Coletânea de exercícios

MIEIC, MPCP

António Araújo, João Canas Ferreira, João Paulo Sousa

{aja|jcf|jpsousa@fe.up.pt}

Nesta coletânea de exercícios encontram-se problemas de análise e de síntese de código. Todos foram retirados de exames e pontualmente adaptados para melhor se poderem testar no ambiente de programação *visual studio* da Microsoft. As resoluções estão reunidas no fim do documento.

FEUP, Maio 2013

P1. **[2006-01-23]** A rotina **rotx** destina-se a processar uma matriz quadrada, de bytes, de tamanho Nd×Nd. Analise e descreva a rotina, referindo as instruções mais importantes. Indique a finalidade da rotina.

```
 1 rotx proc uses edi ebx apt: ptr byte,
 2                         nd: dword
 3     mov  edi, apt
 4     mov  ecx, nd
 5     mov  eax, ecx
 6     mul  ecx
 7     mov  ecx, eax
 8     mov  al, 0
 9     cld
10     rep  stosb
11     mov  ecx, nd
12     mov  ebx, ecx
13     inc  ebx
14     mov  edi, apt
15 @@: mov  byte ptr [edi], 1
16     add  edi, ebx
17     loop @b
18     ret
19 rotx endp
20
```

P2. **[2010-07-01, adaptado]** Explique o funcionamento e indique a finalidade do fragmento de programa apresentado a seguir. Diga qual o valor contido na variável **res** no fim da execução do programa.

```
 1 include mpcp.inc
 2
 3 .data
 4 buf byte 0FFh, 0FFh, 0EFh, 0FFh
 5 n   byte lengthof buf
 6 mascara byte 1 dup(0)
 7 res word 1 dup(0)
 8
 9 .code
10 main:
11      mov   edi, offset buf
12      mov   res, 0
13      movzx ecx, n
14      jecxz fim
15
16 salto2:
17      mov   mascara, 80h
18      mov   bl, 8
19
20 salto1:
21      mov   ah, [edi]
22      and   ah, mascara
23      jz    fim
24      shr   mascara, 1
25      inc   res
26      dec   bl
27      jnz   salto1
28      inc   edi
29      loop  salto2
30
31 fim: invoke ExitProcess, 0
33
34 end main
```

P3. **[2007-01-03]** Descreva e explique o funcionamento da rotina **rotx**, justificando convenientemente a resposta e identificando o objetivo da rotina.

```
 1 rotx proc uses edi ap: ptr word,
 2                   cnt: dword,
 3                     x: byte, y: byte
 4     mov   edi, ap
 5     mov   ecx, cnt
 6     cld
 7     mov   ah, x
 8     mov   al, y
 9 @@: repnz scasb
10     jecxz @f
11     mov   [edi-1], ah
12     jmp   @b
13 @@: jnz   @f
14     mov   [edi-1], ah
15 @@: ret
16 rotx endp
```

P4. **[2006-06-02]** Analise e descreva a rotina **rotx**, apresentada a seguir, referindo as instruções mais importantes. Indique também a finalidade da rotina. Nesta rotina, **ap1** (**ap2**) é o endereço da primeira posição ocupada por uma sequência de bytes de comprimento **n1** (**n2**). Assuma que n1 é sempre maior do que n2.

```
 1 rotx proc uses edi esi ap1: ptr byte,
 2                        ap2: ptr byte,
 3                         n1: dword, n2:dword
 4     mov   edi, ap1
 5     ecx, n1
 6      sub   ecx, n2
 7     inc   ecx
 8     xor   eax, eax
 9     cld
10 @@: push  edi
11     push  ecx
12     mov   esi, ap2
13     mov   ecx, n2
14     repz  cmpsb
15     jecxz @f
16 s1: pop   ecx
17     pop   edi
18     inc   edi
19     loop  @b
20     ret
21 @@: jnz   s1
22     inc   eax
23     jmp   s1
24 rotx endp
```

