SSC0600 - Introdução à Ciência de Computação I **Provinha 3(b)**
Tópico: Recursão **13 de junho 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-multistructural`] ♣ (1 ponto) Em relação ao trecho de código em Linguagem C apresentado na Listagem 1, Marque (X) nas afirmativas verdadeiras

- [x] `zoo0` é recursivo
- [x] `zoo1` é recursivo
- [x] `zoo2` é recursivo
- [x] `zoo3` é recursivo
- [x] `zoo4` é recursivo
- [ ] `zoo5` é recursivo
- [x] `zoo6` é recursivo
- [ ] `zoo0` não é recursivo
- [ ] `zoo1` não é recursivo
- [ ] `zoo2` não é recursivo
- [ ] `zoo3` não é recursivo
- [ ] `zoo4` não é recursivo
- [x] `zoo5` não é recursivo
- [ ] `zoo6` não é recursivo
- [ ] *Nenhuma das alternativas está correta*

Question [understand-multistructural] ♣ (3 pontos) Marque (X) nas afirmativas verdadeiras em relação as funções da Listagem 2

Observações:

- Os quadrados dos `n` primeiros números são: 1^2, 2^2, 3^2, ... , n^2

- [x] `foo1` calcula o fatorial de n
- [ ] `foo1` calcula o fatorial de $n-1$
- [x] `foo1` retorna o valor de 1 quando n é menor ou igual a 1 e retorna o produto de todos os inteiros positivos menores ou iguais a n em outros casos
- [ ] `foo1` retorna o valor de 1 quando n é menor ou igual a 1 e retorna o produto de todos os inteiros positivos menores ou iguais a $n-1$ em outros casos
- [x] `foo1` é a função que calcula $1 * 2 * ... * (n-1) * n$
- [ ] `foo1` é a função que calcula $1 * 2 * ... * (n-2) * (n-1)$
- [x] `foo2` calcula a soma dos quadrados dos n primeiros números
- [ ] `foo2` calcula a soma dos quadrados dos $n-1$ primeiros números
- [x] `foo2` é a função que calcula $(n * n) + [(n-1) * (n-1)] + [(n-2) * (n-2)] + ... + 1$
- [ ] `foo2` é a função que calcula $[(n-1) * (n-1)] + [(n-2) * (n-2)] + ... + 1$
- [x] `foo2` é a função que calcula $1^2 + 2^2 + 3^2 + ... + n^2$
- [ ] `foo2` é a função que calcula $1^2 + 2^2 + 3^2 + ... + (n-1)^2$
- [x] Se n é impar então `foo3` calcula a soma da sequência: $\frac{2}{foo2(1)}$, $\frac{1}{foo1(2)}$, ..., $\frac{1}{foo1(n-1)}$, $\frac{2}{foo2(n)}$
- [x] Se n é par então `foo3` calcula a soma da sequência: $\frac{2}{foo2(1)}$, $\frac{1}{foo1(2)}$, ..., $\frac{2}{foo2(n-1)}$, $\frac{1}{foo1(n)}$
- [ ] Se n é impar então `foo3` calcula a soma da sequência: $\frac{2}{foo2(1)}$, $\frac{1}{foo1(2)}$, ..., $\frac{2}{foo2(n-1)}$, $\frac{1}{foo1(n)}$
- [ ] Se n é par então `foo3` calcula a soma da sequência: $\frac{2}{foo2(1)}$, $\frac{1}{foo1(2)}$, ..., $\frac{1}{foo1(n-1)}$, $\frac{2}{foo2(n)}$
- [ ] Se n é impar então `foo3` calcula a soma da sequência: $\frac{1}{foo1(1)}$, $\frac{2}{foo2(2)}$, ..., $\frac{2}{foo2(n-1)}$, $\frac{1}{foo1(n)}$
- [ ] Se n é par então `foo3` calcula a soma da sequência: $\frac{1}{foo1(1)}$, $\frac{2}{foo2(2)}$, ..., $\frac{1}{foo1(n-1)}$, $\frac{2}{foo2(n)}$
- [ ] Se n é impar então `foo3` calcula a soma da sequência: $\frac{1}{foo1(1)}$, $\frac{2}{foo2(2)}$, ..., $\frac{2}{foo2(n-1)}$, $\frac{1}{foo1(n)}$
- [ ] Se n é par então `foo3` calcula a soma da sequência: $\frac{1}{foo1(1)}$, $\frac{2}{foo2(2)}$, ..., $\frac{2}{foo2(n-1)}$, $\frac{1}{foo1(n)}$
- [ ] *Nenhuma das alternativas está correta*

