

С/к "Введение в современное программирование. Объектные и компонентные технологии" v.5.5

Физфак МГУ. 2006/7 уч. год.

Ф.В.Ткачев (ИЯИ РАН; Информатика-21) и А.Н.Соболевский (каф. квантовой статистики, физфак МГУ).

Лекции: четверг, 16:45-18:30, ауд. 5-26

Комп. класс: четверг 18:30–21:00, понедельник 17:00–21:00.

Напоминание осн. тезисов вводной лекции

Принцип Калашникова:

Избыточная сложность = уязвимость

Сфера ИТ: постоянно генерирует избыточную сложность — причин несколько, главная: источник «ренды сложности».

Н.Вирт: «Раковая опухоль избыточной сложности»

R.Perlman: "Alles ist zu viel kompliziert"

....

Основное правило рационального поведения:

Как огня бояться избыточной сложности

«Выдавливает» всеми способами, при каждой возможности (C++, «оптимизации» ...).

KISS = Keep It Simple, Stupid!!!

Дополнения

Насчет правильного обучения: *Хромой, бегущий по дороге, опережает того, кто бежит без дороги.* (Фрэнсис Бэкон)

Насчет окружающего нас вранья: *Живи и бойся: никогда не знаешь, как тебя в следующий раз зомбируют.*

Насчет стадных эффектов (2005): .. *Ошеломленные турецкие пастухи в местечке Джеवास в провинции Ван в восточной Турции были вынуждены наблюдать, как вслед за одной овцой, которая прыгнула с обрыва и разбилась, все их стадо в количестве 1500 животных последовало ее примеру и бросилось с того же самого обрыва. В конце концов, на дне обрыва образовалась гора из 450 мертвых животных, передает AP со ссылкой на турецкую газету Aksam. Те овцы, которые спрыгнули позже, выжили, поскольку растущая гора трупов смягчила падение. ..*

Письмо гл. архитектора любимой системы зам.декана В.Задкова:

From Niklaus Wirth Jun 19, 2003

Date: Thu, 19 Jun 2003 10:29:22 +0200

To: gutknecht@***, zueff@***, ftkachov@***, moessenboeck@***

46 From: Niklaus Wirth <wirth@***>

47 Subject: Oberon

48

49 It is always nice to receive compliments, such as the one below. But in
50 this case I forward them to you, because it might encourage you in your
51 work. It is nice to hear that also in industry there are people
52 appreciating thoroughness, clarity, and honest work.

53

54 How was your trip to Moscow?

55 Niklaus

56 -----

57 Hello Dr. Wirth,

58

59 I've been reading "Project Oberon" over the last year. And have found
60 it very inspiring.

61 It is common these days for students to consider the elements that go
62 into developing such a systems from such a system to be too complex
63 tackle at once.

64 Today's systems have turned these core blocks into black boxes.

65 It was refreshing to see such a simple approach.

66 In my work on a graphical dataflow programming language called **LabVIEW**
67 it is a constant challenge to keep things simple, and to develop
68 reusable code and patterns that are common to the whole system.

69 Project Oberon has become my favorite example of a system
70 built with these rules in mind and in heart.

71

72 I remember you visiting the University of Texas, I believe while I was
73 in school (I graduated 1987) and perhaps once after that, and wanted to
74 pass on this note of appreciation.

75

76 Sincerely,

77 Paul Austin

78 Principal Architect

79 National Instruments

80

81 ... It's really an oddity why programs like MatLab and LabView exist.

82 I find it a little strange to have a separate language for every application.

83 These applications could have been collections of computer modules if only The

84 Right Language had existed and been in widespread use at the time.

85 ...

86 Sincerely,

87

88 Rex Couture, Ph. D.

89 Dept. of Earth and Planetary Sciences

90 Campus Box 1169

91 Washington University in St. Louis

92 1 Brookings Dr.

93 St. Louis MO 63130

94

95

96

Сегодня

98 Введение в Блэкбокс и Компонентный Паскаль, примеры простейших программ

Введение в Блэкбокс

(К сожалению, в программировании есть элементы *ремесла*.)

Как установить: просто скопировать куда-нить (хоть несколько разных экземпляров).

Как запускать: кликнуть по BlackBox.exe.

Постепенно читать и изучать:

— инструкции на диске в сайте Информатика-21 (ссылка **Работа в Блэкбоксе** на первой странице в правой колонке).

— инструкции в файле Пошаговая/Docu/Обзор.odc (открыть этот файл из ББ, дальше кликать на гиперссылки).

Как настроить под себя ББ

Обязательно сделать такие настройки:

меню **Edit, Preferences...**

Thick Caret

Status bar --> Double

Default font... --> Verdana + любимый размер

Структура папок ББ

Программы состоят из отдельных модулей, хранятся по определенным правилам.

У каждого модуля есть имя.

Модули группируются в **подсистемы**:

Однозначно по имени модуля:

KursПример1 --> Kurs + Пример1 (первая заглавная после незаглавной)

"подсистема" Kurs -- отдельная папка в папке ББ.

в ней несколько папок, важнейшие:

Mod — тут следует хранить исходник для модуля:

Kurs/**Mod**/Пример1.**odc**

именно здесь ББ будет искать исходник модуля, чтобы показать

место прерывания

Code — скомпилированные машинные коды для модулей:

Kurs/**Code**/Пример1.**ocf** — создается автоматически

Sym — символьные файлы для модулей:

Kurs/**Sym**/Пример1.**osf** — создается автоматически

Docu — файлы документации для модулей:

Kurs/**Docu**/Пример1.**odc** — создается руками

Часто там можно найти "главный" файл — Sys-Map.odc

Rsrc — всякого рода ресурсы, специфич. для данной подсистемы: меню, диалоговые формы ...

.odc = Oberon DoCument

.ocf = Oberon Code File

.osf = Oberon Symbol File

Каждый из этих типов можно открыть из ББ (обратить внимание на тип файла внизу диалога).

Упр Пооткрывать разные файлы из подсистем Пошаговая, Info21dialog, Epse21, Lib, Texts.

NB квазистандартные сокращения:

Cmds = commands

Docu = documentation

Rsrc = resources

Модули вне подсистем (т.е. имена нельзя разбить, типа A, B, X0): папки Mod, Code, Sym в самой папке ББ.

Подсистема System — особая. Исходник модуля In хранится в System/Mod/In.odc — и ББ правильно его находит.

Сообщения компилятора — файл Dev/Rsrc/Errors.odc

Поменять (**на лету!!!**), сохранить и выполнить **Dev, Flush Resources**.

В лицейской версии -- на русском языке.

Для совсем начинающих я бы рекомендовал начать с русских сообщений, но освоившись, перейти на английские — *use your judgement*.

Меню — файлы ***/Rsrc/Menus.odc**

Откройте один такой файл и посмотрите, как он устроен. Сообразите связь с командами меню. (Подробнее позже специально.)

Русские меню — в лицейском пакете, но упрощены для школы и (пока) плохо работать с клавиатуры (Alt+буквы).

Если все же хочется, откройте соответствующие английский и русский варианты одновременно в ББ (бросьте мышкой) и скопируйте нужные куски.

Спасите файлы, выполните **Info, Update menus** — меню обновятся на лету.

Что мы видим, когда запускаем ББ

Запускаем ББ с помощью желтой иконки.

Максимизировать (растянуть на два экрана у кого есть).

Откроются какие-то окошки.

Закрыть все, оставив только окошко Log.

(Кстати, его можно в любой момент снова открыть командой меню **Info, Open Log**.)

Окошко **Log** — т.наз. **рабочий журнал**. Там появляются сообщения компилятора и т.п.

Туда же можно печатать всякую всячину как из программы, так и "руками" — это очень удобно, и это мы еще обсудим.

Содержимое окошка Log — "стандартный текст ББ" — как и все исходники.

Текст в окошке Log (как и в других окошках ББ) можно редактировать как обычно в ОС Windows с помощью клавиатуры и мышки: выделять фрагменты, стирать их, печатать с клавиатуры в любое место, делать вставки из системного буфера обмена, и т.п.

NB Не нужно создавать важных текстов в окошке Log: при завершении работы ББ содержимое Log будет потеряно без какого-либо предупреждения.

Блэкбокс как текстовый редактор. Начальные сведения

Почему важно:

Принцип приоритета чтения над написанием

Львиную долю своего времени программист проводит за чтением программ.

Даже когда сочиняете программу — пялитесь на уже имеющийся текст и представляете себе новый.

