

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

Физфак МГУ. 2006/7 уч. год.

Лекция 22

Сортировки

Упорядоченность -- фундаментальное свойство множеств, особенно в алгоритмическом плане.

В комбинаторике -- понятие (частично) упорядоченных множеств одно из самых основных.

В бизнес-приложениях: еще в 60х гг. IBM наблюдали, что их мейнфреймы в банках заняты сортировками львиную часть времени.

В компьютерной алгебре: большая часть времени -- на (алгебраические) сортировки (с сопутствующими вещами -- сложение коэффициентов и т.п.).

Систематическое использование упорядоченности данных --> последовательная обработка --> способствует масштабируемости.

Пример из жизни: поиск нетривиальных решений больших однородных систем линейных уравнений (10^5 уравнений, примерно столько же неизвестных) с коэффициентами, являющимися рац. функциями над кольцом целых чисел (произвольная точность).

Алгоритм тривиальный, но есть несколько множеств, и в каждом возможна разная упорядоченность:

термов в уравнениях; еще необработанных уравнений; термов в правых частях уже найденных подстановок; совокупности этих подстановок.

Правильно согласованный (заранее неочевидный) выбор упорядоченностей для всех этих множеств радикально ускоряет вычисления + позволяет масштабировать.

Некий реальный учебник: "Методы сортировки можно разбить на три основные группы: сортировка выбором, сортировка обменом, сортировка вставками."

ЧУШЬ.

Главное деление — по асимптотическому поведению времени в зависимости от числа эл-тов.

Квадратичные ($N \times N$) и **"быстрые"** ($N \ln N$).

(Возможны и промежут. вар-ты.)

Нужно знать одну простую квадратичную (пузырек, вставки) для простых задач, и самую быструю -- для массивов quicksort (C.A.Hoare) и соотв. аналог для списков (но там слияния).

Есть еще сортировка хешированием. Быстрая, но специфич.; огранич., читайте книжки.

Уметь — для массивов и списков.

Сортировки массивов

Вставки

квадратичная

схема в общем пригодна и для списков

```

51
52
53 PROCEDURE Сорт* ( VAR a: ARRAY OF INTEGER );
54   VAR i: INTEGER;
55   BEGIN
56     i := 1;
57     (* инвариант: сегмент ниже i упорядочен *)
58     WHILE i < LEN( a ) DO
59       Вставить( a, i );
60       INC( i )
61     END;
62     (* инвариант по-прежнему выполнен, но i = LEN( a ).
63       Значит, все упорядочены.
64       *)
65   END Сорт;
66

```

NB Внимательно следить за "краевыми эффектами": в данном случае предполагается, что упорядочен сегмент $0 \dots i-1$, но элемент в позиции i может эту упорядоченность разрушать. Цель шага цикла -- обеспечить упорядоченность сегмента $0 \dots i$, включая позицию i , чтобы можно было увеличить i на 1, и снова получить выполняющийся инвариант.

Т.о. задача сводится к более простой.

NB Такого рода рассуждения возможны обычно *после* завершения разработки программы.

Напоминание: идеальная разработка пошаговым уточнением -- это цель реальной разработки в том смысле, что окончательная программа должна допускать естественное объяснение, как будто она именно так идеально и разрабатывалась.

Вставка i -го элемента в уже упорядоченный сегмент $0 \dots i-1$ -- более простая задача:

(**NB** Нарисовать картинки для всех интересных мест.)

```

86
87 PROCEDURE Вставить ( VAR a: ARRAY OF INTEGER; i: INTEGER );
88   VAR j, k, t: INTEGER;
89   BEGIN
90     (* на входе: сегмент ниже i -- т.е.  $0 \dots i-1$  -- уже упорядочен;
91       нужно: добиться упорядоченности всего сегмента  $0 \dots i$ , включая i-й
92       элемент. *)
93     (* Значение в i-й позиции нарушает упорядоченность.
94       Будем искать справа налево первую позицию j такую, что  $a[j-1] \leq$ 
95        $a[i]$ .
96       Соотв. схема поиска: *)
97     j := i;
98     WHILE ( j > 0 ) & ~( a[j-1] <= a[i] ) DO
99       DEC( j )
100     END;
101     (* успех поиска -- j > 0; неуспех -- j = 0. Оба случая в дальнейшем
102       обрабатываются одинаково *)
103