Question [apply-unistructural] ♣ (1.5 ponto) Seja o vetor `v[9]={3, 2, 4, 6, 8, 6, 7, 8, 1}`.
Marque (X) nas afirmativas verdadeiras em relação à função `foo` da Listagem 3

- [x] retorna 2 quando i é 0 e j é 3
- [ ] retorna 4 quando i é 0 e j é 3
- [x] retorna 1 quando i é 5 e j é 8
- [ ] retorna 7 quando i é 5 e j é 8
- [ ] retorna 3 quando i é 0 e j é 3
- [ ] retorna 6 quando i é 0 e j é 3
- [ ] retorna 8 quando i é 5 e j é 8
- [ ] retorna 6 quando i é 5 e j é 8
- [ ] *Nenhuma das alternativas está correta*

Question [apply-relational] ♣ (2 pontos) Marque (X) nas afirmativas verdadeiras em relação ao programa da Listagem 4.

- [x] Depois que o código for executado, *v1* contém os valores: {1, 3, 5, 6, 7, 2, 4, 6, 8, 0}
- [ ] Depois que o código for executado, *v1* contém os valores: {1, 3, 5, 5, 5, 2, 4, 6, 8, 0}
- [ ] Depois que o código for executado, *v1* contém os valores: {1, 3, 1, 1, 1, 2, 4, 6, 8, 0}
- [ ] Depois que o código for executado, *v1* contém os valores: {1, 3, 1, 2, 3, 2, 4, 6, 8, 0}
- [ ] Depois que o código for executado, *v1* contém os valores: {1, 3, 5, 8, 0, 2, 4, 6, 7, 8}
- [ ] Depois que o código for executado, *v1* contém os valores: {1, 3, 5, 8, 0, 2, 4, 4, 5, 6}
- [x] Depois que o código for executado, *v2* contém os valores: {4, 6, 2, -3, 0, 2, 3, 8, 9, 10}
- [ ] Depois que o código for executado, *v2* contém os valores: {4, 6, 2, -3, 0, 2, 3, 8, 8, 8}
- [ ] Depois que o código for executado, *v2* contém os valores: {4, 6, 2, -3, 0, 2, 3, 3, 3, 3}
- [ ] Depois que o código for executado, *v2* contém os valores: {4, 6, 2, -3, 0, 2, 3, 3, 4, 5}
- [ ] Depois que o código for executado, *v2* contém os valores: {4, 6, 6, 7, 8, 2, 3, 8, 4, 9}
- [ ] Depois que o código for executado, *v2* contém os valores: {4, 6, 2, 3, 4, 2, 3, 8, 4, 9}
- [ ] *Nenhuma das alternativas está correta*

Question [evaluate-multistructural] ♣ (1 ponto) Marque (X) nas afirmativas verdadeiras em relação ao programa da Listagem 4.

- [x] A chamada para a função `bar` é efetuada 5 vezes
- [ ] A chamada para a função `bar` é efetuada 6 vezes
- [ ] A chamada para a função `bar` é efetuada 7 vezes
- [ ] A chamada para a função `bar` é efetuada 8 vezes
- [ ] A chamada para a função `bar` é efetuada 9 vezes
- [x] A chamada para a função `foo` é efetuada 8 vezes
- [ ] A chamada para a função `foo` é efetuada 7 vezes
- [ ] A chamada para a função `foo` é efetuada 9 vezes
- [ ] A chamada para a função `foo` é efetuada 6 vezes
- [ ] A chamada para a função `foo` é efetuada 5 vezes
- [ ] *Nenhuma das alternativas está correta*

Question [analyse-relational-1] ♣ (2 pontos) Marque (X) nas modificações que, de maneira independente umas das outras, façam com que a função `is_prime` apresentada na Listagem 5 calcule se um número `n` é primo. A função retorna 1 se `n` é primo e a função retorna 0 se `n` não é primo.

