

# Desenvolvimento de programas

João Canas Ferreira

Março 2014



## *Assuntos*

- 1 Aspectos gerais
- 2 Exemplo 1: Varrimento de uma sequência
- 3 Exemplo 2: Ordenação e pesquisa

## Programas com vários módulos

- Programas são geralmente compostos por vários módulos
  - Bibliotecas de sub-rotinas pré-compiladas
  - Grupos de sub-rotinas relacionadas
- A utilização de módulos facilita o desenvolvimento:
  - facilidade de edição e visualização
  - recompilação parcial após modificação
  - módulos escritos em diferentes linguagens (assembly, C/C++)

### Alguns aspetos importantes

- Como manter a coerência entre definição e utilização?
  - Organização do código
- Como gerir a recompilação?
  - Aplicações dedicadas: make, ant, ambientes de desenvolvimento (Eclipse, Visual Studio),...

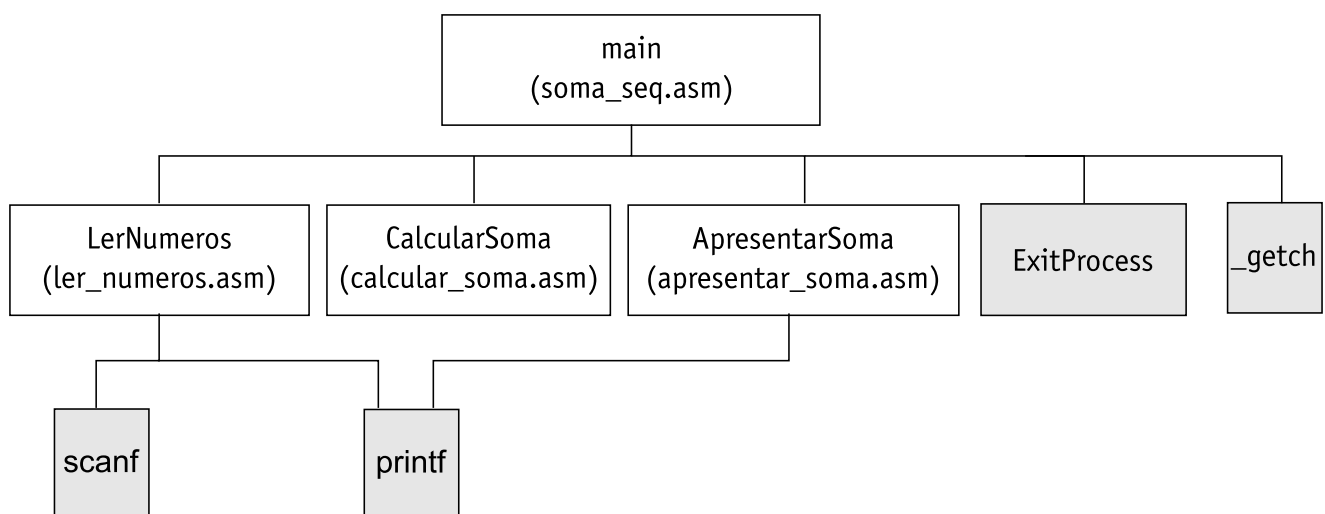
## Regras básicas de organização do código

- 1 Criar o módulo principal com o *ponto de entrada* para o programa.
  - Em C/C++ corresponde à função `main()`
- 2 Separar sub-rotinas por ficheiros
  - Sub-rotinas grandes: uma por ficheiro
  - Sub-rotinas pequenas: várias por ficheiro (se estiverem logicamente relacionadas)
- 3 Usar a diretiva `PROTO` para declarar as interfaces das sub-rotinas
- 4 Criar um ficheiro de protótipos (`.inc`) para cada módulo.  
O ficheiro de protótipos deve ser incluído:
  - no próprio módulo (`.asm`) [Porquê?]
  - em todos os outros módulos que invoquem sub-rotinas declaradas nesse ficheiro