```

```

104      (* сдвинем все значения в позициях j .. i-1 на одну позицию вправо,
105 чтобы освободить нужное место для значения в позиции i. Оно предварительно
106 сохраним (Q зачем?): *)
107      t := a[ i ];
108      FOR k := i TO j+1 BY -1 DO
109          a[ k ] := a[ k-1 ]
110      END;
111
112      (* втыкаем куда надо значение, бывшее в начале в a[ i ] и сохраненное в
113 t: *)
114      a[ j ] := t;
115      (* не забываем проверить, что все правильно работает и в случае j = i.
116      NB важность правильно определять семантику конструкций ЯП для
117 вырожденных случаев. Например, иногда требуют, чтобы тело FOR выполнялось
118 хотя бы раз -- все сразу осложнится проверками ... *)
119      END Вставить;
120
121 Просто и понятно как репа.
122 Повозимся для упражнения -- пристально разглядываем.
123 Будем упрощать маленькими, заведомо правильными шажками.
124
125 1) a[ i ] многократно фигурирует и в первом цикле. Перенесем присваивание
126 t := a[ i ] перед j := i, тогда все случаи значения a[ i ] можно заменить на t.
127
128      t := a[ i ];
129      j := i;
130      WHILE ( j > 0 ) & ~( a[ j-1 ] <= t ) DO
131          DEC( j )
132      END;
133      FOR k := i TO j+1 BY -1 DO
134          a[ k ] := a[ k-1 ]
135      END;
136      a[ j ] := t;
137
138 2) замечаем, что k во втором цикле пробегает в точности те же значения, что и j
139 в первом цикле.
140 Для пушего удобства сравнения циклов перепишем второй через WHILE:
141
142      k := i;
143      WHILE k > j DO
144          a[ k ] := a[ k-1 ];
145          DEC( k )
146      END;
147
148 И еще раз как схему поиска (ведь j >= 0, поэтому можно вставить более слабое
149 условие k > 0):
150
151      k := i;
152      WHILE ( k > 0 ) & ( k > j ) DO
153          a[ k ] := a[ k-1 ];
154          DEC( k )
155      END;
156

```

```

157 Осталось объединить в один цикл, т.к. присваивание во втором цикле не портит
158 охрану на следующем шаге:
159
160      t := a[ i ];
161      j := i;
162      WHILE ( j > 0 ) & ~( a[ j-1 ] <= t ) DO
163          a[ j ] := a[ j-1 ];
164          DEC( j )
165      END;
166      a[ j ] := t;
167
168 Подобный трюк -- вещь опасная.
169 KEEP IT SIMPLE, т.е. выбирайте первый вариант.
170
171
172 Quicksort ("быстрая")
173 простейший вариант.
174
175 MODULE Kurs2006QuickSort;
176     IMPORT Log := StdLog, Math, In := FVTsysIn (* Epse21SysIn *);
177
178     PROCEDURE Разделить ( VAR a: ARRAY OF INTEGER; i, j: INTEGER; OUT k:
179 INTEGER );
180         VAR i0, j0, pivot, t: INTEGER;
181         BEGIN
182             ASSERT( j - i > 2, 20 );
183             i0 := i; j0 := j;
184             pivot := a[ i + ( j - i ) DIV 2 ];
185             (* i и j движутся навстречу с краев,
186             оставляя за собой эл-ты, соответственно, <= и => pivot
187             *)
188             (* три сегмента: i0 -- i; i -- j; j -- j0 *)
189             WHILE i < j DO
190                 IF a[ i ] < pivot THEN
191                     INC( i )
192                 ELSIF a[ j-1 ] > pivot THEN
193                     DEC( j )
194                 ELSE
195                     t := a[ i ]; a[ i ] := a[ j-1 ]; a[ j-1 ] := t;
196                     INC( i ); DEC( j )
197                 END
198             END;
199             ASSERT( ( i > i0 ) & ( i < j0 ), 101 );
200             k := i
201         END Разделить;
202
203     PROCEDURE RecSort ( VAR a: ARRAY OF INTEGER; i, j: INTEGER );
204         (* Процедура сортирует сегмент массива i, j.
205         Сегмент i, j = с i по j-1 включительно -- но не j-й!
206         Так определять массив обычно удобней всего.
207         *)
208         VAR k, t: INTEGER;
209         BEGIN