- [x] A linha 2 deve ser mudada para: `if (div*div > n)`
  a linha 5 deve ser mudada para: `if (n % div == 0)`
  a linha 8 deve ser mudada para: `return is_prime_t(n, div+1);`
- [ ] A linha 2 deve ser mudada para: `if (div*div >= n)`
  a linha 5 deve ser mudada para: `if (n % div == 0)`
  a linha 8 deve ser mudada para: `return is_prime_t(n, div+1);`
- [ ] A linha 2 deve ser mudada para: `if (div*div > n)`
  a linha 5 deve ser mudada para: `if (n / div == 0)`
  a linha 8 deve ser mudada para: `return is_prime_t(n, div+1);`
- [ ] A linha 2 deve ser mudada para: `if (div*div >= n)`
  a linha 5 deve ser mudada para: `if (n / div == 0)`
  a linha 8 deve ser mudada para: `return is_prime_t(n, div+1);`
- [x] A linha 2 deve ser mudada para: `if (div >= n)`
  a linha 5 deve ser mudada para: `if (n % div == 0)`
  a linha 8 deve ser mudada para: `return is_prime_t(n, div+1);`
- [ ] A linha 2 deve ser mudada para: `if (div > n)`
  a linha 5 deve ser mudada para: `if (n % div == 0)`
  a linha 8 deve ser mudada para: `return is_prime_t(n, div+1);`
- [ ] A linha 2 deve ser mudada para: `if (div >= n)`
  a linha 5 deve ser mudada para: `if (n / div == 0)`
  a linha 8 deve ser mudada para: `return is_prime_t(n, div+1);`
- [ ] A linha 2 deve ser mudada para: `if (div > n)`
  a linha 5 deve ser mudada para: `if (n / div == 0)`
  a linha 8 deve ser mudada para: `return is_prime_t(n, div+1);`
- [ ] *Nenhuma das alternativas está correta*

Question [analyse-relational-2] ♣ (2 pontos) O método `search` apresentado na Listagem 6 tem sido proposto para efetuar a busca de um elemento `v` em uma `matriz` de tamanho $n * m$ (linhas x colunas). Os elementos da matriz sempre estão em ordem ascendente ou descendente (sempre de esquerda para direita, e logo de acima para abaixo). Assim, o algoritmo funciona para ambos casos. Marque (X) nas modificações que, de maneira independente umas das outras, façam o método `search` funcionar adequadamente.

Exemplo para `matriz[3][4]` (ascendente):

10 11 12 13
14 15 16 17
18 19 20 22

Se $v = 12$, a posição é (0,2)
Se $v = 19$, a posição é (2,1)

Exemplo para `matriz[4][3]` (descendente):

22 21 20
19 17 16
15 14 13
12 11 10

Se $v = 10$, a posição é (3,2)
Se $v = 20$, a posição é (0,2)

- [x] A linha 9 deve ser mudada para: `if (matriz[i/m][i%m] < matriz[j/m][j%m])`
  a linha 10 deve ser mudada para: `if (matriz[row][col] > v)`
  a linha 15 deve ser mudada para: `if (matriz[row][col] < v)`
- [ ] A linha 9 deve ser mudada para: `if (matriz[i/m][i%m] < matriz[j/m][j%m])`
  a linha 10 deve ser mudada para: `if (matriz[row][col] < v)`
  a linha 15 deve ser mudada para: `if (matriz[row][col] > v)`
- [ ] A linha 9 deve ser mudada para: `if (matriz[i/m][i%m] < matriz[j/m][j%m])`
  a linha 10 deve ser mudada para: `if (matriz[row][col] < v)`
  a linha 15 deve ser mudada para: `if (matriz[row][col] < v)`
- [ ] A linha 9 deve ser mudada para: `if (matriz[i/m][i%m] < matriz[j/m][j%m])`
  a linha 10 deve ser mudada para: `if (matriz[row][col] > v)`
  a linha 15 deve ser mudada para: `if (matriz[row][col] > v)`
- [x] A linha 9 deve ser mudada para: `if (matriz[i/m][i%m] > matriz[j/m][j%m])`
  a linha 10 deve ser mudada para: `if (matriz[row][col] < v)`
  a linha 15 deve ser mudada para: `if (matriz[row][col] > v)`
- [ ] A linha 9 deve ser mudada para: `if (matriz[i/m][i%m] > matriz[j/m][j%m])`
  a linha 10 deve ser mudada para: `if (matriz[row][col] > v)`
  a linha 15 deve ser mudada para: `if (matriz[row][col] < v)`
- [ ] A linha 9 deve ser mudada para: `if (matriz[i/m][i%m] > matriz[j/m][j%m])`
  a linha 10 deve ser mudada para: `if (matriz[row][col] < v)`
  a linha 15 deve ser mudada para: `if (matriz[row][col] < v)`
- [ ] A linha 9 deve ser mudada para: `if (matriz[i/m][i%m] > matriz[j/m][j%m])`
  a linha 10 deve ser mudada para: `if (matriz[row][col] > v)`
  a linha 15 deve ser mudada para: `if (matriz[row][col] > v)`
- [ ] *Nenhuma das alternativas está correta*

```c
int zoo6(int a, int b) {
    if (b < a) {
        return zoo6(a--, b++);
    } else {
        return a+b;
    }
}

int zoo5(int a, int b) {
    if (a > b) {
        return zoo6(a+a, b*b);
    } else {
        return zoo6(a/2, b/2);
    }
}

void zoo4(int a) {
    ...
    int c = zoo5(a, a*a);
    zoo1(a, c);
}

int zoo3(int a, int b) {
    ...
    zoo1(a-1, b+1);
    ...
    if (a > b) {
        return a;
    } else {
        return zoo2(b);
    }
}

int zoo2(int b) {
    return zoo3(zoo4(b), b+1);
}

void zoo1(int a, int b) {
    if (a > b) {
        printf("%d", zoo2(b));
    } else {
        printf("%d", zoo3(a, b));
    }
}

int zoo0(int a, int b) {
    if (a != 0) {
        zoo1(a*a, b*b);
        return zoo0(a%b, a+b);
    } else return -1;
}
```

