A TObjectList használata Delphiben

Bevezető

Az adatstruktúráknak is sok figyelmet szentelnek újabban, bizonyos körökben egyre népszerűbbek az absztrakt adattípusok, pointerek, listák, stringek. Wirth (a fent említett hátulgombolós) képes volt megírni egy egész könyvet erről, amiben azt mondja, hagy a programot nem akárhogy kell megírni, hanem az adatstruktúrákra kell építeni. Viszont amint ezt minden igazi programozó tudja, az egyetlen használható adatstruktúra a tömb. Stringek, listák, struktúrák, halmazok: ezek mind a tömb speciális esetei, és ugyanúgy is kezelhetjük őket, anélkül, hogy a programozási nyelvünket túlbonyolítanánk velük. A legrosszabb mindamellett ezekben az adatstruktúrákban az, hogy ráadásul külön deklarálni is kell őket, pedig az igazi programozási nyelvekben, amint azt mindannyian tudjuk, implicit típusdefiníció van a (6 betűs) változónév első karaktere alapján.
Az igazi programozó, http://www.cab.u-szeged.hu/local/doc/UNIX/orlando/igazi.html

Listák a Turbo Delphiben

A listákat a tömbökhöz hasonlóan tudjuk kezelni. A lista, mint adatszerkezet egyik fő előnye a tömbbel szemben, hogy nem kell előre meghatározni a méretét, hanem az az új elemek hozzáadásával folyamatosan növekszik. A Turbo Delphi többféle listaosztályt is biztosít (a Classes és a Contnrs unitokban), ezek közül most a TObjectList-et emeljük ki.

A TObjectList osztály a Delphi Contnrs unitjában található, és objektumok listáját lehet kezelni segítségével. Tapasztalataim szerint a TList osztály, a TObjectList őse, a listát egy pointereket tartalmazó tömbbel valósítja meg, ezzel is bizonyítva a fenti idézet igazságát :-) A listában nem csak egy megadott típusba tartozó elemeket tárolhatunk, hanem tetszőleges objektumokat (közös ősük ui. a TObject osztály).

Az alábbi példaprogramban elkészítünk egy TRobot nevű osztályt, amely egy egyszerű robotot valósít meg. A robotnak van neve, amit a konstruktorban lehet átadni, illetve ki tudja írni a nevét. A TRobot osztály az uRobot unitban található:

```
unit uRobot;

interface

type TRobot = class

private
  FName : string;

public
  property Name : string read FName;

  constructor Create(name : string);
  destructor Destroy; override;
  procedure MyName;
end;

implementation

{ TRobot }

constructor TRobot.Create(name: string);
begin
  self.FName:=name;
end;
```

```
destructor TRobot.Destroy;
begin
  Writeln(Name,' was destroyed...');

  inherited;
end;

procedure TRobot.MyName;
begin
  Writeln('My name is ',Name);
end;

end.
```

Ezután a TObjectList osztályból létrehozunk egy új listát, feltöltjük robotokkal (a rendezés kedvéért 5-től 1-ig sorszámozzuk őket visszafelé), majd egy for ciklussal bejárjuk azt. A lista OwnsObjects tulajdonságát true-ra állítjuk, ennek következtében a lista felszabadítása előtt nem kell törődnünk az egyes listaelemek felszabadításával, ui. azt a Delphi elvégzi helyettünk. Erre bizonyítékul szolgál a példaprogram futása során a robotok destruktorának meghívása.

A példaprogram a következő listaműveleteket és -tulajdonságokat szemlélteti:

```
procedure Clear();
```

A lista összes elemének törlése.

```
function Add(AObject: TObject): Integer;
```

A művelettel egy új elemet lehet a lista végére fűzni. A visszatérési érték az új elem indexe (az indexelés 0-tól kezdődik).

```
procedure Delete(Index: Integer);
```

A megadott indexű elemet törli a listából. Az indexelés 0-tól kezdődik, ha nem létező indexszel hívjuk meg az eljárást, akkor kivételt kapunk (List index out of bounds).

```
property Count: Integer read FCount write SetCount;
```

A Count property a lista elemeinek számát adja meg.

```
function First(): TObject;
```

A lista első elemét adja vissza. Ha üres listán hívjuk meg, akkor kivételt kapunk (List index out of bounds).

```
function Last(): TObject;
```

A lista utolsó elemét adja vissza. Ha üres listán hívjuk meg, akkor kivételt kapunk (List index out of bounds).

```
procedure Sort(Compare: TListSortCompare);
```

A lista rendezése QuickSort algoritmussal. A paraméterként átadandó Compare egy függvény,

amely két elemet hasonlít össze. A példában az egyik robot akkor nagyobb a másiknál, ha a neve nagyobb, ettől függően kell -1, 0 vagy 1 értéket visszaadni. A robots lista Sort metódusát a `robots.Sort(@Compare);` utasítással lehet meghívni.

```
function Compare(Item1, Item2 : TRobot) : integer;

  var res : integer;

begin
  if (Item1.Name = Item2.Name) then res:=0
  else if (Item1.Name < Item2.Name) then res:=-1
  else res:=1;

  Result:=res;
end;
```

A teljes példaprogram és a futási eredmény a következő:

```
program RobotList;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  Contnrs,
  uRobot in 'uRobot.pas';

procedure ShowRobots(robots : TObjectList);

  var i : integer;

begin
  // A lista elemeinek szama
  Writeln('Count: ',robots.Count);

  // A lista bejarasa
  for i:=0 to robots.Count - 1 do

    begin
      if (robots[i] is TRobot) then

        (robots[i] as TRobot).MyName;
    end;

  Writeln;
end;

function Compare(Item1, Item2 : TRobot) : integer;

  var res : integer;

begin
  if (Item1.Name = Item2.Name) then res:=0
  else if (Item1.Name < Item2.Name) then res:=-1
  else res:=1;

  Result:=res;
end;

var robots : TObjectList;
    i      : integer;

begin
  // Lista letrehozasa
  Writeln('Creating...');
  robots:=TObjectList.Create;
  robots.OwnsObjects:=true;

  // Lista feltoltese robotokkal
  for i := 1 to 5 do
```

```
    begin
      robots.Add(TRobot.Create('Robot' + IntToStr(6 - i)));
    end;

  ShowRobots(robots);

  Writeln('First and last...');
  // Elso elem elerese
  (robots.First as TRobot).MyName;
  // Utolso elem elerese
  (robots.Last as TRobot).MyName;
  Writeln;

  // Torles a listabol
  Writeln('Deleting...');
  robots.Delete(2);
  ShowRobots(robots);

  //Rendezes
  Writeln('Sorting:');
  robots.Sort(@Compare);
  ShowRobots(robots);

  // Osszes elem torlese
  Writeln('Clearing...');
  robots.Clear;
  ShowRobots(robots);

  // Lista eldobasa
  // Ez felszabaditja a robotok altal lefoglalt helyet is (a robots.OwnsObjects true)
  robots.Free;
  Readln;
end.
```