```

```

210  ASSERT( i < j, 20 );
211  IF j - i = 1 THEN
212    (* нечего делать *)
213  ELSIF j - i = 2 THEN
214    IF a[ i ] > a[ i+1 ] THEN
215      t := a[ i ]; a[ i ] := a[ i+1 ]; a[ i+1 ] := t
216    END
217  ELSE
218    Разделить( a, i, j, k );
219    ASSERT( ( i < k ) & ( k < j ), 100 );
220    (* any in segment i, k <= any in segment k, j *)
221    RecSort( a, i, k );
222    RecSort( a, k, j )
223  END
224 END RecSort;
225
226 PROCEDURE QuickSort* ( VAR a: ARRAY OF INTEGER );
227 BEGIN RecSort( a, 0, LEN( a ) )
228 END QuickSort;
229
230 PROCEDURE Do*;
231 VAR x, n: INTEGER; a: POINTER TO ARRAY OF INTEGER;
232 BEGIN
233   In.Open; In.Int( x ); n := 0;
234   WHILE In.Done DO INC( n ); In.Int( x ) END;
235   NEW( a, n );
236   In.Open; In.Int( x ); n := 0;
237   WHILE In.Done DO a[ n ] := x; INC( n ); In.Int( x ) END;
238   QuickSort( a );
239   FOR n := 0 TO LEN(a) - 1 DO Log.Int( a[ n ] ) END; Log.Ln
240 END Do;
241
242 END Kurs2006QuickSort.
243 !Kurs2006QuickSort.Do 9 4 3 -23 2435 5 11 -4 0 !
244 compiling "Kurs2005QuickSort"
245   new symbol file 880 0
246   -23 -4 0 3 4 5 9 11 2435
247 !Kurs2006QuickSort.Do 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 !
248 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
249
250 Возможные оптимизации
251 Более равномерное деление: выбор pivot из большего числа эл-тов
252 (упорядочить два крайних и средний, взять средний).
253 Более простая (пусть квадр.) сортировка для малых сегментов.
254
255 Упр Реализовать обе оптимизации.
256
257 Отложим сортировку списков до след. раза, чтобы не
258 интерферировало с массивами.
259
260 Для N <= 20 -- вставки, для N > 20 -- quicksort.
261

```

Профилирование

в связи с сортировками, эффективность которых обычно есть важный фактор

```

262
263 в связи с сортировками, эффективность которых обычно есть важный фактор
264
265 MODULE Kurs2006Profiler;
266   IMPORT Math, Kernel, P := DevProfiler;
267
268   PROCEDURE ЛюбаяПрограмма;
269     CONST n = 100000; VAR s, r: REAL; i: INTEGER;
270     BEGIN s := 0; r := 2 * Math.Pi();
271     FOR i := 0 TO n DO s := s + Math.Sin( r * i / n ) END
272   END ЛюбаяПрограмма;
273
274   PROCEDURE MakeProfiles*;
275   BEGIN
276     P.Reset; P.SetModuleList("Kurs2005Profiler,Kernel,Math"); P.Start;
277     ЛюбаяПрограмма;
278     P.Stop; P.ShowProfile
279   END MakeProfiles;
280
281 END Kurs2006Profiler. !Kurs2006Profiler.MakeProfiles
282
283 окошко Profile:
284 -----
285 Module                                     % per module
286   Procedure                               % per procedure
287
288 Math                                     87
289   Sin                                    50
290   Reduce                                 37
291
292 Kurs2006Profiler                         12
293   ЛюбаяПрограмма                         12
294
295 samples:                                8          100%
296   in profiled modules                    8          100%
297   other                                   0           0%
298 -----
299
300 еще удобная фица в меню Dev, Timed Execute (сначала выделить мышкой
301 команду — то, что обычно после командера) — в Log'e будет время в
302 миллисекундах:
303   62 msec
304
305 Упр Исследовать скорость сортировок как функцию кол-ва элементов (до самых
306 больших N ...). Использовать случайные последовательности чисел.
307

```

308 Понятие о связи с "внешним 309 миром"

310 Подробно для C++ и т.п. см. документацию в подсистеме Eps21.

311 Вызываем фортран

312 Пусть есть процедуры на фортране, которые хотим вызывать из Блэкбокса.
313 Это может выглядеть так:

```
317 SUBROUTINE proc1 ( a, res )
318   INTEGER a
319   DOUBLE PRECISION res
320   res = a + res
321 END
```

```
323 SUBROUTINE proc2 ( a, res )
324   INTEGER a
325   DOUBLE PRECISION res
326   res = a
327 END
```

328 Предположим, что работаем в Fortran Power Station 4.0.

329 Тогда основной источник — раздел *Mixed-Language Programming* в
330 документации FPS.

331 Оттуда и взята информация.

332
333 В FPS создадим новый "workspace" ("проект") класса Dynamic Link Library,
334 назовем его BBtest, и создадим в нем один файл prog.f, в который поместим
335 вышеук. код.

