B.17 Formative Evaluation: Multiple Choice Knowledge Questionnaires of Loop Structures (provinha2b)

Catalog

SSC0600 - Introdução à Ciência de Computação I **Provinha 2(b)**
Tópico: Estruturas de Repetição e Tipos de Dados Compostos (Strings e Vetores)
8 de maio de 2017

N.º USP:

☐0 ☐0 ☐0 ☐0 ☐0 ☐0 ☐0 ☐0
☐1 ☐1 ☐1 ☐1 ☐1 ☐1 ☐1 ☐1
☐2 ☐2 ☐2 ☐2 ☐2 ☐2 ☐2 ☐2
☐3 ☐3 ☐3 ☐3 ☐3 ☐3 ☐3 ☐3
☐4 ☐4 ☐4 ☐4 ☐4 ☐4 ☐4 ☐4
☐5 ☐5 ☐5 ☐5 ☐5 ☐5 ☐5 ☐5
☐6 ☐6 ☐6 ☐6 ☐6 ☐6 ☐6 ☐6
☐7 ☐7 ☐7 ☐7 ☐7 ☐7 ☐7 ☐7
☐8 ☐8 ☐8 ☐8 ☐8 ☐8 ☐8 ☐8
☐9 ☐9 ☐9 ☐9 ☐9 ☐9 ☐9 ☐9

← Por favor codifique seu Número USP na esquerda e escreva seu nome abaixo.

Nome e sobrenome:

.....

.....

Question [`remember-unistructural`] ♣ Marque (X) nas opções que são usadas para representar as **estruturas de repetição** (*loops*) em pseudocódigo ou na Linguagem C.

- ■ while (*expressão*) { ... }
- ■ for (*expressão*; *expressão*; *expressão*) { ... }
- ■ do { ... } while (*expressão*);
- ☐ if (*expressão*) { ... }
- ☐ if (*expressão*) { ... } else { ... }
- ☐ if-not (*expressão*) { ... }
- ☐ if-not (*expressão*) { ... } else { ... }
- ☐ while (*expressão*) { ... } other-case { ... }
- ☐ do { ... } until (*expressão*) { ... }
- ■ PARA *expressão* ATÉ *expressão* PASSO *expressão* FAÇA ... FIMPARA
- ■ ENQUANTO *expressão* FAÇA ... FIMENQUANTO
- ■ REPITA ... ATÉ *expressão*
- ☐ SE *expressão* ENTÃO ...
- ☐ SE *expressão* ENTÃO ... SENÃO ...
- ☐ SENÃO *expressão* ENTÃO ...
- ☐ SENÃO *expressão* ENTÃO ... SENÃO ...
- ☐ ENQUANTO *expressão* FAÇA ... EMOUTROSCASOSFAÇA ... FIMENQUANTO
- ☐ *Nenhuma das alternativas está correta*

CATALOG

Question [`understand-relational`] ♣ Sejam os quatro tipos de pirâmides: (a) pirâmide completa, (b) pirâmide completa invertida, (c) meia pirâmide e (d) meia pirâmide invertida (Figura 1), Marque (X) nas afirmativas verdadeiras em relação as Listagem 1, 2, 3 e 4.

- *Observação*: `n` é a altura (ou número de filas) das pirâmides

- [x] Listagem 1 imprime uma meia pirâmide
- [ ] Listagem 1 imprime uma meia pirâmide invertida
- [ ] Listagem 1 imprime uma pirâmide completa
- [ ] Listagem 1 imprime uma pirâmide completa invertida
- [x] Listagem 2 imprime uma pirâmide completa invertida
- [ ] Listagem 2 imprime uma pirâmide completa
- [ ] Listagem 2 imprime uma meia pirâmide
- [ ] Listagem 2 imprime uma meia pirâmide invertida
- [x] Listagem 3 imprime uma meia pirâmide invertida
- [ ] Listagem 3 imprime uma meia pirâmide
- [ ] Listagem 3 imprime uma pirâmide completa
- [ ] Listagem 3 imprime uma pirâmide completa invertida
- [x] Listagem 4 imprime uma pirâmide completa
- [ ] Listagem 4 imprime uma pirâmide completa invertida
- [ ] Listagem 4 imprime uma meia pirâmide
- [ ] Listagem 4 imprime uma meia pirâmide invertida
- [ ] *Nenhuma das alternativas está correta*

Question [`apply-multistructural-1`] ♣ Marque (X) na saída do programa apresentado na Listagem 5.

