

С/к Введение в современное программирование (v.5.5)

Физфак МГУ. 2006/7 уч. год.

Лекция 13

Замечания

Возник дисбаланс:

— не закончены "основы" (формальные аспекты языков -- грамматики, марковская парадигма...);

— слишком мало по алгоритмам/"программированию в малом".

И то, и другое -- слишком концептуально: но нужно хотя бы начать.

Не говоря про "нормальный" материал.

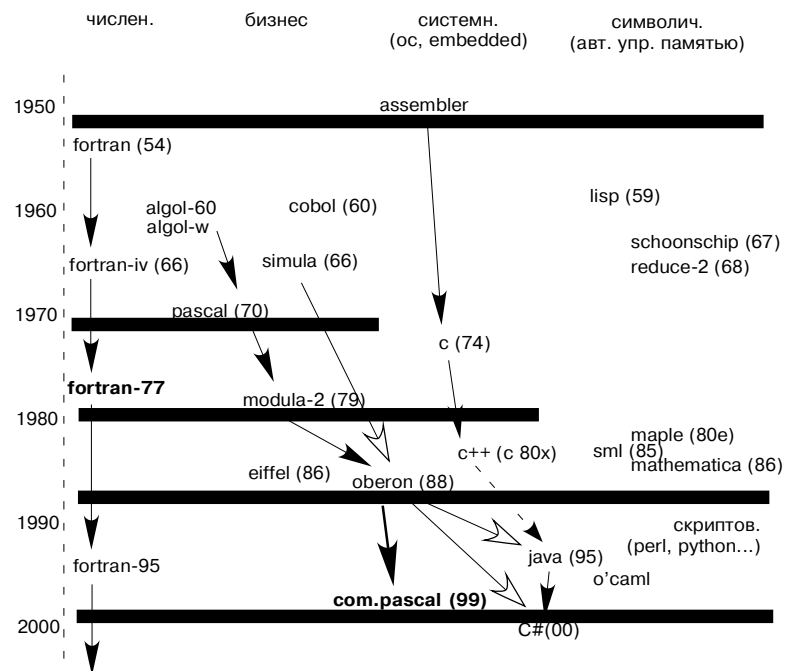
13

По поводу комментария от слушателя курса в прошлый раз:

15

Еще раз: почему именно Блэкбокс и КП

17 (печатается картинка правильно, т.е. хорошо, хотя на экране %\$#@^&*!? ... ББ
18 показывает, как составные документы *могут и должны* работать ... уж не так, как в MS
19 Windows ...)



20

21 традиционно отдельно:

22 числ. расчеты

23 аналитич. расчеты

24 упр. аппаратурой и сбор данных (системн.)

25

26		числ.	системн.	симв.(АУП)	usability
27	фортран	+	-	--	+
28	паскаль	+	-	-	+
29	модула-2	+	+	-	+
30	КП/оберон	+	+	+	+
31	ада	+	+	-	+? (сложн.)
32	функц. яз.	?	-	+	+/-
33	эйфель	+?	+?	?	+? (сложн)
34	с	+	+	-	--
35	c++	+	+	-?	---
36	Java	-	-	+	+? (сложн)
37	C#	?	?	+	+? (сложн)
38	скрипт.	-	-	+?	+

39 новые тенденции — все вместе

40 нетрадиц. числ. расчеты (эл-ты символич.)

41 нетрадиц. аналитич. расчеты (числодробилки)

42 параллельн. вычисления (системн.)

43 сложность математики и физики > острее проявляются общие требования

44 (ясность, ортогональность, систематичность, **расширяемость вместо фич**,

45 минимализм)

46 фортран умирает (последн. надежная версия fortran-77) -- впрочем, как кобол...

47 ада от паскаля с 1980 для US military, **очень** громоздкий и дорогой (\$5K/seat)

48 eiffel (автор В.Meyer) — громоздкий, через C, авто. упр. памятью не в языке

49 функц. языки — обычно проблемы с производительностью (говорят, по longer

50 за счет очень сложного оптимизир. компилятора)

51 **Размер компилятора SML: 899KB.**

52 **Размер описания SML — 93 стр. против 15-20 у О/КП.**

53 **Что касается хвальной верифицируемости функц. языков:**

54 **О свежем (дек. 2002) update для компилятора SML/NJ (с 80-х гг.) говорилось**

55 **так:**

56 Working version **110.42** available.

57 This relatively small but important update (among other things) fixes

58 a **memory leak problem** in CML ...

59 Java, скриптовые языки — проблемы производительности (фактор **10**)

60 Java, C# — сложность, производительность

61 с форума, С.Губанов (Н.Новгород, где Intel):

62 **Вопрос: "Почему <для системы образования> Component Pascal а не C#,**

63 **ведь C# - тоже безопасный язык?"**

64 **Один из возможных ответов: Сравним минимальные программы**

65 **Component Pascal:**

66 MODULE MyModule;

67 IMPORT StdLog;

68 PROCEDURE Do*;

69 BEGIN

70 StdLog.String("Здравствуй мир!")

71 END;

72 END MyModule;

73 **Все что нужно объяснить начинающему:**

```

74 1) Существуют модули (пальцем показываем на модуль)
75 2) Один модуль может импортировать другой (опять показываем пальцем)
76 3) В модулях есть процедуры, которые можно вызывать из других модулей
77 (в третий раз показываем пальцем)
78 C# (код минимального консольного приложения предлагаемый средой MS
79 Visual Studio по умолчанию):
80 using System;
81 namespace CApplication1
82 {
83     class Class1
84     {
85         [STAThread]
86         static void Main(string[] args)
87         {
88             System.Console.WriteLine("Здравствуй Мир!");
89         }
90     }
91 }
92 Что нужно объяснить начинающему:
93 Что такое namespace? (пальцем не покажешь) Что такое class? (пальцем не
94 покажешь) Что за крокозябл [STAThread]? Что такое static? Что такое void? Чем
95 метод отличается от подпрограммы (процедуры)? Почему именно Main? string[]
96 args?
97 ... Аналогично про Java ...
98 <И главное — на кой ляд это нужно?!?!?!>

99 C++
100 Linux creator Linus Torvalds: публичное заявление проектного характера на
101 http://kerneltrap.org/node/view/2067, Oct. 2004:

102 "In fact, in Linux we did try C++ once already, back in 1992. It sucks.
103 Trust me - writing kernel code in C++ is a BLOODY STUPID IDEA.
104 The fact is, C++ compilers are not trustworthy.
105 They were even worse in 1992, but some fundamental facts haven't changed:
106 1) the whole C++ exception handling thing is fundamentally broken.
107 It's especially broken for kernels.
108 2) any compiler or language that likes to hide things like memory allocations
109 behind your back just isn't a good choice for a kernel.
110 3) you can write object-oriented code (useful for filesystems etc) in C, without
111 the crap that is C++."

112 Сравнение по скорости в реалистич. физ. задаче
113 ФТ http://cern.ch/oberon.day
114 BlackBox
115 3.6±0.1 sec — со всеми проверками безопасности и отладочной инфой!
116 MS Visual C++ 6.0
117 debug
118 9.3±0.1 sec
119 full speed optimization
120 3.5±0.1 sec
121 компиляция медленнее в O(10^n) раз
122 еще без множественного наследования...
```

