

С/к Введение в современное программирование (v.5.5) Физфак МГУ. 2006/7 уч. год. Лекция 10

Из метафизики

Еще о "постановке техники": "Воспитание есть **отучение от произвола** и приучение .. к дисциплине, к закону." --И.А.Ильин

Еще о рекламе и стадных эффектах: "... Когда разные виды ранее неизвестных раздражителей – заузные слова, фрагменты мелодий, фотографии незнакомых людей – просто находятся в поле зрения, это само по себе заставляет людей оказывать предпочтение именно им. ... Какие из букв алфавита – ваши любимцы? Самые разные люди, вне зависимости от их возраста, языка и национальности, предпочитают буквы, которые содержатся в их собственном имени или те, которые являются частотными в их родном языке. ... Французские студенты называют самой нелюбимой буквой заглавную W, то есть букву, наименее употребительную в письменном французском. ... После многократных повторений **называния** товара покупатели совершенно автоматически одобрительно отзываются о рекламируемом продукте. Если кандидаты малоизвестны, побеждают обычно те, кого чаще упоминают в СМИ. ..."

Еще о стадах: "Глубина заблуждений не уменьшается от широты их распространенности."

Еще о "системах без памяти" (к каковым относится сфера ИТ): "... отсутствие исторической памяти лишает осознания дальней перспективы, делает электорат **весьма манипулируемым**. ..."

Еще о минимализме:

"... Хороший рассказ не тот куда нечего добавить, а тот из которого ничего нельзя выбросить." --И.Э.Бабель
"... Они одеты были по строгим правилам лучшего вкуса: ничего лишнего." -- М.Ю.Лермонтов "Герой нашего времени"

О мифах:

"... <один из основоположников культурологии> Элиаде справедливо подчеркивал, что степень мифологичности сознания современного человека ничуть не уменьшилась по сравнению с первобытными временами, только проявления этой мифологичности приняли другие внешние формы ... Хорошо это или плохо, но с этим фактом нужно считаться."

Об изучении Оберона/КП: "Мудр тот, кто знает не многое, а нужное." --Эсхил
Q Что такое "нужное"?

В прошлой лекции: научились делать список, бродить по списку и находить в нем что угодно.

```
52 TYPE
53   Link = POINTER TO LinkDesc;
54   LinkDesc = RECORD
55     next: Link;
56     n: INTEGER;
57   END;
58 ...
59 VAR first: Link;
```

Научились искать любой (в т.ч. последний) элемент.

```
61
62
63 PROCEDURE Find ( first: Link; x: INTEGER ): Link;
64   VAR l: Link;
65 BEGIN
66   l := first;
67   WHILE ( l # NIL ) & ~( l.n = x ) DO
68     l := l.next
69   END;
70   ASSERT( ( l = NIL ) OR ( l.n = x ), 60 );
71   (* ( l # NIL ) => l.n = x *)
72   RETURN l
73 END Find;
```

+Упр* Написать процедуру, составляющую упорядоченный по возрастанию список из чисел в потоке ввода.

```
74
75
76
77
78
79 PROCEDURE Last ( first: Link ): Link;
80   VAR l: Link;
81 BEGIN
82   l := first;
83   WHILE ( l # NIL ) & ~( l.next = NIL ) DO
84     l := l.next
85   END;
86   ASSERT( ( l = NIL ) OR ( l.next = NIL ), 60 );
87   (* ( l # NIL ) => l последний *)
88   RETURN l
89 END Last;
```

Добавление эл-та в конец-0

```
91
92
93 PROCEDURE Append ( VAR first: Link; x: INTEGER );
94   VAR new, last: Link;
95 BEGIN
96   NEW( new ); new.x := x;
97   IF first = NIL THEN
98     first := new
99   ELSE
100    last := Last( first );
101    ASSERT( ( last # NIL ) & ( last.next = NIL ) );
102    last.next := new;
103  END;
104 END Append;
```

105 Пример:

```
106 PROCEDURE Do10*;
107   VAR last: Link; x: INTEGER;
108   BEGIN
109     In.Open; ASSERT( In.Done );
110     In.Int( x );
111     WHILE In.Done DO
112       Append( first, x );
113       In.Int( x );
114     END;
115     Log
116   END Do10;
```

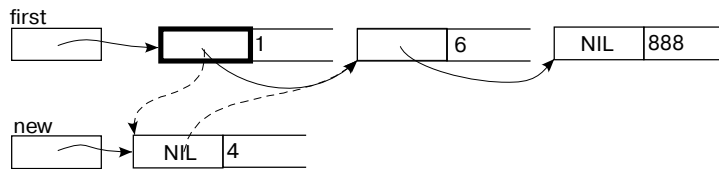
117 !Kurs2006Example34Lists0.Do10 1 6 888!