- [x]
```
0 3 6
1 4 7
2 5 8
```
- [ ]
```
0 1 2
3 4 5
6 7 8
```
- [ ]
```
0 1 2
1 2 3
2 3 4
```
- [ ]
```
0 3 6
1 4 5
2 7 8
```
- [ ] *Nenhuma das alternativas está correta*

CATALOG

Question [apply-multistructural-2] ♣ Marque (X) nas afirmativas verdadeiras em relação ao programa da Listagem 6.

- [x] Depois que o código for executado, x contém os valores: {5, 2, 4, 3, 4, 0}
- [ ] Depois que o código for executado, x contém os valores: {5, 1, 2, 3, 4, 0}
- [ ] Depois que o código for executado, x contém os valores: {5, 1, 2, 6, 8, 0}
- [ ] Depois que o código for executado, x contém os valores: {5, 4, 4, 3, 8, 0}
- [x] Depois que o código for executado, y contém os valores: {10, 2, 4, 6, 8, 0}
- [ ] Depois que o código for executado, y contém os valores: {0, 8, 6, 2, 1, 5}
- [ ] Depois que o código for executado, y contém os valores: {0, 8, 6, 2, 2, 10}
- [ ] Depois que o código for executado, y contém os valores: {5, 2, 4, 6, 8, 0}
- [ ] *Nenhuma das alternativas está correta*

Question [evaluate-multistructural] ♣ Marque (X) nas afirmativas verdadeiras em relação ao programa da Listagem 6.

- [x] A condição `i1 > 0` da Linha 10 é avaliada 3 vezes
- [ ] A condição `i1 > 0` da Linha 10 é avaliada 2 vezes
- [ ] A condição `i1 > 0` da Linha 10 é avaliada 4 vezes
- [x] A condição `i2 < j2` da Linha 17 é avaliada 7 vezes
- [ ] A condição `i2 < j2` da Linha 17 é avaliada 5 vezes
- [ ] A condição `i2 < j2` da Linha 17 é avaliada 6 vezes
- [x] O código na estrutura de repetição externa (*outer loop*), linhas 11 até 26, é executado 2 vezes
- [ ] O código na estrutura de repetição externa (*outer loop*), linhas 11 até 26, é executado 3 vezes
- [ ] O código na estrutura de repetição externa (*outer loop*), linhas 11 até 26, é executado 4 vezes
- [x] O código na estrutura de repetição interna (*inner loop*), linhas 18 até 22, é executado 5 veces
- [ ] O código na estrutura de repetição interna (*inner loop*), linhas 18 até 22, é executado 4 vezes
- [ ] O código na estrutura de repetição interna (*inner loop*), linhas 18 até 22, é executado 6 vezes
- [ ] *Nenhuma das alternativas está correta*

Question [analyse-relational-1] ♣ Marque (X) nas modificações que, de maneira independente umas das outras, façam com que o programa apresentado na Listagem 5 imprima:

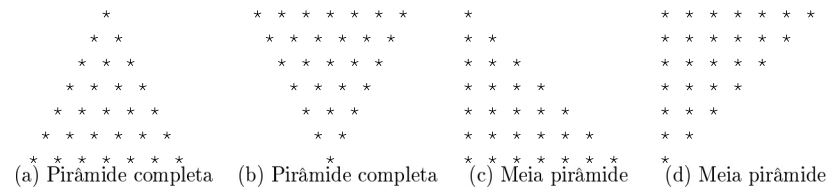
8 5 2
7 4 1
6 3 0

- [x] A linha 13 deve ser mudada para: `for (i=2; i >= 0; i--) {`
  e a linha 14 deve ser mudada para: `for (j=2; j >= 0; j--) {`
- [ ] A linha 13 deve ser mudada para: `for (i=2; i >= 0; i--) {`
- [ ] A linha 14 deve ser mudada para: `for (j=2; j >= 0; j--) {`
- [x] A linha 10 deve ser mudada para : `m[j][i] = k;`
  e a linha 13 deve ser mudada para: `for (i=2; i >= 0; i--) {`
  e a linha 14 deve ser mudada para: `for (j=2; j >= 0; j--) {`
  e a linha 15 deve ser mudada para: `printf("%d ", m[j][i]);`
- [ ] A linha 10 deve ser mudada para: `m[j][i] = k;`
- [ ] A linha 15 deve ser mudada para: `printf("%d ", m[j][i]);`
- [x] A linha 10 deve ser mudada para: `m[2-i][2-j] = k;`
- [ ] A linha 10 deve ser mudada para: `m[2-j][2-i] = k;`
- [ ] *Nenhuma das alternativas está correta*

