

## 1 С/к Введение в современное 2 программирование (v.5.5) Физфак МГУ. 2006/7 уч. год. 3 Лекция 8

### 4 Обычная доза метафизики

5 Почему нужно печатать вслепую:  
6 иначе в день глаза перефокусируются с экрана на клавишу **до 20К раз**.

7 В прошлой лекции:

8 **толпа -- абсолютно заразная вещь**

9 "короля играет свита"

10 безусловный рефлекс -- механизм формирования стаи

11 **+ очень опасная** в нескольких смыслах

12 не было раскрыто:

13 **эксперимент** про "когнитивный диссонанс"

14

15 "... Лучше не братья за это страшное исследование. К счастью, мы мало к тому  
16 и склонны. *Критический дух составляет высшее, очень редкое качество, между  
17 тем как подражательный ум представляет собой весьма распространенную  
18 способность: громадное большинство людей принимает без критики все  
19 установившиеся идеи, какие ему доставляет общественное мнение и передает  
20 воспитание.*"

21 Гюстав Ле Бон. Психология народов и масс. 1896.

22 **Gustave Le Bon** = знаменитый французский психолог, социолог и историк  
23 (1841-1931), бесспорный основатель социальной психологии.

24 **NB** Штука в том, что мы не можем не принимать без критики огромное  
25 количество знаний и идей...

26

27 Кстати, о визите Б.Гейтса, механизмах работы СМИ и совковых комплексах...

28

29 Просто цитата: "Глубина заблуждений не уменьшается от широты их  
30 распространенности."

31 -----

32 **Старое Упр\*** Найти ошибку в RETURN z/m в модуле Kurs2006Example13.

33 //

34 "пренебрежение" чем-то должно быть оговорено -- скажем, в виде комментария.

35

36 **Q** Кто сделал генератор бином. коэффициентов?

37 **Q** Кто сделал генератор пар простых чисел?

38

### 39 Циклы-1. Схемы "полный проход" и "линейный 40 поиск"

41 (последние задачи с композицией генераторов)

42

43 **уметь делать во сне среди ночи**

44 **уметь узнавать в задачах**

45

46 Во всех случаях подразумевается наличие линейных цепочек данных —  
47 реальных или "виртуальных" (генерируемых "на лету"). Орг-я данных может  
48 быть разная (массив, список, чтение из файла, ...).

### 49 Схема "полный проход"

#### 50 Вариации схемы "полный проход"

51 фильтрация данных

52 классификация данных

53 в обоих случаях в теле какой-нить IF.

54 *взять первый эл-т данных*

55 WHILE *взял успешно* DO

56 *что-то с ним сделать;*

57 *взять следующий эл-т данных*

58 END;

59 In.Open; In.Real( x );

60 WHILE In.Done DO

61 (\* что-то делаем с x \*)

62 In.Real( x )

63 END;

64 **NB** первый элемент часто берется по-особому (см. пару инструкции в примере).

65 **NB** типичная ошибка: забыли "взять след. эл-т" --> бесконечный цикл,

66 **NB** бесконечный цикл обычно останавливается посредством **Ctrl+Pause|Break**.

67 MODULE Kurs2006Example29;

68 IMPORT Log := StdLog, In := FVTsysIn (\* Info21sysIn\*), Math;

69 PROCEDURE **Traverse\***;

70 VAR n: INTEGER; x: INTEGER;

71 BEGIN

72 In.Open; *ASSERT( In.Done );*

73 n := 0; In.Int( x );

74 WHILE In.Done DO (\* охрана цикла \*)

75 (\* что-то сделать: \*)

76 Log.Int( x );

77 INC( n ); (\* посчитали успешно взятый \*)

78 In.Int( x ) (\* попытка "взять" следующий эл-то \*)

79 END;

80 *ASSERT( ~In.Done ); (\* отрицание охраны \*)*

81 Log.Ln; Log.String('конец.');

82 END Traverse;

83 END Kurs2006Example29.

84 **!**Kurs2006Example29.Traverse 1 2 3 -1 5 6 **!**

85 1 2 3 -1 5 6

86 конец.

87 **traverse verb, transitive**

88 **1.** To travel or pass across, over, or through.

89 **5.** To extend across; cross: a bridge that traverses a river.

90 **6.** To look over carefully; examine.

91 **9.** To survey by traverse.

92 **посчитать:**

93 — есть счетчик;

94 — в начале счетчик := 0.

95 — каждый раз, когда есть очередной подходящий, увеличить счетчик на 1.

96

97 **NB** Только кажется, что просто: сколько наблюдается циклов, где в начале 1,  
 98 потом у людей болит голова, где прибавлять 1 -- в начале тела, в конце, или  
 99 после цикла вычесть 1 -- иногда вычесть, иногда не надо.....\*?:%;Nº;";%\*?

100 **Причина** Программер слишком поглащен конкретными частями цикла, забывая  
 101 про картину в целом — *постоянная ошибка, всегда, во всех задачах — в науке,*  
 102 *в бизнесе, ....* (= неумение подниматься над уровнем 1, т.к. "картина в целом" -  
 103 - всегда абстракция).

104 **Типичные применения**  
 105 — Преобразование данных из одного формата в другой.  
 106 — Вычисление интегральных х-к (всякого рода статистика).

107 **Вариация: фильтрация**  
 108 *взять первый эл-т данных*  
 109 WHILE *взят успешно* DO  
 110 IF *эл-т удовл. свойству* THEN  
 111 *куда-то его деть (например, напечатать в Log)*  
 112 END;  
 113 *взять следующий эл-т данных*  
 114 END;

115 In.Open; In.Real( x );  
 116 WHILE In.Done DO  
 117 IF x > 0 THEN  
 118 Log.Real( x )  
 119 END;  
 120 In.Real( x )  
 121 END;

122 **Типичные применения**  
 123 — Отбор кандидатов (например, при обработке событий с детектора  
 124 космических лучей: отсеивание событий, пришедших сверху [снизу идут почти  
 125 только нейтрино]).  
 126 — Как правило, критерий отбора плохо определен (что значит, сверху?) -- надо  
 127 оптимизировать его (подбирать предельные углы для "сверху"), чтобы  
 128 максимизировать отношение сигнал/шум.

