

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

Физфак МГУ. 2006/7 уч. год.

Лекция 5

Нотация про компьютерный класс

Ctrl+N = новый документ

Ctrl+O = открыть существующий

Ctrl+S = сохранить

двойной клик по имени модуля

+ **Ctrl+O** = открыть исходник

редактирование

эффективная работа

заполняем пропасть между мозгом и "камнем"

расширяем bandwidth

с др. стороны:

предельная аккуратность

максимально задействуем зрит. процессор в мозгу

--> освободить логические регистры (**их всего 7**)

лучше точнее, чем быстрее/грязнее

"оставим быстроту попугаям и электровеникам"

Кстати, "глухота" старшекурсников -- пример того, насколько неохотно люди меняют уже зафиксированные в их головах "паттерны".

мышка

для грубого попадания

выделения идентификатора (**дв. клик!!**)

клавиатура

для основной работы

двойной клик = выделение слова/идентификатора

Ctrl+X, Ctrl+C, Ctrl+V = cut, copy, insert (paste)

Home, End = начало, конец строки

Ctrl+Home, Ctrl+End = начало, конец документа

Ctrl+Z = отмена

строки программного текста (операторы) переставляются/копируются **целиком:**

мышкой попасть в строку

Home -- встать в начало

Shift+стрелка вниз -- выделить несколько строк

Ctrl+X = вырезать (cut)

мышкой встать в строку, перед которой хотим вставить

Home -- встать в начало

Ctrl+V = вставить

О синтаксисе

Где ошибка:

IF (s\$ = "M" OR s\$ = "MODULE") THEN

-- скобки вокруг логич. выражения **не нужны** ("здесь нам не Ц")

-- скобки **нужны**, чтобы разделить сравнения и OR

Синтаксис в О/КП можно и надо (будет) четко знать и понимать.

Почаще нажимать Ctrl+K

0) Как можно раньше находить ошибки -- потом будет резко дороже.

1) На каждом шаге структурно правильная программа -- пошаговое уточнение.

Принцип раннейшего обнаружения ошибок

Для всего этого и нужно прийти в компьютерный класс.

(Посмотрели BOOLEAN, SET, надо бы посмотреть INTEGER, CHAR, REAL, но чтоб не так скушно, "переставим ходы".)

Модули

Нормальных настоящих модулей нет ни в С/С++, ни в Яве/Шарпе, ни в фортране, ни в дельфи...

(Mesa), Модуля-2, Оберон, Ада (кстати: \$5K за рабочее место)

Дети не старше 5 лет: "В высоком стакане воды больше"

Старшие уже знают, что еще и ширина сосуда важна.

Но все равно впечатляются, когда чего-то **много**.

Проектировщики намеренно стремятся поразить клиента.

(Хотя новая тенденция: DviX; японская мобила без фич.)

Мальчики балдеют от "профессиональных сред" -- чтобы **"как большой"**.

При встрече с Блэкбоксом достают: **"где ехе?!", "как сделать ехе?!"**

Пример некритической фиксации "паттерна" в гипертворсприимчивом мозгу.

В подавляющем б-ве случаев (даже в коммерч. проекте) ехе не нужно делать.

Изобретение схемы работы с модулями вместо ехе -- пример 2й уровень

мышления -- критический пересмотр базовых паттернов/парадигм методом

пристального разглядывания.

NB Ничего, кроме пристального разглядывания простейших базовых вещей, тут обычно не требуется!!

NB Обычно люди не опускаются до таких "тривиальностей": разве можно проявить свой недюжинный ум на простых вещах??

Откуда взялся ехе

1978. В зале mainframe аж на 640K, инженеры, операторы.

Дисковая операционная система.

Сдается "задание" на перфокартах:

загрузить и выполнить компилятор:

входные данные -- исходник с перфокарт,

выходные данные -- объ. файл на диске.

104 (выгрузить компилятор)
 105 загрузить и выполнить линкер:
 106 ввод -- объ. файл на диске, объ. файлы библиотек
 107 вывод -- ехе на диске.
 108 (выгрузить компилятор)
 109 загрузить и выполнить ехе
 110
 111 Модель работы компа:
 112 **пакетная обработка** (batch processing)
 113
 114 Позже: несколько потоков пакетной обработки одновременно.
 115 Позже: не с колоды перфокарт, а с терминала -- но все остальное по-старому.
 116 Еще позже: комп стал персональным -- но все остальное по-старому...
 117
 118 *На дворе уж давно XXI век, а мы все занимаемся "пакетной обработкой".*
 119
 120 dynamic link libraries / shared libraries как еще одна нашивка поверх той же
 121 старой схемы.
 122
 123 Обнаружено, что в типичной работающей линукс-установке одновременно
 124 *многие десятки* копий библиотеки обработки цепочек литер (strings),
 125 подлинкованные в разные ехе и проч.

