрипта
ScriptName	То же, что и ScriptFullName, но без пути

Version	Возвращает версию установленного WSH
Метод	Описание
CreateObject(progid[, prefix])	Создает объект по его ProgID
ConnectObject(object[, prefix])	Позволяет подключиться к событиям объекта. В качестве параметра принимает объект, к которому надо подключиться и префикс соответствующих событиям процедур, реализованных в скрипте
DisconnectObject(object)	Отключает от объекта, подключенного предыдущим методом
Echo(string)	Выводит текстовую строку (в cscript - в StdOut, в WScript - в виде диалогового окна).
GetObject(file[, progid [, prefix]])	Позволяет получить указатель на объект OLE, созданный из файла или объекта, указанного в параметре file.
Quit([errorcode])	Завершает скрипт и передаёт errorcode запускающему приложению
Sleep(time)	Переводит скрипт в неактивное состояние на время time, указанное в миллисекундах

Как уже говорилось раньше, объект **WScript** уже создан, и можно обращаться к нему непосредственно. Пример использования:

```
// wscript.js – создание объекта Internet Explorer и открытие указанного сайта
if(WScript.Arguments.Count()==1)
{
```

```
var iexplorer=WScript.CreateObject("InternetExplorer.Application");
iexplorer.Visible=true;
iexplorer.Navigate(WScript.Arguments(0));
}else
WScript.Echo("Использование: cscript //nologo "+WScript.ScriptFullName+
" http://www.rsdn.ru/");
```

Работа с Microsoft Word и Microsoft Excel.

Для работы с Microsoft Word или Microsoft Excel необходимо сначала создать соответствующий объект:

```
var wdApp = new ActiveXObject("Word.Application");
var xlApp = new ActiveXObject("Excel.Application");
```

или

```
var wdApp = WScript.CreateObject("Word.Application");
var xlApp = WScript.CreateObject("Excel.Application");
```

Затем нужно установить свойство **Visible** созданного объекта в **True**, чтобы приложение стало видимым:

```
wdApp.Visible = true;
xlApp.Visible = true;
```

Чтобы добавить новый, пустой документ к коллекции открытых документов, нужно использовать метод **Add**.

Синтаксис

expression.Add(Template, NewTemplate, DocumentType, Visible)

expression Обязательный. Выражение, которое возвращает объект **Documents**.

Template Необязательный. Variant. Имя шаблона, которое используется для нового документа. Если данный параметр опускается, то используется шаблон Normal.

NewTemplate Необязательный. Variant. **True** для открытия документа в качестве шаблона. Значение по умолчанию – **False**.

DocumentType Необязательный. Variant. Может быть одной из следующих **WdNewDocumentType** констант: **wdNewBlankDocument**, **wdNewEmailMessage**, **wdNewFrameset**, или **wdNewWebPage**. Значение по умолчанию - **wdNewBlankDocument**.

Visible Необязательный. Variant. **True** для открытия документа в видимом окне. Если это значение равно **False**, свойство **Visible** устанавливается в **False**. Значение по умолчанию – **True**.

```
wdApp.Documents.Add()
```

Для создания новой рабочей книги в Excel так же используется метод **Add**.

Синтаксис

expression.**Add**(*Template*)

expression Обязательный. Выражение, которое возвращает объект **Workbooks**.

Template Необязательный. Variant. Определяет, как создается новая рабочая книга. Если этот аргумент – строка, определяющая имя существующего файла Microsoft Excel, создается новая рабочая книга, использующая существующий файл в качестве шаблона. Если этот аргумент – константа, новая книга содержит единственную страницу определенного типа. Можно использовать одну из следующих **XlWBATemplate** констант: **xlWBATChart**, **xlWBATExcel4IntlMacroSheet**, **xlWBATExcel4MacroSheet**, или **xlWBATWorksheet**. Если этот аргумент опускается, Microsoft Excel создает новую рабочую книгу с пустыми листами.

```
xlApp.Workbooks.Add()
```

Чтобы открыть существующий документ или рабочую книгу, используется метод **Open** объекта **Documents** (для Microsoft Word) или **Workbooks** (для

Microsoft Excel). Данные методы могут принимать множество параметров, однако в самом простом случае достаточно передать полный путь и имя файла в качестве параметра:

```
wdApp.Documents.Open("C:\\temp\\mydoc.doc")  
xlApp.Workbooks.Open("C:\\temp\\salary.xls")
```

Контрольные вопросы

1. Объекты **WSH**.
2. Объект **WScript**.
3. Свойства объекта **WScript**.
4. Методы объекта **WScript**.
5. Передача параметров в процедуру JScript.
6. Создание и освобождение объектов.
7. Вывод данных из процедур Jscript.
8. Особенности вывода для различных режимов запуска процедуры.
9. Работа с Microsoft Word.
10. Работа с Microsoft Excel.

Варианты заданий

I) Разработать процедуру расчета суммы n первых членов ряда, в соответствии с индивидуальным заданием. Организовать ввод исходных данных в процедуру с помощью параметров командной строки при вызове процедуры.

1. $(1!)/(1) + (2!)/(2^2) + (3!)/(3^3) + \dots + (n!)/(n^n)$
2. $(1!)^2/(2!) + (2!)^2/(4!) + \dots + (n!)^2/(2n)!$
3. $A_n = (((n-1)!)^2 / ((2*n)!)) * (2*x)^{2*n}$
4. $A_n = (((n!)^2) / (2*n!)) * x^n$
5. $A_n = (n^2 * x^n) / (2*n+1)!$

$$6. A_n = ((-1)^n * (2 * n^2 + 1) / (2 * n!)) * x^{2 * n}$$

$$7. A_n = (1! + 2! + \dots + n!) / (2 * n!)$$

$$8. A_n = (2^n) / (n + 1)!$$

$$9. A_n = ((n + 1)!)^n / (2! * 4! * \dots * (2 * n!))$$

$$10. (3 * 1!) / 1 + (3^2 * 2!) / 2^2 + \dots + (3^n * n!) / n^n$$

II) Разработать процедуру расчета суммы n первых членов ряда, в соответствии с индивидуальным заданием. Организовать ввод исходных данных в процедуру с помощью параметров командной строки при вызове процедуры.

$$1) \sin(x)/2 + \sin(2 * x)/2^2 + \dots + \sin(n * x)/2^n$$

$$2) (\cos(x) - \cos(2 * x)) / 1 + (\cos(2 * x) - \cos(3 * x)) / 2 + \dots + (\cos(n * x) - \cos((n + 1) * x)) / n$$

$$3) A_n = 1 / (n^p) * \sin(3.1415 / n)$$

$$4) A_n = (1 / (n^p)) * (1 - (x * \ln(n)) / n)^n$$

$$5) A_n = (n! * \exp(n)) / n^{(n+p)}$$

$$6) A_n = 1 / (\ln^2(\sin(1/n)))$$

$$7) A_n = (n!) / (n^{\sqrt{n}})$$

$$8) A_n = (n^{\ln(n)}) / (\ln(n))^n$$

$$9) A_n = \ln(1/n^a) - \ln(\sin(1/n^a))$$

$$10) 2630 \quad A_n = \ln(n!) / n^a$$

III) Разработать процедуру обработки текста, в соответствии с индивидуальным заданием. Организовать ввод исходных данных в процедуру с помощью параметров командной строки при вызове процедуры.

- 1) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более

NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте убирает лишние пробелы между словами, оставляя их по одному. В качестве результата вывести исходный и преобразованный тексты.

2) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слово максимальной длины. В качестве результата вывести исходный текст, найденное слово и его длину.

3) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, оканчиваются заданной буквы. В качестве результата вывести исходный текст, найденные слова и их количество.

4) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова заданной длины. В качестве результата вывести исходный текст, найденные слова и их количество.

5) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, начинающиеся с гласной буквы. В качестве результата вывести исходный текст, найденные слова и их количество.

6) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более

NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, оканчивающиеся гласной буквой. В качестве результата вывести исходный текст, найденные слова и их количество.

7) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, начинающиеся с заданной буквы. В качестве результата вывести исходный текст, найденные слова и их количество.

8) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, в которые входит заданная буква, но она не является первой буквой слова. В качестве результата вывести исходный текст, найденные слова и их количество.

9) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, в которые входит заданная буква, но она не является последней буквой слова. В качестве результата вывести исходный текст, найденные слова и их количество.

10) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, в которые не входит заданная буква. В качестве результата вывести исходный текст, найденные слова и их количество.

Лабораторная работа № 4

Отладка сценариев WSH

Цель работы:

- 1) Изучение отладчика **WSH**.
- 2) Изучение приемов отладки сценариев **WSH**.
- 3) Получение навыков отладки сценариев.

Содержание работы:

- 1) Изучение основных средств и методов отладки **JScript**.
- 2) Выработка навыков отладки сценариев **JScript**..
- 3) Изучение и выполнение всех рассмотренных в руководстве примеров.
- 4) Разработка процедуры решения поставленной задачи.
- 5) Отладка и выполнение процедуры.
- 6) Составление отчета.

Содержание отчета:

- 1) Краткие теоретические сведения.
- 2) Постановка задачи.
- 3) Описание процедуры.
- 4) Листинг процедуры.
- 5) Результаты трассировки выполнения и отладки процедуры.

Краткие теоретические сведения

Исправив все синтаксические ошибки, можно приступить к проверке функциональности сценария. В большинстве случаев, кроме простейших, исполнение сценария, скорее всего не даст ожидаемых результатов. Иногда полезно выводить сообщения с промежуточными результатами, помещая для этого в исход-

ный текст, соответствующие операторы. Однако такая методика пригодна только для отладки простых сценариев. Если при исполнении сценариев возникают ошибки периода выполнения или получаются неожиданные результаты, для трассировки процедуры и ее переменных следует использовать отладчик - Microsoft Script Debugger.

WSH поддерживает команды для запуска отладчика и исполнения сценариев под его управлением. Однако исполнение отладчика в разных версиях WSH несколько отличается.

В WSH 1 исполнение оператора Stop, при использовании языка VBScript, или оператора debugger в случае JScript, приводит к автоматическому вызову отладчика.

Однако, если поместить в сценарий WSH 2, написанный на VBScript, оператор Stop или оператор debugger (в случае JScript), как это делалось в WSH 1, отладчик не среагирует.

Для работы с отладчиком в WSH 2 надо вызывать сценарий с явным указанием параметра отладки //D. В результате такого вызова, при исполнении первого же оператора Stop (в VBScript) или debugger (в JScript) запускается отладчик, и ему передается управление. После чего возможно пошаговое исполнение сценария под управлением отладчика.

При желании использовать в процедуре операторы Stop и debugger можно вызвать отладчик, запустив сценарий с параметром //X.

При этом WSH 2 запускает отладчик и передает ему управление, встретив первую же исполняемую команду процедуры.

Для проверки сценариев служат несколько панелей инструментов и команд меню Microsoft Script Debugger (рис. 4.1.).

Основные команды отладчика

Run (запуск) исполняет операторы сценария до точки прерывания в сценарии.

Stop Debugging (остановить отладку) прерывает исполнение сценария и завершает процесс отладки.

