PARADIGMAS DE LINGUAGENS DE PROGRAMAÇÃO – 3ª LISTA DE EXERCÍCIOS – parte b
Programação Estruturada – 2019-1 – Profa. Heloisa Camargo

1. Considere o seguinte programa escrito em uma linguagem semelhante ao C:

```
void main ( ) {
    int index = 2, lista [5] = { 1, 3, 5, 7, 9};
    troca (index, lista [ 0 ] );
    troca (lista [ 0 ], lista [ 1 ] );
    troca (index, lista [ index ] );
}
void troca ( int a, int b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

Para cada um dos métodos de passagem de parâmetros seguintes, descreva as ações realizadas com relação a essa passagem e diga quais são os valores das variáveis index e lista antes e depois de cada uma das três chamadas a troca. Mostre o conteúdo da pilha de execução no ponto em que termina a execução da segunda chamada de troca para o segundo item (passagem por referência).

- passados por valor
- passados por referência
- passados por valor-resultado

1. passados por valor

- index = 2; valor = {1, 3, 5, 7, 9}
- chamada de troca (index, lista [0]):
  - copia o valor de index em a (a=2)
  - copia o valor de lista[0] em b (b=1)
  - executa as instruções de troca, trocando valores de a e b
  - retorna
- valores de index e lista[0] não são trocados :
- index = 2; valor = {1, 3, 5, 7, 9}
- chamada de troca (lista [0], lista [1]):
  - copia o valor de lista[0] em a (a=1)
  - copia o valor de lista[1] em b (b=2)
  - executa as instruções de troca, trocando valores de a e b
  - retorna
- valores de lista[0] e lista[1] não são trocados
- index = 2; valor = {1, 3, 5, 7, 9}
- chamada de troca (index, lista [index]):
  - copia o valor de index em a (a=2)
  - copia o valor de lista[index] em b (b=5)
  - executa as instruções de troca, trocando valores de a e b
  - retorna
- valores de index e lista[index] não são trocados