147

<i>module name</i>	<i>bytes used</i>	<i>clients</i>	<i>compiled</i>	<i>loaded</i>	Update
Epse21SysEdit 17:32:22	4978	0	2004-11-17 16:02:34	2006-10-17	
Epse21SysWindows 17:29:25	986	1	2004-02-15 18:13:32	2006-10-17	
TextCmds 17:29:30	15615	0	2004-12-29 18:47:07	2006-10-17	
TextMappers 17:29:24	10303	10	2005-06-26 02:32:34	2006-10-17	
TextControllors 17:29:24	26907	9	2004-12-29 18:47:06	2006-10-17	
TextViews 17:29:24	26799	16	2004-12-29 18:47:06	2006-10-17	
TextSetters 17:29:24	20252	17	2005-06-26 02:32:32	2006-10-17	

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

.....

152 Это обычный текстовый документ Блэкбокса, в котором по команде меню
 153 напечатан список модулей, наличествующих в системе на данный момент.
 154 *<Для полноты смысл колонок: имя модуля; размер в байтах; кол-во модулей-*
 155 *клиентов, т.е. таких, у которых данный встречается в списке IMPORT; дата и*
 156 *время компиляции; дата и время загрузки.>*
 157
 158 Первым стоит модуль, ответственный за превращение module в MODULE, т.е.
 159 вызванный при нажатии на F5. (В зависимости от конфигурации ББ вместо
 160 Epse21SysEdit может быть Info21SysEdit или CpcAllCaps и т.п.)
 161 Дважды кликнем по нему (это обычный текстовый документ ББ!!).
 162 Нажмем Ctrl+D (или из меню: Info, Client Interface) -- откроется окошко с
 163 "внешним интерфейсом" данного модуля:

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

Программирование как расширение возможностей системы

Клиент, юзающий готовую систему (кассир в банке; тетенька в ОВИРе) --

вырожденный случай.

При работе на компе программирование *лезет из всех щелей.*

Особенно здорово для инженеров и ученых, решающих свои *уникальные* задачи.

Как правило: разрабатываемые алгоритмы + несколько "ехе" -- разные варианты тестов и т.п.

Модули = строительные блоки компонентной программной системы

По-грубому:

Создаем (пишем+компилируем) модуль; вызываем процедуру из него; модуль становится частью системы к нашим дальнейшим услугам.

<ДЗ обязательно повторить следующие действия>

Предварительно: в новом окошке наберем слово module, нажмем F5.

Затем выполним команду меню Info, Loaded modules -- откроется окошко вроде:

164

165 DEFINITION Epse21SysEdit;

166

167 PROCEDURE Capitalize;

168 PROCEDURE CapitalizeExpand;

169

170 END Epse21SysEdit.

171

172 Внешний интерфейс -- то, что про модуль известно "вовне" -- всем другим модулям ББ (в т.ч. тем, которые интерпретируют команды меню).

174 **NB** Сгенерировался автоматически, кем -- Блэкбоксом (командой, привязанной к соотв. пункту меню), на основании какой информации -- по коду модуля, сидящего в памяти.

177 **NB** В Модуле-2 DEFINITION нужно было писать отдельно, в чем, конечно, нет
 178 никакого смысла (пример того, как трудно делать простые вещи -- сначала
 179 всегда почему-то все через <сложность> выходит...).

180 **NB** Это -- аналог header-файлов в Ц, где головная боль из-за возможной
 181 рассогласованности, т.к. никаких проверок... В О/КП: этой проблемы нет в
 182 принципе, **по построению**.

183

184 Для сравнения: дважды кликнув по имени модуля в любом документе ББ,
 185 нажмем Ctrl+0 (ноль) -- откроется исходник:

186

```
187 MODULE Eipse21SysEdit; (** derived from an example that came with Blackbox.  

188 **)  

189   IMPORT Ports, Strings, Fonts, Views, TextModels, TextControllers, TextViews,  

190   TextMappers;  

191  

192   VAR w: TextModels.Writer; r: TextModels.Reader; f: TextMappers.Formatter;  

193  

194   PROCEDURE CapitalizeExpand*;  

195   .....  

196   END CapitalizeExpand;  

197  

198   PROCEDURE Capitalize*;  

199   .....  

200   END Capitalize;  

201  

202 END Eipse21SysEdit.


203



204 Две процедуры без параметров (команды), помеченные символом экспорта * --  

  205 именно они и видны "наружи" (DEFINITION ...). Больше никакой информации  

  206 из модуля "с его наружи" не видно.



207



208 Кстати, смысл процедур:  

  209 смотрят на "переднее" окошко ("фокус") и пытаются там найти перед курсором  

  210 максимальную цепочку маленьких буковок. Заменяют ее на заглавные, смотрят,  

  211 что получилось, и в зависимости от этого (IF) что-то пишут в текст.



212



213 Пример модификации-перезагрузки модуля  

  214 Текст уже открыт, в процедуре CapitalizeExpand есть длинный IF-THEN-ELSIF-...-  

  215 ELSE-END.  

  216 В одной из ветвей сделать модификацию, например, строку  

  217     ELSEIF string$ = "REPEAT" THEN  

  218 заменить на  

  219     ELSEIF ( string$ = "REPEAT" ) OR ( string$ = "REP" ) THEN  

  220



221 Q В чем смысл замены?



222



223 После замены строки скомпилировать модуль Ctrl+K. На диске  

  224 скомпилированный модуль обновился.



225 Попробовать в новом окне (Ctrl+N) набрать rep, нажать F5 -- срабатывает  

  226 старая версия (получаем просто REP; сравнить, что получается, если набрать  

  227 gereat и нажать F5).



228 Это потому, что в памяти -- старая версия модуля.


```

229 Нужно ее "выгрузить" (= стереть из памяти). Тогда при попытке вызвать соотв.
 230 команду загрузится новая версия (т.к. именно она уже сидит на диске на месте
 231 старой).

232 Выгрузка модулей

233

234 — Выгрузить модуль, исходник которого в переднем окне: Dev, Unload
 235 — Выполнить команду с перезагрузкой: клик по командеру при нажатой Ctrl.

236 Проблема: Модуль не хочет выгружаться

237

238 Причина: у него есть клиенты — т.е. в памяти сидит модуль, который
 239 импортирует выгружаемый.
 240 Почему: разрабатываем некий сервисный модуль ModuleS, а тестирующие
 241 программы — в отдельном модуле ModuleT. После вызова программ модуля
 242 ModuleT: Info, Loaded Modules:

module name	bytes used	clients	compiled
* * * * *			
ModuleT	16	0	2004-10-27 12:36:01 ...
ModuleS	7	1	2004-10-27 12:35:57 ...
.....			
HostTextConv	26228	0	2001-05-22 14:10:45 ...
HostPictures	13329	0	2001-05-22 14:10:27 ...

252

253 Способы выгружать (оба годятся для случая >2 модулей):

254 А)

255 — в окошке Loaded Modules, мышкой потянуть примерно от верхней звездочки
 256 до нижней, чтобы захватились имена всех выгружаемых модулей;
 257 — нажать Alt+D, D (= из меню выполнить Dev, Unload Module List).
 258 Можно выделить имена в любом окошке, но порядок должен быть правильный.

259 Б) Командер:

260 **⚡**DevDebug.UnloadThis ModuleT ModuleS **⚡** порядок **важен!**

262 Для обслуживания длительного проекта

263 удобно завести документ Tools, в нем:

264 **⚡**DevDebug.UnloadThis ModuleT ModuleS **⚡**

265 **⚡**DevCompiler.CompileThis ModuleS ModuleT **⚡**

266 **NB** Выгружаются и компилируются в противоположном порядке.

268 В) Из программы вызвать процедуры (следить за порядком!):

269 Kernel.UnloadMod(Kernel.ThisMod("ModuleT"));

270 Kernel.UnloadMod(Kernel.ThisMod("ModuleS"));

271

272

273 **Упр** Выгрузить как можно больше модулей Блэкбокса, можно за несколько
 274 попыток.

275 В случае информационных сообщений "illegal execution" нажимать ОК.

276 Начинать с модулей, имеющих 0 клиентов.

277 **NB** **Ctrl+A** выделяет весь текст.

278

279 **Упр*** Написать и выполнить процедуру, которая выгружает некоторые модули
 280 ББ.