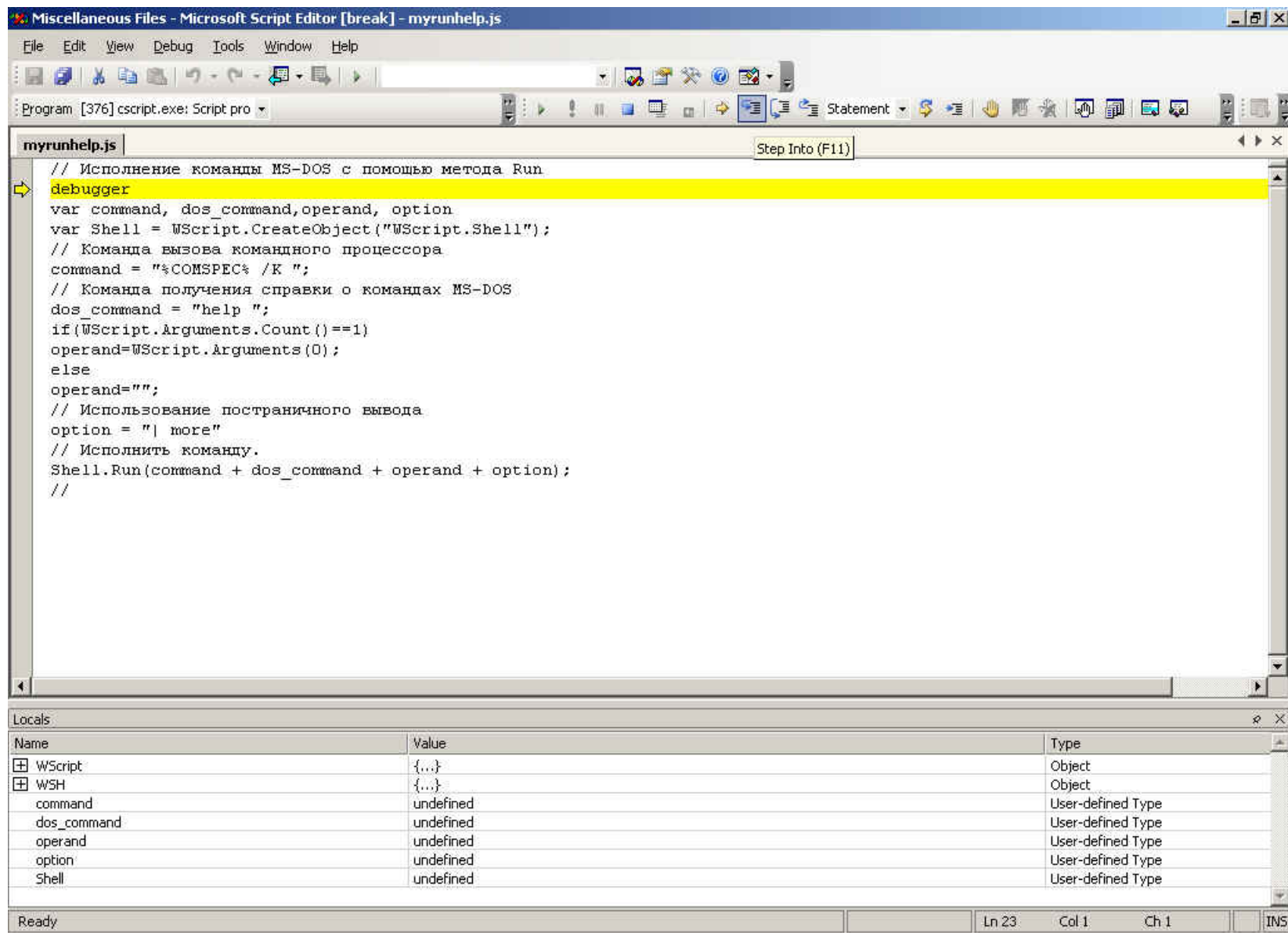


Рис. 4.1.

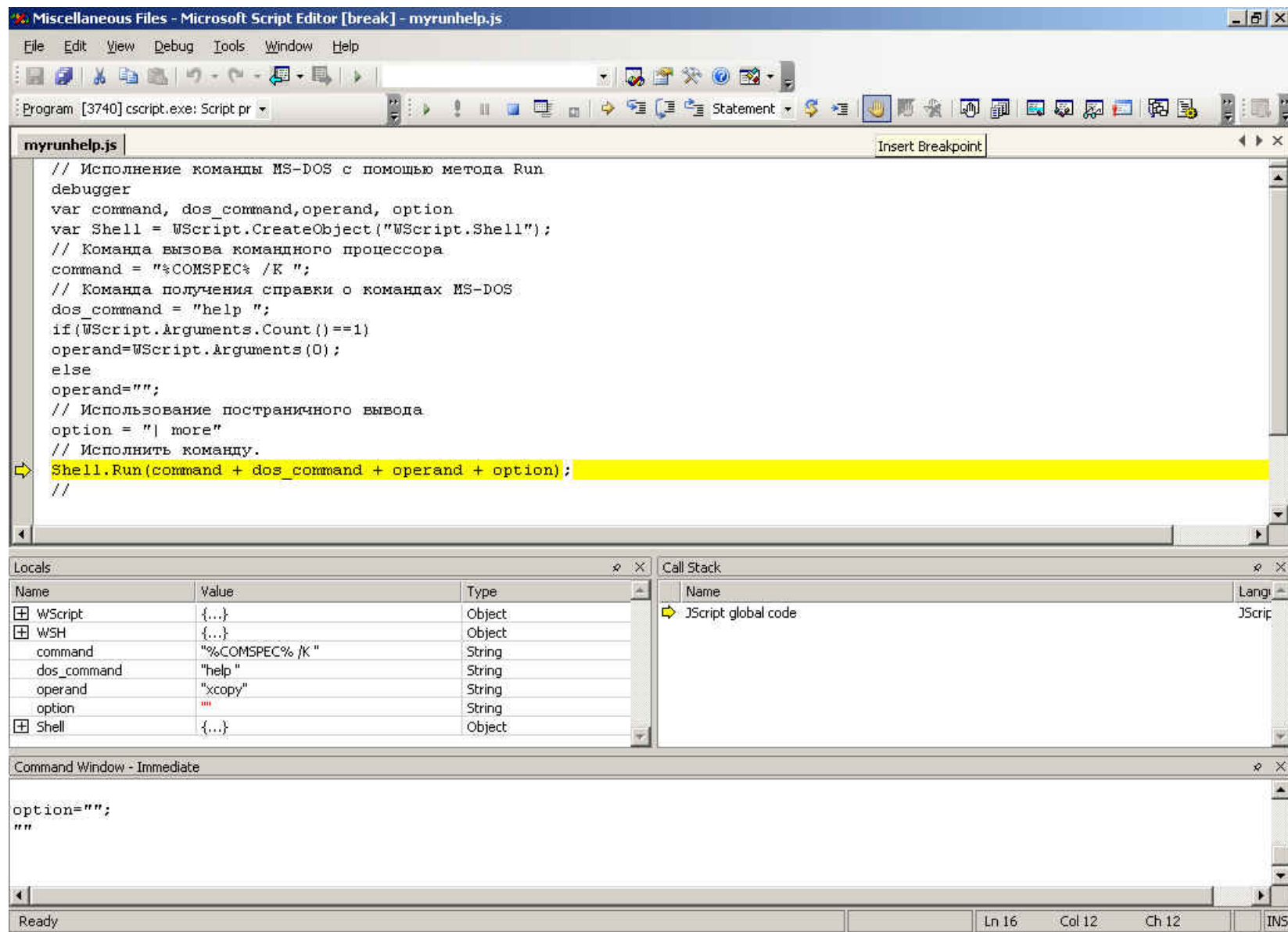


Рис. 4.2.

Break At Next Statement (остановиться при исполнении следующего оператора) гарантирует, что исполнение сценария не продолжится до конца без остановок. Эта команда нужна для ручного управления исполнением сценария из окна отладчика Running Documents. Если исполнение сценария приостановлено при выводе сообщения, Break At Next Statement остановит исполнение сценария на следующем операторе, как только пользователь закроет окно сообщения.

Step Into (шаг вперед) исполняет следующий оператор сценария.

Step Over (шаг над) заставляет отладчик выполнить процедуру или функцию полностью, прежде чем управление вернется к вызывающему модулю. Если текущий оператор не является вызовом процедуры или функции, отладчик работает в режиме Step Into.

Step Out (шаг до) исполняет все операторы процедуры до команды, возвращающей управление.

Toggle Breakpoint (установить контрольную точку прерывания) выбирает в окне отладчика исполняемый оператор и устанавливает или удаляет в строке выбранного оператора точку прерывания. Строки с точками прерывания помечены точками около левой границы окна отладчика. При достижении точки прерывания отладчик останавливает исполнение сценария.

Clear All Breakpoints (удалить все точки прерывания) удаляет все точки прерывания, установленные в сценарии.

Кнопка **Command Window (Immediate)** открывает окно, в котором можно напрямую вводить и исполнять команды. При нажатии кнопки **Call Stack** открывается окно со списком активных процедур в стеке. Когда сценарий имеет исключительно линейную структуру (без вызовов процедур или функций), стек остается пустым.

Пошаговое исполнение сценария

При отладке полезен режим Step Into (режим пошагового исполнения). Чтобы исполнить следующую команду сценария, щелкните кнопку Step Into. Сле-

дующий исполнимый оператор помечен стрелочкой около левой границы окна отладчика (рис. 4.2.).

Если сценарий содержит проверенные процедуры и функции, для них бессмысленно использовать режим пошагового исполнения. Вместо этого можно щелкнуть кнопку Step Over. Если следующий исполняемый оператор содержит вызов процедуры/функции, то вся процедура/функция выполняется безостановочно. Исполнение останавливается лишь после возврата управления. Команда Step Over, применяемая для исполнения других операторов, не содержащих вызовов процедур или функций, работает аналогично режиму Step Into.

Если при исполнении процедуры/функции отладчик встречает точку прерывания, то завершить исполнение всех операторов процедуры или функции позволяет кнопка Step Out. Отладчик остановится на первом же операторе после вызова процедуры/ функции.

Использование точек прерывания

Точки прерывания (контрольные точки) позволяют прерывать исполнение сценария на определенных операторах. Установить точку прерывания можно, выбрав, оператор в окне отладчика и нажав кнопку Breakpoint (рис. 4.2.). По возобновлении исполнения процедуры, отладчик вновь остановит ее, встретив строку с установленной точкой прерывания.

Просмотр стека вызовов

Окно Call Stack (рис. 4.2.) со списком всех активных процедур и функций открывает команда Call Stack из меню View или кнопка Call Stack на панели инструментов.

Вывод значений промежуточных результатов

Окно Command позволяет изучать текущие значения переменных (или свойств), пока исполнение программы остановлено. Значение переменной на JScript позволяет вывести в окне Command оператор, например: WScriptEcho(operand). Исполняя его, отладчик выводит окно сообщения со значением переменной operand.

Чтобы установить значение переменной, введите в окне команд оператор, присваивающий переменной нужное значение:

Рассмотренные методы облегчают отладку сценариев WSH, написанных на VBScript или JScript

Контрольные вопросы

1. Способы запуска сценария для отладки.
2. Назначение оператора отладки.
3. Различные варианты пошагового исполнения процедур.
4. Контрольные точки.
5. Отличие команд **Step Into** и **Step Over**.
6. Трассировка выполнения функций сценария.
7. Трассировка выполнения операторов сценария.
8. Выполнение промежуточных вычислений при отладке сценария.
9. Отличие команд **Step Over** и **Step Out**.
10. Различия в запуске отладчика в разных версиях WSH.

