СОДЕРЖАНИЕ

Введение……4
1. Лабораторная работа № 1.
Конкретная среда программирования (интегрированная среда разработки) на базе одного из конкретных алгоритмических языков. Программирование алгоритмов линейной структуры (линейных процессов)……6
3. Лабораторная работа № 2.
Программирование алгоритмов разветвляющейся структуры (разветвляющихся процессов)…….....22
4. Лабораторная работа № 3.
Программирование алгоритмов циклической структуры (циклических процессов)……..28
4. Лабораторная работа № 4.
Программирование алгоритмов, реализующих работу с массивами данных и их обработку…….36
5. Лабораторная работа № 5.
Программирование алгоритмов, реализующих работу со строковым типом данных (действия с символьными строками)…….41
Программирование алгоритмов, реализующих работу в графическом режиме (построение графиков)……46
6. Лабораторная работа № 6.
Комплексное программирование в интегрированной среде разработки (среде программирования) на базе алгоритмического языка Visual Basic……53
Литература……...60
ПРИЛОЖЕНИЕ 1……60
ПРИЛОЖЕНИЕ 2……64

Введение

Данное учебно-методическое пособие (часть 2) предназначено для студентов I-го курса очной формы обучения по направлениям подготовки: 25.03.02, 25.03.03, специальности 25.05.03 и содержит ряд материалов, связанных с изучением, описанием и выполнением 6 лабораторных работ. Предполагается выполнение предложенных лабораторных работ во втором семестре после успешной сдачи студентами части 1 лабораторного практикума (в первом семестре) и приобретения ими умений и навыков работы на “вычислителе” (компьютере, вычислительной машине). Вопросы, рассматриваемые в этих материалах, позволяют и способствуют студентам получить более конкретные практические навыки и умения, необходимые для освоения и усвоения теоретико-практических материалов раздела «Алгоритмизация и программирование».

В качестве базового языка программирования (на основе которого рассматриваются типовые структуры алгоритмов и их программные реализации) выбран процедурно-ориентированный язык типа Basic (Бейсик) - Beginner's All-purpose Symbolic Instruction Code (в частности, его разновидности и версии: Quick Basic 4.5, QB 64, Free Basic). Его теоретико-практические конструкции и структуры в целом отражают общий подход к изучению и освоению основ алгоритмизации и программирования.

Целью проведения и выполнения предлагаемых лабораторных работ является:

- приобретение конкретных практических навыков и умений, необходимых для разработки и написания программных реализаций (программных кодов) различных типов (структур) разрабатываемых алгоритмов конкретных типовых, вычислительных и прикладных практических задач в рамках конкретной среды программирования (интегрированной среды) для конкретного алгоритмического языка (его версии, разновидности);
- приобретение умений и навыков по освоению конкретной среды (системы) программирования на конкретном алгоритмическом языке на базе конкретной интегрированной среды:
- овладение навыками и умениями отладки разработанных программных структур на “вычислителе” (компьютере) с целью получения результатов.

Любая конкретная интегрированная среда (среда программирования) для любого алгоритмического языка, его версий и разновидностей имеет:

- мощный экранный текстовый редактор;
- управляющую среду с многооконным меню;
- подсистему помощи (подсказки);
- отладчик и встроенный компилятор.

Правила подготовки и проведения лабораторных работ (лабораторных занятий):

1. Предполагается, что студенты приходят на конкретное текущее лабораторное занятие, предварительно изучив материалы, излагаемые в предлагаемом пособии, которые касаются вопросов, связанных с проведением и выполнением текущей лабораторной работы. Кроме того, студенты должны подготовить теоретико-практические материалы, необходимые для успешного проведения лабораторной работы и её защиты, по рекомендуемым учебным литературным источникам и материалам из Интернета.

3. В рамках самостоятельной работы – СРС (внеаудиторных занятий) студенты прорабатывают материалы лабораторных работ, приведенных в данном пособии, составляют алгоритмы и разрабатывают программы для рассматриваемых в пособии контрольных вопросов и задач, апробируют свои полученные программы на компьютерах как в рамках СРС, так и на текущих лабораторных занятиях.

4. Студенты должны представить отчеты по своим индивидуальным домашним заданиям по каждой конкретной лабораторной работе, апробировать на компьютере разработанную ими программу, продемонстрировать результаты выполнения домашнего задания. Отчеты по конкретному текущему индивидуальному домашнему заданию для конкретной текущей лабораторной работы подготавливаются и представляются к текущему лабораторному занятию, программная реализация (программные коды) демонстрируется на компьютере в процессе проведения и выполнения текущей лабораторной работы. После чего, данная конкретная работа (окончательный отчет) защищается студентом.

Отчеты (подписанные студентом, с датой выполнения) предоставляются студентом на защиту в бумажном и электронном виде (файл отчета, программный файл). Преподаватель после защиты отчета выставляет оценку, подписывает отчет. Бумажные отчеты после защиты сдаются преподавателю.

Варианты индивидуальных домашних заданий (ИДЗ) из имеющегося фонда заданий по каждой конкретной лабораторной работе выдаются преподавателем студентам на первых лабораторных занятиях во втором семестре.

В Приложении 1 представлены:

- структура и содержание отчета (образец) по лабораторной работе;
- образцы примерных вариантов индивидуальных домашних заданий для каждой конкретной лабораторной работы.

В Приложении 2 представлен:

- образец титульного листа для отчета по лабораторной работе (с учетом соответствующего конкретного варианта ИДЗ для соответствующей конкретной текущей лабораторной работы).

Лабораторная работа № 1

КОНКРЕТНАЯ СРЕДА ПРОГРАММИРОВАНИЯ (ИНТЕГРИРОВАННАЯ СРЕДА) НА БАЗЕ ОДНОГО ИЗ КОНКРЕТНЫХ АЛГОРИТМИЧЕСКИХ ЯЗЫКОВ. ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ЛИНЕЙНОЙ СТРУКТУРЫ (ЛИНЕЙНЫХ ПРОЦЕССОВ)

Цель работы:

1. Приобретение навыков в составлении простейших программ (реализация алгоритмов линейной структуры) на конкретном алгоритмическом языке (типа Basic, версии: Quick Basic 4.5, QB 64, Free Basic).

2. Изучение конкретной интегрированной среды разработки (среды программирования) на примере Quick Basic (Free Basic). Приобретение практических навыков работы в редакторе конкретной интегрированной среды разработки (например, в экранном редакторе Quick Basic, версия 4.5).

Работа в интегрированной среде разработки

Рассмотрим работу в конкретной интегрированной среде разработки (среде программирования) на примере системы программирования (интегрированной системы разработки) Quick Basic.

Начальная информация о системе Quick Basic

Система программирования Quick Basic (версия 4.5) - интегрированная система, включающая текстовый редактор, управляющую среду с многооконными меню, подсистему помощи HELP, отладчик и встроенный компилятор. Работает и функционирует в операционной системе MS Windows.

Загрузка системы Quick Basic

Загрузка (вход в систему) Quick Basic может осуществляться следующими способами:

- двойным щелчком по ярлыку на рабочем столе **Windows;**
- через программы (окна) **Проводник, Мой компьютер**: после запуска одной из этих программ открыть папку ***QBasic)***, установить указатель мыши на файле ***Qbasic.exe (QB.exe)*** и дважды щелкнуть или нажать **Enter**.

После входа в Quick Basic на экране появляется окно редактора, в центре которого на сером фоне может стоять приглашение в систему, в этом случае нажмите **Esc**. Экран примет вид, представленный на рис. 1.

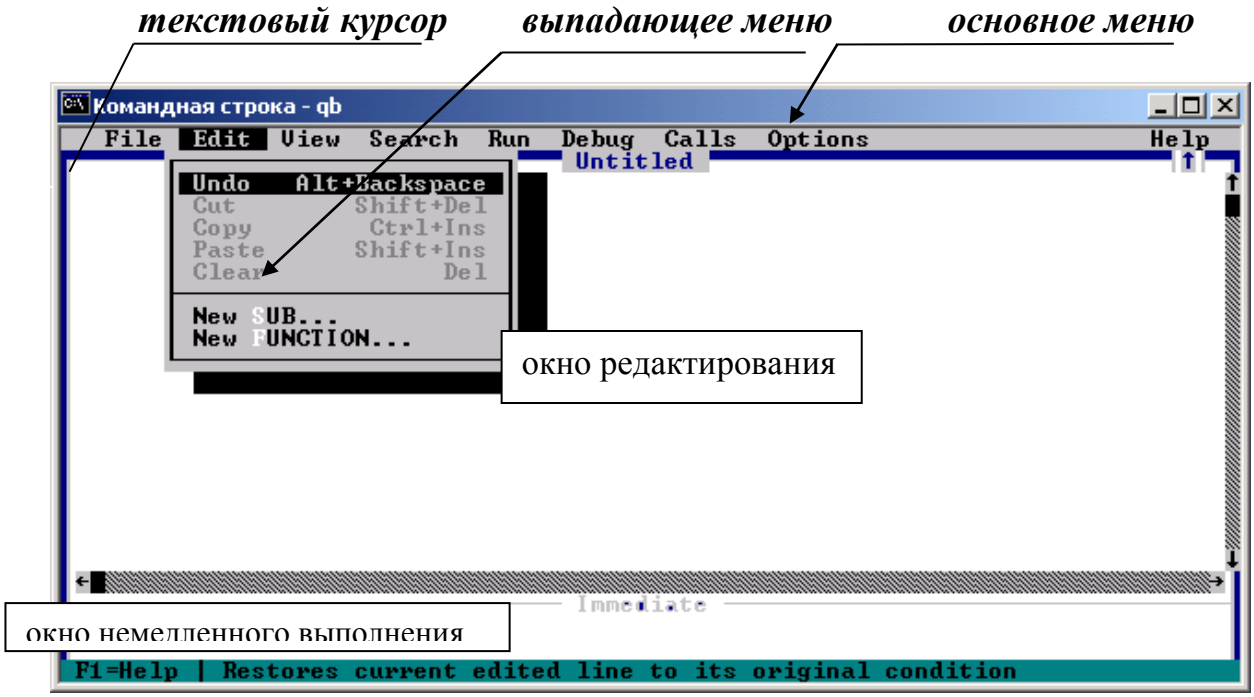


Рис. 1. Экран редактора Quick Basic.

На экране можно выделить три основные зоны:

- верхняя строка экрана - основное меню системы;
- окно редактирования с курсором в начале первой строки окна;
- нижнее прямоугольное окно с названием Immediate (немедленно).

Основное меню системы Quick Basic состоит из ряда пунктов:

File - работа с файлами и операционной системой;
Edit - ввод и редактирование текста;
View - отображение отдельных страниц с текстами;
Search - работа с фрагментами текста (поиск, выделение, замена);
Run - запуск программ;
Debug - отладка программ;
Calls - вызов отдельных процедур;
F1=Help - помощь.

Основное меню предназначено для выбора режимов работы системы. Вход в меню осуществляется щелчком мыши по выбранному пункту, после чего раскрывается (выпадает) подменю выбранного пункта. Подменю содержит список команд (рис. 1).

С помощью клавиатуры также можно войти в меню, нажав клавишу **Alt**, при этом выделится пункт **File**. Дальнейший выбор пункта меню осуществляется передвижением курсора с помощью клавиш управления курсором и нажатием клавиши Enter, после чего откроется подменю.

Подменю позволяет детализировать работу выбранного режима. Для выполнения команды подменю следует установить на нее указатель мыши (текстовый курсор) и щелкнуть (нажать ENTER)**.**

Часто вход в пункт подменю открывает диалоговое окно. Диалоговые окна обеспечивают дальнейшую детализацию выбранного режима и назначения необходимых условий работы в нем (рис. 2, 3).

Диалоговые окна имеют несколько полей, ограниченных прямоугольной рамкой. Переход от одного поля к другому осуществляется щелчком мыши или нажатием клавиши TAB. Курсор, находящийся в окне, указывает на активное поле, в котором можно осуществлять необходимые записи, (например: записать путь к файлу). Выход из окна осуществляется нажатием клавиши ESC .

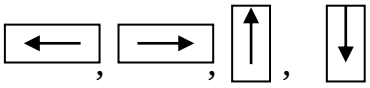
Текстовый редактор системы Quick Basic

Окно редактирования предназначено для записи и редактирования программ с использованием встроенного текстового редактора системы Quick Basic.

При загрузке системы вышеуказанным способом автоматически устанавливается режим редактирования. После входа в режим редактирования курсор установлен в левом верхнем углу чистого поля экрана и показывает место, с которого можно набирать программу. Текст программы вводится с клавиатуры.

Далее перечислены наиболее часто употребляемые команды текстового редактора, выполняемые с помощью клавиш управления курсором:

Смещение курсора:



- на символ влево, вправо, вверх, вниз;

PGUP - на экранную страницу вверх;

PGDN - на экранную страницу вниз;

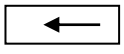
HOME - в начало строки;

END - в конец строки;

CTRL-HOME - в начало текста;

CTRL-END - в конец текста.

Команды редактирования:


, BACKSPACE - стереть символ слева от курсора;

SHIFT - СМЕЩЕНИЕ КУРСОРА - выделение текста;

SHIFT - DEL - удаление выделенного текста с сохранением в буфере;

SHIFT - INS - восстановить удаленный текст из буфера;

DEL - стереть символ над курсором,

удалить выделенный текст;

ENTER - вставить пустую строку,
разрезать строку;

CTRL - Y - удалить текущую строку
с сохранением в буфере.

Нормальный режим работы редактора - режим **вставки,** текстовый курсор имеет вид подстрочной черточки. В режиме вставки пропущенный символ вставляется перед тем символом, под которым установлен курсор.

Для перехода к режиму замены, нажимают клавишу INS, текстовый курсор примет форму прямоугольника. В этом режиме ошибочный символ заменяется на правильный, если под ним установлен курсор.

Выполнение программы

После загрузки системы программирования необходимо:

- ввести текст программы;
- отладить программу;
- выполнить и получить результат.

После ввода текста программы следует:

а) войти в основное меню;

б) установить курсор на пункт **RUN** и нажать **ENTER** ;

в) в открывшемся подменю выбрать пункт **START** и нажать **ENTER**.

Пункты а)÷ в) можно заменить нажатием "горячих клавиш" - **SHIFT-F5**.

После запуска программы на выполнение возможны две ситуации:

- в программе транслятор системы обнаружил ошибки;
- в программе ошибок не обнаружено.

Если в программе имеются ошибки, то сообщение о первой высвечивается в открывающемся на экране окне, а место ошибки отмечается в тексте программы курсором. Для продолжения работы нужно нажать клавишу **Esc**, исправить ошибку и снова запустить программу на выполнение.

Если больше ошибок не обнаружено, то программа выполняется и на экране появляется результат. В нижней строке экрана сообщение "*Press any key to continue*". Нажмите любую клавишу для продолжения, и Вы вернетесь в окно редактора. Для повторного просмотра результатов нужно нажать **F4.**

Создание и Сохранение программы - команды меню File

В таблице 1 приведены основные команды подменю File:

Таблица 1

Команда подменю	**Выполняемое действие**
New Program	Устанавливается режим редактирования. При выборе этого режима очищается ОЗУ от старых программ и подготовка к вводу новой программы под именем UNTITLED, которое при сохранении файла на диске нужно изменить**.**

Open Program	Загружается ранее созданный файл.
Save , Save As	Команды позволяют сохранить программу на диске.
Dos Shell	Временный выход в среду MS-DOS, возврат в Quick BASIC осуществляется командой **EXIT.**
Exit	Выход из среды Quick BASIC

При выполнении программы, набранной непосредственно в редакторе, вся информация хранится в оперативной памяти “вычислителя” (компьютера, ЭВМ). Чтобы сохранить программу на диске, необходимо установить курсор на пункт верхнего меню **File** и нажать **Enter.** Откроется подменю, в котором нужно выбрать пункт **Save**. Система откроет для этого новое диалоговое окно (см. рис. 2) и предложит вам ввести имя файла программы в поле окна **File name**. В поле **Dirs/Drives** выберите диск и каталоги, а в поле **Format – Text**, затем нажмите **Enter**.

При повторном сохранении программы ее имя не запрашивается.

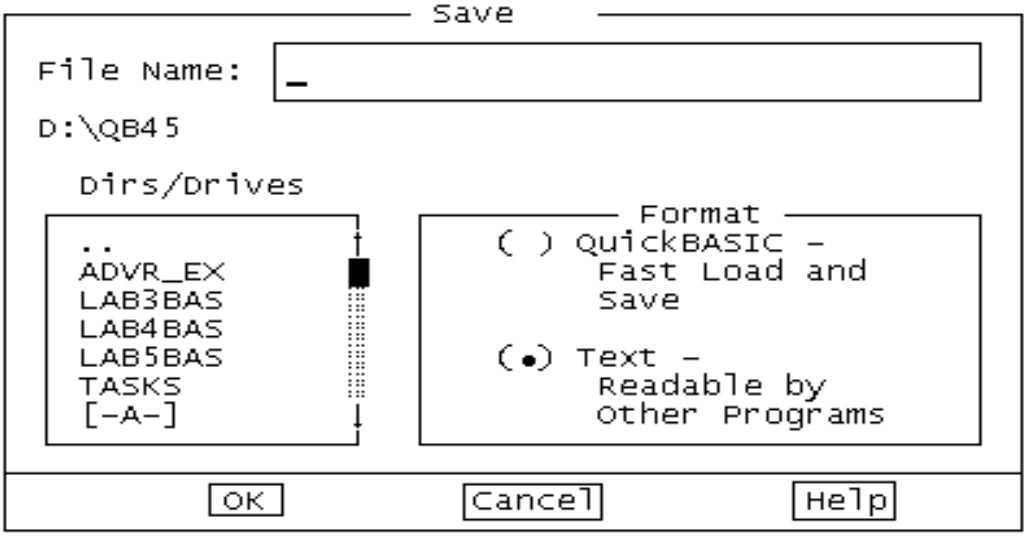


Рис. 2. Диалоговое окно Save

В пункте меню **File** также имеется опция **Save As** (сохранить как), которая позволяет сделать копии файлов под новыми именами, сохранив при этом оригинал, или записать на диск измененный файл под новым именем или в другой папке. При обращении к этой опции система открывает диалоговое окно, подобное **Save**.

Сохраняйте программу по частям в процессе ее набора

При обращении к сохраненному ранее файлу нужно войти в меню **File** и выбрать пункт **Open Program.** Откроется диалоговое окно, представленное на рис. 3. С помощью мыши в поле **Dirs/Drives** следует выбрать диск, а затем папку. В поле **Files** выбрать файл.

Для ввода следующей программы нужно войти в меню **File** и выбрать пункт **New Program.** Окно очистится, и Вы приступаете к набору текста следующей задачи.

