Paradigmas de Linguagens de Programação
Lista 3
Prof. Sergio d Zorzo

---

1. Considere o seguinte programa, escrito em uma linguagem tipo PASCAL:

```
program main;
  var x, y, z : integer;
  procedure sub1;
    var a, y, z : integer;
    begin {sub1}
    ...
    end; {sub1}
  procedure sub2;
    var a, b, z : integer;
    begin {sub2}
    ...
    end; {sub2}
  procedure sub3;
    var a, x, w, : integer;
    begin {sub3}
    ...
    end {sub3}
  begin { main }
  ...
  end. { main }
```

Para a sequência de chamadas: main chama sub1; sub1 chama sub2; sub2 chama sub3, diga quais variáveis (locais e não locais) são visíveis durante a execução de sub3, considerando:

a. vinculação de escopo dinâmica

```
Locais: a, x, w
Não-locais: b, z (sub2), y (sub1)
```

b. vinculação de escopo estática

```
Locais: a, x, w
Não-locais: y, z (main)
```

2. Considere o seguinte programa em Pascal. Supondo regras de escopo estático, qual valor de x é impresso no procedimento sub1? E para regras de escopo dinâmico?

```
program main;
  var x : integer;
  procedure sub1;
    begin { sub1 }
    writeln ('x = ', x);
    end; { sub1 }
  procedure sub2;
    var x : integer;
    begin { sub2 }
    x := 10;
    sub1;
    end; { sub2 }
  begin { main }
  x : = 5;
  sub2;
  end. { main }
```

```
Escopo estático:
x = 5
Escopo dinâmico:
x = 10
```

3. Considere o seguinte programa C esquemático:

```
void fun1 (void);
void fun2 (void);
void fun3 (void);
void main ( ) {
  int a, b, c;
  ...
}
void fun1 (void) {
  int b, c, d;
  ...
}
void fun2 (void) {
  int c, d, e;
  ...
}
void fun3 (void) {
  int d, e, f;
  ...
}
```

Dadas as seguintes seqüências de chamadas e supondo-se que seja usado o escopo dinâmico, quais variáveis são visíveis durante a execução da última função chamada? Diga, para cada variável visível na última função, o nome da função em que ela foi declarada.

a. main chama fun1; fun1 chama fun2; fun2 chama fun3.

```
Locais: d, e, f (fun3)
Não-locais: c (fun2), b (fun1), a (main)
```

b. main chama fun2; fun2 chama fun3; fun3 chama fun1.

```
Locais: b, c, d (fun1)
Não-locais: e, f (fun3), a (main)
```

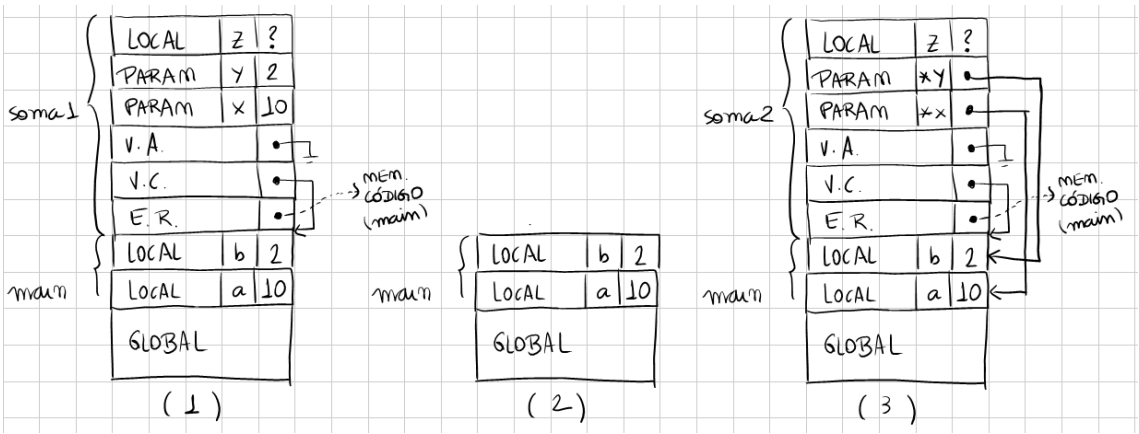
4. Considere o seguinte programa em C:

```
main ( ) {
  int a, b ;
  a = 10;
  b = 2;
  soma1 (a, b) ; ----------------------------> 2
  soma2 (&a, &b);
}

soma1 (int x, int y) {
  ------------------------------------------> 1
  int z;
  z = x + y ;
  printf("%d \n", z) ;
}

soma2 (int *x, int *y) {
  int z; ------------------------------------> 3
  z = *x + *y ;
  printf("%d \n", z);
}
```

