

```

1: unit MedialRepStructure; (*Получен из StructureSkel*)
2:   (*Модуль содержит описание структуры скелета в терминах классы-объекты.
3:   Этот модуль наследует многие вещи из модуля StructureTD, поскольку
4:   исторически был написан после него. Модуль StructureSkel описывает
5:   скелет как подмножество диаграммы Вороного, а модуль StructureTD
6:   описывает граф смежности сайтов - обобщённую триангуляцию Делоне,
7:   являющуюся двойственным графом для диаграммы Вороного*)
8:
9: interface
10: USES graphics, classes, Math, SysUtils, Dialogs,
11:     Simset, BitMatrix, MedialRep;
12:
13: CONST MaxElem=100000; (*Число сайтов в контуре*)
14:
15: TYPE
16:   AMatr=ARRAY [0..50000000] OF BYTE;
17:   Link=SIMSET.Link;
18:   Head=SIMSET.Head;
19:   TBone = class;
20:   TConnected=CLASS;
21:   TContour=CLASS;
22:   TSite=CLASS;
23:
24:   ArrElem = ARRAY [0..MaxElem-1] OF TSite;
25:   AdArrElem = ^ArrElem;
26:
27:   ResultTable = RECORD (*Таблица для формирования статистики вычислений*)
28:     BMPm, BMPn: INTEGER; (* размеры образа строки - столбцы *)
29:     Polygons: INTEGER; (* количество полигональных контуров *)
30:     ConnectComp: INTEGER; (* количество связанных компонент *)
31:     Points: INTEGER; (* количество вершин в полигонах *)
32:     Vertex: INTEGER; (* количество вершин в скелете *)
33:     Edges: INTEGER; (* количество ребер в скелете *)
34:     Parabol: INTEGER; (* количество параболических рёбер в скелете*)
35:     TimeTrace: CARDINAL; (* время трассировки *)
36:     TimeTree: CARDINAL; (*Время построения дерева смежности контуров*)
37:     TimeSkelet: CARDINAL; (* время скелетизации *)
38:     TimePrun: CARDINAL; (* время стрижки*)
39:     Total: CARDINAL; (* время общее*)
40:   END;
41:
42:   NodeKind = (Isolated, TailNode, JointNode, ForkNode);
43:   {Возможные типы узла: изолированный, хвост, сустав,
44:   развилка - по числу выходящих костей 0,1,2,3 }
45:
46:   TDisc = CLASS (Link) (* Окружность *)
47:     X, Y, Rad: DOUBLE; (* Координаты центра и радиус *)
48:     HalfPlane: BOOLEAN; (* Признак вырожденного круга - полуплоскости*)
49:     CONSTRUCTOR Create(XC, YC, Radius: DOUBLE);
50:   END;
51:
52:   (*Здесь термин Элемент используется в смысле Сайт - старая терминология*)
53:   TSite = CLASS (Link) (* Элемент коллекции (сайт) - вершина или ребро *)
54:     Nel: INTEGER; (*Номер*)
55:     Cont: TContour; {Контур, в котором находится сайт}
56:   END;
57:
58:   TPoint = CLASS (Link) (* Точка на плоскости *)
59:     PROTECTED
60:       u, v: DOUBLE; (* Координаты *)
61:     PUBLIC
62:       Number: INTEGER; (*Порядковый номер*)
63:       PROPERTY X: DOUBLE READ u WRITE u;

```

```

64:         PROPERTY Y: DOUBLE READ v WRITE v;
65:         CONSTRUCTOR Create(x,y: DOUBLE);
66:     END;
67:
68: TVertex = CLASS (TSite)(* Сайт-точка *)
69:     p: TPoint;          (* Точка *)
70:     Reflex: BOOLEAN; (*TRUE - угол при вершине больше 180 град*)
71:     CONSTRUCTOR Create(OwnPoint: TPoint);
72: //     FUNCTION ConformWithDisc(Cr: TDisc): BOOLEAN;
73:     (* Согласованность с диском: он не "за спиной" *)