- index = 2; valor = {1, 3, 5, 7, 9}

2. passados por referência

- index = 2; valor = {1, 3, 5, 7, 9}
- chamada de troca (index, lista [0]):
  - copia o endereço de index em a (a=&index)

- copia o endereço de lista[0] em b (b=&lista[0])
- executa as instruções de troca, trocando valores de a e b
- retorna

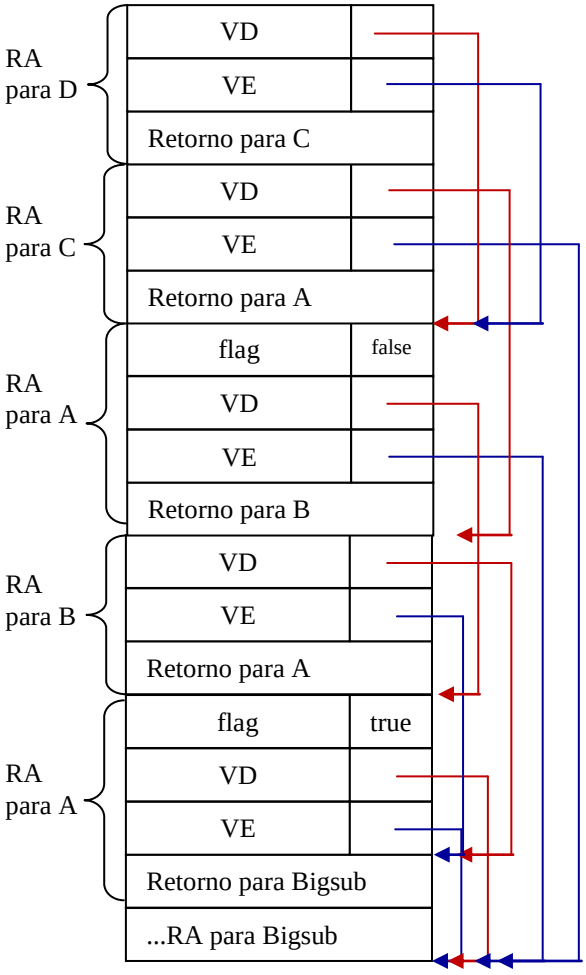
- valores de index e lista[0] são trocados :
- index = 1; valor = {2, 3, 5, 7, 9}
- chamada de troca (lista [0], lista [1]):
  - copia o endereço de lista[0] em a (a=&lista[0])
  - copia o endereço de lista[1] em b (b=&lista[1])
  - executa as instruções de troca, trocando valores de a e b
  - retorna
- valores de lista[0] e lista[1] são trocados
- index = 1; valor = {3, 2, 5, 7, 9}
- chamada de troca (index, lista [index]):
  - copia o endereço de index em a (a=&index)
  - copia o endereço de lista[index] em b (b=&lista[index])
  - executa as instruções de troca, trocando valores de a e b
  - retorna
- valores de index e lista[index] são trocados
- index = 2; valor = {3, 1, 5, 7, 9}

3. passados por valor-resultado

- index = 2; valor = {1, 3, 5, 7, 9}
- chamada de troca (index, lista [0]):
  - salva o endereço do parâmetro real index (addr_index = &índex)
  - salva o endereço do parâmetro real lista[0] (addr_lista_0 = &lista_0)
  - copia o valor de index em a (a=*addr_index, fica a = 2)
  - copia o valor de lista[0] em b (b=*addr_lista_0, fica b = 1)
  - executa as instruções de troca, trocando valores de a e b
  - copia o valor de a na posição ocupada por index (*addr_index=a, fica index=1)
  - copia o valor de b na posição ocupada por lista[0] (*addr_lista_0=b, fica lista[0] = 2)
  - retorna
- valores de index e lista[0] são trocados :
- index = 1; valor = {2, 3, 5, 7, 9}
- chamada de troca (lista [0], lista [1]):