336 Добавим в начало файла два "meta statements", по одному для каждой
337 процедуры:

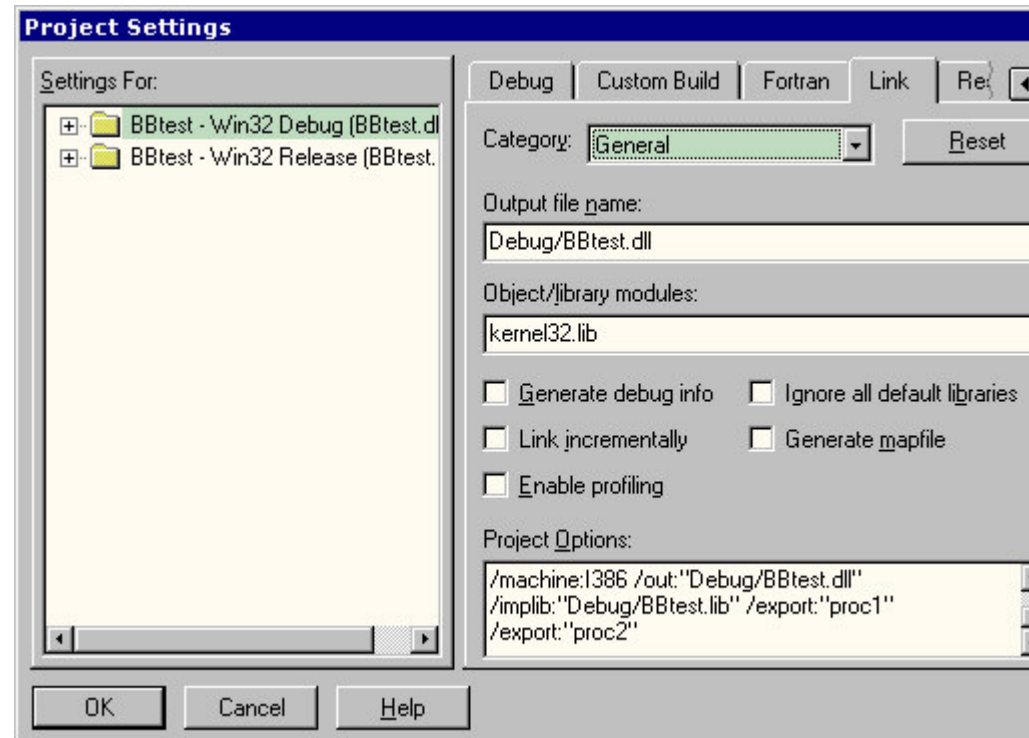
```
340 !MS$ATTRIBUTES STDCALL, REFERENCE, ALIAS:'proc1' :: proc1
341 !MS$ATTRIBUTES STDCALL, REFERENCE, ALIAS:'proc2' :: proc2
```

342
343 (у меня текст начинался с первой позиции).

344 REFERENCE означает, что параметры будут передаваться по ссылке (VAR).
345 STDCALL означает, что передача параметров на уровне машинного кода будет
346 происходить по некоему стандартному соглашению. (Есть и другое — ccall.)

347 Обеспечим, чтобы эти две процедуры экспортировались в нашей dll.
348 Для этого в меню Build → Settings ... пойдём в закладку Link:

350
351



352

353

354 и в конце Project Options вставим /export:"proc1" /export:"proc2" (как на
355 картинке). OK.

356 (Если окошко серое, значит в левой панели отмечены обе строки; обеспечьте,
357 чтобы была отмечена только одна, например, как показано на картинке.)

358 После этого выполним Build → Build ... (Shift+F8).

359 Создастся BBtest.dll.

360 Скопируем ее в домашнюю папку Блэкбокса (можно вторичную).

361 dll-ка готова.

362

363 Теперь в Блэкбоксе напишем **интерфейсный модуль**:

364

```
365 MODULE Kurs2006Dll ["BBtest2"];
```

366

```
367   PROCEDURE Add* ["proc1"] ( VAR a: INTEGER; VAR res: REAL );
```

368

```
369   PROCEDURE Assign* ["proc2"] ( VAR a: INTEGER; VAR res: REAL );
```

370

```
371 END Kurs2006Dll.
```

372

373 Скомпилируем его.

374 Теперь можно писать программы, использующие приготовленную dll-ку -- в
375 программах достаточно обращаться к интерфейсному модулю. Например:

376

```

377 MODULE Kurs2006Dllcall;
378
379 IMPORT Log:=StdLog, Kurs2006Dll;
380
381 PROCEDURE Do* ( i: INTEGER );
382     VAR a, res: REAL;
383
384     PROCEDURE Print;
385     BEGIN Log.Int( i ); Log.Real( res ); Log.Ln
386           END Print;
387
388 BEGIN
389     res := 1000; Print;
390     Kurs2006Dll.Add( i, res ); Print;
391     Kurs2006Dll.Assign( i, res ); Print
392 END Do;
393
394 END Kurs2006Dllcall.
395
396 !"Kurs2006Dllcall.Do(77)"
397 вот что будет напечатано:
398 77 1.0E+3
399 77 1077.0
400 77 77.0
401
402 Больше примеров, в т.ч. с C++ — см. документ Epse21/Docu/DLL.odc
403
404 -----
405 по этому принципу устроена связь с
406 WinApi — модуль WinApi и т.д., а также с Office.
407
408 Есть еще интерфейс к OpenGL (подсистема Ogl) -- там и примеры модулей,
409 демонстрирующих разные средства. См. файл Ogl/ModuleList.odc — там и
410 команды, и имена модулей (выделить двойным кликом, Ctrl+O чтобы
411 открыть исходники).
412
413

```

О построении диалогов в Блэкбоксе

Диалог = блэкбковский документ, подобный тексту — "диалоговая форма" (в дальнейшем часто просто **форма**).
Состоит не из литер, организованных в текст,
а из "управляющих элементов" — визуальных объектов-картинок, обладающих некоторым поведением.