123 образцово сделанный язык, образцово сделанная среда
124 (**NB** не значит, что нечего улучшать)

125 Сравнение кол-ва исходного кода операционок:

OS\Code Lines	Kernel	Total
Aos	7'210	22'770
Linux 4.4BSD	58'289	202'251
Linux 2.4	420'000	2'400'000
Windows 2000	???	29'000'000
Total = Kernel + Service Support + File System + User Interface + Network		

140
141 **Aos = Active Oberon System** = версия Оберона с полной concurrency
142 одно из направлений развития Оберона, J.Gutknecht и Ко.
143 Сейчас **Bluebottle**

144
145 -----
146 БПЛА = беспилотные летательные аппараты
147 ДПЛА = дистанционно-пилотируемые летательные аппараты
148 UAV = unmanned aerial vehicle

149 **ДПЛА ГРАНТ (Россия)**

150 От 5 до 8 микроЭВМ, все объединены. Вся система управления — на Паскале.
151 Главный конструктор Н.В.Чистяков пишет (2003):

152
153 "... Меня поучают, какими языками программирования пользоваться. Это удивляет.
154 Возникает ощущение, что молодые люди думают, что до их появления на свет никто
155 ничего не знал, не умел делать, или вообще никого и ничего не было.
156 .. В 70-х было такое же размежевание между Алголом и Фортраном. Сейчас Паскаль и
157 Си. Остальные языки я не беру ввиду их малозначимости. <для встроенных систем> Я
158 даже характер людей могу предсказывать, узнав только, какой язык им больше
159 нравится. ...

160 Известны 10 преимуществ Паскаля перед Си :) Я приведу только одно, но самое важное:

161 10. На Си Вы можете написать:

```

162 for(;P("\n").R-;P("|"))for(e=3DC;e-;P("_"+(*u++/8)%2))P("|")
163 "+(*u/4)%2);
```

164 На Паскале Вы НЕ МОЖЕТЕ <это> написать.

165 .. **комплекс <БПЛА> -- это и есть программы.** В программах вся логика, мозги
166 комплекса. Аппаратура только перегоняет данные. ... Программы комплекса по своей
167 сути являются кладезем наших знаний о том, как комплекс должен работать в той
168 или иной ситуации. По мере проведения новых и новых испытаний программы
169 уточняются и проверяются вновь, с уточнёнными алгоритмами. Поэтому
170 читабельность программ **исключительно важна**. ...

171 .. если даже откажет прокатный стан (программа управления им), то его можно
172 обесточить, и с ним ничего не случится по-крупному. Если откажет ДПЛА, то он
173 может только УПАСТЬ :(В этом отличие ДПЛА даже от пилотируемого самолёта,

174 где лётчик до конца борется за спасение уникальной опытной машины. У ДПЛА конец
 175 один. Или слетал нормально, или -- дрова, стоимостью... <\$5K для младших моделей>
 176 .. Мы в ДПЛАстроении пока лучшие в мире. С Россией по научно-технической школе
 177 ДПЛА вровень стоит только Израиль. Даже США идейно слабее нас. .."

178
 179 И раз уж речь зашла о надежности программ:
 180

181 Инвариант цикла

182 Раздел "программирования в малом"

183 **NB** Ключ к корректному построению сложных циклов.

184 Конец 60-х, Эдсгер Дейкстра (**Edsger Dijkstra**)

185 **NB** Устранено главное обвинение процедурного программирования —
 186 "нематематичность".

187 **NB** Цель математики — не столько ярлыки наклеивать (что, конечно, есть
 188 довольно приятная часть...), а изучать реальность.

189 **Пример.** Умножение двух неотрицат. целых

190 Оsn. тождество: $x*y = (x'+1)*y = x'*y + y$
 191 Т.к. появляется слагаемое, введем с самого начала:
 192 $x*y + s = (x'+1)*y + y + s = x'*y + s' = x0*y0$

```
193
194 MODULE Kurs2006LoopInv0;
195   IMPORT Log := StdLog;

196   PROCEDURE Do*; (* умножение x на y *)
197     VAR x0, y0, res: INTEGER; s, x, y: INTEGER;
198     BEGIN
199       x0 := 3; y0 := 6;
200       x := x0; y := y0; s := 0;
201       ASSERT( ( x >= 0 ) & ( y >= 0 ) );
202       WHILE x > 0 DO
203         ASSERT( x*y + s = x0*y0 );
204         s := s + y;
205         x := x - 1;
206         ASSERT( x*y + s = x0*y0 );
207       END;
208       ASSERT( ( x*y + s = x0*y0 ) & ( x = 0 ) );
209       res := s;
210       ASSERT( res = x0 * y0 );
211       Log.String("pel conforto"); Log.Ln
212     END Do;
```

213 END Kurs2006LoopInv0.  Kurs2006LoopInv0.Do

214 **NB** На каждом шаге воспроизводится некое логич. тождество — **инвариант**
 215 **цикла.**

216 **Схема построения циклов:**

217 — придумать инвариант + условие завершения;
 218 — придумать шаги, приближающие к завершению ($x := x - 1$);
 219 — для каждого такого шага восстановить инвариант;
 220 — гарантировать, что для любого допустимого нач. состояния предусмотрен шаг,
 221 обязательно приближающий к окончанию цикла.

222 **Пример аккуратного рассуждения.** Хочется уменьшить число сложений.

223 Оsn. тождество: $x*y = (2*x'+x'')*y = (2*y)*x' + x''*y$

224 Т.к. появляется слагаемое, введем с самого начала:

225 $x*y + s = (2*x'+x'')*y + s = (2*y)*x' + (x''*y+s) = y'*x' + s'$

```
226 MODULE Kurs2006LoopInv1;
227   IMPORT Log := StdLog;

228   PROCEDURE Do*; (* умножение x на y *)
229     VAR x0, y0, res: INTEGER; s, x, y: INTEGER;
230     BEGIN
231       x0 := 3; y0 := 6;
232       x := x0; y := y0; s := 0;
233       ASSERT( ( x >= 0 ) & ( y >= 0 ) );
234       WHILE x > 0 DO
235         ASSERT( x*y + s = x0*y0 );
236         IF ODD( x ) THEN
237           s := s + y
238         END;
239         y := ASH( y, 1 ); (* y*2 *)
240         x := ASH( x, -1 ); (* x DIV 2 *)
241         ASSERT( x*y + s = x0*y0 );
242       END;
243       ASSERT( ( x*y + s = x0*y0 ) & ( x = 0 ) );
244       res := s;
245       ASSERT( res = x0 * y0 );
246       Log.String("pel conforto"); Log.Ln
247     END Do;
```

248 END Kurs2006LoopInv1.  Kurs2006LoopInv1.Do

249 **+Упр** Написать две аналогичные программы возведения в степень.

250 **+Упр** Определить инварианты цикла в схемах "полный просмотр" и "линейный
 251 поиск".