129 **Вариация: классификация**  
 130 *взять первый эл-т данных*  
 131 WHILE *взят успешно* DO  
 132 IF *эл-т удовл. свойству* THEN  
 133 *куда-то его деть (напр., просуммировать)*  
 134 ELSIF *другое свойство* THEN  
 135 *деть в другое место (напр., в другую сумму)*  
 136 .....  
 137 ELSE  
 138 *деть в куда-то еще (напр., в Log)*  
 139 END;  
 140 *взять следующий эл-т данных*  
 141 END;

142 **Применения:** классификация событий (напр., по кол-ву "адронных струй").

143 **Схема "линейный поиск"**  
 144 **Важнейший нетривиальный цикл.**  
 145 Основан на укороченном вычислении логических выражений.

146 **Задача.** Найти во входном потоке (целые числа) отрицательное.  
 147 **NB** Что значит "найти"? Знать где (+знать что) — позиция (+значение).  
 148 **Огромное количество (под)задач входит в эту категорию.**  
 149 **NB** Что есть "позиция"? смещение = offset = кол-во эл-тов **до**.  
 150 **Не** порядковый номер. **Всегда**.  
 151 *И не надо ничего больше выдумывать.*

152 MODULE Kurs2006Example30; (\* схема линейного поиска \*)  
 153 IMPORT Log := StdLog, In := FVTsysIn (\* Info21sysIn\*), Math;

154 PROCEDURE Find\*;  
 155 VAR n: INTEGER; x: INTEGER;  
 156 BEGIN In.Open; ASSERT( In.Done );  
 157 n := 0; In.Int( x );  
 158 WHILE In.Done & ~( x < 0 ) DO (\* охрана цикла \*)  
 159 INC( n ); (\* посчитали пройденный, т.е. не удовлетворяющий;  
 160 раз мы в тело цикла "провалились", значит, ~( x < 0 ) \*)  
 161 In.Int( x )  
 162 END;  
 163 (\* выполняется отрицание охраны, т.е. охрана = FALSE: \*)  
 164 ASSERT( ~In.Done OR ( x < 0 ) );


165 (\* обработка окончания: \*)  
 166 IF In.Done THEN  
 167 Log.String('кол-во неотрицат.:'); Log.Int( n ); Log.Ln;  
 168 Log.String('найденное:'); Log.Int( x ); Log.Ln;  
 169 ELSE  
 170 Log.String('не нашли, что искали.');

171 Log.Ln;  
 172 END Find;

173 END Kurs2006Example30. ① Kurs2006Example30.Find 1 2 3 -1 5 6 ①  
 174 кол-во неотрицат.: 3  
 175 найденное: -1

176 **Общая схема (знать назубок)**  
 177 *взять первый кандидат;*  
 178 WHILE (*ограничение поиска*) & ~(*условие поиска*) DO  
 179 *что-то сделать для проверенного кандидата;*  
 180 *взять очередной кандидат*  
 181 END;  
 182 (\* вышли из ограничения ИЛИ нашли \*)  
 183 ASSERT( ~(*ограничение поиска*) OR (*условие поиска*) ); --не обязательно --  
 184 *может быть дорого.*  
 185 IF *ограничение поиска* THEN  
 186 *действия при успехе*  
 187 ELSE  
 188 *действия при НЕуспехе*  
 189 END;

190

191 **NB** Порядок двух условий **важен!**  
 192 **NB** IF в конце (или присваивание BOOLEAN) **обязателен!** За  
 193 отсутствие — **расстрел.**  
 194 **NB** Успех поиска распознается по выполнению **условия ограничения!**  
 195 **NB** Применяя, четко сформулировать каждый элемент схемы.  
 196 **NB** Условия ограничения и поиска копировать (даже при  
 197 исправлениях!), ставить ~,  
 198 **избегать** преобразований  $\sim(x < 0)$  в  $(x >= 0)$ .  
 199 Глаз должен непосредственно видеть. "Спрятали" чего -- сразу шанс ошибок.  
 200  
 201 **Другой пример.** Проверить простоту заданного целого n.  
 202 Ищем делитель между 2 и корень(N)  
 203 MODULE Kurs2006Example31;  
 204 IMPORT Log := StdLog, Math;  
 205  
 206 PROCEDURE Do\*;  
 207 VAR n: INTEGER;  
 208 VAR кандидат: INTEGER; корень: REAL;  
 209 BEGIN  
 210 n := 131; Log.Int( n ); Log.Ln;  
 211 корень := Math.Sqrt( n );  
 212 кандидат := 2;  
 213 WHILE (кандидат <= корень) &  $\sim((n \text{ MOD кандидат}) = 0)$  DO  
 214 INC( кандидат )  
 215 END;  
 216 IF кандидат <= корень THEN  
 217 Log.String('есть сомножитель');  
 218 ELSE  
 219 Log.String('простое');  
 220 END;  
 221 Log.Ln  
 222 END Do;  
 223 END Kurs2006Example31.  Kurs2006Example31.Do  
 224 **Упр** Уметь воспроизводить **как дышать.**  
 225  
 226 **Полный проход + схема поиска = 98% всех циклов.**  
 227 Причина (одна из): последовательности всех сортов и последовательная  
 228 обработка -- **самый важный** элемент построения алгоритмов. (Подчеркивается  
 229 в лиспе; мы и так подчеркнем.)  
 230 **NB** Иерархия памяти (регистры; кэш; RAM; кэш диска; диск; локальная сеть  
 231 [LAN]; глобальная сеть [WAN]).  
 232 **NB** Алгоритмы, построенные на последовательной обработке,  
 233 масштабируются лучше всего.  
 234 **Пример:** "подкачка" файлов в файловом кэше ОСи. Аналогично поток данных  
 235 через TCP/IP.  
 236 **Связь с кванторами общности:**  
 237 1) вычисление логического И для однородной группы условий (логич.  
 238 выражение с квантором "для любого") — ищем первое условие, где FALSE.