P5. **[2010-07-21]** Analise o funcionamento do programa em linguagem *assembly* IA-32 abaixo descrito. a) Explique como funciona e qual a finalidade da rotina **rotinaY**. b) Que valor é apresentado no monitor no final da execução do programa? c) Apresente o conteúdo da pilha imediatamente antes da execução da instrução assinalada com (**).

```
 1 include mpcp.inc
 2 rotinaY proto x: word, y: word
 3
 4 .data
 5 item1 word 0100100100010001B
 6 item2 word 1110100001010101B
 7
 8 msg SBYTE "%d",13,10,0
 9
10
11 .code
12 main:
13   invoke rotinaY, item1, item2
14   invoke printf, offset msg, eax
15   invoke _getch
16   invoke ExitProcess, 0
17
18 rotinaY proc x: word, y: word
19   xor eax, eax
20   xor ecx, ecx    ; (**)
21   mov cx, x
22   xor cx, y
23 ciclo:
24   jcxz fim
25   shr cx,1
26   jnc ciclo
27   inc eax
28   jmp ciclo
29 fim: ret
30 rotinaY endp
31
32 end main
```

P6. **[2007-01-03]** Descreva a evolução da pilha durante a execução do seguinte fragmento de código. Descreva o estado da pilha antes da execução das instruções assinaladas com 1, 2 e 3.

```
 1 include mpcp.inc
 2 rot1 proto arg1: DWORD, arg2: DWORD
 3 rot2 proto val: DWORD
 4
 5 .code
 6 main:
 7    invoke rot1, 6, 51
 8    invoke ExitProcess, 0
 9
10 rot2 PROC val: DWORD
11    mov ecx, val        ; (1)
12    ;...
13    ret
14 rot2 ENDP
15
16 rot1 PROC USES EDI arg1: DWORD, arg2: DWORD
17    mov eax, arg2       ; (2)
18    mov ebx, arg1
19    ;...
20    push eax
21    ;...
22    invoke rot2, 11
23    ;...
24    pop eax             ; (3)
25    ret
26 rot1 ENDP
27
28 end main
```

P7. **[2006-01-23]** Considere as rotinas sub1 e sub2 apresentadas a seguir. Indique o estado da pilha imediatamente após a execução das instruções assinaladas com 1 e 2.

```
 1 include mpcp.inc
 2 sub1 proto v: dword
 3 sub2 proto x: ptr byte, t: byte
 4
 5 .data
 6 buf dw 200 dup(0)
 7 num db lengthof buf
 8
 9 .code
10 main:
11   invoke sub2, offset buf, num
12   invoke ExitProcess, 0
13
14 sub1 proc v:dword
15     ;..
16     add eax, ecx    ; (2)
17     ;...
18     ret
19 sub1 endp
20
21 sub2 proc x:ptr byte, t:byte
22 local temp:dword
23     ;...
24     xor eax, eax    ; (1)
25     push eax
26     ;...
27     invoke sub1,eax
28     ;...
29     pop eax
30     ;...
31     ret
32 sub2 endp
33
34 end main
```

P8. **[2006-06-02]** Considere o fragmento de um programa em *assembly* IA-32 apresentado a seguir que usa a convenção de invocação de sub-rotinas *stdcall*. Apresente o código gerado pelo assemblador: a) para implementar a diretiva invoke; b) para o prólogo e o epílogo da rotina.

```
1 include mpcp.inc
2 rotina proto x:dword, t:dword
3
4 .code
5 main:
6   invoke rotina, eax, ebx
7   invoke ExitProcess, 0
```

```
 8
 9 rotina proc uses edi a:dword, b:dword
10     ;...
11     ret
12 rotina endp
13
14 end main
```

P9. [2006-06-02] Considere uma sequência de N números inteiros sem sinal. Escreva uma rotina para determinar qual a subsequência de M elementos, cuja soma é máxima. A função deve retornar o índice da posição inicial da subsequência encontrada. Se existir mais que uma subsequência nas condições indicadas, a função deve retornar a posição da primeira. Assuma que N >= M. O protótipo da rotina deve ser:

```
MAIOR_SUBSEQ PROTO vect:PTR DWORD, N:DWORD, M:DWORD
```