74:     END;
75:
76: TEdge = CLASS (TSite) (* Сайт-сегмент *)
77:     org,dest: TPoint;    (* Точки - концы сегмента *)
78:     PROPERTY Origin: TPoint READ org WRITE org;
79:     PROPERTY Destin: TPoint READ dest WRITE dest;
80:     CONSTRUCTOR Create(p1,p2: TPoint);
81: //     FUNCTION WestDirect: BOOLEAN;
82:     DESTRUCTOR Destroy; OVERRIDE;
83:     (*западная направленность*)
84:     END;
85:
86: TNode = class (Link)(*Элемент скелета типа 'узел'*)
87:     Number: INTEGER; (*Порядковый номер*)
88:     Disc: TDisc; (*Пустой круг узла*)
89:     Sites: array[1..3] of TSite; (*Массив инцидентных сайтов*)
90:     Bones: ARRAY [1..3] OF TBone; (*массив инцидентных костей*)
91:     Depth: DOUBLE; (*глубина, используемая при стрижке*)
92:     Merge: TBone; (* Кость от присоединяемого штриха *)
93:     FUNCTION X: DOUBLE; // координаты
94:     FUNCTION Y: DOUBLE; // узла
95:     FUNCTION R: DOUBLE; // и радиус его пустого круга
96:     CONSTRUCTOR Create;
97:     DESTRUCTOR Destroy; override;
98:     PROCEDURE AddBone(Bone: TBone); (*добавить кость*)
99: end;
100:
101: TBone = class(Link)(*Элемент скелета типа 'кость'*)
102:     Com: TConnected; (*компонента, к которой относится кость*)
103:     Org, Dest: TNode; (*Начальный и конечный узлы кости*)
104:     Met: BOOLEAN;      (* Метка *)
105:     Virt: TPoint; (*Контрольная точка для ребра-параболы, NIL для отрезка*)
106:     CONSTRUCTOR Create;
107:     DESTRUCTOR Destroy; override;
108: end;
109:
110: TConnected=CLASS(Link)
111:     (*Связная компонента многоугольной фигуры *)
112:     Border: TContour; (*Внешний контур компоненты*)
113:     Nodes: Head; (*Список узлов скелета компоненты*)
114:     Bones: Head; (*Список костей скелета компоненты*)
115:     (*скелет компненты - только то, что лежит внутри нее*)
116:     HoleList: TList; (*Список дыр - внутренних контуров компоненты*)
117:     Area: DOUBLE; (*площадь компоненты*)
118:     CONSTRUCTOR Create;
119:     DESTRUCTOR Destroy; OVERRIDE;
120:     END;
121:
122: Domain = CLASS (* Непрерывный бинарный объект (черный на белом фоне)
123:     Boundary: Head; (* Список замкнутых контуров - границ объекта *)
124:     ElementsExist: BOOLEAN; (* TRUE - Списки элементов составлены *)
125:     MapExist: BOOLEAN;      (* TRUE - Триангуляция построена *)
126:     CONSTRUCTOR Create;

```

```

127:     DESTRUCTOR Destroy; OVERRIDE;
128:     PROCEDURE ProduceElements;
129: END;
130:
131: TPolFigure = class (Domain)
132:     (*Граничное и скелетное представление бинарной области *)
133:     (*Описание скелета области в виде списков узлов и костей*)
134:     Components: Head; (*Список компонент многоуг. фигуры*)
135:     SkelExist: BOOLEAN; (*Признак, что скелет построен*)
136:     InternalSkel: BOOLEAN; (*Текущий скелет - внутренний*)
137:     RTab: ResultTable; (*Статистика вычислений*)
138:     PROCEDURE ClearAll; (*Чистка всех списков*)
139:     DESTRUCTOR Destroy; OVERRIDE;
140:     CONSTRUCTOR CreateFromStructure(BS: PBondarySkeleton);
141: end;
142:
143: TContour = CLASS (Link) (*Замкнутый контур - граница области. Область слева*)