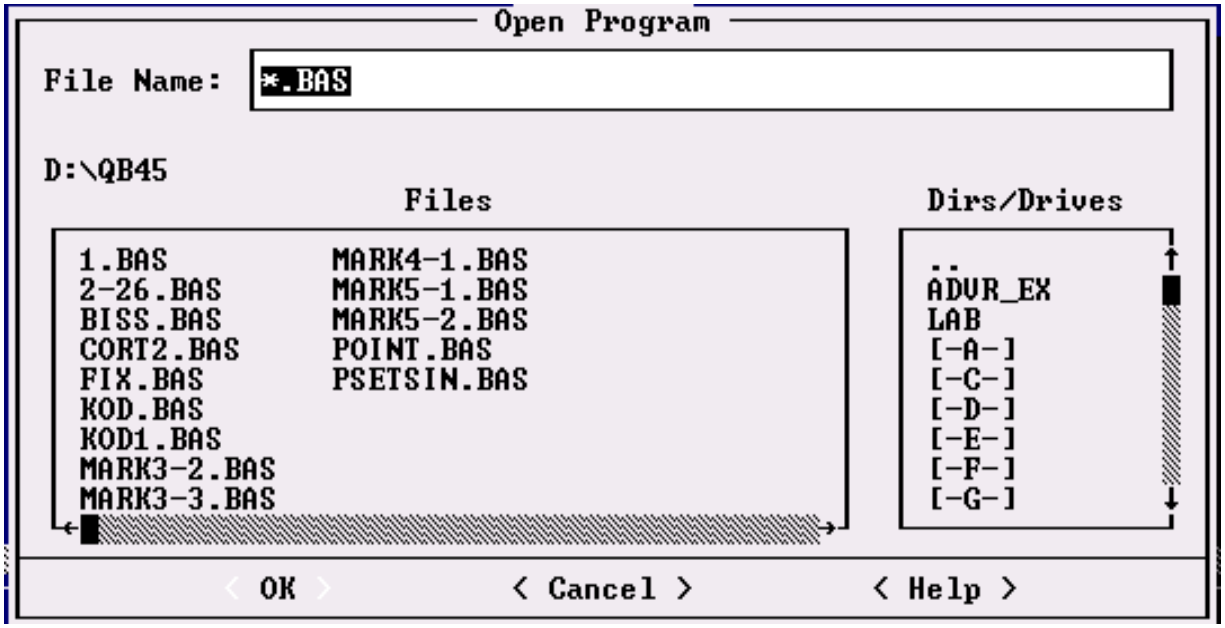


Рис. 3. Диалоговое окно " Open Program"

Запуск и перезапуск программы - команды меню Run

Ранее уже говорилось о том, как программа запускается на выполнение с помощью подменю **Run**. Но этот пункт основного меню среды предоставляет собой набор команд, позволяющих управлять программой, режимами компилятора и редактора связей. После входа в пункт **Run** открывается подменю (таблица 2):

Таблица 2

Команда подменю	Выполняемое действие
Start, Shift+F5	Запускает программу на выполнение.
Restart	Устанавливает пошаговый режим исполнения программы, переход к следующему шагу осуществляется нажатием F8. Этот режим удобен при отладке программы совместно с использованием режима отображения дисплейной памяти, который устанавливается клавишей F4
Continue, F5	Продолжение выполнения программы с точки ее останова.
Make EXE File	Позволяет создать исполняемый файл с расширением "exe". Такой файл запускается из ОС.

Отладка программы - Команды и режимы меню Debug

Пункт **Debug** основного меню среды задает режимы выполнения программы и позволяет отображать промежуточные результаты на этапе отладки. После входа в пункт **Debug** открывается подменю, список команд которого приведен в таблице 3:

Таблица 3

Команда	Выполняемое действие

подменю	
Add Watch	Позволяет указать имена переменных и выражения, значения которых будут отображаться в окне под основным меню. Окно удобно использовать при пошаговом исполнении программы.
Watchpoint	Позволяет указать имена переменных и выражения **логического** типа, значения которых проверяются на достижение значения **True** (истина). Информация отображается в окне под основным меню. Как только условие выполнено, программа приостанавливается.
Delete Watch	Используется для удаления из окна отдельных переменных или выражений.
Delete All Watch	Полностью удаляет окно со всеми контролируемыми переменными.
Trace On	Включает и выключает режим трассировки.
Toggle Breakpoint, F9	Включает или выключает режим прерывания программы в тех строках, где находится курсор.
Clear All Breakpoint	Отключает все ранее установленные прерывания.

Подсистема помощи - HELP

Вход в подсистему осуществляется выбором пункта меню **HELP** основного меню. В развернувшемся подменю предлагается четыре режима помощи, список которых приведен в таблице 4:

Таблица 4

Режим помощи	Выполняемое действие
Index	Выдается справка по всем ключевым словам, операторам и функциям
Contents	Выдается перечень разделов справочника, по которым пользователь может получить справку
Topic	Справка по конкретному оператору или функции (Shift-F1)
Help	Выдается справка о самой подсистеме помощи (F1)

Использование окна "Immediate"

При работе в среде Quick Basic, возможны два способа исполнения программных строк: автоматический, т.е. в соответствии с введенной программой, и командный. В первом случае осуществляется компиляция программы в памяти, а затем производится исполнение.

В командном режиме работы возможно непосредственное исполнение отдельных программных строк. Этот режим работает, если программные строки или операторы занесены в окно "Immediate" и запускаются на выполнение нажатием клавиши **Enter**. Для того, чтобы поместит текст строки в окно **"Immediate",** нужно нажать клавишу **F6**, курсор переместится

в окно и затем набрать нужный оператор. Также можно поместить текст оператора в окно следующим образом:

отметить текст	-	SHIFT + клавиши управления курсором;
скопировать текст в буфер	-	CTRL + INS;
перейти в окно "Immediate"	-	F6;
скопировать текст из буфера	-	SHIFT + INS

При использовании других версий и диалектов языков программирования и, соответственно, других интегрированных сред разработки (систем программирования) прослеживается определенная аналогия процесса ознакомления, освоения и применения конкретных интегрированных сред разработки для разработки пользователями (студентами) программных продуктов (программ, программных кодов), программной реализации разрабатываемых, проектируемых и создаваемых алгоритмов решения конкретных типовых, прикладных практических задач.

Для успешного освоения конкретных систем программирования (соответственно, интегрированных сред разработки для соответствующих языков программирования) используется как известные литературные источники, так и специальная литература (конкретная доступная документация), Интернет-ресурсы, Интернет-технологии.

Программирование алгоритмов линейных структур (линейных вычислительных процессов)

При решении любой конкретной теоретико-практической задачи с использованием компьютерных технологий предполагается выполнение следующих этапов:

1. Постановка задачи (словесное описание задачи) (указание исходной информации, предполагаемых достигаемых целей, методологии получения необходимого предполагаемого результата, “путей” реализации предполагаемого подхода к решению поставленной задачи с целью получения необходимого или “желаемого” результата решения).

2. Математическое (формализованное) описание задачи (математическая формулировка условия задачи, предполагаемых целей и результата решения) (описание математической модели, описание математических – численных – методов реализации предполагаемой математической модели).

3. Разработка алгоритма решения поставленной предполагаемой задачи (его описание и представление в словесной, математической, графической форме) – алгоритмизация задачи.

4. Разработка, проектирование, описание программной реализации (программы, программных кодов) предлагаемого разработанного алгоритма – программирование (запись алгоритма решения задачи) на конкретном алгоритмическом языке в конкретной интегрированной среде разработки (среде программирования).

5. Апробация разработанной программной реализации на “вычислителе” (компьютере, ЭВМ) с целью получения результата решения задачи. Осуществляется отладка программы (трансляция, компиляция, редактирование) и получение результатов решения (выполнение, “машинный счет”).

Во многих предлагаемых практических заданиях к лабораторным работам (задачах, примерах, индивидуальных домашних заданиях), как правило, условия задач, которые нужно решить, уже представлены в математической формулировке с указанием численных методов решения и, поэтому, необходимость в выполнении второго этапа может отпасть.

Определения.

Алгоритм – четкое конкретное описание (предписание) последовательности действий, которые необходимо выполнить (совершить) для решения задачи (над исходной информацией с целью получения результата).

Алгоритмизация - п**роцесс построения** (проектирования, создания, разработки, объяснения, определения) **алгоритма**, то есть определение последовательности действий, необходимых для решения задачи.

Программа – программная реализация *алгоритма*, записанная на конкретном языке программирования в конкретной среде программирования.

Программирование – процесс (процедура) написания (разработки, проектирования) программы (программных кодов) на конкретном языке программирования в конкретной среде программирования.

Алгоритм, как правило, представляется на естественном языке, понятном человеку, который будет писать (разрабатывать) программу, либо в виде графической структуры (схемы) алгоритма, которая состоит из неких стандартных символов и базовых “управляющих” структур (см.[3]). Графическое представление алгоритма это графическое изображение процесса решения конкретной задачи в виде графических условных обозначений (символов) и их связей.

Программа (программные коды) пишется на одном из конкретных языков программирования, который понятен и человеку (иначе он не сможет написать программу) и “вычислителю” (компьютеру, ЭВМ, ВМ) (иначе он не сможет её выполнить).

Форма представления алгоритма может быть как текстовой, так и графической - в виде схемы. Решение всего многообразия задач может быть сведено к трем типам алгоритмов: линейному, разветвляющемуся и циклическому. Чаще встречается комбинация этих типов.

Программирование линейных алгоритмов

Линейный алгоритм - алгоритм, в котором к результату решения задачи приводит последовательное выполнение действий**.**

Алгоритм решения такой задачи в словесной форме состоит из следующих пунктов: начало программы; ввод исходных данных; вычисления; вывод результатов; окончание программы.

Программы, реализующие линейный алгоритм, как правило, очень просты и для их реализации используются операторы ввода данных, выполнения действий (вычислений) и вывода результатов.

Программирование на алгоритмическом языке Бейсик

Программа на языке Бэйсик - это последовательность строк, описывающих алгоритм решения конкретной задачи. Строка может содержать один или несколько операторов, разделенных двоеточием, а также комментарии.

Оператор представляет собой строго формализованное указание на выполнение конкретного действия.

Каждая строка может начинаться с метки. Метка может быть цифровой или буквенно-цифровой. Буквенно-цифровые метки могут иметь от 1 до 40 символов и начинаться с буквы, а завершаться двоеточием, тогда как цифровая метка завершается пробелом. Метка не определяет порядок выполнения строк программы, не обязательна, и служит, как правило, для ссылки на нее. Программные строки выполняются в порядке их записи. Длина программной строки не должна превышать 256 символов.

Современные алгоритмические языки используют наборы различных типов данных.

Программа, написанная на языке Бэйсик, обрабатывает числовые и символьные данные. Данные представляются в программе в виде констант и переменных. Тип данных определяет возможные значения констант и переменных, форму представления в компьютере, объем занимаемой памяти, операции, которые могут выполняться над данными этого типа.

Числа. Бэйсик оперирует с двумя типами чисел: вещественными и целыми.

Под целое число отводится 2 байта памяти, и оно хранится в форме с фиксированной точкой. Запись целого числа представляет собой последовательность цифр со знаком или без него (например:5487; -7821; +3841).

Вещественные числа хранятся в ячейке памяти длиной 4 байта в форме с плавающей точкой. Возможны две формы "внешней" записи вещественных чисел в программах:

- с фиксированной точкой (например: - 3.7);
- с плавающей точкой (например: -00.45E2; 0.78D-3, здесь буквы “E” и “D” означают основание “10”, обычной и двойной точности соответственно, разделяют мантиссу и порядок).

Числовое или символьное значение может быть присвоено переменной или константе.

Переменная - величина, которая может меняться при выполнении программы. Переменная всегда имеет имя, которое содержит не более 40 буквенно-цифровых символов и начинается с латинской буквы.

Способы описания типа данных в Бэйсике:

Явно - с помощью определенных суффиксов, которые добавляются к имени переменных или констант.

Явно - с помощью операторов описания типа.

Неявно - с помощью оператора объявления типа данных по первой букве имени переменной.

В таблице 5 приведены диапазоны числовых данных.

Таблица 5

тип	**диапазон**
целый	-32768 ÷ +32767
длинный целый	-2147483648 ÷ +2147483647
веществ. обычной точности	-3.402823E+38 ÷ -1.40129E-45 +1.40129E-45 ÷ +3.402823E+38
веществ. двойной точности	-1.79769E+308 ÷ -4.94965E-324 +4.94965E-324 ÷ +1.79769E+308

В таблице 6 приведены описания типа данных.

Таблица 6

тип	**суф-фикс (явно)**	**оператор описания (явно)**	**оператор объявления (неявно)**	**объем памяти в байтах**
Целый	%	DIM имя as integer	DEFINT	2
пример	NAME1%	DIM NAME1 AS INTEGER	DEFINT N	
Длинный целый	&	DIM имя as LONG	DEFLNG	4
пример	NAME2&	DIM NAME2 AS LONG	DEFLNG N	
Веществ. обычной точности	!	DIM имя as SINGLE	DEFSNG	4
пример	NAME3!	DIM NAME3 AS SINGLE	DEFSNG N	
Веществ. двойной точности	#	DIM имя as DOUBLE	DEFDBL	8
пример	NAME4#	DIM NAME4 AS DOUBLE	DEFDBL N	
Символь-ный	$	DIM имя as STRING	DEFSTR	4+n байт (n-кол-во символов)
пример	NAME5$	DIM NAME5 AS STRING	DEFSNG N	

Константы. Значения констант не меняются в процессе работы программ. В Бэйсике различают два вида констант: - **неименованные** и **именованные**. Константы бывают **числовые** и **символьные.** Неименованная числовая константа - это число, а именованная константа должна быть объявлена с помощью ключевого слова CONST, например:

```
CONST   PI = 3.14
CONST   PL = 0.23E-3        ' ФОРМА E
```

CONST Z$ = " ПРИВЕТ " ' СИМВОЛЬНАЯ

Если тип числовой переменной не указан, то он автоматически становится - Single (для Quick Basic) и автоматически становится – Integer (для Free Basic). Если тип константы (то есть переменной в её записи) явно не указан, то он “привязывается” к типу выражения, присваиваемого константе.

В данной лабораторной работе № 1 мы ограничимся рассмотрением числовых данных.

Ввод числовой информации (данных)

Ввод данных в программах, написанных на алгоритмическом языке Бэйсик можно осуществлять несколькими способами:

1) **input a, b, c -** оператор ввода данных в диалоговом режиме**,**
где **a, b, c** - список имен переменных**.**

Встретив этот оператор, компьютер останавливает выполнение программы. На экране появляется знак "**?**", затем значения переменных нужно ввести через запятую и нажать **Enter**. Количество, последовательность и тип вводимых данных должны соответствовать именам переменных оператора "**input**".

В списке ввода оператора "**input**" на первом месте можно дать подсказку - текст, заключенный в двойные кавычки, например: **input "a,b,c:",a,b,c.** При выполнении такого оператора "**input**" на экране появится подсказка (**a,b,c:**), после чего нужно ввести значения переменных и нажать **Enter**.

2) **read a, b, c; data -** оператор **input** считывает данные, перечисленные в операторе **data,** причем количество данных должно соответствовать переменным в операторе **read** по количеству, типу и порядку следования, например:

data 5, -8, 12.5
read a, b, c .

При выполнении оператора **read a, b, c:**

- переменной **a** будет присвоено значение **5**;
- переменной **b** будет присвоено значение **-8**;
- переменной **c** будет присвоено значение **12.5**.

3) задать исходные данные можно также операторами присваивания, например:

a = 5 : b = -8 : c = 12.5.

Выполнение вычислений

Для вычисления арифметических выражений используется оператор **присваивания**, частный случай которого применяется и для ввода данных.

Общий вид оператора:

Q=P,

где **Q -** имя переменной; **P -** арифметическое выражение.

Арифметические выражения соответствуют общепринятым алгебраическим выражениям, в них могут входить числа, переменные,

функции, соединенные знаками арифметических операций и круглыми скобками.

В Бэйсике используются следующие арифметические операции:

^ - возведение в степень;

*, / - умножение, деление;

\ - целочисленное деление;

MOD - определение остатка от деления;

+,- - сложение, вычитание.

Операции перечислены в порядке убывания приоритета их выполнения. Действия внутри круглых скобок выполняются первыми.

Примеры записи некоторых арифметических операций и их результаты приведены в таблице 7.

Таблица 7

ВЫРАЖЕНИЕ	РЕЗУЛЬТАТ	**ПРИМЕЧАНИЕ**
2 ^ -2	0.25	возведение в степень -2
7 / 2	3.5	
7 \ 2 1.6 \ 2.2	3 0	делимое и делитель (операнды) округляются, тип результата INTEGER
7 MOD 2 8.3 MOD 3.3 8.6 MOD 3.3	1 2 2	операнды округляются, тип результата INTEGER

Математические функции

При записи функций на Бэйсике *аргумент функции заключается в круглые скобки*. В качестве аргумента математических функций может быть число, переменная или арифметическое выражение. Следует заметить, что в тригонометрических функциях аргумент должен быть задан в радианной мере.

Наиболее распространенные функции языка Бэйсик:

ABS(X) - вычисляет модуль аргумента, что соответствует математической записи $|\mathbf{x}|$;

EXP(X) - экспонента, соответствует математической записи $\mathbf{e}^{\mathbf{x}}$;

LOG(X) - вычисляет натуральный логарифм аргумента, что соответствует математической записи **ln(x);**

SQR(X) - вычисляет корень квадратный из аргумента;

ATN(X) - вычисляет арктангенс аргумента;

COS(X) - вычисляет косинус аргумента;

SIN(X) - вычисляет синус аргумента;

TAN(X) - вычисляет тангенс аргумента;

RND(X) - выдает случайное число обычной точности в интервале 0÷1. Аргумент может быть опущен. Рекомендуется в начале программы запустить генератор случайных чисел оператором **RANDOMIZE TIMER**;

SGN(X) - определяет знак аргумента. Если аргумент отрицательный, функция принимает значение (**-1**), если положительное - (**+1**). При нулевом аргументе функция также принимает значение **0;**

FIX(X) - отбрасывает дробную часть значения аргумента;
INT(X) - округляет аргумент в сторону уменьшения;
CINT(X) - округляет аргумент по математическим правилам.

Примеры записи функций округления и их результаты в таблице 8.

Таблица 8

Выражение	Результат	Выражение	Результат	Выражение	Результат
FIX(5.7)	5	INT(5.7)	5	CINT(5.7)	6
FIX(5.1)	5	INT(5.1)	5	CINT(5.1)	5
FIX(-5.7)	-5	INT(-5.7)	-6	CINT(-5.7)	-6
FIX(-5.1)	-5	INT(-5.1)	-6	CINT(-5.1)	-5

Для более подробного ознакомления с набором встроенных функций и их синтаксисом необходимо обратиться к документации по описанию конкретной версии (диалекта) конкретного алгоритмического языка (конкретной системы программирования).

Вывод данных и результатов

print x,y,z - оператор вывода данных и результатов,
где **x,y,z -** список элементов вывода.

В качестве элементов вывода могут быть имена переменных, арифметические выражения, а также текст, заключенный в двойные кавычки.

При выполнении оператора на экран выводятся значения переменных, арифметических выражений, текст. Список может отсутствовать и в этом случае на экране пропускается строка.

Разделителем элементов вывода может быть запятая или точка с запятой, от этого зависит интервал в строке вывода на экране между выводимыми данными. В Бейсике строка делится на пять равных зон. Если разделителем является запятая, то очередной элемент выводится в начале следующей зоны. В том случае, когда разделителем является точка с запятой, очередной элемент выводится через пробел.

Пример 1.1. Составить программу вычисления и вывода на экран радиусов описанной и вписанной окружностей R1 и R2 правильного многоугольника, а также площади правильного многоугольника. Количество сторон многоугольника - n и длину его стороны - *a* задать с экрана монитора (клавиатуры). Для вычисления воспользуемся следующими формулами:

$R1 = \dfrac{a}{2\sin(3{,}14/n)}$ - радиус описанной окружности;

$R2 = \dfrac{a}{2tg(3{,}14/n)}$ - радиус вписанной окружности;

$S = \dfrac{n \cdot a \cdot R2}{2}$ - площадь правильного многоугольника.

Алгоритм решения задачи в словесной форме состоит из следующих пунктов: начало; ввод значений переменных **a** и **n**; вычисление функции R1, R2 и S; вывод значений функций R1, R2 и S; окончание программы.

```
REM Пример 1.1. Вычисление по формулам
CLS
PRINT "Введите значения переменной N"
INPUT N
INPUT "Введите значение переменной A"; A
R1=A/(2*SIN(3.14/N)): R2 = A/(2*TAN(3.14/N)): S = N*A*R2/2
PRINT "N="; N,"A="; A, "R1="; R1, "R2="; R2,"S="; S
END
```

Пояснения к программе:

- оператор **REM**, позволяет вводить комментарии, пояснения к программе;
- **CLS** очищает экран;
- **PRINT** предназначен для вывода на экран текста, заключенного в апострофы, и значений переменных;
- **INPUT** служит для ввода значений переменных по запросу после "**?**";
- для вычисления искомых величин используется оператор присваивания;
- **END** - окончание программы.