Отсюда масса следствий ... будем разбирать конкретно.

Например, важность предельно четкого, предельно выразительного оформления программных текстов.

Загнать в привычку, в подсознание.

Все программы в ББ (как и эти лекции) создаются как "стандартные тексты" ББ. Тексты в ББ — *намного богаче и мощнее*, нежели текст в Notepad, и даже чем в MS Word (правда, не во всем).

Фактически ББ можно рассматривать как весьма мощный текстовый редактор.

Линейка меню имеет обычный вид для виндусовых программ.

Для простейшей работы с текстами достаточно команд меню File, Edit, Attr, Text.

Основные команды — стандартные (в "классическом" [1988] Обероне — нет)

File, New или Ctrl+N открывает новый пустой текст в новом окошке;

File, Save или Ctrl+S сохраняет текущее состояние в файл на диске;

File, Save As... сохраняет копию текущего состояния в другой файл;

стрелочки -- движение по тексту;

Ctrl+стрелочки -- движение по словам;

Shift(+Ctrl)+стрелочки -- выделение фрагмента текста для послед. копирования и т.п.

Edit -- редактирование:

Edit, Copy|Cut|Paste (Ctrl+C|X|P);

и т.д.

Attr (attributes) -- "раскраска" текста

Ctrl+B|I = жирный, курсив.

размеры, цвет шрифта

и т.д.

Text -- команды, специфические для текстов

Find/Replace (Ctrl+F) -- диалог для поиска/замены.

ББ поддерживает неограниченное кол-во Undo (Edit, Undo или Ctrl+Z).

ББ помнит все изменения, начиная с последней спасенной версии.

Упр Потренируйтесь в работе с простыми текстами:

создать, напечатать, сохранить [как ББ-документ или в другом формате — Save as type...];
открыть [документ ББ или стандартный текст], поредактировать, сохранить;
создать/открыть два текста, покопировать из одного в другой, сохранить.

Еще раз: Все программы в ББ создаются как тексты ББ.

Капитализация

Во всех языках, производных от Оберона, используется много слов, которые должны писаться только заглавными буквами. Для облегчения набора таких слов в нашем комплекте Блэкбокса есть спец. средство: если сразу после набора с клавиатуры слова строчными буквами нажать клавишу **F5**, то все буквы слова будут преобразованы в заглавные (включая русские буквы). Если слово является первым в одной из синтаксических конструкций языка (**или даже первой буквой**), то в текст будет вставлена скелет-заготовка всей конструкции, причем с правильными отступами в начале строк (отступы оформляются символами табуляции; для этого в строке данному слову должны предшествовать лишь символы табуляции).

Например,

while (или даже просто w)

(с двумя отступами-табуляциями в начале строки) превратится в

WHILE DO

END;

причем в средней строке будет вставлено уже три символа табуляции — на один больше.

Простая капитализация без раскрытия конструкции достигается нажатием **Shift+F5**.

Для справки:

0) Эти операции можно выполнить и через меню: **Text, Capitalize** и т.п.

1) Если перед выполнением операции выделить мышкой фрагмент текста любой длины и структуры, то капитализация будет выполнена для всего фрагмента, а не только для слова, предшествующего текстовому курсору.

2) Вместо (Shift+)F5, эти операции можно "посадить" на другие горячие клавиши по усмотрению пользователя. Задание горячих клавиш осуществляется в документе, описывающем меню. В курсе мы это будем обсуждать.

3) Процедуры, осуществляющие эти операции, заданы в модуле **FVTsysEdit**, находящемся в файле FVTsys/Mod/Edit.odc. Их можно изменить, перекомпилировать и активизировать, не выходя из ББ (для активизации нужно уметь выгружать модули -- см. отдельно).

Об отступах

Отступы — лучшее средство выявления структуры программы (**срочно забудьте** про "блок-схемы", "структурограммы" и т.п.). Факт известен с 70-х гг. J.M.Fox Software and its development, Prentice-Hall, 1982
перевод: Дж.Фокс Программное обеспечение и его разработка, Мир, 1985
автор -- очень видный специалист.

301 В ББ все **отступы следует делать только с помощью**
 302 **табуляций (клавиша Tab)** — а не пробелами!

303
 304 **NB** Перевод строки (Enter) делает в новой строке столько же отступов.
 305

306 Клавиши **F11/F12**: сдвиг группы строк влево-вправо на один отступ
 307 (отметить мышкой от середины первой до середины последней строки).
 308

309 Мелкая техника

310 Часто нужно переставлять строки: **выделять строки целиком** (клик в
 311 середину первой строки, Home, Shift+стрелка вниз), потом Ctrl+X, стрелочка
 312 вверх/вниз в нужное место (идем по началам строк), Ctrl+V.
 313

314 ***** Нравоучение.** О важности уметь менять привычки
 315

316 Простейшие примеры программ

317 Не будем входить во все детали и тонкости, просто дадим образцы, по которым
 318 можно начать решать не слишком сложные задачи (на самом деле очень много).

319 Отдельные темы:

320 а) читать-понимать

321 б) набивать ("ремесло") -- см. документы Пошаговая\Доси\Обзор.odc

322 Сначала сосредоточимся на чтении/понимании, чтобы привыкнуть.

323

324 Предположим, что есть такой текст, который и будем разбирать:

325

326 MODULE Kurs2006Пример0;

327 IMPORT StdLog;

328

329 PROCEDURE Do*;

330 VAR x, y: REAL;

331 BEGIN

332 StdLog.Real(x); StdLog.Ln;

333

334 x := 3.14159;

335 StdLog.Real(x); StdLog.Ln;

336

337 y := x;

338 StdLog.Real(y); StdLog.Ln;

339

340 y := - x * x;

341 StdLog.Real(y); StdLog.Ln;

342

343 y := - x * x;

344 StdLog.Real(y); StdLog.Ln;

345 END Do;

346

347 (*комментарий (*вложенный комментарий*)*)

348 END Kurs2006Пример0.

349

350 Обычно такой текст располагается в начале документа

351 Kurs2006/Mod/Пример0.odc (Kurs2006Пример0 --> Kurs2006+Пример0).

352

353 Компилируется нажатием Ctrl+K (здесь латинское K; через меню: Dev, Compile).

354 Могут быть информационные диалоги насчет создания новых папок (Code, Sym).

355 В окошке Log системное сообщение:

356

357 compiling "Kurs2006Пример0"

358 new symbol file 136 0

359

360 Значит:

361 Не было ошибок.

362 Не было старого варианта модуля с таким же именем.

363 136 -- размер созданного файла с машинными командами

364 (Kurs2006/Code/Пример0.ocf).

365 0 -- размер блока глобальных переменных.

366

367 Ошибки отмечаются черными квадратами.

368 Правило при написании текста модуля:

369 Почаще нажимать Ctrl+K!

370

371 **NB** Успешное компилирование делает модуль частью системы (ББ) -- его теперь
 372 можно использовать.

373

374 Повторное нажатие Ctrl+K вызывает повторную компиляцию.

375 Т.к. модуль уже имелся, сообщение в логе будет:

376 compiling "Kurs2006Пример0" 136 0


377

378 Как выполнить?

379 Три способа: командер (простейший), меню, диалог.

380

381 Командер: в любом тексте (чаще после модуля) вставляем такую штуку:

382  Kurs2006Пример0.Do

383 фамилия.имя

384

385 Черный кружок = командер (Tools, Insert Commander или Ctrl+Q -- в меню

386 подсказка насчет Q)

387 После командера -- полное имя процедуры в системе.

388

389 Выполнение процедуры -- клик по командеру.

390 В логе появятся такие строчки:

391

392 5.25061707259783E-308

393 3.14159

394 3.14159

395 -9.869587728099999

396

397 Можно поменять текст (например, 3.14159 заменить на 2.71828),

398 скомпилировать (Ctrl+K),

399 и снова кликнуть по командеру -- напечатается то же самое, т.е. срабатывает
 400 старая версия модуля.

401 Чтобы сработала новая: **при клике нажимать Ctrl.**

402 Строчки в логе будут другие.