252

253 **Упражнения от Вирта на инварианты цикла**

254 *"... I assume that you are back in Moscow and busy with physics work. So you*
 255 *might welcome some digression into computing. If you want to present your clever*
 256 *students with some puzzles, here are two of them. I am not so sure at all whether*
 257 *they are educationally valuable. They are rather tricky and for those who love such*
 258 *things.*
 259 *The two exercises deal with a very old and simple problem: Multiply and divide*
 260 *natural numbers using addition and shifting only.*
 261 *The program assumes that the computer uses 32-bit integers, and works, if the*
 262 *operands are in the range $0 \dots 2^{15} - 1$.*
 263 *The challenge lies in finding out, why and how the algorithms work.*
 264 *I suggest that for this purpose one find the loop invariants.*
 265 *Please do not spend much time on this. But perhaps someone else might.*
 266 *How is life in Moscow these days with winter approaching? No more fires, I hope ..."*
 267 (2002-10-02)

268 **MODULE Kurs2006LoopInvWirth; (*NW 1.10.02*) (* converted by FVT 2002-**
 269 **10-08 *)**

270 **IMPORT** In := FVTsysIn (* Info21sysIn *), Log := StdLog;

271 **PROCEDURE Multiply*;**
 272 **VAR** y, z, i: INTEGER;
 273 **BEGIN**
 274 In.Open; In.Int(z); In.Int(y);
 275 Log.Int(z); Log.String(" *"); Log.Int(y);
 276 Log.String(" =");
 277 y := y * 10000H; i := 16;
 278 **REPEAT**
 279 **IF** ODD(z) **THEN** z := z + y **END**;
 280 z := z DIV 2; DEC(i)
 281 **UNTIL** i = 0;
 282 Log.Int(z); Log.Ln
 283 **END** Multiply;

284 **PROCEDURE Divide*;**
 285 **VAR** rq, y, i: INTEGER;
 286 **BEGIN**
 287 In.Open; In.Int(rq); In.Int(y);
 288 Log.Int(rq); Log.String(" /"); Log.Int(y);
 289 Log.String(" =");
 290 y := y * 10000H; i := 16;
 291 **REPEAT** DEC(i); rq := 2 * rq;
 292 **IF** rq >= y **THEN** rq := rq - y + 1 **END**
 293 **UNTIL** i = 0;
 294 Log.Int(rq MOD 10000H); Log.String(" rem");
 295 Log.Int(rq DIV 10000H); Log.Ln
 296 **END** Divide;

297 **END** Kurs2006LoopInvWirth.

298 **!**Kurs2006LoopInvWirth.Multiply 1 1
 299 **!**Kurs2006LoopInvWirth.Multiply 10 10
 300 **!**Kurs2006LoopInvWirth.Divide 11 3
 301 **!**Kurs2006LoopInvWirth.Divide 139 13
 302 Это не все про инварианты цикла... **цикл Дейкстры в весеннем семестре.**

303 **О формальных аспектах синтаксиса ЯП**304 **EBNF (РНФБ), Сообщение о языке**

305 Extended Backus Normal Formalism = Расширенный нормальный
 306 формализм Бэкуса

307 **История**

308 системы **Поста** (1936)
 309 в русле мат. логики и констр. математики
 310 описание семейств литерных строк с помощью правил порождения
 311
 312 питерский математик **А.Марков** (1954) интерпретировал как универсальный
 313 выч. механизм ("алгорифмы Маркова" = замена подцепочек символов в соотв. с
 314 заданным набором правил = программой), эквив. машине Тьюринга и
 315 рекурсивным ф-ям
 316 **марковская парадигма** — наравне с процедурной и функциональной
 317 очень полезна в
 318 обработке текстов ("**рег. выражения**"; SNOBOL-4)
 319 компьют. алгебре (**SCHOONSCHIP**, M.Veltman, CERN 1967, Нобель 1999)
 320 ряде ОО-паттернов (rider, фильтр)
 321 **NB** на "верхних" этажах прог-я парадигмы смешиваются
 322 **Hint** Если надоест отбиваться от какого-нибудь пропагандиста функциональных
 323 языков, спросите, писал ли он на марковских языках...
 324
 325 работы лингвиста Хомского (1955, 1957)
 326 формальные языки и формальные грамматики
 327 классификация грамматик ("иерархия Хомского")
 328 **Noam Chomsky**, род. 1928, профессор MIT, выдающийся лингвист и
 329 известный в США лево-радикальный (=quasi-сумасшедший) публицист
 330 анархистского толка, последователь Михаила Бакунина. (**NB** о quasi-
 331 сумасшедших.)
 332
 333 Бэкус: "окончательно" изобрел BNF ("an application of Noam Chomsky's
 334 generative grammar to formal computer languages") для описания языков прог-я
 335 (ICIP, Paris, 1959)
 336 **John Backus**, род. 1924, забавная биография до прихода в IBM, где он
 337 придумал фортран (1954-1957) — **первый** язык прог-я "высокого
 338 уровня" (т.е. не ассемблер), вместе с первым компилятором.
 339 Turing award citation:
 340 *For profound, influential, and lasting contributions to the design of practical*
 341 *high-level programming systems, notably through his work on FORTRAN,*
 342 *and for seminal publication of formal procedures for the specification of*
 343 *programming languages.*
 344
 345 Обсуждалась в комитете по Алголу-60 в 1959. Peter Naur был quasi-
 346 председателем комитета и редактором очень важной публикации —
 347 определения Алгола-60 [Naur P (ed.), 1963, Revised report on the algorithmic
 348 language Algol 60, Comm. ACM 6:1 pp1-17]. В качестве редактора (**NB** о
 349 председателях) предложил мелкие поправки — чтобы легче печатать символы
 350 на стандартной клавише. Позднее (1964) Д.Кнут заметил a technicality — что
 351 нотация Бэкуса, строго говоря, не может быть охарактеризована как normal — и
 352 предложил "Бэкус-Наур".

353 Но члены комитета по Алголу-60 отрицают важность вклада Наура — важность
354 нотации Бэкуса комитету была очевидна и т.п. (в этом контексте даже
355 употребляется выражение running into an open door = ломиться в открытые
356 двери).

357 **Мораль**

358 — к началству "прилипают" открытия (происходит само собой);
359 — энергичные ученые способны запускать "дезу" и создавать мифы;
360 — а "влиятельность" нередко основана на внешних признаках (рост, голос...)

361 **О фортране:**

362 ".. In late 1953, Backus wrote a memo asserting that <50-75%> of the operating
363 costs of a computer were from programming and testing. .."

364 ".. In those days it seemed that the only practical way is to program in assembly
365 language. The pioneers of FORTRAN didn't invent the idea of writing programs in a
366 High Level Language (HLL) and compiling the source code to object code with an
367 optimizing compiler, but they produced the first successful HLL. .."

368 ".. Grace Hopper of Remington Rand's Eckert Mauchly division had created the A-O
369 compiler, designed to do roughly the same thing. But the compiler "was clumsy and
370 ran slowly and was difficult to use," Backus says. .."

371 ".. the FORTRAN I compiler held the record for optimizing code for 20 years! .."

372 ".. The phenomenal success of the FORTRAN I team, can be attributed in part to the
373 friendly non-authoritative group climate. Another factor may be that IBM
374 management had the sense to shelter and protect the group, even though the
375 project took much more time than was first anticipated. .."

376 **(Расширенная) Нормальная Форма Бэкуса**

377 Dijkstra: ".. It turned out to have all the properties of a helpful formalism, viz.
378 compact, unambiguous and amenable to mechanical manipulation: before the end
379 of the decade the construction of parsers had been mechanized, an achievement
380 that 10 years earlier would have baffled the imagination.