- 1 Aspectos gerais
- 2 Exemplo 1: Varrimento de uma sequência
- 3 Exemplo 2: Ordenação e pesquisa

## Exemplo 1: Somar uma sequência

### Estrutura geral do programa



## Exemplo 1: Módulo principal

```
include mpcp.inc
include apresentar_soma.inc
include ler_numeros.inc
include calcular_soma.inc
NInteiros = 3 ; constante simbólica
.data
msg1 BYTE "Introduza um numero inteiro:",0
msg2 BYTE "Soma = ", 0
seq DWORD NInteiros DUP(?)
.code
main: invoke LerNumeros,offset msg1,offset seq,NInteiros
        invoke CalcularSoma, offset seq, NInteiros
        invoke ApresentarSoma, offset msg2, eax
        invoke _getch
        invoke ExitProcess, 0
END main
```

## Exemplo 1: Leitura de números

```
include mpcp.inc
include ler_numeros.inc
.data
num DWORD ?
format BYTE " %d",0
.code
LerNumeros PROC USES ESI ptrMsg:PTR BYTE, ptrSeq:PTR DWORD, nElem:DWORD
    mov     ecx, nElem
    cmp     ecx, 0
    jle     fim ; verificação do argumento
    mov     esi, ptrSeq
@@: push    ecx ; ECX pode ser alterado por print, etc.
    invoke  printf, ptrMsg
    invoke  scanf, offset format, esi
    pop     ecx ; recuperar o contador
    add     esi, 4
    loop    @B ; saltar para trás (B), para @@ mais próximo
fim: ret
LerNumeros ENDP
END
```

## Exemplo 1: Cálculo

```
include mpcp.inc
include calcular_soma.inc
.code
CalcularSoma PROC ptrSeq:PTR DWORD, nElem:DWORD
    mov     eax, 0
    mov     ecx, nElem
    cmp     ecx, 0
    jle     fim          ; verificação do argumento
    mov     edx, ptrSeq

@@:        add     eax, [edx] ; resultado em EAX
           add     edx, 4
           loop    @B
fim:       ret
CalcularSoma ENDP
END
```

## Exemplo 1: Apresentação de resultados

```
include mpcp.inc
include apresentar_soma.inc

.data
formato BYTE " %d",13,10,0
.code

ApresentarSoma PROC ptrMsg:PTR BYTE, soma:DWORD
    invoke  printf, ptrMsg
    invoke  printf, offset formato, soma
    ret
ApresentaSoma ENDP

END
```

- 1 Aspectos gerais
- 2 Exemplo 1: Varrimento de uma sequência
- 3 Exemplo 2: Ordenação e pesquisa

## Ordenação e pesquisa

### Funcionalidade

- Preencher sequência com números pseudo-aleatórios
- Ordenar a sequência (Bubble sort)
- Procurar um valor na sequência (pesquisa binária)

### Geração de números pseudo-aleatórios

- Escolher um valor inicial (semente)
- Calcular o próximo valor da sequência:

$$x_{n+1} = f(x_n)$$

- A função  $f()$  deve ser escolhida de forma a produzir uma sequência muito longa sem repetições.
- Funções de C:
  - `rand PROC` retorna número em `[0;MAX_INT]`
  - `srand PROC semente:DWORD` estabelece valor inicial

## Módulo principal (1/2)

```
include mpcp.inc
```

```
include preencher.inc
```

```
include bubblesort.inc
```

```
include imprimirseq.inc
```

```
include pesqbin.inc
```

```
MostrarResultado PROTO res:SDWORD
```

```
PedirValor PROTO
```

```
NELEM = 5
```

```
.data
```

```
seq DWORD NELEM dup(?)
```

```
.code
```

```
main:
```

```
invoke srand, 123
```

```
invoke Preencher, offset seq, NELEM
```

```
invoke ImprimirSeq, offset seq, NELEM
```

```
invoke _getch
```

```
invoke BubbleSort, offset seq, NELEM
```

```
invoke ImprimirSeq, offset seq, NELEM
```

```
invoke PedirValor
```

```
invoke PesqBin, offset seq, NELEM, eax
```

```
invoke MostrarResultado, eax
```

```
invoke _getch
```

```
invoke ExitProcess, 0
```