P10. [2006-01-23] Uma sequência de N *bytes* contém apenas dois valores diferentes, 00_H e FF_H. Pretende-se comprimir o seu conteúdo de acordo com o seguinte esquema: Cada sub-sequência ininterrupta de valores iguais é representada por um *byte*, cujo *bit* mais significativo indica o tipo de sequência, e os restantes 7 *bits* representam o número de elementos da sequência (valor máximo 127). O tipo é **0** para sequências de zeros, e **1** para sequências de FF_H. Exemplo (representado em hexadecimal):

Sequência inicial: 00 00 00 00 FF FF 00 00 00 00 00 00 00 FF FF FF FF FF FF	Comprimento: 19
Sequência comprimida: 04 82 07 86	Comprimento: 4

Escreva a rotina COMPRIMIR, que aceita uma sequência VECT de *bytes* e produz a correspondente versão comprimida em VECT_RES. A rotina retorna o número de bytes da sequência comprimida. O protótipo da rotina deve ser:

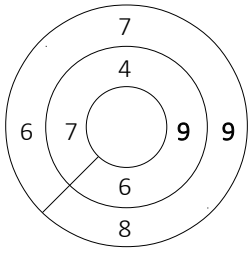
```
comprimir proto vect:ptr byte, n:dword, vect_res:ptr byte
```

Assuma que a sequência original não contem sub-sequências de mais de 127 elementos iguais seguidos.

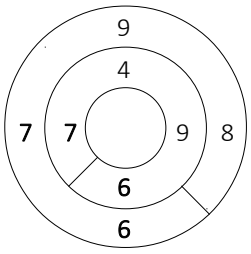
P11. [2007-02-02] Considere duas sequências de números inteiros de comprimento N, dispostas em anel conforme indicado na figura. Escreva a rotina N_SEQ que retorna em EAX o maior número de coincidências de valores que se podem obter rodando o anel exterior. A função deve ter o seguinte protótipo:

```
RodarAneis PROTO anelInterno:PTR DWORD, anelExterno:PTR DWORD, N:DWORD
```

Exemplo: sequências 6,7,9,8 e 7,4,9,6:



Uma coincidência



Duas coincidências
(anel rodado uma vez)

P12. [2010-07-21] Segundo a Regra de Simpson para integração numérica, o integral definido de uma função num intervalo [a, b] pode ser calculado aproximadamente pela seguinte equação:

$$\int_a^b f(x)\,dx \;\approx\; (b-a)\,\frac{f(a)+4\,f(\frac{a+b}{2})+f(b)}{6}$$

Escreva uma rotina em *assembly*, com instruções da unidade de vírgula flutuante, que aproxima o valor do integral definido da função **cos(x²)** (argumento em radianos) num intervalo [a, b], por aplicação da equação indicada. O protótipo da rotina é:

```
SIMPSON PROTO a:real4, b:real4
```

P13. [2007-01-03] A rotina **TabDist** calcula **N** valores da função *F(α)*, quando *α* varia uniformemente entre 0 e π/2, usando a unidade de vírgula flutuante.

$$F(\alpha)=\frac{2V^2\cos(\alpha)\,sen(\alpha)}{9.8}$$

A rotina tem o seguinte protótipo:

```
TabDist PROTO tabela:PTR REAL4, V:REAL4, N:DWORD
```

O parâmetro **tabela** especifica o endereço do 1º elemento da tabela.