403
 404 Причина: при вызове процедуры модуль загружается в память **и там остается**.
 405 Последующие попытки вызвать процедуру вызывают ее из уже сидящего в
 406 памяти модуля.
 407 Нажатие Ctrl при клике заставляет ББ сначала стереть (выгрузить) модуль из
 408 памяти, и только потом вызвать процедуру как если бы модуль был совершенно
 409 новым.
 410
 411 **Выполняются процедуры,**
 412 **а модули — упаковка/контейнеры для них.**
 413
 414 Такой способ работы соответствует диалоговой разработке программ.
 415 Старый способ (компилировать, линковать в ехе, запускать) -- архаическое
 416 наследие времен "пакетной обработки" -- примерно до конца 1970-х.
 417 **NB** В Оберонах/ББ линковка динамическая, т.е. выполняется в момент загрузки,
 418 автоматически. Подробнее позже.
 419
 420 **Читаем — синтаксис**
 421 Сначала сосредоточимся на чисто внешних аспектах текста, не въезжая в смысл.
 422 Прежде всего, в КП заглавные и строчные буквы различаются. Это важно, и об
 423 этом ниже.
 424 Первая строка представляет собой фразу (заголовок модуля), которая состоит
 425 из двух слов, MODULE и Kurs2005Пример3, разделенных пробелом, а
 426 завершается фраза точкой с запятой ";".
 427 Слово MODULE — одно из т.наз. **зарезервированных слов**
 428 Оберона/Компонентного Паскаля.
 429 Другие примеры: IMPORT, PROCEDURE, и т.д.
 430 Таких слов в КП немного и они имеют четко фиксированный смысл, близкий к
 431 их значению в естественном языке.
 432 Зарезервированные слова пишутся **только заглавными буквами** (об этом
 433 ниже)
 434 Их нельзя (не следует) использовать ни в каких других целях (т.е. в качестве
 435 идентификаторов).
 436
 437 **О написании зарезервир. слов заглавными буквами**
 438 *Это поперву м.б. непривычно, но чрезвычайно удобно, и вот почему:*
 439
 440 *30% мозга человека занято обработкой зрительной*
 441 *информации.*
 442 *Эти 30% и представляют собой настоящие "глаза" —*
 443 *мощнейший инструмент "распознавания образов".*
 444 *Учитывая, что чтение программ занимает львиную долю*
 445 *времени программиста,*
 446 *нужно стремиться к тому, чтобы задействовать "глаза" с*
 447 *максимальной пользой.*
 448
 449 *Написание зарезерв. слов заглавными буквами позволяет глазу:*
 450 *0) легко выделять структуру программного текста.*

451 *1) освободить цвет для более важных вещей: выделять измененные*
 452 *фрагменты красным и т.п.*
 453 *(б-во ошибок связаны с модификациями программных текстов).*
 454
 455 Слово Kurs2006Пример0 — пример идентификатора.
 456
 457 **Идентификатор** — это любая последовательность латинских и русских букв
 458 (заглавных и строчных) и цифр, начинающаяся с буквы. Начинаться с буквы
 459 идентификатор должен, чтобы компилятору было легче отличать его от
 460 числовых констант (которые в КП всегда начинаются с одной из цифр 0..9).
 461
 462 КП разрешает использовать в идентификаторах и символ подчеркивания "_" и
 463 считает его строчной буквой.
 464 Символ подчеркивания разрешен, чтобы было легче сопрягаться с программами,
 465 написанными на других языках. **Не следует** его использовать для других целей!
 466
 467 Насчет русских букв: возможны варианты.
 468
 469 **Упр** Найти в данном примере модуля все идентификаторы.
 470
 471 Идентификаторы задаются программистами произвольно для обозначения
 472 различных сущностей программы — модулей, процедур, переменных и типов.
 473
 474 Хотя с точки зрения языка идентификатор Kurs2006Пример0 — это просто
 475 идентификатор как и любой другой, Блэкбокс интерпретирует его как
 476 состоящий из двух частей: Kurs2006 и Пример0. Это нужно для организации
 477 удобной группировки модулей при хранении.
 478
 479 В строке
 480 `x := 3.14159;`
 481 фигурирует **явная константа** 3.14159. Она задает дробное число в
 482 соответствии с обычными правилами арифметики ("три целых, четырнадцать
 483 сотых ...").
 484 *В некотором глубоком смысле цепочка литер 3.14159 — это,*
 485 *фактически, тоже имя для некоторой сущности (числа). Но не будем*
 486 *углубляться в метафизические тонкости.*
 487
 488 После слова MODULE следуют пробелы, отделяющие слово MODULE от
 489 идентификатора Kurs2006Пример0.
 490 Пробелы, символы табуляции, символы концов строк — это всё т.наз. **белые**
 491 **символы** (white space).
 492 Где можно вставить один такой символ, там можно вставлять их любое
 493 количество, причем в любых комбинациях — смысл от этого не изменится.
 494 Не должно быть белых символов внутри зарезервированных слов,
 495 идентификаторов, числовых констант и т.п.
 496 (Кстати, в фортране -- можно, откуда чудовищные ошибки:
 497 `DO I=0,10` заголовок цикла по I от 0 до 10
 498 `DO I=0.10` присваивание переменной DOI значения одна десятая
 499)
 500
 501 Обязательно нужно вставлять хотя бы один белый символ в тех местах, где
 502 иначе сольются соседние зарезервированные слова, идентификаторы и/или
 503 числовые константы.

504
 505 Например, с учетом сказанного, модуль можно переписать сл. образом без
 506 изменения смысла, оставив лишь минимально необходимое число пробелов:
 507
 508 `MODULE Kurs2006Пример0;IMPORT StdLog;PROCEDURE Do*;VAR x:REAL;BEGIN`
 509 `x:=3.14159;StdLog.Real(x);StdLog.Ln;x:=-x*x;StdLog.Real(x);StdLog.Ln;END`
 510 `Do;END Kurs2006Пример0.`
 511
 512 Разумеется, только безумный программист будет писать подобные тексты, т.к.
 513 человеку работать с ними невозможно.
 514 Но подобные тексты могут порождаться программно, если они не
 515 предназначены для чтения, а только для передачи непосредственно
 516 компилятору. (ср. маш. язык, C)
 517
 518 **NB** В программе на КП под управлением ББ можно "на лету", т.е. полностью
 519 автоматически в исполняемой программе без участия человека,
 520 сгенерить текст модуля, скомпилировать его и вызвать процедуры этого
 521 модуля.
 522 Например, программа может вычислить коэффициенты некоторого
 523 аппроксимирующего выражения для какой-то весьма сложной функции,
 524 сгенерить и скомпилировать модуль с процедурой, описывающей это
 525 выражение, и затем поработать с этой функцией, используя
 526 скомпилированную процедуру (например, заняться порождение
 527 случайных чисел, распределение которых задается данной функцией). В
 528 курсе будут даны примеры того, как этот трюк делается.
 529 С концептуальной точки зрения это возможно благодаря динамической
 530 загрузке модулей, а с практической — благодаря еще и весьма высокой
 531 скорости компиляции в Оберонах.
 532
 533 *Данное средство уникально для эффективно*
 534 *компилируемых процедурных языков.*
 535 Его эквиваленты обычно имеются только в интерпретируемых
 536 функциональных языках (лисп, SML и т.п.). В обертоноподобных системах
 537 это средство вынесено за пределы самого языка программирования в
 538 библиотеки по практическим соображениям, но это обстоятельство
 539 не принципиально с точки зрения его (средства) практического
 540 использования.
 541
 542 **Символы-ограничители.** В приведенном тексте модуля есть еще точки с
 543 запятой, двоеточия, точки, скобки. Эти символы играют синтаксическую роль,
 544 подобную их роли в обычных текстах и в математике. Например, точка "."
 545 встречается в роли десятичной точки в числовой константе 3.14159, но также и
 546 в самом конце модуля. Есть и неразделимый двухлитерный символ, состоящий
 547 из двух точек.
 548
 549 Дважды встречается двухлитерная комбинация := (оператор присваивания).
 550 В языке КП есть еще несколько подобных двухлитерных комбинаций, например,
 551 операции нестрогого неравенства <= и >=.
 552 Все они неразделимы: между двумя литерами не может быть ни пробелов, ни др.
 553 белых символов.
 554
 555 В строке
 556 `x := - x * x;`