118 Kurs2006Example34Lists0 unloaded
119 1 6 888
120

121 **NB** А если список длинный, и/или надо прицепить много? Отдельная песня с вариациями.

124 Вставка эл-та в середину списка

125 Вставим 4 в список 1, 6, 888, не нарушив упорядоченности (**sic**).



126

127 Пунктирные стрелки — что должно быть.

128 Нужно иметь доступ к жирному полю.

129 Ограничение поиска: l.next = NIL.

130 Условие поиска: l.next.x >= 4.

131 Не забыть проверить первый эл-т!!!

```
132
133 PROCEDURE Insert ( VAR f: Link; x: INTEGER );
134   VAR new, l: Link;
135   BEGIN
136     NEW( new ); new.x := x;
137     IF f = NIL THEN
138       f := new
139     ELSIF f.x >= x THEN
140       new.next := f; f := new
141     ELSE
142       l := f;
143       WHILE ~( l.next = NIL ) & ~( l.next.x >= x ) DO
144         l := l.next
145       END;
146       IF l.next = NIL THEN
147         (* прицепляем в конце: *)
148         l.next := new
149       ELSE
```

```
150         (* стандартная вставка: *)
151         new.next := l.next; l.next := new
152       END;
153       ASSERT( l.n <= new.n );
154     END
155   END Insert;
```

```
156
157 PROCEDURE Do11*;
158   VAR last: Link; x: INTEGER;
159   BEGIN
160     last := Last( first );
161     In.Open; ASSERT( In.Done );
162     In.Int( x );
163     WHILE In.Done DO
164       Insert( first, x );
165       In.Int( x );
166     END;
167     Print( first ); StdLog.Ln
168   END Do11;
169 !Kurs2005Lists4.Do11 4 5 6 1 2 3 7 8
170 Kurs2006Example34Lists0 unloaded
171 1 2 3 4 5 6 7 8
```

172 **NB** Код все время усложняется из-за разниц: первый/последний/серединный. --
173 > отдельная темка.

175 Исключение эл-та из списка

176 Исключить элемент, следующий за данным:

```
177 PROCEDURE DropAfter ( del: Link );
178   VAR l: Link;
179   BEGIN
180     ASSERT( del # NIL, 20 );
181     IF del.next # NIL THEN
182       del.next := del.next.next
183     END;
184   END DropAfter;
```

185 **Упр** Нарисовать картинки (на доске).

186 Исключить данный:

```
187 PROCEDURE DropLink ( VAR del: Link );
188   VAR l: Link;
189   BEGIN
190     IF del # NIL THEN
191       l := del;
192       del := del.next
193     END;
194   END DropLink;
```

195 **NB** Передавать нужно соответствующее поле:

```
196 DropLink( first );
197 DropLink( l.next ); — исключаем эл-т, следующий за l.
```

198 **+Упр*** Исключить из списка все отрицательные числа за один проход.

199 "Съедание" списка

```

200  Еще одно удобство списка: можно перегруппировать эл-ты, поменяв только
201  ссылки, не пересылая само содержимое (может быть много-много байт...)
202  Например, растолкаем эл-ты неупорядоченного списка на две части:
203  положительные и отрицательные, а нули выбросим вообще.
204  Добавлять элемент умеем:
205      PROCEDURE PushLink ( VAR f: Link; new: Link );
206      BEGIN
207          ASSERT( new.next = NIL, 20 );
208          new.next := f; f := new
209      END PushLink;
210  Надо научиться "отрывать" от списка:
211      PROCEDURE PopLink ( VAR f: Link; OUT new: Link );
212      BEGIN
213          new := f;
214          IF new # NIL THEN
215              f := f.next; new.next := NIL
216          END;
217      END PopLink;
218  Вот разделение списка на положительные и отрицательные:
219      PROCEDURE Do12*;
220      VAR l: Link; pos, neg: Link; (* два будущих списка *)
221      BEGIN
222          first := NIL;
223          Do10; (* создали список из входного потока с сохранением порядка *)
224          (* схема полного прохода: *)
225          PopLink( first, l );
226          WHILE l # NIL DO
227              IF l.n > 0 THEN
228                  PushLink( pos, l )
229              ELSIF l.n < 0 THEN
230                  PushLink( neg, l )
231              ELSE
232                  l := NIL; (* для ясности *)
233              END;
234              PopLink( first, l )
235          END;
236          Print( pos ); (* Print отличается от Log наличием аргумента *)
237          Print( neg );
238      END Do12;
239  ! Kurs2006Example34Lists0.Do12 1 2 -9 0 3 -4 8 5 6 -7 0 !
240  Kurs2006Example34Lists0 unloaded
241  1 2 -9 0 3 -4 8 5 6 -7 0
242  6 5 8 3 2 1
243  -7 -4 -9
244
245  Упр* Написать процедуру, сортирующую список вставками (N^2).