Хранятся в файлах *.odc, обычно в папках Rsrc в соотв. подсистемах.
Например, форма для поиска в текстах (та, что Ctrl+F) хранится в Text/Rsrc/Cmds.odc.

Форма может быть открыта двумя основными способами (причем одновременно!):
для редактирования (клики мышки интерпретируются соотв. образом: изменить положение или увеличить размер эл-тов и т.п.)
для работы (клики мышкой интерпретируются как нажатие командной кнопки и т.п.).

ВВ Для редактирования открываем как любой файл.

Обязательное упражнение

Откройте форму Text/Rsrc/Cmds.odc.
Если не можете найти файл, используйте командер:
!"StdCmds.OpenDoc('Text/Rsrc/Cmds.odc')".
Тут же **обязательно** выполните команду меню File, Save As ... и сохраните форму с любым **другим** именем (например, 0) — иначе испортите свой диалог Find/Replace.
Затем потаскайте мышкой кнопки. Выполните пункт меню Controls, Open As Aux Dialog — параллельно откроется второе окно с диалогом в активизированном, рабочем состоянии. Продолжайте таскать кнопки в первом окне: смотрите, как одновременно меняется второе!
Поменять размер кнопки: кликнуть, потянуть за черные квадратики.
Несколько эл-тов (для одновременного таскания) можно выделить как обычно — нажимая Shift при клике.
Сделайте Ctrl+C, Ctrl+V.
(Закройте второе окно, чтобы не мешалось; идею поняли.)
Кликните в Log, сделайте Ctrl+V там. Или мышкой че-нить перетащите из формы в Log.

Вернитесь в (первое) окно с формой. Вытяните любую кнопку частично за край формы, выполните Layout, Recalc Focus Size: форма увеличится, чтобы все эл-ты помещались полностью.
Выделив одну кнопку, нажмите Alt+Enter — откроется Inspector: поиграйте с ним (например, измените содержимое поля Label).
С открытым инспектором, покликайте по другим эл-там формы (в первом окне).

*Теперь вы можете перенастроить (переименовать, изменить горячие клавиши и т.п.) **любые** диалоги ББ на свой вкус.*

465 Логика кнопок и т.п.

466
467 Каждая **кнопка** в форме связана с командой (эксп. процедура без параметров)
468 в каком-то модуле (поле Link в инспекторе). Клик по кнопке в активном диалоге
469 вызывает команду. Это действие кнопки *не зависит* от того, где располагается
470 кнопка — в диалоге или в тексте.

471 В диалоге она "приклеена" к месту, в тексте она плавает в тексте — вот и вся
472 разница.

473
474 Каждое **числовое поле** связано с экспортир. глобальной числовой переменной
475 в каком-то модуле. Поле показывает состояние данной переменной. Но надо
476 понимать, что ББ всегда нужно "подтолкнуть", чтобы были видны новейшие
477 значения (процедуры Dialog.Update и др.). В обратную сторону (из диалога в
478 модуль) все делается автоматически и немедленно при каждом изменении.
479 "Подталкивать" надо, т.к. **модуль не знает** про те формы, которые его данные
480 показывают — и что это (показывание) вообще происходит. Упр. эл-т всегда
481 знает, что он показывает.

482
483 Если переменная **read-only** (экспортирована минусом вместо звездочки), то в
484 диалоге соотв. поле будет что-то показывать, но менять значение в диалоге
485 нельзя.

486 И т.д.

487
488 **Упражнение.** Набить и скомпилировать сл. модуль:

```
489
490 MODULE Kurs2004Dialog0;
491   VAR n*: INTEGER;
492 END Kurs2004Dialog0.
```

493
494 В любом текстовом документе выполнить команду Controls, Insert Edit Field —
495 получится серенькая штучка. Выделить ее клавишей (поставить курсор сразу
496 после, нажать Shift+стрелка влево) — должна быть видна черная рамка с
497 квадратиками (но без штриховки!). Нажать Alt+Enter для вызова инспектора
498 (диалог Inspector). В оном: в поле Link записать (без кавычек и скобок) полное
499 имя полное имя переменной n: Kurs2004Dialog0.n (можно копировать имя
500 модуля из исходника при открытом инспекторе; если инспектор "потерял след",
501 кликнуть на него, потом на числовое поле). Нажать в инспекторе OK.