237 2) вычисление логического ИЛИ для однородной группы условий (логич.  
 238 выражение с квантором "существует") — ищем первое условие, где TRUE.  
 239 **Упр** Проверить положительность всех чисел в потоке (поток ввода, через In).  
 240 явно выписать вычисляемый предикат; реализовать код.  
 241 **Упр** Проверить, что числа в потоке идут по возрастанию.  
 242 явно выписать вычисляемый предикат; реализовать код.  
 243 **Упр** Проверить, что числа в потоке идут по неубыванию.  
 244 явно выписать вычисляемый предикат; реализовать код.  
 245 **+Упр** Найти в потоке пару подряд идущих отрицательных чисел.  
 246 **+Упр** Найти в потоке пару подряд идущих простых чисел.  
 247 Начиная писать любой цикл, спросить себя: который из этих двух?  
 248 Иногда схема поиска в задачах, где речь идет не о поиске, а о полном проходе.  
 249 Например: вычислить корень для всех положительных чисел в  
 250 последовательности, причем последовательность ограничена нулем:  
 251 In.Open; ASSERT( In.Done );  
 252 In.Real( x );  
 253 WHILE In.Done &  $\sim(x \neq 0)$  DO  
 254 (\* вычисляем корень \*)  
 255 In.Real( x )  
 256 END;  
 257 (\* в таких случаях отсутствие нуля -- ошибка,  
 258 и обработка окончания вырождается в проверку: \*)  
 259 ASSERT( In.Done );  
 260 **Упр\*** Исследовать свои решения для генератора пар простых чисел на предмет  
 261 схемы поиска.  
 262 **Упр\*** Кто еще не делал — **делать!**  
 263 **Обратно:** некоторые "поиски" на самом деле суть варианты полного прохода.  
 264 **Пример** Найти максимальное значение в последовательности.  
 265 Для любого x из последовательности,  $y \geq x$ . Т.е. мы должны просмотреть все x.  
 266 PROCEDURE Max;  
 267 VAR x, y: INTEGER;  
 268 BEGIN  
 269 In.Open; ASSERT( In.Done );  
 270 In.Int( x );  
 271 WHILE In.Done DO  
 272 IF x > y THEN  
 273 y := x  
 274 END;  
 275 In.Int( x );  
 276 END;  
 277 Log.Int( y )  
 278 END Max;  
 279 **Упр** Написать программу поиска максимума и минимума:  
 280 1) за один проход по данным;  
 281 2) и чтобы поменьше сравнений.

## 282 Организация численных "поисков"

283 Организовывать по **этой же** схеме -- хотя там нет жесткого ограничения поиска  
 284 (т.к. нет прямого перебора множества -- оно слишком велико), и есть момент  
 285 условности, что есть ограничение поиска, а что -- условие поиска.

286 Пример: **Задача** Поиск корня методом деления пополам.

287 Корректное условие для начала поиска:  $a < b$ ,  $f(a)$  и  $f(b)$  имеют разные знаки.

288 Итерации:  $c := (b - a)/2$ ; в зависимости от знака  $f(c)$ , либо  $a := c$ , либо  $b := c -$   
 289 - чтобы можно было продолжить цикл.

290 Продолжать, покуда ... в матанализе условие окончания не оговаривается.

291 Варианты:

292 количество итераций не больше  $N$ ;

293  $|b - a| < \text{eps}$ ;

294  $\text{MAX}(|f(a)|, |f(b)|) < \text{eps}'$ ;

295 Что из них взять в качестве "ограничения поиска", что в качестве "условия  
 296 поиска" -- момент произвола. Зависит от задачи. Use your judgment.

297 Но раз решив -- выстроить цикл четко по схеме.

298 **Кстати** Снова видим множество возможностей. Попробовать записать их все в

299 "универсальную" процедуру -- программист замучается согласовывать все  
 300 варианты, а клиенты замучаются понимать, когда и что делать, и когда что  
 301 получается.

302 *KEEP IT SIMPLE, STUPID!*

303 Во всяком случае, "заливать" все в стандартную форму...

## 304 Пусть

305 Ограничение: число шагов (чтобы не было бесконечного цикла).

306 Условие поиска: сумма абс. значений двух границ  $<$  заданного малого числа.

307 Переводим на язык схемы поиска:

308 "Элемент" — пара  $a, b$  такая, что  $fa * fb < 0$ , либо  $fa = fb = 0$ .

309 Ищем пару  $a, b$  такую, что оба значения  $fa$  и  $fb$  малы ( $\text{feps}$ );

310 Ограничение: количество сделанных шагов не больше  $N$ .

311 Весь шаг цикла — построение и переход к очередной паре методом деления  
 312 пополам.

313 MODULE Kurs2006Example32;

314 IMPORT Log := StdLog, Math;

315 PROCEDURE Fun ( x: REAL ): REAL;

316 VAR res: REAL;

317 BEGIN

318 res := Math.Sin( x ) + 0.1;

319 RETURN res

320 END Fun;

321 PROCEDURE Do\*;

322 VAR a, b, c, fa, fb, fc: REAL; n, N: INTEGER; feps: REAL;

323 BEGIN

324 N := 100; feps := 1.0E-6;

325 a := -1; fa := Fun( a );

326 b := +1; fb := Fun( b );

327 ASSERT( a < b, 20 ); ASSERT( fa \* fb < 0, 21 );

328 n := 0; (\* количество уже сделанных шагов \*)

329 WHILE ( n < N ) & ~( ABS( fa ) + ABS( fb ) < feps ) DO

330 INC( n );

331 c := ( a + b ) / 2; fc := Fun( c );

332 IF fc = 0 THEN

333 a := c; b := c; fa := fc; fb := fc;

334 ELSIF fa \* fc < 0 THEN

335 b := c; fb := fc;

336 ELSE

337 a := c; fa := fc;

338 END;

339 END;

340 IF n < N THEN

341 Log.Real( a ); Log.Real( fa ); Log.Ln;


342 Log.Real( b ); Log.Real( fb ); Log.Ln;

343 ELSE

344 Log.String("not found."); Log.Ln;

345 END;

346 END Do;

347 EOLN Kurs2006Example32.  Kurs2006Example32.Do

348 -0.1001682281494141 -8.029427443253441E-7

349 -0.1001672744750977 1.45951188104409E-7

350 **Упр** Уметь воспроизводить **как дышать**.

351 **Упр** Переделать для более сложного ограничения поиска (  $n < N$  ) & (  $\text{ABS}( a -$   
 352  $b ) < \text{eps}$  ).