```

```

246  Упр Уметь в любом состоянии делать следующее:
247  — Подсчет кол-ва эл-тов (N).
248  — Найти последний эл-т списка. (Что значит "найти"?)
249  — Найти эл-т списка с номером N DIV 2.
250  — Проверить, что заданный элемент принадлежит списку.
251  — Найти элемент, предшествующий данному.
252  — Удалить из списка заданный элемент.
253  — Печатать в Log содержимое каждого эл-та на четном месте списка.
254  — Печатать в Log списка в обратном порядке с помощью одной короткой
255  процедуры.
256  — Обратить порядок элементов в списке.
257  — Разбить список на два: на четных и нечетных местах в исходном списке;
258  положительных и отрицательных; четных и нечетных ...
259  — Удалить из списка все элементы-дубликаты.
260  — Распечатать все неупорядоч. пары эл-тов списка. (Что значит "неупорядоч.?" )
261  — Создать список неупорядоч. пар эл-то заданного списка.
262
263  Упр Используя списки, реализовать модуль нахождения средних, матрицы
264  корреляций, ошибок для измерений пар x, y, вводимых с помощью In, причем
265  можно только один раз вызвать In.Open.
266
267  "Игра в кубики" Списки списков, списки массивов и массивы списков —
268  свободное комбинирование. Единств. ограничение: физич. память (RAM, page
269  file на диске).
270
271  Сравним массивы и списки:
272  Списки менее экономны — но гораздо более динамичны и гораздо более гибки.
273  В простых (почти статических) задачах — массивы проще и эффективнее.
274  Чем больше в задаче динамики, тем проще решения со списками, а разница в
275  эффективности исчезает и меняет знак.
276
277  Проблема на скорое будущее Операции со списками (слабо) не
278  зависят от их содержимого. Как избежать повторения кода?
279  Например, алгебраические выражения могут быть самой разной природы, а
280  манипуляции (сортировки) в сущности одинаковые.
281  Это фундаментальная проблема в програмном проектировании — code
282  reuse (=повторное использование кода).
283

```

284 Псевдо-модуль SYSTEM

285 средства программирования низкого уровня -- драйверы и т.п.
 286 F1, Platform-Specific Issues
 287 Чтобы использовать, нужно IMPORT SYSTEM -- сигнал, что **все** свойства
 288 безопасности (статическая типизация и т.п.) могут быть нарушены!
 289
 290 Name Argument types Result type Description
 291 ADR(v) any INTEGER address of variable v
 292 ADR(P) P: PROCEDURE INTEGER address of Procedure P
 293 ADR(T) T: a record type INTEGER address of Descriptor of T
 294 LSH(x, n) x, n: integer type* type of x logical shift (n > 0: left, n < 0:
 295 right)
 296 ROT(x, n) x, n: integer type* type of x rotation (n > 0: left, n < 0: right)
 297 TYP(v) record type INTEGER type tag of record variable v
 298 VAL(T, x) T, x: any type T x interpreted as of type T
 299 * integer types without LONGINT
 300
 301 Name Argument types Description
 302 GET(a, v) a: INTEGER; v: any basic type, **v := M[a]**
 303 pointer type, procedure type
 304 PUT(a, x) a: INTEGER; x: any basic type, **M[a] := x**
 305 pointer type, procedure type
 306
 307
 308 Против лома нет приема...
 309
 310 специальные атрибуты:
 311 ARRAY [untagged] OF ...
 312 значит, нет дескриптора, в котором хранится длина; LEN Работать не будет --
 313 длину нужно сообщать в процедуру отдельным параметром -- как в фортране и Ц.
 314 Для общения с этими языками и предназначено.
 315 RECORD [union]
 316 n: INTEGER;
 317 s: SET
 318 END
 319 поля n и s ссылаются на одну и ту же четверку байтов.
 320 Можно написать n := -10 и посмотреть, что это означает на уровне отдельных
 321 битов (s).
 322
 323 Короче, есть полный набор средств для "взлома" всех средств защиты самого
 324 Компонентного Паскаля.
 325
 326 + можно делать процедуры из двоичных машинных команд... не смотрим.
 327
 328 Правильное отношение:
 329 использовать SYSTEM только под угрозой расстрела.
 330

331

332