**Содержание**

[1. Введение 2](#_Toc318292598)

[1.1. Общие сведения 2](#_Toc318292599)

[1.2. Основные этапы решения задач на компьютере 3](#_Toc318292600)

[1.3. Свойства и способы описания алгоритмов 4](#_Toc318292601)

[1.4. Система программирования Турбо Паскаль 5](#_Toc318292602)

[1.5. Диалоговая среда разработчика Турбо Паскаль 6](#_Toc318292603)

[2. Элементы языка Турбо Паскаль 7](#_Toc318292604)

[2.1. Алфавит 7](#_Toc318292605)

[2.2. Идентификаторы 7](#_Toc318292606)

[2.3. Данные в языке Паскаль 7](#_Toc318292607)

[2.3.1. Понятие типа данных в Турбо Паскаль 7](#_Toc318292608)

[2.3.2. Простые типы данных 8](#_Toc318292609)

[2.3.3. Структурированные типы данных 9](#_Toc318292610)

[2.3.4. Указатель (ссылочный тип) 9](#_Toc318292611)

[2.4. Константы 9](#_Toc318292612)

[2.5. Переменные и типы переменных 10](#_Toc318292613)

[2.6. Структура программы на Турбо Паскаль 12](#_Toc318292614)

[3. Выражения 13](#_Toc318292615)

[3.1. Математические операции 14](#_Toc318292616)

[3.2. Логические операции 14](#_Toc318292617)

[3.3. Операции отношения 14](#_Toc318292618)

[3.4. Приоритет операций 15](#_Toc318292619)

[3.5. Основные математические функции (стандартные функции) 15](#_Toc318292620)

[3.6. Примеры 16](#_Toc318292621)

[3.7. Тип выражения 17](#_Toc318292622)

[4. Виды вычислительных алгоритмов 17](#_Toc318292623)

[4.1. Линейный вычислительный процесс 18](#_Toc318292624)

[4.1.1. Оператор присваивания 18](#_Toc318292625)

[4.1.2. Операторы ввода и вывода 19](#_Toc318292626)

[4.1.2.1. Вводданных 19](#_Toc318292627)

[4.1.2.2. Выводданных 19](#_Toc318292628)

[4.1.2.3. Управлениевыводомданных 20](#_Toc318292629)

[4.1.2.4. Выводнапечать 21](#_Toc318292630)

[4.2. Разветвляющийся вычислительный процесс 21](#_Toc318292631)

[4.2.1. Оператор условного перехода 21](#_Toc318292632)

[4.2.2. Оператор выбора 22](#_Toc318292633)

[4.2.3. Оператор безусловного перехода 22](#_Toc318292634)

[4.3. Операторы повторений 23](#_Toc318292635)

[4.3.1. Оператор цикла while-do (цикл с предусловием) 23](#_Toc318292636)

[4.3.2. Оператор цикла repeat-until (цикл с постусловием) 23](#_Toc318292637)

[4.3.3. Оператор цикла for (цикл с параметром) 24](#_Toc318292638)

[5. Массивы 24](#_Toc318292639)

[6. Подпрограммы 26](#_Toc318292640)

[6.1. Понятие подпрограммы 26](#_Toc318292641)

[6.2. Подпрограмма–функция 27](#_Toc318292642)

[6.3. Подпрограмма–процедура 28](#_Toc318292643)

[7. Создание и отладка компьютерных программ 30](#_Toc318292644)

[7.1. Общие сведения. 30](#_Toc318292645)

[7.2. Система программирования Турбо Паскаль 30](#_Toc318292646)

[7.3. Диалоговая среда разработчика Турбо Паскаль 30](#_Toc318292647)

[7.4. Основные этапы создания компьютерной программы. 31](#_Toc318292648)

[7.4.1. Работа в текстовом редакторе среды программирования ТР. 31](#_Toc318292649)

[7.4.2. Сохранение текста программы на диске. 31](#_Toc318292650)

[7.4.3. Поиск ошибок в тексте программы. 31](#_Toc318292651)

[7.4.4. Просмотр результатов выполнения программы: 32](#_Toc318292652)

# Введение

## Общие сведения

Компьютер используется для решения самых разнообразных информационных задач: работа с текстом, создание графических изображений, получение информационной справки из базы данных, табличные расчёты, решение математических задач, расчёт технических конструкций и т.д. Для их решения существует специальное программное обеспечение (ПО): прикладное программное обеспечение (программы, предназначенные для решения узкого класса задач прикладного характера) и системы программирования (средства для создания программ на языках программирования). Пользователь сам решает вопрос о том, каким программным средством воспользоваться. Если в составе прикладного ПО имеется программа, подходящая для решения данной задачи, то пользователь выбирает её (табличный процессор, система управления базами данных, математический или графический пакет и т.д.). В том случае, если готовым прикладным ПО воспользоваться нельзя, то приходится прибегнуть к программированию на алгоритмических языках.

Решение любой задачи начинается с составления алгоритма её решения. Термин происходит от имени арабского математика Мухаммеда бен аль-Хорезми, предложившего в IX веке первые алгоритмы решения задач. *Алгоритм* – это последовательность действий, направленных на решение поставленной задачи.

Последовательность действий (команд, операторов), предназначенная для выполнения компьютером, называется *программой*. Программа для компьютера должна быть написана на языке, понятном ему. У компьютера имеется перечень команд, которые он может понять и выполнить. Такой перечень команд называется системой команд компьютера. *Компьютер* – это электронно-вычислительная машина и систему команд компьютера принято называть языком машинных команд (ЯМК). Программа на ЯМК представляет собой набор двоичных кодов – последовательность нулей и единиц.

Записать алгоритм в виде набора цифр чрезвычайно сложно, поэтому вводится промежуточный этап – разработка текста программы. Совокупность средств и правил представления алгоритма в виде, понятном для компьютера, называется *языком программирования*. Все языки программирования разделяются на две группы – языки высокого и низкого уровней. Языком самого высокого уровня считается человеческий язык. Языком самого *низкого* уровня считается язык машинных кодов. Все остальные алгоритмические языки занимают промежуточное положение.

К языкам *низкого* уровня принадлежат языки семейства *ассемблеров* и *автокод*. Ассемблер фактически состоит из набора команд данной машины, записанных в виде сокращений на английском языке. Автокод – вариант ассемблера на основе русского языка. Программы на этих языках почти полностью понятны компьютеру, почти не требуют перевода и практически сразу выполняются компьютером. Но писать программы на языках низкого уровня может только очень квалифицированный программист. Однако программы, написанные на языках низкого уровня, отличаются более высокой скоростью работы, меньшим объёмом и более полным использованием ресурсов.

Языки *высокого* уровня приближены к человеческому языку, но выполнение алгоритма компьютером тормозится необходимостью предварительного перевода на язык машинных кодов. Примерами таких языков являются Фортран, Паскаль, Бейсик, Си и т.д. программирование на языках высокого уровня проще, чем программирование на языках низкого уровня. Оно не требует глубоких знаний устройства компьютера и доступно неспециалисту.

Создание текста программы на языке программирования выполняется человеком вручную, а перевод текста программы в машинные двоичные коды выполняется специальными компьютерными программами. Для перевода программ с языков высокого уровня на язык машинных команд используются специальные программы, называемые *трансляторами*, а сам процесс перевода называется трансляцией. Трансляторы подразделяются на две большие группы – *интерпретаторы* и *компиляторы*. Компиляторы сначала переводят всю программу в машинные коды и после этого очень быстро исполняют её. Достоинством компиляции является быстрота исполнения, а недостатком – невозможность увидеть промежуточные результаты. Результат выполнения программы мы увидим только после успешной компиляции, а на этапе предварительной подготовки программа-компилятор обычно требует устранить все синтаксические ошибки. Интерпретаторы покомандно переводят программу на язык машинных кодов и тут же исполняют команду. Программа- интерпретатор при обнаружении ошибки прекращает работу и предлагает исправить ошибку. Достоинство интерпретаторов – возможность видеть промежуточные результаты и вносить в исполняемый алгоритм изменения. Недостаток – гораздо более медленная работа по сравнению с компиляторами.

## Основные этапы решения задач на компьютере

Работу по решению задачи на компьютере можно разбить на следующие этапы:

* Постановка задачи;
* Математическая формализация;
* Построение алгоритма;
* Программирование, отладка и тестирование программы;
* Проведение расчётов и анализ результатов;
* Документирование.

**Постановка задачи.** На этапе постановки задачи должно быть определено, что дано и что требуется найти. Постановка задачи включает в себя сбор информации о задаче, формулировку условия задачи, определение конечных целей решения задачи, определение формы выдачи результатов, описание данных (их типов, диапазонов величин, структуры и т.п.). Решение прикладной задачи с помощью компьютера называют моделированием, так как при этом используют упрощённое представление о реальном объекте, процессе или явлении.

**Математическая формализация.** На этом этапе объект или явление описывается с помощью формального языка (формул, уравнений, неравенств ит.д.). Такой процесс называется формализацией. Результатом формализации является создание математической модели задачи. Создавая математическую модель задачи, нужно определить, что считать исходными данными и результатами, а также описать математические соотношения, связывающие результаты с исходными данными.

**Построение алгоритма.** Для реализации математической модели на компьютере создаётся алгоритмическая модель. Разработка алгоритма включает в себя структуризацию, т. е разбиение задачи на последовательность простых модулей, каждый из которых легко может быть реализован на языке программирования.

**Программирование, отладка и тестирование программы.** Программирование – это запись словесного алгоритма, его блок-схемы или псевдокода на выбранном языке программирования. Затем текст программы вводится в компьютер и начинается отладка программы, т.е. процесс испытания работы программы и устранения ошибок и неточностей, допущенных на предыдущих этапах. Обнаружить ошибки, связанные с нарушением правил записи программы на языке программирования (синтаксические и семантические) помогает используемая система программирования. Пользователь получает от системы программирования сообщение об ошибке, исправляет её и снова повторяет попытку исполнить программу. Затем проводится проверка правильности алгоритма с помощью тестов. Тест – это конкретный вариант значений исходных данных, для которых известен ожидаемый результат. На тестах проверяется правильность реализацией программы поставленной задачи.

**Проведение расчётов и анализ результатов.** На этом этапе программа используется для проведения расчётов с необходимыми исходными данными и получение искомого результатов. Производится анализ полученных результатов и в случае необходимости уточнение математической модели, корректировка алгоритма и программы.

**Документирование.** Завершающим этапом работы по решению задачи является документирование – т.е. распечатка текста программы, снабжённого необходимыми комментариями автора (листинг).

## Свойства и способы описания алгоритмов

Алгоритмам должен обладать следующими свойствами:

* *Однозначность* – чёткое предписание, что и как делать в каждой конкретной ситуации.
* *Универсальность* – применимость данного алгоритма к решению любой задачи данного типа.
* *Результативность* – отсутствие зацикливаний (образование замкнутого круга, из которого можно выйти только принудительным прерыванием работы программы). Любая программа должна всегда приводить к результату, даже если это будет аварийное сообщение.