381 In the case of ALGOL 60, the use of BNF has had **one regrettable effect**. Its
382 power should have been used exclusively to shorten the language definition, but it
383 made the introduction of new syntactic categories so easy that the final syntax
384 became more elaborate and more complicated than desirable. .."

385 ".. EBNF does not allow us to write anything that can't be written in BNF, it just
386 makes the grammar easier to understand .. "

387 **Зачем нужна**

388 строить

389 компилятор (структуры грамматик влияют на возможные алгоритмы
390 разбора и т.п.)

391 тулзовины (tools)

392 человеку понимать структуру (в т.ч. сообщения компилятора об ошибках)
393 программирование

394 верификация программ

395 грамматика, вообще говоря, задается неоднозначно — тоже нужно уметь еще

396 **NB** формальная грамматика для C++ была задана лишь в 4-м (??) издании
397 описания языка — неоднозначности и т.п. — как Страуструп предполагал
398 строить компилятор, уму непостижимо...

399 *Из книги Н.Вирт "Программирование на языке Модула-2" ("...
400 Оберон")*

401 ...

402 *Формальный язык — бесконечное множество цепочек символов. Элементы этого
403 множества называются предложениями языка. В случае языка
404 программирования такими предложениями являются программы. Символы
405 берутся из конечного множества, называемого словарем. Так как множество
406 программ бесконечно и не может быть задано прямым перечислением, то вместо
407 этого оно определяется правилами образования его элементов.*

408 *Последовательности символов, которые могут быть образованы в соответствии с
409 этими правилами, называют синтаксически правильными программами. Такой
410 набор правил представляет собой синтаксис языка.*

411 *Программы формального языка соответствуют грамматически правильным
412 предложениям разговорных языков. Каждое предложение имеет структуру и
413 состоит из отдельных частей, таких как подлежащее, сказуемое, дополнение.
414 Аналогично, программа состоит из частей, называемых синтаксическими
415 понятиями, таких как операторы, выражения, описания. Если грамматическая
416 конструкция A состоит из следующих друг за другом конструкций B и C, т.е. их
417 конкатенации (сцепления) BC, то мы будем называть B и C — синтаксическими
418 факторами и описывать A следующей синтаксической формулой:*

419 $A = BC.$

420 *Если же A состоит либо из B, либо из C, то мы будем называть B и C
421 синтаксическими термами и выражать A в виде:*

422 $A = B|C.$

423 *Для группировки термов и факторов можно использовать круглые скобки.*

424 *Следует заметить, что A, B и C обозначают синтаксические понятия
425 описываемого формального языка, символы равно "=", вертикальная черта "|",
426 скобки "(", ")" и точка "." — символы метанотации, называемые метасимволами.
427 Введенная здесь метанотация называется <расширенной нормальной формой
428 Бэкуса (РНФБ)>.*

429 *Кроме конкатенации и выбора РНФБ позволяет выразить условное вхождение и
430 повторение. Если конструкция A может состоять либо из B, либо из пустой
431 цепочки, то это выражается в виде*

432 $A = [B].$

433 *Если же A может состоять из конкатенации любого числа (включая нуль)
434 конструкций B, то это обозначается*

435 $A = \{B\}.$

436 **Вот мы и объяснили, что такое РНФБ.**

437 *Приведем несколько примеров того, как множества предложений описываются
438 формулами в РНФБ.*

439 $(A|B)(C|D) \rightarrow AC AD BC BD$

440 $A[B]C \rightarrow AC ABC$

441 $A\{BA\} \rightarrow A ABA ABABA \dots$

442 $\{A|B\}C \rightarrow C AC BC AAC ABC BAC BBC \dots$

443 -----конец цитаты-----

Пример

```

445 РусскоеПредложение = Модель1|Модель2|...
446 Модель1 = [Обстоятельство] Сказуемое Подлежащее
447 Обстоятельство = "вчера"|"вдали"|...
448 Сказуемое = "была"|"виднелся"|...
449 Подлежащее = "жара"|"парус"|...

```

450 порождаются такие предложения:

```

452
453     была жара
454     вчера была жара
455     виднелся парус
456     вдали виднелся жара
457     ...

```

Замечания

459 **1)** Русская грамматика не является "контекстно-свободной", но по-грубому (т.е. без учета согласования слов, падежей и т.п.) структура предложений передается.

463 К счастью в языках прог-я падежей нет ... (правда, есть C++ ...)

464 **2)** В любом случае есть смысловые (**семантические**) ограничения на "правильные по смыслу" предложения: "жара" не может "виднеться", а значение типа BOOLEAN нельзя присвоить переменной никакого другого типа.

466 **3)** Члены комбинации "правила порождения + семант. ограничения" могут одновременно варьироваться без изменения порождаемого семейства правильных предложений. Например, в С естественной синтаксич. единицей является не if, а if(, причём между if и скобкой допустимы пробелы ...

471 В сложных грамматиках (тот же C++) черт ногу сломит — откуда и несовместимость компиляторов etc.

472 **4)** Грамматика английского языка (без падежей и проч.) очень хорошо описывается таким манером (лучшие учебники) ... и вообще *"общекультурный аспект информатики"*

476 математика <> информатика <> естественные языки

478 **NB** Образовательная реформа 1871 г. (мин. обр. граф Д.А.Толстой; по идеям литератора Н.М.Каткова)

481 Важные грамматические понятия

483 Обстоятельство и т.п. — **нетерминальная** лексема — обозначает целый класс возможных конструкций из нескольких слов). Таких символов в формальной грамматике немного (см., однако, диаграмму сложности яп на прошлой лекции). "вчера" и т.п. — **терминальная** лексема (токен, token) — обозначает уже нечто более конкретное. Может быть бесконечное количество, надо как-то перечислить (отдельная задача; например, словарь).

Пример Формальные правила записи целых констант из

493 Сообщения

```

494 digit    = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
495     т.е. десятичн. цифра — это одна из перечисленных литер.

```

```

496 hexDigit = digit | "A" | "B" | "C" | "D" | "E" | "F".

```

497 т.е. 16-ричн. цифра — это десятичн. цифра или одна из перечисленных литер.

```

498 integer  = digit {digit} | digit {hexDigit} ( "H" | "L" ).

```

500 т.е. целая константа — это десятичн. цифра,
501 за которой следует любое число десятичн. цифр
502 ИЛИ десятичн. цифра,
503 за которой следует любое число десятичн. цифр
504 И (обязательно) одна из литер H или L.

506 NB В "Сообщении о языке Компонентный Паскаль" все чуть сложнее.

Пример Отрывки из Сообщения (начиная с середины раздела 2):

510 -----начало цитаты-----

511 ... Нетерминальные лексемы начинаются с большой буквы (например, Statement). Терминальные лексемы либо начинаются с маленькой буквы (например, ident), либо записаны только большими буквами (например, BEGIN), либо обозначаются цепочками литер (например, "!=").

3. Словарь и изображение

518 Изображение (терминальных) лексем посредством литер использует стандарт ISO 8859-1, т.е. расширение Latin-1 набора литер ASCII. Такие лексемы суть идентификаторы, числа, операции и ограничители. Следует соблюдать следующие лексические правила: Пробелы и концы строк не должны появляться внутри лексем (за исключением комментариев, а также пробелов в литерных цепочках). Они игнорируются, если они не нужны для разделения двух последовательных лексем. Большие и маленькие буквы различаются.