Варианты заданий

I) Разработать, отладить и построить трассу выполнения процедуры расчета суммы n первых членов ряда, в соответствии с индивидуальным заданием. Организовать ввод исходных данных в процедуру с помощью параметров командной строки при вызове процедуры.

1. $(1!)/(1) + (2!)/(2^2) + (3!)/(3^3) + \dots + (n!)/(n^n)$
2. $(1!)^2/(2!) + (2!)^2/(4!) + \dots + (n!)^2/(2n)!$
3. $A_n = (((n-1)!)^2 / ((2*n)!)) * (2*x)^{2*n}$
4. $A_n = (((n!)^2) / (2*n!)) * x^n$

$$5. A_n = (n^2 * x^n) / (2 * n + 1)!$$

$$6. A_n = ((-1)^n * (2 * n^2 + 1) / (2 * n)!) * x^{2 * n}$$

$$7. A_n = (1! + 2! + \dots + n!) / (2 * n)!$$

$$8. A_n = (2^n) / (n + 1)!$$

$$9. A_n = ((n + 1)!)^n / (2! * 4! * \dots * (2 * n)!)^n$$

$$10. (3 * 1!) / 1 + (3^2 * 2!) / 2^2 + \dots + (3^n * n!) / n^n$$

II) Разработать, отладить и построить трассу выполнения процедуры расчета суммы n первых членов ряда, в соответствии с индивидуальным заданием. Организовать ввод исходных данных в процедуру с помощью параметров командной строки при вызове процедуры.

$$1) \sin(x)/2 + \sin(2 * x)/2^2 + \dots + \sin(n * x)/2^n$$

$$2) (\cos(x) - \cos(2 * x)) / 1 + (\cos(2 * x) - \cos(3 * x)) / 2 + \dots + (\cos(n * x) - \cos((n + 1) * x)) / n$$

$$3) A_n = 1 / (n^p) * \sin(3.1415 / n)$$

$$4) A_n = (1 / (n^p)) (1 - (x * \ln(n)) / n)^n$$

$$5) A_n = (n! * \exp(n)) / n^{(n+p)}$$

$$6) A_n = 1 / (\ln^2(\sin(1/n)))$$

$$7) A_n = (n!) / (n^{\sqrt{n}})$$

$$8) A_n = (n^{\ln(n)}) / (\ln(n))^n$$

$$9) A_n = \ln(1/n^a) - \ln(\sin(1/n^a))$$

$$10) 2630 \quad A_n = \ln(n!) / n^a$$

III) Разработать, отладить и построить трассу выполнения процедуры обработки текста, в соответствии с индивидуальным заданием. Организовать ввод

исходных данных в процедуру с помощью параметров командной строки при вызове процедуры.

- 1) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте убирает лишние пробелы между словами, оставляя их по одному. В качестве результата вывести исходный и преобразованный тексты.
- 2) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слово максимальной длины. В качестве результата вывести исходный текст, найденное слово и его длину.
- 3) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, оканчиваются заданной буквы. В качестве результата вывести исходный текст, найденные слова и их количество.
- 4) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова заданной длины. В качестве результата вывести исходный текст, найденные слова и их количество.
- 5) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, начинающиеся с глас-

ной буквы. В качестве результата вывести исходный текст, найденные слова и их количество.

- 6) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, оканчивающиеся гласной буквой. В качестве результата вывести исходный текст, найденные слова и их количество.
- 7) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, начинающиеся с заданной буквы. В качестве результата вывести исходный текст, найденные слова и их количество.
- 8) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, в которые входит заданная буква, но она не является первой буквой слова. В качестве результата вывести исходный текст, найденные слова и их количество.
- 9) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, в которые входит заданная буква, но она не является последней буквой слова. В качестве результата вывести исходный текст, найденные слова и их количество.
- 10) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, в которые не входит

заданная буква. В качестве результата вывести исходный текст, найденные слова и их количество.

Лабораторная работа № 5

Работа со средой

Цель работы:

- 1) Изучение объектов **WSH**.
- 2) Изучение приемов программирования с использованием объектов **WSH**.
- 3) Получение навыков разработки сценариев.

Содержание работы:

- 1) Изучение основных свойств и методов объектов **WshShell**, **WshShortcut**.
- 2) Выработка навыков использования объектов **WshShell**, **WshShortcut**.
- 3) Изучение и выполнение всех рассмотренных в руководстве примеров.
- 4) Разработка процедуры решения поставленной задачи.
- 5) Отладка и выполнение процедуры.
- 6) Составление отчета.

Содержание отчета:

- 1) Краткие теоретические сведения.
- 2) Постановка задачи.
- 3) Описание процедуры.
- 4) Руководство пользователю процедуры.
- 5) Листинг процедуры.
- 6) Результаты выполнения процедуры.

Краткие теоретические сведения

Объект **WshShell** позволяет скриптам запускать и конфигурировать другие приложения. Он также дает возможность общаться с пользователем, изменять *Реестр* и находить папки. Некоторые его свойства и методы перечислены в таблице 5.1.

Таблица 5.1.

Свойство	Описание
CurrentDirectory	текущая (активная) директория
SpecialFolders	Возвращает полный путь к специальным папкам типа меню Пуск . Вот список доступных папок: AllUsersDesktop , AllUsersStartMenu , AllUsersPrograms , AllUsersStartup , Desktop , Favorites , Fonts , MyDocuments , NetHood , PrintHood , Programs , Recent , SendTo , StartMenu , Startup , Templates .
Метод	Описание
AppActivate(title)	Активирует окно приложения
CreateShortcut(path)	Создает ярлык path и возвращает объект WshShortcut
Environment(envType)	Возвращает объект-коллекцию WshEnvironment , который позволяет получить значения переменных среды.

Exec(strCommand)	Передаёт strCommand в командную строку
ExpandEnvironmentStrings (string)	Раскрывает строку string , содержащую переменные среды
LogEvent(evType, strMessage[, strTarget])	Позволяет записать сообщение strMessage в event log Windows NT или W2K или файл WSH.log в случае Windows 9x
Popup(strText[, nSecondsToWait[, strTitle[, nType]]])	Ещё один способ вывести текст strText в окне на nSecondsToWait секунд
RegDelete(regKey)	Удаляет ключ или запись реестра regKey
RegRead(regKey)	Возвращает значение ключа или записи реестра по его имени
RegWrite (regKey, value[, regType])	Создает новый ключ или запись реестра или записывает новое значение существующих.
Run(strCommand[, intWindowStyle[, bWaitOnReturn]])	Запускает команду strCommand . intWindowStyle позволяет управлять положением и поведением окон. Если bWaitOnReturn==true , то ожидает окончания выполнения запущенного приложения.
SendKeys(strKeys)	Передаёт активному приложению

	strKeys так, как будто она была введены с клавиатуры
--	---

Специальные обозначения:

- **envType** может принимать два значения: "**System**" (только для **Windows NT/2000/XP**) – системные значения переменных окружения; "**Process**" – переменные окружения процесса
- **evType** определяет тип (важность) сообщения (таблица 5.2.).

Таблица 5.2.

	Тип	Описание
0	SUCCESS	Удача
1	ERROR	Ошибка
2	WARNING	Предупреждение
4	INFORMATION	Информация
8	AUDIT_SUCCESS	Отслеживание: удача
16	AUDIT_FAILURE	Отслеживание: неудача

- **regType** определяет тип значения (Таблица 5.3.).

Таблица 5.3.

regType	Значение
REG_SZ	строка
REG_EXPAND_SZ	строка
REG_DWORD	целое число
REG_BINARY	строка

Пример:

```
// wshExec.js – запуск процесса и ожидание его завершения
```

```
var of;  
var shell=WScript.CreateObject("WScript.Shell");  
    WScript.Echo("Сейчас я запущу блокнот и подожду, пока вы его не  
закроете.");  
    shell.Run(shell.ExpandEnvironmentStrings("%windir%\\notepad.exe"),1,true);
```

Метод **Run** позволяет исполнять не только приложения Windows, но и команды MS-DOS. Например, можно использовать такую команду на JScript:

```
Shell.Run("Edit.com");
```

Этот оператор открывает окно редактора MS-DOS. Если добавить к команде имя файла документа, документ будет загружен в редактор. Поскольку путь не указан, Windows ищет файл программы в каталогах, заданных переменной окружения PATH.

Однако, если вместо Edit.com записать другую команду MS-DOS, например, Dir:

```
Shell.Run("Dir C:\");
```

этот оператор не будет работать, так как программы с именем Dir.com или Dir.exe не существуют. Методу Run требуется исполнимый файл (.exe, .com, .bat или .pif). В Windows 95/98/Me внутренние команды MS-DOS, такие как Dir, Copy, Type и т.д., хранятся в командном процессоре Command.com (в отличие от внешних команд типа Edit.com). Command.com находится в папке Windows. В Windows NT/2000/XP для обработки команд MS-DOS служит командный процессор Cmd.exe. Если Windows NT/2000/XP установлен в папке Windows, то Cmd.exe находится в Window\system32. Чтобы исполнить внутреннюю команду MS-DOS, необходимо запустить командный процессор и передать ему эту внутреннюю команду (команда выполняется в окне командной строки).

Команду вывода содержимого папки Windows можно записать следующим образом:

Shell.Run("cmd.exe /K Dir C:\Windows");

Однако с этой командой связаны две проблемы. Во-первых, следует сделать этот оператор независимым от папки, куда установлена Windows. Для этого путь C:\Windows можно заменить переменной окружения %WINDIR%. Во-вторых, чтобы гарантировать работу сценария на всех платформах, следует вместо явного указания имени командного процессора поместить в команду переменную окружения %COMSPEC%. Эта переменная окружения содержит имя и путь командного процессора. В результате в командной строке всегда будет нужный командный процессор.

Полный список параметров, поддерживаемых cmd.exe можно получить, набрав в командной строке

cmd.exe /?.

Следующий пример выводит справку о командах MS DOS, для вывода справки о конкретной команде MS DOS необходимо после самой команды Help указать имя интересующей команды.

```
// Исполнение команды MS-DOS с помощью метода Run
var command, dos_command, option
var Shell = WScript.CreateObject("WScript.Shell");
// Команда вызова командного процессора
command = "%COMSPEC% /K ";
// Команда получения справки о командах MS-DOS
dos_command = "help ";
// Использование постраничного вывода
option = "| more"
// Исполнить команду.
Shell.Run(command + dos_command + option);
//
```

Объект **WshShortcut** представляет собой ярлык. Свойства и методы объекта представлены в таблице 5.4.

Таблица 5.4.