Пример 1.2. Составить программу вычисления по формуле и вывода на экран результаты вычислений:

$$z = 7{,}5e^{xy} + \sqrt[3]{\frac{5y}{7x}} + \log_9(xy).$$

```
REM Пример 1.2. Вычисление по формуле
CLS
INPUT "Введите через запятую значения переменных x,y", X,Y
P=X*Y   ' отдельно вычислим xy и обозначим      P
Z=7.5*EXP(P)+(5*Y/(7*X))^(1/3)+LOG(P)/LOG(9)
PRINT "Z="; Z
END
```

Пояснения к программе:

В приведенной программе используются те же операторы, что и в примере 1.1. Следует обратить внимание на запись арифметического выражения:

- скобки определяют последовательность выполнения вычислений, количество открытых скобок равно количеству закрытых;
- для вычисления корня использовано возведение в степень;
- для вычисления логарифма по основанию 9 используется формула перехода от одного основания к другому.

Комментарий также можно записывать после апострофа.

Ниже приводятся практические задания с образцами примеров типовых заданий по программированию линейных алгоритмов. По этим типовым заданиям студенты в рамках самостоятельной работы студентов (обучающихся) – СРС или СРО) разрабатывают алгоритмы (составляют схемы) решения этих задач и их программные реализации (программные коды) в конкретной среде программирования, апробируют на компьютере и сохраняют в электронном виде (в отдельных файлах) на носителе.

Задание 1.

а) $4^{\ln x} + \log_3 7x^2 - \dfrac{\sin(x+y)}{\cos(x-y)}$.

б) $2x^2 + \arccos\dfrac{1}{x} \cdot \log_3 2y$.

Задание 2.

1. По заданным коэффициентам a1, a2, b1, b2, c1, c2 найти решение системы уравнений:

$$\begin{cases} a1 * x + b1 * y = c1 \\ a2 * x + b2 * y = c2 \end{cases}$$

2. Определить сумму квадратов цифр, входящих в заданное трехзначное число f.

Задание 3.

а) $\dfrac{1+\sin^2(x+y)}{2+\left|x-\dfrac{2x}{1+\left|\sin^2(x+y)\right|}\right|}$ б) $\dfrac{x}{1+\cos\dfrac{x}{1+x}} + \left|\sqrt[3]{x-y^{\sin 2x}}\right|$

При написании программ, связанных с вычислением формул, желательно и целесообразно реализовывать вычисление формул “по частям”, разбивая более сложные по записи формулы на отдельные части (подформулы), отдельно их последовательно программируя и вычисляя, производя контроль за процессом вычисления путем организации промежуточного вывода результатов как промежуточных вычислений, так и окончательных результатов.

На лабораторной работе № 1 преподаватель выдает каждому студенту конкретный вариант *индивидуального домашнего задания (ИДЗ)*, в котором содержатся комплексные практические задания, касающиеся тематики конкретной лабораторной работы (ЛР № 1, ЛР № 2, ЛР № 3, ЛР № 4, ЛР № 5, ЛР № 6).

Лабораторное задание

1. Изучить и освоить (начальное освоение) окно редактирования, его режимы соответствующего экранного редактора конкретной интегрированной среды.

2. Выполнить приведенные в лабораторной работе № 1 типовые задания: написать алгоритм (линейной структуры) решения каждого примера и его программную реализацию; сохранить каждую программу каждого типового задания в отдельном файле (программном файле); каждую программу апробировать (отладить) на компьютере; получить результаты вычислений (“машинного счета”).

3. Подготовить решение заданий из ИДЗ, касающийся тематики ЛР № 1 (разработать алгоритмы (линейной структуры) решения заданий, написать их программные реализации, апробировать их на компьютере, получить результаты и их проанализировать), подготовить отчет по ЛР № 1, касающийся выполнения ИДЗ, и защитить данный отчет, сдав ЛР № 1.

Лабораторная работа № 2
ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ РАЗВЕТВЛЯЮЩЕЙСЯ СТРУКТУРЫ (РАЗВЕТВЛЯЮЩИХСЯ ПРОЦЕССОВ)

Цель работы:

1. Дальнейшее изучение методов и приемов программирования на алгоритмическом языке Бэйсик (версии Quick Basic, Free Basic).

2. Программирование алгоритмов разветвляющихся процессов и структур. Приобретение навыков в составлении программ условных алгоритмов.

3. Дальнейшее изучение конкретной интегрированной среды разработки (среды программирования) (версии Quick Basic, Free Basic) и приемов отладки программ.

Программирование алгоритмов разветвляющихся процессов

Алгоритм разветвляющейся структуры - алгоритм, в котором последовательность выполнения действий зависит от результатов проверки каких-либо условий.

В языке Бэйсик (и его версиях Quick Basic, Free Basic) для ветвления (разветвления), то есть для изменения последовательности выполнения операторов программы используются нижеследующие конструкции языка программирования:

1.Оператор безусловной передачи управления

GOTO N , где N метка (или номер) строки.

Этот оператор передает управление строке с меткой N.

Замечание. Следует реже применять операторы типа **GOTO**, так как большое число подобных операторов затрудняет отладку программы (лучше использовать другие средства языка, например подпрограммы).

2. Операторы условной передачи управления используются для изменения хода выполнения программы в зависимости от результата проверки условий. Известны несколько видов форматов записи (синтаксиса):

а) однострочный формат (однострочный IF):

IF < логическое выражение **> THEN <** операторы **>.**

При выполнении условного оператора **IF** этой модификации сначала определяется результат логического выражения: ИСТИНА (TRUE) или ЛОЖЬ (FALSE). Если ИСТИНА, то управление передается операторам, следующим за словом THEN (может быть один оператор или более операторов, разделенных двоеточием и расположенных в одной логической строке), если ЛОЖЬ - то оператору, записанному в программе после оператора **IF**;

IF <логическое выражение**>THEN<** операторы **>ELSE<**операторы**>.**

При выполнении условного оператора **IF** данной модификации сначала определяется результат логического выражения: ИСТИНА (TRUE) или ЛОЖЬ (FALSE). Если ИСТИНА, то управление передается операторам, следующим за словом THEN (может быть один оператор или более операторов, разделенных двоеточием и расположенных в одной логической строке), если ЛОЖЬ - то оператору или нескольким операторам, разделенных двоеточием и расположенных в одной логической строке, записанному(ым) в после ELSE;

б) блочный (блоковый) формат (блочный IF) (записывается в нескольких строках):

IF < логическое выражение 1> **THEN**

< операторы >

[ELSEIF < логическое выражение > **THEN**

< операторы >

ELSE

< операторы > **]**

END IF

В записи структуры блокового IF квадратные скобки означают необязательность элемента, заключенного в “[ ]”.

При выполнении блочного IF сначала определяется результат первого логического выражения 1. Если ИСТИНА, то управление передается операторам, следующим за первым словом THEN, а затем к строке, следующей за END IF. Если ЛОЖЬ, то определяется результат следующего логического выражения 2, и в случае ИСТИНЫ управление передается операторам, записанным за следующим THEN, а потом к строке, идущей за END IF и т.д. Если же ни одно из условий оператора не выполняется, то выполняются операторы, записанные после слова ELSE, а потом к строке, следующей за END IF.

Логические выражения состоят из числовых или текстовых данных, знаков отношений и логических операций. Информация представлена в таблице 9.

Таблица 9

ЗНАКИ СРАВНЕНИЯ		**ЛОГИЧЕСКИЕ ОПЕРАЦИИ**	
Название знака	В программе	Название операции	В программе
Равно	=	Логическое умножение	AND
Не равно	<>		
Больше	>	Логическое сложение	OR
Больше или равно	>=		
Меньше	<	Логическое отрицание	NOT
Меньше или равно	<=		

Блочный оператор **IF** применяется тогда, когда нужно при выполнении заданного (заданных) условия (ий) выполнить не один оператор, а несколько операторов (или группы операторов). Блочный условный оператор имеет значительные преимущества перед однострочным условным оператором:

- возможность располагать блоки операторов в нескольких строках;
- возможность сложного ветвления программы;
- более простое написание и отладка программ.

Еще одной структурой разветвления является оператор ветвления (вариант) **SELECT CASE.**

SELECT CASE *Выражение для проверки* - ‘ “ключ”
CASE *Список 1*
Блок 1
CASE *Список 2*
Блок 2
……..
CASE *Список n*
Блок n
[CASE ELSE
Блок m **]**
END SELECT

Параметр *Выражение для проверки* – любое числовое или строковое выражение, в зависимости от значения которого производится выполнение одного из блоков операторов *Блок 1, Блок 2* и т.д. Каждый из этих блоков может содержать любое число операторов в или более строках. Параметры *Список 1, Список 2* и т.д. – списки выражений (могут быть, например, как одиночными числовыми значениями, так и списками – несколькими числами, разделенными запятыми или др.) соответствующие своему ключевому слову CASE и имеющие одну из следующих форм:

- *выражение* [, *выражение*,…]
- *выражение* TO *выражение*
IS *операция отношения*

Параметр *выражение* – любое числовое или строковое выражение, тип которого должен совпадать с типом проверяемого выражения. Параметр

операция отношения задает любую из операций («меньше», «меньше или равно», «больше», «больше или равно», «не равно», «равно» - см таблицу 9).

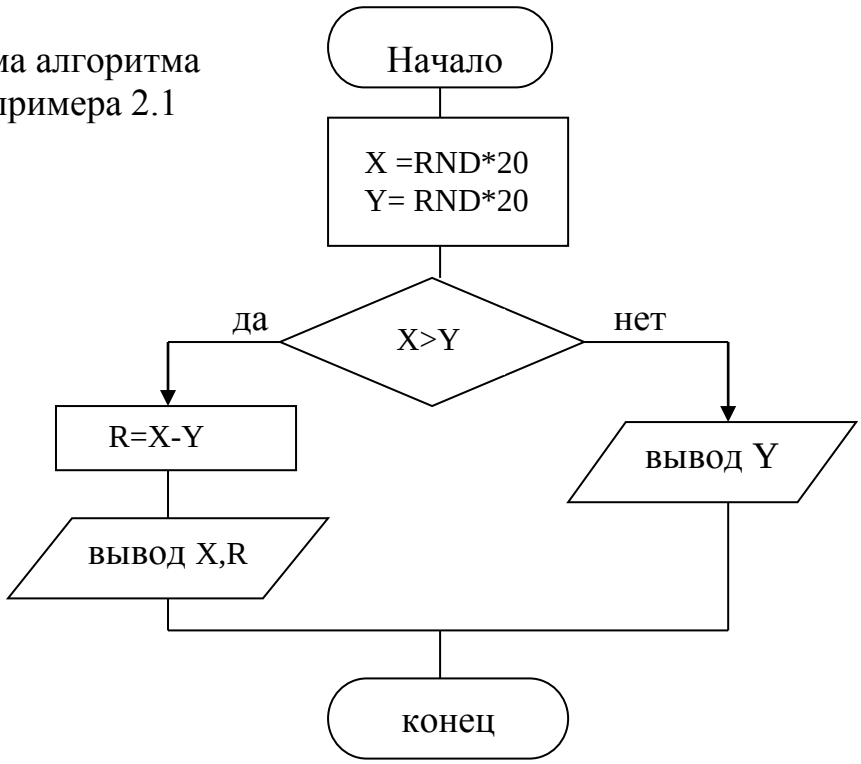
Если проверяемое выражение совпадает с каким-либо элементом списка выражений (*Список 1, Список 2* и т.д.) приведенного вслед за ключевым словом CASE, то выполняется блок операторов вплоть до следующего ключевого слова CASE или END SELECT оператора. Если задан диапазон величин для сравнения с проверяемым выражением, то меньшая величина должна стоять слева от ключевого слова TO. Операторы отношения могут использоваться только совместно с ключевым словом IS.

При применении варианта CASE ELSE соответствующий блок операторов (если он существует) выполняется, если ранее совпадения не произошло. Этот вариант рекомендуется для обработки непредвиденных ситуаций. CASE ELSE можно опустить, если точно известно, что все возможные ситуации исчерпаны (что отражено в структуре оператора возможным наличием квадратных скобок).

Операторы SELECT CASE….END SELECT могут быть вложенными.

Пример 2.1. Из двух случайных чисел X,Y вывести наибольшее и, если X>Y, также вывести разность этих чисел.

Рис. 4. Схема алгоритма решения примера 2.1




```
REM Пример 2.1. Программирование ветвящегося алгоритма
CLS
RANDOMIZE TIMER            ' Запущен датчик случайных чисел
X=RND*20: Y=RND*20         ' X, Y - случайные числа
IF X>Y THEN R=X-Y: PRINT "X="; X, "X-Y="; R ELSE PRINT "Y=";Y
END
```

Пояснения к программе:

В данной программе использован оператор IF - однострочная форма. При выполнении условия X>Y после THEN в соответствии со схемой алгоритма записаны два оператора, разделенные двоеточием - оператор присваивания R=X-Y (разность чисел X и Y) и вывода результатов. Если условие не выполняется, то выводится Y после ELSE.

Пример 2.2. Составить схему алгоритма и программу вычисления и печати функции F(x) для заданного значения x:

$$F(x) = \begin{cases} \sin x, & \text{если } x \le a \\ \cos x, & \text{если } a < x < b \\ \operatorname{tg} x, & \text{если } x \ge b \end{cases}$$

На рис.5 представлена схема алгоритма решения примера 2.2.

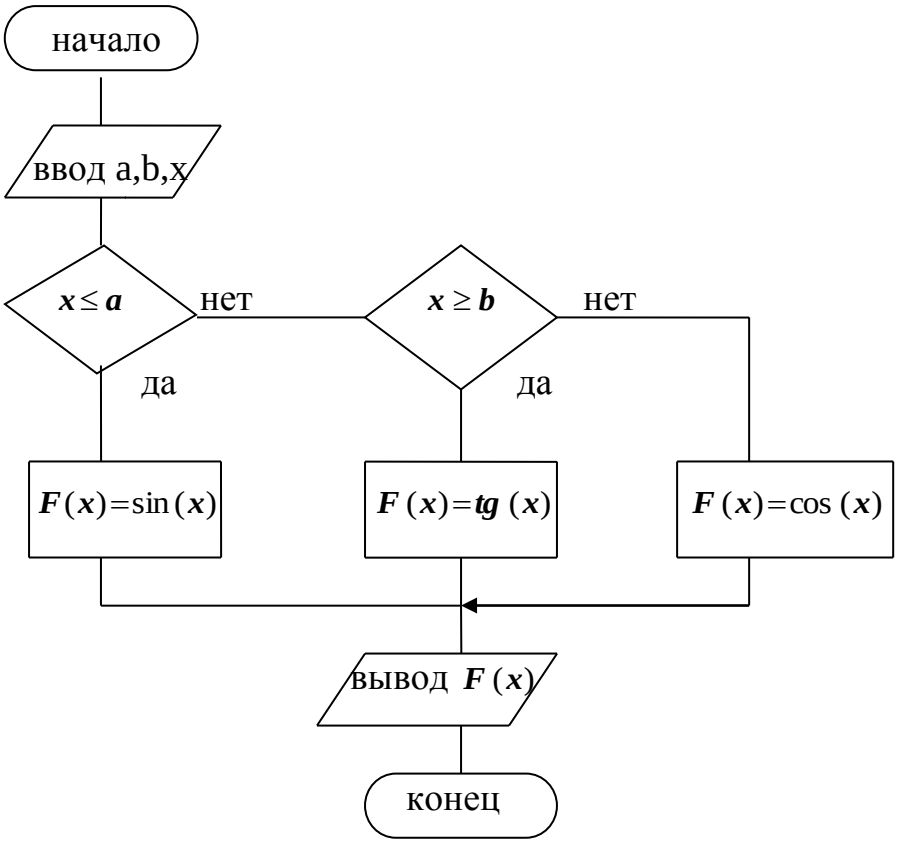


Рис. 5. Схема алгоритма решения примера 2.2

```
REM Пример 2.2 Программирование ветвящегося алгоритма
CLS
INPUT " Введите значения переменных A,B,X через запятые" , A,B,X
IF X <= A THEN
F = SIN (X)
ELSEIF  X >= B THEN
F = TAN (X)
ELSE
F = COS (X)
END IF
PRINT "X ="; X,"F ="; F
END
```

Пояснения к программе:

В программе для организации ввода данных в диалоговом режиме используется только оператор " INPUT " (сравните с примером 1.1).

" IF " - блочный обеспечивает ветвление. В зависимости от введенных значений переменных A, B, X процесс вычисления F пойдет в соответствии с алгоритмом (рис. 5) по одной из ветвей.

Ниже приводятся практические задания с образцами примеров типовых заданий по программированию условных алгоритмов. По этим типовым заданиям студенты в рамках самостоятельной работы студентов (обучающихся) – СРС или СРО) разрабатывают алгоритмы (составляют схемы) решения этих задач и их программные реализации (программные коды) в конкретной среде программирования, апробируют на компьютере и сохраняют в электронном виде (в отдельных файлах) на носителе.

Задание 1.

1. $F(x,y) = \begin{cases} \ln(x+y), & \text{если } x > y; \\ \max(7x,4y), & \text{если } x < y; \\ 1/x + 1/y, & \text{если } x = y. \end{cases}$

Задание 2.

1. Определить, равна ли сумма первых двух цифр четырехразрядного числа **a** сумме его последних цифр.

Задание 3.

1. Дано натуральное число **a.** Определить, является оно полным квадратом или кубом.

Лабораторное задание

1. Продолжать изучение и освоение различных режимов окна редактирования соответствующего экранного редактора конкретной интегрированной среды.

2. Выполнить приведенные в лабораторной работе № 2 типовые задания: написать алгоритм (условные алгоритмы) решения каждого примера и его программную реализацию; сохранить каждую программу каждого типового задания в отдельном файле (программном файле); каждую программу апробировать (отладить) на компьютере; получить результаты вычислений (“машинного счета”).

3. Подготовить решение заданий из ИДЗ, касающийся тематики ЛР № 2 (разработать алгоритмы (условные алгоритмы) решения заданий, написать их программные реализации, апробировать их на компьютере, получить результаты и их проанализировать), подготовить отчет по ЛР № 2, касающийся выполнения ИДЗ, и защитить данный отчет, сдав ЛР № 2.

Лабораторная работа № 3
ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ЦИКЛИЧЕСКОЙ СТРУКТУРЫ (ЦИКЛИЧЕСКИХ ПРОЦЕССОВ).

Цель работы:

1. Дальнейшее изучение методов и приемов программирования на алгоритмическом языке Бэйсик (версии Quick Basic, Free Basic).

2. Программирование алгоритмов циклической структуры (циклических процессов). Приобретение навыков и приемов составления программ циклических алгоритмов.

3. Дальнейшее изучение конкретной интегрированной среды разработки (среды программирования) (версии Quick Basic, Free Basic) и приемов отладки программ.

Циклические алгоритмы

Алгоритм называется циклическим, если все или отдельные его этапы в процессе решения задачи неоднократно повторяются. Известны математические и программные циклические процессы. Программные циклы подразделяются на арифметические и итерационные (интерактивные, итеративные) циклы.

Цикл обеспечивает повторное выполнение или, иначе говоря, циклическую работу операторов. Оператор или группа операторов, повторяющаяся в цикле, называется "телом цикла".

Арифметические циклы – циклы с известным заранее числом повторений цикла (числом выполнений). Итерационные (итеративные) циклы – циклы с неизвестным заранее числом повторений (числом выполнений).

Рассмотрим два типа циклических задач:

а) задачи, в которых вычисления многократно ведутся по одним и тем же формулам с различными значениями входящих в нее величин. Такие задачи иногда называются задачами на табулирование;

б) задачи, где значение некоторой величины вычисляется через значение этой же величины, полученное в предыдущем цикле (рекурсии). Примерами таких задач являются задачи вычисления сумм и произведений рядов, а также вычисление значений факториала.