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.



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

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

void troca (int a, int b) {
  int temp;
  temp = a;
  a = b;
  b = temp;
}
```

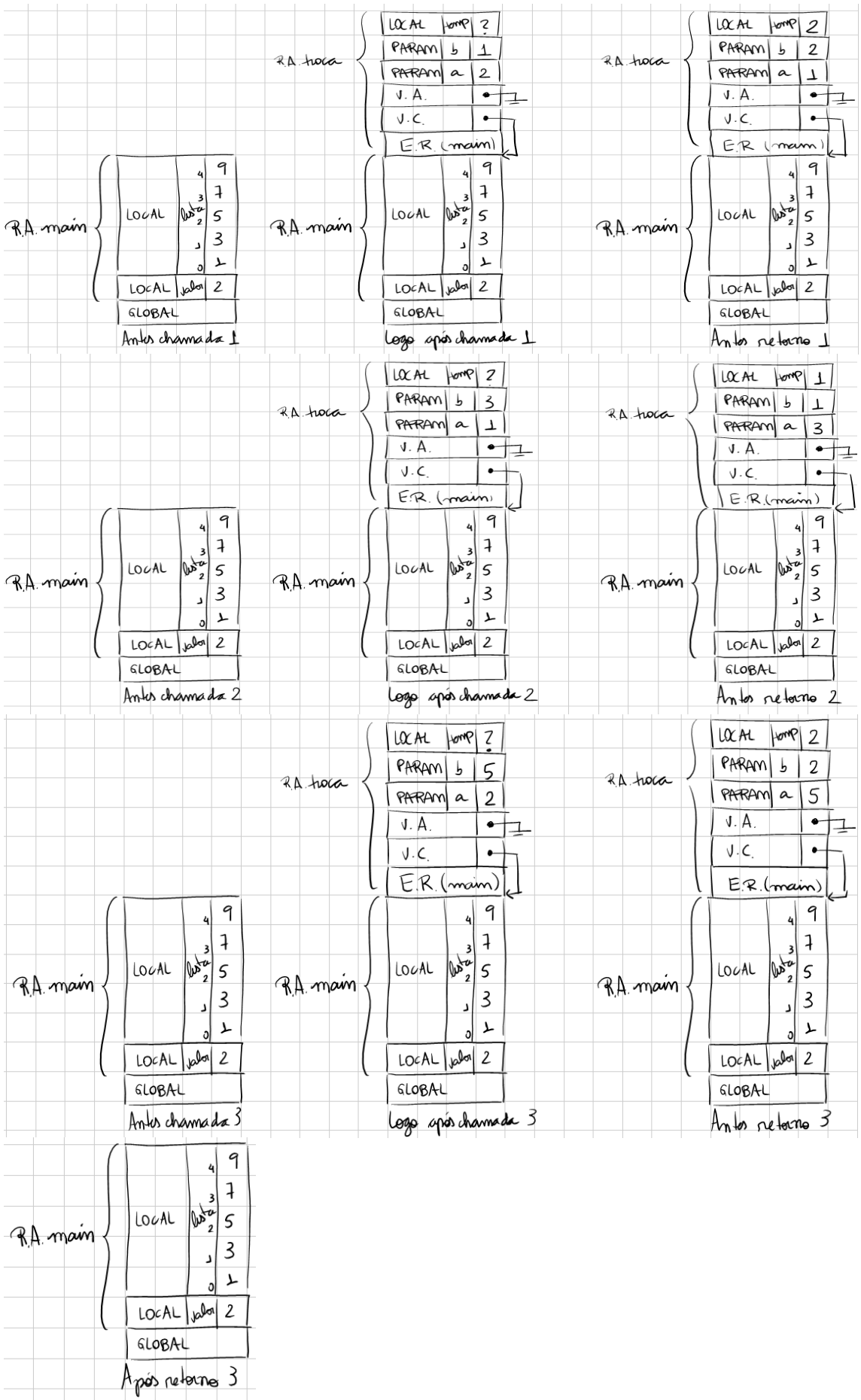
Para cada um dos métodos de passagem de parâmetros seguintes, descreva os passos para realizar essa passagem e diga quais são os valores das variáveis valor e lista antes e depois de cada uma das três chamadas a troca. Mostre o conteúdo da pilha de execução para o item "a" abaixo (passagem por valor).

a. passados por valor