CATALOG

Question [analyse-relational-2] ♣ O trecho de código apresentado na Listagem 7 tem sido proposto para efetuar a ordenação descendente (de maior a menor) de um vetor de inteiros `arr[n]` de tamanho *n*. No entanto, o código não funciona adequadamente. Marque (X) nas modificações necessárias que, em conjunto, façam o programa funcionar adequadamente.

- [x] A linha 1 deve ser mudada para: `for (i = 0 ; i < n-1 ; i++) {`
- [ ] A linha 1 deve ser mudada para: `for (i = 1 ; i < n-1 ; i++) {`
- [ ] A linha 1 deve ser mudada para: `for (i = n-2 ; i >= 0; i--) {`
- [x] A linha 2 deve ser mudada para: `j = i+1;`
- [ ] A linha 2 deve ser mudada para: `j = i-1;`
- [x] A linha 3 deve ser mudada para: `while (j > 0 && arr[j] > arr[j-1]) {`
- [ ] A linha 3 deve ser mudada para: `while (j > 0 && arr[j] < arr[j-1]) {`
- [ ] A linha 3 deve ser mudada para: `while (j > i && arr[j] > arr[j-1]) {`
- [ ] *Nenhuma das alternativas está correta*



(a) Pirâmide completa (b) Pirâmide completa invertida (c) Meia pirâmide (d) Meia pirâmide invertida

Figura 1: Exemplos de tipos de pirâmides com altura n=7

```
1 for (i=0 ; i < n ; i++) {
2     for (j=0 ; j <= i ; j++) {
3         printf("* ");
4     }
5     printf("\n");
6 }
```

Listagem 1: Trecho de código para imprimir uma pirâmide

```
1 for (i=0; i < n; i++) {
2     for (k=0; k < i; k++) {
3         printf(" ");
4     }
5     for (k=0 ; k < n-i; k++) {
6         printf("* ");
7     }
8     printf("\n");
9 }
```

Listagem 2: Trecho de código para imprimir uma pirâmide

```
1 for (i=n ; i > 0 ; i--) {
2     for (j=0 ; j < i ; j++) {
3         printf("* ");
4     }
5     printf("\n");
6 }
```

Listagem 3: Trecho de código para imprimir uma pirâmide

```
1 for (i=0; i < n; i++) {
2     for (k=0; k < (n-i)-1; k++) {
3         printf(" ");
4     }
5     for (k=0 ; k < i+1; k++) {
6         printf("* ");
7     }
8     printf("\n");
9 }
```

Listagem 4: Trecho de código para imprimir uma pirâmide

```c
#include <stdio.h>

int m[3][3];
int row, col, i, j, k;

int main() {
    for (k = 0; k < 9; k++) {
        i = k % 3;
        j = k / 3;
        m[i][j] = k;
    }

    for (i=0; i < 3; i++) {
        for (j=0; j < 3; j++) {
            printf("%d ", m[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Listagem 5: Código de programa na linguagem C

```c
#include <stdio.h>

int x[6] = {5, 4, 3, 2, 1, 0};
int y[6] = {0, 1, 2, 3, 4, 5};

int i1 = 2, j1 = 3;
int i2, j2, temp;

int main() {
    while (i1 > 0) {
        temp = x[i1];
        x[i1] = x[j1] * 2;
        x[j1] = temp;

        i2 = i1 - 1;
        j2 = j1 + 1;
        while (i2 < j2) {
            temp = y[j2];
            y[j2] = y[i2];
            y[i2] = 2 * temp;
            i2++;
            j2--;
        }

        i1--;
        j1++;
    }
    return 0;
}
```

Listagem 6: Código de programa na linguagem C

```
for (i = 1 ; i < n-1 ; i++) {
    j = i;
    while (j > 0 && arr[j-1] > arr[j]) {
        aux = arr[j];
        arr[j] = arr[j-1];
        arr[j-1] = aux;
        j--;
    }
}
```

Listagem 7: Trecho de código para ordenar um vetor de inteiros `arr[n]`