281

```

282 MODULE Kurs2006Пример11;
283   IMPORT Log := StdLog, In; (* In := Info21SysIn; *)
284   VAR накопитель-: REAL; счетчик-: INTEGER;
285   PROCEDURE Добавить*;
286     VAR x: REAL;
287     BEGIN
288       ASSERT( счетчик >= 0, 20 );
289       In.Open;      ASSERT( In.Done, 21 );
290       In.Real( x );  ASSERT( In.Done, 22 );
291       накопитель := накопитель + x;
292       INC( счетчик );
293     END Добавить;

294   PROCEDURE Обнулить*;
295     BEGIN накопитель := 0; счетчик := 0;
296   END Обнулить;

297   PROCEDURE ПоказатьСреднее*;
298     BEGIN
299       IF счетчик > 0 THEN
300         Log.Real( накопитель / счетчик );
301       ELSE
302         Log.Real( 0 );
303       END;
304       Log.Ln;
305     END ПоказатьСреднее;
306   BEGIN
307     Обнулить
308   END Kurs2006Пример11.

309   !Kurs2006Пример11.Добавить 243.0 !
310   !Kurs2006Пример11.ПоказатьСреднее
311   !Kurs2006Пример11.Обнулить

312   двойной клик по имени модуля, Ctrl+D:

313   DEFINITION Kurs2006Пример11;
314     VAR
315       накопитель-: REAL;
316       счетчик-: INTEGER;

317     PROCEDURE Добавить;
318     PROCEDURE Обнулить;
319     PROCEDURE ПоказатьСреднее;

320   END Kurs2006Пример11.

321   Новые элементы:
322   глобальные переменные модуля (накопитель, счетчик):
323     существуют и помнят свои значения, покуда модуль не выгружен.
324     видны из всех процедур модуля (если там не объявлено таких же -- об этом
325     позже).
326     снабжены не звездочкой, а минусом -- символ ограниченного экспорта
327     т.е. снаружи их можно читать, но нельзя изменять (read-only).
328     доступ к ним из других программ как обычно:
329   Kurs2006Пример11.накопитель

```

```

330     не обязательно снабжать их символом экспорта -- тогда они не видны
331     снаружи, например, в DEFINITION -- и составляют "внутреннее состояние
332     модуля".
333     NB Имеет место "сокрытие информации".

334   секция BEGIN в модуле
335     как обычная секция BEGIN в любой процедуре -- можно вызывать внешние
336     процедуры.
337     выполняется сразу после загрузки, один раз.
338     используется для инициализации глобальных переменных.

339   Как посмотреть текущие значения глобальных переменных (не только
340     экспортированных):
341     выделить имя модуля (двойной клик),
342     выполнить команду меню Info, Global Variables

343   Упр Посмотреть значения глобальных переменных в разных модулях ББ --
344     например, в модуле Mechanisms.

345   Модуль = глобальные переменные + процедуры
346     — единица компиляции
347     — единица загрузки
348     — основное средство инкапсуляции
349     ср. фортран, Ц...

350   Программная система = набор модулей
351     пишем новый (--> исходник),
352     компилируем (--> машинный код, спец. упаковка, на диске)
353     загрузка в память + связывание
354     — установление взаимных ссылок (экспорт/импорт)
355     — "динамическое", т.е. непосредственно в момент загрузки
356       "Look, there's no make file!"
357     — контроль типов переменных/сигнатур процедур через границы модулей
358     NB не как в DLL!

359   NB отсутствует шаг статической линковки (атавизм "систем пакетной
360     обработки")
361   NB очень важные следствия (плохо понимаются народом!)
362     (почти) не нужны:
363       make,
364       version control system (sic!)
365       нет нужды в вещах типа "пятничный билд" (как у MS)
366       не надо перекомпилировать все!
367     гибкость работы -- как в традиц. интерпретируемой системе
368     настоящие разделяемые библиотеки
369     в среде C++ code sharing — хронически обменом исходным кодом (?!)

370   Исключит. удобное средство систематического построения
371   системы — можно (и нужно) всю ОС строить на этих принципах.
372   Модули пользователя — просто добавляются к модулям системы.
373   Программирование как наращивание возможностей системы.

374   NB В Ц-образных языках: afterthought (namespaces, packages) -- и все равно
375     удобство настоящих модулей не достигается.
376

```

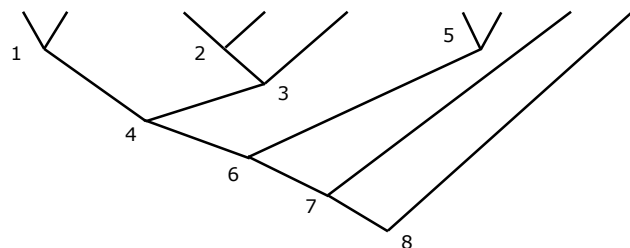
377 **Арифметика**378 **Элементарный тип INTEGER**379 **NB** Элементарный = basic. Также: простой, основной, атомарный.

380 Элементарный = неделимый в смысле базовых средств языка.