Характерные моменты циклического алгоритма:

- первоначальный вход в цикл выполняется через блок подготовки;
- цикл всегда характеризуется некоторой переменной, называемой параметром цикла. Начальное значение параметра задается перед циклом в блоке подготовки, а при каждом повторении цикла выполняются операторы тела цикла и параметр изменяется на определенную величину - шаг;
- число повторений цикла должно быть конечным (арифметический цикл), то есть число повторений известно или может быть вычислено заранее. Выход из цикла осуществляется при выполнении некоторых условий. Когда число повторений известно или может быть определено заранее, выход их цикла осуществляется при достижении параметром некоторой заранее заданной величины.

Для вышеназванных задач (их решения) используют оператор цикла типа пересчета:

FOR I=K1 **TO K**2 **STEP K**3
Операторы "тела цикла"
NEXT I

Здесь:

- I – параметр цикла (переменная целого или вещественного типа, управляющая переменная) (счетчик цикла);
- K1 – начальное значение параметра цикла (переменная или число);
- K2 – конечное значение параметра цикла (переменная или число);
- K3 – шаг изменения (модификация) параметра цикла (переменная или число). Если шаг не указан, то по умолчанию он принимается равным единице (и его можно опустить).

Начальное и конечное значения - числа, задающие границы интервала изменения параметра цикла.

Этот оператор многократно выполняет ("повторяет действия") операторы "тела цикла", находящиеся между FOR и NEXT для всех значений параметра цикла I (из интервала, диапазона значений) от K1 до K2. Повторение продолжается до тех пор, пока счетчик (параметр цикла) не достигнет конечного значения. Структура самого оператора FOR….NEXT включает и подготовку, и изменение параметра цикла, и проверку условия выхода из цикла. Различают циклы с "прямым" и "обратным" счетчиком.

Проиллюстрируем использование оператор цикла "**FOR….NEXT"** примерами.

Пример 3.1 (задача типа а)). Составить схему алгоритма и программу вычисления всех значений функции F(x) для всех значений аргумента x:

$F(x) = \sin(x) + \cos(x)$, при $-\pi \le x \le \pi$, шаг изменения аргумента $\Delta x = 0.1$.

На рис. 6 приведен алгоритм решения примера 3.1.

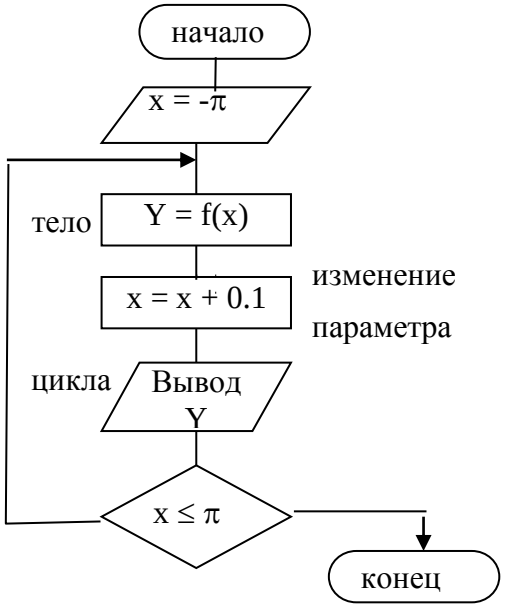


Рис. 6. Схема алгоритма решения примера 3.1

```
REM ПРИМЕР 3.1 Циклический
REM алгоритм
   CLS
   CONST PI=3.14
   PRINT "  X ", " Y "
 FOR  X= - PI TO PI STEP 0.1
      Y= SIN(X)+COS(X)
       PRINT X ,Y
   NEXT X
                END
```

Пояснение к программе. В цикле FOR многократно вычисляются значения функции для всех значений аргумента. Значения аргумента и проверка условия выхода из цикла осуществляется самим оператором FOR.

Пример 3.2 (задача типа б)). Вычислить сумму n слагаемых ряда:

$$S = 1 + \frac{1}{3} + \frac{1}{6} + \frac{1}{9} + ... + \frac{1}{3n} = 1 + \sum_{i=1}^{n} \frac{1}{3i}$$

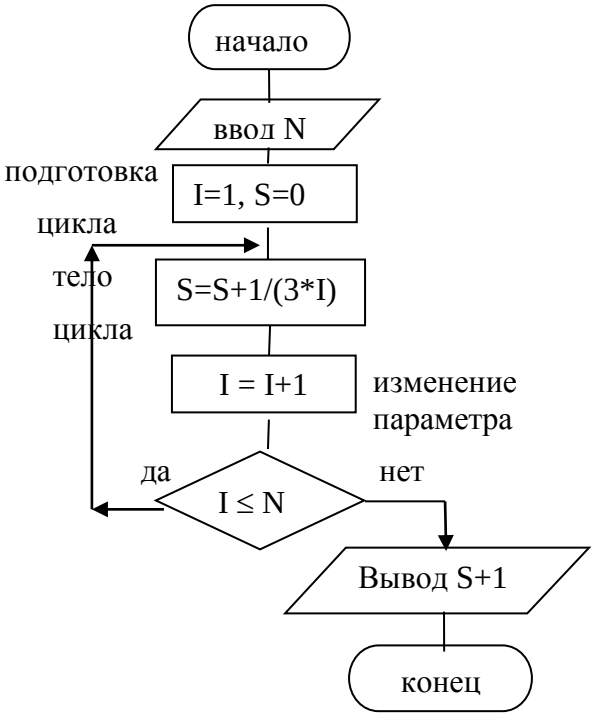


Рис. 7. Схема алгоритма решения примера 3.2

```
REM Цикл с известным числом
REM повторений
INPUT "Число слагаемых:", N
S = 1
FOR I=1 TO N
S=S+1/(3*I)
NEXT I
PRINT "       Сумма ряда :"; S+1
END
```

Результат выполнения на экране:

Сумма ряда: < число >

Пример 3.3 (задача типа б)). Вычислить факториал n чисел:

P= n!=1*2*3*….*n

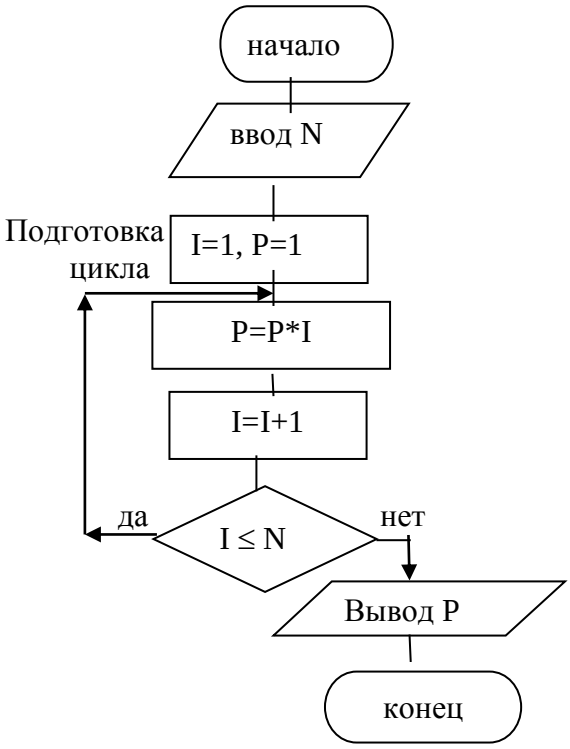


Рис. 8. Схема алгоритма решения примера 3.3

```
REM Цикл с известным числом
REM повторений
INPUT "Число сомножителей:", N
P = 1
FOR I=1 TO N
P=P*I
NEXT I
PRINT "       N факториал :"; P
END
```

Результат выполнения на экране:

N факториал : < число >

<u>Пояснение к программам 3.2, 3.3.</u>

С помощью циклов в этих программах последовательно вычисляется значение суммы и факториала (произведения). При этом каждое следующее значение вычисляется через предыдущее. Результатом многократных вычислений является одна величина - сумма ряда заданного числа слагаемых (пример 3.2) или произведение заданного числа сомножителей (пример 3.3).

В приведенных примерах 3.1 - 3.3 число повторений легко определяется заранее. В примере 3.1 параметром цикла является переменная "x", а в примерах 3.2 и 3.3 - переменная "I".

Когда условие задачи не позволяет определить заранее число повторений цикла, выход из цикла зачастую осуществляется по достижении заданной точности вычислений или по какому-либо другому условию, оговоренному в задании. Тогда имеем дело с циклами с неизвестным заранее числом повторений (итерационными, итеративными циклами). В этих случаях удобно использовать следующие операторы цикла:

а) **DO WHILE L** < операторы «тела цикла» > **LOOP**	б) **DO UNTIL L** < операторы «тела цикла» > **LOOP**

в) **DO** < операторы «тела цикла» > **LOOP WHILE L**	г) **DO** < операторы «тела цикла» > **LOOP UNTIL L**

Здесь **L** – логическое выражение (условие).

Операторы, находящиеся между DO и LOOP повторяются до тех пор, пока выражение L, стоящее после WHILE – истинно или до тех пор, пока выражение L, стоящее после UNTIL – ложно. Оператор цикла типа а) – оператор цикла (повторений) с предусловием; оператор цикла типа в) – оператор цикла (повторений) с постусловием. Операторы "тела цикла" в циклических структурах типа а) и в) - повторяются до тех пор, пока условие (логическое выражение) **L** остаётся *истинным*. Когда условие становится *ложным*, то названные циклы а) и в) прекращают работу. Другие конструкции: оператор цикла типа б) – оператор цикла (повторений) с предусловием; оператор цикла типа г) – оператор цикла (повторений) с постусловием. Операторы "тела цикла" в этих других циклических структурах типа б) и г) - повторяются до тех пор, пока условие (логическое выражение) **L** не станет *истинным* (т.е. циклы повторяются, пока условие L – *ложно*).

Пример 3.4. Вычислить сумму S ряда с заданной точностью e и количество слагаемых N:

$$\boldsymbol{S} = 1 + \frac{1}{3} + \frac{1}{6} + \frac{1}{9} + ... + = 1 + \sum_{\boldsymbol{i}=1}^{\infty} \frac{1}{3\boldsymbol{i}} .$$

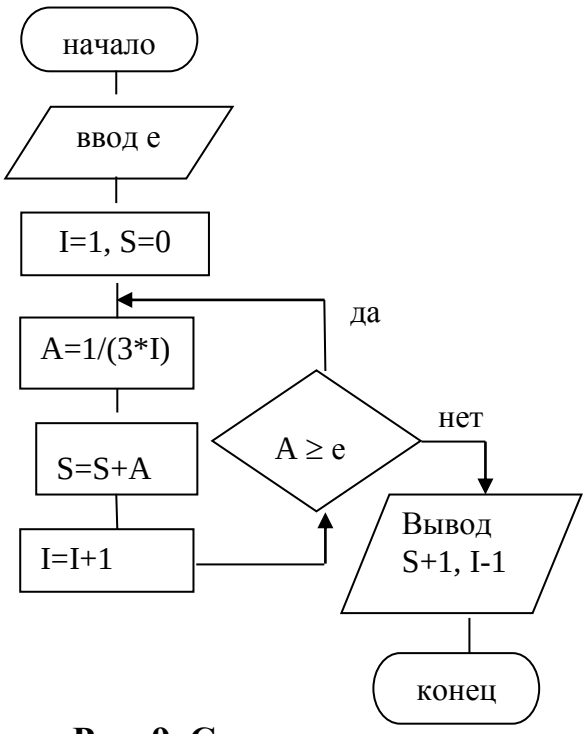


Рис. 9. Схема алгоритма решения примера 3.4

```
REM Цикл с неизвестным числом
REM повторений
CLS
INPUT "Введите точность", e
S = 0: I=1
DO
A = 1/( 3*I ) ' слагаемое
 S = S+A :    I=I+1
LOOP WHILE A >= e
PRINT "Сумма ряда:"; S+1,"N="; I-1
END
```

Результат выполнения на экране:
Сумма ряда:<число> N=<число>

Вычислить сумму ряда с точностью **e** - это значит производить вычисления до тех пор, пока очередной член последовательности не станет меньше или равен заданной точности. В примерах 3.2 и 3.4 в обоих случаях требуется вычислить сумму ряда. В первом случае количество слагаемых определено условием задачи, а во втором - слагаемое последовательно в цикле прибавляется к сумме до тех пор, пока оно (слагаемое) не станет меньше или равно некоторого маленького числа (заданной точности). Такая постановка задачи имеет смысл только для **сходящихся** рядов, т.е. слагаемое должно стремиться к нулю.

Далее приведено еще несколько примеров циклических программ.

Пример 3.5. Найти максимум функции Y=sin (x)*exp(X+2.5) / (X-0.9) на интервале -5.1 < X < 3.

```
CLS
'МАКСИМУМ ФУНКЦИИ ОБОЗНАЧИМ ПЕРЕМЕННОЙ  YMAX
'ЗАДАДИМ НАЧАЛЬНОЕ ЗНАЧЕНИЕ ПЕРЕМЕННОЙ YMAX
YMAX = -1E+19
FOR X = -5.1 TO 3 STEP .5              'СРАВНИМ В ЦИКЛЕ ТЕКУЩЕЕ
Y = SIN(X) * EXP(X + 2.5) / (X - .9)   'ЗНАЧЕНИЕ ФУНКЦИИ Y  C  YMAX
 IF Y > YMAX THEN YMAX = Y       'И ЕСЛИ YMAX ОКАЖЕТСЯ МЕНЬШЕ
                                 ‘’ТО ЕГО  ЗНАЧЕНИЕ  ЗАМЕНЯЕТСЯ НА ТЕКУЩЕЕ
NEXT X
 PRINT
 PRINT "YMAX="; YMAX       ' ВЫВОДИМ ЗНАЧЕНИЕ МАКСИМУМА ФУНКЦИИ
 END
```

Пример 3.6. Определить, является ли случайное число X простым.

```
CLS
RANDOMIZE TIMER
X=FIX(RND*100)
FOR j = 2 TO X\2
‘В ЦИКЛЕ ОПРЕДЕЛЯЕМ, ДЕЛИТСЯ ЛИ ВВЕДЕННОЕ ЧИСЛО НА КАКОЕ-ЛИБО
 'ДРУГОЕ,  КРОМЕ 1 И САМОГО СЕБЯ БЕЗ ОСТАТКА
 IF X  MOD  j = 0 THEN  20                'ЕСЛИ ДЕЛИТСЯ, ТО ОНО НЕ ПРОСТОЕ
NEXT j
PRINT USING " простое число: ####"; X
GOTO 50
20 PRINT X, " - это число не простое"
50 END
```

Сложные циклы

Цикл называется сложным, если он содержит в себе другой, вложенный в него цикл. Количество вложенных друг в друга циклов (глубина вложений) может быть достаточно большим. Каждому циклу соответствует свой параметр. Типы циклов, из которых образован сложный, могут быть различными, это зависит от конкретной задачи. Первоначальный вход в любой цикл допустим только через блок подготовки соответствующего цикла. В общем виде схема алгоритма сложного цикла приведена на рис. 10. Тело цикла включает в себя операторы соответствующего цикла. Причем, для каждого значения параметра внешнего цикла параметр внутреннего цикла пробегает все свои значения.

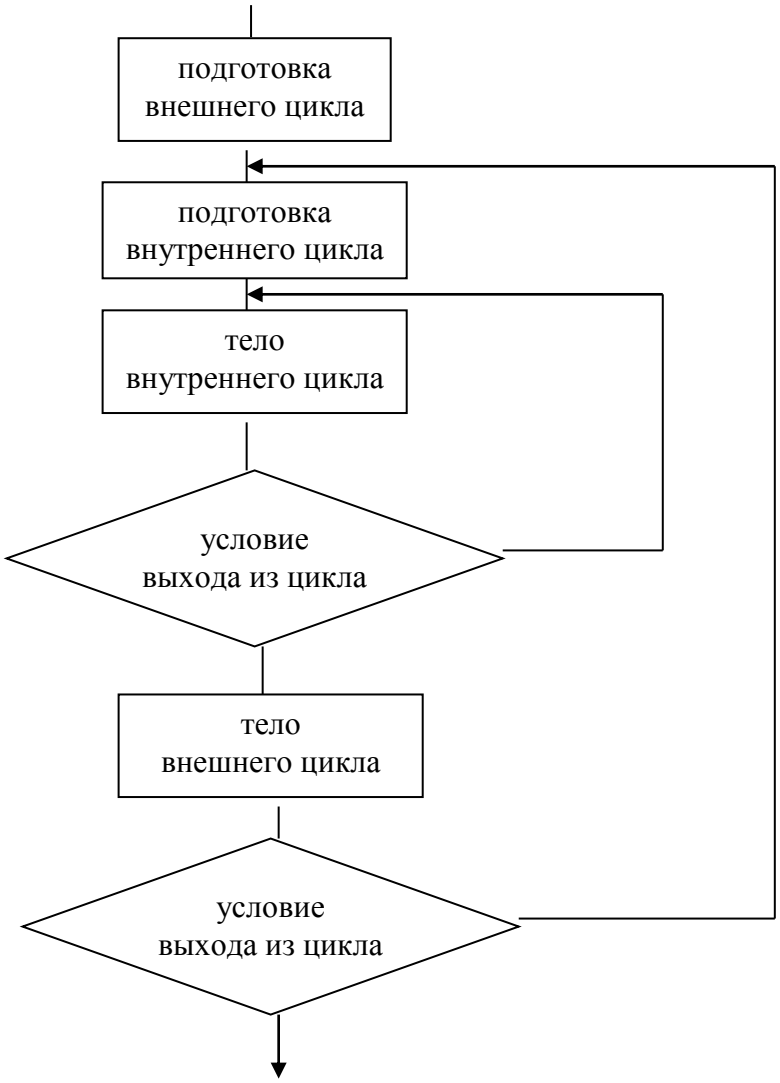


Рис. 10. Схема алгоритма сложного цикла глубиной два

Ниже приводятся практические задания с образцами примеров типовых заданий по программированию циклических алгоритмов. По этим типовым заданиям студенты в рамках самостоятельной работы студентов (обучающихся) – СРС или СРО) разрабатывают алгоритмы (составляют схемы) решения этих задач и их программные реализации (программные

коды) в конкретной среде программирования, апробируют на компьютере и сохраняют в электронном виде (в отдельных файлах) на носителе.

Задание 1.

1. Числа Фибоначчи f_n определяются следующим образом: $f_0=f_1=1$; $f_n=f_{n-1}+f_{n-2}$. Определить сумму чисел Фибоначчи, не превосходящих некоторого заданного числа a.

2. Даны натуральные числа n, m. Определить НОД (наибольший общий делитель) этих чисел с помощью алгоритма Евклида.

Задание 2.

Написать схему алгоритма и программу вычисления в двух вариантах:

а) вычислить сумму ряда для заданного количества слагаемых N. Значения переменных задать в диалоговом режиме самостоятельно. На экран вывести значение суммы ряда;

б) вычислить сумму ряда с заданной точностью ε. Значения переменных задать в диалоговом режиме самостоятельно. На экран вывести значение суммы ряда и количество повторений.

1. $S = 1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{8} + \frac{1}{16} - .... + (-1)^n \frac{1}{2^n}.$

2. $S = 1 + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{1 \cdot 2 \cdot 3 \cdot 4} + \frac{1}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5} + ... + \frac{1}{N!}.$

Задание 3.

1. Дано натуральное число n. Вывести такие натуральные неотрицательные числа a, b, c, d, что a^2+ b^2+ c^2+ d^2 = n. (Это справедливо для любого n согласно теореме Лагранжа).

2. Дано натуральное число n. Вывести все *совершенные* числа, меньшие n. (Число является совершенным, если оно равно сумме его делителей, кроме себя самого).

Лабораторное задание

1. Продолжать изучение и освоение различных режимов окна редактирования соответствующего экранного редактора конкретной интегрированной среды.