  - salva o endereço do parâmetro real lista[0] (addr_lista_0 = &lista_0)
  - salva o endereço do parâmetro real lista[1] (addr_lista_1 = &lista_1)
  - copia o valor de lista[0] em a (a=*addr_lista_0, fica a = 2)
  - copia o valor de lista[1] em b (b=*addr_lista_1, fica b = 3)
  - executa as instruções de troca, trocando valores de a e b
  - copia o valor de a na posição ocupada por lista[0] (*addr_lista_0=a, fica lista[0]=3)
  - copia o valor de b na posição ocupada por lista[1] (*addr_lista_1=b, fica lista[1] = 2)
  - retorna
- valores de lista[0] e lista[1] são trocados
- index = 1; valor = {3, 2, 5, 7, 9}
- chamada de troca (index, lista [index]):
  - salva o endereço do parâmetro real index (addr_index = &index)
  - salva o endereço do parâmetro real lista[1] (addr_lista_1 = &lista_1)
  - copia o valor de index em a (a=*addr_index, fica a = 1)
  - copia o valor de lista[1] em b (b=*addr_lista_1, fica b = 2)
  - executa as instruções de troca, trocando valores de a e b
  - copia o valor de a na posição ocupada por index (*addr_index=a, fica index=2)
  - copia o valor de b na posição ocupada por lista[1] (*addr_lista_1=b, fica lista[1] = 1)
  - retorna
- valores de index e lista[1] são trocados
- index = 2; valor = {3, 1, 5, 7, 9}

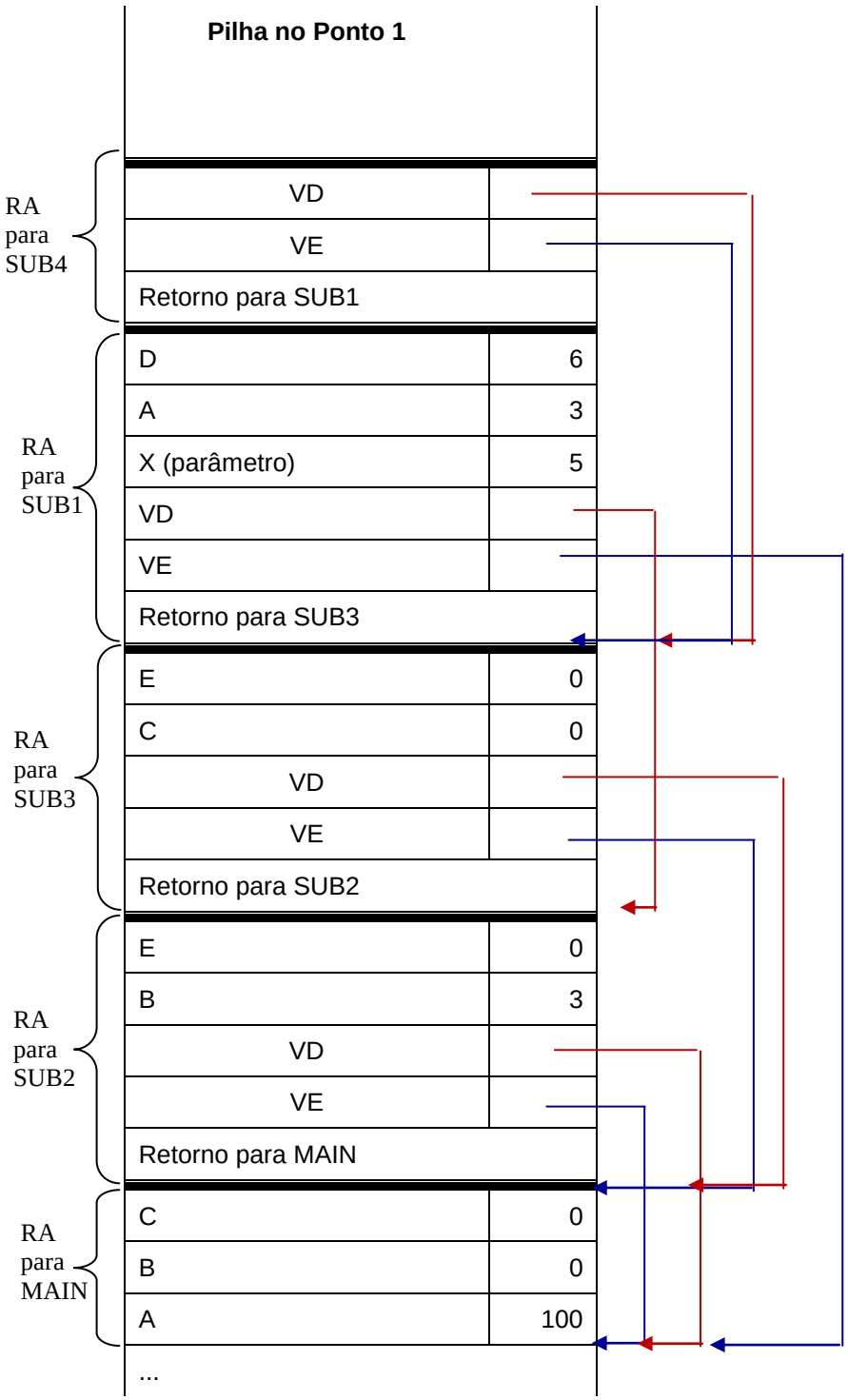
2. Mostre o conteúdo da pilha de execução com todas as instâncias do registro de ativação, incluindo encadeamentos estáticos e dinâmicos e valores da variável ***flag***, quando a execução atingir o ponto 1 no programa esquemático seguinte, escrito em linguagem Ada que, como o Pascal, permite aninhamento de subprogramas. Suponha que BIGSUB esteja no nível 1. A sequência de chamada desse programa para que a execução atinja D é:
BIGSUB chama A, A chama B, B chama A, A chama C, C chama D.