381 VAR a: INTEGER;

382 **Размер переменной:** 32 бита = 4 байта.383 **Вывод в Log:** StdLog.Int(*любое выражение с результатом типа INTEGER*),384 **Константы:** см. выше. Примеры: 12, +34, -98765. 16-ричные — для нас экзотика.386 Также: MAX(INTEGER) = $2^{31} - 1$ и MIN(INTEGER) = -2^{31} (значения зафиксированы в определении языка).388 **6 операций сравнения** =, #, <, <=, >, >= — у всех тип результата BOOLEAN.390 **5 арифметических операций** +, -, *, DIV, MOD — у всех тип результата INTEGER.392 **Приоритеты**

393 Сначала мультипликативные операции, потом аддитивные.

394 В случае равенства приоритетов, **слева направо**.395 **NB** +/- могут быть унарными операциями, но нельзя писать пару операций подряд --2, DIV + и т.п. (Здесь вам не цы!)397 **NB** Смешивать без скобок сравнения и арифм. операции — низзя! Сравнения нужно обязательно заключать в скобки.399 **Зачем** знать, что слева направо? Чтобы контролировать выход результата за рамки диапазона для INTEGER.401 **NB** В КП **все** арифм операции с целыми производятся с 32-битной точностью (если там не встретятся объявленные длинные целые).403 **NB** Но в КП результаты выражений вроде $2 * \text{MAX}(\text{INTEGER})$ или404 $\text{MIN}(\text{INTEGER}) - 1$ непредсказуемы — никаких прерываний не происходит.405 **NB** В случае сомнений расставляйте скобки — KEEP IT SIMPLE.406 **Пример** Вычисление $1 * 78 + 54 \text{ MOD } 5 \text{ DIV } 7 + 10 * 8 - 5 + 7$:
 $(((((1 * 78) + ((54 \text{ MOD } 5) \text{ DIV } 7)) + (10 * 8)) - 5) + 7)$ 407 Промежуточный результат в узле вычисляется, только когда известны оба
408 операнда.
409410 **NB** Где хранятся промежуточные значения? Компилятор сам решает это за (для)
411 нас.412 Можно ли вообще не вводить переменные? Можно (функц. языки), но крайне
413 неудобно.414
415 **Рекомендация** Ставьте пробелы вокруг операций и вокруг скобок.416 **NB** Профессионалы по клавише Space бьют большими пальцами рук.417 **Упр** Отработать движение.

418

419 ***Соглашения о знаках** для DIV, MOD в случае отрицательных операндов:
420 Сообщение, 8.2.2.

421

422 Еще встроенные функции для INTEGER с INTEGER-значениями (Сообщение,
423 10.3):

424 ABS(a) абсолютное значение

425 MIN(a, b), MAX(a, b) только два параметра

426 ASH(x, y) арифметический сдвиг: $x * 2^y$

427 Эти штуки можно использовать в выражениях.

428

429 Процедуры для **переменных** типа INTEGER:430 INC(a); по смыслу эквивалентно $a := a + 1$;431 DEC(a); по смыслу эквивалентно $a := a - 1$;

432 часто используются в циклах.

433 **NB** Здесь a должно быть переменной (например, VAR a: INTEGER;).434 **NB** На самом деле переменная может быть любая, например, вычисляемая
435 выражениями специального вида вроде сл.:

436 Fun()(Type)[arr[i]].field.AnotherFun(u)

437 но о таких переменных позже.

438

439 Редко используются

440 INC(a, n); по смыслу эквивалентно $a := a + n$;441 DEC(a, n); по смыслу эквивалентно $a := a - n$;

442

443

444 **Элементарный тип REAL**

445 VAR a: REAL;

446 **Размер переменной:** 64 бита = 8 байт.447 **Вывод в Log:** StdLog.Real(*любое выражение с результатом типа REAL*).448 **Константы:** 1.0, -2.3E-4.449 **NB** Цифры до и после точки обязательны.450 **Точность** внутреннего представления — 16-17 значащих десятичных цифр.

451 На современных процессорах: стандарт IEEE

452 (см. об этом Л.Турчак, П.Плотников, Основы численных методов, 2-е
453 изд., М., Физматлит, 2002).454 **Также** константы MAX(REAL), MIN(REAL), INF, - INF (-1/0 --> -INF).455 **Упр** Написать и выполнить простую программу, печатающую эти значения в
456 Log.457 **6 операций сравнения** =, #, <, <=, >, >= — у всех тип результата
458 BOOLEAN.459 **4 арифметических операции** +, -, *, / — у всех тип результата REAL.460 **Приоритеты**

461 Сначала мультипликативные операции, потом аддитивные.

462 В случае равенства приоритетов, **слева направо**.463 **NB** +/- могут быть унарными операциями, но нельзя писать пару операций
464 подряд --2, / + и т.п. (Здесь вам не цы!)465 **NB** Смешивать без скобок сравнения и арифм. операции — низзя! Сравнения
466 нужно обязательно заключать в скобки.