2. Выполнить приведенные в лабораторной работе № 3 типовые задания: написать алгоритм (циклические структуры) решения каждого примера и его программную реализацию; сохранить каждую программу каждого типового задания в отдельном файле (программном файле); каждую программу апробировать (отладить) на компьютере; получить результаты вычислений (“машинного счета”).

3. Подготовить решение заданий из ИДЗ, касающийся тематики ЛР № 3 (разработать алгоритмы (циклические структуры) решения заданий, написать их программные реализации, апробировать их на компьютере, получить результаты и их проанализировать), подготовить отчет по ЛР № 3, касающийся выполнения ИДЗ, и защитить данный отчет, сдав ЛР № 3.

Лабораторная работа № 4
ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ РАБОТУ С МАССИВАМИ ДАННЫХ И ИХ ОБРАБОТКУ

Цель работы:

1. Дальнейшее изучение методов и приемов программирования на алгоритмическом языке Бэйсик (версии Quick Basic, Free Basic).

2. Программирование алгоритмов циклической структуры (циклических процессов), связанных с работой с массивами, их обработкой. Изучение и приобретение навыков и приемов составления программ с использованием массивов, программированием циклических алгоритмических структур обработки массивов.

3. Дальнейшее изучение конкретной интегрированной среды разработки (среды программирования) (версии Quick Basic, Free Basic), приемов отладки программ, закрепление навыков работы в отладочных режимах среды(д) разработки.

Массив – организованная группа или таблица величин, имеющая имя для ссылки на массив и совокупность индексов для ссылки на элементы массива. Индекс является целым числом или выражением и помещается в круглые скобки вслед за именем массива. Следует различать размер и число размерностей массива. Число размерностей массива эквивалентно числу составляющих вектора. Размер массива определяет объём занимаемой им оперативной памяти компьютера и зависит от типа и количества (числа индексов) хранимых в нем данных. Положение элемента в массиве определяется индексами: одним - для одномерных массивов, двумя - для двумерных (матриц) и т.д. Максимальное число размерностей в массиве – 255. Максимальное количество элементов каждой размерности (максимальное значение каждого индекса) – 32767. Массивы подразделяются: массивы статические и динамические, массивы фиксированной и переменной длины. **Массив** - совокупность данных *одного типа*, обозначаемая одним именем. В зависимости от типа данных массивы могут быть как числовыми, так и текстовыми.

Имя массива образуется также, как имя простой переменной. Индексы заключаются в круглые скобки и разделяются запятой, если массив не одномерный. В качестве индексов могут быть числа, переменные или арифметические выражения, значения которых автоматически округляются до целого. Если индексы не числовые, то их значения должны быть определены заранее.

Примеры обозначения в Бэйсике элементов массивов:

AQ(33), AQ(I), AQ(I + 4/3) - для одномерного массива;

AD(12,3), AD(I,J), AD(I/2,J+3) - для двумерных массивов**.**

Если число индексов массива по любой его размерности не превышает 10, то массив можно не объявлять. Если превышает 10, то такой массив должен быть объявлен оператором **DIM** заранее. Так справедливо для Quick

Basic. Для Free Basic независимо от величины размерности массив должен объявляться оператором: **DIM**, **REDIM** (заранее, явно).

В операторе DIM указываются имена массивов и в круглых скобках верхние и нижние границы изменения индексов, которые должны быть целыми положительными числами или переменными, значение которых определено в программе ранее.

Если в процессе выполнения программы значение индекса превысит верхнюю границу массива, то система выдаст сообщение ***Subscript out of range*** (Индекс вне диапазона).

Значение нижней границы индексов может быть опущено, и, тогда по умолчанию, оно принимается равным нулю.

При обозначении двумерных массивов индекс строки стоит на первом месте, индекс столбца - на втором.

В Бейсике обработка массивов осуществляется поэлементно, в том числе и ввод - вывод массива. Если массив содержит всего несколько элементов, то задать их значения можно с помощью операторов присваивания:

```
DIM Q(4)
Q(1)=0.25: Q(2)=0.12: Q(3)=0.35: Q(4)=0.28
```

или с помощью оператора ввода:

```
DIM Q(4)
INPUT Q(1), Q(2), Q(3), Q(4)
```

Аналогичным образом осуществляется и вывод массива:

```
PRINT Q(1), Q(2), Q(3), Q(4)
```

В том случае, когда массив содержит много элементов и перечисление их при вводе - выводе становится неудобным, то **организуется цикл**.

Далее приведем ряд примеров с программами по данной тематике.

```
REM ВВОД  МАССИВА с помощью
' оператора " INPUT "
INPUT n
DIM Q(n)
FOR I=1 TO n
INPUT  Q(I)
NEXT I
```

```
REM ВВОД МАССИВА с помощью
'оператора " INPUT " из файла
OPEN "d:\dan.dat" FOR INPUT AS #1
INPUT n
DIM Q(n)
FOR I=1 TO n
INPUT #1, Q(I)
NEXT I
```

По запросу " INPUT" после ввода каждого элемента массива нужно нажать клавишу **Enter.**

REM ВВОД МАССИВА с помощью 'оператора " READ " DATA 5,7,12,2,0,7,23,6,4,8,1 DIM Q(11) FOR I=1 TO 11 READ Q(I) NEXT I	REM ВВОД МАССИВА с помощью ' датчика случайных чисел RANDOMIZE TIMER INPUT n DIM Q(n) FOR I=1 TO n Q(I)= FIX(RND*60) NEXT I
'ВЫВОД МАССИВА НА ЭКРАН FOR I=1 TO N PRINT Q(I); NEXT I	'ВЫВОД МАССИВА В ФАЙЛ OPEN "d:\rez.dat" FOR OUTPUT AS #2 FOR I=1 TO N PRINT #2,Q(I); NEXT I

Оператору "READ" в программе должен сопутствовать оператор "DATA", в котором перечисляются значения всех элементов массива через запятую. При использовании функции "RND" рекомендуется предварительно запустить датчик случайных чисел "RANDOMIZE TIMER"

Пример 4.1. Нахождение максимального элемента массива.

```
   'НАХОЖДЕНИЕ МАКСИМАЛЬНОГО ЭЛЕМЕНТА МАССИВА
DIM X(15)
  'ВВЕДЕМ ПЕРЕМЕННУЮ YMAX ДЛЯ ХРАНЕНИЯ В НЕЙ ЗНАЧЕНИЯ
  'МАКСИМАЛЬНОГО ЭЛЕМЕНТА В МАССИВЕ:
YMAX = -1E+19 'НАЧАЛЬНОЕ ЗНАЧЕНИЕ ПЕРЕМЕННОЙ YMAX
FOR I = 1 TO 15      'В ЦИКЛЕ
INPUT X(I)      ' ВВОДИМ ЭЛЕМЕНТЫ МАССИВА И СРАВНИВАЕМ ИХ С YMAX
IF X(I) > YMAX THEN     'И ЕСЛИ YMAX ОКАЖЕТСЯ МЕНЬШЕ
YMAX = X(I)     ' ТО ЕГО ЗНАЧЕНИЕ ЗАМЕНЯЕМ
N=I                ' ЗАПОМНИМ НОМЕР МАКСИМАЛЬНОГО ЭЛЕМЕНТА
END IF
NEXT I
PRINT     ' ПРОПУСКАЕМ СТРОКУ
  'ВЫВОДИМ ЗНАЧЕНИЕ МАКСИМАЛЬНОГО 'ЭЛЕМЕНТА МАССИВА И ЕГО НОМЕР:
PRINT "YMAX="; YMAX, "НОМЕР=";N
 ‘'ЕЩЕ РАЗ ВЫВОДИМ МАКСИМАЛЬНЫЙ ЭЛЕМЕНТ МАССИВА,НО ПО-ДРУГОМУ:
PRINT "X(";N;")=";YMAX
END
```

Для ввода-вывода двумерных массивов - матриц организуется сложный (вложенный - глубиной два) цикл:

'ВВОД МАССИВА A(n,m) с помощью 'оператора INPUT, построчно INPUT n, m DIM A(n, m) FOR I=1 TO n FOR J=1 TO m INPUT; A(I,J) NEXT J PRINT NEXT I	'ВВОД МАССИВА A(n,m) с помощью 'оператора INPUT по столбцам INPUT n, m DIM A(n, m) FOR I=1 TO m FOR J=1 TO n INPUT; A(J, I) NEXT J PRINT NEXT I
'ВВОД МАССИВА A(n,m) с помощью 'датчика случайных чисел и вывод 'на экран RANDOMIZE TIMER INPUT n, m DIM A(n, m) FOR I=1 TO n FOR J=1 TO m A(I,J) = FIX(RND*50)-20 PRINT A(I, J) ; NEXT J PRINT NEXT I	REM ВВОД МАССИВА с помощью 'оператора " INPUT " из файла и 'вывод на экран OPEN "d:\dan.dat" FOR INPUT AS #1 INPUT n, m DIM Q(n, m) FOR I=1 TO n FOR J=1 TO m INPUT #1, Q(I, J) PRINT Q(I, J) ; NEXT J PRINT NEXT I

Пример 4.2. Сформировать одномерный массив из максимальных элементов столбцов матрицы A (22,5). В свою очередь матрицу A получить с помощью датчика случайных чисел.

```
  ' ФОРМИРОВАНИЕ МАССИВА
CLS                    'ОЧИЩАЕМ  ЭКРАН
DIM A(22, 5), B(5)        'ОПИСЫВАЕМ ИСХОДНЫЙ МАССИВ А И ИСКОМЫЙ-В
     "ЗАПУСКАЕМ ДАТЧИК СЛУЧАЙНЫХ ЧИСЕЛ, ЧТОБЫ СФОРМИРОВАТЬ МАССИВ А
RANDOMIZE TIMER
FOR I = 1 TO 22
FOR J = 1 TO 5
A(I, J) = RND*100                              'ФОРМИРУЕМ  МАССИВ А
PRINT USING "####.#####"; A(I, J);     'И ВЫВОДИМ ЕГО ПО СТРОКАМ
NEXT J
PRINT
NEXT I
PRINT
                  'ПЕРЕБИРАЕМ ПОЛУЧЕННУЮ МАТРИЦУ А ПО СТОЛБЦАМ:
FOR J = 1 TO 5
                   'ИСПОЛЬЗУЕМ ПЕРЕМЕННУЮ AMAX ДЛЯ ХРАНЕНИЯ В НЕЙ
                'ЗНАЧЕНИЯ МАКСИМАЛЬНОГО ЭЛЕМЕНТА СТОЛБЦА МАТРИЦЫ
AMAX = A(1, J)                  'НАЧАЛЬНОЕ ЗНАЧЕНИЕ ПЕРЕМЕННОЙ
                              'AMAX:  - ЗНАЧЕНИЕ ПЕРВОГО ЭЛЕМЕНТА СТОЛБЦА
FOR I = 2 TO 22
                           'СРАВНИМ ЭЛЕМЕНТЫ СТОЛБЦА С AMAX:
IF AMAX <= A(I, J) THEN AMAX=A(I, J)       'И ЕСЛИ AMAX ОКАЖЕТСЯ
                                        'МЕНЬШЕ, ТО ЕГО  ЗНАЧЕНИЕ  ЗАМЕНЯЕМ
NEXT I
B(J) = AMAX          ' ЭЛЕМЕНТУ МАССИВА  В  ПРИСВОИМ  ЗНАЧЕНИЕ
   'МАКСИМАЛЬНОГО ЭЛЕМЕНТА СТОЛБЦА МАССИВА  А
NEXT J
FOR J = 1 TO 5                              'ЦИКЛ ВЫВОДА МАССИВА В
PRINT USING "####.#####"; B(J);     'ВЫВОД ФОРМАТИРОВАННЫЙ
NEXT J
END
```

Ниже приводятся практические задания с образцами примеров типовых заданий по программированию циклических алгоритмов с использованием массивов и их обработки. По этим типовым заданиям студенты в рамках самостоятельной работы студентов (обучающихся) – СРС или СРО) разрабатывают алгоритмы (составляют схемы) решения этих задач и их программные реализации (программные коды) в конкретной среде программирования, апробируют на компьютере и сохраняют в электронном виде (в отдельных файлах) на носителе.

Задание 1.

1. Дан массив целых чисел X(50). Сформировать из него массив Y(50), в котором первыми располагаются четные элементы массива X, а затем - нечетные элементы. Вывести оба массива.

2. Дан массив целых чисел X(50) и число K. Сформировать массив Y(50-i) элементов после i-го, (i - номер максимального элемента x_i среди $x_1,\ldots,x_k$). Вывести оба массива.

Задание 2.

1. Дана матрица X(n, m) целых чисел. Обнулить элементы ее главной диагонали, расположенные в столбцах с четной суммой элементов и строках с нечетной суммой элементов. Вывести исходную и полученную матрицы.

2. Дана матрица X(n, m) целых чисел. Обнулить те ее столбцы, в которых элементы, принадлежащие одной из главных диагоналей, являются максимальными (сами максимальные элементы не обнулять). Вывести исходную и полученную матрицы.

Задание 3.

1. Дан список N учеников класса с указанием фамилии, имени, отчества и даты рождения: число, месяц, год. Вывести (в полном формате) список учеников, родившихся между двумя заданными датами, а также количество учеников, родившихся по месяцам.

2. Дан список N торговых отделов, торгующих N однотипными товарами с указанием дневной выручки по каждому товару. Вывести номера отделов, получивших выручку больше заданной, а также номера отделов, получивших наибольшую выручку (по каждому товару в отдельности).

Лабораторное задание

1. Продолжать изучение и освоение различных режимов окна редактирования соответствующего экранного редактора конкретной интегрированной среды.

2. Выполнить приведенные в лабораторной работе № 4 типовые задания: написать алгоритм решения каждого примера и его программную реализацию; сохранить каждую программу каждого типового задания в отдельном файле (программном файле); каждую программу апробировать (отладить) на компьютере; получить результаты вычислений (“машинного счета”).

3. Подготовить решение заданий из ИДЗ, касающийся тематики ЛР № 4 (разработать алгоритмы решения заданий, написать их программные реализации, апробировать их на компьютере, получить результаты и их проанализировать), подготовить отчет по ЛР № 4, касающийся выполнения ИДЗ, и защитить данный отчет, сдав ЛР № 4.

Лабораторная работа № 5

ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ РАБОТУ СО СТРОКОВЫМ ТИПОМ ДАННЫХ (ДЕЙСТВИЯ С СИМВОЛЬНЫМИ СТРОКАМИ)

ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ РАБОТУ В ГРАФИЧЕСКОМ РЕЖИМЕ (ПОСТРОЕНИЕ ГРАФИКОВ)

Часть 1. Программирование алгоритмов, реализующих работу со строковым типом данных

Цель работы:

1. Дальнейшее изучение методов и приемов программирования на алгоритмическом языке Бэйсик (версии Quick Basic, Free Basic).

2. Программирование алгоритмов, реализующих работу со строковым типом данных, с использованием циклических структур обработки строк. Изучение и приобретение практических навыков и приемов составления программ с использованием строковых данных, а также получения опыта работы и обработки строковой информации.

3. Дальнейшее изучение конкретной интегрированной среды разработки (среды программирования) (версии Quick Basic, Free Basic), приемов отладки программ, закрепление навыков работы в отладочных режимах среды(д) разработки.

Описание и ввод строковых данных

Данными строкового типа являются строковые константы и строковые переменные. Строковая **константа** представляет собой произвольную последовательность символов, заключенную в двойные кавычки, длиной до 32567 символов, например, " Hello", "Добрый день".

Строковые **переменные** бывают переменной или фиксированной длины. Строка переменной длины (STRING) представляет собой последовательность длиной до 32567 символов из таблицы ASCII. В памяти под такую символьную переменную отводится количество байт равное количеству символов переменной плюс 4. Объявить строковый тип переменной длины можно одним из приведенных ниже способом:

1) явно - с помощью суффикса $: Hello$ = "Привет";

2) явно - с помощью операторов описания типа:

DIM Hello AS STRING;

3) неявно - с помощью оператора объявления типа данных:

DEFSTR H ' с буквы H начинаются имена переменных типа STRING

Hello = "Привет".

Строка фиксированной длины (STRING * N) представляет собой строку длиной N символов. В памяти под такую символьную переменную отводится N байт. Описать символьную переменную фиксированной длины можно таким образом:

```
DIM Hello AS STRING*12
Hello = "Привет - Hello"
PRINT " Результат:"; Hello.
```

На экран будет выведено: Результат: Привет - Hello

Строковой переменной можно присвоить значение либо с помощью оператора присваивания, например:

```
St$ ="abcde",
```

либо с помощью операторов ввода, например:

```
DATA "abcde"
READ St$
или
INPUT St$.
```

Однако при вводе значения строковой переменной оператором INPUT возникают определенные трудности. Если среди символов вводимой строки встречаются запятые, то запятая воспринимается оператором INPUT, как

разделитель. Чтобы обойти эту проблему, предусмотрена модификация оператора INPUT:

LINE INPUT St$.

Этот оператор предназначен специально для ввода в одну строковую переменную полной строки текста независимо от ее содержания.

Работа со строками. Строковые операции

Строковые выражения используются в различных операторах языка Бэйсик: присваивания, условного перехода, вывода и т.д.

Строковое выражение может содержать строковые константы, строковые переменные, вызов функций и строковые операции.

1. Операция **"+"** (конкатенация) предназначена для объединения строк. Результат операции имеет строковый тип. Например, после выполнения фрагмента программы:

L$ = "MOSCOW"
AGE$ = "We" + " live in " + L$

строковая переменная AGE$ примет значение:

We live in MOSCOW.

2. Операции сравнения (=, <>, <, >, <=, >=). Сравнение двух строк выполняется слева направо с учетом кодов ASCII. То есть сравниваются сначала коды первых символов, затем вторых и т.д. Результат операций сравнения имеет логический тип, то есть принимает значения **ДА** или **НЕТ**, например,

"A" < "B" (результат ДА);
"RA" > "RR" (результат НЕТ);
"2" > "12" (результат ДА).

Если две строки имеют различную длину, но их начальные символы совпадают, включая последний символ более короткой строки, то короткая строка считается меньшей, например:

"12.0" > "12" (результат ДА).

Строки считаются равными тогда и только тогда, когда имеют одинаковую длину и одинаковую последовательность символов, например:

"TURBO" = "TURBO" (результат ДА);
"TURBO" = " TRUBO " (результат НЕТ).

Строковые функции и операторы

Приведем наиболее часто употребляемые строковые функции:

–**ASC(St$)** в качестве результата дает числовое значение ASCII-кода первого символа в символьном выражении.

–**CHR$**(код) в качестве результата дает символ ASCII, код которого является аргументом.

–**LEN(St$)** в качестве результата дает целое число, равное длине строкового выражения St$.

–**MID$(St$, n, m)** в качестве результата дает фрагмент строки St$, длиной m, начиная с позиции n.

–*Оператор* **MID$(St$, n, m)= Zt$** заменяет фрагмент строки St$ длиной m, начиная с позиции n на строку Zt$.

–**STR$**(числовое выражение) - преобразует числовое выражения в символьное. Если его значение положительно, то к полученной строке слева добавляется пробел.

–**VAL(St$)** - преобразует строку в числовое выражение. Функция VAL является обратной по назначению функции STR$. Если первый символ аргумента - не числовой, то функция VAL возвращает ноль. Если первый символ - знак минус, то ликвидируются все левые пробелы. Если первый символ цифра, то остается один левый пробел.

– **LEFT$(St$, n)** - выделяет из выражения n левых символов.

– **RIGHT$(St$, n)** - выделяет из выражения n правых символов.

– **LTRIM$(St$)** - удаляет левые пробелы.

– **RTRIM$(St$)** - удаляет правые пробелы.

– **INSTR(n, St$, Zt$)** - определяет позицию вхождения выражения Zt$ в St$, начиная с n.

– **INKEY$** - функция анализирует информацию о нажатых клавишах. Эта функция, помещенная в цикл DO, может быть использована для создания паузы произвольной продолжительности, управления графическим объектом, выхода из цикла и т.д.