К основным способам описания алгоритмов можно отнести следующие:

* словесно-формульный;
* структурный или блок-схемный.

При *словесно-формульном* способе алгоритм записывается в виде текста с формулами по пунктам, определяющим последовательность действий.

Пусть, например, необходимо найти значение следующего выражения:

у = 2а - (х+6).

Словесно-формульным способом алгоритм решения этой задачи может быть записан в следующем виде:

* Ввести значения а и х.
* Сложить х и 6.
* Умножить а на 2.
* Вычесть из 2а сумму (х+6).
* Вывести y как результат вычисления выражения.

При *блок-схемном* описании алгоритм изображается геометрическими фигурами (блоками), связанными по управлению линиями (направлениями потока) со стрелками. В блоках записывается последовательность действий. Графическая интерпретация алгоритма называется блок-схемой. Данный способ по сравнению с другими способами записи алгоритма имеет ряд преимуществ. Он наиболее нагляден: каждая операция вычислительного процесса изображается отдельной геометрической фигурой. Кроме того, графическое изображение алгоритма наглядно показывает разветвления путей решения задачи в зависимости от различных условий, повторение отдельных этапов вычислительного процесса и другие детали.

Оформление программ должно соответствовать определенным требованиям. В настоящее время действует единая система программной документации (ЕСПД), которая устанавливает правила разработки, оформления программ и программной документации. В ЕСПД определены и правила оформления блок-схем алгоритмов (ГОСТ 10.002-80 ЕСПД, ГОСТ 10.003-80 ЕСПД).

Операции обработки данных и носители информации изображаются на схеме соответствующими блоками. Большая часть блоков по построению условно вписана в прямоугольник со сторонами **a** и **b**. Минимальное значение **а** = 10 мм, увеличение **а** производится на число, кратное 5 мм. Размер **b**=1,5**а**. Для отдельных блоков допускается соотношение между **а** и **b**, равное 1:2. В пределах одной схемы рекомендуется изображать блоки одинаковых размеров. Все блоки нумеруются. Виды и назначение основных блоков приведены в таблице.

Таблица 1.1. Условные обозначения блоков схем алгоритмов

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Обозначение** | **Функции** |
| Процесс |  | Выполнение операции или группы операций, в результате которых изменяется значение, форма представления или расположение данных. |
| Ввод-вывод |  | Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод). |
| Решение |  | Выбор направления выполнения алгоритма в зависимости от некоторых переменных условий. |
| Предопределенный процесс |  | Использование ранее созданных и отдельно написанных программ (подпрограмм). |
| Документ |  | Вывод данных на бумажный носитель. |
| Магнитный диск |  | Ввод-вывод данных, носителем которых служит магнитный диск. |
| Пуск-останов |  | Начало, конец, прерывание процесса обработки данных. |
| Соединитель |  | Указание связи между прерванными линиями,  соединяющими блоки. |
| Межстраничный соединитель |  | Указание связи между прерванными линиями, соединяющими блоки, расположенные на разных листах. |
| Комментарий | [ | Связь между элементом схемы и пояснением. |

Линии, соединяющие блоки и указывающие последовательность связей между ними, должны проводится параллельно линиям рамки. Стрелка в конце линии может не ставиться, если линия направлена слева направо или сверху вниз. В блок может входить несколько линий, то есть блок может являться преемником любого числа блоков. Из блока (кроме логического) может выходить только одна линия. Логический блок может иметь в качестве продолжения один из двух блоков, и из него выходят две линии. Если на схеме имеет место слияние линий, то место пересечения выделяется точкой. В случае, когда одна линия подходит к другой и слияние их явно выражено, точку можно не ставить.

Схему алгоритма следует выполнять как единое целое, однако в случае необходимости допускается обрывать линии, соединяющие блоки.

Если при обрыве линии продолжение схемы находится на этом же листе, то на одном и другом конце линии изображается специальный символ соединитель – окружность диаметром 0,5 а. Внутри парных окружностей указывается один и тот же идентификатор. В качестве идентификатора, как правило, используется порядковый номер блока, к которому направлена соединительная линия.

Если схема занимает более одного листа, то в случае разрыва линии вместо окружности используется межстраничный соединитель. Внутри каждого, соединителя указывается адрес — откуда и куда направлена соединительная линия. Адрес записывается в две строки: первой указывается номер листа, во второй — порядковый номер блока.

Блок-схема должна содержать все разветвления, циклы и обращения к подпрограммам, содержащиеся в программе.

## Система программирования Турбо Паскаль

Система программирования Турбо Паскаль (Turbo Pascal) представляет собой сочетание компилятора с языка программирования Паскаль и программной оболочки, помогающей в создании программ. В названии языка Турбо – это торговая марка разработчика системы Турбо Паскаль фирмы Borland International, Inc (США), а Паскаль – название широко распространённого языка программирования (язык назван в честь выдающегося французского математика и философа Блеза Паскаля). Программной оболочкой предоставляется диалоговая среда, которая способствует повышению эффективности создания программ. Эту диалоговую среду называют также средой разработчика Турбо Паскаля.

Система программирования Турбо Паскаль является одной из наиболее популярных систем программирования. Это объясняется сочетанием двух безусловных её достоинств: исключительной простотой и естественностью языка программирования Паскаль и хорошими сервисными возможностями диалоговой среды программирования фирмы Borland. С помощью Турбо Паскаля можно создавать любые программы – от программ, предназначенных для решения простейших вычислительных задач, до сложных современных систем управления базами данных и операционных систем.

## Диалоговая среда разработчика Турбо Паскаль

Вход в диалоговую среду осуществляется с помощью ярлыка Турбо Паскаль рабочего стола Windows. После вызова системы на экране компьютера откроется диалоговое окно среды разработчика Турбо Паскаля. Для управления средой разработчика Турбо Паскаля используются меню и функциональные клавиши. Верхняя строка открывшегося окна содержит меню возможных режимов работы, а нижняя – краткую справку о назначении основных функциональных клавиш. Вся остальная часть экрана является окном редактора и предназначена для ввода и корректировки текста программ. В его верхней строке приводится имя того дискового файла, откуда был прочитан текст программы, или имя NONAME00.PAS, если текст не читался с диска.

В Турбо Паскале можно работать одновременно с несколькими программами (или частями одной программы), каждая из которых может располагаться в отдельном окне редактора. Номер окна указывается справа в строке имени файла. Кроме окна редактора, используются также окна: отладочного режима, вывода результатов работы программы, справочной службы, стека, регистров. По желанию они могут вызываться на экран поочерёдно или присутствовать на нём одновременно.

С каждым пунктом меню или функциональной клавишей связывается некоторая команда, управляющая средой. Функциональные клавиши обозначаются F1,F2, …, F12 и располагаются в самом верхнем ряду клавиатуры. Действие функциональных клавиш можно модифицировать одновременным нажатием особых клавиш: ALT, CTRL, SHIFT и т.д. Наиболее часто используемые функциональные клавиши и их сочетание называют «горячими клавишами». Ниже приведены основные команды среды разработчика Турбо Паскаль и соответствующие им горячие клавиши (знак «+» между клавишами означает одновременное нажатие клавиш).

* **F10** – вход в меню
* **Ctrl+F9 –** запуск программы
* **Alt+F5** – просмотр пользовательского экрана
* **F2** – сохранение программы
* **F3** – открытие сохраненной программы
* **Alt+F3** – закрытие активного окна
* **Alt+X** – выход из Турбо Паскаль
* **F1** – контекстная помощь
* **Ctrl+F1** – справка об операторе, на котором установлен курсор
* **Alt+Backspace** – отмена последнего изменения
* **Ctrl+Y** – удаление строки
* **Shift+стрелки** – выделение блока текста
* **Ctrl+Insert** – копирование выделенного блока в буфер
* **Shift+Insert** – вставка из буфера

**Последовательность работы с программой в среде Турбо Паскаль:**

1. *Вход в среду ТР:*

Открыть ярлык TurboPascal на рабочем столе.

1. *Организация нового файла:*

**File⇒New**

При этом откроется новый файл со стандартным именем NONAME00.PAS.

1. *Изменение имени диска или текущей директории:*

**File⇒Chang dir…**⇒выбрать диск или папку⇒**OK**

1. *Открытие существующего файла:*

**File⇒Open** (или нажать клавишу **F3**)

1. *Запись файла на диск с новым именем:*

**File⇒Save as…**⇒выбрать путь к папке, в которой хотите сохранить файл⇒вписать имя файла вместо \*⇒**OK**

1. *Сохранение файла на диске со старым именем (дозапись):*

**File⇒Save** (или нажать клавишу **F2**)

1. *Запуск программы на исполнение:*

**Run⇒Run** (или одновременно нажать клавиши **Ctrl+F9**)

1. *Просмотр результатов:*

**Debug⇒User Screen** (или одновременно нажать клавиши **Alt+F5**)

1. *Уменьшение или увеличение размера окна:*

Одновременно нажать клавиши **Alt+Enter**

1. *Копирование результатов в окно Word (для составления отчёта):*

Уменьшить окно⇒в контекстном меню заголовка окна выбрать **Свойства**⇒установить флажок **∨ Выделение мышью**⇒**ОК**⇒**ОК**⇒выделить мышью текст результата расчёта⇒ в контекстном меню заголовка окна выбрать **Изменить**⇒**Копировать** ⇒открыть программу MS Word⇒**Правка**⇒**Вставить**

1. *Выход из среды ТР:*

**File⇒Exit** (или одновременно нажать клавиши **<Alt>+<X>)**

# Элементы языка Турбо Паскаль

## Алфавит

Алфавит языка Турбо Паскаль включает в себя буквы, цифры, шестнадцатеричные цифры, специальные символы и зарезервированные служебные слова. *Буквы* – это буквы латинского алфавита от a до z и от A до Z. В Турбо Паскале нет различия между строчными и прописными буквами алфавита. *Цифры* – арабские цифры от 0 до 9. *Шестнадцатеричные цифры* – от 0 до 9, остальные шесть обозначаются латинскими буквами от A до F или от a до f. *Специальные знаки* – это символы +, - , \* , / , = , ‘ , . , : , ; , < , > , [ , ] , ( , ) , { , } , ^ , @ , $ , #, а также пары символов: <>, <=, >=, := , (\* , \*), (..). Особое место занимает символ *пробела*, который рассматривается как ограничитель при написании текста программы.

К *зарезервированным словам* относятся: begin end const var for if и т.д Познакомиться с зарезервированными словами мы сможем в процессе изучения языка.