Свойство	Назначение
Arguments	коллекция аргументов запуска
Description	описание ярлыка
FullName	полное имя
HotKey	«горячая» клавиша
IconLocation	путь к иконке
TargetPath	целевой путь ярлыка
WindowStyle	стиль окна (см.ниже)
WorkingDirectory	каталог для запуска
Метод	Назначение
Save()	сохраняет ярлык на диск

Пример

```
//wshShk.js - создание ярлыка Блокнота на Рабочем столе
var WSHShell=WScript.CreateObject("WScript.Shell");

//путь к Рабочему столу
var DesktopPath = WSHShell.SpecialFolders("Desktop");

//создаем ярлык на Рабочем столе
var MyShortcut=WSHShell.CreateShortcut(DesktopPath+"\\Ярлык на
блокнот.lnk");

//Задаем свойства объекта-ярлыка и сохраняем их
```

```
MyShortcut.TargetPath=WSHShell.ExpandEnvironmentStrings("%windir%\notepad.exe");  
MyShortcut.WorkingDirectory=WSHShell.ExpandEnvironmentStrings("%windir%")  
;  
MyShortcut.WindowStyle=4;  
MyShortcut.IconLocation=WSHShell.ExpandEnvironmentStrings("%windir%\notepad.exe,0");  
MyShortcut.Save();  
  
WScript.Echo("Теперь на Рабочем столе есть ярлык для Notepad.");
```

Контрольные вопросы

11. Объект **WshShell**.
12. Свойства объекта **WshShell**.
13. Методы объекта **WshShell**.
14. Работа с командной строкой.
15. Работа с переменными окружения.
16. Запуск оконных приложений.
17. Запуск и ожидание завершения процесса.
18. Объект **WshShortcut**.
19. Свойства объекта **WshShortcut**.
20. Методы объекта **WshShortcut**.
21. Создание ярлыков.

Варианты заданий

I) Разработать процедуру вывода справки о командах MS DOS с возможностью перенаправления справочной информации в файл. Организовать ввод имени интересующей команды, пути и имени результирующего файла в процедуру с помощью параметров командной строки при вызове процедуры.

II) Разработать процедуру запуска Блокнота для создания новой или редактирования существующей процедуры с возможностью ее последующего запуска в оконном режиме или режиме командной строки. Организовать ввод пути и имени редактируемой процедуры и режим ее запуска с помощью параметров командной строки.

III) Разработать процедуру, выполняющую заданную в индивидуальном задании последовательность операций над файлами (каталогами) с помощью команд MS. Организовать ввод всех исходных данных в процедуру с помощью параметров командной строки при вызове процедуры. Предусмотреть возможность перенаправления результатов выполнения процедуры в файл.

1. Копирование файлов:

- 1.1. Создать исходный каталог;
- 1.2. С помощью Блокнота создать исходный файл;
- 1.3. Создать результирующий каталог;
- 1.4. Скопировать исходный файл в результирующий каталог;
- 1.5. Запустить Блокнот для редактирования файла в результирующем каталоге
- 1.6. Сравнить файлы в исходном и результирующем каталогах;

2. Копирование каталогов:

- 2.1. Создать исходный каталог;
- 2.2. С помощью Блокнота создать исходный файл;
- 2.3. Скопировать исходный каталог в результирующий каталог;
- 2.4. Изменить атрибуты файла в результирующем каталоге;
- 2.5. Вывести оглавление исходного и результирующего каталогов;
- 2.6. Вывести содержимое файла результирующего каталога;

3. Переименование каталогов:

- 3.1. Создать исходный каталог;
- 3.2. С помощью Блокнота создать исходный файл;
- 3.3. Переименовать исходный каталог в результирующий каталог;

- 3.4.Изменить атрибуты файла в результирующем каталоге;
- 3.5.Вывести оглавление исходного и результирующего каталогов;
- 3.6.Вывести содержимое файла результирующего каталога;
- 4. Переименование файлов:
 - 4.1. Создать исходный каталог;
 - 4.2.С помощью Блокнота создать исходный файл;
 - 4.3.Создать результирующий каталог;
 - 4.4.Скопировать исходный файл в результирующий каталог;
 - 4.5.Переименовать файл в результирующем каталоге;
 - 4.6.Запустить Блокнот для редактирования файла в результирующем каталоге
 - 4.7.Сравнить файлы в исходном и результирующем каталогах;
- 5. Перемещение файлов:
 - 5.1. Создать исходный каталог;
 - 5.2.С помощью Блокнота создать исходный файл;
 - 5.3.Создать результирующий каталог;
 - 5.4.Переместить исходный файл в результирующий каталог;
 - 5.5.Запустить Блокнот для редактирования файла в результирующем каталоге
 - 5.6.Вывести содержимое файла результирующего каталога;
- 6. Перемещение каталогов:
 - 6.1. Создать исходный каталог;
 - 6.2.С помощью Блокнота создать исходный файл;
 - 6.3.Переместить исходный каталог в результирующий каталог;
 - 6.4.Переименовать файл в результирующем каталоге;
 - 6.5.Запустить Блокнот для редактирования файла в результирующем каталоге;
 - 6.6.Вывести оглавление результирующего каталога;
- 7. Копирование дерева каталогов:
 - 7.1. Создать исходное дерево каталогов;

- 7.2.С помощью Блокнота создать исходный файл;
- 7.3.Скопировать исходное дерево каталогов в результирующий каталог;
- 7.4.Изменить атрибуты файла в результирующем каталоге;
- 7.5.Вывести оглавление исходного и результирующего каталогов;
- 7.6.Вывести содержимое файла результирующего каталога;
- 8. Копирование файлов:
 - 8.1. Создать исходный каталог;
 - 8.2.С помощью MS Word создать исходный файл;
 - 8.3.Создать результирующий каталог;
 - 8.4.Скопировать исходный файл в результирующий каталог;
 - 8.5.Запустить MS Word для редактирования файла в результирующем каталоге;
 - 8.6.Вывести оглавление исходного и результирующего каталогов;
- 9. Переименование файлов:
 - 9.1. Создать исходный каталог;
 - 9.2.С помощью MS Excel создать исходный файл;
 - 9.3.Создать результирующий каталог;
 - 9.4.Скопировать исходный файл в результирующий каталог;
 - 9.5.Переименовать файл в результирующем каталоге;
 - 9.6.Запустить MS Excel для редактирования файла в результирующем каталоге
 - 9.7.Сравнить файлы в исходном и результирующем каталогах;
- 10.Замена файлов:
 - 10.1. Создать исходный каталог;
 - 10.2. С помощью MS Excel создать исходный файл;
 - 10.3. Создать результирующий каталог;
 - 10.4. Скопировать исходный файл в результирующий каталог;
 - 10.5. Запустить MS Excel для редактирования файла в результирующем каталоге;
 - 10.6. Заменить файлы исходного каталога файлами результирующего каталога;

щего каталога;

10.7. Сравнить файлы в исходном и результирующем каталогах;

Лабораторная работа № 6,7

Работа с файловой системой

Цель работы:

- 1) Изучение объектов **WSH** работы с файловой системой.
- 2) Изучение приемов программирования с использованием объектов работы с файловой системой.
- 3) Получение навыков разработки сценариев.

Содержание работы:

- 1) Изучение основных свойств и методов объектов **FileSystemObject, Drive, Folder, File** и **TextStream**.
- 2) Выработка навыков использования объектов **FileSystemObject, Drive, Folder, File, TextStream**.
- 3) Изучение и выполнение всех рассмотренных в руководстве примеров.
- 4) Разработка процедуры решения поставленной задачи.
- 5) Отладка и выполнение процедуры.
- 6) Составление отчета.

Содержание отчета:

- 1) Краткие теоретические сведения.
- 2) Постановка задачи.
- 3) Описание процедуры.
- 4) Руководство пользователю процедуры.
- 5) Листинг процедуры.
- 6) Результаты выполнения процедуры.

Краткие теоретические сведения

Объект **FileSystemObject** нужен для дисковых операций. Он позволяет читать, писать, удалять файлы и создавать каталоги. **ProgId** этого компонента – **Scripting.FileSystemObject**. **FileSystemObject** имеет методы, описанные в таблице 6.1.

Специальные обозначения:

- [...] – необязательный параметр
- **src, dst** – источник и приёмник. Если особо не оговаривается, то это строки с путями
- **path** – путь к файлу или каталогу
- **name** – имя файла или каталога
- **spec** – идентификатор системного каталога. (0 – Windows, 1 – System, 2 – Temp)
- **unicode** – определяет режим текста (**TristateTrue** – *Unicode*, **TristateFalse** – *ASCII*, **TristateUseDefault** – по умолчанию)
- **iomode** – определяет режим открытия (1 – для чтения, 2 – для записи, 8 – для добавления в конец файла)

Таблица 6.1.

Свойство	Описание
Drives	Список дисков, доступных на данной машине, включая сетевые
Метод	Описание
BuildPath(path,name)	Возвращает путь, состоящий из двух аргументов (просто вставляет между ними разделитель)
CopyFile(src,dst[,ovw])	Копирует файл src в dst . Если ovw=true , то заменяет dst

CopyFolder(src,dst[,ovw])	Копирует каталог src со всеми подкаталогами в dst .
CreateFolder(path)	Создаёт каталог name
CreateTextFile(FileName[,ovw[,Unicode]])	Создаёт текстовый файл FileName и связывает с ним текстовый поток. Если unicode=true , то создаётся файл Unicode .
DeleteFile(name[,force])	Удаляет файл name . Если force==true , то не обращает внимания на атрибут ReadOnly
DeleteFolder(path[,force])	Удаляет каталог src со всеми подкаталогами name
DriveExists(path)	Возвращает true , если диск path доступен
FileExists(path)	Возвращает true , если файл path существует
FolderExists(path)	Возвращает true , если каталог path существует
GetAbsolutePathName(path)	Нормализует путь path
GetBaseName(path)	Возвращает имя файла (без расширения)
GetDrive(drive)	Возвращает объект Drive от указанного диска
GetDriveName(path)	Возвращает имя диска
GetExtensionName(path)	Возвращает расширение файла
GetFile(path)	Возвращает объект File , расположенный по указанному пути.
GetFileName(path)	Возвращает путь к файлу (без расширения)

GetFolder(path)	Возвращает объект Folder , расположенный по указанному пути.
GetParentFolderName(path)	Возвращает имя последнего каталога в пути
GetSpecialFolder(сpec)	Возвращает путь к системным каталогам
GetTempName()	Возвращает случайно сгенерированное имя файла (для временных файлов)
MoveFile(src,dst)	Перемещает файл src в dst
MoveFolder(src,dst)	Перемещает каталог src в dst
OpenTextFile(path[,iomode[,create[,unicode]]])	Открывает текстовый файл для чтения/записи (определяется io-mode)

Пример использования:

```
// fsoDirs.js - получение системных каталогов
var fso=WScript.CreateObject("Scripting.FileSystemObject");
var s="Специальные каталоги:";
s+="\nWindows "+fso.GetSpecialFolder(0);
s+="\nSystem "+fso.GetSpecialFolder(1);
s+="\nTemp "+fso.GetSpecialFolder(2);
WScript.Echo(s);
```

Наряду с **FileSystemObject** используются так же объекты **Drive**, **Folder**, **File** и **TextStream**. Свойства и методы их приведены в таблицах 6.2- 6.5 соответственно.

Объект **Drive** (таблица 6.2.) представляет собой логический или сетевой диск операционной системы. Получить этот объект можно с помощью метода **FileSystemObject.GetDrive**.

Таблица 6.2.