557 наличествуют арифметические **операторы**: "-" (минус или вычитание) и "*" (умножение).
 558 Есть еще операторы "+" (плюс или сложение) и "/" (деление).
 559 Есть и еще два арифметических оператора, записываемых с помощью
 560 зарезервированных слов DIV и MOD (целая часть и остаток деления двух
 561 целых).
 562 О точном смысле всех этих операторов мы будем говорить специально.
 563
 564 В строке
 565 `PROCEDURE Do*;`
 566 которая является заголовком процедуры, встречается т.наз. **символ экспорта**
 567 (обычная звездочка, как и у оператора умножения). В качестве символа
 568 экспорта может выступать и "-" (минус).
 569 Символы экспорта играют ключевую роль в обеспечении механизмов
 570 **инкапсуляции** ("упрятывания информации"), о чем мы будем подробно
 571 говорить.
 572
 573 Наконец, в предпоследней строке содержится **комментарий**: произвольных
 574 набор литер, заключенный в составные скобки (* и *).
 575 Комментарии могут быть вложенными. Это позволяет временно "исключать из
 576 игры", прекращая в комментариях, фрагменты программы вместе с
 577 комментариями.
 578 Но в любом случае открывающие и закрывающие комментаторные скобки, (* и
 579 *), должны быть сбалансированы по обычным правилам.
 580
 581 Итак, главное, что нужно уяснить сейчас:
 582 Программные тексты на Компонентном Паскале состоят из:
 583 слов из заглавных букв, зарезервированных в языке для специального
 584 использования;
 585 определяемых программистом идентификаторов;
 586 явных констант;
 587 специальных символов, которые могут состоять из одной или двух литер.
 588 Пробелы и т.п. могут (а иногда и должны) стоять между этими элементами,
 589 но никогда — в их середине.
 590
 591 В Блэкбоксе тексты могут содержать и нелитерные визуальные объекты
 592 (например, картинки; наряду с блэкбуксовскими визуальными объектами можно
 593 вставлять любые OLE-объекты, допустимые в среде MS Windows, например,
 594 **формулы**).
 595 При компиляции такие объекты полностью игнорируются (т.е. не считаются
 596 даже белыми символами, т.е. их можно вставлять в середину идентификатора и
 597 т.п.).
 598
 599 **Читаем — смысл**
 600
 601 Разберемся теперь со смыслом нашего модуля.
 602 Первая и последние строки,
 603
 604 `MODULE Kurs2006Пример0;`
 605 `и`
 606 `END Kurs2006Пример0.`
 607
 608 оформляют начало и конец модуля. Они действуют как своеобразные скобки.

609 Слова MODULE и END определены языком (точнее, Н.Виртом).
 610 Идентификатор Kurs2006Пример0 придуман программистом. Он подчиняется
 611 правилам языка, но в остальном произволен.
 612 Смысл этого идентификатора — имя данного модуля. Используя это имя,
 613 Блэкбокс будет размещать/находить скомпилированный двоичный код для
 614 данного модуля.
 615
 616 **NB** Имя модуля повторяется в конце, чтобы облегчить поиск ошибок
 617 (ограничить их распространение).
 618 **NB** Здесь и в других подобных случаях будут ошибки, если в первом случае
 619 использовать, например, латинскую K, а втором случае — кириллическую. Для
 620 компилятора это совершенно разные буквы. То же и для р и е.
 621 **NB** Заключительная фраза модуля заканчивается точкой, а не точкой с запятой,
 622 как остальные фразы.
 623
 624 Имя модуля — автоматически является **внешним** в том смысле, что оно "видно"
 625 (т.е. может использоваться) в других модулях.
 626 Данный модуль становится "видимым" после первой успешной компиляции —
 627 когда на диске появится соответствующий машинный код, к которому ББ и
 628 другие модули могут обращаться по мере надобности.
 629 Другие имена вовсе не являются внешними автоматически.
 630
 631 В самом начале изучения ББ можно использовать простейшие имена для
 632 модулей типа M или Y1 и т.п.
 633 В любом случае следует придерживаться правила, чтобы имена модулей
 634 начинались с заглавных букв.
 635
 636 Сразу же посмотрим на аналогичную пару строк:
 637
 638 PROCEDURE Do*;
 639 и
 640 END Do;
 641
 642 Эта пара оформляет начало и конец процедуры.
 643 Идентификатор Do — придуманное программистом имя процедуры, оно
 644 повторяется в заключительной фразе, чтобы ограничить распространение
 645 ошибок вроде пропущенного END в середине процедуры.
 646
 647 **Рекомендуется** начинать имена процедур с заглавных букв.
 648
 649 **Звездочка** заголовке процедуры после ее имени — **символ экспорта**.
 650 Именно наличие этой звездочки делает данную процедуру видимой вне модуля,
 651 т.е. доступной из других модулей. (Весь Блэкбокс состоит из модулей,
 652 написанных на КП, и ББ не сможет вызвать данную процедуру на исполнение,
 653 если он ее "не видит".)
 654
 655 **NB** Экспорт процедуры — ответственное решение.
 656 Ибо чем меньше "торчит" всего из ваших модулей, тем надежней вся система.
 657 Именно поэтому решение об экспорте процедуры из модуля требует явного
 658 действия (вставка звездочки).
 659 *Поэтому глупо вставлять звездочки для всех процедур подряд.*
 660

661 Фраза во второй строке модуля:
 662 IMPORT StdLog;
 663 обозначает наше намерение использовать в данном модуле средства
 664 (процедуры и т.п.), экспортированные из модуля StdLog — т.е. мы импортируем
 665 (IMPORT) модуль StdLog.
 666 StdLog экспортирует что-то для всех желающих, и вот мы решаем этим
 667 воспользоваться.
 668
 669 Если нужно воспользоваться средствами нескольких модулей, то их имена
 670 нужно перечислить через запятую, например:
 671 IMPORT StdLog, Math;
 672
 673 Поскольку имена модулей могут быть довольно длинными (ср.
 674 Kurs2006Пример0), то разрешается вводить для них краткие псевдонимы:
 675 IMPORT Log := StdLog, Math;
 676 Такой псевдоним действует на протяжении данного модуля.
 677 Раз ввели псевдоним, то к процедурам нужно будет обращаться через
 678 псевдоним:
 679 Log.Real(x); Log.Ln;
 680 Но об этой строке позднее.
 681
 682 **Пока суммируем:**
 683 Модуль Kurs2006Пример0 импортирует (использует) модуль StdLog,
 684 а сам в свою очередь экспортирует одну процедуру без параметров с именем Do.
 685 *Если нарисовать все модули в виде точек, а связи импорта между ними*
 686 *нарисовать стрелочками, то получится "ориентированный граф" — граф,*
 687 *в котором нельзя вернуться в исходную точку, идя по направлению*
 688 *стрелок, откуда бы ни стартовать.*
 689
 690 Как посмотреть, что импортирует модуль (= его интерфейс):
 691 — дважды кликнуть по имени модуля (в любом окне ББ)
 692 — нажать Ctrl+D ("definition")
 693 появится окошко с текстом:
 694
 695 DEFINITION Kurs2006Пример0;
 696
 697 PROCEDURE Do;
 698
 699 END Kurs2006Пример0.
 700
 701 Вот так выглядит наш модуль для всех других модулей.
 702 "Виден" только интерфейс -- сигнатура процедуры Do.
 703 Его нутро скрыто, из других модулей "влезть" в него нельзя -- "черный ящик"
 704 (black box).
 705
 706
 707 **Процедура Do**
 708
 709 Содержимое процедуры Do разбивается на две части: до и после слова BEGIN.
 710 До — раздел объявлений.
 711 После — исполняемая часть, собственно инструкции.
 712
 713 **Раздел объявлений**

После слова VAR (variables = переменные) мы "заказываем" ячейки памяти (**переменные**), которые нам понадобятся в программе, указывая для каждой имя (идентификатор) и **тип**.

Важность явного объявления переменных: из-за одной необъявленной переменной в 1962 была потеряна ракета Mariner, запущенная на Марс. В языках семейства Паскаля (Модуль-2, Оберон/КП) компилятор требует от программиста объявлять переменные -- принципиальное требование.

Ячейка памяти — устройство, которое можно перевести в одно из некоторого конечного числа состояний.

Подробнее потом отдельно, сейчас быстренько:

Бит: up, down (0, 1; T, F).
 Байт: 8 бит, 256 состояний.
 Полуслово: 2 байта = 16 бит, $2^{16} \sim 32K$ состояний.
 Слово: 4 байта = 32 бита, $2^{32} \sim 4G$ состояний.
 Двойное слово: 8 байт = 64 бита, 2^{64} состояний.

NB терминология "плавает" (слово = ?)

Состояния могут интерпретироваться по-разному: подмн-во, литера, целое без знака, целое со знаком, вещественное, цепочка литер, машинный адрес ...