353 **Упр** Кто не хочет корни искать, пусть ищет минимум.

354 *Профессионализм - это отсутствие лишних движений*  
 355 *при выполнении четко поставленной задачи.*

## 356 Циклы REPEAT, FOR, LOOP

357 REPEAT, FOR — для выразительности в частных случаях.

358 LOOP — для оптимизации; для Дейкстра-циклов "общего вида".

### 359 Цикл REPEAT

360 REPEAT

361 тело цикла -- последовательность операторов

362 UNTIL условие окончания;

363 ASSERT( условие окончания );

364 **NB** Почти эквивалентен циклу WHILE, отличия больше косметические.

365 Единственное существенное отличие: тело выполняется **хотя бы один** раз.

366 **NB** Достоинство в плане читабельности:

367 условие окончания выглядит ровно так, как после цикла (без  $\sim$ ).

### 368 Цикл FOR

369 Пока только информация на будущее.

370 в О/КП определяется через WHILE

371 FOR i := xx [BY zz] TO yy DO

372 тело

373 END;

```

374 для положит zz:
375   i := xx;
376   WHILE i <= yy DO
377     тело;
378     i := i + zz
379   END;
380   ASSERT( i = yy + 1 );
381 Никаких ограничений (напр., на знак zz).
382 NB i = "управляющая переменная" цикла.
383 NB после выхода из FOR значение i определено!!!! (не как в старом Паскале)

384 Где полезен
385 Циклы при работе с матрицами, решетками и т.п.
386   FOR i := 0 TO N - 1 DO
387     FOR j := 0 TO M - 1 DO
388       .....
389     END;
390   END;

391 ***Цикл LOOP
392 Для оптимизации WHILE со схемой поиска (актуально в вычислит.
393 приложениях).
394 NB Для информации только. Опасен при неконтрольном
395 использовании -- м.б. трудно верифицировать программу.
396 Избегать!!!

397 Оператор LOOP
398   LOOP
399     любая последовательность операторов
400   END;

401 Спец. оператор EXIT выбрасывает на конец (ближайшего объемлющего) цикла-
402 LOOP.
403 Соглашение Всегда его писать болдом! (нелокальная "передача управления")
404 Ситуация: условие поиска -- сложное, вычисление длинное. Промежуточные
405 результаты понадобятся в теле цикла.
406   взять первый кандидат;
407   WHILE (о.п.) & ~(у.п.) DO
408     взять очередной кандидат
409   END;
410 эквивалентно:
411   взять первый кандидат;
412   LOOP
413     IF ~( ограничение поиска ) THEN EXIT END;
414     IF условие поиска THEN EXIT END;
415     взять очередной кандидат
416   END;
417   (* обработка окончания цикла: *)
418   IF ограничение поиска THEN
419     действия при успехе
420   ELSE
421     действия при НЕуспехе
422   END;

```

```

423 Теперь можно запомнить промежут. вычисления:
424   взять первый кандидат;
425   LOOP
426     вычислить условие ограничения; (*****
427     IF ~( ограничение поиска ) THEN EXIT END;
428     вычислить условие поиска; (*****
429     IF условие поиска THEN EXIT END;
430     взять очередной кандидат
431   END;
432   (* обработка окончания цикла: *)
433   IF ограничение поиска THEN
434     действия при успехе
435   ELSE
436     действия при НЕуспехе
437   END;

438 NB Не использовать для "оптимизации" трив. сложения и т.п.
439 (возможно, будет только хуже: ведь процессор пытается сам че-то
440 оптимизировать).
441 NB оптимизации всегда ухудшают читабельность --> понимание.
442 NB Обычная ошибка начинающих программистов -- преждевременная
443 оптимизация. Вся дальнейшая работа сильно затрудняется -- ведь структура
444 программы/алгоритмов еще меняется.
445 NB Программисты в общем плохо определяют, где нужно оптимизировать --
446 только через профайлинг! "But not yet..."
447 NB Ваши "оптимизации" чаще всего демонстрируют, что вы еще чайники в
448 программировании.

449 Массивы-1
450 Записи = агрегаты из (а) фиксированного числа (б) индивидуально
451 поименованных полей (переменных):
452   Complex = RECORD x, y: REAL END;
453 Обращение к полям с помощью спец. селектора: напр., с.x — переменная типа
454 REAL.
455 Как агрегировать большое число полей?
456 Простейший случай: есть однородность: напр., все поля REAL или INTEGER (или
457 Complex).
458 Рассмотрим такое объявление типа:
459   TYPE FancyVector = ARRAY 26 OF REAL; (* array = массив *)
460 И потом объявление локальной или глобальной переменной:
461   v: FancyVector;
462 Связный блок памяти из 26 поставленных подряд переменных REAL, длина
463 SIZE(REAL) * 26.
464 NB длина = только константа или константное выражение (в будущем: м.б.
465 отсутствие длины).

466   Константное выражение типа INTEGER: то, что может вычислить уже
467   компилятор, например, 52 MOD 2 или 25+один, где один — константа
468   (CONST один = 1;).
469 Длину массива всегда можно узнать с помощью псевдо-функции LEN( v ):
470   INTEGER.

```

471 В данном случае REAL — **тип элементов** данного массива.  
 472 Вместо REAL может быть "заказан" INTEGER, ... почти любой тип ("игра в  
 473 кубики"; ниже).

474 Итак: **массив ~ длина + тип эл-тов.**

475 Как обратиться к отд. составляющим этого **агрегата**:  $v[n]$ , где  $n$  — любое  
 476 выражение типа INTEGER со значением в сегменте  $[0, \text{LEN}(v) - 1]$ .

477 **NB** привыкнуть к этой комбинации:  $\text{LEN}(v) - 1$ , встречается постоянно.  
 478  $v[n]$  — переменная типа REAL.  
 479 Можно использовать **всюду**, где нужна переменная (ячейка памяти) данного  
 480 типа.

481 **Почему эл-ты нумеруются от 0**

482 Адресация:  
 483 переменная после компиляции = адрес.  
 484  $\text{ADR}(v[i]) = \text{ADR}(v[0]) + i * \text{SIZE}(\text{отд. эл-та } v)$ .

