**А. Н. Юркевич**

**Г. Б. Рейнгольд**

**М. Г. Рейнгольд**

**ОСНОВЫ**

**алгоритмизации**

**и**

**программирования**

**(издание 3-е)**

**Методическое пособие для обучающихся**

**образовательных учреждений**

**по программированию**

**Иркутск**

**2021**

**Юркевич А. Н.**

**Рейнгольд Г. Б.**

**Рейнгольд М. Г.**

**Основы алгоритмизации и программирования. – 3-е изд.** – Иркутск:

© Юркевич А. Н., 2021

© Рейнгольд Г.Б., 2021

© Рейнгольд М .Г., 2021

***Предисловие***

«Многие вещи нам непонятны не потому, что наши понятия слабы; но потому, что сии вещи не входят в круг наших понятий.»

*(Козьма Прутков)*

***Дорогие читатели, ученики и коллеги-учителя!***

Данное пособие посвящено наиболее трудному и важному разделу школьного курса информатики - ***алгоритмизации*** ***и*** ***программированию***. Этот раздел требует для усвоения больших усилий, а порой и большие усилия оказываются тщетными. Но не надо заранее настраиваться на поражение, тогда его точно не избежать. Как гласит русская народная пословица, ***«Не так страшен чёрт, как его малюют»****.*

По ***программированию***, в том числе и на Бейсике, и на Паскале, написано и издано много книг. Каковы же особенности нашей? Сразу скажем, что чего-либо принципиально нового вы в ней не найдёте. Содержание предмета за последние десятилетия устоялось, сложились общие подходы к преподаванию.

В нашем пособии освещены лишь ***основные***, самые необходимые сведения и понятия ***алгоритмизации и программирования***, ***команды***, ***функции***, ***алгоритмические конструкции*** и ***структуры***. Они необходимы для того, чтобы усвоить на достаточно хорошем уровне общеобразовательную программу по ОИиВТ[[1]](#footnote-1). Именно поэтому мы старались ограничиться лишь необходимым минимумом. Отсюда и сравнительно небольшой объём нашей книжки.

Однако, этих сведений вполне достаточно для того, чтобы решить и гораздо более сложные задачи, далеко выходящие за рамки общеобразовательного курса, в частности олимпиадные задачи. Книга эта может оказаться полезной и ребятам, начинающим заниматься в кружках юных программистов, или осваивающим программирование самостоятельно.

Кроме того, мы стараемся по возможности придерживаться терминологии, которую предлагал пионер отечественной школьной информатики академик А.П. Ершов в 1985 году. Например, вместо традиционного ***«оператор»***, употребляем слово ***«команда»***. Такая терминология, как нам кажется, больше подходит для обучающихся общеобразовательных школ и организаций дополнительного образования.

Большое внимание уделяется и структурности ***программы***, её оформлению, что должно в дальнейшем помочь при самостоятельном составлении ***программ***. Понятие ***алгоритмической структуры*** – одно из главных в этом пособии.

Зачем нужен раздел ***«Алгоритмизация*** ***и*** ***программирование»*** в школьном курсе информатики? Одной из причин, мягко говоря, прохладного отношения многих обучающихся к этому разделу, кроме его трудности, является ошибочное, на наш взгляд, но глубоко укоренившееся мнение, что он не нужен в жизни. Что можно сказать по этому поводу?

Во-первых, без усвоения этого раздела хоть на каком-то уровне, невозможно понять принцип работы компьютера, а можно ли считать человека, не понявшего, как работает компьютер, понявшим информатику?

Во-вторых, на очень многих вузовских специальностях, не только программистских, ***программирование*** изучается, и те, кто манкировал им в школе, всё равно вынуждены его осваивать, будучи студентами.

В-третьих, умение хоть немного программировать, может оказать очень хорошую службу при работе с офисными приложениями, так широко используемыми в настоящее время. Язык VBA, предназначенный для этого, в своей алгоритмической основе ничем не отличается от ***QB***, который рассматривается здесь.

В-четвёртых, тот, кто понял программирование, перешёл на более высокий уровень умственного развития, с этим и спорить нечего… В конце концов, язык и литературу в школе изучают все, сочинения и изложения учатся писать все, а не только те, кто хочет стать писателем и журналистом.

Так вот, если у вас получится, здорово, если нет, что поделаешь? До вас ***программирование*** не поняли тысячи людей, и после вас не поймут тысячи. Вы просто будете среди них. Но давайте попробуем понять этот раздел, разобраться в нём, вдруг у вас получится!

\* \* \*

В качестве одного из изучаемых в наших центрах языка программирования избран ***QuickBASIC (QB)***, усовершенствованный вариант Бейсика, а также ***PASCAL ABC***, усовершенствованный вариант Паскаля, популярных языков начинающих программистов. Однако только изложением основ ***QB*** и ***PASCAL*** не исчерпывается содержание данного пособия. В нём изложены основы ***алгоритмизации***. Необходимо представлять себе разницу между понятиями ***программирование*** и ***алгоритмизация***. В настоящее время, благодаря существованию хороших ***языков программирования***, имеется возможность эти два процесса совмещать, что практически все и делают.

Подробно обо всём этом рассказано в пособии ***«Теоретические вопросы информатики»***. Мы надеемся, что вы уже ознакомились с ним. Во всяком случае, при изложении материала здесь мы исходим из того, что вам знакомы основные понятия информатики, принципы устройства и работы компьютера, файловая структура магнитных дисков, единицы измерения информации и многое другое.

Кроме того, для практической работы с ***Бейсиком и Паскалем*** на компьютере надо владеть основами работы с текстовым редактором, что описано в ***«Наставлении по работе с текстовым редактором»***.

Итак, в путь!

1. ***Алгоритмы***

Каждый человек в своей жизни решает большое количество задач разной сложности. Большинство задач можно разбить на простые этапы или шаги. Для многих таких задач разработаны и предлагаются пошаговые инструкции, при последовательном исполнении которых можно прийти к желаемому результату. Последовательность этих указаний называется **алгоритмом**.

**Алгоритм** – это последовательность действий, указаний исполнителю, после выполнения которых, исполнителем должен быть получен предполагаемый результат.

Само слово «алгоритм» происходит от «algorithmi» - латинской формы написания имени выдающегося математика IX века Мухамеда бен Муссы аль-Хорезми(783-850), который сформулировал правила выполнения арифметически операций. Все алгоритмы обладают определёнными свойствами.

**Свойства алгоритма.**

1. **Результативность** – после выполнения **всех** предписаний алгоритма исполнителем должен быть получен результат.
2. **Массовость** – алгоритм, написанный для решения какого либо класса задач должен выполнятся для любых исходных данных этого класса.
3. **Дискретность** – алгоритм состоит из последовательности шагов. Только выполнив один шаг можно перейти к следующему.
4. **Определённость** – алгоритм должен быть понятен исполнителю и не вызывать у него двоякого толкования.
5. **Конечность** – количество шагов алгоритма должно быть конечным.
6. **Эффективность** – количество шагов и сами шаги алгоритма должны быть такими, чтобы решение могло быть найдено за конечное и, более того, приемлемое время.

Схему работы алгоритма можно представить следующим образом:

Исходные данные Алгоритм Результат

Мы говорим об исполнителе алгоритма. Давайте определим кто такой исполнитель. **Исполнитель** – это некоторый объект (человек, животное, техническое устройство), который исполняет данный алгоритм.

Алгоритмы можно описать несколькими способами:

1. **Формульный;**

Используется для решения задач с помощью формул (в математике, физике, химии и др.).

1. **Словесный;**

Изложение алгоритма ведётся на обычном языке с разделением на последовательные шаги.

1. **Графический;**

Алгоритм изображается в виде последовательности связанных между собой функциональных блоков (**язык блок-схем**), каждый из которых соответствует выполнению одного или нескольких действий. Такое графическое представление алгоритма называется *блок-схемой*. Определённому типу действий соответствует определённая геометрическая фигура - *блочный символ*. Блоки соединяются между собой линиями переходов (линии со стрелками), которые определяют очерёдность выполнения действий.

1. **С помощью формальных алгоритмических языков (языков программирования).**

При записи алгоритмов используют строго определённый набор символов и составленных из них специальных зарезервированных слов. Имеют строгие правила построения языковых конструкций.

1. ***Язык Блок-Схем***

***Язык блок-схем*** состоит из следующих блоков:

|  |  |  |
| --- | --- | --- |
| **Блок** | **Название** | **Комментарии** |
|  | Блок  Начала | Данный блок имеет один выход. |
|  | Блок описания данных | Данный блок имеет один вход и один выход. |
|  | Блок  Ввода данных и Вывода результата | Данный блок имеет один вход и один выход. |
|  | Блок  Действия | Данный блок имеет один вход и один выход. |
| Да Нет | Блок  Условия | Данный блок имеет один вход и два выхода. На каждом из выходов пишется слово «Истина(да или знак +)» или «Ложь (нет или знак -)». |
|  | Блок множественного выбора | Данный блок имеет один вход и более двух выходов. |
|  | Блок  Вывода результата  на печать | Данный блок имеет один вход и один выход. |
| I=N, K, S  Тело цикла | Блок  цикла с параметром (модификации) | I – управляющая переменная цикла, N - начальное значение у. п. ц., K – конечное значение у. п. ц., S – шаг изменения у. п. ц. |
| +(-)  Условие  -(+)  Тело цикла | Блок  цикла с предусловием (цикл пока) | Цикл выполняется до тех пор, пока условие ложно (истинно). Цикл может ни разу не выполнится. |
| Тело цикла  Условие | Блок  цикла с постусловием  (цикл до) | Цикл выполняется до тех пор, пока условие ложно (истинно). Цикл выполнится хотя бы один раз. |
|  | Блок  конца | Данный блок имеет один вход. |
| Имя | Блок подпрограммы | Вызывает подпрограмму с данным именем. Данный блок имеет один вход и один выход. |
|  | Страничные соединители | Соединяют блоки на одной странице. Используются парами. |
|  | Межстраничные соединители | Соединяют блоки на разных страницах. Используются парами. |

Запись ***алгоритма*** на ***языке блок-схем*** состоит из блоков, связанных между собой линиями со стрелочками (линиями переходов). Стрелочки указывают порядок выполнения блоков.

1. Линии в блок-схемах не должны пересекаться
2. Линии поворачивают строго под углом в 90◦

Впрочем, язык блок-схем будет в дальнейшем использоваться лишь как вспомогательное средство при объяснении ***алгоритмических структур***.

1. Величины

Алгоритмы описывают последовательность действий, производимых над некоторыми объектами, определённых условием задачи. Например, при решении задачи по определению суммы для оплаты за электроэнергию такими объектами могут быть даты предыдущего и текущего платежей, предыдущие показания счётчика, текущие показания счётчика, количество истраченной энергии за период, тариф и сумма оплаты.

*В информатике отдельный информационный объект (число, символ, строка, таблица и др.) называется величиной.*

Какие бывают величины можно представить по таблице 1.

*Постоянной (константой)* называется величина, значение которой указывается в тексте алгоритма и не меняется в процессе его исполнения.

*Переменной* называется величина, значение которой меняется в процессе исполнения алгоритма. При исполнении алгоритма в каждый момент времени переменная величина имеет значение, называемое *текущим значением.*

*Имя(идентификатор)* – это обозначение величины в формуле или алгоритме. Транслятор каждому имени, указанному в операторах, присваивает номер ячейки ОЗУ. В эту ячейку и заносится значение величины.

Имя величины состоит из одной или нескольких латинских букв и/или цифр. Имя обязательно начинается с буквы. Рекомендуется выбирать мнемонические имена, т. е. имена, отражающие суть объектов решаемой задачи, например SUMMA, PLAN, CENA и т. д.

Таблица 1:

ВЕЛИЧИНЫ

ПОСТОЯННЫЕ ПЕРЕМЕННЫЕ

характеризуются

ЗНАЧЕНИЯ ИМЯ И ЗНАЧЕНИЯ

Бывают

Входные данные Выходные данные Промежуточные

(Аргументы) (Результат) Величины

Если величину представить как ящик, содержимым которого является некоторое значение, то имя величины – это ярлык, повешенный на ящик.

*Аргументы (входные данные)* – это величины, которые задаются и используются для решения задачи. Указываются в блоках ввода на блок-схемах и операторах ввода.

*Результаты (выходные данные)* – это величины, которые получаются в ходе решения задачи. Указываются в блоках вывода на блок-схемах и операторах вывода.

Величины, которые получаются в ходе решения задачи и используются для получения результата называются *промежуточными (вспомогательными)*. Указываются в блоках действия и операторах присваивания.

Над величинами выполняются некоторые операции. Например:

* Арифметические операции;
* Операции отношения;
* Логические операции.

Объекты, над которыми выполняются операции называются *операндами*. Не всякий объект может быть операндом для выполнения любой операции. Множество величин, объединённых определённой совокупностью допустимых операций, называют величинами определённого *типа*. При составлении алгоритмов используют величины числового (целого или вещественного), символьного, литерного и логического типов.

* Значение величин целого типа – целое число, обозначаются *цел*;
* Значение величин вещественного типа – вещественное число (число с десятичной точкой), обозначаются *вещ;*
* Значение величин символьного типа – один символ (буква, цифра, знак препинания и т. п.), обозначаются *сим*;
* Значение величин литерного типа – последовательность символов (строка), обозначаются *лит*;
* Значение величин логического типа имеют всего два значения: ДА (истина, true, 1) и НЕТ (ложь, false, 0) – обозначаются *лог*.

1. Интерфейс

Чтобы писать и отлаживать, то есть заставлять работать, ***программы*** на ***QB и Pascal***, вам необходимо научиться работать в их среде.

* 1. **Установка**

a) **QB**

Для установки на операционную системы WINDOWS XP и предыдущих версий необходимо скопировать папку QB, с необходимыми программами на диск С: вашего компьютера. Для установки QB на операционную системы WINDOWS 7 и последующих версий необходимо скопировать файл QBasicRus\_DosBox на ваш компьютер.

Запустите его и выполняя инструкции программы установщика установите необходимое ПО.

b) **Pascal**

Для установки на операционную системы WINDOWS XP и предыдущих версий необходимо скопировать установочный файл fpc-2.6.4.x86\_64-win64 на ваш компьютер. Для установки PascalABC на операционную системы WINDOWS 7 и последующих версий необходимо скопировать файл PABCInstall на ваш компьютер.

Запустите установочный файл и выполняя инструкции программы установщика установите необходимое ПО.

* 1. **Запуск**

a) **QB**

Для того чтобы запустить ***QB***, советуем сделать ярлык программы на рабочем столе. Ярлык делается следующим образом:

* Щёлкнуть правой кнопкой мыши на рабочем столе;
* В появившемся окне выбрать позицию *создать → ярлык*
* В появившемся окне в поле «Укажите расположение объекта» нажать на кнопку обзор;
* Далее выбрать в окне «Укажите расположение объекта» локальный диск С: → папку QB( для WINDOWS 7 папку SchoolProg) → файл qb(qbasic) и щёлкните мышкой по клавише OK;
* В окне «Расположение объекта» у вас появится путь к файлу, нажмите клавишу «Далее»;
* Вам предложат ввести имя ярлыка. Можете оставить без изменения или ввести «QB»;
* Нажмите ОК и на рабочем столе у вас появится ярлык для запуска программы.

b)**Pascal**

Для того чтобы запустить PascalABC, советуем сделать ярлык программы на рабочем столе. Ярлык делается следующим образом:

* Щёлкнуть правой кнопкой мыши на рабочем столе;
* В появившемся окне выбрать позицию *создать → ярлык*
* В появившемся окне в поле «Укажите расположение объекта» нажать на кнопку обзор;
* Далее выбрать в окне «Укажите расположение объекта» локальный диск С: → папку Pascal (для WINDOWS 7 папку Program Files(x86) → PABCWork.NET) → файл PascalABCNET.exe и щёлкните мышкой по клавише OK;
* В окне «Расположение объекта» у вас появится путь к файлу, нажмите клавишу «Далее»;
* Вам предложат ввести имя ярлыка. Можете оставить без изменения или ввести «Pascal»;
* Нажмите ОК и на рабочем столе у вас появится ярлык для запуска программы.

При двойном клике мышкой по ярлыку программа QB или Pascal запускается. Программа начинает работать, и вы попадаете в её среду, которая представляет собой, простой текстовый редактор, очень похожий по интерфейсу на **MS-WORD**. Как ни как, их изготовитель – тоже фирма **Microsoft**.

* 1. **Работа с файлами**

Вообще, работа с файлами подробно описана в ***«Наставлении по работе с текстовым редактором»*** Однако, напоминаем, что на собственное имя файла в ***QB*** или ***Pascalе*** выделяется всего 8 символов. Желательно не использовать русских букв. Недопустимы пробелы, запятые, точки, кавычки. Хорошо, если в имени файла зашифрован класс, фамилия, имя автора, назначение или номер программы. Файлы автоматически получают расширение.

Обязательно сохраняйте файлы в текстовом формате!

* 1. **Написание программ**

Для написания ***программы***, нужно владеть хотя бы элементарными навыками работы с ***текстовым редактором***. Какие это навыки? Первоначальный набор текста, его редактирование, выделение фрагментов текста, их копирование, вырезка, вставка. А так же работа с файлами. Всё это также подробно описано в ***«Наставлении…»***.

На что обращаем ваше внимание?

1. Все ***команды***, ***имена переменных***, ***функции*** пишутся исключительно *латинскими* буквами. *Русские* буквы, даже совпадающие по написанию с *латинскими* (и зрительно неразличимые), совершенно не годятся для этого, и вызовут синтаксическую ошибку. Использовать русские буквы можно лишь внутри кавычек в подсказках, текстовых значениях или комментариях.
2. Все ***команды*** и ***функции*** советуем писать *строчными* буквами. В случае синтаксически правильного написания, буквы сами сделаются *заглавными*.
   1. **Запуск программы и возврат в текстовый редактор после её остановки**

a) **QB**

Для того чтобы запустить ***программу***, надо нажать на функциональную клавишу F5. Если этому предшествовала аварийная остановка, то для перезапуска программы надо сделать 🡅Shift + F5.

После окончания работы программы появляется надпись

|  |
| --- |
| **Press any key to continue** |

Это означает «нажмите любую клавишу для продолжения».

После того, как это будет сделано, мы опять окажемся в режиме редактирования ***программы***. Если надо остановить выполнение ***программы*** раньше, чем это произойдет естественным путём (например, при зацикливании), то надо нажать Ctrl + Break.

b) **Pascal**

Для того чтобы запустить ***программу***, надо нажать на функциональную клавишу F9. Если этому предшествовала аварийная остановка, то для перезапуска программы надо сделать Shift+F9.

После окончания работы программы появляется надпись

|  |
| --- |
| **Press any key to continue** |

Это означает «нажмите любую клавишу для продолжения».

После того, как это будет сделано, мы опять окажемся в режиме редактирования ***программы***. Если надо остановить выполнение ***программы*** раньше, чем это произойдет естественным путём (например, при зацикливании), то надо нажать Ctrl + Break.

* 1. Правила написания программ на алгоритмических языках
* Программа на алгоритмическом языке состоит из операторов. Оператор – это указания программиста, направленные на выполнение алгоритма, записанные по правилам алгоритмического языка. Оператор переводится транслятором в набор машинных команд.

При трансляции происходит присваивание именам переменной ячеек ОЗУ и каждый оператор разбивается на отдельные операции. Каждая машинная команда должна содержать указание на выполнение только одного действия (КОП – код операции) и величины над которым производится действия(операнд).

* Операторы записываются в строки.
* В одной строке может быть несколько операторов. Один оператор от другого отделяется чаще всего двоеточием.
* Перед строкой может стоять метка (буквы и/или цифры).
* При записи вещественных чисел, вместо запятой, используется десятичная точка.
* Аргументы функций заключаются в круглые скобки.
* Имена переменных(идентификаторы) обязательно начинаются с латинской буквы и могут состоять из нескольких букв и цифр.
* Операторы делятся на выполняемые и невыполняемые. Выполняемые операторы переводятся транслятором в машинные команды. Невыполняемые операторы обрабатываются транслятором и в машинных кодах не записываются. На невыполняемые операторы нельзя передавать управление.
* Не допускается использование надстрочных и подстрочных знаков (А2→А^2; H2O→H2O; K=→K=F/6).
* Последним оператором в программе должен быть оператор END (исключение составляют подпрограммы).

###### Правила оформления программ[[2]](#footnote-2)

* 1. Первой строкой в***программе*** должна быть строка-заголовок, где коротко и ясно указан её автор (фамилия, имя, класс) и назначение ***программы***. Строка эта является *комментарием*, поэтому она начинается с оператора REM, в противном случае будет выдано сообщение об ошибке(QB). В Pascalе первая строка начинается с служебного слова program.
  2. Последней командой в ***программе*** является команда **END**, что значит *конец*.
  3. Заголовок ***программы*** и ***команда*** **END** пишутся с первой позиции. Тело же ***программы*** сдвигается на несколько позиций(1-3) вправо.
  4. Старайтесь писать ***программу*** так, чтобы в одной строке находилась одна ***команда***.

###### Порядок решения задач[[3]](#footnote-3)

Скоро вам придётся заняться решением ***задач***. Сразу оговорим порядок этой работы. ***Задача*** даётся либо в печатном виде, либо устно под запись. В последнем случае обучающиеся должны записать условие в тетрадь, в случае необходимости попросить повторить, задать вопросы. В процессе работы задание не повторяется. Каков же порядок решения ***задачи***?

* 1. Внимательно прочесть условие ***задачи*** и разобраться с ***входными(аргументы)*** и ***выходными(результаты) данными***[[4]](#footnote-4)***,*** а таже со ***вспомогательными(промежуточными) величинами***.
  2. Составить несколько ***тестов***[[5]](#footnote-5), желательно на все частные случаи.
  3. Составить ***алгоритм***, написать и отладить ***программу***.
  4. Протестировать её.
  5. В случае успешного прохождения ***тестов*** позвать учителя для сдачи ***программы***. В противном случае найти ошибку на одном из предыдущих этапов работы и исправить её.

Особо хотим подчеркнуть, что решать следует именно ту ***задачу***, которая задана, а не ту, которую вы хотите. Порой бывает и так, что ученик сделал ***программу*** совершенно не под свою ***задачу***, и хочет поменять условие ***задачи***. Это недопустимо. Творчество надо проявлять именно на этапе ***программирования*** и ***алгоритмизации***, а условие ***задачи***, ***входные*** и ***выходные данные*** – неприкосновенны!

###### Переменные

***Переменная*** - это одно из важнейших понятий математики и информатики. Даётся оно ещё в начальной школе, но лишь на интуитивном уровне. В результате на вопрос *«Что такое* ***переменная****?»*, от учеников, даже хороших, в лучшем случае можно получить ответ вроде *«Это когда числа обозначаются буквами»*. Впрочем, перечитав много книг, мы не нашли удовлетворяющего определения. Лучшим из них было: *«****Переменная***- *это триада:* ***имя, тип, значение****»*[[6]](#footnote-6). Очень важно, что все три ***атрибута переменной*** тут указаны явно.

Необходимо усвоить:

* **Переменная** - это более высокий способ представления **данных**, по сравнению с конкретными **значениями**.
* **Переменная** – имеет три атрибута: **имя, тип, значение**.
* **Имя** и **тип переменной** неизменны внутри **программы**, а **значение** может в любой момент измениться.
* Старое **значение переменной** при этом пропадает, если оно может понадобиться, то необходимо сохранить его в другой **переменной**.
  1. **Имя переменной**

***Имя переменной*** - это её обозначение, например: **a, b, x, у**.

Все буквы – *латинские!*

Русские и латинские буквы, совпадающие по написанию, это *разные* символы и их замена влечёт ошибку!

Лучше сразу договориться использовать в устной речи именно *латинские*, а не *английские* названия букв, иначе будет путаница (так как не все знают английский). Впрочем, на уроках математики так и делается.

Заглавные *не отличаются* от строчных, но строчные предпочтительнее, т.к. ближе к математической традиции.

В качестве имён переменных нельзя использовать служебные слова ***QB и* Pascal**: имена ***команд, операций*** и встроенных ***функций***.