144:     Number: INTEGER; (*Порядковый номер*)
145:     ListPoints: Head; (*Список точек*)
146:     ListElements: Head; (*Список элементов*)
147:     Elements: AdArrElem {^ArrElem}; (*Массив элементов, составляющих контур*)
148:     NumElem: INTEGER; (*Количество элементов*)
149:     Internal: BOOLEAN; (*TRUE - внутренний (по ЧС), FALSE - внешний (против ЧС)*)
150:     AreaCont: DOUBLE; (*Площадь многоугольника внутри контура*)
151:     ClosedPath: BOOLEAN; (*Признак замкнутого контура*)
152:     Container: TContour; (*Контур, объемлющий SELF*)
153:     MySons: Head; (*Контура, содержащиеся внутри SELF*)
154:     WestElement: TSite; (*самая левая точка в контуре*)
155:     ClosestSite: TSite; (*ближайший сайт слева вне контура*)
156:     Fiction: BOOLEAN; (*Фиктивный внешний контур для внешнего скелета*)
157:     CONSTRUCTOR Create;
158:     DESTRUCTOR Destroy; OVERRIDE;
159:     PROCEDURE CreateElements(VAR Number: INTEGER);
160:         (*Построение массива элементов с номерами, начиная с Number*)
161: END ;
162:
163:
164: FUNCTION TimeInMilSecond: CARDINAL; (*Текущее время в миллисекундах*)
165:
166: VAR NodeCount, BoneCount, VirtCount, CompCount: INTEGER;
167:     (*Число узлов, костей и компонент*)
168:     Matr: ^AMatr;
169:
170: implementation
171:
172:     FUNCTION TimeInMilSecond: CARDINAL;
173:         (*Текущее время в миллисекундах*)
174:     VAR TT: TDateTime;
175:         Hour, Min, Sec, MSec: Word;
176:     BEGIN
177:         TT:=Time;
178:         DecodeTime(TT, Hour, Min, Sec, MSec);
179:         Result:= MSec+1000*(Sec+60*(Min+60*Hour));
180:     END;
181:
182:     FUNCTION TNode.X: DOUBLE;
183:     BEGIN Result:=Disc.X END;
184:
185:     FUNCTION TNode.Y: DOUBLE;
186:     BEGIN Result:=Disc.Y END;
187:
188:     FUNCTION TNode.R: DOUBLE;
189:     BEGIN Result:=Disc.Rad END;

```

```

190:
191:   CONSTRUCTOR TConnected.Create;
192: BEGIN
193:   INHERITED Create;
194:   Bones:=Head.Create;
195:   Nodes:=Head.Create;
196:   HoleList:=TList.Create;
197:   INC(CompCount);
198: END;
199:
200: CONSTRUCTOR TNode.Create;
201: VAR i: INTEGER;
202: BEGIN
203:   INHERITED Create;
204:   FOR i:=1 TO 3 DO Bones[i]:=NIL;
205:   Disc:=NIL;
206:   INC(NodeCount);
207: END;
208:
209: DESTRUCTOR TNode.Destroy;
210: BEGIN out;
211:   DEC(NodeCount);
212:   IF Disc<>NIL THEN Disc.Destroy;
213:   INHERITED Destroy;
214: END;
215:
216: CONSTRUCTOR TBone.Create;
217: BEGIN
218:   INHERITED Create;
219:   Com:=NIL;
220:   Virt:=NIL;
221:   INC(BoneCount);
222: END;
223:
224: DESTRUCTOR TBone.Destroy;
225: BEGIN out;
226:   DEC(BoneCount);
227:   IF Virt<>NIL THEN
228:     BEGIN
229:       Virt.Destroy;
230:       DEC(VirtCount);
231:     END;
232:   INHERITED Destroy;
233: END;
234:
235: DESTRUCTOR TConnected.Destroy;
236:   (*УНИЧТОЖЕНИЕ КОМПОНЕНТЫ*)
237: VAR B: TBone;
238:   N: TNode;
239: BEGIN
240:   HoleList.Clear;
241:   HoleList.Destroy;
242:   WHILE NOT Bones.empty DO
243:     BEGIN B:=Bones.first AS TBone;
244:       B.out; B.Destroy;
245:     END;