Переменная — это некоторая ячейка памяти, для которой фиксирован размер (кол-во байт), а также фиксирована интерпретация ее состояний. Это называется **тип** переменной.

тип = интерпретация (+ размер)

Строгая статическая типизация = это когда каждой переменной приписан тип в самом тексте программы, этот тип в программе не меняется, компилятор отслеживает правильность манипуляций с переменными (присваивания) с предельной жесткостью (никаких предупреждений -- только ошибки).

У нас заказаны две переменных с именами x и y.

Для обеих задан тип **REAL**. Это один из **основных типов**, встроенных прямо в язык КП. REAL соответствует двойной точности (мантисса 16-17 дес. цифр). Каждая такая переменная занимает 8 байт.

Весь раздел объявлений описывает **связный блок памяти** (contiguous memory block), размер которого **известен статически** (т.е. вычисляется компилятором непосредственно из текста программы).

Этот блок иногда называют *фрейм* (frame) или иногда кадр (что хуже).

В скомпилированном коде вместо имен переменных — их адреса (точнее, **относительные адреса** = смещения в байтах относительно начала фрейма, потому что абсолютный адрес начала фрейма становится известен только в момент начала выполнения процедуры).

Связные блоки памяти возникают постоянно (массивы, записи, блок глобальных переменных модуля).

Важное понятие, т.к. процессор грузит память в свой быстрый внутренний кэш целыми блоками...

Кроме объявленных переменных, компилятор может туда вставлять скрытые вспомогательные переменные для промежут. рез-тов вычислений.

Важнейшее свойство всех Оберонов: программный текст очень точно соответствует структуре машинных кодов — сугубо технические детали скрыты, но основная структура вся видна "как на ладони".
Иначе программисту трудно контролировать эффективность создаваемой программы.

Антоним для "статически" — **"динамически"**, т.е. "в процессе выполнения программы".

Чем больше информации об алгоритме известно статически (т.е. выражено прямо в тексте программы), тем лучше.

Антоним для "связный блок памяти" — **"динамическая структура данных"**, т.е. группа связанных блоков, связанных между собой указателями. Это целая отдельная поэма.

Раздел объявлений процедуры — ее **локальные переменные**.
 Локальные, потому что их можно использовать только внутри этой процедуры (т.наз. "правила видимости"; подробнее отдельно).

Где находятся ячейки памяти для x и y?
Нигде. Фрейм выделяется процедуре заново, причем, возможно, в разных местах памяти компа, при каждом ее вызове. И только в этот момент начинают существовать x и y.
 Содержимое числовых локальных переменных в новом, только что размещенном фрейме неопределено, точнее, там мусор (см. печать).

Активация процедуры = конкретный вызов, выделение фрейма локальных переменных, передача параметров (если есть), переход к выполнению инструкций ("передача управления" в тело процедуры).

Наконец:
 Текущее **значение ячейки/переменной** — это ее текущее состояние.

Выполняемая часть — тело процедуры

Первая строка:
 StdLog.Real(x); StdLog.Ln;
 две инструкции, разделяются символом ";", который не является частью инструкций.
 Каждая обозначает вызов одной из процедур модуля StdLog.

В первом случае — процедура называется Real, и инструкция StdLog.Real(x) вызывает печать в конец рабочего журнала (окошко Log) значения переменной x в некоем стандартном формате, заложенном в процедуру авторами ББ (можно написать свой вариант StdLog, который будет печатать по-другому).

818
 819 Процедура StdLog.Ln вызывает переход к началу новой строки в рабочем
 820 журнале (окошке Log).
 821
 822 Смотрим на первую строку распечатки: типичное мусорное число в "научном"
 823 формате: мы пока не задали для переменной x никакого значения.
 824 То же вторая строка — уже для y.
 825
 826 **NB** Пустая строка помогает разделить отдельные части программы.
 827 **Не жалеите пустых строк!**
 828 **Читабельность превыше всего.**
 829
 830 Инструкция $x := 3.14159$ осуществляет т.наз. **присваивание** переменной x
 831 значения, описанного справа от **оператора присваивания** " := ".
 832
 833 Слева — переменная (ячейка памяти), справа — новое значение.
 834 И то, и то может быть выражением.
 835 "Выражение" здесь в неформальном смысле;
 836 в терминах "Сообщения о языке КП" слева — "designator", т.е. по-сути
 837 выражение, значением которого является *переменная*.
 838 Такое выражение-designator может быть довольно сложным (но это не
 839 арифметическое выражение).
 840
 841 В данном случае новое значение представлено явной константой.
 842 Другой вариант: $x := y$. Тогда содержимое y будет скопировано в ячейку x.
 843 (Пример "разыменования": переменная y превращается в значение, которое она
 844 хранит.)
 845
 846 **NB** Почему мы сказали, что значение в правой части "описано", а не
 847 задано?
 848 Потому что компилятор при переводе программы в машинные коды преобразует
 849 точную десятичную дробь 3.14159 в двоичное представление — почти всегда
 850 такое преобразование делается с ошибкой. Последствия этого мы увидим в
 851 четвертой строке распечатки.
 852
 853 После присваивания переменной x следует строчка с инструкциями печати — За
 854 строка распечатки.
 855 Видим, что содержимое ячейки x изменилось так, как мы и затребовали в пред.
 856 строке.
 857 **NB** При печати процедура StdLog.Real осуществляет обратное преобразование
 858 от двоичного представления к десятичному, которое тоже обычно
 859 сопровождается ошибкой.
 860 Процедура ничего не знает про то, откуда взялось двоичное значение.
 861 Тот факт, что напечатанное значение совпало с константой в правой части
 862 присваивания — **случайность**.
 863
 864 Инструкция $y := -x * x$ является оператором присваивания (в данном случае
 865 переменной y), но присваиваемое значение описано арифметическим
 866 выражением.
 867
 868 используются значения указанных переменных;

869 порядок выполнения операций **однозначно** (в Обероне/КП) определяется
 870 **приоритетами операций** и скобками (y умножения и деления приоритет выше,
 871 чем y + и -).
 872
 873 На самом деле компилятор разбивает вычисление выражений на
 874 последовательность элементарных операций, вводя по мере необходимости
 875 скрытые переменные для промежуточных результатов.
 876
 877 Происходит примерно вот что:
 878 компилятор добавляет во фрейм (раздел объявлений) переменную z: REAL; и
 879 заменяет инструкцию $y := -x * x$ на две, чтобы применялись только очень
 880 простые переменные:
 881 $z := x * x$;
 882 $y := -z$;
 883
 884 Затем следует печать, и мы видим, что напечатанное значение содержит
 885 ошибку: точное значение равно
 886 -9.869587728100000
 887
 888 Дошли до конца процедуры — выполнение прекратилось.
 889
 890
 891 **MODULE Kurs2006Пример1;**
 892 (** вычисление формул*
 893 *выражение*
 894 *арифметич. операции для REAL — порядок, приоритет*
 895 *средства Math*
 896 **)*
 897 IMPORT Log := StdLog, Math;
 898
 899 PROCEDURE **Вычислить***;
 900 VAR pi, x, y, z: REAL; (** четыре ячейки памяти по 8 байт (REAL —*
 901 *"двойная точность") **)
 902 BEGIN
 903 Log.Real(pi); Log.Ln((* мусор *)
 904 pi := 3.14159; Log.Real(pi); Log.Ln;
 905
 906 x := pi + 2; Log.Real(x); Log.Ln;
 907
 908 x := x + 2; Log.Real(x); Log.Ln;
 909
 910 y := x + 2 - 3; Log.Real(y); Log.Ln; (** порядок операций: слева*
 911 *направо *)*
 912
 913 y := MAX(REAL); Log.Real(y); Log.Ln; (** есть и MIN **)
 914
 915 y := y + y; Log.Real(y); Log.Ln;
 916
 917 y := INF; Log.Real(y); Log.Ln;
 918
 919 x := pi;
 920 y := 2;
 921 z := x + 2 * pi; (** порядок *)* Log.Real(z); Log.Ln;
 922 z := x + (2 * pi); Log.Real(z); Log.Ln;
 923 z := (x + 2) * pi; Log.Real(z); Log.Ln;
 924

```

917      (* если сомневаетесь, ставьте скобки!!! *)
918
919      z := Math.Sin( x );      Log.Real( z ); Log.Ln;
920      z := Math.Cos( x / 3 );  Log.Real( z ); Log.Ln;
921      z := Math.Sqrt( 2 );    Log.Real( z ); Log.Ln;
922      z := Math.Sqrt( 3 ) * 2 + 3; Log.Real( z ); Log.Ln;
923
924      END Вычислить;
925  END Kurs2006Пример1.
926  ! Kurs2006Пример1.Вычислить
927  5.250617072533072E-308
928  3.14159
929  5.14159
930  7.14159
931  6.14159
932  1.797693134862316E+308
933  inf
934  inf
935  9.424769999999999
936  9.424769999999999
937  16.1527677281
938  2.653589793352726E-6
939  0.5000007660251953
940  1.414213562373095
941  6.464101615137754
942
943  NB Как узнать, что еще есть в Math?
944  Выделить (дв. клик), Ctrl+D (или Info, Client Interface):
945
946  DEFINITION Math;
947
948      PROCEDURE ArcCos (x: REAL): REAL;
949      PROCEDURE ArcCosh (x: REAL): REAL;
950      PROCEDURE ArcSin (x: REAL): REAL;
951      PROCEDURE ArcSinh (x: REAL): REAL;
952      PROCEDURE ArcTan (x: REAL): REAL;
953      PROCEDURE ArcTan2 (y, x: REAL): REAL;
954      PROCEDURE ArcTanh (x: REAL): REAL;
955      PROCEDURE Ceiling (x: REAL): REAL;
956      PROCEDURE Cos (x: REAL): REAL;
957      PROCEDURE Cosh (x: REAL): REAL;
958      PROCEDURE Eps (): REAL;
959      PROCEDURE Exp (x: REAL): REAL;
960      PROCEDURE Exponent (x: REAL): INTEGER;
961      PROCEDURE Floor (x: REAL): REAL;
962      PROCEDURE Frac (x: REAL): REAL;
963      PROCEDURE IntPower (x: REAL; n: INTEGER): REAL;
964      PROCEDURE Ln (x: REAL): REAL;
965      PROCEDURE Log (x: REAL): REAL;
966      PROCEDURE Mantissa (x: REAL): REAL;
967      PROCEDURE Pi (): REAL;