Отличия от математических обозначений:

* ***имя переменной*** не обязательно одна буква, но любое сочетание букв и цифр (до восьми символов), начинающаяся с буквы;
* невозможность использования верхних и нижних маленьких индексов. (**a1, b12, x, x0** ... - допустимые имена.)
  1. **Тип переменной**

***Тип переменной*** тоже очень важное понятие. Из школьного курса математики вы знаете, что числа бывают разные: натуральные, целые, рациональные, вещественные (или действительные). В этой цепочке предыдущий тип входит в следующий за ним как частный случай. Во всех языках программирования числовые типы переменных тоже бывают разные.

Но числа в программировании существенно отличаются от чисел в классической математике. В чём же состоит главное отличие? Во-первых, числовая ось, как она понимается в классической математике, ***бесконечна***, то есть, какое бы большое число мы ни взяли, всегда найдётся больше, и какое бы малое число мы ни взяли, всегда найдётся меньше. Во-вторых, числовая ось (действительных чисел) ***непрерывна***, какие бы, сколь угодно близкие, два числа мы ни взяли, всегда между ними найдётся третье. А значит, между любыми двумя числами находится бесконечное множество чисел! Таким образом, в классической математике числа обладают двумя свойствами бесконечности, внешней и внутренней. Это, конечно, здорово, но как это можно реализовать на компьютере? Ведь компьютерные ресурсы, в том числе и память, конечны. А с помощью конечных ресурсов нельзя сделать бесконечность. Поэтому числа в любом языке программирования всегда имеют нижний и верхний предел, жёсткие рамки, в которых только и могут находиться. Что касается вещественных чисел, то они имеют и предел точности представления, то есть максимально допустимое количество десятичных знаков. Таким образом, в «компьютерной» математике (она называется ***дискретной***, то есть «прерывной»), в отличие от математики непрерывной, целые и вещественные числа качественно не различаются. Разница между ними количественная, под их хранение выделяется различное количество компьютерной памяти.

В Бейсике и Паскале основными числовыми ***типами*** являются целый и вещественный, остановимся на них подробней:

1. ***целый тип***.

Числа этого типа могут принимать значения от **–32768** до **+32767**. Это связано с тем, что под такое число в компьютерной памяти выделяется 2 байта, или 16 бит. Один из них используется для хранения знака, остальные – значения числа. Если вы не поленитесь подсчитать, то сами убедитесь, что 215 = 32768. Описать переменную такого типа можно двумя способами: приписать к её имени знак %, либо с помощи специальной ***команды описания***:

Бейсик:

|  |
| --- |
| **DIM <*имя переменной>* AS INTEGER** |

Паскаль:

|  |
| --- |
| VAR  <***имя переменной>:*INTEGER** |

1. ***вещественный тип***.

Числа вещественного типа могут принимать значения от **-9999999** до **9999999** в обычной форме. В форме «с плавающей запятой» от **–3.4E38** до **+3.4E38**. Впрочем, на этой форме мы сейчас останавливаться не будем. Под хранение ***переменных*** этого ***типа*** выделяется 4 байта.

Обратите внимание на то, что, в отличие от школьной математики, и не только школьной математики, в качестве знака, разделяющего вещественное число на целую и дробную часть, служит не *запятая*, а *точка*! Например, надо писать **123.34**.

Организовать ***переменную*** такого ***типа*** в Бейсике можно двумя способами: по умолчанию (то есть, ничего не приписывая к ***имени переменной***), либо с помощью специальной ***команды описания***:

|  |
| --- |
| **DIM** <*имя переменной>* **AS SINGLE** |

Впрочем, во многих случаях, работая на Бейсике можно позволить себе не вдаваться в такие детали. Кроме вышеназванных числовых типов, есть и другие, но на них сейчас останавливаться не будем.

В Паскале ***переменная*** такого ***типа*** описывается в разделе описания ***переменных:***

|  |
| --- |
| VAR  <***имя переменной>:*REAL** |

Однако кроме числовых типов в Бейсике и Паскале есть и символьный тип.

Как уже говорилось, в Бейсике имеются разные возможности описания типов переменных: явное описание с помощью команды **DIM**, посредством добавления (или не добавления) к имени переменной специального знака, и по умолчанию.

Остановимся подробней на ***символьных переменных***.

*Символьные* значения заключаются в кавычки, например: "клод", "3434"... К имени символьных переменных приписывается знак $, например a$, если она не была описана командой **DIM**, либо этой ***командой***:

|  |
| --- |
| **DIM** <*имя переменной>* **AS STRING** |

Переменные a, a%, a$ - различаются и могут быть одновременно использованы в программе.

В Паскале существует два вида ***символьных переменных***: для одного символа и строки.

Один символ описывается в разделе описания ***переменных*** с помощью ***команды CHAR,*** строка описывается с помощью ***команды* STRING.**

|  |
| --- |
| VAR  <***имя переменной>: CHAR***  <***имя переменной>:*STRING** |

В Паскале переменные описываются в специальном разделе. Он начинается с служебного слова ***var.***Далее записываются имена переменных через запятую, ставится двоеточие и записывается тип величины. Приведём пример записи величин:

var

a,b,c:integer; (целые числа)

x,y:real; (вещественные числа)

z:string; (строка)

d:char; (один символ)

* 1. **Функции символьных величин и действия с ними:**

Пусть имеются символьные ***переменные***:

**a$ = "123456789", b$ = "bbb", c$ = "ccc"**.

a) **QB**

|  |  |  |  |
| --- | --- | --- | --- |
| LEFT$(a$,3)="123" | | «Оторвать» *слева* от строки a$ и сколько. | |
| RIGHT$(a$,4)="6789" | | «Оторвать» *справа* от строки a$ и сколько. | |
| MID$(a$,4,2)="45" | | «Вырезать» из от строки a$, с какой позиции и сколько. | |
| LEN(a$)=9 | | Длина (количество символов) строки a$ | |
| VAL("123")=123 | | Преобразование «числового» текста в число. | |
| STR$(456)=" 456" | | Преобразование числа в текст. Слева добавляется пробел, если число неотрицательное. | |
| CHR$(код) | | Выдаёт символ по введённому коду | |
| ASC(символ) | | Выдаёт код введённого символа | |
| LTRIM$(строка) | | Удаляет все пробелы слева | |
| b$ + c$ = "bbbccc" | | c$ + b$ = "cccbbb" | | Сложение | |

От перемены мест символьных слагаемых сумма ***меняется!***

b) **Pascal**

|  |  |
| --- | --- |
| Length(строка) | Возвращает длину строки |
| Concat(s1,s2,[s3…,sn])=  =s1+s2[+s3….+sn] | Функция выполняет слияние строк-параметров, которых может быть произвольное количество. Каждый параметр является выражением строкового типа. Если длина строки-результата превышает 255 символов, то она усекается до 255 символов. Данная функция эквивалентна операции конкатенации «+» и работает немного менее эффективно, чем эта операция. |
| Copy(s,a,b) | Функция возвращает подстроку, выделенную из исходной строки s длиной b символов, начиная с символа под номером a. |
| Delete(s,a,b) | Процедура удаляет из строки-параметра s подстроку длиной a символов, начиная с символа под номером b. |
| val (s,a,b) | Процедура преобразует строковую запись числа s в числовое представление a. Результат может быть как целой, так и действительной переменной. Если встречается недопустимый(с точки зрения правил записи чисел) символ, то преобразования не происходит, а в b записывается позиция первого недопустимого символа. Выполнение программы при этом не прерывается, диагностика не выдаётся. Если после выполнения процедуры b равно 0, то это свидетельствует об успешно произошедшим преобразованием. |
| STR(a,b) | Процедура преобразует численное выражение a в его строковое представление и помещает результат в b. |
| CHR(код) | Функция возвращает символ по коду. |
| Ord(символ) | Выдаёт код символа |
| Insert(s,a,b) | Процедура предназначена для вставки строки s в строку a, начиная с символа b этой строки. |
| Pos(s,a) | Функция производит поиск в строке a подстроки s. Результатом функции является номер первой позиции подстроки в исходной строке. Если подстрока не найдена, то функция возвращает 0. |
| UpCase(a) | Если a – строчная латинская буква, то функция возвращает соответствующую прописную латинскую букву, в противном случае символ a возвращается без изменения. |

* 1. **Значение переменной**

***Значение переменной*** это то, чему она равна в настоящее время. Оно с помощью специальных ***команд программы*** в любой момент может *перемениться*. В Бейсике, в отличие от других ЯП, пока ***переменной*** не присвоено значение, она считается равной **0** (числовая) или **""** (текстовая). Если использовать это свойство, то программы могут получиться значительно короче, чем, скажем, на Паскале. Однако, это не всегда безопасно, да и ***программа*** получается понятней, если ***переменным*** присваивать начальные значения, даже нулевые и пустые.

***Переменную*** можно представить как *ящик*, на котором написано ***имя*** и в котором находится ***значение***. При получении нового ***значения*** старое *забывается!*

1. ***Команда присваивания. Выражения. Функции.***
   1. **Команда присваивания**

***Команда присваивания*** имеет вид:

a) ***QB***

|  |
| --- |
| LET *<Имя переменной>* = *<выражение>* |

b) ***Pascal***

|  |
| --- |
| *<Имя переменной>* **:=** *<выражение>* |

Это означает: «пусть ***переменной*** с таким-то ***именем*** будет присвоено такое-то значение». Вообще, служебное слово **LET** можно и не писать, но в нём заключён большой смысл, поэтому советую не лениться.

Примеры:

|  |  |
| --- | --- |
| LET d = 3  LET i = i + 1  LET a = x \* b  LET x1 = x2 / 2  LET b$ = "roro" + c$ | Тип ***выражения*** должен совпадать с ***типом переменной***!!! |
| d:=3;  i:=i+1;  a:=x\*b;  x1:=x1/2;  b:=’roro’+c; |

***Команда присваивания*** выполняется так:

1. вычисляется ***значение*** выражения, стоящее *справа* от знака равенства. Если в нём есть ***переменные***, то находятся и подставляются их текущие ***значения***.
2. ***Значение***, которое получилось в первом пункте, считается новым ***значением*** ***переменной***, ***имя*** которой стоит *слева* от знака равенства*.* Старое значение при этом пропадает.

Обратите внимание:

* *Слева* от знака равенства находится ***имя*** только одной ***переменной***!
* Не путать *левую* и *правую* часть!
* ***Переменные***, ***имена*** которых используются в *правой* части ***команды присваивания***, не меняют своих ***значений***. Они *не отдают* свои значения, но происходит процесс *копирования информации*.
* ***Имя переменной***, стоящей *слева* от знака равенства, может использоваться в выражении в *правой* части.

Внимание!!! В пункте II-2 приводятся не ***команды присваивания***, а просто объяснение работы ***функций***.

* 1. **Выражения**

Теперь подробней поговорим о ***выражениях***. Понятие ***выражение*** полностью совпадает с одноимённым математическим понятием.

***Выражение*** это последовательность ***имён переменных*** и констант, связанных между собой знаками операций, ***функций***, скобками, в соответствии с существующими правилами.

Обратите внимание на то, что знаки некоторых арифметических операций (и ***функций***) отличаются, а некоторые, в силу «экранных» ограничений пишутся особо:

|  |  |  |
| --- | --- | --- |
| **+** | сложение | **+** |
| **-** | вычитание | **-** |
| **\*** | умножение | **\*** |
| **/** | деление | **/** |
| **^** | возведение в степень  (в Бейсике) | **NO** |

Дроби пишутся «в один этаж» с помощью знака деления и скобок. Скобки можно использовать только круглые!

* 1. **Встроенные функции**

Поясним для начала слово «встроенные». Это значит сделанные изготовителями системы программирования (а можно создавать и свои собственные). Предполагается, что понятие ***функции*** вам знакомо из курса математики. Ниже приводятся наиболее употребляемые математические функции ***QB***[[7]](#footnote-7) ***и Паскаля[[8]](#footnote-8)***. Что касается функций символьных величин, то они были приведены в разделе ***IV-12***.

a) ***QB***

|  |  |  |
| --- | --- | --- |
| \* | ABS ( x ) | абсолютная величина (модуль) числа х |
|  | COS ( x ) | Косинус числа х, выраженного в радианах |
|  | SIN ( x ) | Синус числа х, выраженного в радианах |
|  | INT ( x ) | целая часть |
| \* | LOG ( x ) | натуральный логарифм |
| \* | a^b | а в степени b |
|  | SGN ( x ) | знак числа (-1, 0, 1) |
| \* | TAN ( x ) | Тангенс числа х, выраженного в радианах |
| \* | ATAN(x) | Арктангенс х |
| \* | EXP(x) | Число е (2,7) в степени х |
| \* | SQR ( x ) | квадратный корень |
| \* | а\b | Целочисленное деление a на b |
| \* | A mod b | Остаток от целочисленного деления a на b |
| \* | TIMER | время в секундах. |
| \* | RND | 0 ≤ случайное число < 1 |
| \* | TIME$ | Значением является текстовая величина, в которой записаны часы, затем минуты, потом секунды |
| \* | DATE$ | Значением является текстовая величина, в которой записаны номер месяца, затем число, потом год |

Если после ***имени функции*** стоит знак $, то функция её ***значение***символьное, в противном случае - *числовое*. ***Аргументы*** - аналогично!!!

b) ***Pascal***

|  |  |  |
| --- | --- | --- |
|  | SIN(x) | Синус числа х, выраженного в радианах |
|  | СОS(x) | Косинус числа х, выраженного в радианах |
| \* | ATAN(x) | Арктангенс х |
|  | EXP(x) | Число е (2,7) в степени х |
|  | Ln(x) | Натуральный логарифм х |
| \* | Pi | Число пи |
| \* | SQRТ(x) | Корень квадратный из х |
| \* | ABS(x) | Модуль х |
| \* | SQR(x) | х в квадрате |
| \* | а div b | Целочисленное деление a на b |
| \* | a mod b | Остаток от целочисленного деления a на b |
| \* | Frac(x) | Возвращает число, равное дробной части числа х. |
| \* | Inc(x,y) | Увеличивает значения числа x на y. Если число y не указано, то увеличение происходит на 1. |
| \* | Dec(x,y) | Уменьшает значения числа x на y. Если число y не указано, то уменьшение происходит на 1. |
| \* | Random(X) | Возвращает случайное целое число в диапазоне 0..Х. Если аргумент опущен, то возвращается случайное вещественное число от 0 до 1. |
| \* | Round(x) | Функция округляет число х. Возвращаемое значение имеет тип longint |
| \* | Power(x,y) | Возвращает x в степени y |
| \* | Trunc(x) | Возвращает число, равное целой части числа х. (Происходит отбрасывание дробной части числа х – тип longint) |
| \* | INT(число) | Определяет целую часть числа. (Происходит отбрасывание дробной части числа х – тип real) |
| \* | Eoln(имя файла) | Возвращает **true**, если была прочитана последняя литера текущей строки. |
| \* | Eof(имя файла) | Возвращает **true**, если была прочитана последняя литера файла (попытка дальнейшего чтения ведет к ошибке). |
| \* | Odd(целое выражение) | Возвращает **true**, если параметр – нечетный, в противном случае возвращает **false**. |

Аргументами ***функций*** должны быть величины соответствующих типов, *константы* (т.е. конкретные значения) или *выражения*. *Авторы* ***программ*** должны заботиться о том, чтобы аргументы ***функций*** получали лишь *допустимые* значения! Иначе неизбежно аварийное окончание работы ***программы***.

Функции используются в *выражениях* (в правой части ***команды присваивания*** или в ***отношениях***).

1. Команды ввода, вывода и очистки экрана .
   1. **Команда ввода**

Мы уже знаем, что ***переменные*** свои значения могут получать с помощью ***команды присваивания***. Однако, это не единственный способ. Бывает, что дать значение ***переменной*** необходимо во время работы ***программы***, а не во время её написания. В этом случае ***команда присваивания***не может помочь, и нужен какой-либо другой способ. Такой способ есть - ***команда ввода*** значений переменных -

a) ***QB***

- «**INPUT**». Ее синтаксис:

|  |
| --- |
| **INPUT** [*<подсказка>*, или ;] *<список переменных>* |

где:

*<подсказка>* - любой текст, заключённый в кавычки;

*<список переменных>* - ***имя*** одной ***переменной*** либо несколько ***имён переменных***, разделённых запятыми.

Как и в других местах, здесь в квадратные скобки берётся необязательная часть.

Пример:

|  |
| --- |
| INPUT "Введите длину и ширину:", a, b |

Работает команда ввода так:

* выводится на монитор подсказка, если она есть, иначе вопросительный знак;
* если после подсказки стоит точка с запятой, то она завершается вопросительным знаком;
* приостанавливается выполнение ***программы****,* пока не будут введены через запятую столько констант и таких типов, сколько и каких типов ***имён переменны****х* перечислено в <списке переменных> и не будет нажата клавиша **↵** Enter;
* ***переменные*** получают ***значения*** соответствующих введённых констант;
* продолжается выполнение ***программы****.*

b) ***Pascal***

- «read[ln]». Ее синтаксис:

|  |
| --- |
| **Read[ln](** *<список переменных>***);** |

*<список переменных>* - ***имя*** одной ***переменной*** либо несколько ***имён переменных***, разделённых запятыми.

Как и в других местах, здесь в квадратные скобки берётся необязательная часть.

Пример:

|  |
| --- |
| **Read[ln](****a, b);** |

Работает команда ввода так:

* выводится на монитор курсор;
* приостанавливается выполнение ***программы****,* пока не будут введены через пробел столько констант и таких типов, сколько и каких типов ***имён переменны****х* перечислено в <списке переменных> и не будет нажата клавиша **↵** Enter;
* ***переменные*** получают ***значения*** соответствующих введённых констант;
* продолжается выполнение ***программы****.*

Как это происходит и в ***команде присваивания***, старое ***значение*** вводимой ***переменной*** забывается.

Отметим, что, в отличие от ***команды присваивания***, вводить надо именно *константы*, а не *выражения!*

Подробней остановимся на *подсказках* (в паскале подсказка записывается с помощью оператора вывода)*.* Зачем они нужны? Представим себе, что в какой-то ***программе*** подсказок нет, и пользователь, работающий с этой ***программой***, сидит в растерянности. Компьютер остановился и чего-то ждёт, а пользователь не знает, что делать. Только с помощью подсказки и можно сообщить ему, сколько и каких значений требуется ввести, что они значат… Надо научиться писать короткие, но толковые подсказки!

Обратите внимание на отличия ***команд присваивания*** и ***ввода***:

* С помощью одной **команды присваивания** можно задать **значение** лишь одной **переменной**, а с помощью одной **команды ввода** – нескольким.
* В случае **команды присваивания** вопрос о **значении переменной** решает **программист**, а в случае **команды ввода** – **пользователь**.
* В **команде присваивания** можно писать и формулы, а в **команде ввода** **пользователь** имеет право вводить лишь конкретные **значения**.
* Если в каком-то случае программист и **пользователь** – одно и то же лицо, **команду ввода** можно заменить **командой присваивания**, но не наоборот.
  1. **Команда Вывода**

Эта ***команд****а* очень похожа на предыдущую, но выполняет *обратное* действие. Если ***команда*** ввода *вводит* ***значения переменных*** в оперативную память компьютера, то ***команда*** вывода их *выводит*. Правда, выводить она может не только значения переменных, но и выражений, в частности, константы.

Синтаксис этой ***команды***:

a) ***QB***

|  |
| --- |
| **PRINT** *<список вывода>* |

где: <список вывода> – одно или несколько, перечисленных через запятую или точку с запятой, ***имён переменных*** или ***выражений****.*

Работает команда **PRINT** так: ищет ***значения переменных*** и вычисляет значения ***выражений***, перечисленных в *списке вывода,* и выводит их на монитор в соответствующем порядке. При этом расстояние между ними зависит от разделителя:

* при точке с запятой – через пробел;
* при запятой - автоматическое форматирование (разбивает экран примерно на пять частей и в каждой части значение имени переменной или выражения).

b) ***Pascal***

|  |
| --- |
| **Write[ln](** *<список вывода>***);** |

где: <список вывода> – одно или несколько, перечисленных через запятую или точку с запятой, ***имён переменных*** или ***выражений****.*

Работает команда **Write** так: ищет ***значения переменных*** и вычисляет значения ***выражений***, перечисленных в *списке вывода,* и выводит их на монитор в соответствующем порядке подряд, без разделения пробелом. Если используется команда **Writeln,** то после вывода всех данных курсор переходит на следующую строчку.

* 1. **Команда очистки экрана**

a) ***QB***

Это, пожалуй, самая простая ***команда - CLS***. Получив её, Бейсик стирает с экрана всю информацию, которая находилась на нём.

b) ***Pascal***

Для использования команды очистки экрана в Паскале необходимо подключить специальный модуль **CRT.** Подключив его, даём команду ***clrscr.***

Разберём теперь несколько ***задач***, для решения которых, достаточно вышеописанных команд.

**Задача VII-1**

|  |
| --- |
| У пяти человек было по некоторому количеству яблок. Каждый съел по половине своих яблок. Потом первый отдал 2 яблока четвёртому, а второй и пятый поделили свои яблоки поровну. Сделать программу для подсчёта, у кого сколько осталось яблок. |

Вспомним, каков порядок решения задач.

Во-первых, вчитаемся в условие и разберёмся, сколько ***входных***, сколько ***выходных данных*** должно быть, каковы их ***типы***. Очевидно, что в этой ***задаче*** должно быть пять ***входных данных*** (числовые) и пять ***выходных*** (тоже числовые). По логике задачи это даже одни и те же ***переменные***. Можно чуть забежать вперёд, и заранее продумать, как эти переменные назвать. Можно так: **a**, **b**, **c**, **d**, **e**. Однако гораздо лучше: **a1**, **a2**, **a3**, **a4**, **a5**.

Входные данные: a1, a2, a3, a4, a5 – цел., количество яблок

Выходные данные: a1, a2, a3, a4, a5 – цел., количество оставшихся яблок

Вспомогательная величина: b – цел., количество яблок у 2 и 5 мальчиков

Во-вторых, составим несколько ***тестов***:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 тест | | | | | | 2 тест | | | | | |
|  | 1 | 2 | 3 | 4 | 5 |  | 1 | 2 | 3 | 4 | 5 |
| ***Входные данные*** | **10** | **10** | **10** | **10** | **30** |  | **10** | **20** | **30** | **40** | **60** |
| ***Выходные данные*** | **3** | **10** | **5** | **7** | **10** |  | **3** | **20** | **15** | **22** | **20** |

Третий этап – написание алгоритма. Мы можем написать его словесно или нарисовать блок-схему.

|  |
| --- |
| 1. Начало 2. Описание переменных 3. Ввод аргументов 4. Деление яблок пополам(Каждый из мальчиков съел половину своих яблок) 5. Деление яблок между ребятами 6. Вывод результата 7. Конец |

|  |  |
| --- | --- |
| REM Иванов Иван 10А  CLS  DIM a1, a2, a3, a4, a5, b AS INTEGER  INPUT "Введите 5 чисел", a1, a2, a3, a4, a5  LET a1 = a1 / 2  LET a2 = a2 / 2  LET a3 = a3 / 2  LET a4 = a4 / 2  LET a5 = a5 / 2  LET a1 = a1 - 2  LET a4 = a4 + 2  LET b = (a2 + a5) / 2  LET a2 = b  LET a5 = b  PRINT a1; a2; a3; a4; a5  END | Program Ivanov\_Ivan\_10a;  Uses crt;  Var  a1,a2,a3,a4,a5,b:integer;  begin  clrscr;  writeln(’Введите 5 чисел через пробел’);  readln(a1,a2,a3,a4,a5);  a1:=a1/2;  a2:=a2/2;  a3:=a3/2;  a4:=a4/2;  a5:=a5/2;  b:=(a2+a5)/2  a2:=b;  a5:=b;  writrln(a1,’ ’,a2,’ ’,a3,’ ’,a4,’ ’,a5  end. |

Четвёртый этап – написание ***программы***. Она получится примерно такая(левая программа на Бейсике, правая – на Паскале):

***Задача*** эта проста лишь на первый взгляд. Во-первых, надо каждой команде найти своё место. Во-вторых, здесь есть одна ловушка, в которую легко попасть начинающему программисту. Найдите её самостоятельно.