```
Chamada 1:
  a = valor      // a recebe 2
  b = lista[0]   // b recebe 1
  temp = a       // temp recebe 2
  a = b          // a recebe 1
  b = temp       // b recebe 2
Após a chamada:
  valor = 2
  lista = {1,3,5,7,9}

Chamada 2:
  a = lista[0]   // a recebe 1
  b = lista[1]   // b recebe 3
  temp = a       // temp recebe 1
  a = b          // a recebe 3
  b = temp       // b recebe 1
Após a chamada:
  valor = 2
  lista = {1,3,5,7,9}

Chamada 3:
  a = valor        // a recebe 2
  b = lista[valor] // b recebe lista[2], ou seja 5
  temp = a         // temp recebe 2
  a = b            // a recebe 5
  b = temp         // b recebe 2
Após a chamada:
  valor = 2
  lista = {1,3,5,7,9}
```


R.A. main
LOCAL
lista
4 9
3 7
2 5
1 3
0 1
LOCAL valor 2
GLOBAL
Antes chamada 1
R.A. troca
LOCAL temp ?
PARAM b 1
PARAM a 2
V. A.
V. C.
E.R. (main)
Logo após chamada 1
LOCAL temp 2
PARAM b 2
PARAM a 1
Antes retorno 1
LOCAL temp 2
PARAM b 3
PARAM a 1
Antes chamada 2
LOCAL temp ?
PARAM b 3
PARAM a 1
Logo após chamada 2
LOCAL temp 1
PARAM b 1
PARAM a 3
Antes retorno 2
LOCAL temp ?
PARAM b 5
PARAM a 2
Antes chamada 3
LOCAL temp ?
PARAM b 5
PARAM a 2
Logo após chamada 3
LOCAL temp 2
PARAM b 2
PARAM a 5
Antes retorno 3
Após retorno 3

b. passados por referência

```
Chamada 1:
  a = &valor      // a "aponta" para valor
  b = &lista[0]   // b "aponta" para lista[0]
  temp = *a       // temp recebe conteúdo de valor (2)
  *a = *b         // valor recebe 1
  *b = temp       // lista[0] recebe 2
Após a chamada:
  valor = 1
  lista = {2,3,5,7,9}

Chamada 2:
  a = &lista[0]   // a "aponta" para lista[0]
  b = &lista[1]   // b "aponta" para lista[1]
  temp = *a       // temp recebe conteúdo de lista[0] (2)
  *a = *b         // lista[0] recebe conteúdo de lista[1] (3)
  *b = temp       // lista[1] recebe 2
Após a chamada:
  valor = 1
  lista = {3,2,5,7,9}

Chamada 3:
  a = &valor         // a "aponta" para valor
  b = &lista[valor]  // b "aponta" para lista[1]
  temp = *a          // temp recebe conteúdo de valor (1)
  *a = *b            // valor recebe conteúdo de lista[1] (2)
  *b = temp          // lista[1] recebe 1
Após a chamada:
  valor = 2
  lista = {3,1,5,7,9}
```

c. passados por nome