246:   WHILE NOT Nodes.empty DO
247:     BEGIN N:=Nodes.first AS TNode;
248:       N.out;
249:       N.Destroy;
250:     END;
251:   Bones.Destroy;
252:   Nodes.Destroy;

```

```

253:     DEC(CompCount);
254:     INHERITED Destroy;
255: END;
256:
257: PROCEDURE TPolFigure.ClearAll;
258: VAR Com: TConnected;
259: BEGIN (*уничтожение компонент*)
260:     WHILE NOT Components.empty DO
261:     BEGIN
262:         Com:=Components.first AS TConnected;
263:         Com.Destroy;
264:     END;
265: END;
266:
267: DESTRUCTOR TPolFigure.Destroy;
268: BEGIN
269:     ClearAll;
270:     Components.Destroy;
271:     Components:=NIL;
272:     INHERITED Destroy;
273: END;
274:
275: PROCEDURE TNode.AddBone(Bone: TBone); (*добавить кость*)
276: BEGIN
277:     IF Bones[1]=NIL THEN Bones[1]:=Bone
278:     ELSE IF Bones[2]=NIL THEN Bones[2]:=Bone
279:     ELSE IF Bones[3]=NIL THEN Bones[3]:=Bone;
280: END;
281:
282: PROCEDURE Domain.ProduceElements;
283: VAR S: TContour;
284:     i: INTEGER;
285: BEGIN
286:     i:=0;
287:     S:=Boundary.first AS TContour;
288:     WHILE S<>NIL DO
289:     BEGIN
290:         S.CreateElements(i);
291:         S:=S.suc AS TContour;
292:     END;
293: END;
294:
295: CONSTRUCTOR TDisc.Create(XC,YC,Radius: DOUBLE);
296: BEGIN
297:     INHERITED Create;
298:     X:=XC; Y:=YC; Rad:=Radius;
299: END;
300:
301: CONSTRUCTOR TPoint.Create(x,y: DOUBLE);
302: BEGIN
303:     INHERITED Create;
304:     u:=x; v:=y;
305: END;
306:
307: CONSTRUCTOR TVertex.Create(OwnPoint: TPoint);
308: VAR PP,PS: TPoint;
309: BEGIN
310:     INHERITED Create;
311:     p:=OwnPoint;
312:     PP:=p.predring AS TPoint;
313:     PS:=p.sucring AS TPoint;
314:     Reflex:=(PS=PP) OR
315:     ((p.X-PP.X)*(PS.Y-p.Y)<(p.Y-PP.Y)*(PS.X-p.X));

```

```

316:      END;
317:
318:  CONSTRUCTOR TEdge.Create(p1,p2: TPoint);
319:  BEGIN
320:      INHERITED Create;
321:      org:=p1; dest:=p2;
322:  END;
323:
324:  DESTRUCTOR TEdge.Destroy;
325:  BEGIN
326:      INHERITED Destroy;
327:  END;
328:
329:  CONSTRUCTOR Domain.Create;
330:  BEGIN
331:      INHERITED Create;
332:      Boundary:=Head.Create;
333:  END;
334:
335:  DESTRUCTOR Domain.Destroy;
336:  VAR C: TContour;
337:  BEGIN
338:      IF Boundary<>NIL THEN
339:          BEGIN
340:              WHILE NOT Boundary.empty DO
341:                  BEGIN C:=Boundary.first AS TContour;
342:                      C.out; C.Destroy;
343:                  END;
344:              Boundary.Destroy;
345:          END;
346:          INHERITED Destroy;
347:      END;
348:
349:  CONSTRUCTOR TContour.Create;
350:  BEGIN
351:      INHERITED Create;
352:      ListPoints:=Head.Create;
353:      ListElements:=Head.Create;
354:      ClosedPath:=FALSE;
355:      MySons:=NIL;
356:      ClosestSite:=NIL;
357:      Fiction:=FALSE;
358:  END;
359:
360:  DESTRUCTOR TContour.Destroy;
361:  VAR P: TPoint;
362:      i: INTEGER;
363:      V: TVertex;
364:      Ed: TEdge;
365:      E: TSite;
366:  BEGIN
367:      IF NumbElem>0 THEN
368:          BEGIN
369:              FOR i:=0 TO NumbElem-1 DO
370:                  BEGIN E:=Elements^[i];