P14. [2010-07-01] A função *C(A,d)* estima o número de circuitos integrados de área A que cabem numa bolacha de silício de diâmetro d.

$$C(A,d)=\frac{\pi\times(d/2)^2}{A}-\frac{\pi\times d}{\sqrt{2\times A}}$$

a) Escreva a rotina CalcNCircuitos que calcula o o número de circuitos integrados, retornando o valor inteiro (DWORD) obtido por arredondamento de *C(A,d)*. Utilize o seguinte protótipo:

```
CalcNCircuitos PROTO A:PTR REAL4, d:PTR REAL4
```

b) Usando a função anterior, escreva uma rotina que preenche uma sequência com o número de circuitos integrados produzidos para A=1.0, 2.0, 3.0,…,100.0 e d=300.0 (fixo). Utilize o seguinte protótipo:

```
CalcTab PROTO seq:PTR REAL4
```

O parâmetro **seq** especifica a sequência (de 100 elementos) que deve ser preenchida com os valores calculados.

P15. [2006-01-23] Pretende-se calcular o centro de massa de N pontos colocados sobre uma recta, com intervalos iguais entre si (de comprimento L), e em que m_i é a massa do *i*-ésimo ponto. A posição do CM é dada por:

$$L\times\sum(i\times m_i)/\sum m_i\ .$$

Escreva uma rotina, usando instruções de vírgula flutuante, que retorna o centro de massa de N pontos, cujas massas estão especificadas no vector VECTM. O protótipo da rotina deve ser:

```
cmassa proto vectm:ptr real8, n:dword, l:real8
```

P16. [2006-06-02] Implemente as sub-rotinas seguintes em linguagem *assembly* IA-32 com recurso à unidade de vírgula flutuante.

a) Rotina MEDIA, que calcula a média de um conjunto de *n* amostras representadas em vírgula flutuante, precisão dupla. Assuma N>=1. O protótipo da rotina deve ser:

```
MEDIA PROTO amostras:PTR REAL8, N:DWORD
```

b) Rotina DPADRAO, que calcula o desvio padrão de um conjunto de *n* amostras representadas em vírgula flutuante, precisão dupla. Assuma N>1.

O desvio padrão de *n* amostras de média $\bar{x}$ vem dado por: $$s=\sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}}$$

O protótipo da rotina deve ser:

```
DPADRAO PROTO amostras:PTR REAL8, N:DWORD, media:REAL8
```

Soluções

P1.

Numa primeira fase a rotina começa por calcular o número de elementos da matriz (linhas 4 a 7) e de seguida coloca-os todos a zero (linha 10). Numa segunda fase (a partir da linha 11) coloca a 1 todos os elementos da diagonal da matriz (linhas 15 a 17) pois em cada iteração do ciclo avança o apontador Nd+1 posições (linhas 11 a 13 e linha 16).

P2.

Uma máscara, iniciada em 10000000b na linha 17, permite verificar (linhas 21-23) se algum dos bits de um elemento do vetor buf é zero. Se isso não acontecer em nenhum dos 8 bits o programa passa a analisar o elemento seguinte (linhas 28 e 29). Os bits de cada elemento são analisados começando pelo MSB e acabando no LSB pois o bit a um da máscara vai deslizando da esquerda para a direita (linha 24). O programa termina assim que é detetado um bit a zero. Na variável res, incrementada sempre que o bit analisado é um, fica o número de bits a um encontrados. No exemplo apresentado ela fica como valor final 19.

P3.

A rotina rotx substitui todas as ocorrências do byte y pelo byte x na cadeia de words apontada por ap e de comprimento cnt. A linha 10 garante que a rotina termina quando a cadeia é totalmente pesquisada (ao fim de cnt iterações) a linha 9 deteta uma ocorrência de y e a linha 11 faz a substituição propriamente dita compensando o facto ([edi-1]) de EDI ter sido automaticamente incrementado pela instrução scasb. A linha 15 faz a substituição no caso de a última ocorrência de y ser na última posição da cadeia.

P4.