526 1. Идентификаторы суть последовательности букв, цифр и символов подчеркивания. Первая литера не должна быть цифрой.

```

529 ident = (letter | "_" ) {letter | "_" | digit}.
530 letter = "A" .. "Z" | "a" .. "z" | "A".."Ц" | "Ш".."ц" | "ш".."я".
531 digit  = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".

```

533 Примеры: x Scan Oberon2 GetSymbol firstLetter

535 < В руссифицированной версии системы Блэкбокс в идентификаторах допустимы все русские буквы кроме "ё" и "Ё".>

538 2. Числа суть целые или вещественные константы (без знака). Типом целой константы является INTEGER, если значение константы принадлежит диапазону значений типа INTEGER, или LONGINT в противном случае (см. 6.1). Если константа задана с суффиксом 'H' или 'L', представление является 16-ричным, в противном случае представление десятичное. Суффикс 'H' используется для записи 32-битных констант в диапазоне -2147483648 .. 2147483647. Разрешается не более восьми значащих 16-ричных цифр. Суффикс 'L' используется для записи 64-битных констант.

546 Вещественное число всегда содержит десятичную точку. Оно также может содержать десятичный масштабный множитель. Буква E означает "умножить на 10 в степени". Вещественное число всегда имеет тип REAL.

```

549
550 number      = integer | real.
551 integer      = digit {digit} | digit {hexDigit} ( "H" | "L" ).
552 real         = digit {digit} "." {digit} [ScaleFactor].
553 ScaleFactor  = "E" ["+" | "-"] digit {digit}.
554 hexDigit     = digit | "A" | "B" | "C" | "D" | "E" | "F".
555

```

```

556 Примеры:
557 1234567      INTEGER 1234567
558 0DH          INTEGER 13
559 12.3         REAL 12.3
560 4.567E8      REAL 456700000
561 0FFFF0000H  INTEGER -65536
562 0FFFF0000L  LONGINT 4294901760
563

```

564 3. *Литеры* [*characters*] обозначаются своим порядковым номером в 16-ричной нотации, за которым следует буква X.

```

566 character = digit {hexDigit} "X".
567

```

568 4. *Литерные цепочки* [*strings*] — последовательности литер, заключенные в одиночные (') или двойные (") кавычки. Открывающая кавычка должна всегда совпадать с закрывающей, и не содержаться внутри цепочки. Число литер в цепочке называется ее *длиной*. Цепочка длины 1 может использоваться всюду, где разрешена литерная константа, и наоборот.

```

574 string = ' ' {char} ' ' | " " {char} " ".
575

```

```

576
577 Примеры: "Component Pascal" "Don't worry!" "x"
578

```

579 5. *Операции и ограничители* [*operators and delimiters*] суть специальные литеры, пары литер или ключевые слова, перечисленные ниже. Ключевые слова содержат только большие буквы и не могут использоваться как идентификаторы.

```

583
584 + := ABSTRACT EXTENSIBLE POINTER
585 - ^ ARRAY FOR PROCEDURE
586 * = BEGIN IF RECORD
587 / # BY IMPORT REPEAT
588 ~ < CASE IN RETURN
589 & > CLOSE IS THEN
590 . <= CONST LIMITED TO
591 , >= DIV LOOP TYPE
592 ; .. DO MOD UNTIL
593 | : ELSE MODULE VAR
594 $ ELSIF NIL WHILE
595 ( ) EMPTY OF WITH
596 [ ] END OR
597 { } EXIT OUT
598

```

```

599 -----конец цитаты-----

```

```

600 -----начало цитаты-----

```

601 9. Операторы [*statements*]

602 Операторы обозначают действия. Есть элементарные и структурированные операторы. Элементарные операторы не содержат частей, которые сами являлись бы операторами. Это: присваивание, вызов процедуры, оператор возврата RETURN и оператор выхода EXIT. Структурированные операторы состоят из частей, которые сами являются операторами. Они используются для выражения последовательного, условного, выборочного и повторяющегося выполнения. Оператор может быть пустым, и в этом случае он обозначает отсутствие действия. Пустой оператор разрешен, чтобы ослабить правила пунктуации в операторных последовательностях.

```

612 Statement = [ Assignment | ProcedureCall | IfStatement | CaseStatement |
613              WhileStatement | RepeatStatement |
614              ForStatement | LoopStatement | WithStatement |
615              EXIT | RETURN [Expression] ].
616

```

```

616 -----конец цитаты-----

```

```

617 -----начало цитаты-----

```

618 9.3 Операторные последовательности

619 Операторная последовательность обозначает последовательность действий, указанных отдельными операторами, разделенными точками с запятой.

```

621 StatementSequence = Statement {";" Statement}.
622

```

```

622 -----конец цитаты-----

```

```

623

```

```

624 -----начало цитаты-----

```

625 9.4 Условный оператор IF

```

626 IfStatement =
627     IF Expression THEN StatementSequence
628     {ELSIF Expression THEN StatementSequence}
629     [ELSE StatementSequence]
630     END.
631

```

631 Оператор IF задает условное выполнение охраняемых операторных последовательностей. Логическое выражение, предшествующее операторной последовательности, называется его *охраной*. Охраны вычисляются в том порядке, в котором они встречаются в тексте, до тех пор, пока одна из них не даст значение TRUE, после чего выполняется соответствующая операторная последовательность. Если ни одна охрана не будет удовлетворена, выполняется операторная последовательность, следующая за лексемой ELSE, если таковая имеется.

639 Пример:

```

640 IF (ch >= "A") & (ch <= "Z") THEN ReadIdentifier
641 ELSIF (ch >= "0") & (ch <= "9") THEN ReadNumber
642 ELSIF (ch = "'") OR (ch = '"') THEN ReadString
643 ELSE SpecialCharacter
644 END
645

```

```

645 -----конец цитаты-----

```

646 **Упр** Прочесть Сообщение и разобраться со всеми синтаксическими правилами

```

647
648

```

649 Работа с текстами.