## Идентификаторы

Любая программа выполняет над исходными данными некоторые расчеты. Отдельные элементы данных обозначаются даваемыми программистом именами (***идентификаторами***). Любые идентификаторы в языке Паскаль строятся по следующим правилам:

* имена могут включать латинские буквы, цифры и знак подчеркивания (для простоты опустим некоторые другие символы, разрешенные в именах);
* имя состоит из одного слова; если требуется пробел в имени, он заменяется на подчеркивание: так, **My\_1**  будет правильным идентификатором, а **My 1** – нет;
* имя всегда начинается с буквы: допуст**и**м объект с именем **A1**, но не **1A**; прописные и строчные буквы в именах не различаются Паскалем: **x1** и **X1** – это одна и та же величина;
* имена не могут совпадать с зарезервированными служебными словами.

## Данные в языке Паскаль

#### Понятие типа данных в Турбо Паскаль

Для обработки ЭВМ данные представляются в виде величин и их совокупностей. Это могут быть константы или переменные. Каждый из этих элементов программы характеризуется своим значением и принадлежит к какому-либо типу данных. Тип определяет:

* возможные значения переменных, констант, функций, выражений, принадлежащих к данному типу;
* внутреннюю форму представления данных в ЭВМ;
* операции и функции, которые могут выполняться над величинами, принадлежащими к данному типу.

В языке Паскаль тип величины задают заранее. Все элементы, используемые в программе, должны быть объявлены с указанием их типа. Обязательное описание типа приводит к избыточности в тексте программ, но такая избыточность является важным вспомогательным средством разработки программ и рассматривается как необходимое свойство современных алгоритмических языков высокого уровня.

Иерархия типов в языке Паскаль такая:

* Простые
  + Порядковые
    - Целые
    - Логические
    - Символьные
    - Перечисляемые
    - Интервальные
  + Вещественные
* Структурированные
  + Массивы
  + Строки
  + Множества
  + Записи
  + Файлы
* Указатели

#### Простые типы данных

К простым типам относятся порядковые и вещественные типы. *Порядковые* типы отличаются тем, что каждый из них имеет конечное число возможных значений. Эти значения можно упорядочить и, следовательно, с каждым из них можно сопоставить некоторое целое число – порядковый номер значения. *Вещественные* типы не имеют точного количества возможных значений, хотя конечное значение определяется форматом внутреннего представления числа. Количество возможных значений настолько велико, что сопоставить с каждым из них целое число (его номер) не представляется возможным.

К любому из порядковых типов применимы функции:  
**ORD(X)** – возвращает порядковый номер значения X,  
**PRED(X)** – возвращает предыдущее значение порядкового типа,  
**SUCC(X)** – возвращает следующее значение порядкового типа.

В таблице приведены простые типы данных Турбо Паскаль, объем памяти, необходимый для хранения одной переменной указанного типа, множество допустимых значений и применимые операции.

Таблица 0.2

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Длина (байт) | Диапазон значений | Операции |
| **Целые типы** | | | |
| integer | 2 | -32768..32767 | +, -, /, \*, Div, Mod, >=, <=, =, <>, <, > |
| byte | 1 | 0..255 | +, -, /, \*, Div, Mod, >=, <=, =, <>, <, > |
| word | 2 | 0..65535 | +, -, /, \*, Div, Mod, >=, <=, =, <>, <, > |
| shortint | 1 | -128..127 | +, -, /, \*, Div, Mod, >=, <=, =, <>, <, > |
| longint | 4 | -2147483648..2147483647 | +, -, /, \*, Div, Mod, >=, <=, =, <>, <, > |
| **Вещественные типы** | | | |
| real | 6 | 2,9x10-39 - 1,7x1038 | +, -, /, \*, >=, <=, =, <>, <, > |
| single | 4 | 1,5x10-45 - 3,4x1038 | +, -, /, \*, >=, <=, =, <>, <, > |
| double | 8 | 5x10-324 - 1,7x10308 | +, -, /, \*, >=, <=, =, <>, <, > |
| extended | 10 | 3,4x10-4932 - 1,1x104932 | +, -, /, \*, >=, <=, =, <>, <, > |
| **Логический тип** | | | |
| boolean | 1 | true, false | Not, And, Or, Xor, >=, <=, =, <>, <, > |
| **Символьный тип** | | | |
| char | 1 | все символы кода ASCII | +, >=, <=, =, <>, <, > |

Примеры описания типов: A : Real; B : Integer; C : Longint;

**Перечисляемый тип** задаётся перечислением тех значений, которые он может получать. Каждое значение именуется идентификатором и располагается в списке. Например,

TypeMonth=(янв, фев, мар);

Номера значений соответствуют порядку перечисления и должны быть не более 65535 (т.е. тип Word).

**Интервальный тип (тип-диапозон)** – это подмножество значений своего базового типа, в качестве которого может выступать любой порядковый тип, кроме типа-диапозона. Задаётся границами своих значений внутри базового типа:

<минимальное значение>.. <максимальное значение>

Например, date=1..31  
 digit=’0’..’9’

#### Структурированные типы данных

Структурированные типы характеризуются множественностью образующих этот тип элементов, т.е. имеют несколько компонентов. Каждый компонент , в свою очередь может принадлежать структурированному типу, т.е. допускается вложенность типов.

**Массивы** представляют собой формальное объединение нескольких однотипных объектов (чисел, символов, строк и т.п.), рассматриваемое как единое целое. Все компоненты массива – это данные одного типа.

Общий вид определения массива:

Type A = array [тип индекса массива] of [тип компонент массива]

Например, М1=array [1..100] of real;

**Строки** – это массив символов, но количество символов в строке может меняться. Строка трактуется как цепочка символов произвольной длины. Максимальное количество символов не более 255. Каждый символ в строке имеет свой индекс (номер).

Line = string[80];

**Запись** – это структура данных, состоящая из фиксированного числа компонентов, называемых полями записи. В отличие от массива, компоненты записи (поля) могут быть различного типа. Записи позволяют объединять значения различных типов.

Date = record

Month : (Jan, Feb, Mar, Apr, May, Jun, July, Aug, Sept, Oct, Nov, Dec);

Day : 1..31;

Year : 2000..2050;

End;

**Множества** – это наборы однотипных, логически связанных друг с другом объектов. Количество элементов, входящих в множество может меняться от 0 до 256. Именно непостоянством своих элементов множества отличаются от массивов и записей.

Digits = Set of 1..5;

**Файл** – именованная область внешней памяти. Файл содержит компоненты одного типа, кроме файлов ( т.е. нельзя создать «файл файлов»). Длина файла не оговаривается и ограничивается только ёмкостью устройств внешней памяти.

F : File of Integer;

Более подробно со структурированными типами ознакомимся при дальнейшем изучении языка.

#### Указатель (ссылочный тип)

Cодержит адрес байта памяти, в котором находится значение данных определённого типа. Этот тип называют также ссылочным. Для описания используется символ ^ и идентификатор типа. Например, P=^integer;

Использование указателей является гибким средством управления динамической памятью и предоставляет возможность обработки массивов данных большой размерности.

## Константы

**Константой** называют величину, значение которой не меняется в процессе выполнения программы.

* **Числовые** константы служат для записи чисел. Различают следующие их виды:

***Целые*** числа: записываются со знаком + или -, или без знака, по обычным арифметическим правилам: -10      +5       5

***Вещественные*** числа могут записываться в одной из двух форм:

*обычная запись:*     2.5      -3.14         2.       - обратите внимание, что целая часть отделяется от дробной символом точки;

*экспоненциальная*форма: в этой записи вещественное число представляется в виде m\*10p, где m – *мантисса* или основание числа, 0.1≤|m|≤1, p – *порядок* числа, это целочисленная константа. Действительно, любое вещественное число можно представить в экспоненциальной форме:

-153.5        -0.1535\*103

99.005        0.99005\*102

Во всех IBM-совместимых компьютерах вещественные числа хранятся как совокупность мантиссы и порядка, что позволяет упростить операции над ними, используя специальную арифметику, отдельно обрабатывающую мантиссу и порядок. Для программной записи числа в экспоненциальной форме вместо "умножить на 10 в степени" используется обозначение **E** или **e** (латинская):

-153.5        -0.1535\*103        -0.1535E3      или  -1.535E02

99.005        0.99005\*102        0.99005E+2    или  9.9005e+01

Без принятия специальных мер программа на Паскале будет выводить на экран и принтер вещественные числа именно в такой форме. Кроме того, такая форма удобна для записи очень маленьких и очень больших чисел:

1030      1e30

-1020     -1E20

10-30      1E-30

Поскольку размер памяти, отводимой под мантиссу и порядок, ограничен, то **вещественные числа всегда представляются в памяти компьютера с некоторой погрешностью**. Например, простейшая вещественная дробь 2/3 дает в десятичном представлении 0,666666... и, независимо от размера памяти, выделяемой для хранения числа, невозможно хранить *все* его знаки в дробной части. Одной из типичных проблем программирования является учет возможных погрешностей при работе с вещественными числами.

Шестнадцатеричные числа состоит из шестнадцатеричных цифр, которым предшествует знак $. Диапазон шестнадцатеричных чисел – от $00000000 до $FFFFFFFF.

Кроме числовых констант существуют и другие их виды:

* **Логические** константы.

Служат для проверки истинности или ложности некоторых условий в программе и могут принимать только **одно из двух значений**: служебное слово **true** обозначает истину, а **false** – ложь;

* **Символьные** константы.

Могут принимать значение любого печатаемого символа и записываются как символ, заключенный в *апострофы* ('одинарные кавычки'):

'Y'  'я'  ' '.

В последнем случае значение символьной константы равно символу пробела. Если требуется записать сам символ апострофа как символьную константу, внутри внешних апострофов он удваивается: ''''

К символьным также относятся константы вида #X, где X – числовое значение от 0 до 255 включительно, представляющее собой десятичный *ASCII-код* символа. Таблицы ASCII-кодов, используемых операционными системами DOS и Windows, приведены в Приложении  1. Например,  значение #65 будет соответствовать коду символа 'A' латинской.

* **Строковые** константы.

Это любые последовательности символов, заключенных в апострофы. Как правило, строковые константы служат для записи приглашений к вводу данных, выдаваемых программой, вывода диагностических сообщений и т.п.:

'Введите значение X:'

'Ответ='

Если в строковой константе необходимо записать сам символ апострофа, это делается так же, как для символьных констант.

Константы в Турбо Паскале могут быть именованными. *Неименованные* константы используются, например, при выводе текста сообщений в предыдущем примере. *Именованные константы* описываются в разделе описаний программы оператором следующего вида:

const Имя1=Значение1;

 Имя2=Значение2;

 . . .

 ИмяN=ЗначениеN;

Здесь ключевое слово const показывает начало раздела описаний именованных констант. Ясно, что зачастую удобнее обращаться к константе по имени, чем каждый раз переписывать ее числовое или строковое значение. Пример раздела констант:

const e=2.7182818285;

  lang='Turbo Pascal 7.1';

Здесь описана числовая константа e со значением основания натурального логарифма и строковая константа с именем lang, содержащая строку 'Turbo Pascal 7.1'.

Каждое даваемое программистом имя должно быть **уникальным** в пределах одной программы. Если мы включим этот раздел в свою программу, мы уже не сможем создать в ней других объектов с именами e и lang.

