Praticas 08b

O objetivo do programa é implementar o algoritmo de ordenação Bubble Sort. Nesse método, o algoritmo caminha elemento a elemento no vetor e compara cada elemento com o elemento vizinho (proximo), iniciando com o primeiro elemento e terminando com o penúltimo (tam-1 elementos percorridos). Se o elemento próximo for maior que o elemento corrente, troca-os de posicao e avança no vetor. Faz isso em todo o vetor e dessa forma, o maior elemento vai parar na ultima posicao. Completa-se então 1 ciclo. Decrementa o tamanho do vetor de 1 (tam←tam-1), para desconsiderar o último elemento que já está na posição correta e repete mais um ciclo de operação com um vetor menor. Repete sucessivamente até tam = 1.

Para gerar o executável, gere primeiro o objeto executando o seguinte comando:

```
as praticas_08b.s -o praticas_08b.o
```

e depois link dinamicamente com o seguinte comando:

```
ld praticas_08b.o -l c -dynamic-linker /lib/ld-linux.so.2 -o praticas_08b
```

O executavel se chamara praticas_08b, sem extensão, e para executá-lo digite:

```
./praticas_08b
```

```
.section .data

# pratica_08b.s: : vide arquivo praticas_08b.pdf

.section .data

titulo:   .asciz    "\n*** Ordena Vetor Bubble Sort ***\n\n"

pedetam:  .asciz    "Digite o tamanho do vetor (maximo=20) => "

formato:  .asciz    "%d"

pedenum:  .asciz    "Entre com o elemento %d => "

mostra1:  .asciz    "Elementos Lidos:"

mostra2:  .asciz    " %d"

mostra3:  .asciz    "\nElementos Ordenados:"

pulalin:  .asciz    "\n"

maxtam: .int     20

tam:  .int  0

tamaux:     .int  0

num:  .int  0

soma: .int  0
```

```
vetor:     .space  80


.section .text


.globl _start
_start:

      pushl $titulo
      call  printf


letam:
      pushl $pedetam
      call  printf
      pushl $tam
      pushl $formato
      call  scanf
      pushl $pulalin
      call  printf

      movl  tam, %ecx
      movl  %ecx, tamaux     # copia auxiliar do tamanho
      cmpl  $0, %ecx
      jle   letam
      cmpl  maxtam, %ecx
      jg    letam

      movl  $vetor,%edi
      addl  $16, %esp
        movl     $0, %ebx


lenum:
      incl  %ebx
        pushl    %edi
      pushl %ecx
      pushl %ebx

      pushl      $pedenum
      call  printf
      pushl $num
      pushl $formato
      call  scanf
      pushl $pulalin
      call  printf
      addl  $16, %esp

      popl  %ebx
      popl  %ecx
      popl  %edi
      movl  num, %eax
      movl  %eax, (%edi)
      addl  $4, %edi
      loop  lenum
```

```
mostravet:
      pushl $mostra1
      call  printf
      addl  $4, %esp
      movl  tam, %ecx
      movl  $vetor, %edi

mostranum:
      movl  (%edi), %ebx
      addl  $4, %edi

      pushl %edi
      pushl %ecx
      pushl %ebx

      pushl $mostra2
      call  printf
      addl  $8, %esp

      popl  %ecx
      popl  %edi
      loop  mostranum

      movl  tam, %ecx
      cmpl  $1, %ecx
      jle   mostravet2  # nao tem mais oque fazer

ordenavetor:

      movl  $vetor, %edi     # inicia a posicao do primeiro
      movl  %edi, %esi
      addl  $4, %esi   # inicia a posicao do vizinho a frente

      subl  $1, %ecx
      pushl %ecx       # backup do nro de elementos a comparar

giro:
      movl  (%edi), %eax     #   contem   o   menor   valor   ateh   entao
identificado
      movl  (%esi), %ebx     # contem o valor do proximo
      cmpl  %eax, %ebx
      jl    trocaelemento

avanca:
      addl  $4, %edi   # avanca para o proximo
      addl  $4, %esi   # avanca para o proximo vizinho
      loop  giro

proximociclo:

      movl  tamaux, %eax
      decl  %eax
      movl  %eax, tamaux
      movl  %eax, %ecx

      cmpl  $1, %ecx
      jle   mostravet2  # nao tem mais oque fazer
```

```
      jmp   ordenavetor

trocaelemento:

      movl  %eax, (%esi)     # troca valores entre os vizinhos
      movl  %ebx, (%edi)
      jmp   avanca

mostravet2:
      pushl $mostra3
      call  printf
      addl  $4, %esp
      movl  tam, %ecx
      movl  $vetor, %edi

mostranum2:
      movl  (%edi), %ebx
      addl  $4, %edi

      pushl %edi
      pushl %ecx
      pushl %ebx

      pushl $mostra2
      call  printf
      addl  $8, %esp

      popl  %ecx
      popl  %edi
      loop  mostranum2

      pushl $pulalin
      call  printf
      pushl $pulalin
      call  printf
      addl  $8, %esp
fim:
      pushl $0
      call  exit
```

DESAFIO 1: Este algoritmo pode ser melhorado com o uso de uma flag. Toda vez que houver uma troca, a posição de troca é guardada na flag. A partir do ponto que não houver troca, significa que todos os elementoS dali até o final já estão ordenados. Desta forma, a cada ciclo, o tamanho não precisa reduzir de 1, mas sim até o ponto da última troca. Implemente essa alteração.

DESAFIO 2: Tente novamente implementar o algoritmo de ordenação “Inserção Direta”. Nesse metodo, o algoritmo percorre o vetor da primeira posição até a última, de um em um elemento. Cada elemento que ele encontra no percurso (elemento corrente) o algoritmo o insere na parte do vetor que ficou pra trás, na sua posição ordenada. Em determinado momento do algoritmo, no meio do percurso, o vetor contém duas metades: uma ordenada (pra trás) e outra a ordenar (pra frente).Quando o algoritmo atingir o ultimo elemento, o vetor ficou totalmente ordenado. OBS: Para inserir o elemento corrente na parte do vetor que ficou pra trás (parte ordenada do vetor), o algoritmo compara esse elemento com todos os elementos de trás, arrastando-os pra frente até encontrar a posicao de inserção, na qual é

inserido o elemento corrente, expandido de 1 a parte ordenada do vetor e diminuindo de 1 a parte a ordenar. O processo é repetido até a parte ordenada atingir todo o vetor e a parte a ordenar ficar vazia.