Пример 5.1 Дан текст. Удалить из текста повторяющиеся символы.

```
CLS
t$ = "ббааабоччкаа"    'Исходный текст
PRINT t$
dl = LEN(t$)            'Определяем длину строки
 i = 1                  ' Считаем символы
 DO
s$ = MID$(t$, i, 1)          ' выделяем каждый символ
ss$ = MID$(t$, i + 1, 1)     ' и следующий
IF s$ = ss$ THEN             ' если они совпадают, то
'левую часть строки, включая i-й символ, объединяем с правой, исключая (i+1)-й
t$ = LEFT$(t$, i) + RIGHT$(t$, dl - i - 1)
dl = dl - 1   ' длина уменьшится
i = i - 1      ' будем сравнивать в следующий раз тот же i-й символ с новым
(i+1)-м
END IF
i = i + 1
LOOP WHILE i < dl ' перебираем все символы по всей длине строки
PRINT t$  :  END
```

На экране будет: **бабочка**

Символ в тексте определяется номером позиции, которую он занимает; переход к следующему символу осуществляется изменением номера позиции на 1. Выбирать из текста можно как по одному символу, так и по несколько

следующих друг за другом. Тексты могут содержать и числовую информацию, заданную в символьной форме.

Пример 5.2. С клавиатуры вводится список фамилий. Распечатать список в алфавитном порядке.

```
' Задача представляет собой  задачу сортировки одномерного массива
CLS
INPUT n
DIM fio(n) AS STRING
FOR i = 1 TO n
INPUT fio(i)
 NEXT
 FOR i = 1 TO n - 1
 FOR j = 1 TO n - i
' Если  фамилии не по алфавиту, то меняем  их местами
' с помощью оператора "SWAP"
 IF fio(j) > fio(j + 1) THEN SWAP fio(j), fio(j + 1)
 NEXT
 NEXT
 FOR i = 1 TO n
 PRINT fio(i)
 NEXT
 END
```

Пояснение к программе: оператор SWAP A,B - обменивает величины двух переменных одного типа.

Ниже приводятся практические задания (для **части 1** ЛР № 5) с образцами примеров типовых заданий по программированию алгоритмов, реализующих работу со строковым типом данных, с использованием циклических структур обработки строк. По этим типовым заданиям студенты в рамках самостоятельной работы студентов (обучающихся) – СРС или СРО) разрабатывают алгоритмы (составляют схемы) решения этих задач и их программные реализации (программные коды) в конкретной среде программирования, апробируют на компьютере и сохраняют в электронном виде (в отдельных файлах) на носителе.

Задание 1.

1. Даны строки a$ и b$. Определить, имеются ли в строке a$ все символы, из которых состоит строка b$ и вывести их.

2. В исходной строке a$ определить наличие и количество пар соседних одинаковых символов и удалить их. Вывести строку a$ до и после модификации, а также найденное число пар.

Задание 2.

1. В исходной строке a$ заменить каждое нечетное вхождение заданного сочетания символов x$ на соответствующее число пробелов. Вывести исходную и полученную строки.

2. В исходной строке a$ удвоить символы, входящие в первую половину алфавита и удалить все символы второй половины алфавита. Вывести строку до и после изменения.

Задание 3.

1. Из заданной в виде строки a$ совокупности чисел с дробной частью (разделитель целой и дробной части - точка, разделитель чисел - пробел) определить сумму целых частей чисел и вывести в текстовой форме.

3. Рассортировать слова исходной фразы a$ по алфавиту (по заданному номеру N буквы в слове).

Лабораторное задание (к части 1 ЛР № 5)

1. Продолжать изучение и освоение различных режимов окна редактирования соответствующего экранного редактора конкретной интегрированной среды.

2. Выполнить приведенные в лабораторной работе № 5 (в части 1) типовые задания (по работе со строковыми данными): написать алгоритм решения каждого примера и его программную реализацию; сохранить каждую программу каждого типового задания в отдельном файле (программном файле); каждую программу апробировать (отладить) на компьютере; получить результаты вычислений (“машинного счета”).

3. Подготовить решение заданий из ИДЗ, касающийся тематики ЛР № 5 (части 1) (разработать алгоритмы решения заданий, написать их программные реализации, апробировать их на компьютере, получить результаты и их проанализировать), подготовить отчет по ЛР № 5 (в части 1 – программирование алгоритмов, реализующих работу со строковым типом данных), касающийся выполнения ИДЗ (в плане подготовки и выполнения части 1 ЛР № 5) , и защитить данный отчет, сдав ЛР № 5,часть 1.

Часть 2. Программирование алгоритмов, реализующих работу в графическом режиме (построение графиков)

Цель работы:

1. Дальнейшее изучение методов и приемов программирования на алгоритмическом языке Бэйсик (версии Quick Basic, Free Basic).

2. Программирование алгоритмов, реализующих работу в графическом режиме (с использованием различных ранее изученных структур языка). Изучение и приобретение практических навыков и приемов составления программ с использованием графического режима (построение и воспроизведение графики).

3. Дальнейшее изучение конкретной интегрированной среды разработки (среды программирования) (версии Quick Basic, Free Basic), приемов отладки программ, закрепление навыков работы в отладочных режимах среды(д) разработки.

Для воспроизведения графики компьютер снабжен специальными аппаратными средствами. К ним относятся монитор и специальное

устройство - видеоадаптер (видеокарта), выполняющее роль переводчика между памятью и экраном. Видеоадаптер вместе с монитором образуют видеоподсистему. Видеоподсистемы работают в двух видеорежимах: текстовом или графическом. В текстовом режиме экран монитора разбивается на отдельные символьные позиции, в каждой из которых может выводиться только один символ.

В графическом режиме для каждой точки изображения, называемой пикселем, отводится от одного (монохромный режим) до 24-бит (цветной). В этом режиме имеется доступ к каждой точке изображения. Любое изображение можно представить в виде множества мельчайших точек, каждой из которых сопоставлены две координаты и номер цвета. Полученный числовой набор, называемый **растром**, более или менее точно опишет изображение. Графические режимы используются для формирования рисунков.

В программировании используется такая характеристика, как *разрешение.* Для графических режимов - это количество доступных точек на экране, для текстовых - количество символов в строке. *Разрешение экрана* является одним из важнейших параметров видеоподсистемы. Чем оно выше, тем больше информации можно отобразить на экране.

Количество различных цветов (ц*ветовое разрешение)*, доступных для раскрашивания изображений - другое важное свойство графического режима. Базовая палитра IBM - совместимых компьютеров включает 16 стандартных цветов. В программах цвета задаются своими номерами, приведенными ниже:

Номера экранных цветов

номер	цвет	номер	цвет
0	черный	8	серый
1	голубой	9	ярко-голубой
2	зеленый	10	ярко-зеленый
3	бирюзовый	11	ярко-бирюзовый
4	красный	12	ярко-красный
5	розовый	13	ярко-розовый
6	коричневый	14	желтый
7	белый	15	ярко-белый

Современные персональные компьютеры комплектуются дисплеями, способными отображать палитры от 256 до 16 млн. цветов. В графическом режиме каждый пиксель определяется цветом и своими координатами - положением относительно левого верхнего угла экрана, который, в свою очередь, имеет координаты 0,0. Программист может управлять цветом

любого пикселя, что позволяет формировать на экране любые изображения, в том числе рисунки, графики, чертежи, символы.

В текстовых режимах можно задавать координаты символа, определяя положение курсора относительно левого верхнего угла экрана (1,1), цвет символа (цвет переднего плана) и цвет фона (цвет заднего плана).

Функции и операторы графического режима

1. Оператор **CLS** - очищает экран дисплея.

2. Оператор **VIEW [[SCREEN](x1,y1)-(x2,y2)[,[n][,m]]]** - устанавливает окно графического вывода, ограниченное прямоугольной областью с координатами (x1,y1) левого верхнего угла и (x2,y2) правого нижнего угла, где n - числовое выражение: цвет закраски окна;

m - числовое выражение: цвет рамки окна вывода;

SCREEN - аргумент. Определяет, что координаты x, y любой выводимой точки имеют абсолютные значения. Графика выводится только внутри окна. Если SCREEN отсутствует, то внутри окна вывода действуют относительные к границам окна (x1,y1) координаты.

3. Оператор **WINDOW [[SCREEN](x1,y1)-(x2,y2)]** - устанавливает размеры текущего окна вывода, где (x1,y1)-(x2,y2) - числа обычной точности, определяющие координаты прямоугольного окна вывода.

Оператор WINDOW позволяет создать координатную систему для рисования линий, графиков и пр. без задания абсолютных координат на экране. Таким образом, координаты всех точек определяются как относительные к данному окну вывода.

Модификация WINDOW SCREEN преобразует направление координаты Y в декартово представление, т.е. значения Y идут от большего к меньшему сверху вниз.

4. Оператор **SCREEN n**, где **n** - выражение или константа целого типа, указывает номер режима экрана.

Оператор SCREEN используется для выбора режима экрана в соответствии с особенностями аппаратуры видеоадаптера и монитора. Этот оператор в программе должен предшествовать любым графическим операторам.

Параметры основных режимов оператора **SCREEN** адаптера **VGA**:

Режимы (n)	Разрешение	Число атрибутов	Число цветов	Примечание
0	текстовый режим (обычно 25 строк по 80 символов)			
1	320 x 200	4	16	
2	640 x 200	2	16	
7	320 x 200	16	16	
8	640 x 200	16	16	
9	640 x 350	16	64	
11	640 x 480	2	256	Необходимо применение оператора PALETTE
12	640 x 480	16	256	
13	320 x 200	256	256	

5.Оператор **COLOR [n],[m]** - устанавливает экранные цвета,
где n - цвет переднего плана (букв или графич. изображений);

m - цвет фона.

Например, COLOR 10,4 - оператор установит на красном фоне ярко-зеленый цвет букв или графических изображений.

6. Оператор **PALETTE [n,m]** - изменяет цвет в палитре,
где n - атрибут палитры, который нужно изменить (0 - 15);

m - цвет, присваиваемый атрибуту. Величина длинного, целого типа.

PALETTE USING A%[(i)] - изменяет все атрибуты одновременно, где A% - имя массива длинного, целого типа. Массив A% должен содержать номера цветов, присваиваемых атрибутам текущего режима экрана.

i - индекс начального элемента массива, используемый для установки палитры.

7. Операторы **PRESET [STEP](x,y)[,n]; PSET [STEP](x,y)[,n]** - рисуют точку на экране, где (x,y) - координаты точки на экране. Величины координат отображаемых на экране точек зависят от установленного графического режима (оператор SCREEN). Значения координат могут выходить за пределы экрана, но не должны превышать значений (от -32768 до 32767);

STEP - указывает, что координаты (x,y) отображаемой точки берутся как приращение к соответствующим координатам предыдущей точки. При отсутствии аргумента STEP координаты (x,y) непосредственно указывают физические координаты экрана;

n - цвет.

Различие операторов PSET и PRESET состоит в том, что если цвет опущен, в первом случае точка выводится цветом переднего плана, а во втором - цветом фона. На атрибут цвета влияют операторы COLOR и PALETTE.

8. Оператор **LINE [[STEP](x1,y1)]-[STEP](x2,y2)[,[n][,[B[F]][,&m]]]** рисует линии, контуры прямоугольников, закрашенные прямоугольники,

где (x1,y1)-(x2,y2) - устанавливают координаты начала и конца отрезка;

n - цвет линии, контура или закраски прямоугольника;

B- параметр устанавливает режим рисования прямоугольника;

BF- параметр устанавливает режим рисования прямоугольника, закрашенного цветом n;

STEP - указывает на относительные координаты (относительно последней точки).

&m - параметр "стиль" - 16-битовое целое число. Используется для рисования пунктирных линий.

9. Оператор **CIRCLE [STEP](x,y),r[,[n][,[fi1][,[fi2][,k]]]]** - рисует эллипс или окружность,

где (x,y) - координаты центра окружности или эллипса;

r - радиус круга или эллипса;

STEP - указывает, что (x,y) задаются как смещение относительно текущей позиции графического курсора;

n - цвет фигуры;

fi1,fi2 - начальный и конечный углы рисования фигуры. Их значения - в диапазоне от -2*PI до 2*PI радиан. При задании положительного угла рисуется лишь дуга, при задании отрицательного угла - та же дуга, но с отрезком, соединяющим точку дуги с центром.

k - коэффициент сжатия: отношение радиуса "y" к радиусу "x". По умолчанию его значение соответствует окружности, а его величина зависит от режима.

10. Оператор **PAINT [STEP](x,y)[,[n][,[m][,k$]]]** - закрашивает графические фигуры,

где STEP - устанавливает режим отсчета координат относительно текущего положения курсора;

(x,y) - координаты начала закраски. Если точка внутри фигуры, то закрашивается сама фигура, если точка вне фигуры, то закрашивается фон;

n - атрибут цвета;

m - атрибут цвета для рамки;

k$ - параметр строкового типа, указывает шаблон цвета фона. Этот параметр используется для закраски уже окрашенной области экрана.

Следует отметить, что данный оператор правильно работает лишь в том случае, когда:

- закрашиваемая область - замкнута;
- координаты (x,y) задают точку внутри окрашиваемой фигуры;
- окрашиваемая фигура ограничена рамкой цвета m.

11. Оператор **VIEW PRINT [n to m]** - устанавливает границы текстового окна вывода,

где n - числовое выражение: первая строка текстового вывода;

m - числовое выражение: последняя строка текстового вывода.

Если аргументы отсутствуют, окном является весь экран.

12. Оператор **LOCATE n, m, i, l, k** - используется в текстовом режиме, перемещает курсор на указанную позицию экрана,

где n - номер текстовой строки экрана; m - номер позиции в строке;

i - двоичное число делает курсор невидимым при значении "0" и видимым при значении "1";

l, k - начало и конец курсора соответственно, эти параметры управляют размером курсора по вертикали.

В качестве параметров могут быть переменные, арифметические выражения или числа. Все они должны быть целого типа. Любой из этих параметров может быть опущен.

Оператор PRINT, следующий за оператором LOCATE, выводит символы на экран в позицию **m** строки **n.** Аналогично оператор INPUT, следующий за оператором LOCATE, запрашивает данные с позиции m, n.

13. Оператор **POINT (x,y)** определяет цвет точки с координатами x,y.

14. Оператор **DRAW "выражение"** - рисует фигуру на экране в соответствии с содержимым параметра "выражение", заключенного в кавычки. Оператор позволяет достаточно просто рисовать сложные изогнутые траектории с помощью макроопределений, сведенных в таблицу. При этом команда перемещения рисует линию от точки последней ссылки к точке с координатами x,y.

Команды перемещения	
Un	Вверх (на n пикселей)
Dn	Вниз
Ln	Влево
Rn	Вправо
En	По диагонали вверх и вправо
Fn	По диагонали вниз и вправо
Gn	По диагонали вниз и влево
Hn	По диагонали вверх и влево
Mx,y	Перемещение в точку с координатами (x,y)
B	Перемещение без отображения
N	Перемещение с возвратом в исходную точку
Команды вращения	
An	Установка угла поворота, кратного PI/2 рад (n=0, 1,2,3)
Ta n	Поворот на угол n (-360 <= n <= 360 град.). Положительный угол соответствует повороту против часовой стрелки.
Sn	Установка масштабного коэффициента n - от 1 до 255. Число точек на экране в масштабной единице равно n/4. По умолчанию n=4.

Команды управления цветом	
Cp	Установка цвета с номером n
Pp, q	Выбор цвета закраски фигуры: p - номер цвета закраски, q - номер цвета границы

Ниже приводятся практические задания (для **части 2** ЛР № 5) с образцами примеров типовых заданий по программированию алгоритмов, реализующих работу в графическом режиме (с использованием различных ранее изученных структур языка). По этим типовым заданиям студенты в рамках самостоятельной работы студентов (обучающихся) – СРС или СРО) разрабатывают алгоритмы (составляют схемы) решения этих задач и их программные реализации (программные коды) в конкретной среде программирования, апробируют на компьютере и сохраняют в электронном виде (в отдельных файлах) на носителе.

Задание 1.

1. Построить столбчатую диаграмму (гистограмму) по заданным пяти исходным числам. Закрасить ее прямоугольники.

Задание 2.

1. Написать программу, перемещающую по нажатию соответствующих клавиш исходный закрашенный круг по вертикали и горизонтали на один пиксель при каждом нажатии.

Лабораторное задание (к части 2 ЛР № 5)

1. Продолжать изучение и освоение различных режимов окна редактирования соответствующего экранного редактора конкретной интегрированной среды.

2. Выполнить приведенные в лабораторной работе № 5 (в части 2) типовые задания (по работе с графикой, в графическом режиме): написать алгоритм решения каждого примера и его программную реализацию; сохранить каждую программу каждого типового задания в отдельном файле (программном файле); каждую программу апробировать (отладить) на компьютере; получить результаты вычислений (“машинного счета”).

3. Подготовить решение заданий из ИДЗ, касающийся тематики ЛР № 5 (части 2) (разработать алгоритмы (реализующие работу с графикой, в графическом режиме с использованием ранее изученных структур языка) решения заданий, написать их программные реализации, апробировать их на компьютере, получить результаты и их проанализировать), подготовить отчет по ЛР № 5 (в части 2 – программирование алгоритмов, реализующих работу в графическом режиме с использованием различных структур языка), касающийся выполнения ИДЗ (в плане подготовки и выполнения части ЛР № 5) , и защитить данный отчет, сдав ЛР № 5,часть .

В целом по ЛР № 5 должен быть подготовлен материал для отчета по ЛР № 5 (охватывающий как часть 1, так и часть 2) для защиты соответствующего варианта ИДЗ.

Лабораторная работа № 6
КОМПЛЕКСНОЕ ПРОГРАММИРОВАНИЕ В ИНТЕГРИРОВАННОЙ СРЕДЕ РАЗРАБОТКИ (СРЕДЕ ПРОГРАММИРОВАНИЯ) НА БАЗЕ АЛГОРИТМИЧЕСКОГО ЯЗЫКА VISUAL BASIC

Цель работы:

Целью работы является освоение приемов создания приложений для операционной системы Microsoft Windows на языке программирования Visual Basic 15.0 в среде разработки Microsoft Visual Studio 2017.

Общие сведения об языке программирования Visual Basic и среде разработки приложений Microsoft Visual Studio

Язык программирования Visual Basic и предназначенные для работы с ним средства разработки приложений были созданы корпорацией Microsoft в 1991 году. Данный язык является объектно-ориентированным, и предназначен для создания оконных приложений для операционных систем семейства Windows. Актуальной на момент публикации настоящего учебного пособия версией языка является Visual Basic 15.0, выпущенная в 2017 году.

Интегрированная среда разработки приложений Microsoft Visual Studio была представлена корпорацией Microsoft в 1997 году и объединила в себе средства разработки приложений на языках программирования Visual Basic, C++, Java, ранее выпускавшиеся в виде отдельных продуктов. В процессе своего развития среда разработки включала в себя новые средства и языки программирования. В настоящее время актуальной является версия Visual Studio 2017. Данная среда разработки включает в себя в том числе средства разработки на языке программирования Visual Basic 15.0. Вид главного окна Visual Studio 2017 показан на рис 11.

Рассмотрим основные этапы создания приложения, на примере задачи вычисления средних арифметических значений для строк матрицы размера 5×3.

Создание проекта в Visual Studio

Для того, чтобы создать в Visual Studio проект на языке Visual Basic, следует выбрать пункт главного меню «Файл» и в открывшихся подменю выбрать пункты «Создать» → «Проект…». В открывшемся окне «Создание проекта» необходимо выбрать язык программирования Visual Basic, вид создаваемого приложения – «Проект приложения с пользовательским интерфейсом Windows» (см. рис. 12). В этом же окне следует указать название создаваемого приложения (поле «Имя», в приведенном примере – Matrix_app) и место сохранения файлов проекта (поле «Расположение», в приведенном примере –D:\Test).