485 Короче: индекс эл-та  $\propto$  расстоянию от начала блока памяти, соотв-го массиву.  
 486 0) в программировании начинать с нуля наиболее естественно с логич. т.зр.  
 487 (Дейкстра, Дисциплина прог-я).  
 488 1) Если с 1, то всюду в маш. кодах будет pesky +/- 1.

489 **Проход по массиву:**

490 цикл WHILE:  
 491  $i := 0;$   
 492 WHILE  $i < \text{LEN}(v)$  DO (\*действия\*); **INC(i)** END;

493 цикл FOR:  
 494 FOR  $i := 0$  TO  $\text{LEN}(v) - 1$  DO (\*действия\*) END; **ASSERT(  $i = \text{LEN}(v)$  );**

495 **Упр** Написать проходы по массиву в обратном порядке с помощью WHILE и FOR.  
 496 **Упр** Написать модуль для вычислений с евклидовыми векторами Euclid = ARRAY  
 497 3 OF REAL  
 498 PROCEDURE Add ( IN a, b: Euclid; OUT c: Euclid );  
 499 PROCEDURE ScalProd ( IN a, b: Euclid ): REAL;  
 500 PROCEDURE VecProd ( IN a, b: Euclid; OUT c: Euclid );  
 501 PROCEDURE Scale ( VAR a: Euclid; r: REAL );  
 502 PROCEDURE Norm ( IN a: Euclid ): REAL;  
 503

504 **Проверки границ массивов**

505 Всегда гарантируется TRAP, если  $n < 0$  или  $n \geq \text{LEN}(v)$ .  
 506 Отключить эти проверки в Оберонах **нельзя** — в отличие от б-ва других  
 507 компиляторов.  
 508 Влияние на эффективность  $\leq 1\%$  — многие проверяли (я тоже).

509 **Buffer overflow (overrun)**

510 один из самых часто встречающихся типов "дыр" в программах на  
 511 фортране/C/C++  
 512 родств: stack overflow, heap overflow, "off-by-one"  
 513 напр., массив ARRAY OF CHAR, принимающий "адрес" из браузера ...  
 514  
 515 **Цитаты из новостей:**  
 516

517 2001-08-28: **Win XP slays buffer overflow bugs** By John Leyden  
 518 *Microsoft has eradicated buffer overflows with Windows XP, following a source*  
 519 *code security audit, group veep Jim Allchin claimed during a keynote at the*  
 520 *Intel Developers Forum in San Jose. ..*

521 2002-01-24: .. *remember last November when Microsoft's Jim Allchin, group*  
 522 *vice president, said in a published interview that all buffer overflows were*  
 523 *eliminated in Windows XP? .. the Universal Plug and Play vulnerability that was*  
 524 *found last month .. was a buffer overflow.*

525 2002-10-25: .. *the US government warned of a critical flaw that could allow*  
 526 *hackers to circumvent the secure networking system. The problem lies with*  
 527 *software in MIT Kerberos 5 called kadmind4*  
 528 *(Kerberos v4 compatibility administration daemon), which allows compatibility*  
 529 *with older administrative clients. A buffer stack overflow allows an attacker*  
 530 *to use a specially formed request to gain access to the KDC with the privileges*  
 531 *of a user running kadmind4. ..*

532 2003-03-26: .. McGraw [эксперт по безопасности] *pierced the belief in a*  
 533 *single safe platform — though some are better than others.*  
 534 *"C is assembly language on steroids. Read Kernighan and Ritchie on input,*  
 535 *Chapter 7, page 164.*  
 536 *Don't do it that way -- the sure way to get a buffer overflow — and that's*  
 537 *the bible!*  
 538 *When you're choosing a language, use a typesafe language that controls*  
 539 *memory in a way that was invented in 1959..."*

540 2003-08-31: .. *OpenBSD's Todd Miller reports that an improper bounds check*  
 541 *[=buffer overflow] in the semget(2) system call can allow a local user to*  
 542 *cause a kernel panic. ..*

543 2004-05-26: .. *"It is a real embarrassment to the industry that people still*  
 544 *produce code with buffer overflows." .. The MSblast and Sasser worms both*  
 545 *used buffer overflows in Microsoft's Windows OS ..*

546 2004-09-03: .. *Cryptography can't help you if there's an exploitable buffer*  
 547 *overflow, and buffer overflows abound in code written in C. ..*

548 2004-09-04: .. *Oracle pushed out a host of long-awaited patches .. The flaws*  
 549 *range from common memory errors known as buffer overflows to ..*

550 2004-09-26: .. *A toolkit designed to exploit a recently-disclosed Microsoft*  
 551 *JPEG vulnerability*  
 552 *(http://www.theregister.co.uk/2004/09/15/windows\_jpeg\_bug) has been*  
 553 *released onto the net. .. Malformed JPEG files are capable of triggering a*  
 554 *buffer overflow in a common Windows component (the GDI+ image viewing*  
 555 *library) ..*

556 2004-10-09: .. *An unpatched security vulnerability in popular older versions of*  
 557 *Microsoft Word poses a severe threat to users .. The flaw stems from an*  
 558 *input validation error in Word. .. A buffer overflow vulnerability, the most*  
 559 *common class of security vulnerability, is to blame. ..*

560 2005-08-20: .. *Adobe has issued updates to guard against a buffer overflow*  
 561 *vulnerability in various versions of its popular Acrobat and Reader software*  
 562 *packages. .. might be used to inject hostile code into vulnerable systems by*

563 *tricking potential victims into opening a maliciously constructed PDF file. ..*  
 564 **critical.**  
 565 *Adobe Reader users on Windows or Mac OS are advised to [upgrade to version](#)*  
 566 *7.0.3 or 6.0.4.*  
 567 2006-11-05 Ночью на 5м часу перекодирования DVD-9 гробанулся Nero v.7:  
 568 \*\*\*BUFFER OVERFLOW\*\*\*  
 569

570 **Помнить о:**

571 **(0) баранах и барашках;**  
 572 **(1) ам. системе образования (MIT, Ivy League ..**  
 573 **BLAH-BLAH-BLAH);**  
 574 **(2) лучших школах прог-я — в Цюрихе,**  
 575 **Новосибирске, Питере;**  
 576 **(3) том, что учить студентов основам прог-я,**  
 577 **используя C/C++, есть преступление против**  
 578 **человечества в совершенно конкретном смысле.**

579 Вспомним записи без явного типа. Аналогично:

580 VAR a, b: ARRAY 26 OF REAL;

581 Можно написать

582 a := b;

583 компилятор "видит", что a и b имеют одинаковый тип.