```

```

968      PROCEDURE Power (x, y: REAL): REAL;
969      PROCEDURE Real (m: REAL; e: INTEGER): REAL;
970      PROCEDURE Round (x: REAL): REAL;
971      PROCEDURE Sign (x: REAL): REAL;
972      PROCEDURE Sin (x: REAL): REAL;
973      PROCEDURE Sinh (x: REAL): REAL;
974      PROCEDURE Sqrt (x: REAL): REAL;
975      PROCEDURE Tan (x: REAL): REAL;
976      PROCEDURE Tanh (x: REAL): REAL;
977      PROCEDURE Trunc (x: REAL): REAL;
978
979  END Math.
980
981  В реальной работе чаще всего достаточно взглянуть на интерфейс модуля,
982  чтобы вспомнить, что там есть.
983
984  Более полное описание в документации: Ctrl+Shift+D или Info, Documentation.
985  Документация есть на русском.
986  NB Документацию делает программмер. (А он может оказаться ленивым.)
987
988  r, F5 --> REAL
989
990
991  Базовый тип BOOLEAN
992
993  b, F5 --> BOOLEAN
994
995  Два значения: TRUE, FALSE
996
997      VAR b: BOOLEAN;
998      ...
999      b := TRUE;
1000
1001  Пока в алгебру логики не въезжаем.
1002
1003  Операции с численными операндами с результатом BOOLEAN:
1004  < <= > >= = #
1005
1006      VAR a, b: REAL; c: BOOLEAN;
1007      ....
1008      c := a = b; (* a равно b *)
1009      c := a # b; (* a не равно b *)
1010      c := a < b; (* a меньше, чем b *)
1011      c := a > b; (* a больше, чем b *)
1012      c := a >= b; (* a больше или равно b *)
1013
1014  Далее BOOLEAN встречается в двух примерах: ввод данных (успех очередной
1015  операции считывания) и управление ходом вычисления корней квадратного
1016  уравнения.
1017
1018

```

```

1019 MODULE Kurs2006Пример2; (* ВВОД ДАННЫХ *)
1020   IMPORT Log := StdLog, Math, In := FVTsysIn;
1021
1022   PROCEDURE Выполнить*;
1023     VAR a, b, c: REAL;
1024     BEGIN
1025       In.Open; ASSERT( In.Done ); (* ass, F5 --> ASSERT( ); *)
1026       In.Real( a ); ASSERT( In.Done );
1027       In.Real( b ); ASSERT( In.Done );
1028       In.Real( c ); ASSERT( In.Done );
1029
1030       Log.String('введенные данные:'); Log.Ln;
1031       Log.Real( a ); Log.Real( b ); Log.Real( c ); Log.Ln;
1032     END Выполнить;
1033
1034 END Kurs2006Пример2.

1035 !Kurs2006Пример2.Выполнить 1 -2 1

1036 Вот что будет в логге:
1037 введенные данные:
1038 1.0 -2.0 1.0
1039
1040 Можно менять числа и кликать по левому командеру.
1041 (Следить, чтобы ничего не было выделено мышкой.)

1042 То, что за командером, до значка — поток ввода.
1043 Значок вставляется посредством Ctrl+Shift+Q (или Tools, Insert End
1044 Commander).
1045 Поток ввода можно ограничить другим командером, или концом модуля.

1046 Поток ввода можно расположить на нескольких строках (конец строки
1047 эквивалентен пробелу).

1048 В зависимости от варианта модуля In, потоком ввода может быть и выделенный
1049 мышкой фрагмент текста (т.е. выделить мышкой нужные данные для ввода и
1050 кликнуть по командеру).

1051 Можно иметь несколько командеров с разными данными — Блэкбоксу все
1052 равно, что вы ставите после модуля.
1053
1054 Модуль In (или его варианты) обеспечивает работу с таким "потоком ввода" —
1055 простой ввод прямо из текста, очень удобно.

1056   In.Open; -- открыть "поток ввода".
1057   In.Done -- переменная типа BOOLEAN, объявленная и экспортированная из
1058   модуля In. Ее значение TRUE соответствует успеху последней операции модуля
1059   In.
1060   ASSERT( ... ); -- встроенная в язык процедура проверки (assert =
1061   удостоверять): в качестве аргумента -- логическое выражение (в данном случае
1062   переменная). Если TRUE, то выполнение продолжается. Если FALSE -- "облом"
1063   без вариантов.

1064 Это очень важно: нельзя давать ошибке "распространяться".

```

```

1065 Чем раньше обнаружена ошибка,
1066 тем дешевле ее устранять.
1067
1068 Именно поэтому любые сомнения трактуются как ошибка.
1069 Все должно быть абсолютно ясно и понятно и однозначно.
1070 Поэтому

1071 Не лениться вставлять ASSERT'ы — особенно при вводе

1072 Мало ли что попадет на вход в результате недосмотра.
1073 Ведь нетрудно скопировать.

1074   In.Real( a ); -- попытка считать в переменную a вещественное число,
1075   начиная с текущей позиции в потоке ввода. Если попытка успешна (например, в
1076   данных нет ошибок), то In.Done установится в TRUE.

1077 Печать кусочка текста (string = цепочка литер):
1078   Log.String('введенные данные:');

1079 Кавычки могут быть одинарные или двойные -- но одинаковые.
1080 Между ними -- что угодно, включая одну кавычку (другую, чем охватывающие).

1081 Упр Вместо числа в потоке ввода поставить букву, посмотреть, как ББ
1082 обломится. После облома закрыть лишние окна (их разберем отдельно).

1083 MODULE Kurs2006Пример3; (* квадр. уравнение *)
1084   IMPORT Log := StdLog, Math;

1085   PROCEDURE Решить*;
1086     VAR a, b, c, d: REAL; x1, x2: REAL;
1087     BEGIN
1088       a := 1;
1089       b := -2;
1090       c := 1

1091       d := b*b - 4*a*c;

1092       IF d > 0 THEN
1093         d := Math.Sqrt( d );
1094         x1 := ( - b + d ) / 2 / a;
1095         x2 := ( - b - d ) / 2 / a;
1096         Log.String("два решения:");
1097         Log.Real( x1 ); Log.Ln;
1098         Log.Real( x2 ); Log.Ln;

1099       ELSIF d = 0 THEN
1100         Log.String("одно решение:");
1101         Log.Real( - b / ( 2 * a ) ); Log.Ln;

1102       ELSE
1103         Log.String("нет решений."); Log.Ln;
1104       END;
1105       Log.Ln;
1106     END Решить;

1107 END Kurs2006Пример2.
1108 !Kurs2006Пример2.Решить
1109