A rotina conta em EAX quantas vezes a sequência 2 (seq2) aparece na sequência 1 (seq1). As linhas 12-15, pesquisam uma ocorrência completa de seq2 em seq1 - se chegamos ao fim de seq2 (teste na linha 15 verdadeiro) e a última comparação efetuada na linha 14 for entre elementos iguais (teste na linha 21 falso) então estamos em presença de uma ocorrência completa de seq2 em seq1 e o registo EAX deve ser incrementado (linhas 21-23), caso contrário é necessário avançar uma posição no ponto inicial de pesquisa em seq1 (linha 18). Alguns pormenores relevantes: a) seq1 deve ser

pesquisada desde o inicio até n1-n2+1 posições do fim (linhas 5-7) pois pode dar-se o caso de as últimas n2 posições coincidirem com uma ocorrência completa de seq2; b) em cada pesquisa de eventual ocorrência de seq2 em seq1 deve ser preservado o valor inicial de EDI e ECX pois são necessários no ciclo de pesquisa principal (linhas 10-19) – para isso estão as linhas 10-11 e 16-17.

P5.

a) O ou exclusivo entre os dois operandos da rotina (linhas 21-22) deixa ativos os bits que são diferentes entre eles. Esses bits (a um) são contados no ciclo implementado nas linhas 24-28. O resultado da contagem é retornado em EAX. A finalidade da rotina é pois contar os bits diferentes entre os dois parâmetros. Neste exemplo será apresentado no monitor o valor 5 (há 5 bits diferentes entre item1 e item2).

b) Antes da execução da linha 20 no topo da pilha está o conteúdo de EBP, colocado lá pelo prólogo de rotinaY; seguem-se (em endereços crescentes) o endereço de retorno e os parâmetros item1 e item2, colocados lá pela diretiva invoke da linha 13.

P6.

Até à linha 11 (ponto 1) a ordem cronológica de entrada na pilha foi a seguinte: arg2, arg1, end ret rot1, EBP, EDI, EAX, val1, end ret rot2, EBP, sucessivamente em endereços decrescentes. Os três primeiros valores (arg2, arg1, end ret rot1) foram colocados na pilha pela diretiva invoke da linha 7, os dois seguintes (EBP, EDI) colocados na pilha pelo prólogo de rot1, o seguinte (EAX) colocado na pilha explicitamente pela rotina rot1, os dois seguintes (val1, end ret rot2) colocados na pilha pela diretiva invoke da linha 22 e finalmente o último (EBP) colocado na pilha pelo prólogo de rot2. Assim, na linha 17 (ponto 2) está na pilha tudo até ao efetuado pelo prólogo de rot1, isto é, tudo até ao EDI inclusive; na linha 24 (ponto 3) já foi retirado tudo o que diz respeito a rot2 ficando ainda tudo até ao EAX inclusive.

P7.

A ordem cronológica de entrada na pilha até ao ponto 2 (linha 16) é a seguinte: num, offset buf, end ret sub1, EBP, temp, EAX, EAX, end ret sub2, EBP, sucessivamente em endereços decrescentes. Os três primeiros valores (num, offset buf, end ret sub1) foram colocados na pilha pela diretiva invoke da linha 11; os dois seguintes (EBP, temp) foram colocados na pilha pelo prólogo de sub1, sendo que temp não foi propriamente lá colocado mas apenas reservado espaço (ESP decrementado 4 unidades); o seguinte (EAX) foi colocado na pilha explicitamente pela rotina sub1 na linha 25; os dois seguintes (EAX, end ret sub2) foram colocados na pilha pela diretiva invoke da linha 27 e finalmente o último (EBP) foi colocado na pilha pelo prólogo de sub2. No ponto 1 (linha 24) está na pilha tudo até ao efetuado pelo prólogo de sub1, isto é, tudo até temp inclusive.