502 Дважды кликнуть в середину кнопки. Появится штрихов. рамочка. Набрать там
503 число 1642 (год рождения сэра Исаака [Q кто таков?]).

504 Проверить, что модуль с переменной загрузился (Info, Loaded Modules).

505 Заметить его размер.

506 Посмотреть его глобальные переменные (где угодно выделить имя модуля,
507 меню: Info, Global Variables). Убедиться, что там в n хранится 1642.

508 Поменять число в числовом поле. Проверить, что в загруженном модуле оно
509 тоже меняется.

510 В модуль добавить процедуру:

511

```
512 MODULE Kurs2004Dialog0;
513   IMPORT StdLog;
514
515   VAR n*: INTEGER;
516
517   PROCEDURE Do*;
518   BEGIN StdLog.Int( n )
519   END Do;
```

518 END Kurs2004Dialog0.

519

520 Скомпилировать. Выгрузить (выделить имя, Dev, Unload Module List).

521 Посмотреть на числовое поле: там 0. Что изображается??

522 Проверить, что модуль в памяти — но у него другой, БОльший размер.

523 Видимый диалоговый элемент вызывает немедленную перезагрузку соотв.

524 модулей (надо же ему что-то показывать), даже после выгрузки "своего" модуля.

525 Но после перезагрузки он "привяжется" к новому модулю.


526 Попробуйте переименовать n на m, скомпилировать и выгрузить модуль.

527 Числовое поле посерело — "потеряло след".

528 Переименуйте переменную назад на n, скомпилируйте и выгрузите модуль.

529 Поле побелело и показывает 0 — правильное начальное значение глобальной
530 переменной.

531 В любом текстовом документе сделать Controls, Insert Command

532 Buttom.  Тут же выделить с клавиши (Shift+стрелка влево), Alt+Enter
533 (вызов инспектора), в поле Link набрать полное имя Do, и набрать просто Do в
534 поле Label. Кликнуть OK.

535 Кликнуть по Do. Проверить, что в Log печатается 0.


536 Кликнуть внутрь числового поля. Набрать там 1727. Кликнуть по Do. Что
537 печатается в Log?

538

539 Вместо того, чтобы вставлять числ. поля и кнопки в тексты, можно создать
540 новый диалог (Controls, New Form..., кликнуть по Empty) и вставлять поля и
541 кнопки туда. Таскать их там мышкой куда надо, с пом. инспектора привязывать
542 их к чему угодно. Тут же можно выполнить Controls, Open As Aux Dialog, чтобы
543 экспериментировать с диалогом.

544 После игр спасти форму в файл.

545 Вызов диалоговой формы как активного диалога (а не документа для
546 редактирования):

547  "StdCmds.OpenAuxDialog('путь+имя файла, где сохранили форму', 'какой-
548 нить аголовек')"

549

550 Сделать перерыв и обдумать виденное.

551

552 Есть и другие диалоговые эл-ты (см. Controls, Insert XXX), см. Form/Docu/User-
553 Man.odc.

554 Диалоговую форму можно генерить из программы "на лету", и тут же открывать
555 (команду внутри двойных кавычек можно вызвать из программы).