Свойство	Описание
AvailableSpace	Доступное пространство (с учётом квоты) на диске, в байтах
DriveLetter	Буква диска
DriveType	Тип диска (см. ниже)
FileSystem	Тип файловой системы (строка)
FreeSpace	Доступное пространство (без учёта квоты) на диске, в байтах
IsReady	true , если устройство готово
Path	Путь к диску (буква диска и двоеточие)
RootFolder	Объект Folder , представляющий корневой каталог
SerialNumber	Строка, содержащая серийный номер диска
ShareName	Имя сетевого ресурса, «шары»
TotalSize	Общий объем диска, в байтах
VolumeName	Метка тома

Тип диска может быть следующим:

- **0** – неизвестен
- **1** – отключаемый
- **2** – жёсткий
- **3** – сетевой
- **4** – **CD-ROM**
- **5** – **RAM** диск

Пример использования:

```

// fsoEnumDrv.js – вывод списка подключенных дисков
var fso=WScript.CreateObject("Scripting.FileSystemObject");
var e=new Enumerator(fso.Drives);
var s="";
var n=""
for(;!e.atEnd();e.moveNext())
{
var x=e.item();
s=s+x.DriveLetter;
s+=" - ";
switch(x.DriveType)
{
case 1: n="Отключаемый";break;
case 2: n="Жёсткий";break;
case 3: n="Сетевой";break;
case 4: n="CD-ROM";break;
case 5: n="РАМ диск";break;
default:
n="Unknown";
}
s+=n+", ";
if(x.DriveType==3)
n=x.ShareName;
else if (x.IsReady)
n=x.VolumeName;
else
n="[Drive not ready]";
s+="\""+n+"\"\\n";
}
WScript.Echo(s);

```


Объект **Folder** (таблица 6.3.) представляет собой каталог операционной системы.

Таблица 6.3.

Свойство	Описание
Attributes	Атрибуты каталога
DateCreated	Дата создания каталога
DateLastAccessed	Дата последнего доступа к каталогу
DateLastModified	Дата последней модификации каталога
Drive	Буква диска, на котором находится каталог
Files	Список файлов, находящихся в каталоге
IsRootFolder	Если это корневой каталог диска, то равно true
Name	Имя файла
ParentFolder	Объект Folder родительского каталога
Path	Путь к каталогу
ShortName	Короткое (MS-DOS - совместимое) имя каталога
ShortPath	Короткий (MS-DOS - совместимый) путь к каталогу
Size	Размер каталога
SubFolders	Список подкаталогов
Type	Строка с информацией о типе каталога (из реестра)
Метод	Описание
Copy(dst[, ovw])	Копирует каталог в dst . Если ovw=true , то заменяет dst
Delete(force)	Удаляет каталог. Если force==true , то не обращает внимания на атрибут ReadOnly
Move(dest)	Перемещает каталог в dst

Пример использования:

```
// fsoCreateText.js - создание файла с деревом каталогов
var of;
function ProcessDirectory(dir,prefix)
{
var foldPrefix=prefix+"-";
    prefix+="|";
var newPrefix=prefix+" ";

var fc = new Enumerator(dir.SubFolders);
while(!fc.atEnd())
{
    of.WriteLine(foldPrefix+fc.item().Name);
    ProcessDirectory(fc.item(),newPrefix);
    fc.moveNext();
}
fc = new Enumerator(dir.Files);
while(!fc.atEnd())
{
    of.WriteLine(prefix+fc.item().Name);
    fc.moveNext();
}
}

if(WScript.Arguments.Count()==2)
{
var fso=WScript.CreateObject("Scripting.FileSystemObject");
var of=fso.CreateTextFile(WScript.Arguments(0),true);
    ProcessDirectory(fso.GetFolder(WScript.Arguments(1)), "")
}
```

```

of.Close()
}else
    WScript.Echo("Использование: cscript //nologo "+WScript.ScriptFullName+
" tree.txt c:\\windows\\temp");

```

Объект **File** (таблица 6.4.) представляет собой файл операционной системы.

Таблица 6.4.

Свойство	Описание
Attributes	Атрибуты файла
DateCreated	Дата создания файла
DateLastAccessed	Дата последнего доступа к файлу
DateLastModified	Дата последней модификации файла
Drive	Буква диска, на котором находится файл
Name	Имя файла
ParentFolder	Объект Folder родительского каталога
Path	Путь к файлу
ShortName	Короткое (MS-DOS - совместимое) имя файла
ShortPath	Короткий (MS-DOS - совместимый) путь к файлу
Size	Размер файла
Type	Строка с информацией о типе файла документа (из реестра)
Метод	Описание
Copy(dst[, ovw])	Копирует файл в dst . Если ovw=true , то заменяет dst
Delete(force)	Удаляет файл. Если force==true , то не обращает внимания на атрибут ReadOnly

Move(dest)	Перемещает файл в dst
OpenAsTextStream([iomode[, unicode]])	Открывает файл как текстовый для чтения/записи (определяется iomode)

Объект **TextStream** (таблица 6.5.) позволяет читать и записывать текстовую информацию. Он может быть использован не только для файлов, но и для любых объектов, предоставляющих интерфейс **Stream**. Внутри его хранится файловый указатель или текущая позиция чтения/записи.

Таблица 6.5.

Свойство	Описание
AtEndOfLine	равно true , если указатель находится в конце строки
AtEndOfStream	равно true , если указатель находится в конце потока
Column	Номер колонки (столбца) указателя
Line	Номер строки указателя
Метод	Описание
Close()	Закрывает поток
Read(count)	Читает count символов из потока и возвращает в виде строки. Передвигает указатель.
ReadAll()	Возвращает весь текстовый поток как строку
ReadLine()	Считывает одну строку текста из потока и возвращает строку. Передвигает указатель.
Skip(count)	Передвигает указатель на count символов
SkipLine	Передвигает указатель на начало следующей строки текста
Write(str)	Записывает строку str в поток в позиции указателя
WriteBlankLines(count)	Записывает count пустых текстовых строк в

	ПОТОК
WriteLine(str)	Записывает строку str в поток в позиции указателя и добавляет символ конца строки

Если методам **Write** и **WriteLine** объекта **TextStream** не-**Unicode**-файла передать строку с **Unicode**-символами, то они выдают ошибку.

// **fsoCreateText.js** - создание файла с таблицей умножения

```

if(WScript.Arguments.Count()==1)
{
var   fso=WScript.CreateObject("Scripting.FileSystemObject");
var   tf=fso.CreateTextFile(WScript.Arguments(0),true);
      tf.WriteLine("Таблица Пифагора:");
var   i;
      tf.Write(" |");
      for(i=1;i<10;i++)
      {
          tf.Write(" "+i+"|");
      }
      tf.WriteLine("");
      tf.WriteLine("-+--+--+--+--+--+--+--+--+");
var   j;
      for(i=1;i<10;i++)
      {
          tf.Write(i);
          for(j=1;j<10;j++)
          {
              tf.Write("|");
              if(i*j<10)

```

```

        tf.Write(" ");
        tf.Write(i*j);
    }
    tf.WriteLine("|");
}
tf.WriteLine("-+-+---+---+---+---+---+---+---+---+");
tf.Close();
} else
    WScript.Echo("Использование: cscript //nologo "+WScript.ScriptFullName+
" pifagor.txt");

```

Контрольные вопросы

22. Объекты работы с файловой системой .
23. Свойства объекта FileSystemObject.
24. Методы объекта FileSystemObject.
25. Свойства объекта **Drive**.
26. Методы объекта **Drive**.
27. Свойства объекта **Folder**.
28. Методы объекта **Folder**.
29. Свойства объекта **File**.
30. Методы объекта **File**.
31. Свойства объекта **TextStream**.
32. Методы объекта **TextStream**.
33. Копирование файлов.
34. Поиск файлов.
35. Поиск каталогов.
36. Переименование файлов.
37. Переименование каталогов.
38. Создание файлов.

- 39.Создание каталогов.
- 40.Удаление файлов.
- 41.Удаление каталогов.
- 42.Обработка текстовых файлов.

Варианты заданий

I) Разработать процедуру, выполняющую заданную в индивидуальном задании последовательность операций над файлами (каталогами) с помощью различных объектов работы с файловой системой. Организовать ввод всех исходных данных в процедуру с помощью параметров командной строки при вызове процедуры. Предусмотреть возможность перенаправления результатов выполнения процедуры в файл.

1. Копирование файлов:

- 1.1. Создать исходный каталог;
- 1.2.С помощью Блокнота создать исходный файл;
- 1.3.Создать результирующий каталог;
- 1.4.Скопировать исходный файл в результирующий каталог;
- 1.5.Запустить Блокнот для редактирования файла в результирующем каталоге
- 1.6.Сравнить файлы в исходном и результирующем каталогах;

2. Копирование каталогов:

- 2.1. Создать исходный каталог;
- 2.2.С помощью Блокнота создать исходный файл;
- 2.3.Скопировать исходный каталог в результирующий каталог;
- 2.4.Изменить атрибуты файла в результирующем каталоге;
- 2.5.Вывести оглавление исходного и результирующего каталогов;
- 2.6.Вывести содержимое файла результирующего каталога;

3. Переименование каталогов:

- 3.1. Создать исходный каталог;
- 3.2. С помощью Блокнота создать исходный файл;
- 3.3. Переименовать исходный каталог в результирующий каталог;
- 3.4. Изменить атрибуты файла в результирующем каталоге;
- 3.5. Вывести оглавление исходного и результирующего каталогов;
- 3.6. Вывести содержимое файла результирующего каталога;
4. Переименование файлов:
 - 4.1. Создать исходный каталог;
 - 4.2. С помощью Блокнота создать исходный файл;
 - 4.3. Создать результирующий каталог;
 - 4.4. Скопировать исходный файл в результирующий каталог;
 - 4.5. Переименовать файл в результирующем каталоге;
 - 4.6. Запустить Блокнот для редактирования файла в результирующем каталоге
 - 4.7. Сравнить файлы в исходном и результирующем каталогах;
5. Перемещение файлов:
 - 5.1. Создать исходный каталог;
 - 5.2. С помощью Блокнота создать исходный файл;
 - 5.3. Создать результирующий каталог;
 - 5.4. Переместить исходный файл в результирующий каталог;
 - 5.5. Запустить Блокнот для редактирования файла в результирующем каталоге
 - 5.6. Вывести содержимое файла результирующего каталога;
6. Перемещение каталогов:
 - 6.1. Создать исходный каталог;
 - 6.2. С помощью Блокнота создать исходный файл;
 - 6.3. Переместить исходный каталог в результирующий каталог;
 - 6.4. Переименовать файл в результирующем каталоге;
 - 6.5. Запустить Блокнот для редактирования файла в результирующем каталоге;

- 6.6. Вывести оглавление результирующего каталога;
- 7. Копирование дерева каталогов:
 - 7.1. Создать исходное дерево каталогов;
 - 7.2. С помощью Блокнота создать исходный файл;
 - 7.3. Скопировать исходное дерево каталогов в результирующий каталог;
 - 7.4. Изменить атрибуты файла в результирующем каталоге;
 - 7.5. Вывести оглавление исходного и результирующего каталогов;
 - 7.6. Вывести содержимое файла результирующего каталога;
- 8. Копирование файлов:
 - 8.1. Создать исходный каталог;
 - 8.2. С помощью MS Word создать исходный файл;
 - 8.3. Создать результирующий каталог;
 - 8.4. Скопировать исходный файл в результирующий каталог;
 - 8.5. Запустить MS Word для редактирования файла в результирующем каталоге;
 - 8.6. Вывести оглавление исходного и результирующего каталогов;
- 9. Переименование файлов:
 - 9.1. Создать исходный каталог;
 - 9.2. С помощью MS Excel создать исходный файл;
 - 9.3. Создать результирующий каталог;
 - 9.4. Скопировать исходный файл в результирующий каталог;
 - 9.5. Переименовать файл в результирующем каталоге;
 - 9.6. Запустить MS Excel для редактирования файла в результирующем каталоге
 - 9.7. Сравнить файлы в исходном и результирующем каталогах;
- 10. Замена файлов:
 - 10.1. Создать исходный каталог;
 - 10.2. С помощью MS Excel создать исходный файл;
 - 10.3. Создать результирующий каталог;
 - 10.4. Скопировать исходный файл в результирующий каталог;

- 10.5. Запустить MS Excel для редактирования файла в результирующем каталоге;
- 10.6. Заменить файлы исходного каталога файлами результирующего каталога;
- 10.7. Сравнить файлы в исходном и результирующем каталогах;

II) Разработать процедуру обработки текстового файла, в соответствии с индивидуальным заданием. Организовать ввод исходных данных в процедуру с помощью параметров командной строки при вызове процедуры. Создание исходного текста на русском языке в текстовом файле произвести в процедуре путем вызова Блокнота. Предусмотреть возможность перенаправления результатов выполнения процедуры в файл.