P8.

a) Diretiva invoke:

```
push ebx
push eax
call rotina
```

b) Prólogo e epílogo da rotina:

```
; Prólogo (código acrescentado antes da primeira instrução da rotina)
push ebp
mov ebp,esp
push edi

; Epílogo (código que substitui a instrução ret)
pop edi
leave
ret 8
```

P9. Maior sub-sequência

```
;-----------------------------------------
; Prob9
; Maior sub-sequência
; -----------------------------------------
maior_subseq proc uses edi esi vect: ptr dword,
                           n: dword, m: dword
local max_val: dword
    mov ecx, n
    sub ecx, m
    inc ecx
    xor eax, eax   ; guarda índice
    mov  max_val, eax
    jecxz fim
    mov  edi, vect

ciclo2:
    mov  esi, edi
    push ecx
    mov  ecx, m
  xor  edx, edx  ; edx contém max

ciclo1:
  add  edx, [esi]
  add  esi, 4
  loop ciclo1

  cmp  edx, max_val
  jbe  continua
  mov  max_val, edx
  mov  eax, edi
  sub  eax, vect
  shr  eax, 2
continua:
  pop  ecx
  add  edi, 4
  loop ciclo2
fim: ret
maior_subseq endp
```

P10. Comprimir

```
;-----------------------------------------
; Prob10
; Comprimir
;-----------------------------------------
comprimir proc uses esi edi vec1: ptr byte,
                  n: dword, vec2: ptr byte
     mov esi, vec1
     mov edi, vec2
     mov ecx, n
     mov al, [esi]
     mov ah, 1         ; contador
     inc esi
     xor edx, edx
     dec ecx

ciclo:
     cmp ecx, 0
     jz escrever       ; fim dos dados
     cmp al, [esi]
     jnz escrever
     inc ah
     jmp proximo
escrever:
     cmp al, 0
     jz zer
     or ah, 080h
zer: mov [edi], ah
     inc edx
     inc edi
     and ecx, ecx    ; ainda há trabalho?
     jz  fim
new: mov al, [esi]
     mov ah, 1

proximo:
     inc    esi
     dec    ecx
     jmp    @b

fim: mov    eax, edx
     ret

comprimir endp
```

P11. Aneis

```
; ------------------------------------------
; Prob11 (aneis)
;  Estratégia:
;  - rodar anelEXterno e contar coincidências
;  - repetir para todas as rotações possíveis
; ------------------------------------------
; JCF, Abril 2013
; ------------------------------------------

;; Funções auxiliares
;; nº de coincidências
NCoinc PROTO C Interno: PTR DWORD,
               Externo: PTR DWORD, N: DWORD

;; rodar uma sequência de N valores
RodarSeq PROTO C seq:PTR DWORD, N: DWORD


RodarAneis PROC USES edi ebx Int: PTR DWORD,
                             Ext: PTR DWORD,
                               N:DWORD

    xor edi, edi  ; contador (EDI não é
                  ; alterado pelas invocações)
    mov ebx, N    ; nº de rotações que ainda
                  ; falta efetuar (garantir
                  ; que sequência anelExterno
                  ; não fica alterada)
    .while (ebx > 0)
        invoke  RodarSeq, Ext, N
        invoke  NCoinc, Int, Ext, N
        .if eax > edi      ; novo máximo?
            mov edi, eax
        .endif
        dec ebx
    .endw
    ;; A sequência anelExterno está na
    ;; situação inicial
    ;; Preparar valor de retorno
    mov eax, edi
    ret
RodarAneis ENDP
```

```
;; Funções auxiliares
;;
;; Calcular nº de elementos iguais
;;   em posições correspondentes

NCoinc PROC USES esi edi Int:PTR DWORD,
                         Ext:PTR DWORD,
                           N: DWORD

    mov esi, Ext
    mov edi, Int
    xor eax, eax
    mov ecx, N
@@: mov edx, [esi]
    .if edx == [edi]
        inc eax
    .endif
    add edi, 4
    add esi, 4
    loop @B
    ret
NCoinc ENDP


;; Rodar uma sequência para a esquerda

RodarSeq PROC USES esi edi seq: PTR DWORD,
                            N: DWORD
    mov esi, seq
    push dword ptr [esi]  ; guardar primeiro
    mov ecx, N            ; valor da sequência
    dec ecx
    mov edi, esi
    add esi, 4
@@: lodsd
    stosd
    loop @B
    pop dword ptr [edi]   ; colocar el. inicial
    ret                   ; no fim da sequência
RodarSeq ENDP
```

