Gestão da pilha e sub-rotinas

1. Analise e descreva o algoritmo apresentado, referindo as instruções mais importantes. Indique também a finalidade da sub-rotina.

```
Rotina  PROC Valor: BYTE
        MOV   ECX, 3
        MOVZX EDX, Valor
        MOV   EAX, EDX
@@:     SHL   EAX, 8
        OR    EAX, EDX
        LOOP  @B
        RET
Rotina  ENDP
```

2. Considere o seguinte fragmento de um programa em *assembly*.

```
ROTINA  PROC uses EDI A:DWORD, B:DWORD
        . . .
        RET
ROTINA  ENDP
INI:    . . .
        invoke ROTINA, EAX, EBX
```

a) Apresente o código gerado pelo *assembler* para implementar a invocação da sub-rotina através da diretiva `invoke`.

b) Apresente o código gerado pelo *assembler* para o prólogo e para o epílogo da sub-rotina.

c) Repita a alínea anterior considerando agora a convenção de invocação de sub-rotinas C.

3. Considerar uma sequência de números inteiros com 16 bits (com sinal). Escrever as sub-rotinas seguintes, as quais devem retornar no registo EAX o valor pedido.

a) `MAX`: determina o valor máximo da sequência;

b) `MIN`: determina o valor mínimo da sequência;

c) `MEDIA`: calcula a média dos valores da sequência.

4. Escreva a sub-rotina `minPot2` que calcula a menor potência de 2 maior que `val` (um parâmetro do tipo DWORD), retornando o resultado em EAX.

5. Escreva a sub-rotina `poly2` que calcula o valor do polinómio de segundo grau em x (uma variável inteira) dado por $c_2 \times x^2 + c_1 \times x + c_0$.

A sub-rotina tem o protótipo `poly2 PROTO coefs:PTR SWORD, x:SWORD`.

A sequência de coeficientes é apontada por `coefs`, sendo c_2 o primeiro elemento.

6. Escrever e testar a sub-rotina `ORDENA` que ordena uma sequência de elementos do tipo DWORD por ordem crescente. Pode assumir que a sequência tem pelo menos dois elementos. Indicar também como alterar a sub-rotina para ordenar por ordem decrescente.

7. Considere o seguinte programa:

```
include mpcp.inc
  .data
outmsg byte "%d",13,10,0
  .code
start:
  call rotina  ; + *
  invoke _getch
  invoke ExitProcess,0

rotina proc
  push 21    ; *
  call double
  add  eax, 5
  invoke printf,offset outmsg,eax
  ret        ; +
rotina endp
```

```
double proc
  mov eax, [esp+4]  ; +
  add eax, eax
  ret 4             ; + *
double endp

end start
```

a) Analise o programa e descreva a sua função.

b) Indique o conteúdo da pilha do sistema após a execução das instruções marcadas com '`*`' e antes das instruções marcadas com '`+`'. Confirme as respostas usando a vista `Assembly` do Visual Studio.

c) Indique o que aconteceria se a instrução `ret 4` fosse substituída por `ret`.

8. Considere as sub-rotinas apresentadas a seguir, em que `buf` é uma sequência de `num` elementos do tipo BYTE. Admitindo que `ESP=20040000H` antes da execução da chamada da sub-rotina `SUBR` (linha indicada com '`*`'), indique o estado da pilha imediatamente após a execução das instruções marcadas com (`a`) e (`b`).

```
      .code
FUNC  PROC    VAL: DWORD
      ...
      ADD     EAX, ECX ;(b)
      ...
      RET
FUNC  ENDP

SUBR  PROC  X:PTR BYTE, T:BYTE
      LOCAL   TEMP:DWORD
      ...
      XOR     EAX, EAX ;(a)
      PUSH    EAX
```

```
      ...
      invoke  FUNC, EAX
      ...
      POP     EAX
      ...
      RET
SUBR  ENDP

main: ...
      invoke SUBR,
           offset buf, num ;*
      ...
END   main
```

9. Considere o seguinte programa:

```
include mpcp.inc

func1 PROTO vect: ptr byte, nelem:dword
func2 PROTO arg1: dword

  .data
val byte 13, 21, 12, 20
outmsg byte "%d",13,10,0
  .code
main:
  call    rotina      ; + *
  invoke  _getch
  invoke  ExitProcess,0

rotina proc
   invoke func1, offset val, lengthof val
   invoke printf, offset outmsg, eax
   ret               ; +
rotina endp
```

```
func1 proc uses ebx esi vect: ptr byte,
                        nelem: dword
  mov    ecx, nelem;  (+)
  mov    esi, vect
  mov    ebx, 0
ciclo:
  movsx  edx, byte ptr[esi]
  push   ecx
  invoke func2, edx    ; * +
  pop    ecx           ; *
  add    ebx, eax
  inc    esi
  loop   ciclo
fim:
  mov    eax, ebx
  ret                  ; +
func1 endp

func2 proc arg1: dword
  mov    eax, 1        ; +
  and    eax, arg1
  jz     @F
  mov    eax, 0
  jmp    fim
@@:
  mov    eax, arg1
fim:
  ret                  ; +
func2 endp

end main
```

a) Analise o programa e descreva a sua função.