- 1) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте убирает лишние пробелы между словами, оставляя их по одному. В качестве результата вывести исходный и преобразованный тексты.
- 2) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слово максимальной длины. В качестве результата вывести исходный текст, найденное слово и его длину.
- 3) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, оканчиваются заданной буквы. В качестве результата вывести исходный текст, найденные слова и их количество.

- 4) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова заданной длины. В качестве результата вывести исходный текст, найденные слова и их количество.
- 5) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, начинающиеся с гласной буквы. В качестве результата вывести исходный текст, найденные слова и их количество.
- 6) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, оканчивающиеся гласной буквой. В качестве результата вывести исходный текст, найденные слова и их количество.
- 7) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, начинающиеся с заданной буквы. В качестве результата вывести исходный текст, найденные слова и их количество.
- 8) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, в которые входит заданная буква, но она не является первой буквой слова. В качестве результата вывести исходный текст, найденные слова и их количество.

- 9) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, в которые входит заданная буква, но она не является последней буквой слова. В качестве результата вывести исходный текст, найденные слова и их количество.
- 10) Задан исходный текст на русском языке. Длина текста - не более NL символов, длина строки - не более NS символов, длина слова – не более NW символов. Исходный текст должен заканчиваться точкой. Составить процедуру, которая в заданном тексте находит слова, в которые не входит заданная буква. В качестве результата вывести исходный текст, найденные слова и их количество.

Лабораторная работа № 8

Оценка производительности вычислительной системы

Цель работы:

- 1) Изучение средств оценки производительности вычислительных систем.
- 2) Получение навыков оценки производительности вычислительных систем.

Содержание работы:

- 1) Изучение средств оценки производительности Windows.
- 2) Выработка навыков использования Системного монитора.
- 3) Изучение и выполнение всех рассмотренных в руководстве примеров.
- 4) Выполнение индивидуального задания.
- 5) Составление отчета.

Содержание отчета:

- 1) Краткие теоретические сведения.
- 2) Постановка задачи.
- 3) Описание процесса выполнения задания.
- 4) Результаты выполнения задания.

Windows XP Professional предоставляет два инструментальных средства для контроля за использованием ресурсов: оснастку Системный монитор (System Monitor) и оснастку Журнал и оповещения производительности (Performance Logs and Alerts). Они включены в консоль Производительность (Performance). Оснастка Системный монитор (System Monitor) предназначена для отслеживания использования ресурсов и производительности сети. Оснастка Журнал и оповещения производительности (Performance Logs and Alerts) применяется для сбора данных о быстродействии локального или удаленного компьютера.

Для доступа к консоли Производительность (Performance) необходимо щелкнуть мышью Пуск (Start), затем - Панель управления (Control Panel), значок Администрирование (Administrative Tools) и дважды щелкнуть ярлык Производительность (Performance). Консоль Производительность (Performance) содержит оснастки Системный монитор (System Monitor) и Журнал и оповещения производительности (Performance Logs and Alerts) (рис. 8.1.).

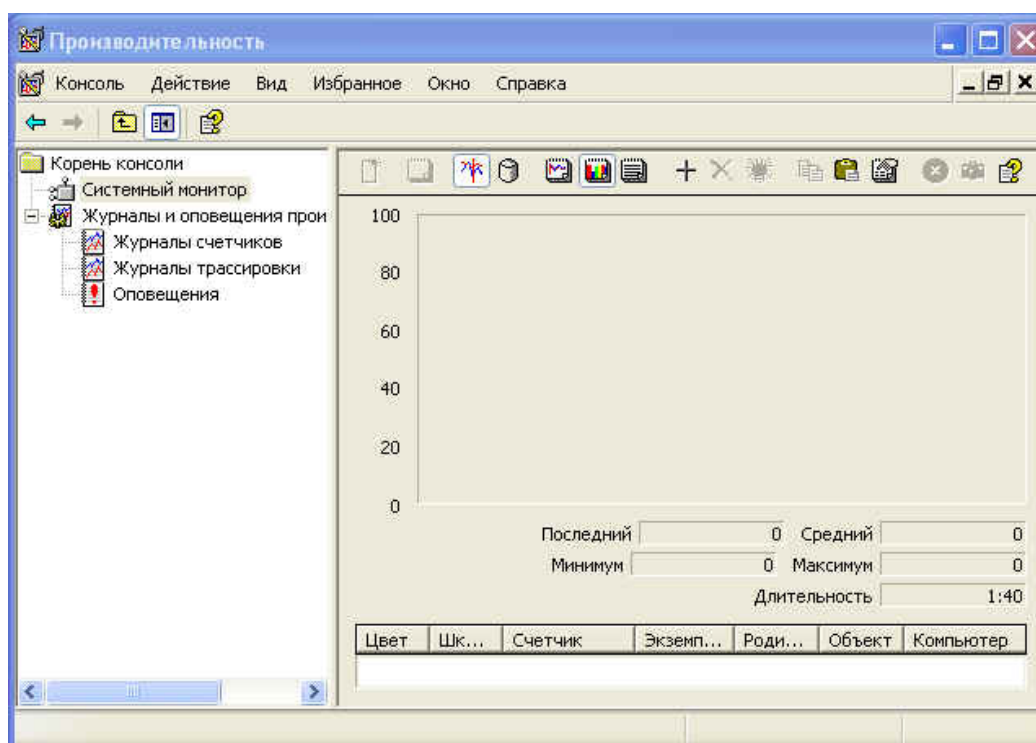


Рис. 8.1.

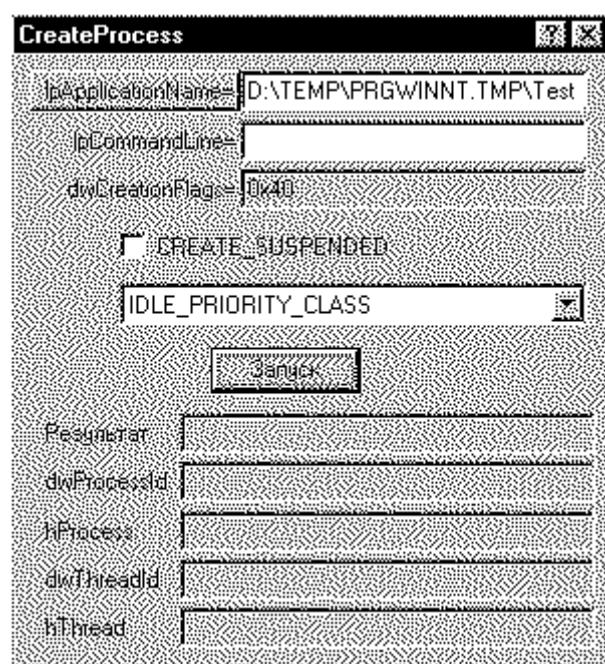


Рис. 8.2.

Пока в него не добавлено никаких счётчиков.

Запустите два тестовых процесса с низким классом приоритета (рис. 8.2.).

Теперь добавьте счётчики процессорного времени, нажав на кнопку .

В поле **Объект** выделите строку **Поток**, в поле **Счётчик** - строку **% загрузки процессора**, а в списке **Вхождения** выделите главные потоки обоих тестовых процессов (TestProc ...) (рис. 8.3.).

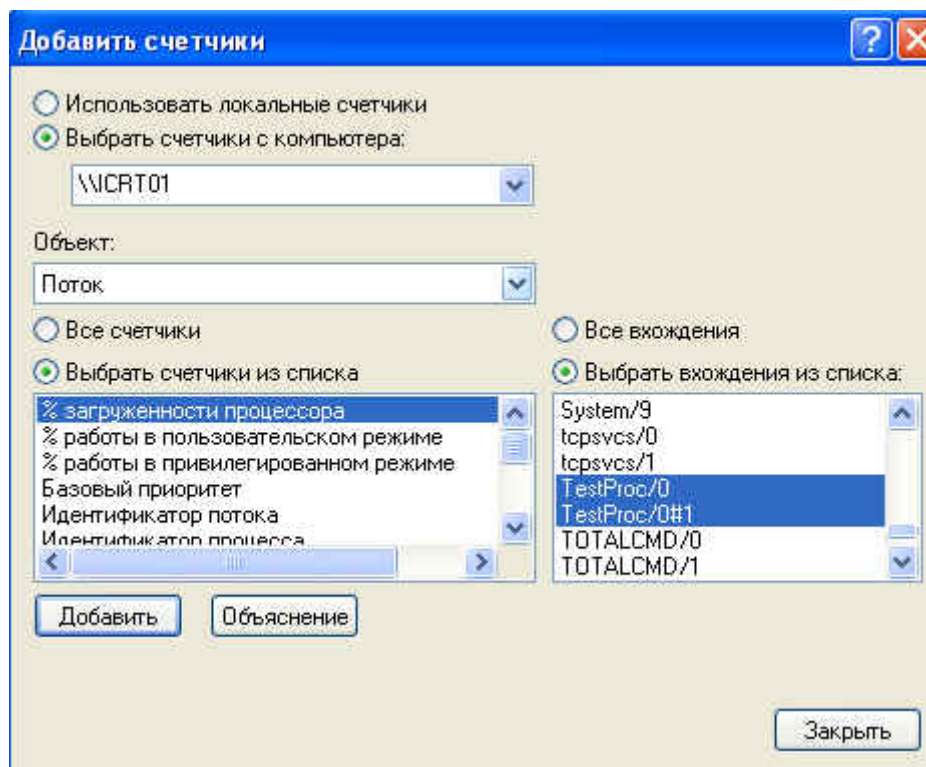


Рис. 8.3.

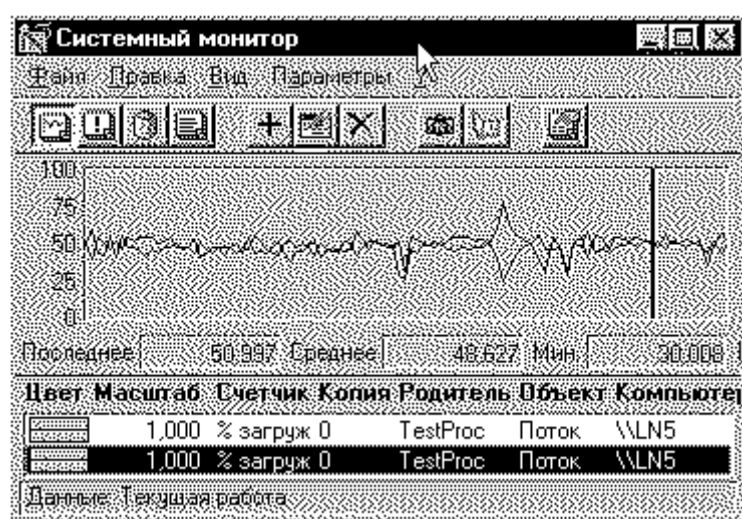




Рис. 8.4.