P12. Regra de Simpson

```
; ------------------------------------------
; Prob12
; Integração numérica (simpson)
; ------------------------------------------
; AJA, Abril 2013
; ------------------------------------------
include mpcp.inc
simpson proto a:real4, b:real4

.data
  msg1 byte "Introduza o valor de 'a' ",
            "(limite inferior): ", 0
  msg2 byte "Introduza o valor de 'b' ",
            "(limite superior): ", 0
  format byte "%f", 0
  msg3 byte "Area = %lf", 0

  val_a    real4  ?
  val_b    real4  ?
  area     real8  ?

  const2   real4  2.0
  const4   real4  4.0
  const6   real4  6.0

.code
main:
  invoke printf, offset msg1
  invoke scanf,  offset format, offset val_a
  invoke printf, offset msg2
  invoke scanf,  offset format, offset val_b

  invoke simpson, val_a, val_b
  invoke printf, offset msg3, area
  invoke _getch
  invoke ExitProcess, 0

;; Rotina pedida,
;; usando instruções de vírgula flutuante

simpson proc a:real4, b:real4
  finit
  fld  b
  fsub a
  fdiv const6 ; ST(0) <== (b-a)/6
  fld  a
  fmul st, st
  fcos        ; ST(0) <== f(a)
  fld  b
  fmul st, st
  fcos        ; ST(0) <== f(b)
  fadd        ; ST(0) <== f(a) + f(b)
  fld  a
  fld  b
  fadd
  fdiv const2
  fmul st, st
  fcos
  fmul const4 ; ST(0) <== 4f((a+b)/2)
  fadd        ; ST(0) <== f(a) + f(b) +
              ;                + 4f((a+b)/2)
  fmul        ; ST(0) <== area
  fstp area
  ret
simpson endp

end main
```

P13. Função trigonométrica

```
; ------------------------------------------
; Prob13
; Função trigonométrica (TabDist)
; ------------------------------------------
; AJA, Abril 2013
; ------------------------------------------

include mpcp.inc
TabDist proto tabela: ptr real8, V: real4,
                                 N: dword

.data
  msg0 byte "Introduza o valor de ", 0
  msg1 byte "V: ", 0
  msg2 byte "N (N<100): ", 0
  format_f byte   "%f", 0
  format_d byte   "%d", 0
  msg3 byte "Valor = %f", 13, 10, 0

  const2   real4  2.0
  constG   real4  9.8
  tab      real8  100 dup(?)
  alfa     real4  0.0
  delta    real4  ?
  VV       real4  ?
  NN       dword  ?

.code
main:
    invoke printf, offset msg0
    invoke printf, offset msg1
    invoke scanf,  offset format_f, offset VV
    invoke printf, offset msg0
    invoke printf, offset msg2
    invoke scanf,  offset format_d, offset NN

    invoke TabDist, offset tab, VV, NN

    ; imprime tabela
    mov ecx, NN
    mov esi, offset tab
@@: push ecx
    invoke printf, offset msg_out,
                   real8 ptr [esi]
    add esi, type real8
    pop ecx
    loop  @B

    invoke _getch
    invoke ExitProcess, 0


; Rotina pedida,
; usando instruções de vírgula flutuante

TabDist proc tabela:ptr real8, V:real4, N:dword
    mov  edi, tabela
    finit
    fldpi
    fld  const2
    fdiv
    fild N
    fld1
    fsub
    fdiv
    fstp delta   ; delta <== (pi/2-0)/(N-1)

    mov  ecx, N
@@: fld  alfa
    fsincos
    fmul          ; cos(x)*sin(x)
    fmul const2
    fld  V
    fmul st, st
    fmul
    fdiv constG
    fstp real8 ptr [edi]  ; guarda F(alfa)

    add  edi, 8
    fld  alfa
    fadd delta
    fstp alfa     ; alfa <== alfa + delta
    loop @B

    ret
TabDist endp

end main
```