584 Достаточно написать VAR a: ARRAY 26 OF REAL; b: ARRAY 26 OF REAL;

585 чтобы компилятор "сбился со следа".

586 **NB** Как и с записями: эта "**тупость**" компилятора заложена в дизайн:

587 ARRAY 4 OF REAL — релятивистский 4-вектор или кватернион?

588 Вспомним записи:

589 TYPE Vector = ARRAY 26 OF REAL;

590 и объявляйте сколько угодно переменных и параметров типа Vector:

591 почти всё, что было сказано про передачу параметров относительно записей,  
 592 работает и здесь:

593 PROCEDURE ScalProd( (\*IN\*) a, b: Vector ): REAL;

594 PROCEDURE Rotate( VAR a: Vector; n: INTEGER );

595 **Кунштюк:** вспомогательный буфер в процедуре.

596 объявление:

597 PROCEDURE Do ( IN a: Euclid; **buf**: Euclid; ... );

598 вызов:

599 Do( a, **a**, ... );

600 buf — внутр. переменная-массив ...

601 **Нельзя:** ассоциировать процедуру:

602 невозможно: PROCEDURE ( v: Vector ) Norm(): REAL;

603 **Но всегда возможно:**

604 TYPE

605 Vector = RECORD

606 a: ARRAY 26 OF REAL

607 END;

608 Непрерывный блок памяти,

609 обращение к эл-там v.a[ i ], в машинных кодах — никакой разницы.

610

611 **"Игра в кубики"**

612

613 Запись у которой одно из полей — массив.

614 TYPE

615 Minkowski = RECORD

616 E: REAL;

617 m: Euclid

618 END;

619 **Упр** Используя тип Euclid и соотв. процедуры, написать модуль для работы с

620 Minkowski:

621 PROCEDURE Prod( a, b ): REAL; (\* a.E \* b.E - a.m \* b.m \*)

622 и т.д.

623

624 Теперь **массив записей:**

625 TYPE

626 Complex = RECORD x, y: REAL END;

627 Vector = ARRAY 26 OF Complex;

628 VAR v: Vector;

629 тогда:

630 v[ i ] — переменная типа Complex (запись с двумя полями),

631 v[ i ].x — переменная типа REAL.

632 [ i ] как ранее .x — **селектор.**

633 Применение к (возможно, составному) имени селектора — получение другой,

634 более "мелкой" переменной, составной части исходной.

635 *Может иметь место нагромождение селекторов — но всегда линейная цепочка.*

636 Теперь **массив массивов:**

637 TYPE Matrix = ARRAY 26 OF Vector;

638 Каждый Vector — непр. блок памяти известного размера — компилятору больше

639 ничего не нужно, чтобы отвести память под

640 VAR m: Matrix;

641 Тогда:

642 m[ i ] — Vector, можно написать m[ i ] := v, где v: Vector;

643 m[ i ][ j ] — Complex;

644 m[ i ][ j ].x — REAL.

645 *Выражения — составные имена [**designators**] — в качестве имен.*

646 **Упр** Написать процедуру, переставляющую в 0-место заданный вектор (Vector)

647 матрицы (Matrix):

648 PROCEDURE Change ( VAR m: Matrix; i: INTEGER );

649 Синтаксич. сокращения:

650 m[ i ][ j ] → m[ i, j ]

651 Компилятора раскрывает m[ i, j ] обратно в m[ i ][ j ] в самом начале своего

652 анализа, еще не зная, к чему относятся индексы.

653 А также:

654 TYPE Matrix = ARRAY 26, 26 OF Complex;

655 вместо ARRAY 26 OF **ARRAY 26 OF Complex**;

656 снова *чисто синтаксическое сокращение.*

657 **LEN( m, 0 )** — длина по первому индексу.  
 658 **LEN( m, 1 )** — длина по второму индексу.  
 659 и т.д.  
 660 **Упр** Выяснить с помощью Сообщения..., можно ли ставить выражение (а не  
 661 константное выражение) в качестве второго аргумента.

### 662 По поводу фортрана:

663 Очевидно, как расположены эл-ты многомерных массивов в памяти:  
 664  $m[ 0, 0 ], m[ 0, 1 ], m[ 0, 2 ], \dots m[ 0,25 ], m[ 1, 0 ], \dots$   
 665 т.е. "по строкам".  
 666 В фортране они хранятся "по столбцам" (Бэкус...) — важно знать для  
 667 сопряжения ....

### 668 Открытые массивы в качестве параметров

669 Например, часто встречающаяся функция: длина вектора.  
 670 Для любых ARRAY (\*что угодно\*) OF REAL.  
 671 Решение:

```
672 PROCEDURE Norm ( (* IN *) v: ARRAY OF REAL ): REAL;
673   VAR i: INTEGER; sum, t: REAL;
674 BEGIN
675   sum := 0; i := 0;
676   WHILE i < LEN( v ) DO
677     t := v[ i ];
678     sum := sum + t * t;
679     i := i + 1
680 END;
681 RETURN Math.Sqrt( sum )
682 END Norm;
```

683 Если

```
684 TYPE Vector = ARRAY 11 OF REAL, и
685 VAR v: Vector,
```

686 то **можно** вызвать  $r := \text{Norm}( v )$ .

687 **Упр** Реализовать процедуру, вычисляющую определитель квадратной матрицы  
 688 (простейшим способом по Гауссу, без выделения "главного" элемента).

689 **Упр** Реализовать процедуру, решающую систему линейных уравнений (тем же  
 690 простейшим способом). Начать с фиксации сигнатуры (коэффициенты левых  
 691 частей, правые части, результат).

692  
 693 Простой пример работы с массивами: цепочки литер.  
 694

### 695 Алфавиты, цепочки литер, лексикографич. 696 упорядочение

697 **Алфавит** = конечный символов-литер.