650 Создание текстового документа

```

651
652 Очень важная вещь, особеннов в бизнесе (но не только!) -- генерация
653 отчетов.
654 В ББ -- мощнее, чем где бы то ни было.
655
656 Данные текстовой вьюшки хранятся в отдельном объекте -- "модели":
657 TextModels.Model
658 Логически: последовательность "элементов" (литер или вьюшек; вьюшки
659 внутри вьюшек НЕ считаются).
660 У каждого "элемента" есть позиция (начиная с нуля), а также атрибуты.
661
662 Операции над такой моделью (Info, Interface):
663 TYPE
664   Model = POINTER TO ABSTRACT RECORD
665   (Containers.Model(Model.Models.Store.Store))
666   ...
667   (m: Model) Length (): INTEGER; --длина = кол-во "элементов"
668   (m: Model) Append (m0: TextModels.Model); m := m + m0
669   (m: Model) Delete (beg, end: INTEGER); --стереть end - beg "элементов"
670 от позиции beg
671   (m: Model) Insert (pos: INTEGER; m0: TextModels.Model; beg0, end0:
672   INTEGER); --"вырвать" из m0 кусок от beg0 до end0 и вставить его в m в
673   позиции pos
674   (m: Model) InsertCopy (pos: INTEGER; m0: TextModels.Model; beg0, end0:
675   INTEGER); --то же, но не "вырывать", а сделать копию этого фрагмента
676   (m: Model) Replace (beg, end: INTEGER; m0: TextModels.Model; beg0,
677   end0: INTEGER); --замена с "вырыванием"
678
679   для чтения/записи в модель два "бегунка"-rider'a -- как для файлов:
680   (m: Model) NewReader (old: TextModels.Reader): TextModels.Reader;
681   (m: Model) NewWriter (old: TextModels.Writer): TextModels.Writer;
682
683   остальное подробно не смотрим:
684   можно установить атрибуты (цвет, шрифт, размер...) для заданного
685 отрезка текста:
686   (m: Model) SetAttr (beg, end: INTEGER; attr: TextModels.Attributes)
687
688   более общо, можно запросить "свойства" заданного фрагмента текста:
689   (m: Model) Prop (beg, end: INTEGER): Properties.Property;
690   (m: Model) Modify (beg, end: INTEGER; old, p: Properties.Property);
691 END;
692
693 TextModels.Reader
694 TYPE
695   Reader = POINTER TO ABSTRACT RECORD
696   eot: BOOLEAN; --"end of text"
697   attr: TextModels.Attributes; --атрибуты прочтенного элемента
698   char: CHAR; --последняя прочитанная литера; значение
699   TextModels.viewcode, если была прочтена вьюшка, тогда см. след. поле:

```

```

700   view: Views.View; --последняя прочитанная вьюшка (если была прочтена
701   вьюшка)
702   w, h: INTEGER; --размеры последней
703   (rd: Reader) Base (): TextModels.Model;
704   (rd: Reader) Pos (): INTEGER; --текущая позиция
705   (rd: Reader) SetPos (pos: INTEGER); --встать в заданную позицию
706   (rd: Reader) Read; --прочсть очередной эл-т
707   (rd: Reader) ReadPrev; --прочсть предыдущийочередной эл-т
708   (rd: Reader) ReadChar (OUT ch: CHAR);
709   (rd: Reader) ReadPrevChar (OUT ch: CHAR);
710   (rd: Reader) ReadView (OUT v: Views.View); --найти и прочсть след.
711   вьюшку
712   (rd: Reader) ReadPrevView (OUT v: Views.View);
713   (rd: Reader) ReadRun (OUT attr: TextModels.Attributes); --найти и
714   прочсть элемент с другими атрибутами, нежели на момент вызова
715   (rd: Reader) ReadPrevRun (OUT attr: TextModels.Attributes);
716   END;
717
718 TextModels.Writer --намного проще:
719 TYPE
720   Writer = POINTER TO ABSTRACT RECORD
721   attr: TextModels.Attributes;
722   (wr: Writer) Base (): TextModels.Model;
723   (wr: Writer) Pos (): INTEGER;
724   (wr: Writer) SetPos (pos: INTEGER);
725
726   (wr: Writer) WriteChar (ch: CHAR);
727   (wr: Writer) WriteView (view: Views.View; w, h: INTEGER)
728
729   (wr: Writer) SetAttr (attr: TextModels.Attributes);
730   END;
731
732 Как задавать атрибуты (т.е. шрифт, размер и т.п.):
733
734 Текущие атрибуты: Reader.attr.
735 Атрибуты по умолчанию (то, что в новых документах): TextModels.dir.attr.
736
737 Можно плясать от атрибутов по умолчанию и модифицировать их с помощью
738 "фасадных" процедур:
739   defattr := TextModels.dir.attr;
740   newattr := TextModels.NewTypeface( defattr, "Arial" );
741   newattr := TextModels.NewColor( newattr, Ports.red );
742   и т.д.
743

```

```

744 DEFINITION TextModels;
745 ...
746 PROCEDURE NewTypeface (a: Attributes; typeface: Fonts.Typeface):
747 Attributes;
748     Fonts.Typeface = ARRAY 64 OF CHAR;
749 PROCEDURE NewColor (a: Attributes; color: Ports.Color): Attributes;
750     Ports.red ...; Ports.RGBColor( red, green, blue: INTEGER ): Ports.Color;
751 PROCEDURE NewSize (a: Attributes; size: INTEGER): Attributes;
752     size например 10*Fonts.point
753 PROCEDURE NewWeight (a: Attributes; weight: INTEGER): Attributes;
754     weight например Fonts.bold или
755 Fonts.normal
756 PROCEDURE NewStyle (a: Attributes; style: SET): Attributes;
757     например, style = {Fonts.italic}, в любой комбинации с Fonts.underline,
758 Fonts.strikeout
759 PROCEDURE NewOffset (a: Attributes; offset: INTEGER): Attributes; --
760 смещение по вертикали, в универсальных единицах ББ
761 ...
762 END TextModels.


763 Но есть еще и "сканеры/форматеры" -- так же как Files.Reader/Writer слишком
764 низкого уровня и были Stores.Reader/Writer, которые оперировали с
765 величинами более высокого логического уровня, чем байты, так и здесь доп.
766 уровень.

767 TextMappers
768     Formatter --для записи в текст чисел, целых цепочек литер и т.п.
769     Scanner --для считывания из текста чисел целиком и т.п.

770 Общая схема:

771 MODULE Kurs2006NewText;
772     IMPORT Log := StdLog, Views, StdCmds, TextModels, TextMappers,
773     TextViews;

774     PROCEDURE Do* ( IN s: ARRAY OF CHAR );
775     VAR
776         t: TextModels.Model;
777         f: TextMappers.Formatter;
778         v: TextViews.View;
779     BEGIN
780         (* создаем пустой текст *)
781         t := TextModels.dir.New();
782         (* подсоединим к нему форматтер: *)
783         f.ConnectTo( t );
784         (* пишем туда че-нить: *)
785         f.WriteString( s ); f.WriteLine;
786         (* проверим, переписывает или вставляет? *)
787         f.SetPos( 0 ); f.WriteChar("=");

788         (* создаем и открываем в отдельном окошке соотв. View *)
789         v := TextViews.dir.New( t );
790         Views.OpenView( v )(* открываем в окне *)
791     END Do;
792 END Kurs2006NewText.  "Kurs2006NewText.Do('привет, курц!')"
```