## Módulo principal (2/2)

```
;-----
```

```
; Pedir um valor ao utilizador
```

```
; Retorna: EAX = valor lido
```

```
;-----
```

```
PedirValor PROC
```

```
    .data
```

```
; Não são variáveis locais
```

```
msg     BYTE 13,10,  
        "Valor a procurar: ",0
```

```
msg1    BYTE " %d",0
```

```
lido    DWORD ?
```

```
    .code
```

```
    invoke printf, offset msg
```

```
    invoke scanf, offset msg1,  
        offset lido
```

```
    mov     eax, lido
```

```
    ret
```

```
PedirValor ENDP
```

```
;-----
```

```
; Mostrar resultado da pesquisa
```

```
;-----
```

```
MostrarResultado PROC res:SDWORD
```

```
    .data
```

```
msg3 BYTE "Valor nao encontrado",  
        13,10,0
```

```
msg4 BYTE "Valor na posicao ",13,10,0
```

```
    .code
```

```
    .IF res == -1
```

```
    invoke printf, offset msg3
```

```
    .ELSE
```

```
    invoke printf, offset msg4,  
        res
```

```
    .ENDIF
```

```
    ret
```

```
MostrarResultado ENDP
```

```
END main
```

## Preencher sequência com números pseudo-aleatórios

```
include mpcp.inc
include preencher.inc

.code

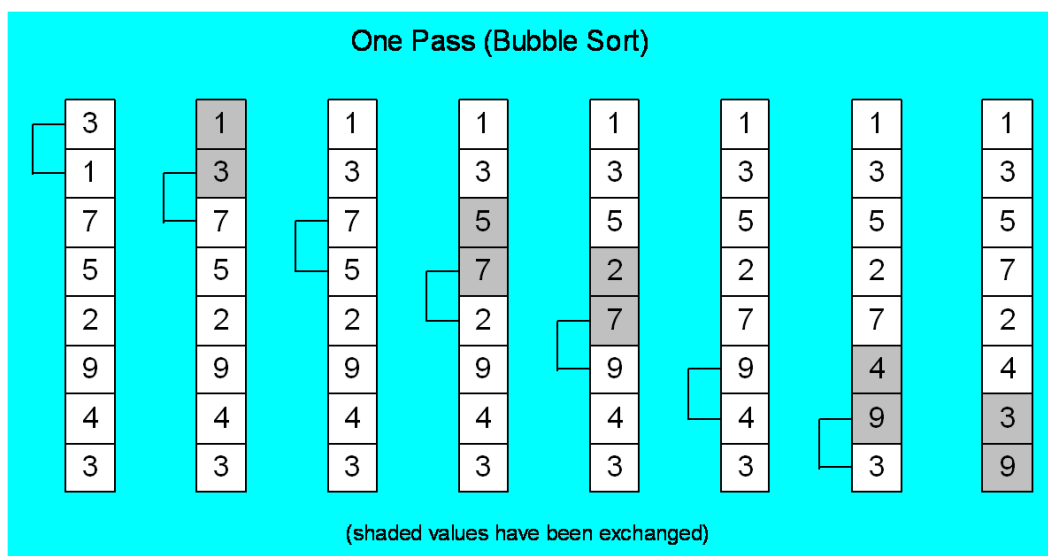
Preencher PROC USES EDI ptrSeq:PTR SDWORD, nElem:DWORD,
    mov     edi, ptrSeq
    mov     ecx, nElem

@@:        push     ecx                ; preservar contador
            invoke   rand              ; EAX com nº aleatório
            pop      ecx              ; recupera valor do contador
            mov      [edi], eax
            add      edi, 4            ; preparar para próximo elemento
            loop     @B
            ret

Preencher ENDP
END
```

## Exemplo de uma passagem de Bubblesort

Fazer passagem pela sequência, trocando elementos sucessivos que estejam fora de ordem.