698 Набор упорядочен, т.е. есть договоренность, всегда позволяющая для двух  
 699 разных литер решить, которая из них "предшествует" ("меньше", "младше")  
 700 другой. Пишут  $a < b$  и т.п.  
 701 Такой набор можно перенумеровать в соответствии с их порядком (не  
 702 обязательно подряд идущими числами).

703 Тогда у каждой литеры есть **числовой код**.

704 Тогда сравнение литер = сравнение их номеров (числовых кодов).

705 **NB** Нумерация и порядок могут быть разными для одного набора символом  
 706 (KOI8-R, Win-1251, Unicode).

707 **Примеры** Обычный алфавит а, б, в, ...; набор литер ASCII; набор литер  
 708 Unicode; литеры языка О/КП.

### 709 Литеры в КП

710 Типы значений CHAR (16 бит) и SHORTCHAR (8 бит).

711 Записываются либо непосредственно,

712 либо как **16-ричное значение с суффиксом X**.

```
713 20X          SPACE
```

```
714 21X      !
```

```
715 22X      "
```

```
716 ...
```

```
717 C0X      А
```

```
718 C1X      Б
```

```
719 ...
```

```
720 E0X      а
```

```
721 E1X      б
```

```
722 ...
```

```
723
```

### 724 Переменные типа CHAR

725 Пусть VAR с: CHAR; тогда:

```
726 с := "!";
```

```
727 с := 21X;
```

```
728 с := 09X; (* tab *)
```

```
729 с := 00X; (* символ конца цепочек литер *)
```

```
730 с := 0; (* ошибка! *)
```

731 **Упр** Написать програмку, в которой проверялась бы корректность таких  
 732 присваиваний.

### 733 Цепочки литер

734 **Цепочка литер** = произвольная конечная последовательность литер. Будем  
 735 заключать в кавычки (О/КП допускают как двойные, так и одинарные).

736 У цепочки есть **длина** (смысл -- естественный).

737 **Примеры** "абаа", "вьяааввв", "\*&^%\$#@, ^\*&#%@\$!".

738 **Упр** Определить длину этих цепочек.

739 **КП** Предопределенная функция **LEN**("абаа") → 4. Вычисляется (в данном  
 740 случае) еще при компиляции.

741 **Лексикографич. (словарное) упорядочение** цепочек литер:

742 основано на порядке литер в алфавите;

743 сначала сравниваются первые (крайние левые) литеры и т.п. (будет  
 744 упражнение на программирование).

745 **КП** Допустимы выражения "абак" < "кабак" (операция сравнения с двумя  
 746 операндами). Результат здесь имеет тип BOOLEAN (т.е. TRUE или FALSE).

747 **Цепочки литер в КП**

748 О записи константных цепочек уже сказали.

749 Переменные: неявный тип **String** (ShortString).

750 Используются массивы литер: ARRAY OF CHAR.

751 Литеры размещаются с начала массива.

752 **В конце ставится 0X.** Т.е. длина массива  $\geq$  длина цепочки + 1.

753 **NB** Возможность хранить значение типа String в массивах разной длины подобна возможности хранить одно и то же числовое значение (скажем, 1) в переменных INTEGER (32 бита), LONGINT (64 бита), SHORTINT (16 бит), BYTE (8 бит). И для целых, и для цепочек значение может не поместиться в соотв. "контейнер".

758 Пусть VAR s: ARRAY 30 OF CHAR; тогда можно записать:

759 s := "Николай Вальтерович";

760 После этого

761 s[0] = "Н" и s[1] = "и" и ... s[18] = "ч" и s[19] = 0X

762 причем значения эл-тов s[20] и выше — неопределены (т.е. там будет "мусор").

763 Но сл. присваивание вызовет ошибку:

764 s := "012345678901234567890123456789";

765 **Q** Почему?766 **Упр** Проверить все вышесказанное с помощью реальных программ.

767 Пусть VAR s: ARRAY 30 OF CHAR; t: ARRAY 100 OF CHAR; и

768 s := "Николай Вальтерович";

769 Тогда можно написать

770 t := s\$;

771 произойдет поэлементное копирование цепочки, включая 0X.

772 **NB** Если в t не хватит места —> облом.

773 Можно и обратно: s := t\$; важно, чтобы String-значение, хранящееся в t,

774 помещалось в s вместе с 0X.

775 Еще раз: **s\$** = цепочка литер (без 0X), хранящаяся в массиве s.776 **Конкатенация цепочек**

777 "Николай" + " " + "Вальтерович" —&gt; "Николай Вальтерович"

778 Выполняется (как всегда в О/КП) слева направо.

779 **NB** Конкатенация (как и операции сравнения литерных цепочек) подразумевает, если нужно, переход от массивов к соотв. значениям типа String (т.е. литеры после 0X игнорируются).

782 Пусть VAR a, b, c, d: ARRAY 100 OF CHAR; и

783 a := "Николай";

784 b := " ";

785 c := "Вальтерович";

786 d := a + b + c;

787 Тогда компилятор поймет, что мы хотим конкатенировать три цепочки и сделает соотв. код.

789 В итоге пройдет проверка

790 ASSERT( d\$ = "Николай Вальтерович" );

791

792 Все остальные операции с цепочками (поиск, замена, преобразование чисел в цепочки и обратно и т.п.) — в модуле Strings (ниже).

794

795 **Массивы литер. Неявный тип String**

796 Минимальные средства обработки строк включены в язык, чтобы с удобством удовлетворять нужды "скриптования" (обычно используют отдельный скриптовый язык типа perl, python, и т.п.).

799 **О/КП — сам себе скриптовый язык!!!**

800

801 Есть целые числа — есть цепочки литер String (состоят из CHAR = 2 байта, подразумевается Unicode).

802 Будем говорить про них. (Есть еще ShortString — Extended ASCII, SHORTCHAR, 1 байт; все аналогично).

805 Есть целые разной величины — есть цепочки разной длины.

806 — константы изображаются как "ждло lkjh \*&amp;^%\$ " или 'длоп0986 \_\_\_';

807 внешние кавычки **не** часть значения;

808 можно в содержимое вставлять "другие" кавычки, напр.: """""" (несколько

809 одинарных внутри пары двойных) или """""" (несколько двойных внутри пары одинарных);

811 — для таких значений разрешаются лексикографические сравнения:

812 &lt;, &gt;, &lt;=, &gt;=, =, # (как в словаре, в соотв. с числовыми кодами соотв. литер;

813 подробнее отдельно);

814 — функция LEN: LEN("123") = 3.