```
Chamada 1:
  a = "valor"           // a recebe o nome "valor"
  b = "lista[0]"        // b recebe o nome "lista[0]"
  temp = valor          // temp recebe 2
  valor = lista[0]      // valor recebe 1
  lista[0] = temp       // lista[0] recebe 2
Após a chamada:
  valor = 1
  lista = {2,3,5,7,9}

Chamada 2:
  a = "lista[0]"        // a recebe o nome "lista[0]"
  b = "lista[1]"        // b recebe o nome "lista[1]"
  temp = lista[0]       // temp recebe 2
  lista[0] = lista[1]   // lista[0] recebe 3
  lista[1] = temp       // lista[1] recebe 2
Após a chamada:
  valor = 1
  lista = {3,2,5,7,9}

Chamada 3:
  a = "valor"           // a recebe o nome "valor"
  b = "lista[valor]"    // b recebe o nome "lista[valor]"
  temp = valor          // temp recebe 1
  valor = lista[valor]  // valor recebe o conteúdo de lista[1] (2)
  lista[valor] = temp   // lista[2] recebe 1
Após a chamada:
  valor = 2
  lista = {3,2,1,7,9}
```

d. passados por valor-resultado

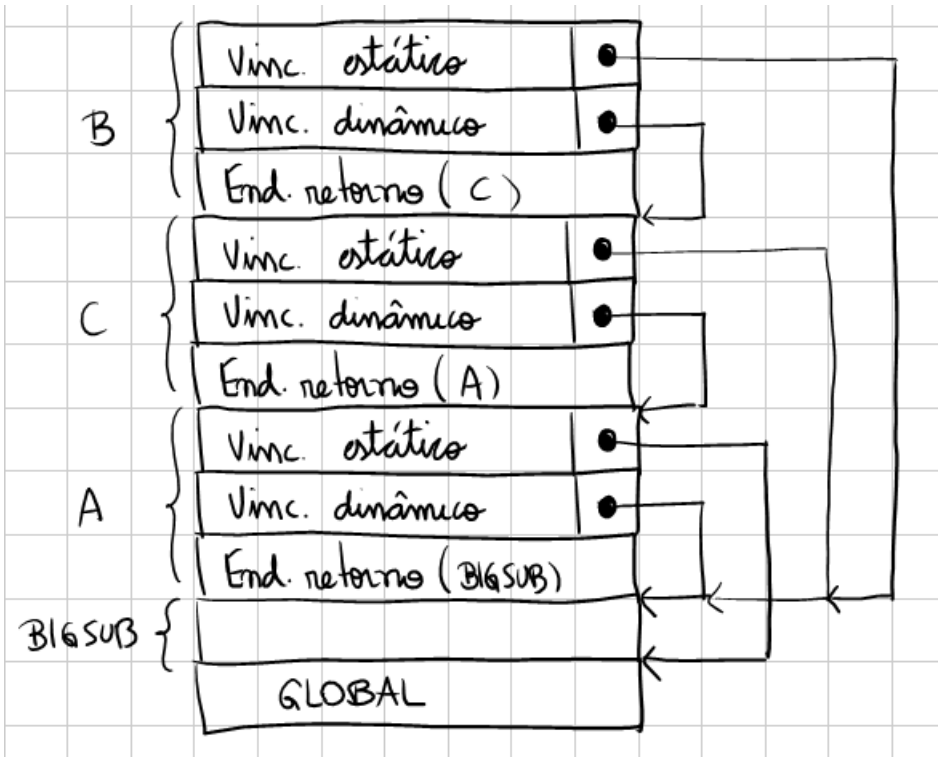
```
Chamada 1:
  end_valor = &valor          // endereço de valor é armazenado
  end_lista0 = &lista[0]      // endereço de lista[0] é armazenado
  a = *end_valor              // a recebe 2
  b = *end_lista0             // b recebe 1
  temp = a                    // temp recebe 2
  a = b                       // a recebe 1
  b = temp                    // b recebe 2
  *end_valor = a              // valor recebe 1
  *end_lista0 = b             // lista[0] recebe 2
Após a chamada:
  valor = 1
  lista = {2,3,5,7,9}