b) Apresente o prólogo e o epílogo das sub-rotinas `func1` e `func2`. Confirme as respostas usando a vista `Assembly` do Visual Studio.

c) Indique o conteúdo da pilha do sistema após a execução das instruções marcadas com '*' e antes das instruções marcadas com '+'.

10. [Exame de 13-06-2011] Um número primo é um número inteiro positivo que só é divisível (i.e. tem resto zero) por 1 e por ele próprio. Um algoritmo simples para verificar se um número N é primo consiste em testar a divisibilidade de N por todos os inteiros no intervalo $[2, N/2]$.

a) Escreva uma sub-rotina para determinar se um número N, passado como argumento, é primo. A rotina deve retornar 1 se N for primo e 0 em caso contrário. O protótipo é:

```
primo PROTO n:DWORD
```

b) Assuma que tem um vetor `Vect` de números naturais (DWORD) declarado em memória. Use a sub-rotina da alínea anterior para escrever um fragmento de código que coloque em EAX o menor e em EDX o maior dos números primos presentes em `Vect`. Por exemplo, se `Vect` = [1, 3, 7, 21, 23, 25], no final da execução `EAX=1` e `EDX=23`.

11. [Exame de 13-06-2011] Considere o seguinte programa:

```
1   include mpcp.inc
2   func1 proto p: ptr sword, n: word, lim: sword
3
4         .data
5   T sword 3, -32, 7, 10, -5
6   k sword 4
7   outmsg byte "%u",13,10,0
8         .code
9   main:
10        invoke func1, offset T, lengthof T, k
11        add    eax, 0
12        invoke printf, offset outmsg, eax
13        invoke _getch
14        invoke ExitProcess, 0
15
16  func1 proc uses ebx esi v: ptr sword, n: word, lim: sword
17        xor    ebx, ebx
18        mov    esi, v
19        movzx  ecx, n
20  nx:   mov    ax, [esi]
21        cmp    ax, lim
22        jle    @F
23        call   func2
24        add    ebx, eax
25  @@:   add    esi, 2
26        loop   nx
27        mov    eax, ebx
28        ret
29  func1 endp
30
31  func2 proc
32        push   ecx
33        imul   ax
34        mov    ecx, eax
35        mov    ax, dx
36        sal    eax, 16
37        mov    edx, ecx
38        mov    ax, dx
39        pop    ecx
40        ret
41  func2 endp
42  end main
```

a) Descreva a sub-rotina `func2` e indique a sua funcionalidade.

b) Determine o valor apresentado no monitor no final da execução do programa. Justifique.

c) Indique, justificando, o valor que seria retornado pela sub-rotina `func1` se executasse `invoke func1, offset T, 3, 7`.

d) Assuma que imediatamente antes da execução da instrução da linha 18 o valor de ESP é `18FF6CH` e que o estado da pilha é o seguinte:

endereço (hex.)	conteúdo (hex.)
18FF84	00000004
18FF80	00000005
18FF7C	00403000
18FF78	0040103B
18FF74	0018FF94
18FF70	7EFDE000
18FF6C	00000000

Para cada uma das alíneas seguintes indique a resposta correta.

i. O valor contido no endereço `18FF6C` refere-se ao conteúdo de:

A. EBX B. EAX C. EBP D. ESI

ii. O endereço de memória (em hexadecimal) a partir do qual se encontra codificada a instrução da linha 11 é:

A. `00403000` B. `0040103B` C. `7EFDE000` D. `0018FF94`

12. A rotina *recursiva* `maxnum` determina o valor máximo de uma sequência de N números do tipo *dword*. O seu protótipo é:

```
maxnum PROTO SEQ: PTR DWORD, N:DWORD
```

a) Escrever e testar a rotina `maxnum`.

b) Produzir um diagrama que represente o conteúdo da pilha na situação em que esta atinge a sua maior extensão durante a execução da seguinte invocação:

```
        .data
valores dword 4, 8, 11, 2, 9, 7
        .code
        invoke maxnum, offset valores, lengthof valores
        ...
```

Fim.