815 Есть переменные для целых чисел разного размера (INTEGER, SHORTINT, BYTE, LONGINT) — есть способ хранить цепочки разной длины: в массивах литер:

817 — для хранения используются переменные типов ARRAY OF CHAR (и ARRAY OF SHORTCHAR) разной длины (как INTEGER, SHORTINT, BYTE, LONGINT все

818 используются для хранения целочисленных значений);

820 — если VAR a: ARRAY 100 OF CHAR; (если POINTER, то ... )

821 то после:

822 a := "курс";

823 содержимое a, поэлементно: "к", "у", "р", "с", **0X**, мусор;

824 при этом

825 LEN( a ) = 100

826 LEN( a\$ ) = 4 — **не 5!**

827 — если еще VAR b: ARRAY 90 OF CHAR;

828 и выполнено

829 b := "2005";

830 то

831 a := b — compiler error;

832 a := b\$ — эквивалентно a := "2005";

833 (**NB** \$ — пример "разыменования", т.е. "превращения переменной в значение" - частный случай превращения имени в называемый объект).

835 Аналогично происходит передача значения value-параметру (можно написать

836 StdLog.String( a\$ ))

837 — есть операция **конкатенации**, обозначаемая +:

838 a := a + b — как если бы a := "курс2005"; (спецлитера 0X всегда только в конце).

840 — TRAP если результат в правой части слишком длинен, чтобы поместиться (вместе с 0X) в массиве, который указан в левой.

841

842 Общая работа с цепочками литер — большая тема, много вариантов.  
 843 Для б-ва простых случаев в Блэкбоксе есть модуль Strings:  
 844  
 845 DEFINITION Strings; (\* переупорядочено \*)  
 846  
 847 CONST  
 848 charCode = -1;  
 849 decimal = 10;  
 850 digitspace = 8FX;  
 851 hexadecimal = -2;  
 852 hideBase = FALSE;  
 853 roman = -3;  
 854 showBase = TRUE;  
 855  
 856 PROCEDURE Valid (IN s: ARRAY OF CHAR): BOOLEAN;  
 857  
 858 PROCEDURE Find (IN s: ARRAY OF CHAR; IN pat: ARRAY OF CHAR; start:  
 859 INTEGER; OUT pos: INTEGER);  
 860 PROCEDURE Extract (s: ARRAY OF CHAR; pos, len: INTEGER; OUT res: ARRAY  
 861 OF CHAR);  
 862 PROCEDURE Replace (VAR s: ARRAY OF CHAR; pos, len: INTEGER; IN rep:  
 863 ARRAY OF CHAR);  
 864  
 865 PROCEDURE Lower (ch: CHAR): CHAR;  
 866 PROCEDURE ToLower (in: ARRAY OF CHAR; OUT out: ARRAY OF CHAR);  
 867 PROCEDURE Upper (ch: CHAR): CHAR;  
 868 PROCEDURE ToUpper (in: ARRAY OF CHAR; OUT out: ARRAY OF CHAR);  
 869  
 870 PROCEDURE RealToString (x: REAL; OUT s: ARRAY OF CHAR);  
 871 PROCEDURE RealToStringForm (x: REAL; precision, minW, expW: INTEGER;  
 872 fillCh: CHAR; OUT s: ARRAY OF CHAR);  
 873 PROCEDURE IntToString (x: LONGINT; OUT s: ARRAY OF CHAR);  
 874 PROCEDURE IntToStringForm (x: LONGINT; form, minWidth: INTEGER; fillCh:  
 875 CHAR; showBase: BOOLEAN; OUT s: ARRAY OF CHAR);  
 876  
 877 PROCEDURE StringToInt (IN s: ARRAY OF CHAR; OUT x, res: INTEGER);  
 878 PROCEDURE StringToInt (IN s: ARRAY OF CHAR; OUT x: LONGINT; OUT res:  
 879 INTEGER);  
 880 PROCEDURE StringToReal (IN s: ARRAY OF CHAR; OUT x: REAL; OUT res:  
 881 INTEGER);  
 882  
 883 END Strings.  
 884  
 885 Результаты процедур — корректно сформированные цепочки литер Strings.  
 886 *The library is optimized for convenience, not for efficiency.*  
 887 *This tradeoff is apparent in that some operations, such as Extract, use value parameters instead*  
 888 *of IN parameters. This allows to pass the same variable both for input and output purposes,*  
 889 *which is often convenient (a variable should never be passed to several IN/OUT/VAR*  
 890 *parameters simultaneously, since this may cause interference between them).*  
 891 *It is not a goal to provide operations for all possible circumstances, since string processing in*  
 892 *different applications simply varies too much to make this practical. Often it is useful to write a few*

893 *string operations fully tailored to a particular application, which is usually easy to do. Moreover,*  
 894 *such custom string operations can be optimized for speed, which is not possible for too general*  
 895 *routines.*  
 896 *Note that the language Component Pascal provides efficient built-in support for string*  
 897 *assignment (implicitly or explicitly with the "\$" operator), for string concatenation (with the "+"*  
 898 *operator), and for counting the number of characters in a string (LEN(string\$)).*  
 899  
 900 **NB** результат справа — противуречит синтаксису присваивания.  
 901 **Упр** Реализовать PROCEDURE Concat ( IN a, b: ARRAY OF CHAR; OUT res: ARRAY  
 902 OF CHAR ).  
 903 **Упр** Реализовать Find.  
 904 **Упр** Научиться реализовывать любую из этих процедур по описанию в  
 905 документации.  
 906  
 907 **Упр** Написать примеры процедур, использующих Find, Replace, Extract для  
 908 разных значений параметров и печатающих значения операндов и результатов  
 909 до и после соотв. операций.  
 910 **Упр** То же для IntToString, StringToInt; RealToString, StringToReal.  
 911 **Упр** То же для IntToStringForm, StringToRealForm.  
 912  
 912 **NB** Печать в Log: StdLog.String(.....).