## Переменные и типы переменных

**Переменными** называют величины, значения которых **могут изменяться** в процессе выполнения программы. Каждая переменная задается своим уникальным именем, построенным по правилам, указанным в начале раздела. Максимально возможная длина имени зависит от реализации Паскаля, теоретически можно давать переменным имена вплоть до 63 символов длиной, что едва ли актуально – обычно имена не длиннее 5-10 символов.

Поскольку любые данные в памяти компьютера хранятся в числовой форме и двоичной системе счисления, кроме имени, переменной обязательно следует присвоить и **тип**, определяющий **диапазон значений**, принимаемых переменной, и **способ ее обработки** машиной. Поясним сказанное на примере. Как видно из Приложения 1, латинская большая буква 'A' имеет десятичный код 65, или 01000001 в двоичном представлении. Без дополнительной информации о *типе* данных, хранящихся в некоторой ячейке памяти, компьютеру было бы невозможно решить, что именно представляют из себя эти данные – число 65, код символа 'A' или что-то еще. В любом языке программирования, в том числе и в Паскале, существует стандартный набор типов, к которым может быть отнесена та или иная совокупность ячеек памяти. Информацию о типах данных Паскаля удобно свести в таблицу. Строки этой таблицы будут упорядочены по **старшинству типов**, от самого "младшего", требующего наименьшее число байт для представления, и, соответственно, представляющего наименьший диапазон возможных значений, до самого "старшего", представляющего наибольший диапазон значений. В таблице представлены не все возможные, а лишь основные типы данных Паскаля.

Таблица 1.3

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое слово Паскаля** | **Название и описание типа** | **Объем**  **памяти,**  **байт** | **Диапазон возможных значений** |
| Boolean | Логический: хранит одну логическую переменную | 1 | true и false |
| Char | Символьный: хранит код одного  символа из набора ASCII-кодов | 1 | от 0 до 255 включительно (28=256) |
| Integer | Целочисленный | 2 | ±215 |
| Word | Целочисленный без знака | 2 | ±216 - диапазон вдвое больше, так как 16-й бит не занят под знак числа |
| Longint | Длинное целое: для представления больших целочисленных значений | 4 | ±231 |
| Real | Вещественное число с точностью представления до 11-12 знака в дробной части | 6 | ~ 2.9\*10-39 - 1.7\*1038 |
| Double | Вещественное число с точностью представления до 15-16 знака в дробной части | 8 | ~ 5\*10-324 – 1.7\*10308 |
| String | Последовательность символов типа Char длиной от 1 до 255 | 2-256  (данные строки  + 1 байт для хране-ния её длины) | Любые строки текста, состоящие из печатаемых символов |

Теоретически для записи переменной типа Boolean было бы достаточно 1 бита, но минимальная адресуемая единица памяти – 1 байт

Целочисленные и символьные типы обобщенно называют **порядковыми**, подчеркивая этим, что данные типы имеют конечный набор значений, которые могут быть упорядочены или перечислены. Напомним, что вещественные значения хранятся в памяти компьютера иначе, чем целые – а именно, как совокупность мантиссы и порядка.

Задача правильного выбора типов данных целиком ложится на программиста. Например, если некоторый счетчик в Вашей программе может принимать целочисленные значения от 1 до 100000, неправильно было бы описывать его как переменную типа Integer – ведь 215=32768 и при достижении счетчиком этой величины произойдет *сброс* его значения, которое станет равно -32768. Разумным в данном случае было бы описание счетчика как переменной типа Longint.

 Переменные описываются в программе оператором следующего вида:

var  Список1:Тип1;

     Список2:Тип2;

     . . .

     СписокN:ТипN;

Здесь *список* – набор имен переменных, разделенных запятыми (или одна переменная), а *тип* – любой из рассмотренных выше типов данных. Например, конструкция

var t,r:real;

 i:integer;

описывает 2 вещественных переменных с именами t и r, а также целочисленную переменную с именем i. Ключевое слово var можно и продублировать, но обычно такой необходимости нет. Сокращение var образовано от английского слова "Variable" (переменная).

## Структура программы на Турбо Паскаль

Программа на языке Турбо Паскаль состоит из *заголовка*, *раздела* *описаний* и *раздела* *исполняемых* *операторов*. ***Заголовок*** программы содержит имя программы, например:

Program PRIM;

Строка заканчивается особым разделителем – точкой с запятой. Этот разделитель в языке Турбо Паскаль отмечает конец оператора или описания. Использование особого разделителя позволяет располагать несколько операторов на одной строке. Поскольку имя программы в дальнейшем не используется, то можно опускать строку объявления имени без каких-либо последствий для программы.

***Раздел описания*** долженсодержать описания всех идентификаторов, используемых в разделе исполняемых операторов.

Описания могут включать в себя:

* раздел подключаемых библиотек (модулей);
* раздел описания меток;
* раздел описания констант;
* раздел описания типов;
* раздел описания переменных;
* раздел описания процедур и функций.

**Раздел описания модулей** определяется служебным словом USES и содержит имена подключаемых модулей (библиотек) как входящих в состав системы Turbo Pascal, так и написанных пользователем. Раздел описания модулей должен быть первым среди разделов описаний. Имена модулей отделяются друг от друга запятыми:

uses CRT, Graph;

Любой оператор в программе может быть помечен меткой. Имя метки задается по правилам образования идентификаторов Турбо Паскаль. В качестве метки также могут использоваться произвольные целые числа без знака, содержащие не более четырех цифр. Метка ставится перед оператором и отделяется от него двоеточием. Все метки, используемые в программе, должны быть перечислены в разделе описания меток и начинаться со служебного слова LABEL, например:

label 3, 471, 29, Quit;

Описание констант позволяет использовать имена как синонимы констант, их необходимо определить в разделе описания констант. Описанию должно предшествовать служебное слово CONST:

const K= 1024; MAX= 16384;

В **разделе описания типов** объявляются идентификаторы типов. Описанию должно предшествовать служебное слово TYPE. Например:

Type Str = string[40];

В **разделе описания переменных** необходимо указать все переменные, используемые в программе, и определить их тип. Описание начинается со служебного слова VAR:

var P,Q,R: Integer;

A,B: Char;

F1,F2: Boolean;

S: Str;

Описание типов, процедур и функций будет рассмотрено ниже. Отдельные разделы описаний могут отсутствовать, но следует помнить, что в Турбо Паскаль - программе должны быть обязательно описаны все компоненты программы.

**Раздел исполняемых операторов** представляет собой составной оператор, который содержит между служебными словами

begin.......end

последовательность операторов. Операторы отделяются друг от друга символом **;**. Текст программы заканчивается символом точка.

Кроме описаний и операторов Турбо Паскаль, программа может содержать комментарии, которые представляют собой произвольную последовательность символов, расположенную между открывающей скобкой комментариев **{** и закрывающей скобкой комментариев **}**.

*Пример 1*

program Primer; {вычисление суммы двух чисел}

var

x,y,s: integer;

begin

WriteLn('Введите через пробел два числа ');

ReadLn(x,y);

s := x + y;

WriteLn('Сумма чисел равна ',s);

end.

Данная программа запрашивает с клавиатуры два числа, находит их сумму и выводит ответ. Теперь сделаем так, чтобы программа сначала очищала экран, выполняла свои действия, а в конце работы позволяла пользователю посмотреть результат, ожидая его нажатия клавиши.

*Пример 2*

program Primer; {вычисление суммы двух чисел}

uses Crt; {подключение модуля, необходимого для процедур

очистки экрана и задержки}

var

x,y,s: integer;

begin

ClrScr; {очистка экрана}

WriteLn('Введите через пробел два числа ');

ReadLn(x,y);

s := x + y;

WriteLn('Сумма чисел равна ',s);

ReadKey; {ожидание нажатия клавиши}

end.

# Выражения

Выражение задает правило вычисления некоторого значения. В общем случае выражение состоит из нескольких элементов (*операндов*) и знаков *операций (операторов)*, а тип определяется типом операндов и видом применённых к ним операций. Операндами могут быть константы, переменные и обращения к функциям. Каждый из этих элементов характеризуется своим значением и принадлежит к какому-либо типу данных.

Все операции можно разделить на 4 группы:  
 - математически операции;  
 - операции отношения;  
 - логические операции;  
 - функции.

## Математические операции

В таблице приведены основные математические операции Турбо Паскаль в соответствии с *приоритетом*, то есть, старшинством. Операции с одинаковым приоритетом равноправны между собой и выполняются слева направо, как и в математике.

. Таблица 1.4

|  |  |  |
| --- | --- | --- |
| **Символ операции** | **Название операции** | **Пример** |
| \* | умножение | 2\*3 (результат: 6) |
| / | деление | 30/2 (результат: 1.5E+01) |
| div | целочисленное деление | 5 div 2 (результат: 2) |
| mod | остаток от деления | 5 mod 2 (результат: 1) |
| + | сложение | 2+3 (результат: 5) |
| - | вычитание | 5-3 (результат: 2) |

Операции div и mod определены только для целочисленных операндов. Приведем пример их использования:

var y,c,n:integer;

. . .

y:=2006;

c:=y div 100;

n:=y mod 100;

Здесь переменная **c** получит значение 20, а **n** – значение 6.

## Логические операции

Над логическими аргументами в Турбо Паскаль определены следующие операции:

* NOT - логическое отрицание ("НЕ")
* AND - логическое умножение ("И")
* OR - логическое сложение ("ИЛИ")
* XOR - логическое "Исключающее ИЛИ"

Результаты выполнения этих операций над переменными А и В логического типа приведены в таблице истинности.

Таблица 1.5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **not A** | **A and B** | **A or B** | **A xor B** |
| true | True | false | true | true | false |
| true | False |  | false | true | true |
| false | True | true | false | true | true |
| false | false |  | false | false | false |

## Операции отношения

К операциям отношения в Турбо Паскаль относятся такие операции, как:

* **>** - больше
* **<** - меньше
* **=** - равно
* **<>** - не равно
* **>=** - больше или равно
* **<=** - меньше или равно

В операциях отношения могут принимать участие не только числа, но и символы, строки, множества и указатели.

## Приоритет операций

Порядок вычисления выражения определяется старшинством (приоритетом) содержащихся в нем операций. В языке Паскаль принят следующий приоритет операций:

* унарная операция not, унарный минус -, взятие адреса @
* операции типа умножения:  \* , /,  div,  mod,  and
* операции типа сложения:  +,  -,  or,  xor
* операции отношения:  =,  <>,  < , >,  <=,  >=,  in

Порядок выполнения операций переопределить можно с помощью скобок. Например, 2\*5+10 равно 20, но 2\*(5+10) равно 30.