P14. Circuitos

```
; ------------------------------------------
; Prob14
; Area de circuitos (CalcNCircuitos)
; ------------------------------------------
; JCF, Abril 2013
; ------------------------------------------

;;;  Alínea a)

CalcNCircuitos PROC A: REAL4, d: REAL4
LOCAL res:SDWORD   ; para conversão VF->DWORD
                   ; (ver fim da função)
    ;; 1º termo
    fld d
    fld1
    fld1
    fadd      ; constante 2 no topo da pilha
    fdiv      ; d/2 no topo da pilha

    fld st(0) ; replicar valor do topo da pilha
    fmul      ; calcular o quadrado: (d/2)^2
    fldpi
    fmul      ; numerador do 1º termo completo
    fdiv A    ; 1º termo completo

    ;; 2º termo
    fldpi     ; calcular numerador
    fmul d
    fld A
    fadd A    ; 2A no topo da pilha UVF
    fsqrt     ; denominador calculado (único
              ; valor na pilha UVF)

    fdiv      ; 2º termo completo
    fsub      ; calcular 1º termo - 2º termo
    fistp res ; converte para inteiro
              ; (arredondamento) e guarda em
              ; memória
    mov eax, res
    ret
CalcNCircuitos ENDP
```

```
;;; Alínea b)
;;; Função para preencher a tabela

CalcTab PROC  USES edi ebx vect:PTR SDWORD
LOCAL diam:REAL4   ; vai conter 300.0
LOCAL tmp: SDWORD
LOCAL area:REAL4   ; vai variar: 1.0, 2.0,...
                   ; ..., 300.0

    ;; converter valor 300 para VF
    ;; sem usar variáveis globais
    mov eax, 300
    mov tmp, eax   ; guardar em memória
    fild tmp       ; pois a UVF só acede
    fstp diam      ;  a memória. Guardar

    ;; valor inicial da área (1 em VF)
    fld1
    fstp area

    ;; ciclo para preencher tabela
    ;; usa EBX e EDI porque não são alterados
    ;; pelas sub-rotinas
    mov edi, vect
    mov ebx, 300
    .while (ebx > 0)
        invoke CalcNCircuitos, area, diam
        mov SDWORD PTR [edi], eax
        add  edi, 4
        ;; atualizar area (inútil na ultima
        ;; iteração)
        fld1
        fadd area
        fstp area
        dec ebx
    .endw
    ret
CalcTab ENDP
```

P15. Centro de massa

```
; ------------------------------------------
; Prob15
; Centro de massa
; ------------------------------------------

cmassa proc uses esi vectm: ptr real8,
                        n: dword, l: real8
local contador: dword
    mov esi, vectm
    mov ecx, n
    mov contador, 1
    fldz
    fldz
```

```
@@: fxch
    fadd real8 ptr [esi]
    fxch
    fild contador
    fmul real8 ptr [esi]
    fadd
    add esi, 8
    inc contador
    loop @b
    fdivr
    fmul l
    ret
cmassa endp
```

P16. Média e desvio padrão

```
; ------------------------------------------
; Prob15
; Média e desvio padrão
; ------------------------------------------

media proc uses edi amostras: ptr real8,
                           n: dword
    mov ecx, n
    mov edi, amostras
    fldz
@@: fadd real8 ptr [edi]
    add  edi, 8
    loop @b
    fild n
    fdiv
    ret
media endp

dpadrao proc amostras: ptr real8, n: dword,
                                  media: real8
    mov  ecx, n
    mov  edi, amostras
    fldz
@@: fld  real8 ptr [edi]
    fsub media
    fld  st(0)
    fmul
    fadd
    add  edi, 8
    loop @b

    fild n
    fld1
    fsub
    fdiv
    fsqrt
    ret
dpadrao end
```