Пятый этап – ***тестирование программы***. Запускаем её, вводим ***входные данные*** из подготовленных нами ***тестов***, сравниваем то, что выдает компьютер с тем, что мы подсчитали вручную. Очевидно, что в этой ***программе*** всё правильно.

**Задача VII-2**

|  |
| --- |
| Задаётся фамилия и имя ученика. Сделать программу, определяющую первые и последние буквы в имени и фамилии, а также количества символов в фамилии и имени. |

Приступим к решению этой ***задачи***, аналогично предыдущей.

Во-первых, вчитаемся в условие и разберёмся, сколько ***входных***, сколько ***выходных данных*** должно быть в этом случае, каковы их ***типы***. В этой ***задаче*** должно быть два ***входных данных*** (текстовые) и шесть ***выходных*** (четыре текстовых и два числовых).

Входные данные: f, i – сим., фамилия и имя ученика

Выходные данные: f1,f2,i1,i2 – сим., первая и последняя буква фамилии и имени ученика; nf,ni – цел., количество символов фамилии и имени ученика

Теперь составим несколько ***тестов***:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ***Входные данные*** | | ***Выходные данные*** | | | | | |
| № теста | Фамилия | Имя | 1-я буква фамилии | последняя буква фамилии | 1-я буква имени | последняя буква имени | Количество символов в фамилии | Количество символов в имени |
| 1 | **Иванов** | **Николай** | **И** | **в** | **Н** | **й** | **6** | **7** |
| 2 | **Гусев** | **Тимофей** | **Г** | **в** | **Т** | **й** | **5** | **7** |

Составим ***алгоритм:***

|  |
| --- |
| 1. Начало 2. Описание переменных 3. Ввод аргументов 4. Определение первой и последней буквы фамилии 5. Определение первой и последней буквы имени 6. Определении количества букв в фамилии 7. Определении количества букв в имени 8. Вывод результата 9. Конец |

Напишем программу:

|  |  |
| --- | --- |
| REM Иванов Иван 10А  CLS  DIM f,i,f1,f2,i1,i2 AS STRING  DIM nf,ni AS INTEGER  INPUT "Введите фамилию и имя", f, i  LET f1 = LEFT$(f, 1)  LET f2 = RIGHT$(f, 1)  LET i1 = LEFT$(i, 1)  LET i2 = RIGHT$(i, 1)  LET nf = LEN(f)  LET ni = LEN(i)  PRINT f1; f2; i1; i2; nf; ni  END | Особо добавить тут нечего, всё просто. Вывод результатов, правда, не очень понятный.  Лучше, конечно, было бы не полениться, и на каждое выходное данное написать свою команду вывода с комментарием, что и советуем вам сделать. |
| Program Ivanov\_Ivan\_10a;  Uses crt;  Var  f,i,f1,f2,i1,i2:string;  nf,ni:integer;  begin  clrscr;  writeln(’Введите фамилию и имя через пробел’);  readln(f,i);  nf:=length(f);  ni:=length(i);  f1:=copy(f,1,1);  f2:= copy(f,nf,1);  i1:= copy(i,1,1);  i2:= copy(i,ni,1);  writrln(f1,’ ’,f2,’ ’,i1,’ ’,i2,’ ’,nf,’ ’ni  end. |

Протестируем программу:

Запускаем её, вводим ***входные данные*** из наших ***тестов***, сравниваем то, что выдает компьютер с тем, что у нас. Очевидно, что и в этой программе всё правильно.

***Вопросы для самоконтроля:***

* 1. Зачем нужна ***команда присваивания***?
  2. Зачем нужна ***команда*** **INPUT**?
  3. Можно ли сказать, что они делают одно и тоже?
  4. Можно ли сказать, что они полностью взаимозаменяемы?
  5. Можно ли в ***программе*** обойтись без ***команды присваивания***?
  6. Зачем нужна ***команда*** **PRINT**?
  7. ***Команды*** **INPUT** и **PRINT** делают одно и тоже?
  8. Можно ли в ***программе*** обойтись без ***команды*** **PRINT**?
  9. Зачем нужны ***переменные***?
  10. Можно ли без них обойтись?
  11. Можно ли изменить ***имя переменной***?
  12. Можно ли изменить ***тип переменной***?
  13. Можно ли изменить ***значение переменной***?
  14. Зачем нужна ***команда*** **CLS**?
  15. Зачем нужны ***тесты***?

1. Структура графического экрана. Основные графические команды.

* 1. **Структура графического экрана**

В ***IV-11*** уже шла речь о понятиях ***бесконечность*** и ***конечность***, ***непрерывность*** и ***дискретность***, применительно к числам. И в школьной геометрии понятия плоскости и прямой подразумевают ***бесконечность*** и ***непрерывность***. Однако, по тем же причинам, по которым числа в компьютере дискретны и конечны, дискретную и конечную природу имеет и графический экран. Точек на нём конечное число, и располагаются они в виде прямоугольной таблицы. Количество возможных цветов тоже конечно. Чем лучше графический режим, тем точек больше, как и количество цветов, а размеры их и расстояния между ними меньше. Но количество точек и возможных цветов *конечно большое*, а размеры точек и расстояния между ними - *конечно малые!* Координаты точек имеют целые значения. Если в результате вычислений или ввода они получают значения не целые, то округляются до целого.

***Между двумя соседними точками ничего нет!***

a) ***QB***

В ***QB*** имеется несколько режимов графического экрана, самый лучший из них, на наш взгляд, двенадцатый. Для того чтобы он заработал, надо дать команду:

|  |
| --- |
| SCREEN 12 |

Система координат выглядит так: Начало её, то есть точка с нулевыми координатами, находится в левом верхнем углу. Если обозначить координаты как это принято в математике, **x** и **y**, то **x** растёт вправо, а **y** – вниз. Вообще, это знакомая нам декартова прямоугольная система координат. Но вид её несколько необычен. Кроме дискретности и конечности, от того, что мы привыкли видеть на уроках математики, её отличает направление роста **y** вниз и отсутствие на видимом экране трёх четвертей. Максимально допустимое значение **x** – 639, а **y** – 479.

И так, каждая точка задаётся координатами, парой чисел, сколько точек отступить слева (**x)**, и сколько сверху (**y)**. В командах и функциях координаты пишутся в скобках через запятую, каждая на своём месте.

Каждая точка может быть окрашена в один из 16 цветов, которые имеют номера от 0 до 15. Нуль соответствует чёрному цвету, а 15 – белому. Остальные цвета показаны в таблице. Маловато, конечно, цветов для написания «красивых» программ, да и точек маловато. Всё-таки этот Бейсик разрабатывался давно, когда ресурсы компьютеров были гораздо меньше.

Таблица 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N  цвета | яркость | R | G | B | цвет |
| 0 | 0 | 0 | 0 | 0 | Чёрный |
| 1 | 0 | 0 | 0 | 1 | Синий |
| 2 | 0 | 0 | 1 | 0 | Зелёный |
| 3 | 0 | 0 | 1 | 1 | Голубой |
| 4 | 0 | 1 | 0 | 0 | Тёмно-красный |
| 5 | 0 | 1 | 0 | 1 | Фиолетовый |
| 6 | 0 | 1 | 1 | 0 | Коричневый |
| 7 | 0 | 1 | 1 | 1 | Белый |
| 8 | 1 | 0 | 0 | 0 | Серый |
| 9 | 1 | 0 | 0 | 1 | Светло синий |
| 10 | 1 | 0 | 1 | 0 | Светло зелёный |
| 11 | 1 | 0 | 1 | 1 | Светло голубой |
| 12 | 1 | 1 | 0 | 0 | Ярко красный |
| 13 | 1 | 1 | 0 | 1 | Розовый |
| 14 | 1 | 1 | 1 | 0 | Жёлтый |
| 15 | 1 | 1 | 1 | 1 | Ярко белый |

* 1. **Графические команды**

Первую графическую ***команду***, ***команду*** перехода в графический режим, была упомянута в предыдущем пункте. Эта команда даётся, как правило, один раз в самом начале программы, по крайней мере, до того, как понадобится другой режим экрана. Ниже перечислены главные команды ***QB*** для работы с графикой:

|  |  |  |
| --- | --- | --- |
| № | Вид команды | Комментарии |
| 1 | PSET (x, y), c | Отметить на экране точку по заданным координатам, заданного цвета. |
| 2 | LINE (x1, y1) – (x2, y2), c | Две точки, заданные своими координатами, соединить отрезком прямой линии, заданного цвета. Эта ***команда*** обладает и дополнительными возможностями, о которых речь ещё впереди. |
| 3 | CIRCLE (x, y), r, c | Начертить на экране окружность по заданному центру, заданного радиуса и цвета. В своё время вы узнаете дополнительные возможности и этой ***команды***. |

Начав пользоваться этими ***командами***, вы сразу ощутите дискретность компьютерной графики. Вообще, графические ***команды*** этим не исчерпываются, а есть ещё и ***функции***. Те из вас, кто захочет с ними ознакомиться, смогут сделать это, читая эту книжку.

b) ***Pascal***

Для работы в графическом режиме необходимо подключить модуль **GraphABC**. Система координат и цвета в Паскале такие же как и в Бейсике. Поэтому и работа с ними строится так же. Приводим команды, необходимые для работы с графикой в Паскале.

|  |  |  |
| --- | --- | --- |
| № | Вид команды | Комментарии |
| 1 | SetPenColor(c) | Задаёт цвет рисования линий. Параметр с – число от 0 до 2563-1 |
| 2 | SetPixel(x, y, c) | Закрашивает на экране цветом с точку с координатами х, у |
| 3 | LINE (x1, y1, x2, y2) | Рисует отрезок из точки с координатами х1, у1 в точку с координатами х2, у2. |
| 4 | CIRCLE (x, y, r) | Рисует на экране окружность в центре с координатами х, у и радиусом **r** |

Теперь для закрепления этой темы рассмотрим ***задачи***.

**Задача VIII-1**

|  |
| --- |
| Сделать ***программу***, которая нарисует на экране домик, у которого высота и ширина равны. Должно быть одно квадратное окно, расположенное симметрично относительно стен и круглое окошко на чердаке. |

Давайте разбираться с ***входными*** и ***выходными данными***. В условии не сказано, какого размера должен быть дом, в каком конкретно месте экрана он расположен. Строго говоря, ***входных данных*** в этой задаче нет! Что ж, порой и такое случается. А что здесь должно быть ***выходным данным***? Неужели тоже нет? Но ***программа*** без ***выходных данных*** – полная бессмыслица. ***Выходным данным*** в этой задаче должно быть изображение, нарисованное ***программой*** на экране.

Поскольку ***входных данных*** здесь нет, то не приходится говорить о ***тестах***, тем более, их составлять. Как же мы будем судить о правильности ***программы***? А по рисунку, который появится на экране. Если он будет удовлетворять условию задачи, то правильно, иначе – нет. Сразу заметим, что отсутствие ***входных данных*** – это очень плохая характеристика ***задачи***. Строго говоря, это и не ***задача***. Однако на практике такие ***задачи*** встречаются, и нередко, поэтому разберём её.

Входные данные: нет

Результат: графический рисунок

Алгоритм:

|  |
| --- |
| 1. Начало 2. Нарисовать стену 3. Нарисовать крышу 4. Нарисовать окно 5. Нарисовать чердачное окно 6. Конец |

Вот, примерно, какая получится ***программа***:

|  |  |
| --- | --- |
| REM Иванов Иван 10А Дом  SCREEN 12  LINE (300, 300) – (400, 300)  LINE (300, 300) – (300, 200)  LINE (300, 200) – (400, 200)  LINE (400, 200) – (400, 300)    LINE (300, 200) – (350, 100)  LINE (400, 200) – (350, 100)  LINE (325, 275) – (375, 275)  LINE (325, 275) – (325, 225)  LINE (325, 225) – (375, 225)  LINE (375, 225) – (375, 275)  CIRCLE (350, 150), 10  END | Пустые строки здесь оставлены для того, чтобы разграничить части ***программы***, где рисуется сам дом, крыша и окна.  В них можно вставить комментарий, где указать какую часть мы рисуем  Проанализируйте эту ***программу*** и определите назначение каждой ***команды***.  Поскольку цвет тут не указан, то по умолчанию он будет 15-й, то есть белый. |
| Program Ivanov\_Ivan\_10a\_dom;  Uses GraphABC;  begin  LINE (300, 300,400, 300);  LINE (300, 300,300, 200);  LINE (300, 200,400, 200);  LINE (400, 200,400, 300);    LINE (300, 200,350, 100);  LINE (400, 200,350, 100);  LINE (325, 275,375, 275);  LINE (325, 275,325, 225);  LINE (325, 225,375, 225);  LINE (375, 225,375, 275);  CIRCLE (350, 150, 10);  end. | Пустые строки здесь оставлены для того, чтобы разграничить части ***программы***, где рисуется сам дом, крыша и окна.  В них можно вставить комментарий, где указать какую часть мы рисуем  Проанализируйте эту ***программу*** и определите назначение каждой ***команды***.  Поскольку цвет тут не указан, то по умолчанию он будет 15-й, то есть белый. |

**Задача VIII-2**

|  |
| --- |
| Сделать ***программу***, которая нарисует на экране домик, у которого высота и ширина равны, одно квадратное окно, расположенное симметрично и круглое окошко на чердаке. Домик должен иметь заданный размер, его нижний левый угол должен находиться в заданном месте. Цвет тоже задаётся |

Эта ***задача***, как следует из условия, уже требует ***входных данных***. Их четыре (все числа): расстояние от левого края экрана до нижнего левого угла дома, расстояние от верхнего края экрана до этого же угла, размер дома и цвет. Имена переменных для них сами просятся: **x**, **y**, **a**, **c,r**.

***Выходным данным***, как и в предыдущей задаче, будет изображение на экране.

Входные данные: x, y, a, c, r – цел., координаты левого нижнего угла, размеры и цвет домика, радиус чердачного окна

Результат: графический рисунок

Алгоритм:

|  |
| --- |
| 1. Начало 2. Описание переменных 3. Ввод аргументов 4. Нарисовать стену 5. Нарисовать крышу 6. Нарисовать окно 7. Нарисовать чердачное окно 8. Конец |

Исходя из вышесказанного, не имеет особого смысла составлять ***тесты*** на бумаге, хотя и можно было бы.

Напишем ***программу***:

|  |  |
| --- | --- |
| REM Иванов Иван 10А Дом по формулам  SCREEN 12  INPUT "Введите координаты х и у, размер, цвет, радиус окна ", x, y, a, c, r  LINE (x, y) – (x + a, y), c  LINE (x, y) – (x, y - a), c  LINE (x, y - a) – (x + a, y - a), c  LINE (x + a, y) – (x + a, y - a), c    LINE (x, y - a) – (x + a / 2, y – 2 \* a), c  LINE (x + a, y - a) – (x + a / 2, y – 2 \* a), c  LINE (x + a / 4, y - a / 4) – (x + 3 / 4 \* a, y - a / 4), c  LINE (x + a / 4, y - a / 4) – (x + a / 4, y - 3 / 4 \* a), c  LINE (x + a / 4, y - 3 / 4 \* a) – (x + 3 / 4 \* a, y - 3 / 4 \* a), c  LINE (x + 3 / 4 \* a, y - a / 4) – (x + 3 / 4 \* a, y - 3 / 4 \* a), c  CIRCLE (x + a / 2, y – 1.5 \* a), r, c  END | Program Ivanov\_Ivan\_10a\_dom\_po\_formulam;  Uses GraphABC;  begin  WRITEln "Введите координаты х и у, размер, цвет, радиус окна через пробел";  READLN(x, y, a, c, r);  SetPenColor(c);  LINE (x, y, x + a, y);  LINE (x, y, x, y - a);  LINE (x, y – a, x + a, y - a);  LINE (x + a, y, x + a, y - a);    LINE (x, y – a, x + a / 2, y – 2 \* a);  LINE (x + a, y – a, x + a / 2, y – 2 \* a);  LINE (x + a / 4, y - a / 4, x + 3 / 4 \* a, y - a / 4);  LINE (x + a / 4, y - a / 4, x + a / 4, y - 3 / 4 \* a);  LINE (x + a / 4, y - 3 / 4 \* a, x + 3 / 4 \* a, y - 3 / 4 \* a);  LINE (x + 3 / 4 \* a, y - a / 4, x + 3 / 4 \* a, y - 3 / 4 \* a);  CIRCLE (x + a / 2, y – 1.5 \* a, r)  END |

Разберитесь с этой ***программой***, добейтесь понимания каждой ***команды***.

Запуская ***программу***, задавая каждый раз разные ***входные данные,*** надо наблюдать за тем, что появляется на экране. При решении таких ***задач*** у части обучающихся, которые ленятся сами придумывать формулы (а это надо делать самостоятельно) возникает мысль словчить, сдать ***программу***, сделанную методом подгонки, по одному ***тесту*** и получить незаслуженную оценку. Что на это можно сказать. Как писал Козьма Прутков: ***«Что есть хитрость? - Хитрость есть оружие слабого и ум слепого.»*** Могу, впрочем, дать такой совет: формулу можно «подогнать», проведя серию экспериментов. Главное, тщательно тестировать ***программу***.

Не удержусь и от ещё одной цитаты того же автора: ***«Бросая в воду камешки, смотри на круги, ими образуемые, иначе такое бросание будет пустою забавою.»***

***Вопросы для самоконтроля:***

* 1. Что такое ***дискретность*** и ***непрерывность***, ***конечность*** и ***бесконечность***?
  2. Почему на компьютере нельзя сделать непрерывную и бесконечную плоскость?
  3. Зачем нужна ***команда*** **SCREEN 12**?
  4. От какой ***программы***, создающей графическое изображение на экране, больше толку, от той, в которой используются ***переменные***, или без них. Какая из этих программ может заменить другую?
  5. Почему на экране видны неровности линий и окружностей?

1. ***Линейные конструкции и алгоритмические структуры***

Мы уже рассмотрели несколько ***простых команд***, то есть таких, выполняя которые Бейсик должен совершить какое-то действие. Например, присвоить ***значение переменной***, вывести на экран информацию, нарисовать окружность… Это, безусловно, очень нужные ***команды***, без них никак не обойтись.

**Линейные конструкции** – это такие участки **программ**, которые включают в себя лишь **простые команды,** в котором все действия выполняются последовательно, без каких либо условий. После выполнения очередной **команды** не возникает вопроса о том, какая **команда** следующая. Каждая **команда** в **линейной конструкции** выполняется один раз, не больше и не меньше. Разумеется, **линейная конструкция** это самое простое, что есть в **программировании**. Иначе такая конструкция называется **Линейным алгоритмом.** Далее приведены блоки линейной конструкции и операторы, которые этим блокам соответствуют.

|  |  |  |
| --- | --- | --- |
| Блок-схема | Бейсик | Паскаль |
| начало  описание  величин  ввод  данных  блок действия    вывод  результатов  конец | REM Название программы  DIM <список величин> AS <описание величин>  INPUT [«Подсказка»;] <список переменных, разделитель запятая>  LET <имя величины> = <значение величины,  арифметического или логического выражения>  PRINT <список выводимых величин ,разделитель запятая или точка с запятой>  END | program Название программы(латинскими буквами);  var  <список величин>: <описание величин>  read[ln](<список переменных, разделитель запятая>);  <имя величины> := <значение величины, арифметического или логического выражения>;  write[ln](<список выводимых величин, разделитель запятая>);  END. |

Однако при решении многих **задач**, приходится писать такие **программы**, в которых порядок выполнения **команд** не всегда совпадает с порядком их написания. Какие-то группы **команд** могут вообще пропускаться в зависимости от выполнения каких-либо **условий**, другие повторяться по несколько раз, тоже в зависимости от выполнения соответствующих **условий**. С помощью специальных **команд** можно запускать группы **команд**, находящихся совсем в другом месте **программы**.

Именно такие участки **программ**, где нарушается естественный порядок выполнения **команд**, имеющие чётко выраженные начало и конец, и есть **алгоритмические структуры**.

**Алгоритмических структур** всего три: **ветвление**, **цикл** и **вспомогательный алгоритм**. Используя их, и **линейные конструкции**, можно решить любую **задачу**.

В нашем центре выработаны правила структурного оформления **программ**, которые призваны облегчить работу учащимся и сделать **программы** понятными. В Бейсике и Паскале каждая **структура** имеет **команду** **начала** и **команду конца**, а **структура ветвления** (полного) и **команду прерывания** (**ELSE**). Вот эти правила:

|  |
| --- |
| 1. Сама **программа** оформляется как **структура.** 2. **Команды начала** и **конца структуры** (а в случае полного **ветвления** и прерывания) пишутся на одном уровне. **Тело структуры** (т.е. те **команды**, которые находятся между ними), сдвигается вправо на две-три позиции[[9]](#footnote-9). 3. Внутри любой **алгоритмической структуры** может находиться любое количество **алгоритмических структур**. Единственным ограничителем здесь является **принцип полной вложенности**, если одна **структура** началась внутри другой, то и закончиться она должна внутри неё. 4. Вход в **структуру** и выход из неё допустимы только через **команды** ее **начала** и **конца!!!** На крайний случай есть группа **команд аварийного выхода**, по которым происходит закрытие **структур**ы и переход на **команду**, следующую за командой конца **структуры**. |

***Вопросы для самоконтроля:***

1. Зачем нужны ***алгоритмические структуры***?
2. Чем характерна ***линейная конструкция***?
3. Зачем структурно оформлять ***программу***?

## Простые и сложные условия

При изучении ***алгоритмических структур*** нам понадобится понятие ***условия***. Разберёмся с ним.

Будем называть простым ***условием*** два однотипных ***выражения***, между которыми стоит знак отношения: «**=**»(равно),«**<**»(меньше),«**>**»(больше),«**<>**»(не равно), «**=>**»(больше или равно),«**<=**»(меньше или равно). Очевидно, что каждое ***условие*** может либо выполняться, либо не выполняться, то есть находиться в одном из двух состояний, которые называются «ИСТИНА» и «ЛОЖЬ». Третьего не дано.

Примеры простых ***условий***:

**a = 2**

**(x - x0) ^ 2 + (y – y0)^2 <= r ^ 2**

**a$ + b$ = "12345"**

Однако при решении многих ***задач*** одних только простых у***словий*** бывает недостаточно. Возникает необходимость в сложных ***условиях***, логических выражениях с использованием специальных ***функций*** или операций, имеющих в качестве аргументов одно, или несколько простых ***условий***. С тремя простейшими логическими ***функциями*** мы сейчас ознакомимся.

Поскольку сейчас нас не будет интересовать конкретное содержание ***условий***, то будем обозначать их заглавными латинскими буквами.

1. **Функция «НЕ» (NOT)**

Эта ***функция*** «одноместная», то есть имеет один аргумент. Её таблица истинности приводится ниже:

|  |  |
| --- | --- |
| A | НЕ (A) |
| ИСТИНА | ЛОЖЬ |
| ЛОЖЬ | ИСТИНА |

На Бейсике и Паскале эта ***функция*** обозначается **NOT** (например, **NOT** a > b) В математической логике принято такое обозначение: **⎤** (например, **⎤**a). Эта ***функция*** самая простая, она ещё называется «отрицание». Её принцип «делай наоборот».

1. **Функция «И» (AND)**

А эта ***функция*** – двуместная, аргумента у неё два (а может быть и больше). Приведём таблицу истинности:

|  |  |  |
| --- | --- | --- |
| A | B | И (A, B) |
| ИСТИНА | ИСТИНА | ИСТИНА |
| ИСТИНА | ЛОЖЬ | ЛОЖЬ |
| ЛОЖЬ | ИСТИНА | ЛОЖЬ |
| ЛОЖЬ | ЛОЖЬ | ЛОЖЬ |

В этой таблице перебраны все варианты значений аргументов. Вообще, тут есть закономерность: эта ***функция*** истинна лишь тогда, когда все аргументы истинны. Достаточно того, чтобы хоть один аргумент имел значение ЛОЖЬ, и значение функции тоже будет ЛОЖЬ.

В математической логике эта ***функция*** чаще рассматривается как операция и называется конъюнкция, или логическое умножение. Обозначается она там с помощью знаков **&** или **Λ**.