## Основные математические функции (стандартные функции)

В этом разделе приведены основные математические функции, встроенные в системную библиотеку Турбо Паскаль. **Стандартные функции** служат для выполнения элементарных математических расчетов, часто требуемых при написании программ. Разработчики Паскаля стремились сделать его программное ядро максимально компактным, поэтому в него не вошел ряд функций, обычно имеющихся в других языках, таких, как вычисление максимума и минимума, возведение числа в произвольную степень и др. Физически коды стандартных функций хранятся в *стандартной библиотеке* Паскаля – файле с именем TURBO.TPL. Все функции оформляются одинаково: после имени функции следует ее аргумент, заключенный **в круглые скобки**. Если аргументов несколько, они разделяются **запятыми**. Информацию об основных стандартных функциях удобно представить в виде таблицы:

Таблица 1.6

|  |  |  |  |
| --- | --- | --- | --- |
| Математическая запись | Запись на языке Турбо Паскаль | Пояснение | Тип  аргумента и результата функции |
| |*x*| | abs(x) | модуль аргумента x | Integer или Real |
| *x*2 | sqr(x) | квадрат аргумента x | аргумент – I или R, результат – R |
| sin *x*  cos *x*  arctg *x* | sin(x)  cos(x)  arctan(x) | Остальные тригонометрические функции выражаются через эти | аргумент – I или R, результат – R |
| *ex*  ln *x* | exp(x)  ln(x) | экспонента и натуральный логарифм | аргумент - I или R, результат – R |
| image003 | sqrt(x) | квадратный корень от аргумента x | аргумент - I или R, результат – R |
|  | pi | функция без аргументов, вернет число  | R |
|  | trunc(x) | функция отбрасывает дробную часть аргумента, аргумент не округляется | аргумент R, результат L |
|  | frac(x) | функция выделяет  дробную часть своего  аргумента | R |
|  | round (x) | округление вещественного числа до ближайшего целого | аргумент R, результат L |
|  | Int(X) | возвращает число, равное целой части числа аргумента. | R |
|  | Random(X) | Возвращает случайное целое число в диапазоне 0..X. Если аргумент опущен (Random), то возвращается случайное вещественное число от 0 до 1. | результат I, если аргумент I;  результат R, если аргумент опущен |
|  | Inc(X,Y) | Увеличивает значение числа X на Y. Если число Y не указано, то увеличение происходит на 1. | I |
|  | Dec(X,Y) | Уменьшает значение числа X на Y. Если число Y не указано, то уменьшение происходит на 1. | I |

Здесь x обозначает любую подходящую по типу переменную, либо результат вычисления выражения соответствующего типа (см. ниже), либо соответствующий по типу результат, вычисленный другой стандартной функцией. Функция pi не имеет аргументов и возвращает число π.

Функции возведения в произвольную степень в Турбо Паскале нет. Используйте многократное умножение для возведения в целочисленную степень, либо функции Exp и Ln для возведения в вещественную степень.

Перед использованием Random в программах рекомендуется сначала инициализировать генератор псевдослучайных чисел процедурой Randomize. В противном случае при каждом запуске программы будет генерироваться одна и та же последовательность случайных чисел.

*Пример*. Вывод на экран 5 случайных чисел в диапазоне -10..10.

var i: integer;

begin

randomize;

for i:=1 to 5 do writeln(random(21)-10);

end.

## Примеры

*1) возвести x в пятую степень*

x\*x\*x\*x\*x или  sqr(x)\*sqr(x)\*x или sqr(sqr(x))\*x, последнее показывает, что результаты одних функций могут быть аргументами других – это называют **вложением функций**. Разумеется, тип результата, возвращаемый вложенной функцией, должен быть подходящим для аргумента внешней функции.

*2) возвести величину a в произвольную степень x*

Так как в Паскале нет функции возведения в произвольную степень, воспользуемся формулой **ax=ex\*ln a**

a:=2.5; x:=0.25;

ax:=exp(x\*ln(a));

обратите внимание, что все скобки в выражении должны быть парными. Или:

image004 = exp(1/3ln(x)).

*3) вычислить sin2x* => sqr(sin(x)).

Нельзя писать sin\*x или sin x, после имени функции может следовать только аргумент в круглых скобках.

4) *вычислить k=tg(t)*. Т.к. функции tg нет, пишем k:=sin(t)/cos(t);

http://pers.narod.ru/study/pascal.files/image005.gif

5) *вычислить:*

При необходимости изменить обычное старшинство операций в записи выражения используются круглые скобки.

Правильная запись: y:=(a+b)/2; Неправильно y:=a+b/2, т.к. это означает .



6) В записи выражений нельзя пропускать знак \*, как часто делается в математике.

b2-4ac   -> sqr(b)-4\*a\*c

## Тип выражения

Турбо Паскаль построен на основе строгого соблюдения концепции типов, в соответствии с которой все применяемые в языке операции определены только для операндов совместимых типов. Поэтому для всех операций оговариваются допустимые типы операндов и тип получаемого результата.

В программе данные одного типа могут преобразовываться в данные другого типа. Такое преобразование может быть явным и неявным.   
При *явном* преобразовании типов используются вызовы специальных функций преобразования, аргументы которых принадлежат одному типу, а значение – другому. Например, это функции ORD, TRUNC, ROUND и т.д.   
*Неявное* преобразование типов возможно в выражениях, составленных из вещественных и целочисленных переменных. Тип выражения в этом случае определяется старшим из типов входящих в него операндов (т.е. стандартных функций, переменных, констант).

**Пример:**

var i,j,x:integer;

f:real;

. . .

i+4\*j           целый тип выражения, можно записать результат в целую переменную.

f+i\*0.5      вещественный, результат пишется в вещественную переменную.

Оператор **x:= i+4\*j** будет синтаксически правильным, а **x:= f+i\*0.5** будет неверным. Среда разработчика при компиляции сообщит об ошибке несовместимости типов.

Операция деления **/** в Паскале всегда дает вещественное число. Для деления целых чисел с целым результатом (остаток отбрасывается) используйте **div**, для взятия остатка от деления двух целых – **mod**.

Тип переменной позволяет не только устанавливать длину её внутреннего представления, но и контролировать те действия, которые выполняются над ней в программе. Такой контроль осуществляется на этапе компиляции программы и это важное преимущество Турбо Паскаля перед другими языками.

# Виды вычислительных алгоритмов

Любой алгоритм, реализующий решение задачи на компьютере, может быть разработан в виде комбинации трёх базовых алгоритмических конструкций: линейной, разветвляющейся и циклической.

В **линейном** алгоритме все действия выполняются в строгом порядке, последовательно одно за другим.

В **разветвляющемся** алгоритме те или иные действия выполняются в зависимости от выполнения или невыполнения некоего условия.

В **циклическом** алгоритме повторяются некоторые действия, составляющие тело цикла, при выполнении некоего условия. При этом в цикле содержатся повторяющиеся действия с какой-либо изменяющейся величиной, называемой параметром цикла. Ниже приведены блок-схемы алгоритмов: рис.5.1,а – линейный; рис.5.1,б – разветвляющийся; рис.5.1,в – циклический.

Рисунок 1.1

условие

**S**

**да**

**нет**

**S1**

**S2**

**Sn**

**S1**

**S3**

**да**

**нет**

условие

**S2**

условие

**S**

**да**

**нет**

а

б

в

**S1**

**S2**

**да**

**нет**

условие

Здесь S, S1, S2 …. – операторы или блоки операторов языка программирования.

Все алгоритмы имеют общее свойство: один вход и один выход. Именно благодаря этому свойству на их основе можно создавать наглядные алгоритмы, уменьшая тем самым вероятность ошибок при решении сложных информационных задач. Допускается неограниченное соединение структур и их вложение друг в друга. Что позволяет проектировать сложные алгоритмы.

## Линейный вычислительный процесс

Линейный вычислительный процесс представляет собой набор операторов, выполняемых последовательно, один за другим. Основу программы линейного вычислительного процесса составляют операторы присваивания, ввода и вывода данных.

#### Оператор присваивания

**Оператор присваивания** используется для сохранения результата вычисления выражения в переменной. Общий вид:

переменная := выражение;

Знак := читается как "присвоить".

*Работа оператора:*

Сначала вычисляется выражение, стоящее справа от знака :=, затем результат записывается в переменную, стоящую слева от знака. Например, после выполнения оператора

k:=k+2;

текущее значение переменной k увеличится на 2.

Тип переменной должен быть **не младше** типа выражения (например, если выражение дает целое число, результат можно писать и в целую, и в вещественную переменную, если результат вычисления выражения вещественный, писать его в целую переменную нельзя, т.к. может произойти потеря точности).

**Примеры:**

1. Записать оператор присваивания, который позволяет вычислить расстояние между двумя точками на плоскости с координатами (*x*1, *y*1) и (*x*2, *y*2).

*Ответ*:d:=sqrt(sqr(x1–x2)+sqr(y1–y2));

2. Записать последовательность операторов присваивания, обеспечивающих обмен значениями переменных x и y в памяти компьютера.

*Ответ*:          c:=x;

                        x:=y;

                        y:=c;

где с – дополнительная переменная того же типа, что x и y, через которую осуществляется обмен. Грубой ошибкой было бы, например, попытаться выполнить обмен операторами x:=y; y:=x; - ведь уже после первого из них мы имеем два значения y, а исходное значение x потеряно.

#### Операторы ввода и вывода

В данном разделе рассмотрим организацию ввода и вывода данных с терминального устройства. Терминальное устройство - это устройство, с которым работает пользователь, обычно это клавиатура и экран (дисплей).

##### **Ввод данных**

Для ввода исходных данных чаще всего используется процедуры Read и ReadLn. Общий вид записи операторов:

Read(A1,A2,...AK);

ReadLn(A1,A2,...AK);

Здесь и далее список данных, передаваемых любому оператору (а позднее и написанным нами подпрограммам), мы будем называть **параметрами**. Таким образом, параметрами оператора (точней, *стандартной процедуры*) Read являются имена переменных, описанных ранее в разделе var.

При вводе исходных данных происходит преобразование из внешней формы представления во внутреннюю, определяемую типом переменных. Переменные, образующие список ввода, могут принадлежать либо к целому, либо к вещественному, либо к символьному типам. *Чтение исходных данных логического типа в языке Паскаль недопустимо!*

*Работа операторов:*

По достижении оператора ввода выполнение программы останавливается и ожидается ввод данных пользователем. Вводимые с клавиатуры значения переменных разделяются пробелом или нажатием клавиш табуляции и Enter. *Не допускается разделение вводимых чисел запятыми!*

После ввода значений всех переменных из списка работа программы продолжается со следующего оператора. Процедура производит чтение К значений исходных данных и присваивает эти значения переменным А1, А2, ..., АК.

Readln отличается от Read только тем, что все переменные должны быть введены в одну строку экрана, Enter нажимается один раз в конце. Readln используется в основном для ввода строк текста, для ввода чисел лучше использовать Read, т.к. в этом случае пользователь может вводить данные более свободно (и в одну, и в несколько строк экрана).