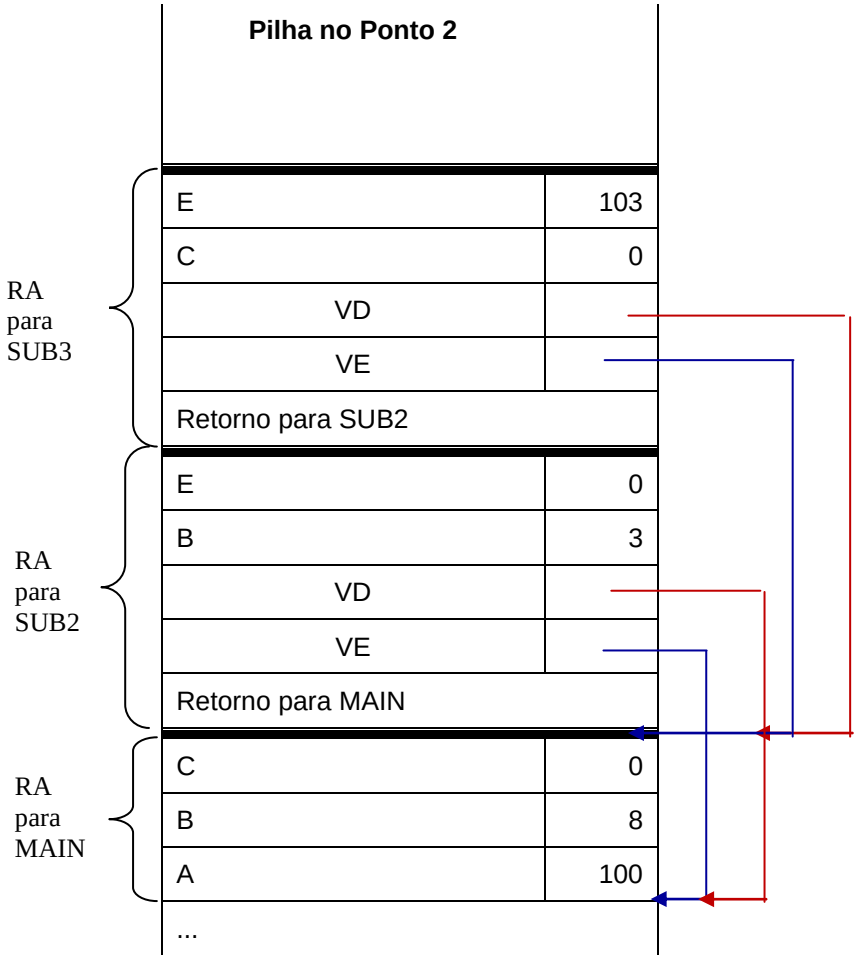
```
procedure BIGSUB  is
    procedure A (flag: boolean)   is
    procedure B  is
            ....
      A(false);
      end;      --  fim de B
    begin       -- começo de A
    if flag
      then B;
      else C;
    ...
    end ;      --  fim de A
  procedure C  is
    procedure D  is
       .....   ----------------------------→ 1
       end;     --   fim de D
     begin – começo de C     ...
    D;
    end;      -- fim de C
  begin    --  começo de Bigsub
    ...
  A(true);
  ....