На Бейсике и Паскале это тоже скорей операция и записывается так:

|  |
| --- |
| a = 3 AND b > 7  x <= x2 AND x >= x1 |
| (a = 3) AND (b > 7)  (x <= x2) AND (x >= x1) |

1. **Функция «ИЛИ» (OR)**

Эта ***функция*** тоже двуместная. Вот её таблица истинности:

|  |  |  |
| --- | --- | --- |
| A | B | ИЛИ (A, B) |
| ИСТИНА | ИСТИНА | ИСТИНА |
| ИСТИНА | ЛОЖЬ | ИСТИНА |
| ЛОЖЬ | ИСТИНА | ИСТИНА |
| ЛОЖЬ | ЛОЖЬ | ЛОЖЬ |

Закономерность такая: эта ***функция*** ложна лишь тогда, когда все аргументы ложны. Достаточно того, чтобы хоть один аргумент имел значение **ИСТИНА**, и значение ***функции*** тоже будет **ИСТИНА**.

В математической логике этой функции соответствует операция, называемая дизъюнкцией, или логическим сложением (обозначается **V**).

А на Бейсике и Паскалеэто выглядит так:

|  |
| --- |
| a = 3 OR b > 7  x <= x2 OR x >= x1 |
| (a = 3)OR (b > 7)  (x <= x2) OR (x >= x1) |

## Ветвление

***Ветвление*** служит для того, чтобы можно было пропустить, не выполнять какую-то группу ***команд*** в ***программе***. Вопрос о выполнении, или невыполнении решается в зависимости от выполнения, или невыполнения соответствующего ***условия***.

|  |  |
| --- | --- |
| Полное ветвление (полная условная конструкция)  Бейсик | |
| ***Ложь***  **2-е**  **действие**  ***Истина***  **1-е**  **действие** Условие | **IF** *<условие>* **THEN**  *<1-е действие>*  **ELSE**  *<2-е действие>*  **END IF**  Паскаль  **IF** *<условие>* **THEN**  **begin**  *<1-е действие>*  **end**  **ELSE**  **begin**  *<2-е действие>*  **end** |
|  | |
| Если ***условие*** истинно, то выполняется ***первое*** ***действие***, иначе – ***второе***. Таким образом, всегда будет выполняться либо ***первое действие***, либо ***второе***. Всегда одно, не больше и не меньше!  Если действия состоят всего из одной команды, то на Бейсике и Паскале, мы можем записать:  **IF** *<условие>* **THEN** *<1-я команда>* **ELSE** *<2-я команда>* | |
| Сокращённое ветвление (неполная условная конструкция)  Бейсик | |
| **действие**  ***Ложь***  ***Истина*** Условие | **IF** *<условие>* **THEN**  *<действие>*  **END IF**  Паскаль  **IF** *<условие>* **THEN**  **begin**  *<1-е действие>*  **end** |
| Если ***условие*** истинно, то выполняется ***действие***, иначе – не делается ничего.  Если действия состоят всего из одной команды, мы можем записать: **IF** *<условие>* **THEN** *<команда>* | |

**IF** по-английски значит «***если***», **THEN –** «***то***», **ELSE** – «***иначе***», **END – «конец»,** а **END IF** – «***конец если***»

***Действия*** могут состоять из произвольного числа ***команд***. Они могут включать в себя ***алгоритмические структуры***. Могут они быть и пустыми.

В принципе, полное и сокращённое ***ветвления*** взаимозаменяемы. Если решаемая подзадача распадается ровно на два альтернативных случая, то удобно воспользоваться полным ***ветвлением***. В других случаях больше подходит сокращённое. Впрочем, всё не так просто.

**Задача XI-1**

|  |
| --- |
| Сделать ***программу***, высчитывающую площадь треугольника по формуле Герона по трём заданным сторонам. ***Входные данные*** – три числа. Если они могут образовать треугольник, то выдать его площадь, иначе выдать сообщение о некорректности ***входных данных***. |

Как и положено, начнём с вопроса о ***входных*** и ***выходных данных***. Как сказано в условии, ***входными данными*** служат три числа, которые служат длинами сторон треугольника. Что касается ***выходных***, то они зависят от того, существует ли треугольник с такими сторонами. Если да, то выходным данным является число – площадь треугольника, если нет, то сообщение о некорректности ***входных данных***.

**Входные данные:** a, b, c – цел., стороны треугольника

**Выходные данные:** s – вещ., площадь треугольника или сообщение, что треугольника не существует

**Вспомогательная величина:** p – вещ., полупериметр

Делаем два ***теста***.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № теста | ***Входные данные*** | | | ***Выходные данные*** |
| **1** | **3** | **4** | **5** | **6** |
| **2** | **1** | **1** | **10** | **некорректные данные** |

Алгоритм:

|  |
| --- |
| 1. Начало 2. Определение переменных 3. Ввод данных 4. Если a+b>c и a+c>b и b+c>a   4.1 то  4.1.1 p=  4.1.2 s=  4.1.3 вывод s  4.2 иначе  4.2.1 вывод сообщения «Треугольника с такими сторонами не существует»  4.3 конец условия   1. конец |

Напишем ***программу***:

|  |  |
| --- | --- |
| REM Иванов Иван 10А Формула Герона  DIM a,b,c AS INTEGER  DIM s,p AS SINGLE  INPUT "Введите три стороны треугольника ", a, b, c  IF (a + b >= c) AND (a + c >= b) AND (b + c >= a) THEN  LET p = (a + b + c) / 2  LET s = SQR((p - a) \* (p - b) \* (p - c) \* p)  PRINT ”s =”; s  ELSE  PRINT "Треугольника с такими сторонами не существует”  END IF  END | program Ivanov\_Ivan\_10a\_Formula\_Gerona;  var  a,b,c:integer;  s,p:real;  begin  writeln(‘ Ведите три стороны треугольника через пробел’);  readln(a,b,c);  IF (a + b >= c) AND (a + c >= b) AND (b + c >= a) THEN  begin  p:=(a+b+c)/2;  s:=sqrt((p-a)\*(p-b)\*(p-c)\*p)  writeln(‘ s= ‘,s)  end  else  writeln(‘ Треугольника с такими сторонами не существует’)  end. |

**Задача XI-2**

|  |
| --- |
| Человек пришёл в магазин с некоторой суммой денег. Он имеет намерение купить некоторое количество сахара. В случае возможности такой покупки, он её совершает, в противном случае – ничего не покупает и уходит. Сумма денег, которая имеется первоначально; цена одного килограмма сахара; количество, которое он хочет купить, задаются.  Сделать ***программу***, определяющую, сколько денег остаётся. |

Приступим к решению. В условии ***задачи*** чётко указаны все ***входные*** и ***выходные данные***, надо лишь ещё раз вдуматься. ***Входные данные***: три числа (количество денег изначально, цена сахара, количество сахара), ***выходные***: одно число, сумма денег, оставшихся после посещения магазина.

**Входные данные**: s – цел., количество денег; c – цел., цена товара;k – цел., количество товара

**Выходные данные**: s – цел., сумма оставшихся денег

Составим несколько ***тестов***.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № теста | ***Входные данные*** | | | ***Выходные данные*** |
| **1** | **30** | **15** | **2** | **0** |
| **2** | **50** | **15** | **2** | **20** |
| **3** | **20** | **15** | **2** | **20** |

Алгоритм:

|  |
| --- |
| 1. Начало 2. Определение переменных 3. Ввод данных 4. Если c\*k ≤ s то s=s-c\*k 5. вывод s 6. конец |

Теперь напишем ***программу***:

|  |
| --- |
| REM Иванов Иван 10А Поход в магазин  DIM s, c, k AS INTEGER  INPUT "Введите количество денег ", s  INPUT "Введите цену товара", c  INPUT "Введите количество товара", k  IF c \* k <= s THEN LET s = s - c \* k  PRINT "Осталось денег "; s  END |
| program Ivanov\_Ivan\_10a\_Poxod\_v\_magazin;  var  s,k,c:integer;  begin  writeln(‘ Введите количество денег’);  readln(s);  writeln(‘ Введите цену товара);  readln(c);  writeln(‘ Введите количество товара’);  readln(k);  if c\*k<=s then s:=s-c\*k;  writeln(‘ Осталось денег ’,s)  end. |

***Вопросы для самоконтроля:***

1. Зачем нужно ***ветвление***?
2. Какое ***ветвление*** лучше, ***полное***, или ***сокращённое***? Можно ли одно заменить другим?
3. Можно ли заменить ***ветвление линейной конструкцией***?
4. Может ли ***ветвление*** включать в себя другое ***ветвление***?
5. Почему ***ветвление*** называется ***ветвлением***?
6. С какого служебного слова начинается ***ветвление***?

## Циклы

Бывает, что при написании *программы* возникает необходимость в том, чтобы написать подряд несколько одинаковых *действий*. В принципе, можно просто взять и написать их одно за другим, если повторять надо не очень много раз, и если точно известно, сколько раз. Однако как быть, если *действие* надо повторить несколько тысяч раз? Как быть, если мы вообще не знаем заранее, сколько раз надо повторять, а это зависит от *входных данных*? Для решения этой проблемы и существует *алгоритмическая структура* ***цикл***, к рассмотрению которой мы и переходим.***Алгоритмы***, в которых серия команд выполняется несколько раз, при выполнении (невыполнении) некоторого условия, называются ***циклическими***.

Каждое повторение циклического алгоритма (цикла) называется – *итерация.*

Как и у ***ветвлений***, у ***циклов*** есть несколько разновидностей.

|  |  |
| --- | --- |
| ***Циклы с предусловием*** | |
| ***Цикл с положительным предусловием*** | ***Цикл с отрицательным предусловием*** |
| ***Ложь***  ***Истина***  **действие** Условие | ***Истина***  ***Ложь***  **действие** Условие |
| ***Бейсик*** | |
| **DO WHILE** *<условие>*  *<действие>*  **LOOP** | **DO UNTIL** *<условие>*  *<действие>*  **LOOP** |
| Паскаль |  |
| **while** *<условие>* **do**  **begin**  *<действие>*  **end;** |  |
| Работа этого ***цикла*** происходит следующим образом: сначала проверяется ***условие*** и, если оно истинно, то выполняется ***действие***. После этого опять проверяется ***условие***. Так происходит до тех пор, пока условие ***истинно***. Когда условие станет л***ожным***, управление будет передано на ***команду***, следующую за ***командами цикла.*** | **UNTIL** – как только.  Разница между циклами с положительным и отрицательным предусловиями такова: первый работает, пока условие истинно, второй - прекращает работу, как только оно станет истинным. Понятно, что заменить один другим – дело техники. |

|  |
| --- |
| **DO** – делать. **WHILE** – пока. **LOOP** –петля.  Такой ***цикл*** может и не выполняться ни одного раза, если с самого начала ***условие*** будет ***ложно***. В случае если ***условие*** всегда будет истинно, ***цикл*** не прекратит работу никогда, ***программа*** «зациклится»[[10]](#footnote-10) |

|  |  |
| --- | --- |
| ***Циклы с постусловием*** | |
| ***Цикл с положительным***  ***постусловием*** | ***Цикл c отрицательным***  ***постусловием*** |
| ***Бейсик*** | |
| **DO**  *<действие>*  **LOOP WHILE** *<условие>* | **DO**  *<действие>*  **LOOP UNTIL** *<условие>* |
|  | Паскаль |
|  | **repeat**  *<действие>*  **UNTIL** *<условие>* |
| ***Ложь***  ***Истина*** Условие **действие** | ***Ложь***  ***Истина*** Условие **действие** |
| Всё это очень похоже на ***цикл с предусловием***, отличие лишь в том, что самый первый раз действие выполняется независимо от ***условия***. Истинность, или ложность ***условия***, влияет на выполнение действия лишь со второго раза. Между ***циклами*** с положительным и отрицательным постусловиями разница точно такая же, как и в случае с предусловиями. | |
| ***Цикл с параметром*** | |
| ***Бейсик***  **FOR** *упц* **=** *нз* **TO** *кз*[**STEP** (*шаг*)]  *действие*  **NEXT** *упц*  Паскаль  **for** *упц* **=** *нз* **to** *кз* **do** (*шаг*1)или **for** *упц* **=** *нз* **downto** *кз* **do** (*шаг минус 1)*  **begin**  *<действие>*  **end;** | |
| *упц = нз, кз, шаг*    *действие* | |
| Цикл с известным количеством повторений называют ***циклом с параметром или цикл пока.***  Для организации этого вида ***цикла*** вводится специальная ***вспомогательная***[[11]](#footnote-11) ***переменная*** – ***управляющая*** ***переменая цикла*** (упц), ***упц*** получает ***начальное значение***. Потом проверяется ***условие*** *упц<=кз* (конечное значение), в случае, если *шаг>0*, и *упц>=кз*, если *шаг<0*. Если это ***условие истинно***, то выполняется *действие*, ***параметр цикла*** изменяется на *шаг*, и опять проверяется ***условие***. Таким образом, это частный случай ***цикла с предусловием***, так как если ***условие*** в начале ***ложно***, то *действие* вообще не выполняется.  ***Начальное значение***, ***конечное значение***, ***шаг управляющей*** ***переменой цикла*** – арифметические выражения. Если ***цикл*** работал хоть раз, то по окончании значение ***управляющей*** ***переменой цикла*** равно *конечному значению + шаг*. | |

**Задача XII-1**

|  |
| --- |
| Даны два целых числа, причём первое не больше второго. Выдать на экран все целые числа от первого до второго. |

Начнём решать. Какие ***входные*** и ***выходные данные***? ***Входные данные***: два числа, ***выходные***: последовательность чисел.

**Входные данные:**n1, n2 – цел., числа в порядке возрастания

**Выходные данные:** последовательность чисел от n1 до n2

**Вспомогательная величина:** i – цел., *у.п.ц.*

Составим несколько ***тестов***.

|  |  |  |  |
| --- | --- | --- | --- |
| № теcта | *Входные данные* | | *Выходные данные* |
| 1 | 1 | 10 | 1 2 3 4 5 6 7 8 9 10 |
| 2 | 2 | 2 | 2 |
| 3 | 37 | 52 | 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 |

Алгоритм:

|  |
| --- |
| Цикл с предусловием   1. Начало 2. Определение переменных 3. Ввод данных 4. Присвоение начальных условий 5. Делать пока i ≤ n2   5.1 Вывод i  5.2 i=i+1  5.3петля   1. конец |
| Цикл пока   1. Начало 2. Определение переменных 3. Ввод данных 4. Для i от n1 до n2   4.1 Вывод i  4.2 Конец цикла   1. конец |

***Программа:***

|  |  |
| --- | --- |
| REM Иванов Иван 10А числовая последовательность  DIM n1, n2, i AS INTEGER  INPUT "Введите два числа в порядке возрастания ", n1, n2  LET i = n1  DO WHILE i <= n2  PRINT i;  LET i = i + 1  LOOP  END | program Ivanov\_Ivan\_10a\_Chislovaya\_  posledovatelnost;  var  n1,n2:integer;  begin  writeln(‘ Введите два числа в порядке возрастания через пробел‘);  readln(n1,n2);  while i <= n2 do  begin  wrieter(‘ ‘,i);  i:=i+1  end  end. |

Но можно и так:

|  |  |
| --- | --- |
| REM Иванов Иван 10А числовая последовательность  DIM n1, n2, i AS INTEGER  INPUT "Введите два числа в порядке возрастания ", n1, n2  FOR i = n1 TO n2 STEP 1  PRINT i;  NEXT i  END | program Ivanov\_Ivan\_10a\_Chislovaya\_posledovatelnost;  var  n1,n2:integer;  begin  writeln(‘ Введите два числа в порядке возрастания через пробел‘);  readln(n1,n2);  for i:=n1 no n2 do  wrieter(‘ ‘,i)  end. |

**Задача XII-2**

|  |
| --- |
| Даны два натуральных числа, сделать ***программу*** для нахождения их наибольшего общего делителя. |

Приступим к решению. Определим ***входные*** и ***выходные данные***? ***Входные данные***: два числа, ***выходные***: одно число, их НОД.

**Входные данные:** a, b – цел., два числа

**Выходные данные:** НОД

**Вспомогательные величины:** a1, b1 – цел., два числа над которыми производим действия

Составим несколько ***тестов***.

|  |  |  |  |
| --- | --- | --- | --- |
| № теcта | *Входные данные* | | *Выходные данные* |
| 1 | 1 | 10 | 1 |
| 2 | 2 | 8 | 2 |
| 3 | 37 | 52 | 1 |

Для составления ***алгоритма*** воспользуемся известным из математики алгоритмом Евклида.

Алгоритм:

|  |
| --- |
| 1. Начало 2. Определение переменных 3. Ввод данных 4. Присвоение начальных условий 5. Делать пока a1 ≠ b1    1. если a1 > b1 то a1=a1-b1 иначе b1=b1-a1 6. петля 7. вывод НОД 8. конец |

Программа:

|  |  |
| --- | --- |
| ’Иванов Иван 10А алгоритм Евклида  CLS  DIM a,b,a1,b1 AS INTEGER  INPUT "Введите два числа ", a, b  LET a1 = a  LET b1 = b  DO WHILE a1 <> b1  IF a1 > b1 THEN a1 = a1 – b1 ELSE LET b1 = b1 – a1  LOOP  PRINT "НОД двух чисел "a; “ и “;b; “равен “; a1  END | Считаем необходимым объяснить, зачем введены ***вспомогательные переменные*** **a1** и **b1**. Если это явно не сказано в условии ***задачи***, не вытекает из её логики, то переменные, являющиеся ***входными данными***, лучше «не портить», то есть не менять их значений. Если это не войдёт в привычку, то в дальнейшем возникнет много проблем. |
| program Ivanov\_Ivan\_10a\_Algoritm\_Evklida;  var  a,b,a1,b1:integer;  begin  writeln(‘ Введите два числачерез пробел‘);  readln(a,b);  a1:=a;  b1:=b;  while a1 <> b1 do  if a1 > b1 then a1 = a1 – b1 else b1 = b1 – a1  wrieter(‘ НОД двух чисел ‘,a, ‘ и ‘,b’ равен ‘,a1)  end. |

***Вопросы для самоконтроля:***

* 1. Зачем нужны ***циклы***?
  2. Какие виды ***циклов*** вы знаете? Полностью ли они взаимозаменяемы?
  3. Можно ли ***ветвление*** заменить ***циклом***?
  4. А наоборот?
  5. Что лучше, ***ветвление***, или ***цикл***?
  6. Какой ***цикл*** может не отработать ни одного раза?
  7. Может ли ***цикл*** никогда не закончиться?

## Вспомогательные алгоритмы

Представим себе, что при написании ***программы*** вы столкнулись с такой ситуацией. Вам необходимо довольно сложное ***действие***, которое в этой ***программе*** уже встречалось, повторить один, или несколько раз. Но повторить не подряд, это можно было бы сделать с помощью ***цикла***, а в разных местах ***программы***. Можно, конечно, не полениться, и написать это ***действие*** каждый раз в отдельности, тем более что переписывать вручную не обязательно, можно и скопировать. Однако, ***программа*** при этом получится очень большой, а надо стремиться к тому, чтобы она «не раздувалась» по пустякам. Есть другой путь: один раз в специальном месте записать всё ***действие***, а потом по мере надобности, это ***действие*** «подгружать» к ***программе***. Именно для этого и существует последняя из ***алгоритмических структур***, ***вспомогательный алгоритм***.

Можно сказать, что ***вспомогательный алгоритм*** это алгоритм решения некоторой задачи, являющейся подчинённой по отношению к основной (исходной) задаче. Чаще всего вспомогательный алгоритм называют ***подпрограммой***

Как и в случае ***ветвления*** и ***цикла***, ***вспомогательные алгоритмы*** также бываю разных видов, разберём по порядку.

1. **Подпрограмма**

*a)* ***Бейсик***

***Подпрограмма*** – самый простой вид ***вспомогательных алгоритмов***. Напомню, что написание ***программы*** должно заканчиваться ***командой*** «**END**». Теперь уже можно уточнить, что не всей ***программы***, а лишь главной её части, ***головной программы***. После же команды **END** могут располагаться ***подпрограммы***. ***Подпрограмма*** начинается с ***метки***. ***Метка*** это указатель, куда надо передать управление. Если ***метка*** не является числом, а такие метки я вам давать не советую, то после неё должно быть двоеточие. В конце же ***подпрограммы*** стоит ***команда*** «**RETURN**». Управление на ***метку подпрограммы*** из ***головной программы***, или другой ***подпрограммы*** передаётся с помощью ***команды*** «**GOSUB** *<метка>*». После этого начинает работать ***подпрограмма***, то есть её ***команды***. Но, как только встретится ***команда*** «**RETURN**», произойдёт возврат на ***команду***, следующую за «**GOSUB**».

|  |
| --- |
| **REM** Фамилия Имя, класс, назначение программы  **. . .**  **GOSUB** *<метка1>*  **. . .**  **END**  *<метка1>***:**  *<действие1>*  **RETURN**  *<метка2>***:**  *<действие2>*  **RETURN** |

Считаем необходимым сделать ряд существенных замечаний касательно ***подпрограмм***.

1. Все ***переменные***, используемые в ***подпрограмме***, являются общими с ***головной программой*** и другими её ***подпрограммами***. Об этом необходимо помнить, заводя в ***подпрограмме вспомогательные переменные***; в этом случае необходимо позаботиться о том, чтобы в этот момент эти ***переменные*** были «свободны».
2. Из любой ***подпрограммы***, «принадлежащей» данной ***головной программе***, можно обращаться к любой ***подпрограмме*** этого семейства.
3. ***Метки*** всех ***подпрограмм*** должны быть разными. Лучше всего имена ***меток*** придумывать со смыслом.
4. Если забыть поставить ***команду*** **END** в ***головной программе***, то произойдёт непроизвольный переход на первую ***подпрограмму***, а потом аварийная остановка ***программы***.
5. Если забыть поставить ***команду*** **RETURN** после последней ***подпрограммы***, то произойдет незапланированная остановка ***программы***.
6. Располагать сами ***подпрограммы*** можно в произвольном порядке относительно друг друга.

*b)* ***Паскаль***

В паскале подпрограмма является частью основной программы. Её описание располагается между разделом **var** и программным блоком главной программы. Если подпрограмм несколько, то их описания располагаются в произвольном порядке одно за другим.

Структура описания подпрограммы аналогичны структуре главной программы. Описание подпрограммы начинается с заголовка и заканчивается оператором **end**.

Согласно концепции структурного программирования, ***вспомогательный алгоритм*** должен:

* иметь имя, по которому его можно вызвать из других алгоритмов;
* возвращать управление тому алгоритму, из которого был вызван. После того, как завершится выполнение вспомогательного алгоритма, вызвавший его алгоритм должен продолжить работу с той точки, в которой был прерван;
* иметь возможность вызывать другие алгоритмы;
* иметь достаточно малые размеры.

Операторы ***подпрограммы***, окаймлённые операторными скобками **begin...end,** называются ***телом*** этой подпрограммы.

В Паскале различают два вида подпрограмм: процедуры и функции.

1. **Процедура**

*a)* ***Бейсик***

***Процедура*** – следующий вид ***вспомогательных алгоритмов***. Главным отличием её от ***подпрограммы*** является то, что все ***вспомогательные переменные процедуры*** – локальны, то есть не являются принадлежностью ***головной программы*** и её ***подпрограмм,*** то есть одноимённые переменные в основной программе и процедуре – ***это разные переменные***, существующие и используемые независимо друг от друга.

Хотя у ***процедуры*** могут быть свои собственные ***подпрограммы***, но это её ***подпрограммы***, и «отношения» между ними полностью совпадают с описанными в предыдущем пункте.

***Процедура*** начинается с заголовка:

|  |
| --- |
| **SUB** *<имя процедуры>*[*<***(***список формальных параметров***)***>*] |

Набрав заголовок, мы попадаем в отдельное окно редактирования ***процедуры***. Автоматически появляется ***команда*** «**END SUB**». После этого вводится текст ***процедуры***. Для возврата в ***головную программу*** или перехода к другой ***процедуре*** надо нажать на функциональную клавишу F2. ***Формальные параметры*** – это ***имена переменных***, ***входных*** и ***выходных данных процедуры***. Если они есть, а порой их может и не быть, то записываются в скобках. ***Входные данные*** должны иметь ***значения*** перед началом работы ***процедуры***. ***Выходные*** же должны получить значения в процессе её работы.