```

1110 Объявления переменных можно группировать как показано -- например, чтобы
 1111 подчеркнуть смысл.
 1112 Сначала задаем значения.
 1113 Потом вычисляем дискриминант по формуле.
 1114
 1115 Далее идет условный оператор (if, F5)
 1116 **NB** оператор = инструкция
 1117 **NB** условный оператор = оператор IF = IF-оператор
 1118
 1119 IF *любое логическое выражение* THEN
 1120 *любой набор инструкций (в т.ч. пустой)*
 1121 ELSE
 1122 *любой набор инструкций*
 1123 END;
 1124
 1125 IF THEN ELSE END = если то иначе конец
 1126 ELSIF = ELSE+IF
 1127
 1128 ЕСЛИ d больше нуля ТО
 1129 *вычислить два решения и выполнить соотв. печать*
 1130 ИНАЧЕСЛИ d равно нулю ТО
 1131 *вычислить единственное решение и выполнить соотв. печать*
 1132 ИНАЧЕ
 1133 *сообщить об отсутствии решений*
 1134 END;
 1135
 1136 В отличие от старого Паскаля:
 1137 — нет нужды в **begin end**
 1138 — нужно закрывать END'ом
 1139 — можно несколько ветвей ELSIF
 1140 — отсутствует неоднозначность интерпретации вложенных IF;
 1141 IF THEN
 1142 IF THEN
 1143
 1144 ELSE
 1145
 1146 END;
 1147 ELSE
 1148 IF THEN
 1149
 1150 ELSE
 1151
 1152 END;
 1153 END;
 1154
 1155 Каждый IF закрыт своим END'ом — следить за отступами!!!
 1156
 1157 **Упр** В первой ветви (где два решения): выделить в отдельную инструкцию
 1158 вычисление значения $1/2/a$, ввести отдельную переменную для хранения этого
 1159 значения.
 1160
 1161 Сей трюк можно рассматривать с двух точек зрения:
 1162

1163 1) Как способ более ясного выражения смысла программы — подчеркиваем, что
 1164 в двух местах используется одно и то же выражение.
 1165 В данном случае это и так видно благодаря расположению текста.
 1166
 1167 2) Как простенькую **оптимизацию** с целью экономии вычислений (вместо
 1168 повторного двойного деления просто берем значение).
 1169
 1170 Не увлекаться!!!
 1171 Процессор зачастую такие мелочи может сделать и сам, причем гораздо лучше,
 1172 и может оказаться, что мы ему только помешаем.
 1173
 1174 **Преждевременная оптимизация — большой**
 1175 **источник проблем (сложность).**
 1176
 1177 Программисту трудно априори определить, где
 1178 программа "тормозит".
 1179
 1180 Вообще оптимизация — дело тонкое и **темное**.
 1181
 1182 **Прежде всего стремиться к предельной ясности**
 1183 **программы.**
 1184
 1185 **Упр** Порешать с разными коэффициентами. (После замены чисел --
 1186 компилируем и выполняем новую версию -- Ctrl+клик.)
 1187
 1188 **Упр** Соединить две идеи: коэффициенты уравнения
 1189 вводить из потока ввода.
 1190
 1191 **Что лучше?** Небольшой модуль -- функционирует как диалог!
 1192 Быстрота компилятора и динамическая загрузка --> большая гибкость работы
 1193 (как в интерпретируемых системах).
 1194 Для более сложных задач уже можно включать ввод данных извне.
 1195
 1196 **Базовый тип INTEGER: 4 байта (32 бита)**
 1197 "базовый" потому что программист может строить свои типы.
 1198 Диапазон значений: примерно от -2 миллиарда до +2G (точнее позже).
 1199 i, F5 --> INTEGER
 1200 Аналогично типу REAL: арифметические операции + - *, но DIV, MOD вместо /.
 1201
 1202 VAR a, b, c: INTEGER;
 1203 ...
 1204 a := 12345;
 1205 b := 17;
 1206 c := a MOD b; Log.Int(c); Log.Ln;
 1207 c := a DIV b; Log.Int(c); Log.Ln;
 1208
 1209 **Упр** Написать модуль с процедурой Do, которая содержит приведенные
 инструкции.

```

1210
1211 INC( a ); -- увеличивает содержимое a на 1, то же, что и a := a + 1;
1212 DEC( a ); -- уменьшает "-"-"-"-"-"-"-"-"-"-"", то же, что и a := a - 1;
1213
1214 (Std)Log.Int( a ) -- печать целого.
1215

```

1216 Смешанные типы в выражениях

```

1217 Что происходит при
1218   VAR x, y: REAL; a: INTEGER;
1219   ...
1220   y := x + a;

```

1221 Неявное преобразование типа

```

1222   VAR x, y: REAL; a: INTEGER; вспомогат: REAL;
1223   ...
1224   вспомогат := a;
1225   y := x + вспомогат;
1226

```

1227 НЕЛЬЗЯ (компилятор возразит): a := x;

1228 Обратно можно **только явно**: a := SHORT(ENTIER(x));

```

1229 ENTIER -- встроенная функция, результат LONGINT (64 бита).
1230 SHORT превращает LONGINT в INTEGER.
1231 Пока не замораживаемся на LONGINT, просто запомним себе про
1232 SHORT( ENTIER( ... ) ).
1233 short, F5 --> SHORT; entier, F5 --> ENTIER.
1234 Встречается не часто, но встречается.

```

1235 Вещественное деление /

```

1236 Что происходит при
1237   VAR a, b: INTEGER; z: REAL;
1238   ...
1239   z := a / b;
1240

```

1241 **NB** Операция / определена в О/КП только для REAL!!!

1242 И результат у нее тоже REAL!!!

1243 Поэтому a и b сначала преобразуются в REAL, потом произойдет умножение.

1244

1245 MODULE Kurs2006Пример4; (* цикл WHILE-ПОКА *)

```

1246   IMPORT Log := StdLog, In := FVTsysIn;

1247   PROCEDURE Do*;
1248   VAR x: REAL; n: INTEGER;
1249   BEGIN
1250     (* посчитаем числа в потоке ввода *)
1251     In.Open; ASSERT( In.Done );
1252     n := 0;
1253     In.Real( x );
1254     WHILE In.Done DO (* пока прочлось ... *)
1255       INC( n );
1256       In.Real( x )
1257     END;

```

```

1258     Log.Int( n ); Log.Ln;
1259   END Do;
1260
1261 END Kurs2006Пример4.

```

1262 **!**Kurs2006Пример4.Do 1.0 2 3 4 5.8 6 7 8 9 **!**

1263 То, что стоит между WHILE и DO -- **охрана цикла**.

1264 Провалимся внутрь цикла, только если удовлетворим охрану.

1265 Типичная структура:

```

1266
1267   In.Real( x );
1268   WHILE проверка DO
1269     что-то делаем;
1270     In.Real( x );
1271   END;
1272

```

1273 Вместо In.Real(x) может быть что-то другое, но общая структура типична.

1274 Типичная ошибка -- отсутствие инструкции перехода к следующему элементу
1275 перед END --> бесконечный цикл, останавливать посредством **Ctrl+Break**.

1276

1277

1278 MODULE Kurs2006Пример5;

```

1279   IMPORT Log := StdLog, In := FVTsysIn, Math;
1280
1281   PROCEDURE Do*;
1282   VAR x, y: REAL;
1283   BEGIN
1284     (* вычислим синусы для чисел на входе *)
1285     In.Open; ASSERT( In.Done );
1286     In.Real( x );
1287     WHILE In.Done DO
1288       y := Math.Sin( x ); Log.Real( y ); Log.Ln;
1289       In.Real( x )
1290     END;
1291     Log.Ln;
1292   END Do;
1293

```

1294 END Kurs2006Пример5.

1295

1296 **!**Kurs2006Пример5.Do 1.0 2 3 4 5.8 6 7 8 9 **!**

1297

1298 **Упр** Вычислить квадраты; кубы всех чисел на входе.

1299 **Упр** Для положительных чисел на входе вычислить квадратные (кубические)
1300 корни, для неположительных -- ничего не делать (вариант: печатать
1301 сообщение).