467

468 **Зачем** знать, что слева направо? Чтобы контролировать ошибки округления,
 469 переполнения и исчезновение точности.
 470 **NB** Язык гарантирует порядок вычислений — и это устраняет большую
 471 головную боль.
 472
 473 Встроенные функции для REAL с REAL-значениями (Сообщение, 10.3):
 474 ABS(a) абсолютное значение
 475 MIN(a, b), MAX(a, b) только два параметра
 476
 477 **Элементарные математические функции:** модуль Math: Math.Sin(x),
 478 Math.Cos(x), Math.Pi() и т.п. можно вставлять всюду, где можно поставить
 479 REAL-значение.
 480
 481 DEFINITION Math;
 482
 483 PROCEDURE Pi (): REAL;
 484 PROCEDURE Eps (): REAL;
 485
 486 PROCEDURE Sqrt (x: REAL): REAL;
 487 PROCEDURE Exp (x: REAL): REAL;
 488 PROCEDURE Ln (x: REAL): REAL; (* натуральный *)
 489 PROCEDURE Log (x: REAL): REAL; (* десятичный *)
 490 PROCEDURE **Power** (x, y: REAL): REAL;
 491 PROCEDURE **IntPower** (x: REAL; n: INTEGER): REAL;
 492
 493 PROCEDURE Sin (x: REAL): REAL;
 494 PROCEDURE Cos (x: REAL): REAL;
 495 PROCEDURE Tan (x: REAL): REAL;
 496 PROCEDURE ArcSin (x: REAL): REAL;
 497 PROCEDURE ArcCos (x: REAL): REAL;
 498 PROCEDURE ArcTan (x: REAL): REAL;
 499 PROCEDURE ArcTan2 (y, x: REAL): REAL;
 500
 501 PROCEDURE Sinh (x: REAL): REAL;
 502 PROCEDURE Cosh (x: REAL): REAL;
 503 PROCEDURE Tanh (x: REAL): REAL;
 504 PROCEDURE ArcSinh (x: REAL): REAL;
 505 PROCEDURE ArcCosh (x: REAL): REAL;
 506 PROCEDURE ArcTanh (x: REAL): REAL;
 507
 508 PROCEDURE Sign (x: REAL): REAL;
 509 PROCEDURE Floor (x: REAL): REAL;
 510 PROCEDURE Ceiling (x: REAL): REAL;
 511 PROCEDURE Trunc (x: REAL): REAL;
 512 PROCEDURE Frac (x: REAL): REAL;
 513 PROCEDURE Round (x: REAL): REAL;
 514
 515 PROCEDURE Mantissa (x: REAL): REAL;
 516 PROCEDURE Exponent (x: REAL): INTEGER;
 517 PROCEDURE Real (m: REAL; e: INTEGER): REAL;
 518
 519 END Math.

521 Пример описания из документации:

522
 523 PROCEDURE **Sin** (x: REAL): REAL
 524 Returns the sine of x.
 525
 526 Pre *****
 527 ABS(x) # INF 20
 528
 529 Post *****
 530 -1.0 <= result <= 1.0
 531
 532 Еще:
 533
 534 PROCEDURE **Mantissa** (x: REAL): REAL
 535 Returns the mantissa of x.
 536
 537 Post
 538 1.0 <= ABS(result) < 2.0 OR result = 0.0
 539 x = INF
 540 result = 1.0
 541 x = -INF
 542 result = -1.0
 543 x = not-a-number
 544 ABS(result) > 1.0
 545

546 **Упр** Попробовать передать в Math.Sin значение INF, а в Math.Sqrt —
 547 отрицательное число.

548 Автоматическое преобразование типов

549 VAR a: INTEGER; b: REAL;
 550
 551 Преобразование из INTEGER в REAL не приводит к потере точности.
 552 **Упр** Почему?
 553 Поэтому допустимо подмешивать целые в выражения, от которых требуется,
 554 чтобы они выдавали результат REAL:
 555 b := a;
 556 Такое присваивание — не просто копирование, а спец. машинная команда.
 557 (Обычно в два этапа: загрузить целое в "плавающий" регистр (спец. внутренняя
 558 ячейка памяти), затем скопировать из регистра в ячейку памяти b.)
 559 Другой пример:
 560 b := a / 2;
 561 В О/КП деление / определено **только** для вещественных операндов.
 562 Поэтому произойдет два преобразования примерно так:
 563 VAR a: INTEGER; b: REAL; t0, t1: REAL;
 564
 565 t0 := a; (* преобразование к вещественному *)
 566 t1 := 2; (* преобразование к вещественному *)
 567 b := t0 / t1; (* уже непосредственно деление *)
 568