Обращаться к ***процедуре*** можно из ***головной программы*** либо из любой ***процедуры***[[12]](#footnote-12). Это делается так:

|  |
| --- |
| **CALL** *<имя процедуры>*[**(***<список фактических параметров>***)**] |

***Фактические параметры*** необходимо подробно разъяснить. ***Входные фактические параметры*** – это выражения соответствующего типа. В частности, это могут быть конкретные значения или ***имена переменных***. Что касается ***выходных фактических параметров***, то это обязательно ***имена переменных***, которые должны получить значения в результате работы ***процедуры***.

В некоторых программах нам необходимо, чтобы величины для основной программы и процедур были общими. Для этого, перед использованием этих величин, мы должны указать, что они общие. При этом используется следующий оператор:

|  |
| --- |
| **DIM SHARED** <*имя переменной>* **AS** <*тип переменной*> |

*b)* *Паскаль*

***Процедура*** – подпрограмма, имеющая произвольное количество входных и выходных данных.

Описание процедуры имеет вид:

**procedure** <имя> (<описание параметров-значений>; **var:** <описание параметров-переменных>);

**begin**

<операторы>

**end;**

В заголовке процедуры после её имени приводится перечень формальных параметров и их типов. Входные параметры, значения которых не изменяются в программе, должны быть параметрами-значениями. Выходные (результирующие) параметры должны быть параметрами-переменными.

Для вызова процедуры достаточно указать её имя со списком фактических параметров. В качестве параметров-значений можно указать имена переменных, константы и выражения.

1. **Функция**

*a)* ***Бейсик***

***Функция*** имеет много общего с ***процедуро***й. Разница состоит в том, что ***функци***я применима лишь тогда, когда должно получиться лишь одно ***выходное данное***. В этом случае ***имя функции*** одновременно является и ***именем выходной переменной***. Это даёт возможность использовать ***функцию*** в правой части ***команды присваивания*** и в ***условиях***. Понятно, что все ***вспомогательные переменные функции*** локальны. К ***функции*** можно обращаться из ***головной программы***, её ***подпрограмм***, ***процедур*** и ***функций***.

Для того чтобы начать запись ***функции***, необходимо набрать

|  |
| --- |
| **FUNCTION** *<имя функции>*[**(***<список формальных параметров>***)**] |

Всё это очень похоже на создание процедуры. Мы попадаем в отдельное окно редактирования ***функции***. Появляется ***команда*** «**END FUNCTION**». После этого надо вводить текст ***функции***. Для возврата в ***головную программу*** или перехода к другой ***процедуре*** или ***функции***, надо нажать на функциональную клавишу F2 выбрать необходимое название и нажать на ок.

*b)* *Паскаль*

***Функция*** – подпрограмма, имеющая единственный результат, записываемый в ячейку памяти, имя которой совпадает с именем функции. Поэтому в блоке функции обязательно должен присутствовать оператор <имя\_функции>:=<результат>.

Для вызова функции достаточно указать её имя со списком фактических параметров в любом выражении, в условиях (после слов ***if, while, until***), или в операторе ***write*** главной программы.

**Задача XIII-2**

|  |
| --- |
| На некотором интервале, определяемом двумя числами, задана некоторая непрерывная функция. На границах интервала она имеет разные знаки. Кроме того известно, что функция монотонна, то есть либо на всём интервале убывает, либо возрастает. Сделать ***программу*** для решения уравнения ***f(x)=0*** с некоторой заданной точностью. |

Для решения этой задачи существует метод деления пополам. Он сводится к тому, что берётся середина интервала, проверяется, не является ли эта точка решением (с учётом точности), и если да, то задача решена, а если нет, то ищем знак функции в этой точке. Ту границу интервала, которая совпадает по знаку с серединой, переносим в середину, и всё начинаем сначала. Поскольку длина интервала будет постоянно уменьшаться и стремиться к нулю, а функция монотонна и непрерывна, то на каком-то шаге найдётся решение.

**Входные данные:** a, b – цел., границы интервала; c – цел., точность

**Выходные данные:** х – вещ., решение уравнения ***f(x)=0*** с некоторой заданной точностью

**Промежуточные величины:** a1, b1 – цел., границы промежуточных интервалов

Алгоритм:

|  |
| --- |
| 1. Начало 2. Описание переменных 3. Ввод аргументов 4. Присвоение начальных условий 5. Цикл делать пока |f(x)| > e    1. Если f(a1)\*f(x) > 0 то a1=x иначе b1=x    2. X=    3. петля |
| 1. Вывод х 2. Конец 3. Нажимаем мышкой на меню: редактирование → новая функция → вводим имя функции → ок 4. В окне функции набираем конкретную функцию 5. Для выхода в основную программу нажимаем функциональную клавишу F2 6. В открывшемся окне щёлкаем мышкой по имени программы. |

Пишем ***программу***:

|  |  |
| --- | --- |
| REM Иванов Иван 10А Решение уравнения методом деления пополам  CLS  DIM a, b AS INTEGER  DIM x,e AS SINGLE  INPUT "Введите границы интервала и точность", a, b, e  LET a1 = a  LET b1 = b  LET x = (a1 + b1) / 2  DO WHILE ABS(f(x))>e  IF f(a1) \* f(x) > 0 THEN LET a1 = x ELSE LET b1 = x  LET x = (a1 + b1) / 2  LOOP  PRINT "x ="; x  END  SUB f(x)  …  END SUB | program Ivanov\_Ivan\_10a\_metod\_deleniya\_popolam;  var {Раздел описания глобальных(относящихся ко всей программе) переменных}  a,b,a1,b1:integer;  x,e:real;  function f:real;  var {Раздел описания локальных(относящихся к данной функции) переменных}  begin  …..  end  begin  writeln(‘ Введите границы интервала и точность через пробел‘);  readln(a,b,e);  a1:=a;  b1:=b;  x:=(a1+b1)/2  while ABS(f) > e do  begin  IF f(a1) \* f(x) > 0 THEN a1: = x ELSE b1: = x;  x: = (a1 + b1) / 2  end;  write(‘ x= ‘,x)  end. |

Так получилось, что мы сразу написали ***программу***, не составив тестов. Это плохо. Вот вам задание: составьте сами несколько тестов и проверьте ***программу***. Разумеется, вам придётся написать конкретную функцию, не забудьте об её необходимых свойствах, непрерывности и монотонности на отрезке.

***Вопросы для самоконтроля:***

* 1. Зачем нужны ***вспомогательные алгоритмы***?
  2. Какие виды ***вспомогательных алгоритмов*** вы знаете? Каковы их особенности?
  3. Можно ли использовать ***циклы*** и ***ветвления*** во ***вспомогательных алгоритмах***?
  4. Можно ли обращаться из одной ***подпрограммы*** к другой? А к самой себе?
  5. Что случится, если забыть написать ***команду*** **END**?
  6. Что случится, если забыть написать ***команду*** **RETURN**?
  7. В каких случаях ***процедура*** лучше ***подпрограммы***?
  8. Можно ли заменить ***процедуру подпрограммой***?
  9. В чём преимущество ***функций***?
  10. Всегда ли можно заменить ***процедуру функцией***?
  11. Что такое ***формальные*** и ***фактические параметры***?
  12. Могут ли у ***процедуры*** совпадать ***входные*** и ***выходные параметры***?
  13. Могут ли у ***вспомогательных алгоритмов*** быть ***вспомогательные переменные***?
  14. Есть ли у ***функции выходные параметры***?
  15. Можно ли обойтись без ***вспомогательных алгоритмов***?

1. ***Массивы***

Вот мы и подошли к одному из самых трудных понятий ***программирования***, а именно, к понятию ***массива***. Что это такое? Это более высокая, чем просто ***переменные***, форма организации данных. Это как бы объединение многих однотипных ***переменных*** под одним ***именем***. А как же они тогда отличаются друг от друга, если у них всех одно ***имя***? По своим номерам или, как говорят программисты, ***индексам***. Количество ***индексов*** может быть произвольным, оно определяет ***размерность массива***. Если просто ***переменную*** можно сравнить с маленьким домиком, где одновременно может находиться лишь один человек, то ***массив*** - большой многоквартирный дом, где каждая квартира имеет свой номер. Впрочем, все сравнения всегда хромают и не могут передать всего существа дела.

Итак, ***массив*** – это объединение многих ***переменных***. (Есть и другие определения массива, например, ***массив в компьютере*** – это последовательность ячеек памяти (ОЗУ) которому дано ***одно имя***.) У них у всех одно ***имя***, но разные ***индексы***. Каждый ***элемент массива*** однозначно определяется своими ***индексами***. ***Индексы*** пишутся в скобках после ***имени массива*** (в Бейсикекруглых, в Паскале квадратных). Если их более одного, то разделителем служит запятая. Как вы уже заметили, в Бейсике и Паскале, разделителем всегда служит запятая. ***Массив*** хорошо представить себе в виде таблицы.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| **3** | **4** | **2** | **4** | **6** | **6** | **8** | **1** | **2** | **3** | **4** | **5** | **5** | **6** | **6** | **6** | **2** | **9** | **0** |

Перед вами одномерный числовой ***массив***, количество элементов – 19. В верхней строке находятся индексы элементов массива, а в нижней – их значения. Очень важно не путать в одномерном числовом ***массиве индексы*** со ***значениями***, хотя на начальном этапе обучения такая опасность есть. Какое значение имеет элемент с ***индексом*** 4? 4. Элемент с ***индексом*** 15? 6.

А вот двумерный символьный ***массив***:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 1 | **ап** | **а** | **м** | **р** | **а** | **ss** | **s** | **ы** | **а** | **п** | **ц** | **qq** | **e** | **м** |
| 2 | **й** | **ы** | **в** | **я** | **я** | **яй** | **р** | **л** | **л** | **л** | **у** | **п** | **кr** | **р** |
| 3 |  | **ц** | **у** | **кк** | **к** | **рр** | **ф** |  | **пп** | **п** | **ы** | **ы** | **м** | **с** |
| 4 | **3** | **3** | **2** | **3** | **в** | **о** | **ф** | **в** | **в** | **в** | **ч** | **щ** | **э** | **х** |
| 5 | **ч** | **чу** | **4** | **с** | **о** | **т** | **ь** | **ю** | **ъ** | **ч** | **и** | **щ** | **х** | **и** |

Вообще, количество ***индексов массива*** может быть произвольным, но наиболее часто употребляются в ***программах массивы*** одномерные, двумерные и трёхмерные. Их преимущество и в том, что человеку, существующему в трёхмерном пространстве, легко себе представить такие ***массивы*** геометрически, в виде таблиц. Таблицы более чем трёхмерные представимы гораздо труднее.

На что советую обратить внимание?

1. В ***массиве*** важно, какое ***значение*** находится на каком месте, которое определяется ***индексами***.
2. Разные ***элементы массива*** могут оказаться равными друг другу, это ничему не противоречит.
3. Работа с ***элементами массивов*** ведётся точно так же, как и с обычными ***переменными***, ***массивы*** иначе и называются ***переменными с индексами***. Присваивание ***значений элементам массива*** происходит либо с помощью ***команды ввода***, либо с помощью ***команды присваивания***. Вывод же ***значений*** – с помощью ***команды вывода***.
4. ***Массивы***, как и просто ***переменные***, могут быть разных ***типов***.

Для того чтобы использовать в ***программе массив***, его необходимо описать специальной ***командой***, которая уже упоминалась:

|  |
| --- |
| *a)* ***Бейсик***  **DIM** *<Имя массива>* **(***<Список ограничителей индексов>***)** [**AS** *<Тип>*]  *b)* *Паскаль*  var  *<Имя массива>*:array [1..N] of *<Тип>,* где N– максимальное количество элементов массива. |

Рассмотрим способы задания элементов массива.

1. Ввод массива с помощью оператора присваивания;
2. Ввод массива с клавиатуры в цикле;
3. Ввод массива с помощью генератора(датчика) случайных чисел в цикле. После ввода массива случайным образом необходимо организовать его вывод на экран.

Желательно ввод и вывод массивов организовать с помощью вспомогательного алгоритма.

Перейдём к разбору ***задач***. Позволим себе все ***задачи*** на ***массивы*** сформулировать на математическом языке. Сначала разберём ряд тривиальных ***задач*** на одномерные числовые ***массивы***. Оформим их в одной ***программе*** так, чтобы каждая ***задача*** будет располагаться в отдельной ***подпрограмме***.

|  |  |  |
| --- | --- | --- |
| Бейсик | Паскаль | Коментарии |
| REM Иванов Иван 10А Массивы  DIM a (100)AS <тип>  CLS  . . .  END | program Ivanov\_Ivan\_10a\_massiv  var  a: array[1..100] of **<тип>**  **begin**  **. . .**  **End.** | Головная ***программа***. |
| vmk:  INPUT "n = "; n  FOR i = 1 TO n STEP 1  INPUT a(i)  NEXT i  GOSUB vime  RETURN | **vmk:**  procedure **vmk:**  var  i,n: integer;  begin  write[ln](‘n=’);  read[ln](n);  for i:=1 to n do  read[ln](a[i]);  vime:  end; | Ввод ***массива*** с клавиатуры.  Улучшить интерфейс. |
| vime:  PRINT "n ="; n  FOR i = 1 TO n STEP 1  PRINT a(i);  NEXT i  PRINT  RETURN | vime:  procedure vime:  var  i,n: integer;  begin  write[ln](‘n=’);  for i:=1 to n do  write[ln](a[i]);  write[ln];  end; | Вывод ***массива*** на экран |
| ssm:  RANDOMIZE TIMER  LET n = 10 + INT(RND\*11)  FOR i = 1 TO n STEP 1  LET a(i) = -10 + INT(RND\*21)  NEXT i  GOSUB vime  RETURN | ssm:  procedure ssm:  var  i,n: integer;  begin  RANDOMIZE;  n:=10+int(random(11));  for i:=1 to n do  a[i]:=- 10 + int(random(21));  vime:  end; | Создание ***массива*** случайным образом. В каких пределах будут находиться значения элементов ***массива***? Что нужно изменить, чтобы все элементы были от 2 до 5? |
| sum:  LET s = 0  FOR i = 1 TO n STEP 1  LET s = s + a(i)  NEXT i  PRINT "s ="; s  RETURN | sum:  procedure sum:  var  i,n: integer;  begin  s:=0;  for i:=1 to n do  s:=s+a[i];  write[ln](‘ s=’; s);  end; | Нахождение суммы всех ***элементов массива***. |
| sa:  GOSUB sum  LET sa = s / n  PRINT "sa ="; sa  RETURN | sa:  procedure sa:  var  n: integer;  begin  sum:  sa:=s/n;  write[ln](‘ sa=’; sa);  end; | Нахождение среднего арифметического всех ***элементов массива*** |
| sumus:  LET su = 0  FOR i = 1 TO n STEP 1  IF a(i) > 0 THEN  LET su = su + a(i)  END IF  NEXT i  PRINT "su ="; su  RETURN | sumus:  procedure sumus:  var  i,n: integer;  begin  s:=0;  for i:=1 to n do  if a[i] > 0 then  su = su + a[i];  write[ln](‘ su=’; su);  end; | Нахождение *суммы* элементов ***массива***, отвечающих заданному ***условию***(например: элемент массива больше нуля). |
| kolus:  LET ku = 0  FOR i = 1 TO n STEP 1  IF a(i) > 0 THEN  LET ku = ku + 1  END IF  NEXT i  PRINT "ku ="; ku  RETURN | kolus:  procedure kolus:  var  i,n: integer;  begin  ku:=0;  for i:=1 to n do  if a[i] > 0 then  ku = ku + 1;  write[ln](‘ ku=’; ku);  end; | Нахождение *количества* ***элементов массива***, отвечающих заданному ***условию***(например: элемент массива больше нуля). Сравните эту ***подпрограмму*** с предыдущей! |
| zamus:  FOR i = 1 TO n STEP 1  IF a(i) < 0 THEN  LET a(i) = 0  END IF  NEXT i  GOSUB vime  RETURN | zamus:  procedure kolus:  var  i,n: integer;  begin  for i:=1 to n do  if a[i] < 0 then  a[i] = 0;  vime;  end; | Замена ***элементов массива***, отвечающих заданному ***условию***(например: элемент массива меньше нуля). |
| maxel:  LET max = a(1)  FOR i = 2 TO n STEP 1  IF a(i) > max THEN  LET max = a(i)  END IF  NEXT i  PRINT "max ="; max  RETURN | maxel:  procedure maxel:  var  i,n: integer;  begin  max = a[1];  for i:=2 to n do  if a[i] > max then  max:=a[i];  write[ln](‘ max =’; max);  end; | Нахождение значения наибольшего ***элемента массива***. Самим придумать наименьшего. Не ***значения***, а ***индекса***. |

Придумайте содержательные ***задачи*** на каждую из вышеприведённых ***программ***.

Самостоятельно разберите вопрос о ***входных*** и ***выходных данных*** к каждой из этих ***задач***, составьте несколько ***тестов***.

Разберём теперь ряд ***программ*** на формирование числовых одномерных ***массивов*** по заданной закономерности.

|  |  |  |
| --- | --- | --- |
| REM Иванов Иван 10А Массивы  DIM a (100) AS INTEGER  DIM i, n AS INTEGER  CLS  INPUT "n =", n  FOR i = 1 TO n STEP 1  LET a(i) = i  NEXT i  GOSUB vime  END | Program massiv;  {Иванов Иван 10А**}**  a: array[1..100] of integer;  var  i,n: integer;  begin  write[ln](‘n=’);  read[ln](n);  for i:=1 to n do  a[i]:=i;  vime  end. | Создание ***массива***, каждый ***элемент*** которого равен своему ***индексу,*** если есть ***подпрограмма*** вывода. |
| REM Иванов Иван 10А Массивы  DIM a (100) AS INTEGER  DIM i, n AS INTEGER  CLS  INPUT "n =", n  FOR i = 1 TO n STEP 1  LET a(i) = 2 \* i  NEXT i  GOSUB vime  END | Program massiv;  {Иванов Иван 10А**}**  a: array[1..100] of integer;  var  i,n: integer;  begin  write[ln](‘n=’);  read[ln](n);  for i:=1 to n do  a[i]:=2\*i;  vime  end. | Создание ***массива***, каждый ***элемент*** которого равен своему удвоенному ***индекс***у***,*** если есть ***подпрограмма*** вывода. |
| REM Иванов Иван 10А Массивы  DIM a (100) AS INTEGER  DIM i, n AS INTEGER  CLS  INPUT "n =", n  FOR i = 1 TO n STEP 1  LET a(i) = i ^ 2  NEXT i  GOSUB vime  END | Program massiv;  {Иванов Иван 10А**}**  a: array[1..100] of integer;  var  i,n: integer;  begin  write[ln](‘n=’);  read[ln](n);  for i:=1 to n do  a[i]:=SQR(i);  vime  end. | Создание ***массива***, каждый ***элемен***т которого равен квадрату своего ***индекса,*** если есть ***подпрограмма*** вывода. |

|  |  |  |
| --- | --- | --- |
| sum2m:  LET s = 0  FOR i = 1 TO n STEP 1  FOR j = 1 TO m STEP 1  LET s = s + a(i, j)  NEXT j  NEXT i  PRINT "s ="; s  RETURN | Sum2m:  procedure sum2m:  var  i, j, n: integer;  begin  s:=0;  for i:=1 to n do  for j:=1 to n do  s = s + a[i];  write[ln](‘ s=’; s);  end; | Нахождение суммы всех элементов двумерного числового ***массива***.  Описание ***массива*** должно быть в головной ***программе***. |

Если вы не пожалеете времени на то, чтобы вникнуть в приведённые здесь тривиальные ***программы***, то, надеемся, поймёте что и как. Тогда вам окажутся по силам и более сложные ***задачи***. Попробуйте переделать все ***программки*** с одномерных ***массивов*** на двумерные.

Вы должны были обратить внимание, что ни одна из вышеприведённых ***программ*** не обошлась без ***циклов***, притом везде использовался лишь ***цикл*** ***с параметром***. Этот вид ***циклов*** действительно больше других подходит для решения простых ***задач*** на ***массивы***, а эти ***задачи*** – простейшие. Для решения ***задач*** посложней необходимо использовать и другие виды ***циклов***.

***Вопросы для самоконтроля:***

1. Зачем нужны ***массивы***? Приведите примеры ***задач***, при решении которых без ***массивов*** не обойтись.
2. Можно ли без них обойтись?
3. Что такое ***размерность массива***?
4. Правильно ли утверждение, что ***размерность массива*** равна количеству его ***элементов***?
5. Обязательно ли ***элемент массива*** равен своему ***индексу***?
6. Могут ли разные ***элементы*** одного ***массива*** быть равны друг другу?
7. Обязательно ли ***элементы массива*** должны находиться в порядке возрастания?
8. Как вы думаете, может ли попасться ***задача*** на ***массивы***, где бы можно было бы обойтись без ***циклов***?
9. ***Сортировка массивов***

Под сортировкой или упорядочением массива понимают перераспределение значений его элементов в некотором порядке.

Порядок, при котором в массиве первый элемент минимальный, а значение каждого последующего элемента не меньше значения предыдущего элемента, называют возрастающим.

Порядок, при котором в массиве первый элемент максимальный, а значение каждого следующего элемента не больше значения предыдущего элемента, называют убывающим.

1. **Метод выбора**

Сортировка выбором (например, по убыванию) осуществляется следующим образом:

1. В массиве выбирается максимальный элемент;
2. Максимальный и первый элементы меняются местами (первый элемент считается отсортированным);
3. В неотсортированной части массива снова выбирается максимальный элемент; он меняется местами с первым неотсортированным элементом массива;
4. Действия, описанные в пункте с, повторяются с неотсортированными элементами массива до тех пор, пока не останется один неотсортированный элемент (его значение будет минимальным).

Работа алгоритма состоит из последовательных шагов. На каждом шаге наибольший элемент области поиска перемещается в уже отсортированную часть массива, за счёт этого отсортированная часть массива растёт, а неупорядоченная сокращается на один элемент. Для сортировки массива из N элементов потребуется N -1 шагов алгоритма.

Время выполнения одного шага прямо пропорционально размеру неупорядоченной части массива. Размер неупорядоченной части массива равно N в начале работы и 2 – в конце. Общее время сортировки выбором: **t=k\*N+k\*(N-1)+k\*(N-2)+…+k\*4+k\*3+k\*2**, где k коэффициент пропорциональности, не зависящий от N. По формуле суммы арифметической прогрессии получим: **t=\*(N-1)=(N+2)\*(N-1)=(N2+N-2) => t~ N2** (отсюда следует, что время прямо пропорционально квадрату количества элементов массива). Зависимость типа t=f(N) , где N – объём входных данных, называется временной сложностью алгоритма. Временная сложность сортировки выбором квадратичная, так как время сортировки пропорционально квадрату числа сортируемых элементов.