Если пользователь вводит данные недопустимого типа (например, строку текста вместо числа), то выводится сообщение об ошибке  и работа программы прерывается.

**Пример**: нужно ввести данные для решения квадратного уравнения

var a,b,c:real;

. . . .

read (a,b,c);

Например, a=1, b=4, c=2.5 На экране вводим:

1\_4\_2.5¬

\_ означает пробел

¬ означает нажатие Enter, переходим на следующую строку

или

1¬

4¬

2.5¬

или

1¬

4\_2.5¬

Пробелов может быть и несколько.

Как правило, перед оператором ввода ставится **оператор вывода**, служащий приглашением к вводу и поясняющий пользователю, что именно следует сделать (см. ниже).

##### **Вывод данных**

Для вывода результатов работы программы на экран используются процедуры:

Write(A1,A2,...AK);

WriteLn(A1,A2,...AK);

Переменные, составляющие список вывода, могут относиться к целому, действительному, символьному или булевскому типам. В качестве элемента списка вывода кроме имен переменных могут использоваться выражения и строки.

Работа операторов:  
Элементы списка выводятся в пользовательское окно программы. Форма представления значений в поле вывода соответствует типу переменных и выражений: величины целого типа выводятся как целые десятичные числа, действительного типа - как действительные десятичные числа с десятичным порядком, символьного типа и строки - в виде символов, логического типа - в виде логических констант TRUE и FALSE.

Вещественные значения выводятся в *экспоненциальной форме* (с буквой E). Строки выводятся "как есть". После вывода работа программы продолжается со следующего оператора.

Оператор writeln отличается от write лишь тем, что после вывода значения последнего элемента списка выполняется перевод курсора на следующую строку экрана.

Примеры**:**

1) *Нужно дать пользователю возможность ввести с клавиатуры число, затем программа возведет это число в квадрат и выведет результат на экран*.

var a,a2:integer;

. . . .

writeln ('Введите целое число:');     {это приглашение к вводу!}

read (a);

a2:=sqr(a);

writeln ('Квадрат числа=',a2);

Если ввести a=2, на экране будет напечатано

Квадрат числа=4

|             *---курсор на следующей строке*

После вывода результата выполнение программы продолжится, а если оператор writeln был в ней последним, то и завершится. Чтобы пользователь успел прочитать результат, следует в конце программы добавить оператор

readln;

который будет ждать нажатия клавиши Enter.

2) *Нужно вывести результаты решения квадратного уравнения x1=1.5 и x2=2.5*

write ('x1=',x1,'\_x2=',x2);

*-пробел здесь нужен, чтоб значение x1 не слилось со строкой 'x2='*

на экране будет напечатано:

x1= 1.5000000000E+00 x2= 2.5000000000E+00|

*-курсор остался в конце строки, т.к. использован write!*

Вещественные числа читать в подобной форме неудобно, для их вывода используйте решение из следующего пункта.

##### **Управление выводом данных**

Оператор вывода позволяет задать ширину поля вывода для каждого элемента списка вывода. В этом случае элемент списка вывода имеет вид

*Для целых чисел и строк:***переменная:ширина**где: **переменная** – выражение или строка,  
 **ширина** – выражение либо константа целого типа. Определяет, сколько экранных позиций отводится для вывода всего числа. Если выводимое значение занимает в поле вывода меньше позиций, чем указанное, то перед этим значением располагаются пробелы. Если выводимое значение не помещается в ширину поля К, то для этого значения будет отведено необходимое количество позиций.

*Для величин действительного типа* элемент списка вывода может иметь вид:  
 **переменная:ширина:точность**где - **переменная** – переменная или выражение вещественного типа,  
 **ширина** - ширина поля вывода,  
 **точность** - число цифр дробной части выводимого значения.  
**Ширина** и **точность** - выражения или константы целого типа. В этом случае действительные значения выводятся в форме десятичного числа с фиксированной точкой. Разумные значения точности – от **0** до **ширина-2** включительно. Недопустимые значения ширины и точности не будут учтены при выводе.

Пример: печатаем значения переменных

var x1,p:real;

i:integer;

. . . .

x1:=2.5; p:=-3.175; i:=2;

writeln ('x1=',x1:8:2,'\_p=',p:9:4);

write ('I=','\_':5,i:2);

На экране будет напечатано:

x1=\_\_\_\_2.50\_p=\_\_-3.1750

I=\_\_\_\_\_\_2

*\_ везде обозначает пробел*

##### **Вывод на печать**

Иногда требуется, чтобы программа вывела результаты своей работы на принтер. Для этого достаточно выполнения двух условий.

1)Первым оператором программы следует указать оператор

uses printer;

Он подключает стандартную библиотеку для работы с принтером.

2)Первым  параметром оператора write или writeln указать символическое имя принтера lst, описанное в библиотеке printer:

write ('Hello');                               строка 'Hello' выведена на экран,

write (lst,'Hello');                     а здесь – уже на принтер.

Отличие между write и writeln сохраняется при выводе на принтер – то есть, при использовании writeln позиция печати на принтере будет переведена на следующую строку.

Здесь не приводится код, позволяющий проверить, готов ли принтер к печати и удалась ли операция вывода данных на него. Подобные выполняются с помощью стандартной функции IoResult.

## Разветвляющийся вычислительный процесс

Разветвляющийся вычислительный процесс представляет собой набор операторов, исполняемых в зависимости от выполнения некоторого условия. Основу программы разветвляющегося вычислительного процесса составляют условный оператор условного перехода IF и оператор выбора CASE, а также оператор безусловного перехода GOTO. Они позволяют нарушить последовательный порядок выполнения инструкций программы.

#### Оператор условного перехода

Оператор условного перехода в Турбо Паскаль имеет вид:

**if** условие **then** оператор 1 **else** оператор 2;

условие - это логическое выражение, в зависимости от которого выбирается одна из двух альтернативных ветвей алгоритма. Если значение условия истинно (TRUE), то будет выполняться оператор 1, записанный после ключевого слова then. В противном случае будет выполнен оператор 2, следующий за словом else, при этом оператор 1 пропускается. После выполнения указанных операторов программа переходит к выполеннию команды, стоящей непосредственно после оператора if.

Необходимо помнить, что перед ключевым словом else точка с запятой никогда не ставится!

else - часть в операторе if может отсутствовать:

**if** условие **then** оператор 1;

Тогда в случае невыполнения логического условия управление сразу передается оператору, стоящему в программе после конструкции if.

Следует помнить, что синтаксис языка допускает запись только одного оператора после ключевых слов then и else, поэтому группу инструкций обязательно надо объединять в составной оператор (окаймлять операторными скобками begin ... end). В противном случае возникает чаще всего логическая ошибка программы, когда компилятор языка ошибок не выдает, но программа тем не менее работает неправильно.

Примеры:

if x > 0 then modul := x else modul := -x;

if k > 0 then WriteLn('k - число положительное');

if min > max then begin

t := min;

min := max;

max := t;

end;

#### Оператор выбора

Часто возникают ситуации, когда приходится осуществлять выбор одного из нескольких альтернативных путей выполнения программы. Несмотря на то, что такой выбор можно организовать с помощью оператора if .. then, удобнее воспользоваться специальным оператором выбора. Его формат:

**case** выражение **of**

вариант : оператор;

...

вариант : оператор;

end;

или

case выражение of

вариант : оператор;

...

вариант : оператор;

else оператор

end;

Выражение, которое записывается после ключевого слова case, называется селектором, оно может быть любого перечисляемого типа. вариант состоит из одной или большего количества констант или диапазонов, разделенных запятыми. Они должны принадлежать к тому же типу, что и селектор, причем недопустимо более одного упоминания вариантав записи инструкции case. Из перечисленного множества операторов будет выбран только тот, перед которым записан вариант, совпадающий со значением селектора. Если такого варианта нет, выполняется оператор, следующий за словом else (если он есть).

Пример:

case ch of

'A'..'Z', 'a'..'z' : WriteLn('Буква');

'0'..'9' : WriteLn('Цифра');

'+', '-', '\*', '/' : WriteLn('Оператор');

else WriteLn('Специальный символ')

end;

#### Оператор безусловного перехода

Помимо операторов условного перехода существует также оператор безусловного перехода goto. Формат:

**goto** метка;

Оператор goto переходит при выполнении программы к определенному оператору программы, перед которым находится метка. Метка должна быть описана в разделе описания меток той программы (процедуры или функции), в которой она используется. Нельзя перейти из одной процедуры или функции в другую.

Необходимо, чтобы в программе существовал оператор, отмеченный указанной меткой. Она записывается перед оператором и отделяется от него двоеточием.

Пример:

label 1;

begin

...

goto 1;

...

1: WriteLn('Переход к метке 1');

end.

Само понятие структурного программирования и общепринятый стиль программирования на структурных языках НЕ ПРИВЕТСТВУЕТ применение меток и операторов перехода в программах. Это затрудняет понимание программы, как автором, так и потребителями, кроме того, применение меток отрицательно сказывается на эффективности генерируемого кода.

## Операторы повторений

В языке Турбо Паскаль имеются три оператора, позволяющих запрограммировать повторяющиеся фрагменты программ (циклы). Это операторы **while, repeat, for**.

#### Оператор цикла while-do (цикл с предусловием)

Общий вид записи:

**While** <условие> **do**

begin

<оператор 1>;

...

< оператор n>

End;

Запись означает «пока выполняется <условие>, делать».

Инструкция while-do означает «пока выполняется <условие>, делать» и

выполняется следующим образом: сначала проверяется условие. Если оно истинно, выполняются < оператор 1>...< оператор n> и так далее. Если же условие ложно, выполнение считается законченным, и программа переходит к обработке следующих команд. Таким образом, если условие с самого начала оказалось ложным, < оператор 1>...< оператор n> не будут выполнены ни разу. В цикле while-do <условие> - это условие выполнения цикла: пока оно истинно, программа из цикла не выйдет. В том случае, если внутри цикла стоит только один оператор, пара begin - end может отсутствовать, если операторов больше одного, то пара begin - end обязательна, так как иначе будет выполняться только один оператор, стоящий сразу после слова do.

Пример: Надо вычислить сумму S=1+1/2+1/3+...+1/50

VAR S: REAL; N:INTEGER;

BEGIN

S:=0; N:=1;

WHILE N<=50 DO

BEGIN

S:=S+1/N;

N:=N+1;

END;

WRITELN(' S=',S);

END.

#### Оператор цикла repeat-until (цикл с постусловием)

Общий вид записи:

**Repeat**

< оператор 1>;

...

< оператор n>

**until** <условие>;

Инструкция repeat-until означает «повторять до тех пор, пока не будет выполнено <условие>» и выполняется следующим образом : сначала выполняются < оператор 1>...< оператор n>. Затем проверяется условие. Если оно ложно, происходит возврат с выполнению <оператор 1>...<оператор n>. Если условие истинно, цикл считается выполненным и программа переходит к обработке следующих команд. Таким образом, в цикле repeat-until <условие> - это условие завершения цикла: цикл повторяется, пока оно ложно.

