Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten WS2004/5
schriftliche Prüfung im Fach Prozesstechnik
gewählter Vorschlag

Dauer : 180 Minuten **keine externen Hilfsmittel**

Aufgabe :

Im Technisch-Wissenschaftlichen Bereich werden nach wie vor gerne Taschenrechner nach der Umgekehrt Polnischen Notation (UPN) eingesetzt. Entwickeln Sie daher auf Basis des 8086-Prozessors einen simplen Prototypen zur Simulation des Rechnens in Umgekehrt Polnischer Notation (UPN) durch möglichst bedeutungstreue Übersetzung des Pascal-Programms *UPN*.

Zur Realisierung der New-Prozedur 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.

Bonusfrage : Welche Werte werden durch die insgesamt vier Writeln-Anweisungen im Programm *UPN* ausgegeben ?

Das PTL-Team wünscht viel Erfolg

```
{$X+}
Program UPN;

Type StackType        = ^StackContentType;
     StackContentType = Record
                          Value    : Integer;
                          Previous : StackType
                        End;

Var Stack : StackType;

Function Top(S:StackType):Boolean;

Begin
  Top := S <> Nil
End;

Function Count(S:StackType):Word;

Begin
  If Top(S) Then
    Count := Count(S^.Previous)+1
  Else
    Count := 0
End;

Procedure Push(Var S:StackType;V:Integer);

Var SV : StackType;

Begin
  New(SV);
  SV^.Value := V;
  SV^.Previous := S;
  S := SV
End;

Function Pop(Var S:StackType):Integer;

Var P : StackType;

Begin
  If Top(S) Then Begin
    Pop := S^.Value;
    P := S;
    S := S^.Previous;
    Dispose(P)
  End Else
    Pop := 0
End;

Function Add(Var S:StackType):Boolean;

Begin
  Push(S,Pop(S)+Pop(S));
  Add := True
End;

Function Subtract(Var S:StackType):Boolean;

var Subtrahend : Integer;

Begin
  Subtrahend := Pop(S);
  Push(S,Pop(S)-Subtrahend);
  Subtract := True
End;
```

```
Function Multiply(Var S:StackType):Boolean;

Begin
  Push(S,Pop(S)*Pop(S));
  Multiply := True
End;

Function Divide(Var S:StackType):Boolean;

Var Dividend,Divisor : Integer;

Begin
  Divisor := Pop(S);
  Dividend := Pop(S);
  If Divisor = 0 Then Begin
    Push(S,0);
    Divide := False
  End Else Begin
    Push(S,Dividend Div Divisor);
    Divide := True
  End
End;

Function Modulo(Var S:StackType):Boolean;

Var Dividend,Divisor : Integer;

Begin
  Divisor := Pop(S);
  Dividend := Pop(S);
  If Divisor = 0 Then Begin
    Push(S,0);
    Modulo := False
  End Else Begin
    Push(S,Dividend Mod Divisor);
    Modulo := True
  End
End;

Begin
  Stack := Nil;
  Push(Stack,24);
  Push(Stack,42);
  Push(Stack,30);
  Push(Stack,40);
  Writeln(Count(Stack));
  Multiply(Stack);
  Push(Stack,50);
  Divide(Stack);
  Push(Stack,60);
  Writeln(Count(Stack));
  Modulo(Stack);
  Subtract(Stack);
  Add(Stack);
  Writeln(Count(Stack));
  Writeln(Pop(Stack))
End.
```