Fonte: [Irvine03]

O algoritmo completo faz  $n - 1$  passagens para ordenar a sequência.



## Bubblesort: pseudo-código

```
bubblesort (int seq[], int nElem)
{
    int i, j, pos;
    i = nElem - 1;

    while (i > 0) /* ciclo exterior */
    {
        j = i; pos = 0;
        while (j > 0) /* varrimento */
        {
            if (seq[pos] > seq[pos+1])
                trocar(seq[pos], seq[pos+1])
            pos = pos+1;
            j = j-1;
        }
        i = i-1;
    }
}
```

Complexidade  $O(n^2)$ : número de comparações proporcional ao quadrado do número de elementos. Há melhor...

## Bubblesort: Implementação

```
BubbleSort PROC USES ESI ptrSeq:PTR SDWORD, nElem:DWORD
    mov     ecx,nElem
    dec     ecx                ; número de iterações
L1:        push    ecx        ; guardar contador ciclo externo
    mov     esi,ptrSeq
L2:        mov     eax,[esi]
    cmp     [esi+4],eax        ; comparar posições sucessivas
    jge     L3                ; se [esi+4] >= [esi], não trocar
    xchg    eax,[esi+4]        ; trocar
    mov     [esi],eax
L3:        add     esi,4
    loop    L2                ; ciclo interno
    pop     ecx                ; obter contador do ciclo externo
    loop    L1                ; ciclo externo
L4:        ret
BubbleSort ENDP
```

## Pesquisa numa sequência

- Problema: determinar se um elemento existe num conjunto (vetor)

### Pesquisa linear

- Executar um varrimento do conjunto do início para o fim
- Se encontra o elemento, terminar
- Complexidade ordem  $O(n)$ : número máximo de comparações proporcional ao número de elementos

### Pesquisa binária

- Algoritmo de pesquisa bem adaptado para conjuntos grandes
- Exige que conjunto esteja ordenado
- Estratégia de divisão-e-conquista
  - testar o elemento central do intervalo de pesquisa
  - se não for o elemento desejado, escolher a metade apropriada e repetir
- A cada iteração, o intervalo de pesquisa é reduzido em metade
- Algoritmo de ordem de complexidade  $O(\log n)$ :
  - No pior dos casos, o número de comparações é proporcional ao logaritmo do número de elementos do conjunto

## Desempenho de pesquisa binária

▢ Medida indireta de desempenho: número máximo de comparações

Número de elementos N	Número máximo de comparações $\lceil \log_2(N) \rceil + 1$
64	7
1024	11
65536	17
1048576	21
4294967296	33

## Exemplo: pesquisa binária

Procurar pelo valor 60 numa sequência ordenada

0	1	2	3	4	5	6	7	8	9
9	12	19	31	45	54	60	71	78	81

Iteração	primeiro	ultimo	meio	Comparação	Ajustar
1	0	9	4	45 < 60	primeiro
2	5	9	7	71 > 60	ultimo
3	5	6	5	54 > 60	primeiro
4	6	6	6	60 = 60	encontrado

➡ Se o valor 60 não existisse na sequência, ficava-se com `ultimo < primeiro`. Quando esta condição ocorre, o algoritmo também deve terminar.