1302 **Упр** Предполагая, что на входе только целые числа, для каждого выяснить
1303 делимость на 7 и вывести соотв. сообщение в рабочий журнал.

1304 **Упр** Избавиться от переменной y.

1305

1306

1307 MODULE Kurs2006Пример6; (* среднее *)

```

1308   IMPORT Log := StdLog, In := FVTsysIn, Math;

```

```

1309
1310 PROCEDURE Do*;
1311   VAR x, y, sum, m: REAL; n: INTEGER;
1312 BEGIN
1313   In.Open; ASSERT( In.Done );
1314   n := 0;
1315   In.Real( x );
1316   WHILE In.Done DO
1317     INC( n );
1318     In.Real( x );
1319   END;
1320   Log.String('всего чисел в потоке ввода:');
1321   Log.Int( n ); Log.Ln;
1322   ASSERT( n > 0 ); (* береженого бог бережет *)
1323
1324   In.Open; ASSERT( In.Done ); (* читаем тот же поток с начала *)
1325   sum := 0;
1326   In.Real( x );
1327   WHILE In.Done DO
1328     sum := sum + x;
1329     In.Real( x );
1330   END;
1331   m := sum / n;
1332   Log.String('среднее:');
1333   Log.Real( m ); Log.Ln;
1334 END Do;
1335
1336 END Kurs2006Пример6.
1337 !Kurs2006Пример6.Do 1 2 3 4 5 6 7 8 9 !
1338
1339 Упр Преобразовать в один цикл.
1340 Упр Также вычислить не только среднее, но и квадратичное отклонение
1341 (ошибку).
1342

```

```

1343 MODULE Kurs2006Пример7; (* процедура-функция *)
1344   IMPORT Log := StdLog, In := FVTsysIn, Math;
1345
1346   PROCEDURE Куб ( x: REAL ): REAL;
1347     VAR y: REAL;
1348     BEGIN
1349       y := x*x*x;
1350       RETURN y
1351     END Куб;
1352
1353   PROCEDURE Do*;
1354     VAR x, y: REAL;
1355     BEGIN
1356       In.Open; ASSERT( In.Done );
1357       In.Real( x );
1358       WHILE In.Done DO
1359         y := Куб( x );
1360         Log.Real( x ); Log.Real( y ); Log.Ln;
1361         In.Real( x );
1362       END;
1363   END Kurs2006Пример7.
1364 !Kurs2006Пример7.Do 1 2 3 4 5 6 7 8 9 !
1365
1366 compiling "Kurs2006Пример7"
1367   new symbol file 172 0
1368   1.0 1.0
1369   2.0 8.0
1370   3.0 27.0
1371   4.0 64.0
1372   5.0 125.0
1373   6.0 216.0
1374   7.0 343.0
1375   8.0 512.0
1376   9.0 729.0
1377
1378 Процедура Куб видна только внутри данного модуля (нет символа экспорта).
1379 Ее можно вызывать всюду в Do.
1380 Когда Do доходит до y := Куб( x ); то активизируется и начинает выполняться
1381 Куб с соответствующим значением x.
1382 x и y в процедуре Куб можно переименовать как угодно (в пределах общих
1383 правил).
1384 Две переменные y -- совершенно независимы, это разные ячейки памяти.
1385

```

```

1384 MODULE Kurs2006WirthSin; (* excerpt from a module by N.Wirth
1385 10.7.02; edited by FVT 2006-09-27 *)
1386   IMPORT Log := StdLog, Math, In := FVTsysIn;

1387   PROCEDURE Sin ( x: REAL ): REAL;
1388   CONST
1389     c1 = 6.366197723675814E-001; (*c1 = 2/pi*)
1390     p0 = 7.853981633974484E-001;
1391     p1 = -8.074551218828054E-002;
1392     p2 = 2.490394570188736E-003;
1393     p3 = -3.657620415845570E-005;
1394     p4 = 3.133616216619040E-007;
1395     p5 = -1.757149292755000E-009;
1396     p6 = 6.877100349000000E-012;
1397     q0 = 9.99999999999998E-001;
1398     q1 = -3.084251375340372E-001;
1399     q2 = 1.585434424373457E-002;
1400     q3 = -3.259918864540401E-004;
1401     q4 = 3.590859123360360E-006;
1402     q5 = -2.460945716614000E-008;
1403     q6 = 1.136381269700000E-010;
1404   (* тип констант определяется по тому, что стоит справа от равенства;
1405   сами константы "видны" (доступны) только в процедуре Sin;
1406   константы похожи на переменные, только нельзя менять их значения
1407   *)
1408   VAR n: INTEGER; y, yy, f: REAL;
1409   BEGIN
1410     y := c1 * x;
1411     n := SHORT( ENTIER( y + 0.5 ) );
1412     y := 2 * ( y - n );
1413     yy := y * y;
1414     IF ODD( n ) THEN (* ODD = нечетный, функция с логическим значением
1415     *)
1416       f := (((((q6*yy + q5)*yy + q4)*yy + q3)*yy + q2)*yy + q1)*yy + q0
1417     ELSE
1418       f := (((((((p6*yy) + p5)*yy + p4)*yy + p3)*yy + p2)*yy + p1)*yy +
1419       p0)*y
1420     END;
1421     IF ODD( n DIV 2 ) THEN f := -f END;
1422     RETURN f
1423   END Sin;

1424   PROCEDURE Do*;
1425   VAR x, y: REAL;
1426   BEGIN
1427     In.Open; ASSERT( In.Done );
1428     In.Real( x ); ASSERT( In.Done );
1429     Log.Real( x ); Log.Ln;
1430     y := Sin( x );
1431     Log.Real( y ); Log.Ln
1432   END Do;
1433
1434   END Do;
1435

```

```

1436 END Kurs2006WirthSin.
1437 ❶Kurs2006WirthSin.Do 3.14159
1438
1439 печать в Log:
1440 3.14159
1441 2.653589793265714E-6
1442
1443 Упр Попытайтесь уловить общий смысл алгоритма.
1444
1445 Упр Вычислить интеграл от синуса (Math.Sin) от 0 до 2*pi (pi получить из
1446 функции Math.Pi()) методом сумм Римана по некоторому числу точек N. Число N
1447 вводить из входного потока, результат печатать в рабочий журнал. Исследовать
1448 скорость сходимости при увеличении N. Сравнить с известной точной формулой.
1449
1450 MODULE Kurs2006Пример9; (* процедурные
1451 переменные *)
1452   IMPORT Log := StdLog, Math, In := FVTsysIn;
1453
1454   TYPE Fun = PROCEDURE ( x: REAL ): REAL; (* первый пример описания типа
1455   *)
1456   PROCEDURE Integral ( a, b: REAL; f: Fun ): REAL;
1457   (* В качестве третьего параметра можно подставлять только процедуры
1458   точно с той же сигнатурой, что описана в Fun. *)
1459   VAR sum: REAL;
1460   BEGIN
1461     sum := ( f( a ) + 2 * f( (a+b)/2 ) + f( b ) ) / 4;
1462     RETURN sum
1463   END Integral;
1464
1465   PROCEDURE Do*;
1466   VAR a, b, i: REAL;
1467   BEGIN
1468     In.Open; ASSERT( In.Done );
1469     In.Real( a ); ASSERT( In.Done );
1470     In.Real( b ); ASSERT( In.Done );
1471     ASSERT( a < b );
1472
1473     i := Integral( a, b, Math.Sin );
1474
1475     Log.Real( i ); Log.Ln
1476   END Do;
1477
1478   END Kurs2006Пример9.
1479
1480 ❶Kurs2006Пример9.Do 0 0.57❶
1481
1482 Kurs2006Пример9 unloaded
1483 0.2754867378311393
1484
1485 Упр Вставить сюда сделанный ранее алгоритм интеграла с каким-нибудь
1486 большим N.
1487

```

1488 **Итого:**
1489 Можем делать модуль с несколькими процедурами.
1490
1491 MODULE, PROCEDURE
1492
1493 REAL, INTEGER, BOOLEAN
1494 VAR, CONST
1495
1496 присваивания, выражения, сравнения
1497 IF
1498 WHILE
1499
1500 экспорт процедур
1501 импорт модулей
1502
1503 процедурные типы
1504
1505 ввод данных из "входного потока"
1506 вывод на печать в рабочий журнал (Log)
1507
1508 связный блок памяти
1509
1510 Более полно и детально -- в дальнейшем.