Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten WS2002/3
schriftliche Prüfung im Fach Prozesstechnik
gewählter Vorschlag

Dauer : 180 Minuten **keine externen Hilfsmittel**

Aufgabe :

Nachteilig an statischen Arrays gegenüber dynamischen Arrays ist die Festlegung der Anzahl der Arrayelemente bereits zur Compilierungszeit. Entwickeln Sie daher auf Basis des 8086-Prozessors einen Prototypen einer Toolbox für zweidimensionale dynamische Arrays - das Assembler-Modul *TOOLS.ASM* - durch möglichst bedeutungstreue Übersetzung der Pascal-Unit *TOOLS.PAS*.

Zwecks Vermeidung von Zeitnot während der Klausur sollte von den Routinen *moveRight*, *moveLeft*, *moveDown* und *moveUp* zunächst nur die Routine *moveRight* sowie von den Routinen *addColumn* und *addRow* zunächst nur die Routine *addColumn* bearbeitet werden.

Zur Realisierung der New-Funktion und der Dispose-Prozedur werden die DOS-Funktionen "Allocate Memory" (Funktionscode 48h) und "Free Memory" (Funktionscode 49h) verwendet. Die Funktion zum Reservieren von Speicherplatz erwartet im BX-Register die Anzahl der benötigten zusammenhängenden 16-Byte Blöcke und liefert als Ergebnis im AX-Register die Segmentadresse der reservierten Blöcke (als Offsetadresse innerhalb des Segments wird Null impliziert). Der Fall "nicht genügend zur Verfügung stehender Speicher" bleibt unberücksichtigt. Die Funktion zum Freigeben von Speicherplatz erwartet im ES-Register die Segmentadresse der freizugebenden Blöcke, also eine Adresse die von der Funktion "Allocate Memory" zurückgeliefert wurde.

Vergessen Sie nicht die hinreichende Kommentierung Ihres Assembler-Programms z.B. durch Zuordnung der Pascal-Befehle zu den Assembler-Befehlen.

Zum besseren Verständnis der Klausur ist das Beispiel-Hauptprogramm *One2One* zur Berechnung des "kleinen Ein-Mal-Einses" samt seiner Ausgabe beigefügt.