**Отсортировать массив методом выбора**

**Арг.**

N – цел., кол-во элементов массива

A(N) – цел., элементы массива

**Рез.**

A(N) – цел., элементы отсортированного массива

**Пром. вел.**

i, j – цел., у. п. ц.

imax – цел., номер максимального элемента

С– цел., элемент сортируемого массива

***Тест***

N = 6

Исходный массив: 18, 3, 11, 1, 6, 16

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i/j | 1 | | 2 | | 3 | | 4 | | 5 | |
|  |  | imax=1 |  | imax=2 |  | imax=3 |  | imax=4 |  | imax=5 |
|  | 2 | imax=1 | 3 | imax=3 | 4 | imax=3 | 5 | imax=5 | 6 | imax=6 |
|  | 3 | imax=1 | 4 | imax=3 | 5 | imax=3 | 6 | imax=5 |  |  |
|  | 4 | imax=1 | 5 | imax=3 | 6 | imax=3 |  |  |  |  |
|  | 5 | imax=1 | 6 | imax=6 |  |  |  |  |  |  |
|  | 6 | imax=1 |  | |  | |  | |  | |
|  | 18, 3, 11, 1, 6, 16 | | 18, 16, 11, 1, 6, 3 | | 18, 16, 11, 1, 6, 3 | | 18, 16, 11, 6, 1, 3 | | 18, 16, 11, 6, 3, 1 | |

***Алгоритм***

1. Начало
2. Определение величин
3. Ввод n
4. [Определение массива A(n) – для бейсика]
5. Цикл ввода массива
6. Цикл вывода массива на экран
7. Цикл i=1,n-1
   1. imax:=i
   2. Цикл j=i,n
      1. Условие если A(imax)<A(j) тогда imax:=j
      2. Конец цикла j
   3. Условие если imax<>i тогда:

7.3.1 C:=A(i)

7.3.2 A(i):=A(imax)

7.3.3 A(imax):=C

7.3.4 конец условия

* 1. Конец цикла i

1. Вывод массива A(n)
2. Конец

|  |  |
| --- | --- |
| ***Программа на BASICe*** | ***Программа на PASCALe*** |
| REM Сортировка выбором  CLS  DIM n, i, j, imax AS INTEGER 'описание величин  INPUT "Введите количество элементов массива"; n  DIM A(n) AS INTEGER 'описание массива  FOR i = 1 TO n 'задание элементов массива  PRINT “ Введите “ , i, “ элемент массива”  INPUT A(i)  NEXT i  FOR i = 1 TO n ' вывод элементов массива на экран  PRINT A(i);  NEXT i  PRINT 'печать пустых строк для разделения массивов  PRINT  REM cортировка  FOR i = 1 TO n - 1  LET imax = i  FOR j = i + 1 TO n  IF A(j) > A(imax) THEN LET imax = j  NEXT j  IF i<>imax THEN SWAP A(i), A(imax)  NEXT i  FOR i = 1 TO n 'вывод отсортированных элементов массива на экран  PRINT A(i);  NEXT i  END | **program** sortirovra\_vyborom; {Сортиpовка массива выбором}  **type** {раздел описания типов переменных}  mas=**array**[1..100] **of** integer;  **var** {раздел описания переменных}  n,i,j,imax,c:integer;  a:mas;  **begin**  writeln('Введите размер массива ');  writeln;  readln(n);  {Ввод элементов массива}  writeln;  randomize;  **for** i:=1 **to** n **do**  **begin**  writeln(' Введите ',i,' элемент массива');  readln(a[i])  **end**;  **for** i:=1 **to** n **do** {Вывод элементов массива на экран}  write(a[i],' ');  writeln;  writeln;  writeln;  **for** i:=1 **to** n-1 **do** {сортировка массива}  **begin**  imax:=i;  **for** j:=i+1 **to** n **do**  **if** a[j]>a[imax] **then** imax:=j;  **if** a[j]>a[imax] **then**  **begin**  c:=a[i];  a[i]:=a[imax];  a[imax]:=c  **end**  **end**;  writeln;  writeln('упорядоченный массив');  writeln;  **for** i:=1 **to** n **do**  write(a[i],' ');  writeln  **end**. |
| **Вариант 2**  **(Задание массива случайным образом)** | |
| REM Сортировка выбором  CLS  DIM n, i, j, imax AS INTEGER 'описание величин  INPUT "Введите количество элементов массива"; n  DIM A(n) AS INTEGER 'описание массива  RANDOMIZE TIMER 'инициализация датчика случайных чисел  FOR i = 1 TO n 'задание элементов массива и вывод их на экран  LET A(i) = RND(1) \* 101  PRINT A(i);  NEXT i  PRINT 'печать пустых строк для разделения массивов  PRINT  REM cортировка  FOR i = 1 TO n - 1  LET imax = i  FOR j = i + 1 TO n  IF A(j) > A(imax) THEN LET imax = j  NEXT j  SWAP A(i), A(imax)  NEXT i  FOR i = 1 TO n 'вывод отсортированных элементов массива на экран  PRINT A(i);  NEXT i  END | **program** sortirovra\_vyborom; {Сортиpовка массива выбором}  **type** {раздел описания типов переменных}  mas=**array**[1..100] **of** integer;  **var** {раздел описания переменных}  n,i,j,imax,c:integer;  a:mas;  **begin**  writeln('Введите размер массива ');  writeln;  readln(n);  {Ввод элементов массива}  writeln;  randomize;  **for** i:=1 **to** n **do**  a[i]:=random(101);  **for** i:=1 **to** n **do** {Вывод элементов массива на экран}  write(a[i],' ');  writeln;  writeln;  writeln;  **for** i:=1 **to** n-1 **do** {сортировка массива}  **begin**  imax:=i;  **for** j:=i+1 **to** n **do**  **if** a[j]>a[imax] **then** imax:=j;  **if** a[j]>a[imax] **then**  **begin**  c:=a[i];  a[i]:=a[imax];  a[imax]:=c  **end**  **end**;  writeln;  writeln('упорядочный массив');  writeln;  **for** i:=1 **to** n **do**  write(a[i],' ');  writeln  **end**. |

1. **Метод прямого выбора**

Сортировка прямым выбором (например, по возрастанию) осуществляется следующим образом:

1. Первый элемент массива сравнивается со всеми остальными элементами;
2. Если первый элемент больше, то сравниваемые элементы меняются местами;
3. Пункты a и b повторяются до тех пор, пока первый элемент не сравнится с последним элементом (первый элемент считается отсортированным);
4. В неотсортированной части массива снова повторяются пункты a-c;
5. Действия, описанные в пункте d, повторяются с неотсортированными элементами массива до тех пор, пока не останется один неотсортированный элемент (его значение будет максимальным).

Временная сложность сортировки прямым выбором квадратичная, так как время сортировки пропорционально квадрату числа сортируемых элементов.

**Отсортировать массив методом прямого выбора**

**Арг.** N – цел., кол-во элементов массива

A(N) – цел., элементы массива

**Рез.** A(N) – цел., элементы отсортированного массива

**Пром.** i, j – цел., у. п. ц.

**вел.** С– цел., элемент сортируемого массива

***Тест***

Исходный массив: 8, 3, 6, 1, 4, 16

N = 6

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i/j | 1 | | 2 | | 3 | | 4 | | 5 | |
|  | 2 | 3,8,6,1,4, 16 | 3 | 1,6,8,3,4,16 | 4 | 1,3,6,8,4,16 | 5 | 1,3,4,6,8,16 | 6 | 1,3,4,6,8,16 |
|  | 3 | 3,8,6,1,4, 16 | 4 | 1,3,8,6,4,16 | 5 | 1,3,4,8,6,16 | 6 | 1,3,4,6,8,16 |  |  |
|  | 4 | 1,8,6,3,4,16 | 5 | 1,3,8,6,4,16 | 6 | 1,3,4,8,6,16 |  |  |  |  |
|  | 5 | 1,8,6,3,4,16 | 6 | 1,3,8,6,4,16 |  |  |  |  |  |  |
|  | 6 | 1,8,6,3,4,16 |  |  |  |  |  |  |  |  |

***Алгоритм***

1. Начало
2. Определение величин
3. Ввод n
4. [Определение массива A(n) – для бейсика]
5. Цикл ввода массива
6. Цикл вывода массива на экран
7. Цикл i=1,n-1
   1. Цикл j=i+1, n
      1. Условие если A(i)>A(j) тогда:

7.1.1.1 C:=A(i)

7.1.1.2 A(i):=A(imax)

7.1.1.3 A(imax):=C

7.1.1.4 конец условия

* + 1. Конец цикла j
  1. Условие если imax<>i тогда:
  2. Конец цикла i

1. Вывод массива A(n)
2. Конец

|  |  |
| --- | --- |
| ***Программа на BASICe*** | ***Программа на PASCALe*** |
| REM сортировка элементов массива методом прямого выбора  CLS  DIM n, i, b, j AS INTEGER  INPUT "Введи количество элементов массива"; n  DIM A(n) AS INTEGER  FOR i = 1 TO n 'задание элементов массива  PRINT “ Введите “ , i, “ элемент массива”  INPUT A(i)  NEXT i  FOR i = 1 TO n ' вывод элементов массива на экран  PRINT A(i);  NEXT i  PRINT  FOR i = 1 TO n - 1  FOR j = i + 1 TO n  IF A(i) > A(j) THEN  LET b = A(i)  LET A(i) = A(j)  LET A(j) = b  END IF  NEXT j  NEXT i  PRINT  FOR i = 1 TO n  PRINT A(i);  NEXT i  PRINT  END | program sortirovra\_vyborom; {Сортировка массива выбором}  uses crt;{подключения блока crt }  type {раздел описания типа переменных}  mas=array[1..100] of integer;  var { раздел описания переменных}  n,i,j,c:integer;  a:mas;  begin  clrscr; {îочистка экрана}  writeln(Введите количество элементов массива ');  readln(n);  {Ввод элементов массива}  for i:=1 to n do  begin  writeln('Введите ',i,' элемент массива');  readln(a[i]);  end;  writeln;  writeln;  writeln;  for i:=1 to n do {Вывод элементов массива на экран}  write(a[i],' ');  writeln;  writeln;  writeln;  for i:=1 to n-1 do {сортировка массива}  for j:=i+1 to n do  if a[i]>a[j] then  begin  c:=a[i];  a[i]:=a[j];  a[j]:=c;  end;  writeln;  writeln;  writeln;  writeln(' упорядоченный массив');  writeln;  for i:=1 to n do  write(a[i],' ');  writeln;  end. |
| **Вариант 2**  **(Задание массива случайным образом)** | |
| REM сортировка элементов массива  CLS  DIM n, i, b, j AS INTEGER  INPUT "Введи количество элементов массива"; n  DIM A(n) AS INTEGER  RANDOMIZE TIMER  FOR i = 1 TO n  LET A(i) = RND(1) \* 100  PRINT A(i);  NEXT i  PRINT  FOR i = 1 TO n - 1  FOR j = i + 1 TO n  IF A(i) > A(j) THEN  LET b = A(i)  LET A(i) = A(j)  LET A(j) = b  END IF  NEXT j  NEXT i  PRINT  FOR i = 1 TO n  PRINT A(i);  NEXT i  PRINT  END | program sortirovra\_vyborom; {Сортировка массива выбором}  uses crt;{подключения блока crt }  type {раздел описания типа переменных}  mas=array[1..100] of integer;  var { раздел описания переменных}  n,i,j,c:integer;  a:mas;  begin  clrscr; {îочистка экрана}  writeln(Введите количество элементов массива ');  readln(n);  {Ввод элементов массива}  writeln;  randomize;  for i:=1 to n do  a[i]:=random(101);  for i:=1 to n do {Вывод элементов массива на экран}  write(a[i],' ');  writeln;  writeln;  writeln;  for i:=1 to n-1 do {сортировка массива}  for j:=i+1 to n do  if a[i]>a[j] then  begin  c:=a[i];  a[i]:=a[j];  a[j]:=c;  end;  writeln;  writeln;  writeln;  writeln(' упорядоченный массив');  writeln;  for i:=1 to n do  write(a[i],' ');  writeln;  end. |

1. **Метод пузырька (обменная сортировка)**

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде, отсюда и название алгоритма). В отличии от сортировки выбором количество шагов обменной сортировки зависит от первоначального значения массива. Временная сложность алгоритма в лучшем случае равна временной сложности 1 шага. В худшем случае для сортировки массива из N элементов потребуется N шагов, то есть временная сложность квадратична.

**Отсортировать массив методом пузырька**

Арг.

N – цел., кол-во элементов массива

A(N) – цел., элементы массива

Рез.

A(N) – цел., элементы отсортированного массива

Пром. вел.

С – цел., величина для замены

i, j – цел., у. п. ц.

S – цел., счётчик замен

Исходный массив: 8, 3, 6, 1, 4, 16

N = 6

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i/j | 1 | | 2 | | 3 | | 4 | |
| 1 |  | 3,8,6,1,4,16 |  | 3,6,1,4,8,16 |  | 1,3,4,6,8,16 |  | 1,3,4,6,8,16 |
| 2 |  | 3,6,8,1,4,16 |  | 3,1,6,4,8,16 |  | 1,3,4,6,8,16 |  | 1,3,4,6,8,16 |
| 3 |  | 3,6,1,8,4,16 |  | 3,1,4,6,8,16 |  | 1,3,4,6,8,16 |  | 1,3,4,6,8,16 |
| 4 |  | 3,6,1,4,8,16 |  | 3,1,4,6,8,16 |  | 1,3,4,6,8,16 |  | 1,3,4,6,8,16 |
| 5 |  | 3,6,1,4,8,16 |  | 3,1,4,6,8,16 |  | 1,3,4,6,8,16 |  | 1,3,4,6,8,16 |

Отсортированный массив: 1,3,4,6,8,16

***Алгоритм***

1. Начало
2. Определение величин
3. Ввод n
4. [Определение массива A(n) – для бейсика]
5. Цикл ввода массива
6. Цикл вывода массива на экран
7. Цикл i=1,n-1
   1. S:=0
   2. Цикл j=i+1, n
      1. Условие если A(i)>A(j) тогда:

7.2.1.1 C:=A(i)

7.2.1.2 A(i):=A(j)

7.2.1.3 A(j):=C

7.2.1.4 S:1

7.2.1.5 конец условия

* + 1. Конец цикла j
  1. Условие если S=0 тогда: выход из цикла i
  2. Конец цикла i

1. Вывод массива A(n)
2. Конец

|  |  |
| --- | --- |
| ***Программа на BASICe*** | ***Программа на PASCALe*** |
| REM сортировка элементов массива  CLS  DIM n, i, b, j AS INTEGER  INPUT "Введи количество элементов массива"; n  DIM A(n) AS INTEGER  FOR i = 1 TO n 'задание элементов массива  PRINT “ Введите “ , i, “ элемент массива”  INPUT A(i)  NEXT i  FOR i = 1 TO n ' вывод элементов массива на экран  PRINT A(i);  NEXT i  PRINT  FOR i = 1 TO n - 1  FOR j = i + 1 TO n  IF A(i) > A(j) THEN  LET b = A(i)  LET A(i) = A(j)  LET A(j) = b  END IF  NEXT j  NEXT i  PRINT  FOR i = 1 TO n  PRINT A(i);  NEXT i  PRINT  END | program sor\_puz; // Сортировка массива методом пузырька, случайный массив  type {раздел описания типов переменных}  mas=array[1..100] of integer;  var {раздел описания переменных}  n,i,j,imax,c,s:integer;  a:mas;  begin  writeln(' Введите размер массива ');  readln(n);  {Ввод элементов массива}  randomize;  for i:=1 to n do  **begin**  writeln(' Введите ',i,' элемент массива');  readln(a[i]);  **end**;  for i:=1 to n do {Вывод элементов массива на экран}  write(a[i],' ');  writeln;  writeln;  writeln;  for i:=1 to n-1 do {сортировка массива}  begin  s:=0;  for j:=i+1 to n do  if a[j-1]>a[j] then  begin  c:=a[j-1];  a[j-1]:=a[j];  a[j]:=c;  s:=1  end;  if s=0 then break  end;  writeln(' упорядочный массив');  for i:=1 to n do  write(a[i],' ');  writeln;  end. |
| **Вариант 2**  **(Задание массива случайным образом)** | |
| REM сортировка элементов массива  CLS  DIM n, i, b, j AS INTEGER  INPUT "Введи количество элементов массива"; n  DIM A(n) AS INTEGER  RANDOMIZE TIMER  FOR i = 1 TO n  LET A(i) = RND(1) \* 100  PRINT A(i);  NEXT i  PRINT  FOR i = 1 TO n - 1  FOR j = i + 1 TO n  IF A(i) > A(j) THEN  LET b = A(i)  LET A(i) = A(j)  LET A(j) = b  END IF  NEXT j  NEXT i  PRINT  FOR i = 1 TO n  PRINT A(i);  NEXT i  PRINT  END | program sor\_puz; // Сортировка массива методом пузырька, случайный массив  type {раздел описания типов переменных}  mas=array[1..100] of integer;  var {раздел описания переменных}  n,i,j,imax,c,s:integer;  a:mas;  begin  writeln(' Введите размер массива ');  readln(n);  {Ввод элементов массива}  randomize;  for i:=1 to n do  a[i]:=random(101);  for i:=1 to n do {Вывод элементов массива на экран}  write(a[i],' ');  writeln;  writeln;  writeln;  for i:=1 to n-1 do {сортировка массива}  begin  s:=0;  for j:=i+1 to n do  if a[j-1]>a[j] then  begin  c:=a[j-1];  a[j-1]:=a[j];  a[j]:=c;  s:=1  end;  if s=0 then break  end;  writeln(' упорядочный массив');  for i:=1 to n do  write(a[i],' ');  writeln;  end. |

1. **Метод слияния массивов**

Алгоритм сортировки слиянием был предложен отцом современных компьютеров – Джоном фон Нейманом в 1945 году. Сам метод является *устойчивым*, т. е. он не меняет одинаковые по значению элементы в списке. В основе сортировки слиянием лежит принцип «разделяй и властвуй». Список разделяется на равные или практически равные части, каждая из которых сортируется отдельно. После чего уже упорядоченные части сливаются воедино. Несколько детально этот процесс можно расписать так:

1. Сортируемый массив разбивается на две части одинакового размера;
2. Рекурсивное разбиение задачи происходит до тех пор, пока размер массива не достигнет единицы (массив длиной один можно считать упорядоченным);
3. Два упорядоченных массива соединяются в один;
4. На каждом шаге берём меньший из двух первых элементов отсортированных массивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, откуда был взят элемент увеличивают на единицу;
5. Когда один из подмассивов кончился, добавляем все оставшиеся элементы второго подмасива в результирующий массив.

Время работы алгоритма порядка t~(n \* log n)

**Отсортировать массив методом слияния**

Арг. n – цел., кол-во элементов массива;

A(n) – цел., массив из n элементов.

Рез. A(n) – цел., массив из n отсортированных элементов.

Пром. вел. i, j – цел., у.п.ц.

nach, kon – цел., начало и конец элементов сортировки

sr, st, fin – цел.,

M(n) – цел.

Исходный массив:

20 72 13 42 62 73 51 30 15 86 21 20 0 5 70 100

↓

20 72 13 42 62 7351 30 15 86 21 20 0 5 70 100

↓ ↓

20 72 13 42 62 73 51 30 15 86 21 20 0 5 70 100

↓ ↓ ↓ ↓

20 72 13 42 62 73 51 30 15 86 21 20 0 5 70 100

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

20 72 13 42 62 73 51 30 15 86 21 20 0 5 70 100

↘↙↘↙↘↙↘↙↘↙ ↘↙↘↙ ↘ ↙

20 72 13 42 62 73 30 51 15 86 20 21 0 5 70 100

↘ ↙ ↘ ↙ ↘ ↙ ↘ ↙

13 20 42 72 30 51 62 73 15 20 21 86 0 5 70 100

↘ ↙ ↘ ↙

13 20 30 42 51 62 72 73 0 5 15 20 21 70 86 100

↘ Отсортированный массив ↙

0 5 13 15 20 20 21 30 42 51 62 70 72 73 86 100

***Алгоритм***

1. Начало
2. Определение величин
3. Ввод n
4. [Определение массива A(n) – для бейсика]
5. Цикл ввода массива
6. Цикл вывода массива на экран
7. Подпрограмма сортировки (sort(A(n),1,n))
8. Вывод массива A(n)
9. Конец

***Алгоритм подпрограммы рекурсивной схемы сортировки***

(sort(A(n),1,n))

1. Начало
2. Описание nach, kon
3. Если nach<kon то
   1. Подпрограмма сортировки (sort(A(n),nach,(nach+kon)/2)
   2. Подпрограмма сортировки (sort(A(n),(nach+ron)/2+1,kon))
   3. Подпрограмма слияния массивов
4. Конец

***Алгоритм подпрограммы слияния массивов***

1 Описание (St,sr,fin,j,M(n))

2 Присвоение начальных условий (st=nach, sr=(nach+kon)/2, fin=sr+1)

3 цикл по j от nach до kon

3.1 если (st<=sr) и ((fin>kon) или (A(st)<A(fin))) то

3.1.1 M(j)=A(st)

3.1.2 St=st+1

иначе

* + 1. M(j)=A(fin)
    2. fin=fin+1

3.2 конец цикла

4 цикл по j от nach до kon

* 1. A(j)=M(j)
  2. конец цикла

5 конец

|  |  |
| --- | --- |
| ***Программа на BASICe*** | ***Программа на PASCALe*** |
| DECLARE SUB sort (A!(), nach!, kon!)  DECLARE SUB sbor (A!(), nach!, kon!)  REM Сортировка массива слиянием  CLS  DIM SHARED n, i AS INTEGER 'описание величин  DIM SHARED nach, kon AS INTEGER  INPUT "Введите количество элементов массива"; n  DIM SHARED A(n) AS INTEGER 'описание массива  RANDOMIZE TIMER 'иницилизация датчика случайных чисел  FOR i = 1 TO n 'задание элементов массива и вывод их на экран  LET A(i) = RND(1) \* 101  PRINT A(i);  NEXT i  PRINT 'печать пустых строк для разделения массивов  PRINT  REM cортировка  CALL sort(A(), 1, n)  FOR i = 1 TO n 'вывод элементов массива на экран  PRINT A(i);  NEXT i  END  SUB sbor (A(), nach, kon)  DIM sr, st, fin, j AS INTEGER  DIM M(n) AS INTEGER  LET sr = (nach + kon) / 2'вычисление среднего элемента  LET st = nach 'начало левой части  LET fin = sr + 1 'начало правой части  FOR j = nach TO kon 'выполнять от начала до конца  IF (st <= sr) AND ((fin > kon) OR (A(st) < A(fin))) THEN  LET M(j) = A(st)  LET st = st + 1  ELSE  LET M(j) = A(fin)  LET fin = fin + 1  END IF  NEXT j  REM Возвращение результатов в массив  FOR j = nach TO ko  LET A(j) = M(j)  NEXT j  END SUB  SUB sort (A(), nach, kon) 'Рекурсивная процедура сортировки  IF nach < kon THEN  CALL sort(A(), nach, (nach + kon) / 2)'Сортировка левой части  CALL sort(A(), (nach + kon) / 2 + 1, kon)'Сортировка правой части  CALL sbor(A(), nach, kon)'слияние массивов  END IF  END SUB | program sortirovra\_sliyaniem; {Сортиpовка массива слиянием}  uses crt;{подключение блока crt}  const {раздел объявления констант}  konec=100;  type {раздел описания типов переменных}  mas=array[1..konec] of integer;  var {раздел описания переменных}  n,i:integer;  a:mas;  {процедура слияния массивов}  procedure sbor(var a:mas;nach,kon:integer);  var sr,st,fin,j: integer;  m:mas;  begin  sr:=(nach+kon) div 2; {вычисление среднего элемента}  st:=nach; {начало левой части}  fin:=sr+1; {начало правой части}  for j:=nach to kon do {выполнять от начала до конца}  if (st<=sr) and ((fin>kon) or (a[st]<a[fin])) then  begin  m[j]:=a[st];  inc(st); {функция прибавления единицы к st}  end  else  begin  m[j]:=a[fin];  inc(fin);  end;  {возвращение результата в массив}  for j:=nach to kon do  a[j]:=m[j];  end;  {рекурсивная процедура сортировки}  procedure sort(var a:mas;nach,kon:integer);  begin  if nach<kon then  begin  sort(a,nach,(nach+kon) div 2); {сортировка левой части}  sort(a,(nach+kon) div 2 + 1,kon);{сортировка правой части}  sbor(a,nach,kon); {слияние массивов}  end;  end;  {основной блок программы}  begin  clrscr; {очистка экрана}  writeln('Введите размер массива ');  readln(n);  {Ввод элементов массива}  begin  randomize;  for i:=1 to n do  a[i]:=random(konec);  for i:=1 to n do  write(a[i],' ');  writeln;  writeln;  writeln;  end;  sort(a,1,n); {вызов сортирующей процедуры}  writeln('упорядочный массив');  for i:=1 to n do  write(a[i],' ');  writeln;  readln;  end. |

1. ***Некоторые дополнительные возможности***

Вообще того материала, что описан в предыдущих главах, вполне достаточно, чтобы решить любую ***задачу***. Однако если вы решили заняться программированием более глубоко и серьёзно, чем этого от вас требует школьная программа, вам могут оказаться полезными некоторые дополнительные возможности языков ***QB*** и ***PASCAL***. Особенно, если вы решили заняться проектной работой. Впрочем, мы и здесь не ставим своей целью полного изложения языков ***QB*** и ***PASCAL***.