Chamada 2:
  end_lista0 = &lista[0]      // endereço de lista[0] é armazenado
  end_lista1 = &lista[1]      // endereço de lista[1] é armazenado
  a = *end_lista0             // a recebe 2
  b = *end_lista1             // b recebe 3
  temp = a                    // temp recebe 2
  a = b                       // a recebe 3
  b = temp                    // b recebe 2
  *end_lista0 = a             // lista[0] recebe 3
  *end_lista1 = b             // lista[1] recebe 2
Após a chamada:
  valor = 1
  lista = {3,2,5,7,9}

Chamada 3:
  end_valor = &valor              // endereço de valor é armazenado
  end_listaValor = &lista[valor] // endereço de lista[1] é armaz.
  a = *end_valor                  // a recebe 1
  b = *end_listaValor             // b recebe lista[1] (2)
  temp = a                        // temp recebe 1
  a = b                           // a recebe 2
  b = temp                        // b recebe 1
  *end_valor = a                  // valor recebe 2
  *end_listaValor = b             // lista[1] recebe 1
Após a chamada:
  valor = 2
  lista = {3,1,5,7,9}
```

6. Mostre a pilha com todas as instâncias do registro de ativação, incluindo encadeamentos estáticos e dinâmicos, quando a execução atingir a posição 1 no programa esquemático a seguir. Suponha que BIGSUB está no nível 1.

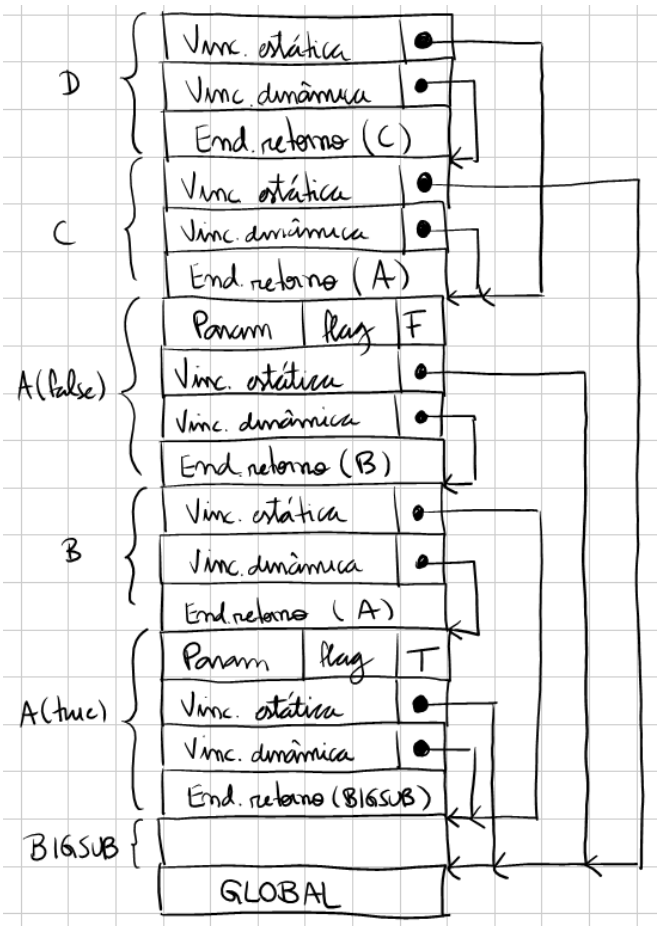
```
procedure BIGSUB;
  procedure A;
    procedure B;
      begin { B }
      ... ----------------------------------------> 1
      end; { B }
    procedure C;
      begin { C }
      ...
      B;
      ...
      end; { C }
    begin { A }
    ...
    C;
    ...
    end; { A }
  begin { BIGSUB }
  ...
  A;
  ...
  end; { BIGSUB }
```



7. Mostre a pilha com todas as instâncias do registro de ativação, incluindo encadeamentos estáticos e dinâmicos, quando a execução atingir a posição 1 no programa esquemático seguinte, escrito em linguagem Ada que, como o Pascal, permite aninhamento de subprogramas. Suponha que BIGSUB esteja no nível 1.