569 **NB** Компилятор может пожаловаться на излишнюю сложность выражения —
 570 механически разбить на более простые куски, с учетом порядка вычисления:
 571 Пусть VAR res, x, y, z: REAL;
 572 и компилятор пожаловался на res := x + y + z;

573 Тогда вводим вспомогательную переменную для хранения промежуточного
 574 результата:
 575 VAR res, x, y, z: REAL; t: REAL;
 576 и пишем:
 577 t := x + y;
 578 res := t + z;
 579 Разумеется, саму res можно тоже использовать для промежуточных значений —
 580 но "береженого бог бережет".

581 Преобразование REAL → INTEGER

582 Потенциально опасно, поэтому в соответствии с принципами паранойи неявное
 583 преобразование такого типа запрещено.
 584 Но **если доказано**, что b: REAL хранит значение, математически эквивалентное
 585 целому из диапазона INTEGER, то преобразовать его к типу INTEGER можно с
 586 помощью сл. кракозябля:
 587 a := SHORT(ENTIER(b));

589 **Упр** Написать программу с таким кракозяблем и печатью a, и выяснить:
 590 что будет для отрицательных b (сравнить с описанием языка п.10.3),
 591 что будет при слишком больших по абс. величине значениях b.

593 **НВ В О/КП никакие** потенциально опасные неявные
 594 преобразования типов не разрешены! (*ЗДЕСЬ ВАМ*
 595 *НЕ ЦЫ!*)

596 **Упр** Почему запреты в компиляторе, а на с помощью тулзовин?
 597

598 Переменные

599 Переменная (в О/КП) = блок памяти, с которым программист связал
 600 интерпретацию (в т.ч. задал размер).
 601 Задается специальным "выражением", чаще всего -- **идентификатором**,
 602 описывающей конкретную переменную.

603 VAR a, b: INTEGER; x: REAL; y: INTEGER;

604 Разделы объявлений: в процедурах и модулях выглядят примерно так:

605
 606 **MODULE Kurs2006Пример12;**

607 (** начало раздела объявлений **)

608 VAR

609 a*, b*, c*: REAL;

610 n-: INTEGER; x1-, x2-: REAL;

611 VAR count: INTEGER;

612 (** конец раздела объявлений **)

613 **PROCEDURE Процедура1*;**

614 VAR k, l, m, n: INTEGER; (** раздел объявлений **)

615 BEGIN

616 (** доступны: локальные k, l, m, n;*

617 *глобальные a, b, c, x1, x2 **)

618 END Процедура1;

619 **PROCEDURE Процедура2;**

620 VAR m: REAL; k, l: CHAR; (** раздел объявлений **)

621 BEGIN

622 (** доступны: локальные k, l, m — другие, чем в 1й;*

623 *глобальные a, b, c, n, x1, x2 **)

624 END Процедура2;

625 BEGIN

626 (** доступны: все глобальные **)

627 END Kurs2006Пример11.  Kurs2006Пример11.Процедура1

628 Пояснения

629 связный блок памяти (contiguous memory block)

630 базовый адрес, смещения, выравнивание

631 после трансляции в маш код -- только адреса, точнее, смещения

632 обычно размер известен *статически*

633 глобальные переменные модуля

634 время существования

635 размещение (allocation)

636 инициализация в нули

637 инициализация в блоке BEGIN

638 экспорт, два вида

639 обращение к глоб. переменным другого модуля

640 "квалифицированный идентификатор"

641 **НВ** основа построения диалогов

642 Глобальные переменные объявляются в модуле перед описаниями процедур.

643 Они составляют связный блок памяти, размер которого компилер знает.

644 Размещается в памяти при загрузке модуля, существует и хранит информацию
 645 до выгрузки модуля.

646 Для их инициализации используется особая цепочка инструкций: исполняемое

647 тело модуля. Выполняется при (пере) загрузке, перед вызовом процедуры.

648 Соответственно, не выполняется, если модуль уже сидит в памяти.

649 **Способ смотреть глобальные переменные модуля:**

650 двойной клик по имени модуля,

651 меню: Info, Global Variables

652 в любой момент существования модуля

653 модуль должен быть в памяти (загружен)

654 увидим примерно сл.:

655 Kurs2006Пример12

[Update](#)

656 .a REAL 0.0

657 .b REAL 0.0

658 .c REAL 0.0

659 .count INTEGER 0

660 .n INTEGER 0

661 .x1 REAL 0.0

662 .x2 REAL 0.0

663

664 локальные переменные процедуры

665 активация процедуры

666 процедурный фрейм

667 время жизни переменной

668 разные активации одной процедуры — разные переменные

669 инициализация .. *отсутствие оной*

670 область видимости идентификатора

671 для глобальных экспортированных

672 правило для локальных

673 в общем случае нельзя сказать, из какой активации

674 Все идентификаторы, объявл. на одном уровне, должны быть разными.