Das PTL-Team wünscht viel Erfolg

```
Unit Tools;

Interface

Procedure Construct(Var P:Pointer);
Procedure Destroy(P:Pointer);
Procedure addColumn(P:Pointer);
Procedure addRow(P:Pointer);
Procedure setValue(P:Pointer;X,Y:Word;Z:Pointer);
Function getValue(P:Pointer;X,Y:Word):Pointer;

Implementation

Type tpNode = ^tNode;
     tNode  = Record
                Right : tpNode;
                Left  : tpNode;
                Down  : tpNode;
                Up    : tpNode;
                Value : Pointer
              End;
     tpNet  = ^tNet;
     tNet   = Record
                Columns : Word;
                Rows    : Word;
                aColumn : Word;
                aRow    : Word;
                aNode   : tpNode
              End;

Function moveRight(P:tpNode;X:Word):tpNode;
Begin
  While X > 0 Do Begin
    P := P^.Right;
    Dec(X)
  End;
  moveRight := P
End;

Function moveLeft(P:tpNode;X:Word):tpNode;
Begin
  While X > 0 Do Begin
    P := P^.Left;
    Dec(X)
  End;
  moveLeft := P
End;

Function moveDown(P:tpNode;X:Word):tpNode;
Begin
  While X > 0 Do Begin
    P := P^.Down;
    Dec(X)
  End;
  moveDown := P
End;

Function moveUp(P:tpNode;X:Word):tpNode;
Begin
  While X > 0 Do Begin
    P := P^.Up;
    Dec(X)
  End;
  moveUp := P
End;
```

```
Procedure moveXY(P:tpNet;X,Y:Word);
Begin
  If X < P^.aColumn Then P^.aNode := moveLeft(P^.aNode,P^.aColumn-X);
  If X > P^.aColumn Then P^.aNode := moveRight(P^.aNode,X-P^.aColumn);
  If Y < p^.aRow Then P^.aNode := moveUp(P^.aNode,P^.aRow-Y);
  If Y > P^.aRow Then P^.aNode := moveDown(P^.aNode,Y-P^.aRow);
  P^.aColumn := X;
  P^.aRow := Y
End;

Procedure Construct(Var P:Pointer);
Var X : tpNode;
Begin
  P := New(tpNet);
  tpNet(P)^.Columns := 1;
  tpNet(P)^.Rows := 1;
  tpNet(P)^.aColumn := 1;
  tpNet(P)^.aRow := 1;
  X := New(tpNode);
  X^.Right := Nil;
  X^.Left := Nil;
  X^.Down := Nil;
  X^.Up := Nil;
  X^.Value := Nil;
  tpNet(P)^.aNode := X
End;

Procedure Destroy(P:Pointer);
Var I,J,X : tpNode;
Begin
  moveXY(tpNet(P),1,1);
  I := tpNet(P)^.aNode;
  While I <> Nil Do Begin
    J := I;
    I := I^.Down;
    While J <> Nil Do Begin
      X := J;
      J := J^.Right;
      Dispose(X)
    End
  End;
  Dispose(tpNet(P))
End;

Procedure addColumn(P:Pointer);
Var I,X,Y : tpNode;
Begin
  I := tpNet(P)^.aNode;
  While I^.Right <> Nil Do I := I^.Right;
  While I^.Up <> Nil Do I := I^.Up;
  Y := Nil;
  Repeat
    X := New(tpNode);
    X^.Right := Nil;
    X^.Left := I;
    X^.Down := Nil;
    X^.Up := Y;
    X^.Value := Nil;
    If Y <> Nil Then Y^.Down := X;
    I^.Right := X;
    Y := X;
    I := I^.Down
  Until I = Nil;
  Inc(tpnet(P)^.Columns);
  tpNet(P)^.aColumn := tpNet(P)^.Columns;
  tpNet(P)^.aRow := tpNet(P)^.Rows;
  tpNet(P)^.aNode := X
End;
```

```
Procedure addRow(P:Pointer);
Var I,X,Y : tpNode;
Begin
  I := tpNet(P)^.aNode;
  While I^.Left <> Nil Do I := I^.Left;
  While I^.Down <> Nil Do I := I^.Down;
  Y := Nil;
  Repeat
    X := New(tpNode);
    X^.Right := Nil;
    X^.Left := Y;
    X^.Down := Nil;
    X^.Up := I;
    X^.Value := Nil;
    If Y <> Nil Then Y^.Right := X;
    I^.Down := X;
    Y := X;
    I := I^.Right
  Until I = Nil;
  Inc(tpnet(P)^.Rows);
  tpNet(P)^.aColumn := tpNet(P)^.Columns;
  tpNet(P)^.aRow := tpNet(P)^.Rows;
  tpNet(P)^.aNode := X
End;

Procedure setValue(P:Pointer;X,Y:Word;Z:Pointer);
Begin
  moveXY(tpNet(P),X,Y);
  tpNet(P)^.aNode^.Value := Z
End;

Function getValue(P:Pointer;X,Y:Word):Pointer;
Begin
  moveXY(tpNet(P),X,Y);
  getValue := tpNet(P)^.aNode^.Value
End;

End.
```

```
Program One2One;

Uses Tools;

Type tpInteger = ^Integer;

Var Net : Pointer;
    X   : tpInteger;
    I,J : Integer;

Begin
  Construct(Net);
  For I := 2 To 10 Do Begin
    addRow(Net);
    addColumn(Net)
  End;
  For I := 1 To 10 Do
    For J := 1 To 10 Do Begin
      X := New(tpInteger);
      X^ := I*J;
      setValue(Net,I,J,X)
    End;
  For I := 1 To 10 Do Begin
    For J := 1 To 10 Do
      Write(tpInteger(getValue(Net,I,J))^:4);
    WriteLn
  End;
  For I := 1 To 10 Do
    For J := 1 To 10 Do
      Dispose(tpInteger(getValue(Net,I,J)));
  Destroy(Net)
End.
```

```
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100
```