Нажмите на кнопку . Монитор немедленно начнёт замеры производительности, что и отобразится на графике (рис. 8.4.):

Если вы добавили слишком много счётчиков, то есть возможность удалить ненужные. В таблице в низу окна выделите ненужный счётчик и нажмите клавишу **Del**

Типы процессорного времени

Сейчас **Системный монитор** показывает процент процессорного времени, занимаемый потоком. Процессор в это время может находиться в двух режимах: **Kernel mode**, т.е. привилегированный режим ядра, и **User Mode**, т.е. режим прикладной программы. **Системный монитор** может показывать процент времени работы процессора в этих режимах по отдельности. Для того, чтобы просмотреть, каково отношение времени, отводимого ядру ОС к времени, занимаемому непосредственно самой программой, добавим счётчики в **Системный монитор**.

Нажмите на кнопку 

В поле **Объект** выделите строчку **Поток**, в поле **Счётчик** - две строчки **% работы...**, а в списке **Вхождений** выделите потоки тестовых процессов (TestProc ==>...) (рис. 8.5.)

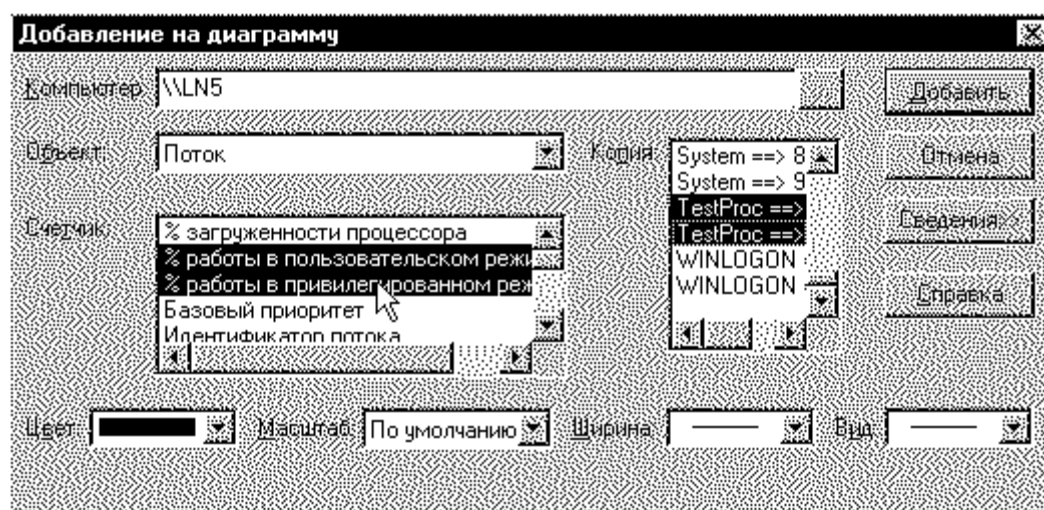


Рис. 8.5.

Нажмите на кнопку 

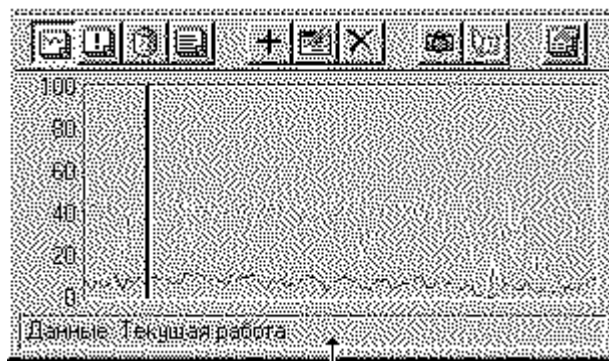



Рис. 8.6.

Видно (рис. 8.6.), что в привилегированном режиме процессор проводит гораздо больше времени, чем в пользовательском. Это обусловлено тем, что окно программы постоянно перерисовывается, а перерисовка выполняется модулем GDI системы, который и занимает столько времени.

Переключения контекста

Как было уже сказано, размер кванта (интервала процессорного времени), определяет быстродействие системы. При большом кванте накладные расходы на реализацию параллельного исполнения невелики, но при большом количестве потоков "время отклика" потока достаточно велико, то есть поток достаточно редко получает в своё распоряжение процессор. При маленьком кванте операционная система может поддерживать большее количество процессов и потоков, но накладные расходы на параллелизм уже значительно больше. Откуда эти "накладные расходы" и в чём они выражаются?

Дело в том, что при передаче процессора другому потоку происходит так называемое переключение контекста, т.е. сохранение данных текущего потока и загрузка данных для потока, которому и выделяется процессорное время. Не вдаваясь в особенности различных процессоров, можно сказать, что это сравнительно долгая операция. Оценка накладных расходов может выражаться в количестве переключений контекста потока в единицу времени. Оценим количество переключений для главных потоков обоих процессов:

Нажмите на кнопку 

В поле **Объект** выделите строчку **Поток**, в поле **Счётчик** - две строки **% Контекстных переключений/сек**, в поле **Масштаб** - **0,1** (или **0,01**), а в списке **Вхождений** выделите потоки тестовых процессов (TestProc ==>...) (рис. 8.7.)

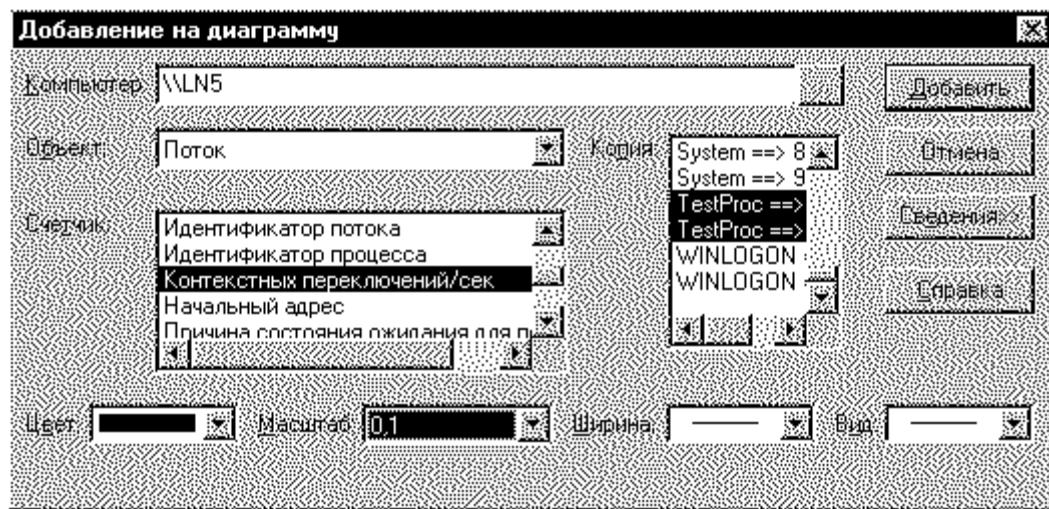


Рис. 8.7.

Нажмите на кнопку

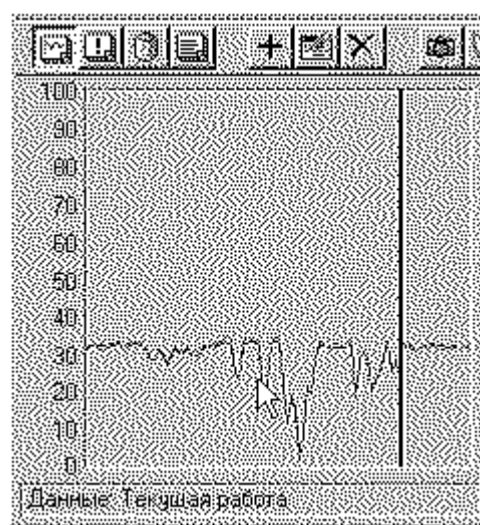


Рис. 8.8.

В среднем у потоков происходит около 300 переключений в секунду.

Мониторинг потоков с разными приоритетами

До сих пор мы наблюдали за работой потоков с одинаковыми приоритетами и классами приоритета процесса. За исключением незначительных отклонений, показания **Системного монитора** для этих потоков не отличались.

Запустим в обоих процессах ещё по потоку (рис. 8.9.):

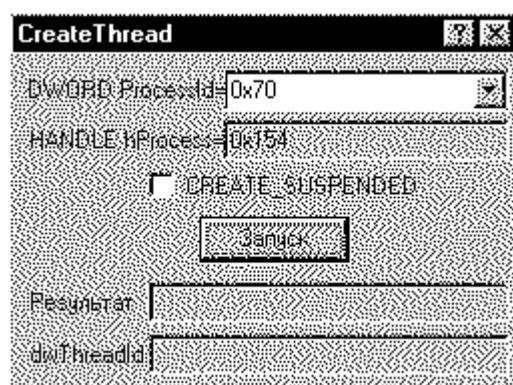


Рис. 8.9.

Добавьте счётчики процессорного времени всех четырёх потоков в Системный монитор.

Пусть они будут иметь такие цвета:

Процесс 1, Главный поток: **Красный**

Процесс 1, Второй поток: **Синий**

Процесс 2, Главный поток: **Желтый**

Процесс 1, Второй поток: **Зелёный**

Теперь назначим одному из потоков первого процесса высокий приоритет (рис. 8.10.):

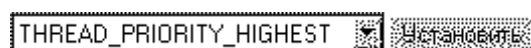


Рис. 8.10.

Пронаблюдав за потоками этого процесса в системном мониторе, замечаем, что этому потоку стало выделяться гораздо больше процессорного времени (рис. 8.11.)

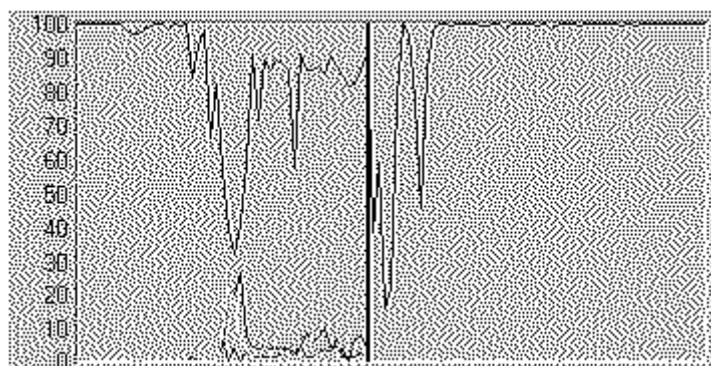


Рис. 8.11.

Теперь назначим второму процессу нормальный класс приоритетов (рис. 8.12.)