что может Formatter:

```

793
794
795 TYPE (* из модуля TextMappers *)
796     Formatter = RECORD
797         rider: TextModels.Writer;
798         (VAR f: Formatter) ConnectTo (text: TextModels.Model), NEW;
799         (VAR f: Formatter) Pos (): INTEGER, NEW;
800         (VAR f: Formatter) SetPos (pos: INTEGER), NEW;
801         (VAR f: Formatter) WriteBool (x: BOOLEAN), NEW;
802         (VAR f: Formatter) WriteChar (x: CHAR), NEW;
803         (VAR f: Formatter) WriteInt (x: LONGINT), NEW;
804         (VAR f: Formatter) WriteIntForm (x: LONGINT; base, minWidth: INTEGER;
805 fillCh: CHAR; showBase: BOOLEAN), NEW;
806         (VAR f: Formatter) WriteLn, NEW;
807         (VAR f: Formatter) WriteMsg (msg: ARRAY OF CHAR), NEW;
808         (VAR f: Formatter) WritePara, NEW;
809         (VAR f: Formatter) WriteParamMsg (msg, p0, p1, p2: ARRAY OF CHAR), NEW;
810         (VAR f: Formatter) WriteReal (x: REAL), NEW;
811         (VAR f: Formatter) WriteRealForm (x: REAL; precision, minW, expW:
812 INTEGER; fillCh: CHAR), NEW;
813         (VAR f: Formatter) WriteSString (x: ARRAY OF SHORTCHAR), NEW;
814         (VAR f: Formatter) WriteSet (x: SET), NEW;
815         (VAR f: Formatter) WriteString (x: ARRAY OF CHAR), NEW;
816         (VAR f: Formatter) WriteTab, NEW;
817         (VAR f: Formatter) WriteView (v: Views.View), NEW;
818         (VAR f: Formatter) WriteViewForm (v: Views.View; w, h: INTEGER), NEW
819     END;
820
821 из документации:
822
823 PROCEDURE (VAR f: Formatter) WriteIntForm (x: LONGINT; base, minWidth:
824 INTEGER; fillCh: CHAR; showBase: BOOLEAN)
825 Пишет целое x. Для представления числа используется цепочка цифр по
826 основанию base. Вся цепочка будет иметь не менее minWidth литер, причем
827 лишние позиции будут заполнены (если требуется) слева с помощью литеры,
828 указанной посредством fillCh. В случае недесятичного основания, можно указать,
829 чтобы основание было показано в представлении с помощью showBase.
830 Специальное значение base = charCode изображает суффикс основания "X", а
831 base = hexadecimal [шестнадцатеричные] изображает суффикс "H". Все другие
832 недесятичные основания показываются с помощью суффикса "%", за которым
833 следует десятичные цифры, изображающие значение основания. Недесятичные
834 представления отрицательных целых формируются в дополнительном виде с
835 длиной minWidth. Например, x = -3 для base = 16 и minWidth = 2 изображается
836 как "FD". Для отрицательных шестнадцатеричных чисел fillCh игнорируется и
837 вместо него используется "F".
838 Дальнейшие детали можно найти в описании процедуры Strings.IntToStringForm.
839
840 Предусловия
841 f.rider # NIL (явно не проверяется)
842 base = charCode OR base >= 2 20
843 base <= 16 21
844 minWidth >= 0 22
845
```

```

846 PROCEDURE (VAR f: Formatter) WriteRealForm (x: REAL; precision, minW, expW:
847 INTEGER; fillCh: CHAR)
848
849 Пишет вещественное x. Используется цифровая цепочка для изображения
850 числа либо в научном формате, либо в формате с фиксированной точкой. См.
851 также описание процедуры Strings.RealToStringForm.
852
853 Предусловия
854 f.rider # NIL (явно не проверяется)
855 precision > 0 20
856 0 <= minW < LEN(s) 21
857 -LEN(s) < expW <= 3 22
858
859 precision обозначает количество значащих цифр (обычно 7 для значений типа
860 SHORTREAL и 16 для REAL).
861 minW обозначает минимальную длину в литерлах. Если нужно, в начале цепочки
862 будут вставлены литеры fillCh.
863 expW > 0: экспоненциальный формат (научный) с не менее expW цифр в
864 показателе экспоненты.
865 expW = 0: формат с фиксированной или плавающей точкой, в зависимости от x.
866 expW < 0: формат с фиксированной точкой с -expW цифр после десятичной
867 точки.
868 Числа всегда округляются до последней действительной или видимой цифры.
869
870
871
872 Можно с текстом делать еще немало (выравнивания, табуляцию, замены...
873 гиперссылки, складки... вьюшки...)
874
875 Вручную: меню Text, в частности, Ctrl+J -- вставляется "линейка", все, что ниже
876 ее, форматируется по ней. Линейку можно модифицировать мышкой
877 (выравнивать влево, по центру, добавлять разные позиции табуляции и т.п.).
878 Спрятать -- Ctrl+H.
879 Все, что можно из меню и мышкой, можно и программно: TextRulers.
880 Схема работы с rulers похожа на случай атрибутов...
881
882 Наконец, созданный документ можно не открывая спасти в файл.
883 Эта функция спасения предусмотрена для любых View, поэтому искать ее надо
884 в модуле Views:
885
886 DEFINITION Views;
887 ...
888 PROCEDURE RegisterView (view: View; loc: Files.Locator; name: Files.Name);
889 ...
890 END Views.
891
892

```

Генерим, компилируем, выполняем

```

893
894
895 MODULE Kurs2006Create;
896 IMPORT Log := StdLog, TextModels, TextMappers, DevCompiler, Dialog;
897
898 PROCEDURE Compile*;
899 VAR
900     t: TextModels.Model; (* аналог File *)
901     f: TextMappers.Formatter; (* аналог Stores.Writer'a, с функциями
902 форматирования *)
903     error: BOOLEAN;
904     name: ARRAY 100 OF CHAR;
905 BEGIN
906     name := "Kurs2006CreateExample";
907     t := TextModels.dir.New();
908     f.ConnectTo( t );
909     f.WriteString("MODULE " + name + ";");
910
911     f.WriteString("IMPORT Log := StdLog;");
912
913     f.WriteString("PROCEDURE DoSomething*;");
914     f.WriteString(" BEGIN Log.String('this is the action!'); ");
915     f.WriteString("END DoSomething;");
916     f.WriteString("END " + name + ".");
917
918     DevCompiler.CompileText( t, 0, error ); ASSERT( ~error, 100 );
919 END Compile;
920
921 PROCEDURE Call*;
922 VAR res: INTEGER;
923 BEGIN
924     Dialog.Call( 'Kurs2006CreateExample.DoSomething', "", res )
925 END Call;
926
927 END Kurs2006Create. !Kurs2006Create.Compile !Kurs2006Create.Call
928 compiling "Kurs2006CreateExample"
929 new symbol file 28 0
930 this is the action!
931
932 чтобы открыть сей текст в окошке как вьюшку:
933
934 IMPORT ... Views, TextViews;
935 ...
936 VAR v: TextViews.View; (* наша первая "вьюшка" *)
937 ...
938 v := TextViews.dir.New( t ); Views.OpenView( v );
939
940

```

Более сложный пример

```

941
942
943 MODULE Kurs2006GenerateInterface;
944     IMPORT StdLog;
945
946     TYPE Proc* = POINTER TO ABSTRACT RECORD END;
947     TYPE StdProc = POINTER TO RECORD ( Proc ) END;
948
949     VAR p*: Proc; stdp: StdProc;
950
951     PROCEDURE ( p: Proc ) DoSomething* ( x: REAL ), NEW, ABSTRACT;
952
953     PROCEDURE ( p: StdProc ) DoSomething* ( x: REAL );
954     BEGIN StdLog.String("by default we do nothing!"); StdLog.Ln
955     END DoSomething;
956
957     PROCEDURE Call*;
958     BEGIN p.DoSomething( 3.14159 )
959     END Call;
960
961 BEGIN NEW( stdp ); p := stdp
962 END Kurs2006GenerateInterface.
963 ❶ Kurs2006GenerateInterface.Call
964 by default we do nothing!
965
966 MODULE Kurs2006GenerateImplementation;
967     IMPORT Log := StdLog, Interface := Kurs2006GenerateInterface;
968
969     TYPE Proc = POINTER TO RECORD ( Interface.Proc ) END;
970
971     PROCEDURE ( p: Proc ) DoSomething* ( x: REAL );
972     BEGIN Log.String("prior to compilation: "); Log.Real( 0 ); Log.Ln
973     END DoSomething;
974
975     PROCEDURE Setup*;
976     VAR p: Proc;
977     BEGIN NEW( p ); Interface.p := p
978     END Setup;
979
980 END Kurs2006GenerateImplementation.
981
982 ❶ Kurs2006GenerateImplementation.Setup
983 ❶ Kurs2006GenerateInterface.Call
984 prior to compilation: 0.0
985