  end;      -- fim de Bigsub
```

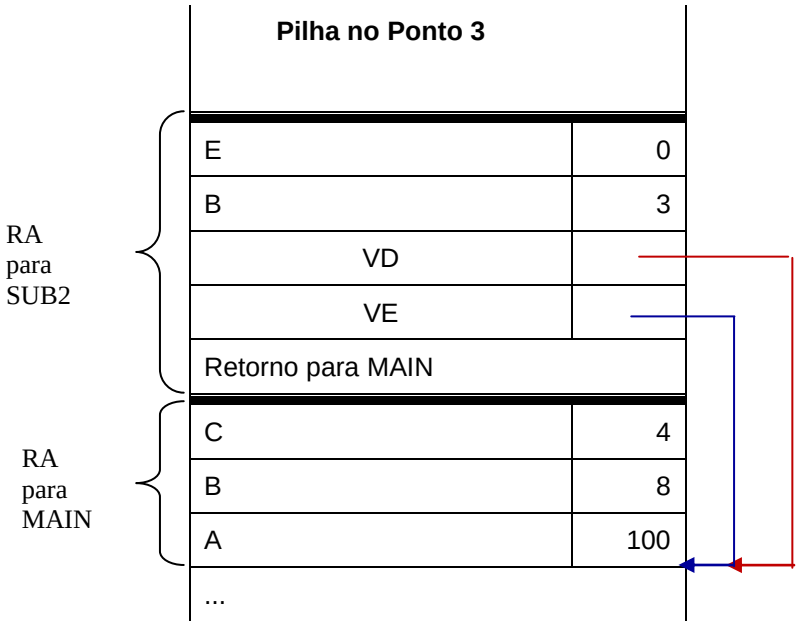


3. Para o programa dado a seguir, mostre o conteúdo da pilha de execução, desenhando uma pilha diferente para cada um dos pontos 1, 2 e 3 indicados no código. O parâmetro X é passado por valor. Todas as variáveis são inicializadas com zero. Relate, seguindo o fluxo de execução, quando ocorreram as ações de: chamada de procedimento; criação de RA para um procedimento; início de execução de procedimento; término de execução de procedimento; retorno de chamada de procedimento; destruição de RA de um procedimento; busca por variável não local; inicialização (feitas por comando de atribuição) e alteração de variáveis e parâmetros.

```
program MAIN;
  var A, B, C : integer;
  procedure SUB1 (X : integer) ;
    var A, D : integer;
    procedure SUB4;
      begin {SUB4 }
       .....
       A : = D / 2; ----------------------------------------→ 1
       ......
      end; { SUB4 }
    begin { SUB1}
    .....
    D := X + 1;
    ......
    SUB4;
    .....
    B : = A + X;
    .....
    end; { SUB1}
  procedure SUB2;
    var B, E  : integer;
    procedure SUB3;
      var C, E : integer;
      begin { SUB3 }
      B := 3;
      SUB1( 5);
      .......
      E := B + A; ------------------------------------------------→ 2
      end ; {SUB3}
    begin { SUB2 }
      .....
      SUB3;
      ......
      C := B + 1; ----------------------------------------------→ 3
      ....
    end; { SUB2 }
  begin { MAIN }
      A := 100;
      .....
      SUB2;
      ....
  end; { MAIN }
```


Pilha no Ponto 1
RA para SUB4
VD
VE
Retorno para SUB1
RA para SUB1
D 6
A 3
X (parâmetro) 5
VD
VE
Retorno para SUB3
RA para SUB3
E 0
C 0
VD
VE
Retorno para SUB2
RA para SUB2
E 0
B 3
VD
VE
Retorno para MAIN
RA para MAIN
C 0
B 0
A 100
...


Pilha no Ponto 2
RA para SUB3
E 103
C 0
VD
VE
Retorno para SUB2
RA para SUB2
E 0
B 3
VD
VE
Retorno para MAIN
RA para MAIN
C 0
B 8
A 100
...


Pilha no Ponto 3
RA para SUB2
E 0
B 3
VD
VE
Retorno para MAIN
RA para MAIN
C 4
B 8
A 100
...