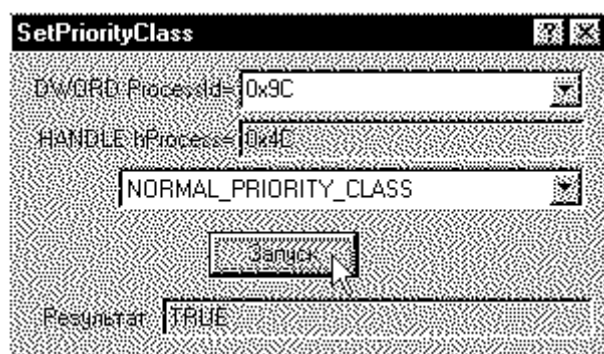


Рис. 8.12.

Видно, что картина сильно изменилась: теперь оба потока второго процесса вытесняют потоки первого, причём предпочтение отдаётся явно не главному потоку второго процесса (рис. 8.13.)

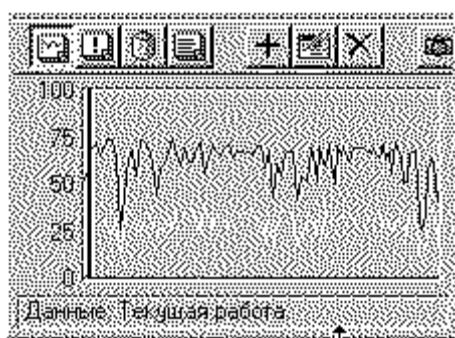


Рис. 8.13.

Попробуем назначить обоим потокам второго процесса низкий приоритет (Рис. 8.14.):

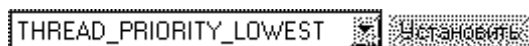


Рис. 8.14.

Теперь количество процессорного времени распределяется так (Рис. 8.15.).

Для сохранения результатов измерений в файле необходимо сначала создать собственный журнал счетчиков, выполнив следующие действия:

1. Открыть оснастку Производительность (Performance).

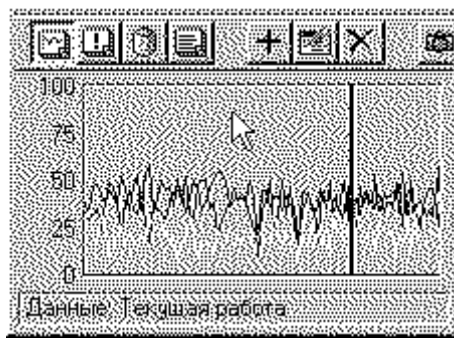


Рис. 8.15.

2. Дважды щелкнуть значок Оповещения и журналы производительности, а затем значок Журналы счетчиков.
3. В области сведений будут показаны все имеющиеся записи журналов. Зеленый значок показывает, что журнал запущен, а красный значок — что журнал остановлен.
4. Щелкните правой кнопкой мыши пустое место в области сведений и выберите пункт меню Новые параметры журнала.
5. В поле Имя введите имя создаваемого журнала и нажмите кнопку ОК.
6. На вкладке Общие (рис. 8.16.) нажмите кнопку Добавить объекты и выберите необходимые объекты; или, нажмите кнопку Добавить счетчики и выберите отдельные счетчики, показания которых следует заносить в журнал.
7. Чтобы изменить параметры файла журнала и расписание, используемые по умолчанию, задайте необходимые параметры на вкладках Файлы журнала (рис. 8.17.) и Расписание (рис. 8.18.).

В открывшемся окне задайте путь, имя и расширение для файла с результатами:

TSV – разделитель данных – табуляция;

CSV - разделитель данных – запятая.

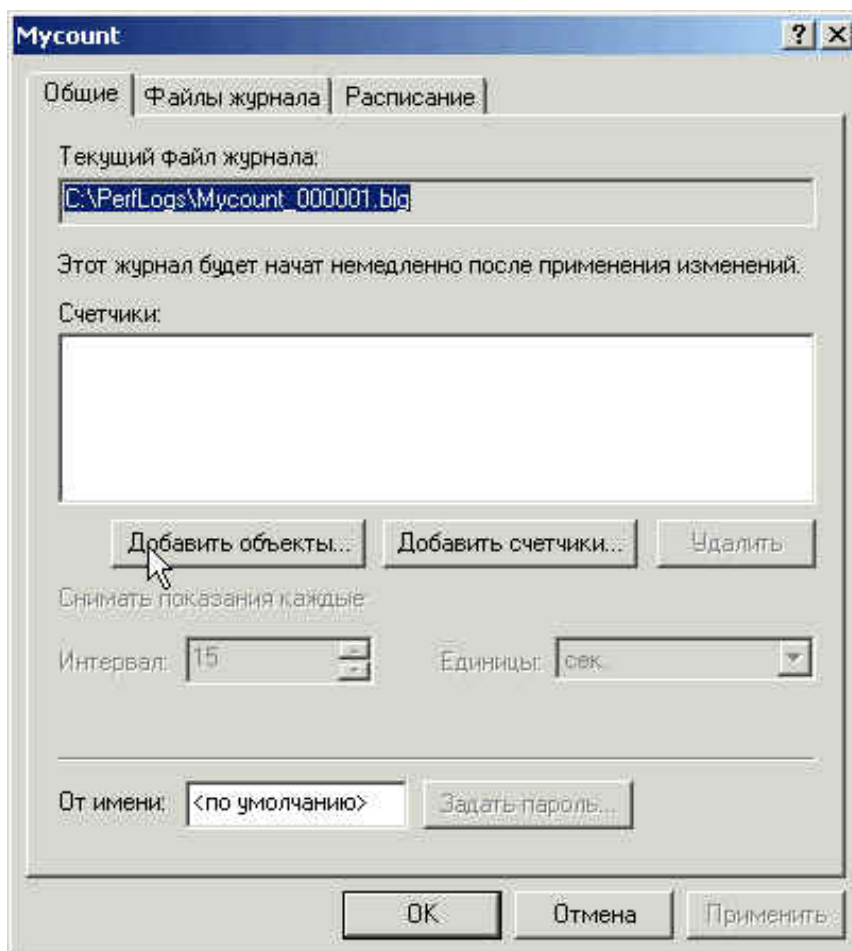


Рис. 8.16.

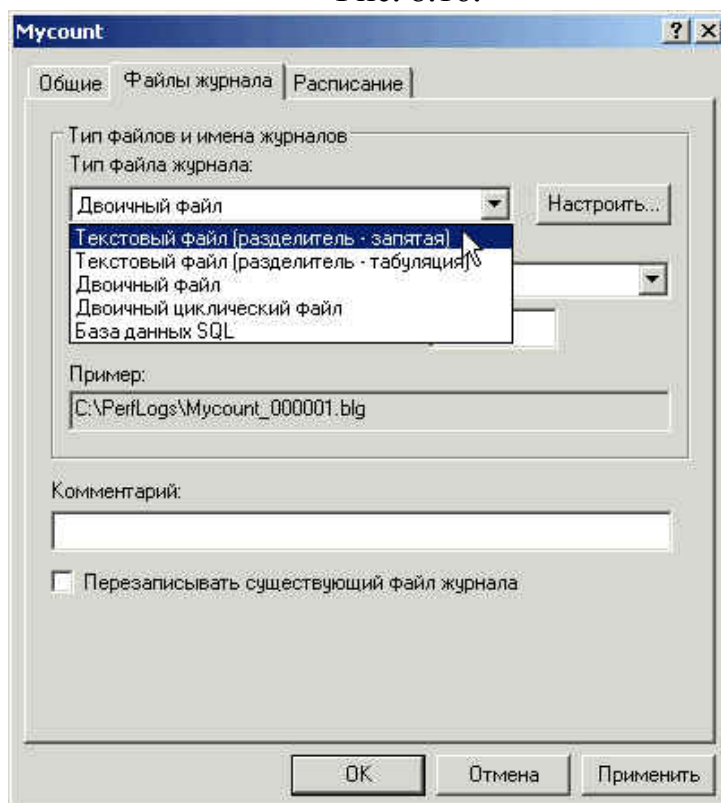


Рис. 8.17.

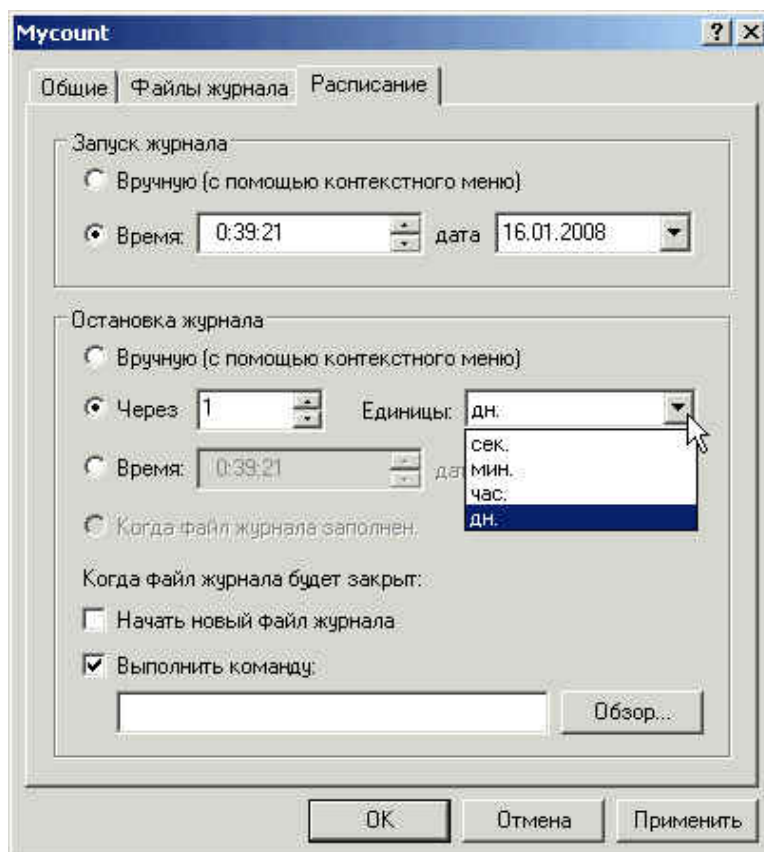


Рис. 8.18.

Файлы с результатами могут быть затем открыты в Excel.

Контрольные вопросы

1. Назначение Системного монитора .
2. Основные параметры настройки Системного монитора.
3. Виды представляемой статистической информации.
4. Типы регистрируемого процессорного времени.
5. Параметры Процессора.
6. Параметры Процесса.
7. Параметры Потока.
8. Параметры Системы.
9. Параметры Дисков.
- 10.Параметры Памяти.

Варианты заданий

Произвести сбор данных о параметрах активности указанного в индивидуальном задании объекта, и сохранить их в файле. Построить диаграммы активности объекта.

- 1) Процессор;
- 2) Память;
- 3) Физические диски;
- 4) Кэш;
- 5) Система;
- 6) Процессы;
- 7) Потoki;
- 8) Файл подкачки;
- 9) Логический диск;
- 10) Количество объектов исполнительной системы;

Литература

1. Джесси М. Торрес. Сценарии администрирования Microsoft Windows. Справочник администратора.: Пер. с англ. – М.: Издательско – торговый дом «Русская редакция», 2005. – 384 с.
2. Борн Г. Руководство разработчика на Microsoft Windows Script Host 2.0. Мастер-класс/ Пер. с англ. — СПб.: Питер; М.: Издательско-торговый дом «Русская Редакция», 2001. — 480 с.