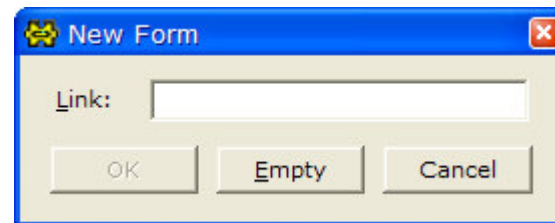
556

557 Автоматич. изготовление простого диалога для 558 простой задачи

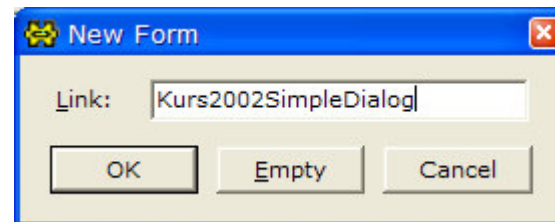
559 Делать диалоги почти так же легко, как вводить данные с помощью In.
560 Этим надо широко пользоваться в своих расчетах. *Диалоги ad hoc! Нигде кроме!*

```

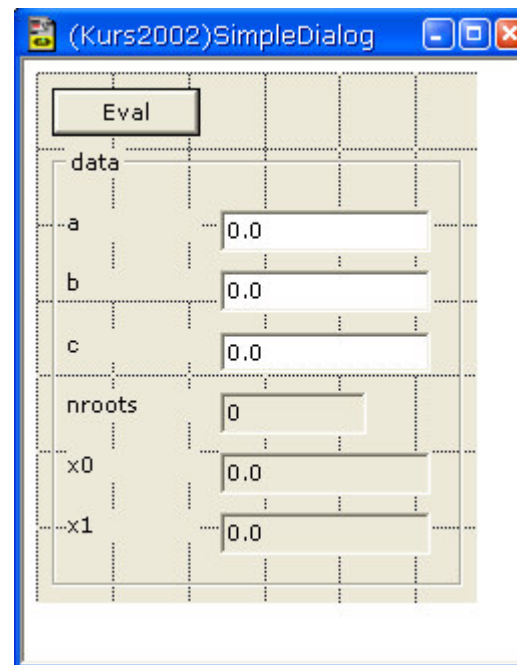
563 MODULE Kurs2002SimpleDialog;
564   IMPORT Math, Dialog;
565
566   VAR
567     data*: RECORD
568       a*, b*, c*: REAL; nroots-: INTEGER;
569       x0-, x1-: REAL
570     END;
571
572   PROCEDURE EvalRoots ( a, b, c: REAL; OUT n: INTEGER; OUT x0, x1: REAL );
573     (* a*x*x + b*x + c = 0 *)
574     VAR D: REAL;
575     BEGIN
576       D := b*b - 4*a*c;
577       IF D > 0 THEN
578         n := 2;
579         x0 := ( - b + Math.Sqrt( D ) ) / ( 2 * a );
580         x1 := ( - b - Math.Sqrt( D ) ) / ( 2 * a );
581       ELSIF D < 0 THEN
582         n := 0;
583         x0 := INF;
584         x1 := INF;
585       ELSE
586         n := 1;
587         x0 := ( - b ) / ( 2 * a );
588         x1 := x0;
589       END
590     END EvalRoots;
591
592   PROCEDURE Eval*;
593   BEGIN
594     EvalRoots( data.a, data.b, data.c, data.nroots, data.x0, data.x1 );
595     Dialog.Update( data )
596   END Eval;
597
598   PROCEDURE Notifier* ( op, from, to: INTEGER );
599   BEGIN Eval
600   END Notifier;
601
602 BEGIN data.a := 1.0
603 END Kurs2002SimpleDialog.
604
605 Controls, New Form ...
606
```



607
608
609 просто копируем в поле Link имя модуля (Ctrl+C, Ctrl+V):
610

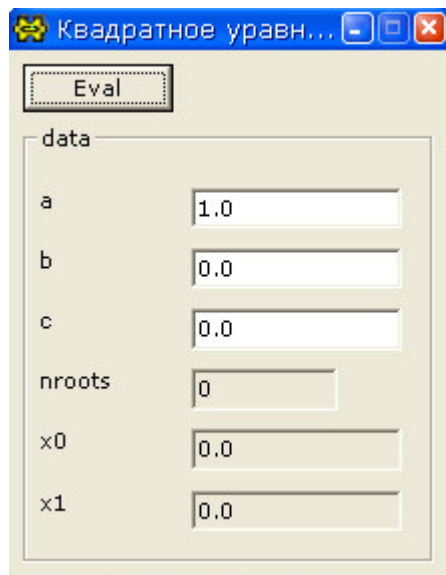


611
612
613 кликнуть OK, появится окошко:
614

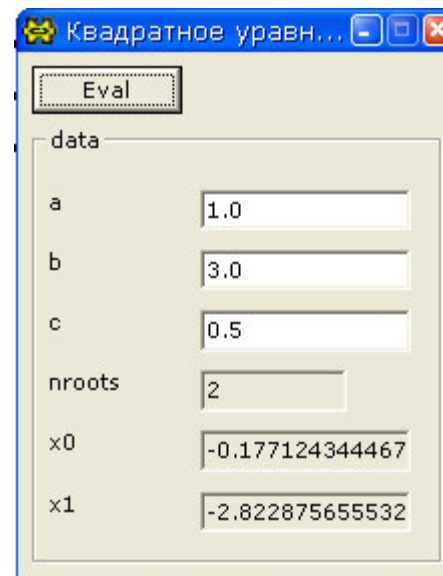


615
616
617 Потаскаем/порастягиваем элементы, если хочется.
618 Спасем эту форму как Kurs2002/Rsrc/SimpleDialog.odc (нужно будет создать
619 папку Kurs2002/Rsrc/).
620 Закроем окошко с формой (станд. образом: Ctrl+F4 и т.п.).