```
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
  ...
  D;
  end; -- fim de C
begin -- começo de BIGSUB
...
A(true);
...
end; -- fim de BIGSUB
```

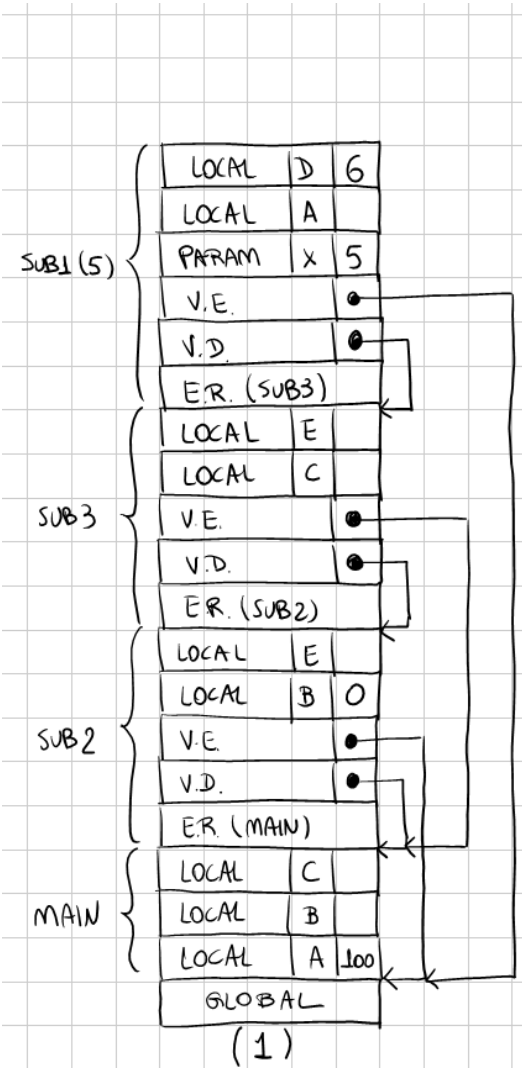
A sequência de chamada desse programa para que a execução atinja D é: BIGSUB chama A, A chama B, B chama A, A chama C, C chama D

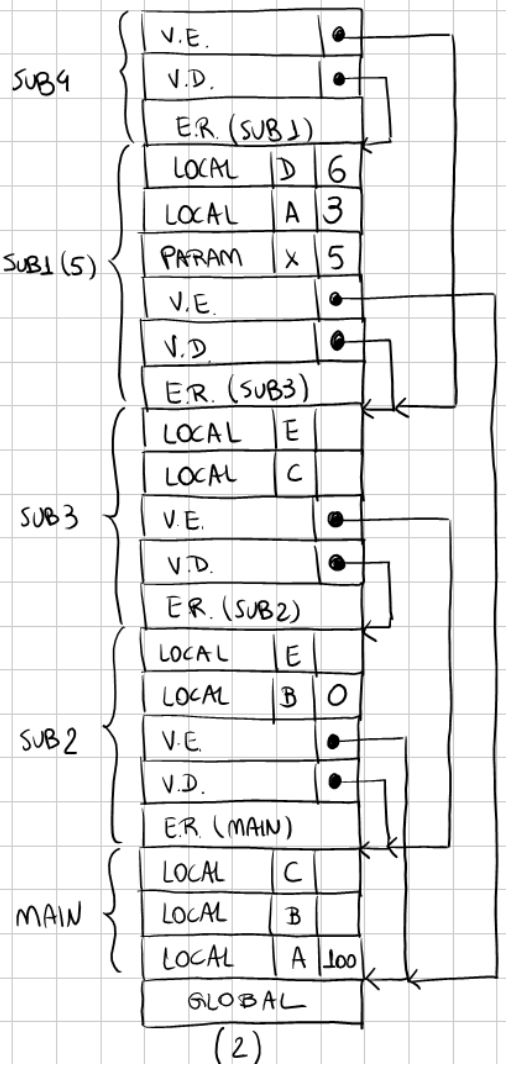


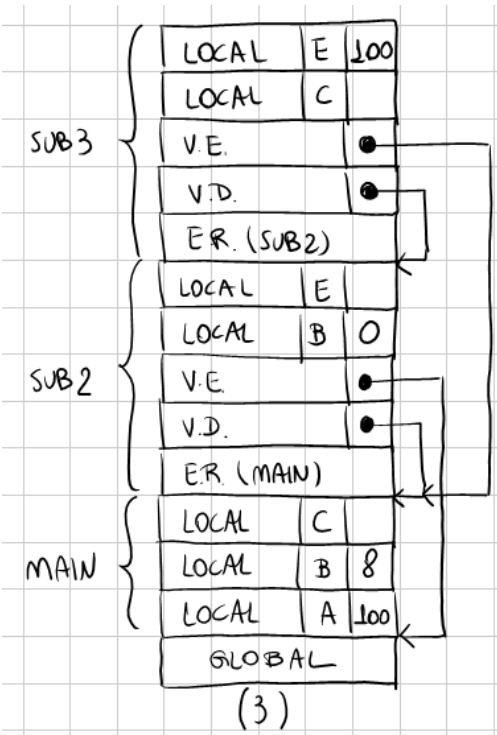
8. Para cada um dos quatro pontos indicados, (1, 2, 3 e 4) diga quais variáveis de quais procedimentos estão sendo referenciadas. Mostre a situação da pilha de execução nos pontos 1, 2, 3 e 4.

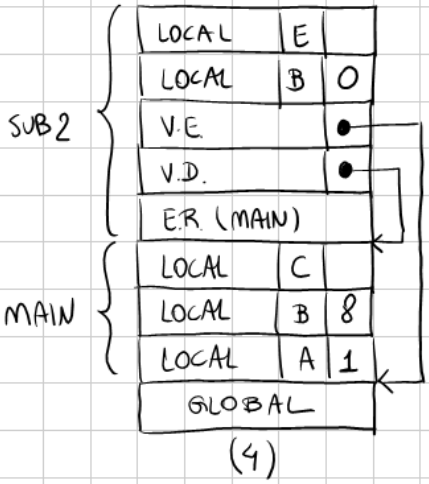
```
program MAIN;
  var A, B, C : integer;
  procedure SUB1 (X : integer);
    var A, D : integer;
    procedure SUB4;
      begin {SUB4 }
      ...
      A : = D / 2; ----------------------------------------> 2
      ...
      end; { SUB4 }
    begin { SUB1}
    ...
    D := X + 1; -----------------------------------------> 1
    ...
    SUB4;
    ...
    B : = A + X;
    ...
    end; { SUB1}
  procedure SUB2;
    var B, E : integer;
    procedure SUB3;
      var C, E : integer;
      begin { SUB3 }
      B := 0;
      SUB1( 5);
      ...
      E := B + A; -----------------------------------------> 3
      end ; {SUB3}
    begin { SUB2 }
    ...
    SUB3;
    ...
    A := B + 1; -----------------------------------------> 4
    ...
    end; { SUB2 }
  begin { MAIN }
  A := 100;
  ...
  SUB2;
  ...
  end; { MAIN }
```