```

```

986 MODULE Kurs2006GenerateCreate;
987     IMPORT Log := StdLog, Views, TextModels, TextMappers, TextViews,
988     DevCompiler, Interface := Kurs2006GenerateInterface;
989
990     PROCEDURE Do*;
991     VAR
992         t: TextModels.Model; (* уже видели эту пару *)
993         f: TextMappers.Formatter;
994         error: BOOLEAN;
995         name: ARRAY 100 OF CHAR;
996         v: TextViews.View;
997     BEGIN
998         name := "Kurs2006GenerateImplementation";
999         t := TextModels.dir.New();
1000         f.ConnectTo( t );
1001         f.WriteString("MODULE " + name + ";");
1002
1003         f.WriteString("IMPORT Log := StdLog, Interface :=
1004 Kurs2006GenerateInterface;");
1005         f.WriteString("TYPE Proc = POINTER TO RECORD ( Interface.Proc ) END;");
1006
1007         f.WriteString("PROCEDURE ( p: Proc ) DoSomething* ( x: REAL );");
1008         f.WriteString("BEGIN Log.String('after compilation we print x:');
1009 Log.Real( x ) ");
1010         f.WriteString("END DoSomething;");
1011
1012         f.WriteString("PROCEDURE Setup*;");
1013         f.WriteString("VAR p: Proc;");
1014         f.WriteString("BEGIN NEW( p ); Interface.p := p ");
1015         f.WriteString("END Setup;");
1016         f.WriteString("END " + name + ".");
1017         f.ConnectTo( NIL );
1018
1019         v := TextViews.dir.New( t ); Views.OpenView( v );
1020
1021         DevCompiler.CompileText( t, 0, error ); ASSERT( ~error, 100 );
1022
1023     END Do;
1024
1025 END Kurs2006GenerateCreate.
1026
1027 ❶ Kurs2006GenerateCreate.Do
1028 Ctrl+❶ Kurs2006GenerateImplementation.Setup
1029 ❶ Kurs2006GenerateInterface.Call
1030 after compilation we print x: 3.14159
1031
1032 MODULE Kurs2006GenerateImplementation;IMPORT Log := StdLog, Interface :=
1033 Kurs2006GenerateInterface;TYPE Proc = POINTER TO RECORD ( Interface.Proc )
1034 END;PROCEDURE ( p: Proc ) DoSomething* ( x: REAL );BEGIN Log.String('after
1035 compilation we print x:'); Log.Real( x ) END DoSomething;PROCEDURE Setup*;VAR
1036 p: Proc;BEGIN NEW( p ); Interface.p := p END Setup;END
1037 Kurs2006GenerateImplementation.

```

1038 чтбы не пропадала страничка:

1039

1040 Гиперссылки в Блэббоксе

1041 <любая команда, как после , но без общих кавычек>любой текст<>

1042 напечатаем:

1043 <StdCmds.OpenAuxDialog('Kurs2006/Rsrc/LineDialog.odc','шапка')>ВЫЗВАТЬ
1044 ДИАЛОГ<>

1045 (текст сформатирован обычными средствами),

1046 потом выберем весь этот кусок текста и выполним Tools --> Create Link (Ctrl+L)

1047 получится нечто: ВЫЗВАТЬ ДИАЛОГ, на что можно кликать (курсор при
1048 подведении мышки на этот кусок текста будет меняться, давая визуальную
1049 обратную связь), при этом будет вызываться соотв. команда.

1050

1051 **Стоять может любая команда Блэббокса!!**

1052 текст/диалог/меню с пояснениями, картинками, и такими

1053 командами

1054

1055 Типичные варианты

1056

1057 -- открыть диалог (**StdCmds.OpenAuxDialog**, см. выше)

1058

1059 -- открыть Help = гиперлинк = открыть документ в режиме браузера ("read-
1060 only")

1061 <**StdCmds.OpenBrowser**('System/Docu/User-Man', 'User
1062 Manual')>[A link to the user manual](#)<>

1063

1064 -- открыть документ для редактирования

1065 <**StdCmds.OpenDoc**('Kurs2006/Mod/Functions.odc')>[Kurs2005Functions](#)<>

1066

1067 -- вызов внешнего exe-шника:

1068

1069 <**HostDialog.Start**('notepad.exe')>[notepad](#)<>

1070

1071 Как подредактировать линк?

1072

1073 Ctrl+N — по краям появятся некие стрелочки (= объекты-"вьюшки",
1074 плавающие среди букв, в этих объектах и упрятана соотв. функциональность).

1075 Ctrl+клик по любой из них возвращает данный линк к виду с угловыми

1076 скобками — можно менять как угодно. Потом select и Ctrl+L.

1077

1078 Мощное средство.

1079 Еще: Можно генерировать программно и вставлять в автоматически
1080 генерируемый текстовый документ:

1081

1082 VAR ... link: StdLinks.Link; ...

1083

BEGIN

1084

...

1085

1086 link := **StdLinks.dir.NewLink**("StdCmds.OpenAuxDialog('Kurs2006/Rsrc/Root
1087 sDialog','Корни')");

1088

f.WriteView(link); (* линк слева *)

1089

f.WriteString("вызвать диалог для корней");

1090

link := StdLinks.dir.NewLink(""); (* линк справа *)

1091

f.WriteView(link);

1092

1093 Как обычно в Оберонах/ББ: все, что можно из меню и диалогов, можно и из
1094 программы — и наоборот...

1095

1096

1097 "Складки"

1098

1099 аналогичное средство — но для изменения куска текста.

1100

1101 Выбрать *любой кусок текста*, потом сделать Tools --> Create Fold, по краям
1102 появятся пустые стрелки (пустые стрелки = "складка раскрыта"). Клякая по
1103 стрелкам, мы "закрываем" ее — стрелки становятся черными.

1104

1105 Выбрав любую из пары стрелок (в любом состоянии) и сделав Alt+Enter,
1106 вызовем диалог, в котором этой паре стрелок можно дать label: потом в данном
1107 документе можно одновременно раскрывать/закрывать все fold'ы с одинаковым
1108 label'ом (Tools --> Fold...)

1108

Удобно использовать:

1109

— чтобы прятать длинные комментарии.

1110

— чтобы прятать отладочные коды (проверки сложных инвариантов).

1111

Программно: StdFolds.

1112

1113 Ни линки, ни складки не используют в реализации ничего, кроме механизма

1114

общего View.

1115

1116 В тексте каждая левая/правая стрелка линка/складки представляет собой один

1117

View (это чтобы считать Pos(), когда работаете с текстом).

1118