* 1. **Конструкция выбора**

Иногда требуется осуществить выбор более чем из двух условий. Команда IF позволяет выполнить выбор из двух ветвей в зависимости от логического условия. Для большего числа ветвей данную команду надо использовать несколько раз. В этом случае лучше применить команду множественного выбора, позволяющий выбрать из списка одно из условий.

Представим себе, что вы пишете ***программу***, в которой надо поставить подряд много сокращённых ***ветвлений***, анализирующих значение одной ***переменной.*** Допустим, такая:

**Задача XV-1**

|  |  |
| --- | --- |
| День недели задаётся своим номером (число от 1 до 7). Сделать программу, выдающую название этого дня. | |
| ***Программа на BASICe*** | ***Программа на PASCALe*** |
| REM Иванов Иван 10А Дни недели  CLS  INPUT "Номер дня - ", i  IF i = 1 THEN  LET a$ = "Понедельник"  END IF  IF i = 2 THEN  LET a$ = "Вторник"  END IF  IF i = 3 THEN  LET a$ = "Среда"  END IF  IF i = 4 THEN  LET a$ = "Четверг"  END IF  IF i = 5 THEN  LET a$ = "Пятница"  END IF  IF i = 6 THEN  LET a$ = "Суббота"  END IF  IF i = 7 THEN  LET a$ = "Воскресенье"  END IF  PRINT  PRINT  PRINT  PRINT a$  END | program Ivanov Ivan Dni nedeli;  uses crt;  var  i:integer;  a:string;  begin  write[ln](‘ номер дня – ‘);  read[ln](i);  if i = 1 then  a = ‘понедельник';  if i = 2 then  a = ‘вторник';  if i = 3 then  a = ‘среда';  if i = 4 then  a = ‘четверг';  if i = 5 then  a = ‘пятница';  if i = 6 then  a = ‘суббота';  if i = 7 then  a = ‘воскресенье';  writeln;  writeln;  writeln;  writeln(a)  end. |

Получится примерно такая программа. Чем она нехороша? Тем, что, найдя нужный день, сделав всё, что нужно, она будет проверять все оставшиеся и тратить на это время. Как сделать так, чтобы, найдя нужный день, присвоив его название, программа не тратила время на проверки заведомо неверных условий? Для этого существует конструкция выбора.

1. ***BASIC***

Она начинается с команды **SELECT CASE,** после чего следует имя анализируемой ***переменной***. Потом следуют ***команды*** **CASE**, в которых говорится, что делать, если анализируемая переменная приобретает указанное значение. Завершается эта конструкция ***командой*** **END SELECT**.

Формат записи команды:

SELECT CASE I

CASE 1: КОМАНДЫ 1

CASE 2: КОМАНДЫ 2

CASE 3: КОМАНДЫ 3

CASE 4: КОМАНДЫ 4

CASE 5: КОМАНДЫ 5

CASE 6: КОМАНДЫ 6

…………………………….

CASE N: КОМАНДЫ N

CASE ELSE: КОМАНДЫ

END SELECT

I принимает значения от 1 до N

Если I=1, то выполняются команды 1, стоявшие после служебного слова CASE 1; Если I=2, то выполняются команды 2, стоявшие после служебного слова CASE 2; Если I=3, то выполняются команды 3, стоявшие после служебного слова CASE 3; …….; Если I= N, то выполняются команды N, стоявшие после служебного слова CASE N; Если I не равно значению от 1 до N, то выполняются команды, стоявшие после служебного слова CASE ELSE.

Блок SELECT CASE … END SELECT является логически законченной и практически независимой частью программы. Число CASE-блоков зависит от решаемой задачи.

Если одно из условий CASE истинное, то выполняется соответствующий блок команд. Если ни одно из них не выполняется, управление передаётся CASE ELSE, или, если CASE ELSE нет, команде, стоящему после END SELECT.

Проверять можно как числовые, так и текстовые выражения. Важно при этом не забывать, что операнды, используемые во всех условиях, должны иметь один и тот же тип.

1. ***PASCAL***

Формат записи команды:

CASE переключатель of

Список выбора 1: КОМАНДЫ 1

Список выбора 2: КОМАНДЫ 2

Список выбора 3: КОМАНДЫ 3

Список выбора 4: КОМАНДЫ 4

Список выбора 5: КОМАНДЫ 5

Список выбора 6: КОМАНДЫ 6

…………………………….

Список выбора N: КОМАНДЫ N

ELSE КОМАНДЫ 0

END;

Переключатель (I) принимает значения от 1 до N

Если I=1, то выполняются команды 1, стоявшие после служебного слова Список выбора 1; Если I=2, то выполняются команды 2, стоявшие после служебного слова Список выбора 2; Если I=3, то выполняются команды 3, стоявшие после служебного слова Список выбора 3; …….; Если I= N, то выполняются команды N, стоявшие после служебного слова Список выбора N; Если I не равно значению от 1 до N, то выполняются операторы, стоявшие после служебного слова ELSE.

Блок CASE … END; является логически законченной и практически независимой частью программы. Число CASE-блоков зависит от решаемой задачи.

Если одно из условий CASE истинное, то выполняется соответствующий блок команд. Если ни одно из них не выполняется, управление передаётся блоку ELSE, или, если блока ELSE нет, команде, стоящему после команды END;.

Проверять можно как числовые, так и текстовые выражения. Важно при этом не забывать, что операнды, используемые во всех условиях, должны иметь один и тот же тип.

|  |  |
| --- | --- |
| ***Программа на BASICe*** | ***Программа на PASCALe*** |
| REM Иванов Иван 10А Дни недели  CLS  INPUT ”Номер дня - ”, i  SELECT CASE i  CASE 1: LET a$ = "Понедельник"  CASE 2: LET a$ = "Вторник"  CASE 3: LET a$ = "Среда"  CASE 4: LET a$ = "Четверг"  CASE 5: LET a$ = "Пятница"  CASE 6: LET a$ = "Суббота"  CASE 7: LET a$ = "Воскресенье"  END SELECT  PRINT a$  END | program Ivanov Ivan Dni nedeli;  uses crt;  var  i:integer;  a:string;  begin  write[ln](‘ номер дня – ‘);  read[ln](i);  case i of  1: a = ‘понедельник';  2: a = ‘вторник';  3: a = ‘среда';  4: a = ‘четверг';  5: a = ‘пятница';  6: a = ‘суббота';  7: a = ‘воскресенье';  writeln;  writeln;  writeln;  writeln(a)  end. |

Вот как будет выглядеть ***программа*** с использованием такой конструкции.

Согласитесь, что так гораздо лучше. Посмотрите на эту программку, разберитесь досконально в ней.

* 1. **Опрос клавиатуры**

Если занятия у вас пойдут успешно, то скоро вы захотите сами сделать какую-нибудь компьютерную игру. А в играх редко обходится без опроса клавиатуры.

Для этого существует функции: в Бейсике - **INKEY$,** в Паскале **- READKEY**. Значение их символьное. Аргументом (входным данным) для неё является нажатие на какую-либо клавишу. Результатом – символьная величина. Если никакая клавиша не нажата, то величина эта пустая.

Для подавляющего большинства клавиш значение этой функции состоит из одного символа, нажатие же клавиш ←, →, ↑ , ↓ , Insert, Delete, Home, End, Page Up, Page Down даст значение, в два символа.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Клавиша | ← | → | ↑ , | ↓ , | Insert | Delete | Home |
| Код | 0, 75 | 0, 77 | 0, 72 | 0, 80 | 0, 82 | 0, 83 | 0, 71 |
| Клавиша | End | Page Up | Page Down | Tab⭾ | 🡄Backspace | **↵** Enter | Esc |
| Код | 0, 79 | 0, 73 | 0, 81 | 9 | 8 | 13 | 27 |

Ещё одно очень важное обстоятельство: большинству клавиш соответствуют обычные, видимые на экране символы, но есть и исключения: Esc, Tab⭾, 🡄Backspace, **↵** Enter и все клавиши, о которых шла речь в предыдущем абзаце. Эти невидимые символы различаются по своим кодам, для чего есть ***функция*** **CHR$()**. Она генерирует символ, соответствующий заданному коду.

В таблице приведены коды этих клавиш.

Пробел тоже невидим, но его можно показать как " " (‘ ‘ – в Паскале).

Как и прочие ***функции***, эта может использоваться либо в правой части ***команды присваивания***, либо в условии!

Как применять эту ***функцию*** на практике? В ***цикле с предусловием*** делается опрос клавиатуры, лучше всего значение функций **INKEY$** или **READKEY** присваивать какой-то символьной переменной. С помощью ***ветвлений***, а лучше ***выбора***, значение этой переменной нужно анализировать, и в зависимости от него выполнять те, или иные действия. При каких-то значениях этой переменной ***цикл*** завершается или продолжается.

А некоторым клавишам вообще не соответствуют никакие символы, ни видимые, ни невидимые. Вот эти клавиши:

Caps Lock, 🡅Shift, Ctrl, Alt, Print Scrn, Scroll Lock, Pause/Break, Num Lock.

Что касается функциональных клавиш, то они в себе содержат те символьные значения, которые запрограммированы специальной ***командой***.

* 1. **Команды, изменяющие естественный ход программы**

Работая ***с алгоритмическими структурами***, мы порой испытываем необходимость в их досрочном окончании. Простейшей такой командой является **goto*.*** Она имеет вид: **goto** *метка*. Команда передаёт управление оператору, помеченному меткой. Метка представляет собой идентификатор. Чтобы пометить оператор меткой необходимо перед ним указать метку с последующим двоеточием. Нельзя одной меткой помечать несколько операторов.

Данная процедура одинакова для Бейсика и Паскаля. Для Паскаля также необходимо описать метку в разделе описания меток:

**label** *метки*

Команду безусловного перехода надо использовать только в крайнем случае. По количеству использований судят о квалификации программиста, хотя иногда её применение бывает оправдано.

1. Бейсик

В бейсике для досрочного окончания алгоритмических структур существуют команды вида **EXIT**.

|  |
| --- |
| **EXIT IF, EXIT DO, EXIT FOR, EXIT SELECT, EXIT SUB.** |

Используются такие ***команды*** с помощью ***ветвления***, в котором написано ***условие*** прекращения выполнения ***структуры***. При этом ***структура*** закрывается, и структурность ***программы*** не нарушается. Однако и тут хочется сказать, что ***команды*** эти излишни, используя ***ветвление***, без них можно прекрасно обойтись.

1. Паскаль

Для завершения работы программ, процедур и функций без предвари- тельного перехода по меткам к закрывающему **end** в Паскале введены процедуры **Exit** и **Halt**.

Вызов **Exit** завершает работу своего программного блока и передает управление вызывающей программе. Если **Exit** выполняется в подпрограм- ме, то выполнение этой подпрограммы прекратится, и далее будет выпол- няться следующий за вызовом этой подпрограммы оператор. Если **Exit** вы- полняется в основной программе, выход из нее будет эквивалентен ее нормальному завершению.

Вызов процедуры **Halt**, где бы она не находилась, завершает работу программы и передает управление операционной системе.

Для завершения текущего шага цикла используется команда **continue**. Сразу после этой команды выполняется проверка условия окончания цикла. Данная команда находится внутри тела цикла.

Для завершения цикла используется команда **break**. Она передаёт управление команде, которая должна выполнятся после окончания цикла. Команда **break** находится внутри тела цикла.

* 1. **Генератор случайных чисел**

***Функция*** генератора случайных чисел уже упоминалась и даже использовалась. Тем не менее, хотим остановиться на ней подробней.

|  |  |
| --- | --- |
| Функция генератора случайных чисел | |
| ***BASIC*** | ***PASCAL*** |
| RND | RANDOM(N) |
| Значением этой ***функции*** является действительное число строго больше нуля и меньше единицы. Чтобы получить случайные числа в нужном диапазоне, нужно придумать нужную формулу.  Например, если вам надо получить число от 0 до N, то можно написать так: LET a=RND\* N+1 | Значением этой ***функции*** является целое число в диапазоне 0… N-1.  Например, если вам надо получить число от 0 до N, то можно написать так: a:= RANDOM(N)+1 |
| Чтобы числа генерировались действительно случайными, перед использованием этой ***функции*** нужно дать команду | |
| **RANDOMIZE TIMER** | **RANDOMIZE** |

Иначе цепочка этих чисел будет всегда одна и та же.

* 1. **Графические команды**

В главе VII уже шла речь о ***командах*** работы с графикой, в частности, о ***командах*,** рисующих линии и окружности. В этом разделе мы рассмотрим ***команды***, рисующие другие фигуры.

1. ***BASIC***

**Рисование прямоугольника.**

**LINE(x1,y1)-(x2,y2) ,[ цвет][[13]](#footnote-13),B[F]**

х1,у1 – координаты левого верхнего угла.

х2,у2 – координаты правого нижнего угла.

F – закрашенный прямоугольник.

цвет – номер цвета прямоугольника.

**Рисование эллипсов.**

**CIRCLE (x,y),r, [цвет] ,,,kt**– абсолютная форма

х,у – координаты центра окружности.

цвет – номер цвета эллипса.

r – радиус окружности.

kt – коэффициент (отношение оси Y к оси X эллипса).

**CIRCLE STEP (x,y) ,r, [цвет] ,,,kt** – относительная форма, где х, у смещение центра окружности относительно ТПС (Точки Последней Ссылки), r – радиус окружности, kt – коэффициент (отношение оси Y к оси X эллипса), цвет – номер цвета эллипса.

**Рисование дуг и секторов.**

**CIRCLE (x,y),r, [цвет] ,nd, kd[, kt]**– абсолютная форма

х,у – координаты центра окружности.

r – радиус окружности.

цвет – номер цвета.

nd – начальная точка дуги, заданная в радианах.

kd – конечная точка дуги, заданная в радианах.

kt – коэффициент (отношение оси Y к оси X эллипса).

Формула перевода угла в градусах в угол в радианах: УР=УГ\*Pi/180

Фрагмент программы: let Pi=3.1415

let ur=ug\*Pi/180

**CIRCLE STEP (x,y) ,r, [цвет] , nd, kd[, kt]** – относительная форма, где х, у смещение центра окружности относительно ТПС (Точки Последней Ссылки), r – радиус окружности, nd – начальная точка дуги, заданная в радианах, kd – конечная точка дуги, заданная в радианах, k – коэффициент (отношение оси Y к оси X эллипса), цвет – номер цвета.

**Закрашивание замкнутой области**

**PAINT (x,y), цвет, цвет контура**

х,у – координаты точки внутри замкнутого контура.

цвет – номер цвета для закрашивания области.

цвет контура – номер цвета для контура области.

**Рисование сложной фигуры**

**DRAW** строка

строка – текстовое выражение(последовательность команд графического языка).

Команды оператора **DRAW(**n, m, x и y –целые числа)

|  |  |
| --- | --- |
| Команда | Выполняемое действие |
| Un | Переместить вверх |
| Dn | Переместить вниз |
| Ln | Переместить влево |
| Rn | Переместить вправо |
| En | Переместить вверх и вправо по диагонали |
| Fn | Переместить вниз и вправо по диагонали |
| Gn | Переместить вниз и влево по диагонали |
| Hn | Переместить вверх и влево по диагонали |
| M x, y | Переместить в точку с координатами (х, у) |
| B | Переместить, но не рисовать |
| N | Переместить, затем вернуть начальное значение ТПС |
| An | Задать угол поворота |
| Tan | Задать угол направления |
| Cn | Задать цвет |
| Sn | Задать масштаб |
| P n, m | Закрасить область |

1. ***PASCAL***

Для работе в графическом режиме необходимо подключить модуль **GraphABC**.

**Рисование прямоугольника.**

**rectangle****(x1,y1,x2,y2)**

рисует контур прямоугольника со сторонами параллельными сторонам экрана. Точка с координатами (х1,у1) определяет левый верхний угол прямоугольника. Точка с координатами (х2,у2) – правый нижний угол прямоугольника.

**roundrect(x1,y1,x2,y2,w,h)**

рисует прямоугольник со скруглёнными краями. Точка с координатами (х1,у1) определяет левый верхний угол прямоугольника. Точка с координатами (х2,у2) – правый нижний угол прямоугольника. w и h – ширина и высота эллипса, используемого для скругления краёв.

**fillrect(x1,y1,x2,y2)**

заливает прямоугольник цветом текущей кисти. Точка с координатами (х1,у1) определяет левый верхний угол прямоугольника. Точка с координатами (х2,у2) – правый нижний угол прямоугольника.

**setbrushcolor(color)**

устанавливает цвет кисти, задаваемый параметром color

**Рисование эллипсов.**

**ellipse (x1,y1,x2,y2)**

рисует эллипс, заданным своим описанным прямоугольником с координатами противоположных вершин (х1,у1) и (х2,у2)

**Рисование дуг и секторов.**

**Arc****(x,y,r,ug1,ug2)**

Рисует дугу окружности с центром в точке (х, у) и радиусом r, заключённой между двумя лучами, образующей углы ug1 и ug2 с осью ОХ (ug1 и ug2 – вещественные, задаются в градусах и отсчитываются против часовой стрелки).

**chord(x,y,r,ug1,ug2)**

Рисует фигуру, ограниченной дугой окружности с центром в точке (х, у) и радиусом r, заключённой между двумя лучами, образующей углы ug1 и ug2 с осью ОХ (ug1 и ug2 – вещественные, задаются в градусах и отсчитываются против часовой стрелки) и отрезком, соединяющим её концы.

**Закрашивание замкнутой области**

**floodfill(x, y, color)**

заливает область одного цвета цветом color, начиная с точки (х, у)

* 1. **Операции целочисленного деления**

Данные ***операции*** используются для целочисленного деления (делят одно число на другое нацело, находят остаток от деления одного числа на другое). Как и любые другие операции, эти можно употреблять лишь в правой части ***команды присваивания*** и в ***условиях***. Например:

|  |  |  |
| --- | --- | --- |
| ***BASIC*** | ***PASCAL*** | ***Пояснения*** |
| a\b | a div b | Целочисленное деление a на b |
| a mod b | a mod b | Остаток от целочисленного деления a на b |

* 1. **Ввод и вывод данных в файлы**

В предыдущих главах при решении ***задач*** нам казалось само собой разумеющимся, что ***входные данные*** всегда вводятся с клавиатуры, а ***выходные*** выводятся на экран. Однако это вовсе не обязательно. Часто, например на олимпиадах, необходимо входные данные вводить из файлов, а выходные выводить в файлы. Вот как это делается.

Во-первых, если мы хотим вводить данные из файла, то он должен быть заранее подготовлен. Это можно сделать либо вручную, либо программным способом.

Находясь в редакторе, можно сделать файл вручную так, как нам надо, а затем сохранить его в текстовом формате, дав нужное имя и расширение.

В программе необходимо открыть файл для чтения из него, для записи в него или для дозаписи в него.

1. ***BASIC***

Это делается с помощью оператора OPEN. Формат записи:

**OPEN <имя файла> FOR <режим> AS #<номер>**

где: имя файла – указывается в кавычках файл текущего каталога. Для записи, чаще всего используется, файл «output.txt**»;** для чтения – «input.txt».

режим:

1. **INPUT** – открывает существующий файл для чтения хранимой там информации;
2. **OUTPUT** – открывает новый файл для записи. Если файл с таким именем уже существует, то вся информация в нём стирается и запись производится с начала файла;
3. **APPEND** – записывает информацию в конец уже в существующего файла. Если такого файла не существует, то он открывается и запись производится с начала файла.

Номер: присваивает открытому файлу определённый номер (число от 1 до 255).

После работы с открытым файлом его надо закрыть. Закрывается он с помощью оператора CLOSE [#<номер >]. Если номер указан, то закрывается файл с указанным номером, если нет, то все открытые файлы.

1. ***PASCAL***

Мы должны описать файловую переменную для работы с текстовым файлом:

var

**<переменная>:text;**

Связать файловую переменную с конкретным файлом:

**assign(<переменная>, <имя файла>)**

где: имя файла – указывается в апострофах файл текущего каталога. Для записи, чаще всего используется, файл «output.txt**»;** для чтения – «input.txt».

режимы работы:

1. **reset(<переменная>)** – открывает существующий файл для чтения хранимой там информации;
2. **rewrite(<переменная>)** – открывает новый файл для записи. Если файл с таким именем уже существует, то вся информация в нём стирается и запись производится с начала файла;
3. **append(<переменная>)** – записывает информацию в конец уже в существующего файла. Если такого файла не существует, то он открывается и запись производится с начала файла.

После работы с открытым файлом его надо закрыть. Закрывается он с помощью оператора **close**

Как вывести данные в файл программным способом, покажем на примере. Допустим, у нас имеется числовой массив **a**, состоящий из **n** элементов. Надо вывести его в файл **output.txt**. Покажем в виде ***подпрограммы***.

|  |  |  |
| --- | --- | --- |
| vimf:  OPEN "output.txt" FOR OUTPUT AS #1  PRINT #1, n  FOR i = 1 TO n STEP 1  PRINT #1, a(i);  NEXT i  CLOSE #1  RETURN | vimf:  var  fail:text;  n:integer;  ………………..  procedure vimf;  var  i:integer;  begin  assign(fail,’output.txt’);  rewrite(fail);  write(fail, n);  for i:=1 to n do  write(fail, ‘ ‘, a[i]);  close  end; | Вывод ***массива*** в файл. Всё почти как при выводе на экран. |

Небольшая разница в программе будет, если нам надо вывести массив в уже существующий файл (дозаписать файл).

|  |  |  |
| --- | --- | --- |
| vimf:  OPEN "output.txt" FOR APPEND AS #1  PRINT #1, n  FOR i = 1 TO n STEP 1  PRINT #1, a(i);  NEXT i  CLOSE #1  RETURN | vimf:  var  fail:text;  n:integer;  ………………..  procedure vimf;  var  i:integer;  begin  assign(fail,’output.txt’);  append(fail);  write(fail, n);  for i:=1 to n do  write(fail, ‘ ‘, a[i]);  close  end; | Вывод ***массива*** в существующий файл. Всё почти как при выводе на экран. |

Ввод из существующего файла осуществляется так:

|  |  |  |
| --- | --- | --- |
| vmf:  OPEN "input.txt" FOR INPUT AS #1  INPUT #1, n  FOR i = 1 TO n STEP 1  INPUT #1, a(i)  NEXT i  CLOSE #1  RETURN | vmf:  var  fail:text;  n:integer;  ……………………………………………………  procedure vimf;  var  i:integer;  begin  assign(fail,’output.txt’);  reset(fail);  read(fail, n);  for i:=1 to n do  read(fail, a[i]);  close  end; | Ввод ***массива*** из файла. Всё почти как при вводе с клавиатуры. |

* 1. **Форматный вывод информации**

a) BASIC

**Функции оператора print**

Операторы PRINT выводят информацию на экран в виде колонки. Например, возьмём фрагмент программы (слева – программа, справа – вид экрана):

при n=3 на экране будут следующие значения

|  |  |
| --- | --- |
| inрut «введи n»;n  for i=1 to n  print i  next i | «введи n»; ? 3  1  2  3 |

Курсор каждый раз переходит на новую строчку и на ней происходит печать очередной переменной.

Для выведения информации в строку необходимо после оператора print поставить точку с запятой. Рассмотрим тот же пример:

при n=3 на экране будут следующие значения

|  |  |
| --- | --- |
| inрut «введи n»;n  for i=1 to n  print i  next i | «введи n»; ? 3  1 2 3 |

Курсор на новую строчку не переходит и печать очередной переменной происходит на той же строке.

Если же у нас в операторе PRINT стоит несколько элементов, разделённых запятой или точкой с запятой, то вывод получится в виде таблицы. Если разделитель точка с запятой, то между выводимыми величинами ставится пробел. Если – запятая, то экран монитора делится примерно на 5 частей и выводимые величины помещаются в эту часть. Например, фрагмент программы подсчёта факториала (слева – программа, справа – вид экрана):

при n=5 на экране будут следующие значения

|  |  |
| --- | --- |
| inрut «введи n»;n  let f=1  print “ i, n!”  for i=1 to n  f=f\*i  print i, n  next i | «введи n»; ? 5  i n!   1. 1 2. 2 3. 6 4. 24 5. 120 |

|  |  |
| --- | --- |
| inрut «введи n»;n  let f=1  print “ i, n!”  for i=1 to n  f=f\*i  print i; n  next i | «введи n»; ? 5  i n!  1 1  2 2  3 6  4 24  5 120 |

Как видите, вывод начинается с первого столбца первой строки. Чтобы вывести, например, с десятой строки, надо набрать девять раз пустые операторы print. А вывести, например, с десятого столбца, надо набрать девять пробелов в операторе print после кавычек.