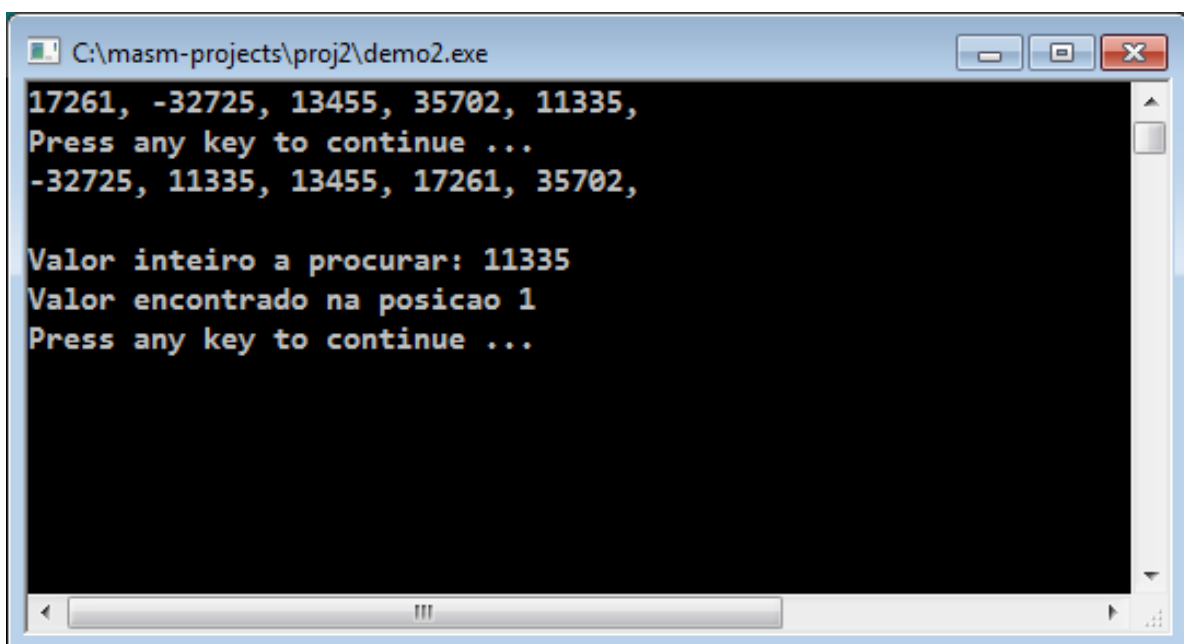
## Pesquisa binária: pseudo-código

```
int PesqBin( int valores[], int nElem, int procurado )
{
    int primeiro = 0; int ultimo = nElem - 1;
    while (primeiro <= ultimo)
    {
        meio = (ultimo + primeiro) / 2;
        if (valores[meio] < procurado)
            primeiro = meio + 1;
        else if (valores[meio] > procurado)
            ultimo = meio - 1;
        else
            return meio;    // encontrado
    }
    return -1;             // não encontrado
}
```

## Pesquisa binária: implementação

```
PesqBin PROC USES EBX EDI ESI
    ptrSeq:PTR SDWORD,
    nElem:DWORD, valor:SDWORD
LOCAL primeiro:DWORD, ultimo:DWORD,
    meio:DWORD
    mov primeiro,0
    mov eax, nElem
    dec eax
    mov ultimo, eax
    mov edi, valor
    mov ebx, ptrSeq
    mov eax, primeiro
.WHILE eax <= ultimo
    mov eax, ultimo
    add eax, primeiro
    shr eax, 1
    mov meio, eax
    mov esi, meio
    shl esi, 2          ; ×4
    mov edx, [ebx+esi]
    .IF SDWORD PTR edx < edi
        mov eax, meio
        inc eax
        mov primeiro, eax
    .ELSEIF SDWORD PTR edx > edi
        mov eax, meio
        dec eax
        mov ultimo, eax
    .ELSE
        mov eax, meio
        jmp fim
    .ENDIF
    mov eax, primeiro
.ENDW
nao_existe:
    mov eax, -1
fim:
    ret
PesqBin ENDP
```

## Exemplo 2: Execução



```
C:\masm-projects\proj2\demo2.exe
17261, -32725, 13455, 35702, 11335,
Press any key to continue ...
-32725, 11335, 13455, 17261, 35702,

Valor inteiro a procurar: 11335
Valor encontrado na posicao 1
Press any key to continue ...
```

► Mais informação sobre estes assuntos está nos capítulos 8 e 9 de:

[Irvine03](#) K. R. Irvine, *Assembly Language for Intel-Based Computers*, 4ª edição, Prentice Hall, 2003.