Пример:

VAR S: REAL; N:INTEGER;

BEGIN

S:=0; N:=1;

REPEAT

S:=S+1/N;

N:=N+1;

UNTIL N>50;

WRITELN(' S=',S);

END.

#### Оператор цикла for (цикл с параметром)

Общий вид:

**For** I:=N1 to N2 **do** <простой или составной оператор>;

I - переменная цикла (параметр цикла),

N1- начальное значение цикла,

N2- конечное значение цикла,

I,N1,N2 должны быть одного и того же типа, но не real.

I принимает последовательные значения данного типа от N1 до N2. Если N1 и N2 - целые числа, а I - целая переменная, то шаг всегда равен единице.

Инструкция **for** означает «для I, изменяющегося от N1 до N2 делать» и выполняется следующим образом: вначале вычисляется значение N1 и оно присваивается I. После этого циклически повторяется:

- проверка условия I <=N2. Если условие не выполнено, оператор **for** завершает свою работу;

- если условие I <=N2 выполнено, то исполняется конструкция «<простой или составной оператор>»;

- наращивание переменной I на единицу.

Пример:

VAR S: REAL; N:INTEGER;

BEGIN

S:=0;

FOR I:=1 TO 50 DO

S:=S+1/I;

WRITELN(' S=',S);

END.

Цикл по убывающим значениям параметра I от N2 до N1 имеет вид:

For I:=N2 downto N1 do <простой или составной оператор>;

Пример 1:

For I:=20 downto 1 do A:=A+1;

I изменяется от 20 до 1 с шагом -1.

# Массивы

**Массив** – упорядоченный набор однотипных значений – компонент массива. Тип компонент называется базовым типом массива.

В Паскале массив рассматривается как переменная структурированного типа. Массиву присваивается имя, посредством которого можно ссылаться на него, как на единое целое, так и на любую из его компонент.

Переменная с индексом – идентификатор компоненты массива. Формат записи:

<имя массива> [<индекс>],

где <индекс> может быть выражением порядкового типа.

Описание массива определяет имя, размер массива и базовый тип. Формат описания в разделе переменных:

**Var** <имя массива> : **array** [<тип индекса>] **of** <базовый тип>

<тип индекса> – любой порядковый тип, кроме longint, чаще всего в качестве <типа индекса> используется интервальный целый тип.

<базовый тип > – любой тип Турбо Паскаля.

Линейный **одномерный массив** - массив, у которого элементы – простые переменные. В одномерных массивах хранятся значения линейных таблиц.

Примеры описания одномерных массивов:

Var B: array [0 . . 5] of real;

R: array [1 . . 34] of char;

N: array [‘A’ . . ‘Z’] of integer;

Ввод и вывод массива производится поэлементно. Обычно для этого используется цикл с параметром, где в качестве параметра применяется индексная переменная.

Пример 1.

Заполнить случайными числами из диапазона [0, 1] вещественный линейный массив из N чисел. Найти максимальное значение и его индекс (первый, если таких значений несколько).

Решение.

Поскольку размер массива в программе должен быть однозначно задан, определим N в разделе констант, например, N = 20. При изменении размера массива достаточно будет отредактировать в программе лишь описание константы N.

Const N=20;

Var X : array [1 .. N] of real; k: integer;

Max : real; Kmax : integer;

Begin

For k:=1 to N do

X[k] :=random; {заполнение случайными числами}

Max :=X[1] ; Kmax := 1 ; {инициализация вычисляемых переменных}

For k:= 2 to N do {поиск максимального значения}

If X[k] > Max then begin max := X[k] ; Kmax := k end;

Writeln (‘Первое максимальное значение: X[‘, Kmax, ’]=’, max)

End.

В Турбо Паскале можно одним оператором присваивания передать все элементы одного массива другому массиву того же типа, например:

Var А, B : array [1..5] of integer;

……

A:=B;

После этого присваивания все элементы массива А получат те же значения, что и в массиве В.

Так как <базовый тип > – любой тип Турбо Паскаля, то он может быть и другим массивом, например:

Var B : array [1 . . 5] of array [1..10] of real;

Такую запись можно заменить более компактной:

Var B : array [1 . . 5, 1..10] of real;

Число вложений ограничено только объемом памяти. В приведённом примере массив двухмерный. В **двумерных массивах** хранятся значения двумерных таблиц (матриц), состоящих из строк и столбцов.

Пример описания двумерных массивов:

Var a: array [1..5,1..2] of integer;

Пример 2.

Дан массив из 5-ти строк и 2-х столбцов, содержащий элементы целого типа.Вычислить произведение отрицательных элементов.

Решение.

Var a: array [1..5,1..2] of integer;

i,j,P:integer;

begin

{Ввод элементов массива с клавиатуры}

For i:=1 to 5 do

For j:=1 to 2 do begin

Write(‘Ведите a[‘,i,’,’,j,’]=‘);

Readln(a[i,j]);

end;

{Печать элементов массива в виде таблицы}

Writeln(‘ Массив a’);

For i:=1 to 5 do begin

For j:=1 to 2 do

Write(a[i,j]:4);

Writeln;

end;

{Вычисление произведения отрицательных элементов}

P:=1;

For i:=1 to 5 do

For j:=1 to 2 do

if a[i,j]<0 then

P:=P\*a[i,j];

Writeln(‘Произведения отрицательных элементов P=’, P);

End.

# Подпрограммы

## Понятие подпрограммы

Если часть программы используется более одного раза, то можно не повторять текст программы, а оформить эту часть программы в виде процедуры или функции. Они вызываются на исполнение каждый раз, когда нужно выполнить эту часть программы. При этом саму программу называют основной программой, а процедуру или функцию подпрограммой.

Отличие функции от процедуры заключается в том, что функция вычисляет одно единственное значение простого, строкового или ссылочного типа. Если результатом работы подпрограммы не может быть единственное значение, то лучше использовать процедуру.

Процедуры и функции являются самостоятельными фрагментами программы, связанными с основной программой лишь с помощью нескольких параметров. Структура подпрограммы повторяет структуру всей программы, но вместо заголовка программы содержит заголовок подпрограммы, который является обязательным. Формат заголовка:

FUNCTION <имя> [ (<список формальных параметров>) ] : <тип> ;

PROCEDURE <имя> [ (<список формальных параметров>) ]

Здесь: <имя> - имя подпрограммы (правильный идентификатор);

<тип> - тип возвращаемого функцией результата.

Например, FUNCTION F(a, b: Real): Real;

PROCEDURE SB (a: Real; b: Integer; c: Char);

Переменные, описанные в основной программе, являются глобальными для подпрограмм. Все переменные из списка формальных параметров подпрограмм могут использоваться в любых выражениях внутри подпрограммы и являются локальными.

Любой из формальных параметров подпрограммы может быть либо параметром–значением, либо параметром–переменной, либо, наконец, параметром–константой. Например,

PROCEDURE MyProcedure (var a: Real; b: Real; const c: String);

Здесь А – параметр-переменная, В – параметр-значение, а С - параметр-константа.

Так как любое имя в программе обязательно описывается перед тем, как оно появится среди исполняемых операторов, то и подпрограмма должна быть предварительно определена в разделе описаний основной программы. Описанием (определением) основной программы является её собственный текст, который содержит заголовок и тело подпрограммы.

Например,

PROGRAM A;

Var x,y: integer;

PROCEDURE REV(a,b: integer);

Begin

Writeln(b:3, a:3);

End;

Упоминание имени подпрограммы в тексте основной программы приводит к активации подпрограммы и называется её вызовом. Сразу после активации подпрограммы начинают выполняться, входящие в неё операторы. После выполнения последнего из них управление возвращается обратно в основную программу и далее выполняются операторы, стоящие непосредственно за оператором вызова подпрограммы.

Для вызова подпрограммы на исполнение используется оператор, который состоит из её имени и списка *фактических* параметров (если при определении подпрограммы был указан список *формальных* параметров). Например,

Begin

REV (x,y);

REV (4,5\*SQRT(x-y));

End.

Каждый раз при вызове подпрограммы вычисляются фактические параметры, которые могут быть константами (4), переменными (x,y) или выражениями (5\*Sqrt(y-x)).

При вызове подпрограммы необходимо соблюдать следующее правило: количество, тип и порядок следования фактических параметров должен строго соответствовать количеству, типу и порядку следования соответствующих формальных параметров.

## Подпрограмма–функция

В теле функции должен присутствовать хотя бы один оператор присваивания с идентификатором функции в левой части.

Пример1.

Реализовать функцию с именем POWER, осуществляющую возведение любого вещественного числа в любую вещественную степень.

Решение.

.В программе вводится пара чисел X и Y и выводится на экран дисплея результат возведения X сначала в степень +Y, а затем – в степень –Y. Для выхода из программы нужно ввести Ctrl-Z и Enter.

Var

X, y: Real;

{-----------}

Function Power (a, b : Real) : Real;

Begin {Power}

If a > 0 then

Power := exp (b \* ln(a))

Else if a < 0 then

Power := exp (b \* ln(abs(a)))

Else if b = 0 then

Power := 1

Else

Power := 0

End {Power};

{--------------}

Begin {main}

Repeat

Readln (x, y);

Writeln (Power (x, y): 12:10, Power(x, -y):15:10);

Until EOF

End {main}.

Для вызова функции POWER мы просто указали ее в качестве параметра при обращении к встроенной процедуре WRITELN. Параметры X и Y в момент обращения к функции – это фактические параметры. Они подставляются вместо формальных параметров A и B в заголовке функции и затем над ними осуществляются нужные действия. Полученный результат присваивается идентификатору функции – именно он и будет возвращен как значение функции при выходе из нее. В программе функция POWER вызывается дважды сначала с параметрами X и Y, а затем X и –Y, поэтому будут получены два разных результата.

## Подпрограмма–процедура

Процедура в отличие от функции может вычислять и передавать в основную программу несколько параметров.

Пример 1:

В программе задаются два целых числа 5 и 7, эти числа передаются процедуре INC2, в которой они удваиваются. Один из параметров передается как параметр–переменная, другой – как параметр–значение. Значения параметров до и после вызова процедуры, а также результат их удвоения выводятся на экран.

Решение.

Const

A : Integer = 5 ;

B : Integer = 7 ;

{------------}

Procedure Inc2 (var c : Integer; b : Integer);

Begin {Inc2}

C := c+c;

B := b+b;

Writeln (‘Удвоенные :’, c:5, b:5)

End {Inc2};

{-------------}

Begin {main}

Writeln (,Исходные :’, a:5, b:5);

Inc2(a, b);

Writeln (‘Результат :’, a:5, b:5);

End {main}.

В результате прогона программы будет выведено:

Исходные : 5 7

Удвоенные : 10 14

Результат : 10 7

Как видно из примера, удвоение второго формального параметра в процедуре не вызвало изменения фактической переменной В, так как этот параметр описан в заголовке процедуры как параметр-значение.