371:                      IF E IS TVertex THEN
372:                          BEGIN
373:                              V:=E AS TVertex;
374:                              V.Destroy;
375:                          END
376:                      ELSE IF E IS TEdge THEN
377:                          BEGIN
378:                              Ed:=E AS TEdge;

```

```

379:         Ed.Destroy;
380:     END
381:     ELSE
382:         MessageDlg('Ошибка: неверный элемент',
383:             mtCustom,[mbOK],0);;
384:     END;
385:     FreeMem(Elements, NumbElem*SIZEOF(Pointer));
386: END;
387: WHILE NOT ListPoints.empty DO
388: BEGIN P:=ListPoints.first AS TPoint;
389:     P.out; P.Destroy;
390: END;
391: ListPoints.Destroy;
392: IF NOT ListElements.empty THEN
393:     MessageDlg('Ошибка NOT ListElements.empty',
394:         mtCustom,[mbOK],0);
395: ListElements.Destroy;
396: IF (MySons<>NIL) AND NOT (MySons.empty) THEN
397:     MessageDlg('Ошибка sons<>NIL and NOT empty',
398:         mtCustom,[mbOK],0);
399: IF MySons<>NIL THEN
400:     MySons.Destroy;
401: INHERITED Destroy;
402: END;
403:
404: PROCEDURE TContour.CreateElements(VAR Number: INTEGER);
405:     (* Построение массива элементов по списку точек *)
406: VAR P,Q,PFirst: TPoint;
407:     n: INTEGER;
408:     V: TVertex;
409:     Ed: TEdge;
410: BEGIN
411:     IF NumbElem<>0 THEN
412: BEGIN
413:         FOR n:=0 TO NumbElem-1 DO Elements^[n].Destroy;
414:         FreeMem(Elements, NumbElem*SIZEOF(Pointer));
415:         NumbElem:=0;
416:     END;
417:     n:=ListPoints.cardinal*2;
418:     GetMem(Elements, n*SIZEOF(Pointer));
419:
420:     (*Образование элементов*)
421:     PFirst:=ListPoints.first AS TPoint;
422:     P:=PFirst.suc AS TPoint;
423:     WHILE P<>NIL DO
424: BEGIN
425:         IF (P.u<PFirst.u) OR (P.u=PFirst.u) AND (P.v<PFirst.v) THEN
426:             PFirst:=P;
427:         P:=P.suc AS TPoint;
428:     END;
429:     WHILE TRUE DO
430: BEGIN
431:         P:=ListPoints.first AS TPoint;
432:         IF P<>PFirst THEN P.into(ListPoints)
433:         ELSE Break;
434:     END;
435:
436:     P:=ListPoints.first AS TPoint;
437:     WHILE P<>NIL DO
438: BEGIN
439:         Q:=P.sucring AS TPoint;
440:         V:=TVertex.Create(P);
441:         V.Cont:=SELF;

```

```

442:         Elements^[NumbElem]:=V;
443:         V.NEl:=Number; INC(Number);
444:         INC(NumbElem);
445:         V.into(ListElements);
446:         Ed:=TEdge.Create(P,Q);
447:         Ed.Cont:=SELF;
448:         Ed.NEl:=Number; INC(Number);
449:         Elements^[NumbElem]:=Ed;
450:         INC(NumbElem);
451:         Ed.into(ListElements);
452:         P:=P.suc AS TPoint;
453:     END;
454:     WestElement:=Elements^[0];
455: END;
456:
457: CONSTRUCTOR TPolFigure.CreateFromStructure(BS: PBondarySkeleton);
458:     (*Построение объекта TPolFigure на основе
459:      "гранично-скелетного описания фигуры" *)
460: VAR   i,j,k: INTEGER;
461:       S: TContour;
462:       Corn: TPoint;
463:       Com: TConnected;
464:       Node,NodeAdj: TNode;
465:       Bone: TBone;
466:       CornerRec: TCornerRec;
