Ejercicio 1

Parte a)

```
function HayKConsecutivosTipo (arr : ArrIntChar; k : Rango; t : Tipo):boolean;
var
   i, cuantos: integer;
begin
   i:= 1;
   cuantos:= 0;
   while (i + k <= 1 + MAX) and (cuantos < k) do
      begin
         {recorrer secuencia de ts y contar}
         cuantos:= 0;
         while (cuantos < k) and (arr[i].TipoElem = t) do
                begin
                     cuantos:= cuantos + 1;
                     i:= i + 1
                end;
         i := i+1
      end;
   HayKConsecutivosTipo:= cuantos = k;
end;
```

Parte b)

```
procedure ArrToResult (arr : ArrIntChar; var res : Resultado);
var i : integer;
begin
  with res do
    with chars do
    begin
      tope:= 0;
      sumPares:= 0;
      for i:= 1 to MAX do
        case arr[i].tipoElem of
         caracter :
               begin
                    tope:= tope + 1;
                    elems[tope]:= arr[i].chrElem
               end;
          entero :
                if (arr[i].intElem mod 2 = 0) then
                   sumPares:= sumPares + arr[i].intElem
        end {case}
    end
end;
```

Ejercicio 2

Parte a)

```
procedure agregar_al_principio (var l: lista; elem: integer);
var p : lista;
begin
   new(p);
   p^.val:=  elem;
   p^.sig:=  l;
   l:= p;
end;

function ArrToList (arr : TArr) : Lista;
var i : integer;
      l : Lista;
begin
   l:= nil;
   for i:= MAX downto 1 do
      agregar_al_principio(l,arr[i]);
   ArrToList:= l
end;
```

Parte b)

```
procedure ListToArr (l : Lista; var arr : TArr);
var
   i, j : integer;
   p : Lista;
begin
   i:= 1;
   p:= l;

   {copiar lista a arreglo}
   while (p <> nil) and (i<=MAX) do
   begin
      arr[i]:= p^.val;
      i:= i + 1;
      p:= p^.sig
   end;

   {rellenar con 0 los que falten}
   for j:= i to MAX do
      arr[i]:= 0;
end;
```

Ejercicio 3

Entrada	0	1	2	3	4	5	6	7	8	9
Salida	3 1 1	5 2 2	7 3 4	9 4 5	11 5 7	13 6 8	15 7 10	17 8 11	19 9 13	21 10 14