621 Вызов с помощью командера (это можно вставить в меню по станд. правилам):
622
623 "StdCmds.OpenAuxDialog('Kurs2002/Rsrc/SimpleDialog', 'Квадратное уравнение')"
624
625 **NB** Такой вызов можно сделать и из программы (без охватыв. двойных
626 кавычек).
627
628 Кликнем по командеру, откроется диалог:
629



630
631
632 Вводя разные коэффициенты и кликая по кнопке Eval, будем получать
633 результаты:
634



635
636
637 Закроем диалог (Ctrl+F4).
638

639 Простейшее редактирование

640
641 Откроем файл Kurs2002/Rsrc/SimpleDialog.odc, в который мы спасли форму
642 диалога.
643 Сделаем правый клик по кнопке Eval и выполним Properties... (или Ctrl+Enter).
644 Увидим:
645



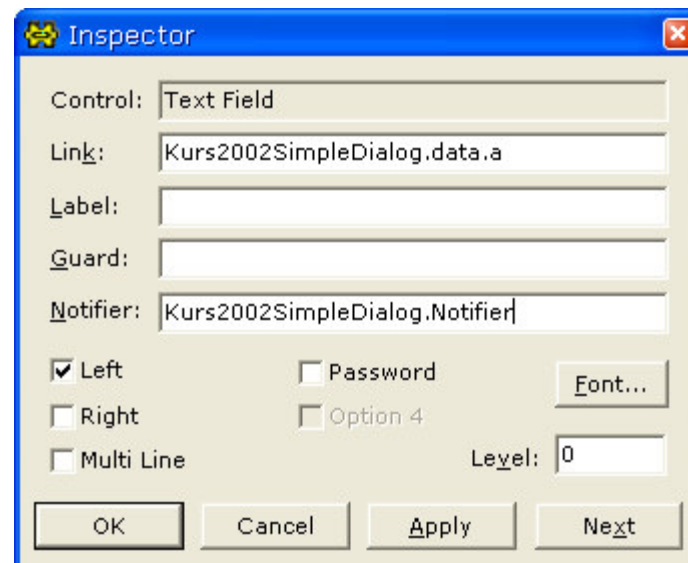
Изменим Eval --> Вычислить. OK. Ctrl+S. Ctrl+F4.
Снова кликнем по командеру вверху. Откроется измененный диалог.
Так же можно изменить любой элемент диалога.

Notifiers

Это "продвинутый" трюк.
Notifier = оповеститель = процедура со стандартной сигнатурой (как указано в примере), которая выполняет нечто сразу после каждой манипуляции с диалогом (клик, нажатие клавиши и т.п).
Что именно произошло -- в параметрах. Не разбираемся.
Просто просим обновлять все поля (Eval, а там внутри Update).

Откроем файл Kurs2002/Rsrc/SimpleDialog.odc. Кликнем по Вычислить, нажмем Delete.
Тут же спасем форму в новый файл Kurs2002/Rsrc/SimpleDialogN.odc.
Ctrl+A. Ctrl+<стрелка вверх 5 раз>.
Layout, Recalc Focus Size, Ctrl+S.

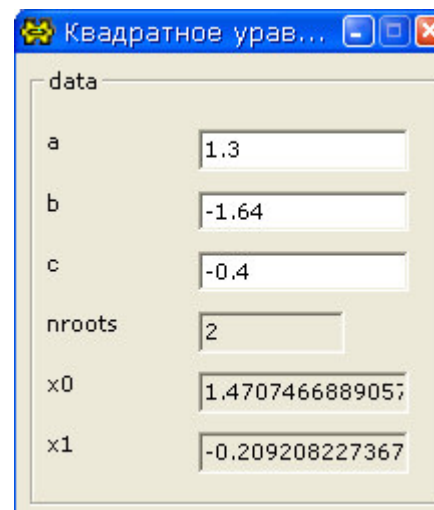
Для каждого каждого коэф-та a, b, c:
кликнуть по соотв. белому полю, Ctrl+Enter, напечатать в поле Notifier следующее:



OK. Ctrl+S. Ctrl+F4.

Вызовем новый диалог:

"StdCmds.OpenAuxDialog('Kurs2002/Rsrc/SimpleDialogN', 'Квадратное уравнение')"



В новом диалоге уже не нужно нажимать никаких кнопок: результаты обновляются на лету.

684 **Что еще можно делать с диалогами в ББ?**
685 — Генерировать форму динамически. Более того, можно на лету менять форму,
686 с которой взаимодействует клиент. (Разумеется, требуется внимательно
687 рассчитывать геометрию элементов.)
688 — Можно в форме иметь, например, график: картинка может меняться в
689 зависимости от нажатия кнопок и т.п. (В общем случае форма — это просто
690 список т.наз. "вьюшек" — объектов типов-расширений абстрактного типа View.
691 Чаще всего специальные View: Controls.Control, но не обязательно.
692