467:       NodeRec: TNodeRec;
468:       BoneRec: TBoneRec;
469:       SiteRec: TSiteRec;
470:       TempContour,TempComp,TempNodes: TList;
471: BEGIN
472:     INHERITED Create;
473:     Components:=Head.Create;
474:     (*Порождение границы*)
475:     TempContour:=TList.Create;
476:     FOR i:=0 TO BS^.OutlineCount-1 DO
477:     BEGIN
478:       S:=TContour.Create;
479:       TempContour.Add(S);
480:       S.into(Boundary);
481:       S.Internal:=(BS.Boundary^[i].Intern=1);
482:       FOR j:=0 TO BS.Boundary^[i].CornerCount-1 DO
483:       BEGIN
484:         CornerRec:=BS.Boundary^[i].CornerList^[j];
485:         Corn:=TPoint.Create(CornerRec.Xcorner,CornerRec.Ycorner);
486:         Corn.into(S.ListPoints);
487:       END;
488:     END;
489:     ProduceElements;
490:
491:     (*Порождение компонент*)
492:     TempComp:=TList.Create;
493:     TempNodes:=TList.Create;
494:     FOR i:=0 TO BS.ComponentCount-1 DO
495:     BEGIN
496:       Com:=TConnected.Create;
497:       TempComp.Add(Com);
498:       Com.into(Components);
499:       (*Формирование границ компоненты*)
500:       Com.Border:=TempContour.Items[BS.Components^[i].Border];
501:       FOR j:=0 TO BS.Components^[i].HolesCount-1 DO
502:         Com.HoleList.Add(TempContour.Items[BS.Components^[i].Holes^[j]]);
503:       (*Формирование списка узлов*)
504:       FOR j:=0 TO BS.Components^[i].NodesCount-1 DO

```



```

505:      BEGIN
506:      Node:=TNode.Create;
507:      Node.Number:=j;
508:      NodeRec:=BS.Components^[i].Nodes^[j];
509:      Node.Disc:=TDisc.Create(NodeRec.Xnode,NodeRec.Ynode,NodeRec.Radius);
510:      TempNodes.Add(Node);
511:      Node.into(Com.Nodes);
512:      (*Формирование списка инцидентных сайтов*)
513:      FOR k:=0 TO 2 DO
514:      BEGIN
515:      SiteRec:=NodeRec.Site[k];
516:      IF SiteRec.Corner=-1 THEN
517:      Node.Sites[k+1]:=NIL
518:      ELSE
519:      BEGIN
520:      S:=TempContour.Items[SiteRec.Outline];
521:      Node.Sites[k+1]:=S.Elements^[SiteRec.Corner*2];
522:      END;
523:      END;
524:      END;
525:      (*Формирование списка рёбер*)
526:      FOR j:=0 TO BS.Components^[i].BonesCount-1 DO
527:      BEGIN
528:      Bone:=TBone.Create;
529:      Bone.Com:=Com;
530:      BoneRec:=BS.Components^[i].Bones^[j];
531:      Bone.Org:=TempNodes.Items[BoneRec.Origin];
532:      Bone.Dest:=TempNodes.Items[BoneRec.Destin];
533:      BoneRec:=BS.Components^[i].Bones^[j];
534:      IF BoneRec.Control<>-1 THEN
535:      BEGIN
536:      CornerRec:=BS.Components^[i].Controls^[BoneRec.Control];
537:      Bone.Virt:=TPoint.Create(CornerRec.Xcorner,CornerRec.Ycorner);
538:      END;
539:      Bone.into(Com.Bones);
540:      Node:=TempNodes.Items[BoneRec.Origin];
541:      NodeAdj:=TempNodes.Items[BoneRec.Destin];
542:      Node.AddBone(Bone);
543:      NodeAdj.AddBone(Bone);
544:      END;
545:      TempNodes.Clear;
546:      END;
547:      SkelExist:=TRUE;
548:      TempContour.Free;
549:      TempComp.Free;
550:      TempNodes.Free;
551:      // Result:=0;
552:      END;
553:
554:      end.

```