Для вывода с необходимого столбца в QBASICe существует **функция TAB.** Формат записи: **print tab(число);**. Число показывает с какого столбца будет производится печать и принимает значения от1 до 80. В нашем примере для десятого столбца оператор будет записан так: print tab(10);.

Мы можем разделить наши переменные необходимым количеством пробелов. Для этого к оператору print необходимо добавить **функцию SPC.** Формат записи: **print spc(количество пробелов);.** Количество пробелов – это число от 1 до 80. В примере print spc(17);“ i n!” между переменными i и n! будет 17 пробелов.

В записи примера без пробелов мы должны дать такую команду:

print tab(2);”i”; spc(17);“n!”

**Оператор print using.**

Но иногда бывают ситуации, когда всех вышеперечисленных функций оператора print не хватает. Например слишком большое число, которое не помещается на одной строке. Для вывода переменных в определённом формате существует оператор **print using.** Он позволяет:

* Размещать числа в колонки, выравнивая их по десятичной точке;
* Выводить часть текстовой информации;
* Ставить знак $ перед числом;
* Выводить числа в экспоненциальном формате;
* Определять точность выводимого числа.

Формат оператора**: print using «Спецификатор формата» <список выводимых выражений> разделитель ; или ,**

Спецификаторы формата

|  |  |
| --- | --- |
| Символ | Значение |
| # | Определяет положение для одного разряда числа |
| . | Десятичная точка |
| , | Разделяет каждые три разряда числа |
| + | Знак плюс |
| - | Знак минус |
| \*\* | Определяющие звёздочки перед числом |
| $ | Знак доллара перед числом |
| ^^^^ | Вывод в экспоненциальной форме |
| & | Вывод всего текстового значения |
| ! | Вывод первого символа текстового значения |
| \\ | Определение количества выводимых символов текстового значения (определяется количеством пробелов между знаками \\) |

Например:

Фрагмент программы (слева – программа, справа – вид экрана):

|  |  |
| --- | --- |
| a = 12398.7351  b = "Ехал Грека Через реку"  PRINT a; b  PRINT USING "###.#####";a  PRINT USING "#######.#" a  PRINT USING "\*\*\*#####.###";a  PRINT USING "#####.###^^^^^";a  PRINT USING "\*#######.##";a  PRINT USING "$########";a  PRINT USING "&"; b  PRINT USING "!"; b  PRINT USING "\\"; b  PRINT USING "\ \"; b | 12398.7351 Ехал Грека Через реку  %12398.73510  12398.7  %12399\*  1239.874E+001  \*12398.74  $12399  Ехал Грека Через реку  E  Ex  Ехал Грека |

Знак % ставится если число имеет больше знаков, чем указано в формате как символ несоответствия формата данному числу. Если формат содержит звёздочки, то это значит, что результат будет иметь столько символов до и после десятичной точки, сколько знаков \* и # содержится в соответствующих позициях числового поля. Если в числовом поле до десятичной точки символов больше, чем в самом выводимом числе, то соответствующие позиции при выводе будут заполнены звёздочками.

Две наклонные черты влево (\\) определяют сколько символов текстового значения предназначено для вывода на экран. По одному символу закреплено за каждой наклонной чертой и за каждым пробелом между ними.

Оператор **print using** удобен, если надо распечатать на экране несколько переменных по единому формату. Тогда можно завести переменную текстового формата, поместить в неё спецификации необходимого (текстового или числового) формата, и использовать её как параметр в операторе **print using.**

**Оператор locate**

Операторы print и его функции начинают выводит переменные с того места где стоит курсор. Курсор передвигается последовательно, после выполнения операторов, с первой строки до последней. Напомню, что же такое курсор.

***Курсор*** – это специальная метка активной в данный момент позиции экрана. Он существует только в текстовом режиме. Его ширина всегда равна ширине символа, а высота может регулироваться.

Текст всегда выводится на экран с той позиции, где находится курсор. Начальное положение курсора – по умолчанию, в левом верхнем углу экрана, - определяет местоположение вывода результатов работы программы, независимо от параметров текстового режима. Местоположение курсора меняется в течении вывода результатов. После вывода информации курсор находится после последнего выведенного символа.

QBASIC позволяет производить следующие операции с курсором:

* Включать и выключать курсор;
* Менять размеры курсора;
* Изменять местоположения курсора (строку и/или колонку);
* Определять местоположение курсора в любой момент выполнения программы.

Бывает необходимость начать вывод с определённого места экрана. Для этого в это место мы должны поместить курсор с помощью **оператора locate.**

**Формат вывода**: LOCATE строка, колонка, флаг, начало, конец

где

строка – номер строки, в которой будет размещён курсор. Это значение должно быть от 1 до значения максимально возможного количества строк (чаще всего – 25);

колонка – номер колонки (значение может быть от1 до 80);

флаг – делает курсор видимым или невидимым (1 или 0);

начало – номер первого светящегося сегмента курсора;

конец – номер последнего светящегося сегмента курсора.

Все значения, присваиваемые этим параметрам, должны быть целого типа. Каждый параметр может быть использован как в сочетании с другими, так и независимо от них.

Чаще всего оператор LOCATE с параметрами строка и/или колонка используется перед оператором PRINT для указания, в каком месте программа должна вывести значение или сообщение.

Используя третий параметр оператора LOCATE можно выключить изображение курсора на экране. Если параметр флаг равен 0, то курсор будет выключен, если – 1, тогда курсор будет включен.

Параметры начало и конец определяют какая часть курсора видна.

Курсор по своей структуре состоит из нескольких горизонтальных сегментов, каждая из которых характеризуется определённым номером, начиная сверху.

|  |  |  |
| --- | --- | --- |
| Структура курсора | | Такая структура курсора означает, что можно менять его вертикальный размер, так и положение относительно базы строки, высвечивая различные сегменты. |
|  | 0 |
|  | 1 |
|  | 2 |
|  | 3 |
|  | 4 |
|  | 5 |
|  | 6 |
|  | 7 |

Хотя чаще всего применяется процедура изменения положения курсора, иногда необходимо определить его текущее положение. Для этого используются функции: CSRLIN и POS.

Функция CSRLIN возвращает номер строки, в которой находится курсор (от 1 до 25).

Функция POS возвращает номер колонки (от 1 до 80). Функция POS имеет так называемый фиктивный параметр, который может быть любым целым числом от 0 до 32767. Формат вывода: POS(параметр). Например POS(10). Этот параметр не играет ни какой роли, он просто должен быть.

***b) PASCAL***

**Формат вывода** в языке Паскаль – это указываемое после двоеточия **число**, определяющее, сколько позиций на экране должна занимать выводимая величина. Если цифр в числе меньше, чем зарезервированных под него позиций на экране, то свободные позиции дополняются пробелами слева от числа. Если указанное число в формате вывода после двоеточия меньше, чем необходимо, то оно автоматически будет увеличено до минимального необходимого.

Для вывода вещественного числа в формате с фиксированной десятичной точкой в списке вывода для каждого выражения указываются два параметра:

1. общее количество позиций, отводимых под число;
2. количество позиций в дробной части числа.

Десятичная точка тоже считается отдельным символом.

Пример выполнения фрагмента программы:

var

А:real;

begin

А:=250;

Write(‘A=’,A)

Writeln(‘ A=’,A:3:0)

Write(‘A=’,A:3:2)

В результате выполнения данного фрагмента на экране появится:

А=2.5Е+02

А=250

А=250.00

**Установка курсора в определённое место**

Операторы write и writeln начинают выводит переменные с того места где стоит курсор. Курсор передвигается последовательно, после выполнения операторов, с первой строки до последней. Напомню, что же такое курсор.

***Курсор*** – это специальная метка активной в данный момент позиции экрана. Он существует только в текстовом режиме. Его ширина всегда равна ширине символа, а высота может регулироваться.

Текст всегда выводится на экран с той позиции, где находится курсор. Начальное положение курсора – по умолчанию, в левом верхнем углу экрана, - определяет местоположение вывода результатов работы программы, независимо от параметров текстового режима. Местоположение курсора меняется в течении вывода результатов. После вывода информации курсор находится после последнего выведенного символа.

Паскаль позволяет установить курсор в любое место экрана. Для реализации этой возможности в комплект языка входит особый дополнительный модуль CRT (это английская аббревиатура, обозначающая ЭЛТ). При работе в текстовом режиме информация на экран выводится в виде символов, каждый из которых отображается на экране в определённой позиции. Экран представляет из себя таблицу из 25 строк и 80 столбцов. Каждая ячейка этой таблицы имеет две координаты – х и у, где х – номер столбца, а у – номер строки. Левый верхний угол имеет координаты (1,1). Строки нумеруются сверху вниз, столбцы – слева направо.

Для вывода в любом месте экрана нам необходимо установить в эту точку курсор. Это делает процедура gotoxy(x,y). х меняется от 1 до 25, а у от 1 до 80.

Для того, что бы модуль работал его надо указать в начале программы в специальной секции объявления библиотечных модулей. Она начинается словом USES. Затем через запятую перечисляются подключаемые модули. Список заканчивается точкой с запятой.

Текст может выводится не только белым, но и цветным. В паскале используются 16 цветов. Каждому цвету присвоен номер от 0 до 15. Цвета приводятся в таблице:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N цвета | яркость | R | G | B | цвет |
| 0 | 0 | 0 | 0 | 0 | Чёрный |
| 1 | 0 | 0 | 0 | 1 | Синий |
| 2 | 0 | 0 | 1 | 0 | Зелёный |
| 3 | 0 | 0 | 1 | 1 | Голубой |
| 4 | 0 | 1 | 0 | 0 | Тёмно-красный |
| 5 | 0 | 1 | 0 | 1 | Фиолетовый |
| 6 | 0 | 1 | 1 | 0 | Коричневый |
| 7 | 0 | 1 | 1 | 1 | Белый |
| 8 | 1 | 0 | 0 | 0 | Серый |
| 9 | 1 | 0 | 0 | 1 | Светло синий |
| 10 | 1 | 0 | 1 | 0 | Светло зелёный |
| 11 | 1 | 0 | 1 | 1 | Светло голубой |
| 12 | 1 | 1 | 0 | 0 | Ярко красный |
| 13 | 1 | 1 | 0 | 1 | Розовый |
| 14 | 1 | 1 | 1 | 0 | Жёлтый |
| 15 | 1 | 1 | 1 | 1 | Ярко белый |

Для задания цвета символа используется процедура textcolor(номер цвета от 0 до 15). Вместо номера можно написать название цвета на английском языке. Данная процедура не меняет цвет уже выведенных символов, она устанавливает цвет следующих символов.

Мы также можем менять фоновый цвет экрана. Для этого служит процедура Textbackground (номер цвета от 0 до 15). Цвет также может указываться на английском языке.

Если на экране уже есть какая-либо надпись, а нам надо вывести новую, убрав старую. Для этого служит процедура очистки экрана: Clrscr

## Послесловие

Вот и закончилось ваше первое знакомство с ***алгоритмизацией*** и ***программированием***. Хочется надеяться, что оно не оказалось для вас бесполезным, что вы поняли самое главное.

Для кого-то это первая встреча с ***программированием*** окажется и последней, и при этом слове у него будет портиться настроение.

Кто-то, столкнувшись с ***программированием*** уже в студенческие годы, добром помянет годы школьные, и будет сам справляться с теми несложными задачами, к которым многие его сокурсники не будут знать, как подступиться.

А кто-то станет программистом, освоит современные ***языки программирования***, будет не просто писать ***программы***, а жить ими. Очень хочется надеяться, что среди читателей этой книжки найдутся и такие.

**Иркутск, 1996-2021.**

***Литература:***

* 1. А.П. Ершов, В.М. Монахов, А.А. Кузнецов, Я.Э. Гольц, М.П. Лапчик, А.С. Лесневский, Ю.А. Первин, Д.О. Смекалин ***«Основы информатики и вычислительной техники»***, ч. первая. М., «Просвещение», 1985.
  2. А.П. Ершов, В.М. Монахов, А.А. Кузнецов, Я.Э. Гольц, М.П. Лапчик, А.С. Лесневский, Ю.А. Первин, Д.О. Смекалин ***«Основы информатики и вычислительной техники»***, ч. вторая. М., «Просвещение», 1986.
  3. А.Г. Кушнеренко, Г.В. Лебедев, Р.А. Сворень ***«Основы информатики и вычислительной техники»***, учебник для ср. уч. заведений, М., «Просвещение», 1990.
  4. Г.Н. Загарий, В.М. Добрянский, В.С. Коновалов ***«Введение в технологию программирования»***, уч. пособие, Харьков, 1990.
  5. ***«Толковый словарь по вычислительным системам»***, под редакцией *В. Иллингуорта, Э.Л. Глейзера, И.К. Пайла*..., «Машиностроение», М., 1991.
  6. А.Г. Гейн, В.Г. Житомирский, Е.В. Линецкий, М.В. Сапир, В.Ф. Шолохович ***«Основы информатики и вычислительной техники»***, пробный учебник для 10-11 классов ср. школы, М., «Просвещение», 1992.
  7. Поляков Д. Б., Круглов И. Ю. **«Программирование в среде Турбо Паскаль (версия 5.5)»**: Справ.-метод. Пособие. – М.: Изд-во МАИ, 1992. – 576 с.
  8. Л.Г. Алсынбаева, Л.А. Голубева, Л.А. Москвина ***«Методика начального обучения алгоритмизации и программированию»***, Метод. материалы для преподавателя, НГУ, Новосибирск, 1996.
  9. Т.В. Добудько ***«Информатика 7 класс»***, корпорация «Фёдоров», Самара, 1997.
  10. А.Г. Гейн, А.И. Сенокосов, Н.А. Юнерман ***«Информатика 10-11»***, «Просвещение», М., 2000.
  11. Ю. Федоренко ***«Алгоритмы и программы на Qbasic»***, учебный курс, «Питер», Санкт-Петербург – Москва – Харьков – Минск, 2002.
  12. О.И. Мельникова, А.Ю. Бонюшкин ***«Начала программирования на языке Qbasic»***, издательство «ЭКОМ», Москва, 2000.
  13. С. Бобровский ***«Программирование на языке Qbasic для школьников и студентов»***, «Десс» «Инфоркм-Пресс», М., 1999.
  14. Т. Гутман ***«Изучаем*** ***Qbasic»***, «Питер», Санкт-Петербург – М. – Харьков – Минск, 2002.
  15. Ушаков Д. М., Юркова Т. А. «Паскаль для школьников».- СПб.:Лидер, 2010.-256 с.:ил
  16. Г.Б. Рейнгольд ***«О месте алгоритмизации и программирования в школьном курсе информатики»***. *(«Проблемы развития мышления в процессе преподавания математики и информатики». Материалы VI межрегиональной научно-практической конференции преподавателей школ, инновационных учебных заведений и вузов. ИГПУ, Иркутск, 1999.)*
  17. Г.Б. Рейнгольд **«В защиту Бейсика»**. *(«Совершенствование качества образования по курсам математики и информатики в современной школе». Материалы VII межрегиональной научно-практической конференции преподавателей школ, инновационных учебных заведений и вузов. ИГПУ, Иркутск, 2000.)*
  18. Г.Б. Рейнгольд ***«Использование средств компьютерной графики при изучении алгоритмических конструкций»***. *(«Теория и практика преподавания математики и информатики», сборник методических статей, выпуск 1, ИГПУ, Иркутск 2000.)*
  19. Г.Б. Рейнгольд ***«Аргументы и результаты»*** (сборник методических статей для учителей математики и информатики ИГПУ *«Теория и практика преподавания математики и информатики. Выпуск 2», Иркутск, 2001)*
  20. Г.Б. Рейнгольд ***«Разработка компьютерных игр, как средство первоначального обучения программированию»****. («Совершенствование методов преподавания математики и информатики». Материалы IX межрегиональной научно-практической конференции преподавателей школ, инновационных учебных заведений и вузов. ИГПУ. Иркутск, 2002.)*
  21. Г.Б. Рейнгольд, М.Г. Рейнгольд ***«Тестирование, как важный этап работы над программой»***. *(«Проблемы развития мышления в процессе преподавания математики и информатики». Материалы X межрегиональной научно-практической конференции преподавателей школ, инновационных учебных заведений и вузов. ИГПУ, Иркутск, 2003.)*
  22. Г.Б. Рейнгольд, М.Г. Рейнгольд, А.Н. Юркевич, Я.В. Курзыбова ***«Роль заочных олимпиад в деле подготовки юных программистов»***. *(«Совершенствование методов преподавания математики и информатики в условиях модернизации Российского образования». Материалы XI межрегиональной научно-практической конференции преподавателей школ, инновационных учебных заведений и вузов. ИГПУ, Иркутск, 2004.)*
  23. Г.Б. Рейнгольд, М.Г. Рейнгольд, А.Н. Юркевич, Я.В. Курзыбова ***«Играем в программистов!»***. *(Методические рекомендации по подготовке юных программистов. Иркутский ОЦТТУ, Иркутск, 2004)*
  24. Г.Б. Рейнгольд, А.Н. Юркевич, М.Г. Рейнгольд ***«V Областная заочная олимпиада по программированию среди обучающихся общего, профессионального и дополнительного образования «Играем в программистов» Шпиономания(шифрование и дешифрование)».****(Проблемы образования на современном этапе. Материалы XII межрегиональной научно-практической конференции преподавателей школ, инновационных учебных заведений и вузов. ИГПУ, Иркутск, 2005)*
  25. Г.Б. Рейнгольд, А.Н. Юркевич, О.В.Щербатов, М.Г. Рейнгольд **«Особенности второго тура олимпиады по информатике (из опыта работы в городе Иркутске)».** («Преподование информатики в школе и вузе: Проблемы и Перспективы». *Материалы XIII межрегиональной научно-практической конференции преподавателей школ, инновационных учебных заведений и вузов. ИГПУ, Иркутск, 2006.)*
  26. Г.Б. Рейнгольд, А.Н. Юркевич, М.Г. Рейнгольд **«Заочные олимпиады «Играем в программистов», шестилетняя история»**. («*Содержание математического образования в профильных классах с учётом единого государственного экзамена. Современные информационные образовательные технологии», Материалы XIV региональной научно-практической конференции преподавателей математики и информатики школ, инновационных учебных заведений и вузов. ИГПУ, Иркутск, 2007.)*
  27. Г.Б. Рейнгольд, А.Н. Юркевич, М.Г. Рейнгольд ***«Играем в программистов!(второй выпуск)»***. *(Материалы областных заочных олимпиад по программированию среди учащихся общего, профессионального и дополнительного образования. Иркутский ОЦДТТ, Иркутск, 2008*
  28. Г.Б. Рейнгольд, А.Н. Юркевич, М.Г. Рейнгольд **«Проблемы муниципального тура Всероссийской олимпиады школьников по информатике в г. Иркутске».***(«Современные проблемы обучения математике и информатике», Материалы II Всеросийской (XVI региональной) научно-практической конференции преподавателей математики и информатики школ, инновационных учебных заведений и вузов. ИГПУ, Иркутск, 2009.)*
  29. Г.Б. Рейнгольд, А.Н. Юркевич, М.Г. Рейнгольд **«О методике проведения соревнований юных исследователей и инжинеров».** (Материалы VII Международной конференции «Психологически-педагогические проблемы одарённости: теория и практика», Иркуктск, 2011)
  30. Г.Б. Рейнгольд, А.Н. Юркевич, М.Г. Рейнгольд **Заочные олимпиады «Играем в программистов».** (Материалы Международных педагогических чтений «Трапдиции и новации образовательной системы», Чебоксары, 2011)
  31. Г.Б. Рейнгольд, М.Г. Рейнгольд, А.Н. Юркевич ***«Из опыта работы творческого объединения Юный программист»***. *(Материалы Всероссийской научно-практической конференции «Повышение интереса детей и молодёжи к инженерному образованию», Москва, 2014)*

***Содержание:***

|  |  |  |
| --- | --- | --- |
| № | ***Раздел*** | Стр. |
|  | ***Предисловие*** | 3 |
| ***I*** | ***Алгоритмы*** | 5 |
| ***II*** | ***Язык Блок-Схем*** | 6 |
| ***III*** | ***Величины*** | 8 |
| ***IV*** | ***Интерфейс*** | 10 |
| ***V*** | ***Команда присваивания. Выражения. Функции.*** | 19 |
| ***VI*** | ***Команды ввода, вывода и очистки экрана*** | 23 |
| ***VII*** | ***Структура графического экрана. Основные графические команды*** | 29 |
| ***VIII*** | ***Линейные конструкции и алгоритмические структуры*** | 35 |
| ***IX*** | ***Простые и сложные условия*** | 37 |
| ***X*** | ***Ветвления*** | 39 |
| ***XI*** | ***Циклы*** | 43 |
| ***XII*** | ***Вспомогательные алгоритмы*** | 49 |
| ***XIII*** | ***Массивы*** | 55 |
| ***XIV*** | ***Сортировка массивов*** | 61 |
| ***XV*** | ***Некоторые дополнительные возможности*** | 77 |
|  | ***Послесловие*** | 94 |
|  | ***Литература*** | 95 |

Учебное издание

***Юркевич Александр Никитич***

***Рейнгольд Григорий Борисович***

***Рейнгольд Михаил Григорьевич***

***«Основы алгоритмизации и программирования» - 3-е издание***

Методическое пособие для обучающихся образовательных

учреждений по программированию

Рецензент

Редактор: Юркевич В. П.

Корректор

Компьютерный набор, вёрстка и оформление авторов.

1. Основы информатики и вычислительной техники [↑](#footnote-ref-1)
2. Принятые в ГАУ ДО ИО ЦРДОД и МБОУ ДОД ЦДТТ г. Иркутска. В главе ***IX***, посвящённой ***алгоритмическим структурам***, эти правила будут дополнены. [↑](#footnote-ref-2)
3. Порядок, описанный здесь, несколько отличается от того, что приведён в ***«Теоретических вопросах информатики»***. В дальнейшем этому будет дано объяснение. [↑](#footnote-ref-3)
4. Что такое ***входные*** и ***выходные данные***, подробно описано в ***«Теоретических вопросах информатики»***. [↑](#footnote-ref-4)
5. ***Тест*** это совокупность ***входных*** и соответствующих им ***выходных данных***. [↑](#footnote-ref-5)
6. ***«Толковый словарь по вычислительным системам»***, под редакцией *В. Иллингуорта, Э.Л. Глейзера, И.К. Пайла*..., «Машиностроение», Москва, 1991 [↑](#footnote-ref-6)
7. Звёздочками помечены те, обозначения которых отличаются от принятого в математике, или отсутствующие в математике! [↑](#footnote-ref-7)
8. Звёздочками помечены те, обозначения которых отличаются от принятого в математике, или отсутствующие в математике! [↑](#footnote-ref-8)
9. Это поможет вам не запутаться при написании сложной ***программы***, в которой степень вложений ***алгоритмических структур*** велика. Вы будете видеть, что над вами «висит» начало структуры, а значит, рано или поздно, надо её закрыть. Как бы высоко вы ни воспарили, как бы глубоко ни залезли, всегда надо помнить о том, что придётся возвратиться. [↑](#footnote-ref-9)
10. Если не произойдёт переполнение, тогда последует аварийная остановка ***программы***. [↑](#footnote-ref-10)
11. В ***программе*** есть ***переменные*** ***входные*** и ***выходные***. ***Переменные*** же, не являющиеся ни первыми, ни вторыми, называются ***вспомогательными***. Редко какая ***программа*** обходится без них, разве что самая простая. [↑](#footnote-ref-11)
12. В том числе, вызывать ***процедуру*** можно и из самой себя. Это ***рекурсия***, высший пилотаж ***программирования***. Рекурсия возможна и для ***подпрограмм*** и ***функций***. [↑](#footnote-ref-12)
13. параметр, стоящий в квадратных скобках, необязателен [↑](#footnote-ref-13)