675 Идентификатор, объявленный "ближе", "закрывает" другое объявление того же
676 идентификатора.

677 MODULE Kurs2006Visibility;

678 VAR x: INTEGER;

679 PROCEDURE F; (* нельзя назвать x — на одном уровне *)

680 VAR x: REAL;

681 BEGIN

682 x := 3.14 (* если бы x был INTEGER, то не компилировалось бы *)

683 END F;

684

685 END Kurs2006Visibility.

686

687 **Общее правило:** идентификаторы (переменные, константы,
688 процедуры) "видны" до окончания блока, в котором объявлены
689 (минус "перекрывающие" определения).

690

691 **"побочный эффект"** (side effect)

692 изменение значения переменной, не являющейся локальной для данной
693 процедуры

694 **NB** п. э-ты должны быть четко описаны

695 **NB** ошибочный п. э-т — очень противная вещь

696 **NB** избегать:

697 поменьше глобальных переменных

698 для локальных имена "короче", чем для глобальных

699 избегать вложенных процедур

700 избегать взаимодействия процедур посредством п. э-тов

701 **Рецепты:**

702 выделить процедур(у/ы) в отдельный модуль

703 переименовать глобальную переменную, Ctrl+K

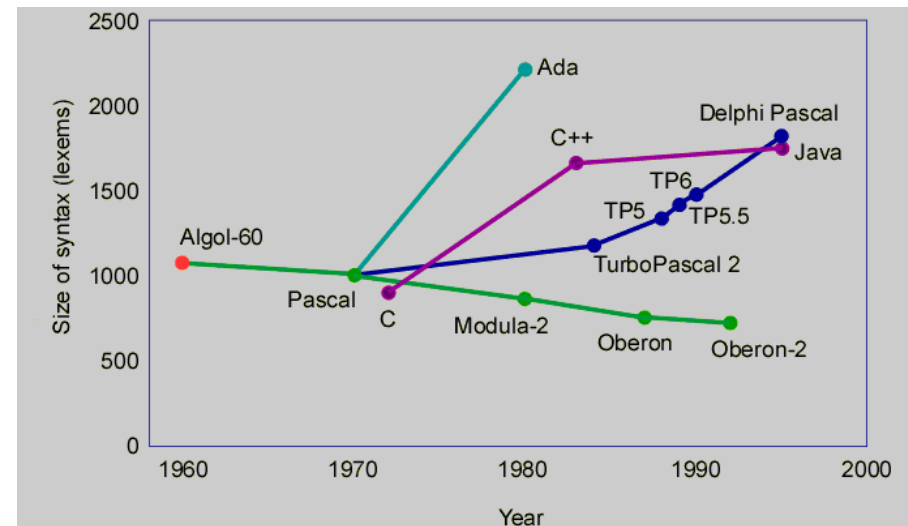


Figure 1. Complexity of syntax of programming languages

704

705 Из доклада Н.Вирта на Oberon Day @ CERN. Картинка по работе С.З.Свердлова и др.

706

707 **Задача (в честь 200-летия метода наименьших квадратов**
708 **[Лежандр])**

709 Определение параметров линейной функции по набору измерений методом
710 наименьших квадратов:

711 N пар чисел x_i, y_i — ввод с помощью In;

712 нужно найти два параметра a, b минимизацией функции

$$713 \sum_i (y_i - ax_i - b)^2$$

714 Минимум определяется из двух условий:

$$715 \frac{\partial}{\partial a} \sum_i (y_i - ax_i - b)^2 = -2 \sum_i (y_i - ax_i - b) x_i = 0$$

716 и

$$717 \frac{\partial}{\partial b} \sum_i (y_i - ax_i - b)^2 = -2 \sum_i (y_i - ax_i - b) = 0$$

718 которые дают два линейных уравнения с двумя неизвестными.

719 Рассмотрим три варианта:

720 0) a и b неизвестны;

721 1) a известно, найти b;

722 2) b известно, найти a.

723 Т.е. в случаях 1 и 2 решается только одно уравнение (которое?).

724 Построить модуль, которые содержал бы в себе средства для решения

725 уравнений во всех трех вариантах задачи,

726 а также соответствующий набор командеров.

727 Ввод делать с помощью In.