Пример 2.

В целочисленной матрице размером 10x10 произвести сортировку чисел в строках по возрастанию значений. Первоначально заполнить матрицу целыми случайными числами в диапазоне от 0 до 100.

Решение.

Для решения задачи составим процедуру SortVec сортировки одномерного массива по возрастанию значений. Используем для этого «алгоритм пузырька». В основной программе используем эту процедуру для сортировки каждой строки матрицы.

Program SortMatr;

Const N=10;

Type Vector=array[1..N] of integer;

Var A: array[1..N] of Vector;

K, l: integer;

Procedure SortVec(Var X: Vector);

Var I, j, z : integer;

Begin

For i:=1 to N-1 do

For j:=1 to N-I do

If X[j] < X[j+1] then

Begin z := X[j]; X[j] := X[j+1]; X[j+1] := z

End;

End;

Begin {Заполнение матрицы случайными числами}

For k:=1 to N do

for l := 1 to N do

A[k, l] := random(100);

For k := 1 to N do SortVec (A[k]); {сортировка строк}

For k := 1 to N do {вывод отсортированной матрицы}

Begin writeln;

For l := 1 to N do write (A[k, l]: 5)

End

End.

При задании массива в качестве формального параметра в заголовке программы типы формальных параметров должны обязательно быть стандартными или производными ранее объявленными типами. Например, следующий заголовок является недопустимым:

Procedure J ( var A:array[1..10] of byte);

В данном случае тип параметра А определяется программистом и поэтому следует воспользоваться идентификатором:

Type

MyArray=array [1..10] of byte;

…………….

Procedure Correct (var A: MyArray);

# Создание и отладка компьютерных программ

## Общие сведения.

Создание текста программы на языке программирования выполняется человеком вручную, а перевод текста программы в машинные двоичные коды выполняется специальными компьютерными программами. Для перевода программ с языков высокого уровня на язык машинных команд используются специальные программы, называемые *трансляторами*, а сам процесс перевода называется трансляцией. Трансляторы подразделяются на две большие группы – *интерпретаторы* и *компиляторы*. Компиляторы сначала переводят всю программу в машинные коды и после этого очень быстро исполняют её. Достоинством компиляции является быстрота исполнения, а недостатком – невозможность увидеть промежуточные результаты. Результат выполнения программы мы увидим только после успешной компиляции, а на этапе предварительной подготовки программа-компилятор обычно требует устранить все синтаксические ошибки. Интерпретаторы покомандно переводят программу на язык машинных кодов и тут же исполняют команду. Программа- интерпретатор при обнаружении ошибки прекращает работу и предлагает исправить ошибку. Достоинство интерпретаторов – возможность видеть промежуточные результаты и вносить в исполняемый алгоритм изменения. Недостаток – гораздо более медленная работа по сравнению с компиляторами

## Система программирования Турбо Паскаль

Система программирования Турбо Паскаль (Turbo Pascal) представляет собой сочетание компилятора с языка программирования Паскаль и программной оболочки, помогающей в создании программ. В названии языка Турбо – это торговая марка разработчика системы Турбо Паскаль фирмы Borland International, Inc (США), а Паскаль – название широко распространённого языка программирования (язык назван в честь выдающегося французского математика и философа Блеза Паскаля). Программной оболочкой предоставляется диалоговая среда, которая способствует повышению эффективности создания программ. Эту диалоговую среду называют также средой разработчика Турбо Паскаля.

Система программирования Турбо Паскаль является одной из наиболее популярных систем программирования. Это объясняется сочетанием двух безусловных её достоинств: исключительной простотой и естественностью языка программирования Паскаль и великолепными сервисными возможностями диалоговой среды программирования фирмы Borland. С помощью Турбо Паскаля можно создавать любые программы – от программ, предназначенных для решения простейших вычислительных задач, до сложных современных систем управления базами данных и операционных систем.

## Диалоговая среда разработчика Турбо Паскаль

Вход в диалоговую среду осуществляется с помощью ярлыка Турбо Паскаль рабочего стола Windows. После вызова системы на экране компьютера откроется диалоговое окно среды разработчика Турбо Паскаля. Для управления средой разработчика Турбо Паскаля используются меню и функциональные клавиши. Верхняя строка открывшегося окна содержит меню возможных режимов работы, а нижняя – краткую справку о назначении основных функциональных клавиш. Вся остальная часть экрана является окном редактора и предназначена для ввода и корректировки текста программ. В его верхней строке приводится имя того дискового файла, откуда был прочитан текст программы, или имя NONAME00.PAS, если текст не читался с диска.

В Турбо Паскале можно работать одновременно с несколькими программами (или частями одной программы), каждая из которых может располагаться в отдельном окне редактора. Номер окна указывается справа в строке имени файла. Кроме окна редактора, используются также окна: отладочного режима, вывода результатов работы программы, справочной службы, стека, регистров. По желанию они могут вызываться на экран поочерёдно или присутствовать на нём одновременно.

С каждым пунктом меню или функциональной клавишей связывается некоторая команда, управляющая средой. Функциональные клавиши обозначаются F1,F2, …, F12 и располагаются в самом верхнем ряду клавиатуры. Действие функциональных клавиш можно модифицировать одновременным нажатием особых клавиш: ALT, CTRL, SHIFT и т.д. Наиболее часто используемые функциональные клавиши и их сочетание называют «горячими клавишами». Ниже приведены основные команды среды разработчика Турбо Паскаль и соответствующие им горячие клавиши (знак «+» между клавишами означает одновременное нажатие клавиш).

* **F10** – вход в меню
* **Ctrl+F9 –** запуск программы
* **Alt+F5** – просмотр пользовательского экрана
* **F2** – сохранение программы
* **F3** – открытие сохраненной программы
* **Alt+F3** – закрытие активного окна
* **Alt+X** – выход из Турбо Паскаль
* **F1** – контекстная помощь
* **Ctrl+F1** – справка об операторе, на котором установлен курсор
* **Alt+Backspace** – отмена последнего изменения
* **Ctrl+Y** – удаление строки
* **Shift+стрелки** – выделение блока текста
* **Ctrl+Insert** – копирование выделенного блока в буфер
* **Shift+Insert** – вставка из буфера

## Основные этапы создания компьютерной программы.

#### Работа в текстовом редакторе среды программирования ТР.

Текст программы на языке ТР необходимо набрать в текстовом редакторе ТР.

При работе в строенном редакторе ТР можно:

-удалить ненужную строку <**Ctrl**>+<**Y**>;

-восстановить случайно удалённую строку или фрагмент текста – **Edit**⇒**Undo**. Эта же команда отменяет последнее изменение;

-удаление фрагмента текста **Edit**⇒**Cut**;

-копирование фрагмента текста **Edit**⇒**Copy**, а затем **Edit⇒Paste**.

#### Сохранение текста программы на диске.

- с новым именем **File⇒Save as…**

- дозапись **File⇒Save** (или нажать клавишу <**F2**>)

#### Поиск ошибок в тексте программы.

После подготовки текста программы можно попытаться исполнить её, т.е. откомпилировать её, связать её (если необходимо) с библиотекой стандартных процедур и функций (компоновка), загрузить в оперативную память и передать ей управление. Эта последовательность действий называется прогоном программы и реализуется командой **Run⇒Run** (или одновременно нажать клавиши **<Ctrl>+<F9>**).

При обнаружении ошибки строка программы, в которой допущена ошибка, будет помечена курсором, а в статусной строке (верхняя часть экрана на красном фоне) будет выведено диагностическое сообщение об ошибке. Чтобы посмотреть подсказку, нажмите клавишу <**F1**>.

Таким образом, легко устраняются синтаксические ошибки. Если же ошибка возникла на этапе работы программы, то простого указания места возникновения ошибки может оказаться недостаточно. При обнаружении ошибок логики и других ошибок в протоколе работы программы в окне вывода появляется сообщение об ошибке следующего формата:

**Runtime error (errnum) at <segment>:<off set>**, где

Runtime error – «ошибка запуска»;

Errnum – содержит код ошибки;

<segment>:<off set> – содержит адрес ошибки.

Для входа в окно вывода выполнить команду

**Debug⇒User Screen** (или одновременно нажать клавиши <**Alt>+<F5**>).

Для поиска конкретного места ошибки можно выполнить команду **Search⇒Find Error** и ввести адрес ошибки из диагностического сообщения в окно ввода адреса.

После исправления ошибки необходимо сохранить исправленный вариант программы (см. п. 5). Затем повторно запустить на исполнение.

Если не удалось устранить ошибку, то в таких случаях обычно прибегают к более детальному анализу. Для этого можно воспользоваться следующими возможностями для отладки программы:

а) *Трассировка*. – пошаговое исполнение программы.

Она выполняется тремя способами:

- **Run⇒Trace Into** (или нажатие <**F7**>) – пошаговое исполнение с трассировкой процедур и функций по мере их вызова.

- **Run⇒Step over** (или нажатие <**F8>)** –пошаговое исполнение программы, обходя вызовы процедур и функций.

- **Run⇒Go to cursor** (или нажатие <**F4>**)–выполнение программы до строки, в которой находится курсор. Далее можно продолжить отладку одним из предыдущих способов.

После выполнения команды первый исполняемый оператор будет подсвечен контрастным цветом. Для его исполнения повторно выполнить команду. При этом система перейдёт в окно исполнения и вернётся в окно редактирования. Если же на экране останется окно исполнения, то это команда ввода и надо ввести исходные данные и нажать клавишу <**Enter**>. Далее выполнять команду трассировки до конца программы.

Для выхода из режима одновременно нажать<**Ctrl**>+<**F2**>.

б) *Использование окна отладчика Watch* – позволяет просматривать в ходе выполнения программы текущие значения переменных и выражений.

Для вызова окна выполнить команду   
**Debug⇒Add watch…** (или нажмите <**Ctrl**>+<**F7>).**

В появившемся окне указать имя программы. Затем повторить команду и вписать имя переменной или выражение, значение которого вы хотели бы наблюдать при отладке программ.

Доступ к окну отладчика возможен с помощью клавиши <**F6**>. Если сделать окно активным <**F5**>, то можно перемещаться в нём, «прокручивая» его содержимое.

Можно также в «подозрительном» месте программы просто «встать» на имени переменной и выполнить <**Ctrl**>+<**F4>**). В открывшемся окне в верхнем поле будет стоять имя переменной. Нажмите <**Enter**> и в среднем поле появится текущее значение переменной. Можно ввести с помощью клавиатуры выражение, значение которого Вас интересует.

#### Просмотр результатов выполнения программы:

**Debug⇒User Screen** (или одновременно нажать клавиши <**Alt>+<F5**>).

При этом откроется окно вывода результатов. Для возврата из окна вывода в окно редактирования нажать клавишу <**Enter**> или <**Esc**>.

Чтобы одновременно видеть текст программы и результаты выполнить команду **Debug⇒Output**. Для отмены повторить ту же команду.