Início da execução com main
Inicializa variável A com 100 (A := 100);
Main chama sub2;
RA de sub2 é criado;
Inicia execução de sub2;
Sub2 chama sub3;
RA de sub3 é criado;
Inicia execução de sub3;
Busca a variável não local B, encontra em sub2;
Atribui 3 à variável B de sub2;
Sub3 chama sub1 com parâmetro 5;
RA de sub1 é criado;
Parâmetro X de sub1 (passagem por valor) é inicializado com 5;
Inicia execução de sub1;
Variável local D de sub1 recebe 6 (X+1);
Sub1 chama sub4;
RA de sub4 é criado;
Inicial execução de sub4;
Busca variáveis não locais A e D, encontra em sub1;
Atribui 3 a variável A de sub1 (A:= D/2)
Atingiu ponto 1 do código;
Encerra execução de sub4;
Destrói RA de sub4;
Retorna para sub1;
Busca variável não local B, encontra em main;
Atribui 8 à variável B de main (B:= A + X);
Encerra execução de sub1;
Destrói RA de sub1;
Retorna para sub3;
Busca variáveis não locais A e B, encontra B em sub2 e A em main;
Atribui 103 à variável local E, somando os valores de B de sub2 com A de main; (E:=B+A);
Atingiu ponto 2 do código;
Encerra execução de sub3;
Destrói RA de sub3;
Retorna para sub2;
Busca variável não local C, encontra em main;
Altera variável C de main, atribuindo 4, somando o valor de B de sub2 com 1(C := B+1);
Atingiu ponto 3 do código;

4. Considere o programa dado a seguir, escrito em uma linguagem tipo Pascal (que permite subprogramas aninhados). Construa a pilha de execução para este programa até o ponto 1 indicado. Não é necessário mostrar os valores das variáveis.

```
program MAIN_1;
  var P : real;
  procedure A(X : integer);
    var Y : boolean;
    procedure C(Q : boolean);
      begin { C }
      ..... ----------------------------------------→ 1
      end; { C }
   begin { A }
    .....
    C (Y);
    .....
    end; { A }
  procedure B (R : real) ;
    var S, T : integer;
    begin { B }
    .....
    A (S);
    .....
    end; { B }
  begin { MAIN_1 }
  .....
  B (P);
  .....
  end.  { MAIN_1 }
```

5. Considere o seguinte programa em uma linguagem tipo C, que não permite aninhamento de subprogramas. Mostre o conteúdo da pilha de execução nos pontos 1, 2 e 3 indicados, mostrando inclusive o conteúdo das variáveis e dos parâmetros alocados na pilha. Relate, seguindo o fluxo de execução, quando ocorreram as ações de: chamada de procedimento; criação de RA para um procedimento; início de execução de procedimento; término de execução de procedimento; retorno de chamada de procedimento; destruição de RA de um procedimento; busca por variável global; inicialização e alteração de variáveis e parâmetros.

```
main ( )
{     int a, b ;
      a = 2;
      b = 1;
      sub1 ( a, b) ;  ----------------------------→ 2
      soma2 (&a, &b);
}
sub1 (int x, int y)
{
       int z; ------------------→ 1  // se este ponto for atingido mais de uma vez,
                                     // fazer uma pilha para cada passagem
      z = x – y;
      if  (z>0)  sub1(y, z)
      else z = 0;
      printf("%d \n", z) ;
}
soma2 (int *x, int *y)
{     int z, w;    ----------------------------------------→ 3
      z = *x + *y ; w = *x - *y;
      printf("%d \n", z) ;
}
```

6. Considere o programa dado a seguir, escrito em uma linguagem tipo Pascal (que permite subprogramas aninhados). Construa a pilha de execução para este programa até o ponto 1 indicado. O parâmetro X de A é passado por referência. O parâmetro Q de C e o parâmetro R de B são passados por valor. Relate, seguindo o fluxo de execução, quando ocorreram as ações de: chamada de procedimento; criação de RA para um procedimento; início de execução de procedimento; término de execução de procedimento; retorno de chamada de procedimento; destruição de RA de um procedimento; busca por variável não local; inicialização e alteração de variáveis e parâmetros.

```
program MAIN_2;
  var P : real;
  procedure A(var X : integer);
    var Y : boolean;
    procedure C(Q : boolean);
      begin { C }
      .........
      X := X+1;  ----------------------------------------→ 1
      end; { C }
   begin { A }
    …..
    if (X>0) then Y := true;
    C (Y);
    ......
    end; { A }
  procedure B (R : real) ;
    var S, T : integer;
    begin { B }
    ......
    S:= 10;
    A (S);
    ......
    end; { B }
  begin { MAIN_1 }
  P:= 1.5;
   ......
  B (P);
  ......
  end.   { MAIN_1 }
```