После нажатия кнопки OK, произойдет создание проекта, содержащего единственную пустую форму Form1, открытую в визуальном конструкторе. В правой части окна Visual Studio появится окно «Свойства», отображающее свойства объекта, выделенного с помощью мыши.

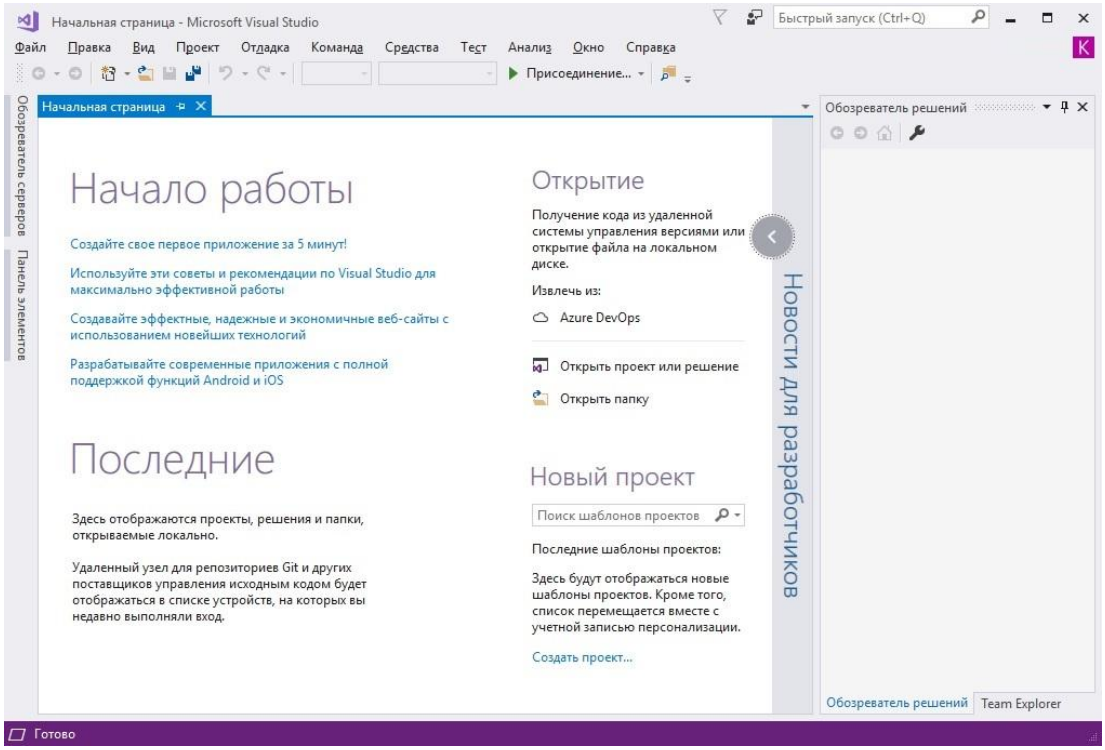


Рис. 11. Главное окно Visual Studio 2017

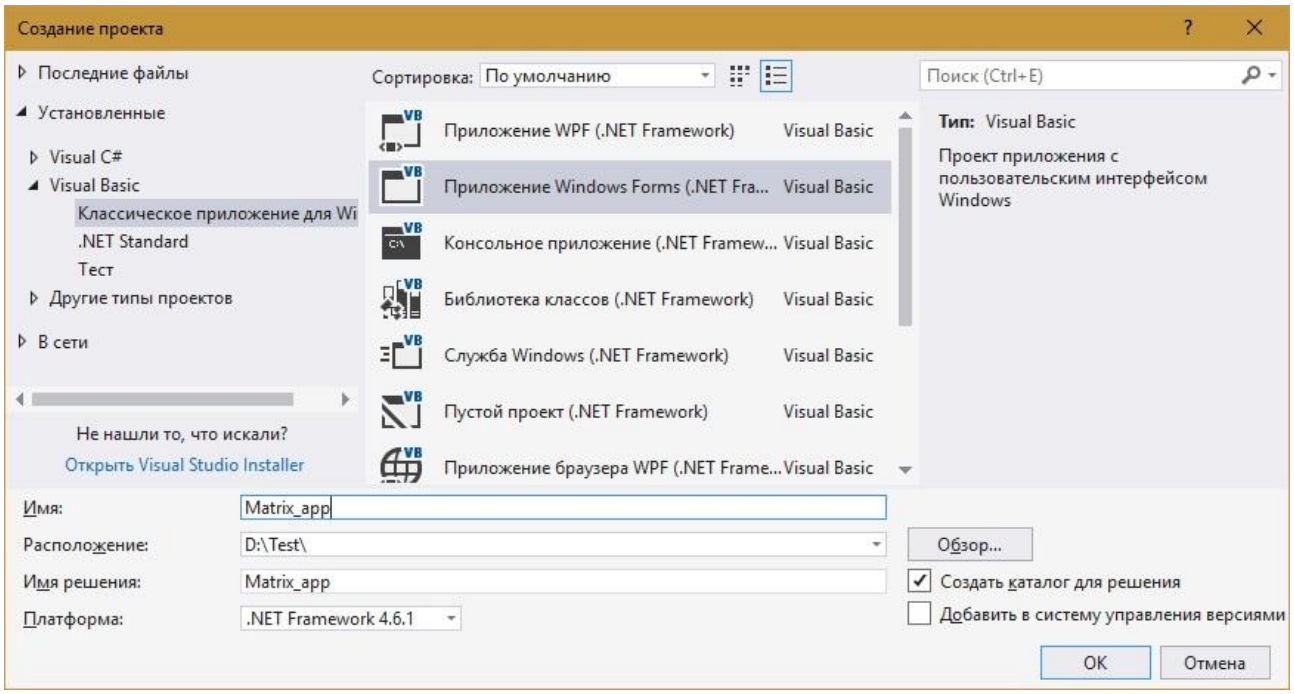


Рис. 12. Окно «Создание проекта»

Для того, чтобы изменить заголовок формы Form1, необходимо в окне «Свойства» задать ее свойство text (см. рис. 13).

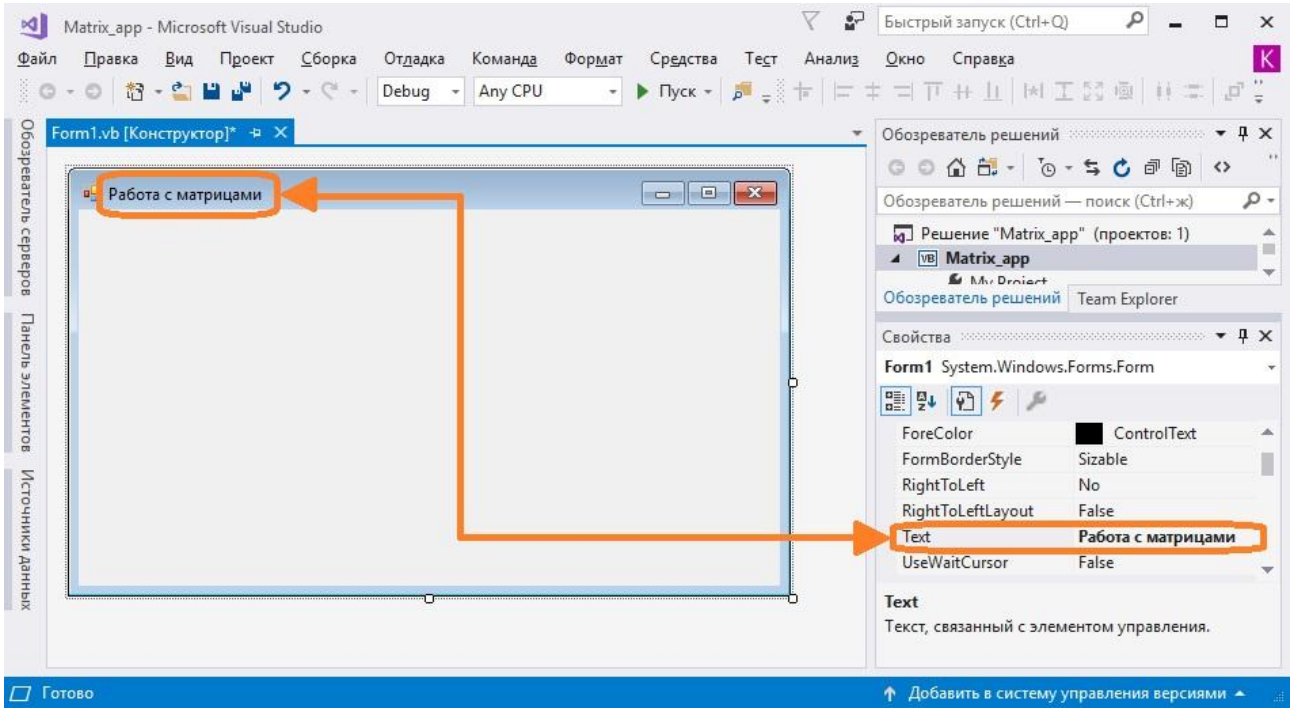


Рис. 13. Использование окна «Свойства»

Добавление объектов на форму

Для ввода значений матрицы следует добавить на форму Form1 12 объектов типа TextBox, каждый из которых представляет собой текстовое поле для ввода одного из элементов матрицы. Для этого необходимо открыть Панель элементов (на рис.13 слева показана вкладка, открывающая одноименную панель), в разделе «Стандартные элементы управления» выбрать объект типа TextBox и перенести его на форму. В проект будет включен экземпляр объекта типа TextBox с именем TextBox1. Данный объект может быть с целью сокращения времени на добавление последующих текстовых полей скопирован с помощью контекстного меню, вызываемого щелчком правой кнопкой мыши на нем, либо с помощью комбинации клавиш Ctrl+C. Для вставки копий объекта требуется использовать контекстное меню, вызываемое щелчком правой кнопкой мыши по пустой части формы, либо с помощью комбинации клавиш Ctrl+V. Аналогично можно выделять и копировать различные объекты, расположенные на форме. После вставки всех 12 текстовых полей, соответствующие им объекты получат имена TextBox1 – TextBox12.

Для визуального отделения элементов матрицы от прочих объектов на форме, можно использовать рамку типа GroupBox (раздел «Контейнеры» Панели элементов). Перенеся рамку на форму, и создав таким образом объект GroupBox1, следует в окне «Свойства» задать значение его свойства Text: «Матрица:» и перенести в данную рамку текстовые поля TextBox1 – TextBox12, как это показано на рис. 14.

Аналогично создается рамка GroupBox2 с четырьмя текстовыми полями TextBox13 – TextBox16 для вывода сумм элементов строк. Подписи к этим суммам выполнены с помощью объектов типа Label (метка), свойство которых Text определяет содержимое метки.

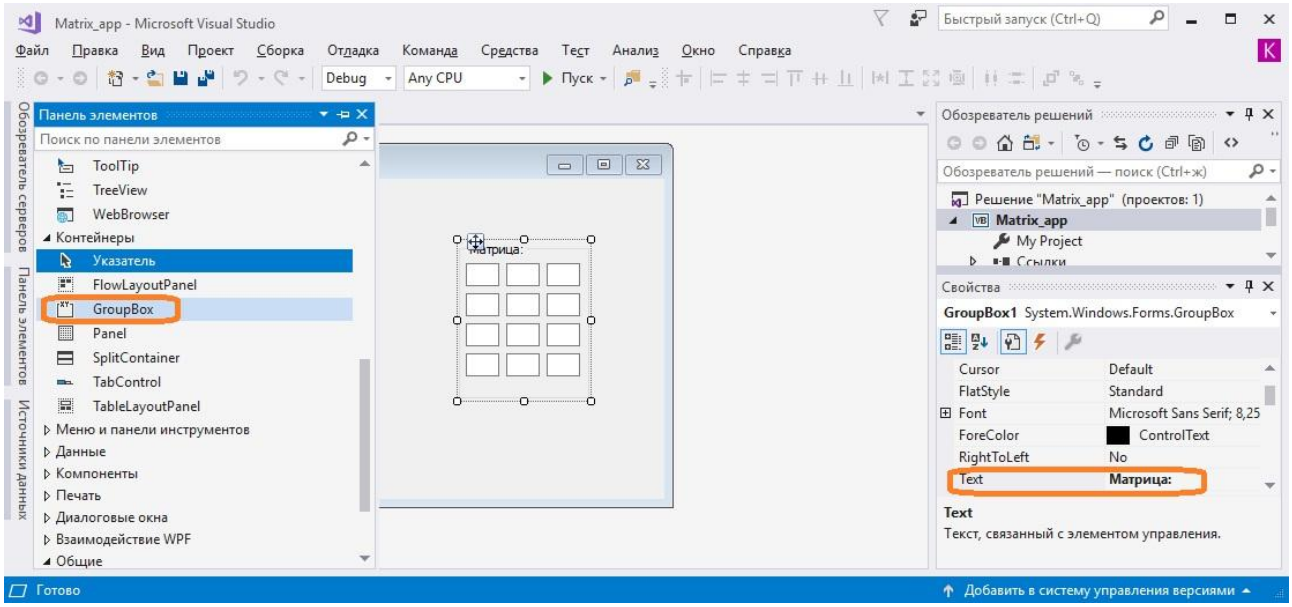


Рис. 14. Создание рамки и текстовых полей для ввода матрицы

Для запуска процедуры вычисления средних арифметических значений строк матрицы требуется разместить кнопку (объект типа Button, см. раздел «Стандартные элементы управления» Панели элементов). Свойство Text данного объекта Button1 определяет надпись на кнопке. Окончательный вид формы с размещенными на ней объектами показан на рис. 15.

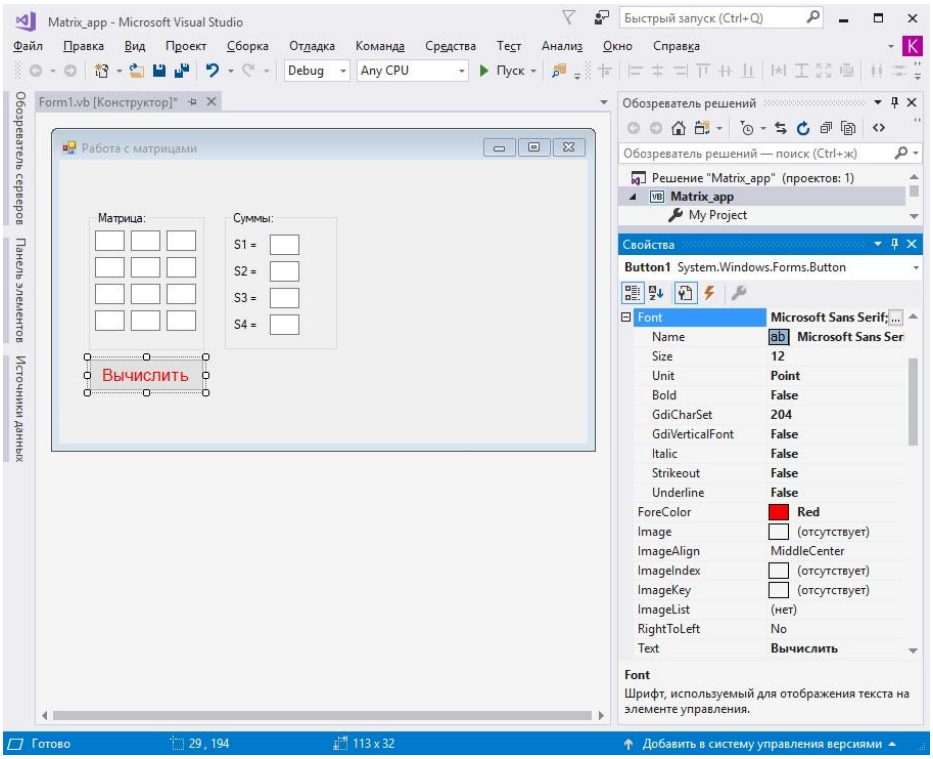


Рис. 15. Сформированное окно приложения

Реализация процедуры расчета средних арифметических и вывода результатов

Для того, чтобы создать процедуру, вызываемую нажатием кнопки Button1, необходимо в окне Конструктора сделать двойной щелчок левой кнопкой мыши по названной кнопке. При этом откроется вкладка редактора

кода, позволяющая создавать и редактировать методы, привязанные к каждому действию для каждого объекта на форме. В нашем случае создается метод Button1.Click, описывающий процедуру, вызываемую щелчком мыши (событие Click) по кнопке Button1. Программный код, обеспечивающий решение поставленной задачи, в нашем случае имеет вид, показанный на рис. 16.

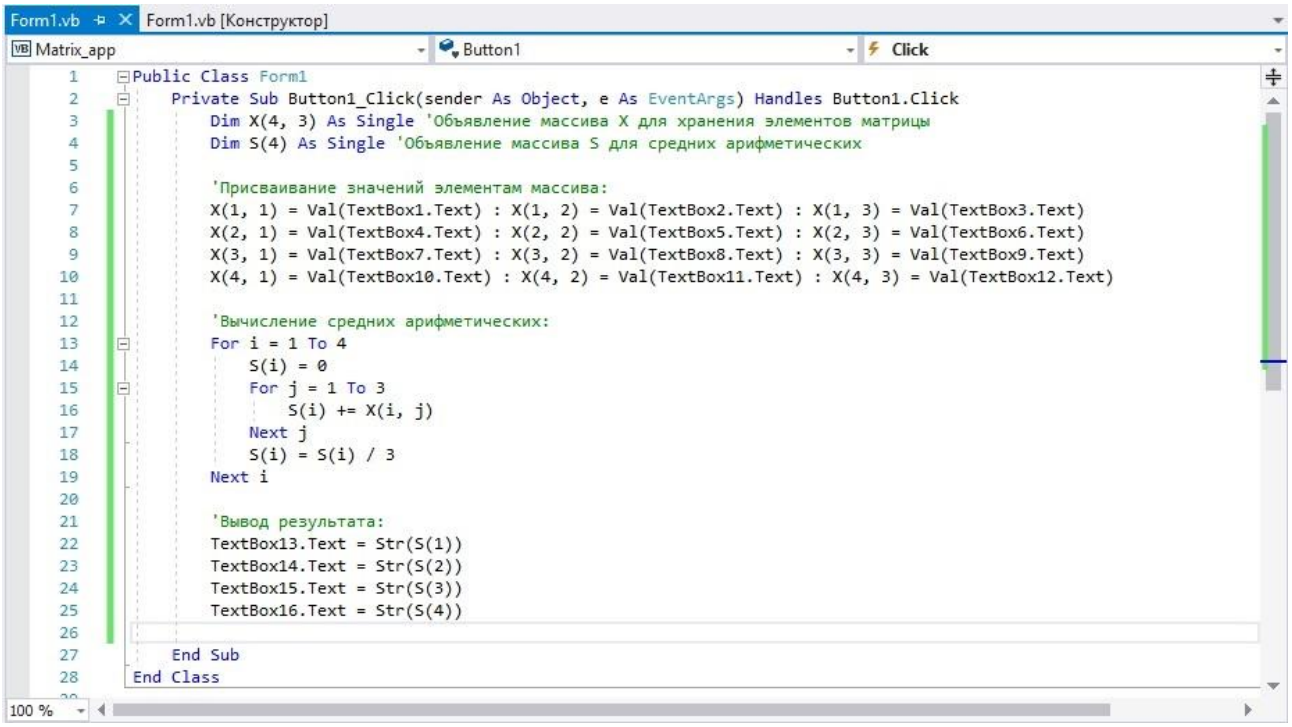


Рис. 16. Описание метода Button1.Click

Для запуска приложения в режиме отладки следует выбрать режим сборки приложения Debug в выпадающем списке «Конфигурация решения» (см. рис. 17) и нажать кнопку «Пуск». Результат работы программы показан на рис. 18. Если в качестве режима сборки выбрать Release, то будет осуществлена компиляция исполняемого файла Matrix_app.exe, расположенного в каталоге D:\Test\Matrix_app\Matrix_app\bin\Release.