```
Obs: considerando escopo estático!
Ponto 1
  D é variável local de SUB1
  X é parâmetro de SUB1
Ponto 2
  A é não-local, definida em SUB1
  D é não-local, definida em SUB1
Ponto 3:
  E é variável local de SUB3
  B é não-local, definida em SUB2
  A é não-local, definida em MAIN
Ponto 4:
  A é não-local, definida em MAIN
  B é variável local de SUB2
```


SUB1 (5)
LOCAL D 6
LOCAL A
PARAM X 5
V.E.
V.D.
E.R. (SUB3)
SUB3
LOCAL E
LOCAL C
V.E.
V.D.
E.R. (SUB2)
SUB2
LOCAL E
LOCAL B 0
V.E.
V.D.
E.R. (MAIN)
MAIN
LOCAL C
LOCAL B
LOCAL A 100
GLOBAL
(1)

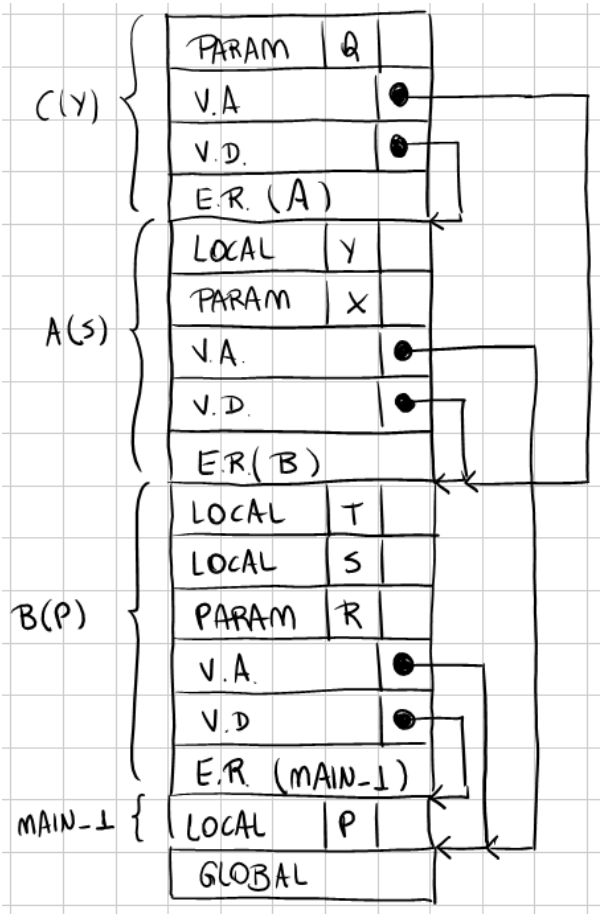

SUB4
V.E.
V.D.
E.R. (SUB1)
SUB1 (5)
LOCAL D 6
LOCAL A 3
PARAM X 5
V.E.
V.D.
E.R. (SUB3)
SUB3
LOCAL E
LOCAL C
V.E.
V.D.
E.R. (SUB2)
SUB2
LOCAL E
LOCAL B 0
V.E.
V.D.
E.R. (MAIN)
MAIN
LOCAL C
LOCAL B
LOCAL A 100
GLOBAL
(2)


SUB3
LOCAL E 100
LOCAL C
V.E.
V.D.
E.R. (SUB2)
SUB2
LOCAL E
LOCAL B 0
V.E.
V.D.
E.R. (MAIN)
MAIN
LOCAL C
LOCAL B 8
LOCAL A 100
GLOBAL
(3)


SUB2
LOCAL E
LOCAL B 0
V.E.
V.D.
E.R. (MAIN)
MAIN
LOCAL C
LOCAL B 8
LOCAL A 1
GLOBAL
(4)

9. 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.

```
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 }
```



Observações:

- Sempre que for solicitada a construção de uma pilha de execução, essa pilha deve ser completa, com todas as informações, inclusive de conteúdo das variáveis e parâmetros.
- As linguagens que permitem aninhamento de subprogramas precisam de um campo a mais no registro de ativação, para o vínculo estático. As que não permitem, não precisam ter esse campo.
- Quando o exercício não especifica qual o método de passagem de parâmetros utilizado, valem as regras da linguagem (Ex: no Pascal, parâmetros com passagem por referência são precedidos pela palavra var).