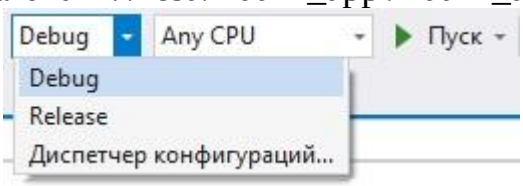


Рис. 17. Выпадающий список «Конфигурация решения» и кнопка «Пуск»

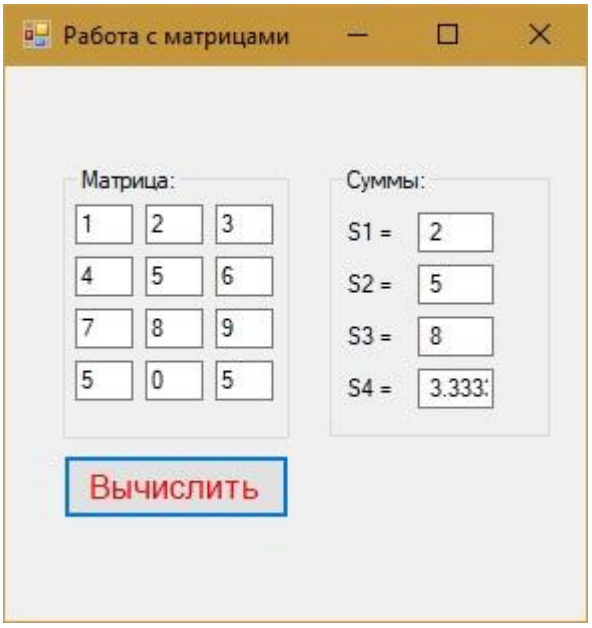


Рис. 18. Результат работы программы

Создание консольного приложения в среде Visual Studio 2017

Хотя Visual Basic ориентирован на создание оконных приложений для операционных систем Windows, он также поддерживает возможности создания консольных приложений. В качестве примера создадим приложение, обеспечивающее вычисление средних арифметических строк матрицы и работающее в текстовом режиме.

Для этого создадим проект консольного приложения: в окне «Создание проекта» выберем вид создаваемого приложения «Проект приложения для командной строки» - «Консольное приложение (.NET framework)».

В открывшемся окне редактора кода введем текст процедур, обеспечивающих ввод матрицы (Matrix_input), вывод матрицы (Matrix_print), вычисление и вывод средних арифметических значений строк (Calculate). Вызовы данных процедур осуществляются в процедуре Main, обеспечивающей работу приложения. Текст вышеназванных процедур приведен на рис. 19. Результат выполнения программы показан на рис. 20.

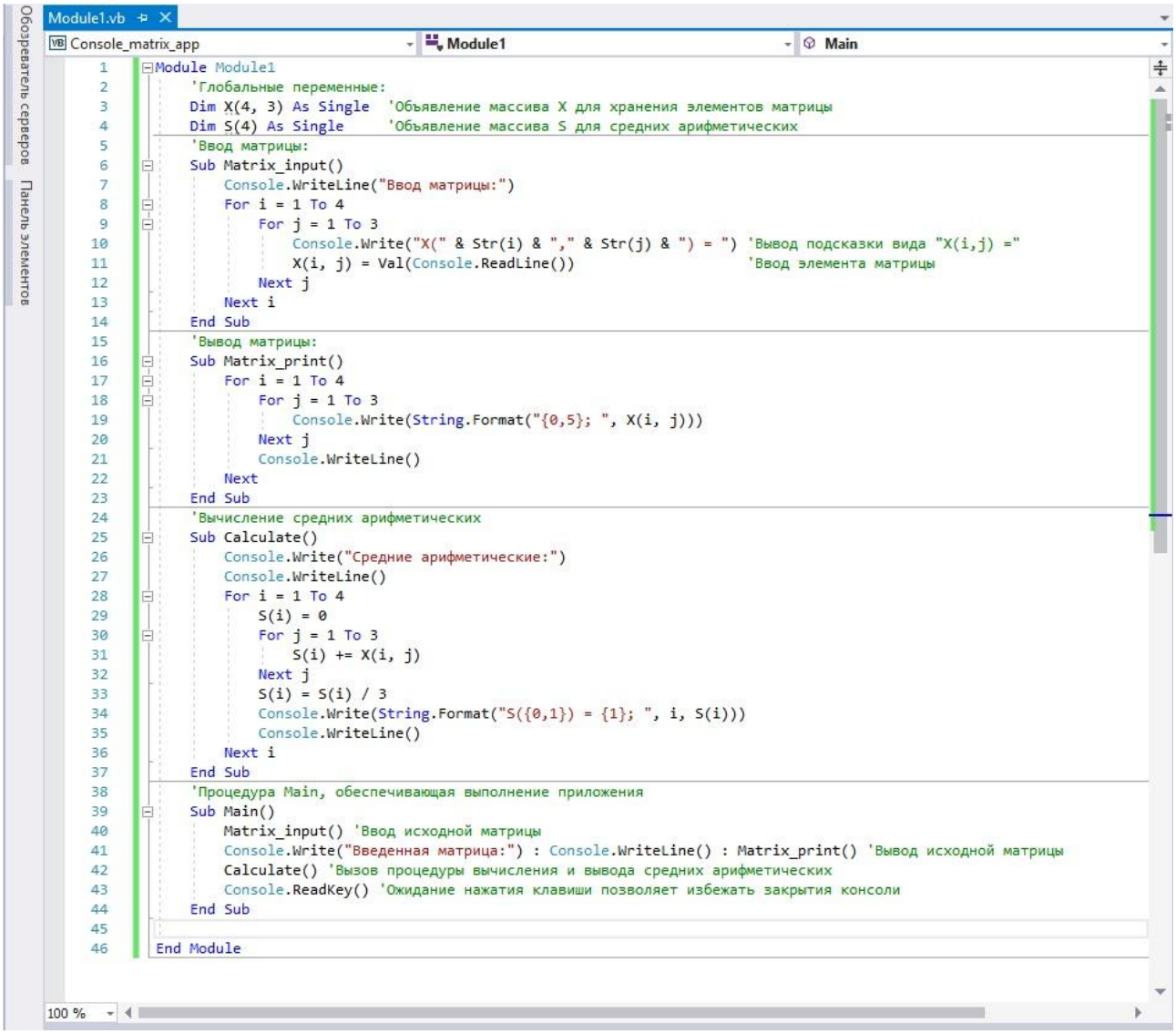
Отметим, что в качестве разделителя целой и дробной частей чисел в Visual Basic используется точка.

Примечание. Использование метода String.Format позволяет осуществить форматированный текстовый вывод строк. Общий вид оператора:

String.Format("…{0}…{1}…{2}…{N}…", x_0, x_1, x_2, … ,x_N).

Вместо выражения {i} в выводимую строку будет вставлено значение i-го аргумента x_i (нумерация аргументов начинается с 0).

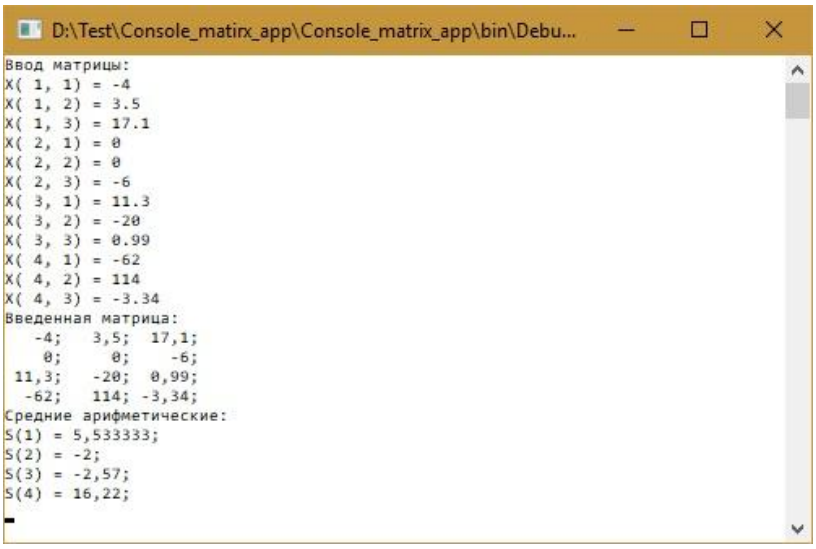
Выражение вида {i,k} заменяется значением i-го аргумента списка вывода, на которое отводится k знакомест консоли.



```
Module Module1
    'Глобальные переменные:
    Dim X(4, 3) As Single  'Объявление массива X для хранения элементов матрицы
    Dim S(4) As Single     'Объявление массива S для средних арифметических
    'Ввод матрицы:
    Sub Matrix_input()
        Console.WriteLine("Ввод матрицы:")
        For i = 1 To 4
            For j = 1 To 3
                Console.Write("X(" & Str(i) & "," & Str(j) & ") = ") 'Вывод подсказки вида "X(i,j) ="
                X(i, j) = Val(Console.ReadLine())                    'Ввод элемента матрицы
            Next j
        Next i
    End Sub
    'Вывод матрицы:
    Sub Matrix_print()
        For i = 1 To 4
            For j = 1 To 3
                Console.Write(String.Format("{0,5}; ", X(i, j)))
            Next j
            Console.WriteLine()
        Next
    End Sub
    'Вычисление средних арифметических
    Sub Calculate()
        Console.Write("Средние арифметические:")
        Console.WriteLine()
        For i = 1 To 4
            S(i) = 0
            For j = 1 To 3
                S(i) += X(i, j)
            Next j
            S(i) = S(i) / 3
            Console.Write(String.Format("S({0,1}) = {1}; ", i, S(i)))
            Console.WriteLine()
        Next i
    End Sub
    'Процедура Main, обеспечивающая выполнение приложения
    Sub Main()
        Matrix_input() 'Ввод исходной матрицы
        Console.Write("Введенная матрица:") : Console.WriteLine() : Matrix_print() 'Вывод исходной матрицы
        Calculate() 'Вызов процедуры вычисления и вывода средних арифметических
        Console.ReadKey() 'Ожидание нажатия клавиши позволяет избежать закрытия консоли
    End Sub

End Module
```

Рис. 19. Исходный код консольного приложения



```
D:\Test\Console_matirx_app\Console_matrix_app\bin\Debu...
Ввод матрицы:
X( 1, 1) = -4
X( 1, 2) = 3.5
X( 1, 3) = 17.1
X( 2, 1) = 0
X( 2, 2) = 0
X( 2, 3) = -6
X( 3, 1) = 11.3
X( 3, 2) = -20
X( 3, 3) = 0.99
X( 4, 1) = -62
X( 4, 2) = 114
X( 4, 3) = -3.34
Введенная матрица:
   -4;   3,5;  17,1;
    0;     0;    -6;
 11,3;   -20;  0,99;
  -62;   114; -3,34;
Средние арифметические:
S(1) = 5,533333;
S(2) = -2;
S(3) = -2,57;
S(4) = 16,22;
```

Рис. 20. Результат работы программы

Лабораторное задание

1. Выбрать диск, на котором студентам разрешено создавать объекты.

2. В соответствии с указаниями преподавателя создать на указанном диске папку, название которой соответствует ФИО студента.

3. В соответствии с указаниями преподавателя разработать оконное приложение в среде Microsoft Visual Studio.

4 В соответствии с указаниями преподавателя разработать консольное приложение в среде Microsoft Visual Studio.

5. Разработать комплексное задание в рамках варианта ИДЗ.

Литература

1. Информатика. Общий курс. Учебник / А.Н. Гуда и др.; под ред. В.И. Колесникова. – М.: Издательско-торговая корпорация «Дашков и К°»; Ростов н/Д: Наука-Спектр, 2011. – 400 с.

2. Курилёнок А.С., Пичугин А.А. Информатика и информационные технологии. Информатика. Учебно-методическое пособие по выполнению практических занятий и комплексных домашних заданий. – М.: ИД Академии Жуковского, 2018. – 32 с.

3 Андреева Т.И., Кишенский С.Ж. Информатика. Часть 2: Пособие по выполнению лабораторных работ. – М.: МГТУ ГА, 2008. – 64 с.

ПРИЛОЖЕНИЕ 1.

В рамках конкретных лабораторных работ студентами решаются конкретные теоретико-практические задачи с использованием компьютерных технологий (в конкретных задачах конкретных лабораторных работ осуществляются различные виды обработки: численной и строковой информации; структур и массивов данных; работа с графикой, в графическом режиме). Студенты разрабатывают, описывают и исследуют алгоритмы решения указанных конкретных задач с последующей апробацией их программных реализаций (программных кодов для событийных процедур) в конкретной среде программирования на конкретном языке программирования (Quick Basic, Free Basic, Visual Basic).

В каждом варианте соответствующего индивидуального домашнего задания (ИДЗ) для соответствующей конкретной лабораторной работы необходимо выполнить следующее:

1. Разработать алгоритм(ы) решения предлагаемой(ых) задачи(ч).
2. Представить и нарисовать схему(ы) разработанного(ых) алгоритма(ов) решения этой(их) задачи(ч).
3. Разработать экранную форму с набором элементов управления.
4. Осуществить программную(ые) реализацию(и) (программные коды) разработанного(ых) алгоритма(ов) в конкретной среде программирования на конкретном языке программирования (типа Quick Basic, Free Basic, Visual Basic) (разработать и написать программу).
5. Получить результаты решения предложенной(ых) задачи(ч) (апробировать (осуществить выполнение, “счет”) разработанную(ые) программу(ы) на компьютере.

По результатам выполнения ИДЗ для каждой соответствующей лабораторной работы студент предоставляет материалы отчета, имеющего нижеследующую структуру и содержание.

Структура отчета по лабораторной
работе № 1 (2, 3, 4, 5, 6)

1. Титульный лист (1 лист).

(В ПРИЛОЖЕНИИ 2 приведён образец).

2. Содержание (оглавление) отчета (1 лист).
3. Задание (ИДЗ) для конкретной лабораторной работы (№ _) (1 лист).
4. Схема(ы) алгоритма(ов) решения предложенной(ых) задачи(ч) из ИДЗ (1 или несколько листов).
5. Программная(ые) реализация(и) (программные коды) разработанного(ых) алгоритма(ов) решения предложенной(ых) задачи(ч) из ИДЗ (текст(ы) программы(м) на конкретном языке программирования – типа Quick Basic, Free Basic, Visual Basic) (1 или несколько листов).
6. Результаты выполнения (результаты "счета") предложенной(ых) задачи(ч) на компьютере ("машинные" распечатки) (1 или несколько листов).
7. Материала теории по используемому конкретному языку программирования (1 или несколько листов).

Содержание (оглавление) отчета по
лабораторной работе № 1 (2, 3, 4, 5, 6)

1. Задание (ИДЗ) для лабораторной работы № _.....
2. Схема(ы) алгоритма(ов) решения задачи(ч) ИДЗ……
3. Программная(ые) реализация(и) алгоритма(ов) решения задачи(ч) ИДЗ….
4. Результаты решения (апробации алгоритма(ов) и программной(ых) реализации(й) задач ИДЗ)……..
5. Материалы теории по используемому конкретному языку программирования (типа Quick Basic, Free Basic, Visual Basic)…….

Образец типового варианта ИДЗ

Пример конкретного варианта ИДЗ для конкретной лабораторной работы

Лабораторная работа № 1

ВАРИАНТ 11

Лабораторная работа №1. Простые прямые вычисления.

Задача 1.

Вычислить выражения (значения буквенных переменных задавать с клавиатуры):

1) $\sin^2(|x+y|)+\cos^3(xy)$

2) $\lg(1+tg^2\frac{\sin xy}{e^x})$

Задача 2.

Дана длина ребра куба *a*. Найти объем куба и площадь его боковой поверхности.

Задача 3.

Определить произведение двух первых цифр заданного четырехзначного числа *f*.

Лабораторная работа № 2

ВАРИАНТ 11

Лабораторная работа №2. Ветвления.

Задача 1.

Вычислить *f(x, y).* Значения *x, y* задаются с клавиатуры:

$$f(x,y)=\begin{cases}4^x-2^{y-1}-24, & x<=0\\ \max(x,12), & x>5\\ 5y+4, & 0<x<=5\end{cases}$$

Задача 2.

Определить, есть ли среди младших трех цифр целой части заданного числа *s* цифра 5.

Задача 3.

Даны координаты двух полей шахматной доски (в виде двух пар чисел от 1 до 8). Определить, являются ли эти поля полями одного цвета.

Лабораторная работа № 3

ВАРИАНТ 11

Лабораторная работа №3. Простые циклы.

Задача 1.

Вычислить выражения, используя для организации цикла операторы FOR … NEXT (значения переменной *n* задавать с клавиатуры):

$$1+\sum_{i=1}^{n}\frac{1}{i^2+2i}$$

Задача 2.

Определить сумму ряда с заданной точностью t $(t>0)$ и число слагаемых, необходимых для достижения этой точности. Точность считается достигнутой, если очередное слагаемое по модулю меньше t (это и последующее слагаемое не учитываются). Использовать для организации цикла операторы DO … LOOP или WHILE … WEND (значения буквенных переменных задавать с клавиатуры):

$$\sum_{i=1}^{\infty}\frac{(-1)^{i+1}}{i^2}$$

Задача 3.

Дано натуральное число *n*. Определить количество цифр в этом числе.

Задача 4.

Вводится последовательность ненулевых чисел, завершаемая нулем. Определить номер максимального числа в данной последовательности.

Лабораторная работа № 4

ВАРИАНТ 11

Лабораторная работа №4. Массивы.

Задача 1.

Определить сумму элементов массива *X(n)* с нечетными номерами. Вывести массив и полученную сумму. Упорядочить массив *X(n)* по неубыванию. Вывести полученный массив. Исходный массив элементов

получить с помощью датчика случайных чисел. Значения буквенных переменных задавать с клавиатуры.

Задача 2.

Дана матрица целых чисел *X(n, m)*. Получить новую матрицу, элементы строк которой больше элементов исходной матрицы на величину максимального элемента соответствующей строки исходной матрицы. Вывести исходную и полученную матрицы. Значения буквенных переменных задавать с клавиатуры.

Лабораторная работа № 5

ВАРИАНТ 11

Лабораторная работа №5 Символьные функции. (часть1)

Задача 1.

В исходной строке *a$* после каждого вхождения заданного сочетания символов *x$* ввести сочетание символов *y$*. Вывести исходную и полученную строки.

Задача 2.

В исходной строке *a$* произвести циклическую перестановку слов влево на *n* слов. Вывести исходную и полученную строки.

Лабораторная работа №5 (продолжение). Графика. (часть 2)

Задача 1.

Построить совокупность *n* кругов радиуса *r*, закрашенных в случайные цвета, центры которых расположены на диагонали используемого экрана.

Задача 2.

Сформировать движущийся круг радиуса *r*, центр которой перемещается попеременно в обоих направлениях вдоль вертикального отрезка с заданными границами.

Лабораторная работа № 6

ВАРИАНТ 11

Лабораторная работа №6.

Комплексное индивидуальное задание на алгоритмизацию и программирование прикладных практических задач в рамках интегрированной среды разработки (среды программирования) MS Visual Studio на базе алгоритмического языка Visual Basic.

ПРИЛОЖЕНИЕ 2

Образец

Титульный лист

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ

_____

Кафедра прикладной математики

Домашнее задание защищено:
_____(дата)
_____(оценка)
Преподаватель:_____
(Фамилия, инициалы)
Дата:_____
Подпись:_____

Отчет по лабораторной работе № 1
Работа в конкретной среде программирования (интегрированной среде) на конкретном алгоритмическом языке.
Разработка алгоритма линейной структуры для решения конкретных прикладных задач (линейных вычислительных процессов) и его программной реализации в конкретной среде программирования на конкретном языке программирования
Вариант № ___

Работу выполнил:
Студент (ка) группы_____
_____
(Фамилия, инициалы)
Дата: _____
Подпись: _____

Москва – 201_ г.