Listagem 1: Trecho de código para as funções `foo` na Linguagem C

```c
int foo1t(int n, int resp) {
    if (n <= 1) {
        return resp;
    } else {
        return foo1(n-1, n*resp);
    }
}

int foo1(int n) {
    return foo1t(n, 1);
}

int foo2(int n) {
    if (n < 1) {
        return 0;
    } else {
        return (n*n) + foo2(n-1);
    }
}

int foo3t(int n, int resp) {
    if (n < 1) {
        return resp;
    } else {
        if (n % 2 == 0) {
            return foo3t(n-1, resp + 1/foo1(n));
        } else {
            return foo3t(n-1, resp + 2/foo2(n));
        }
    }
}

float foo3(int n) {
    return foo3t(n,  0);
}
```

Listagem 2: Trecho de código para as funções `foo` na Linguagem C

```c
int foo(int v[], int i, int j) {
    if (i >= j) {
        return v[i];
    } else {
      int x = foo(v, i+1, j);
      if (x < v[i]) {
        return x;
      } else {
        return v[i];
      }
   }
}
```

Listagem 3: Função `foo` na Linguagem C

```c
#include <stdio.h>

int bar(int v[], int, int);
void foo(int v[], int, int, int);
void foobar(int v[], int, int);

void foo(int v[], int value, int i, int j) {
    if (i <= j) {
        v[i] = value;
        foo(v, value+1, i+1, j);
    }
}

int bar(int v[], int i, int j) {
    if (i >= j) {
        return v[i];
    } else {
        int x = bar(v, i+1, j);
        if (x > v[i]) {
            return x;
        } else {
            return v[i];
        }
    }
}

void foobar(int v[], int i, int j) {
    int m = (i + j)/2;
    int x = bar(v, i, m);
    foo(v, x, m, j);
}

int main (void) {
    int v1[10] = {1, 3, 5, 8, 0, 2, 4, 6, 8, 0};
    int v2[10] = {4, 6, 2, -3, 0, 2, 3, 8, 4, 9};

    foobar(v1, 0, 4);
    foobar(v2, 6, 9);

    return 0;
}
```

Listagem 4: Código de programa na Linguagem C

```c
int is_prime_t(int n, div) {
    if (div == n)
        return 1;
    else {
        if (n % div != 0)
            return 0;
        else
            return is_prime_t(n, div+2);
    }
}

int is_prime(int n) {
    if (n <= 1) return 0;
    else if (n <= 3) return 1;
    else return is_prime_t(n, 2);
}
```

Listagem 5: Função que calcula se um número `n` é primo

```c
int search_aux(int matriz[][100], int m, int v, int i, int j) {
    if (i > j) return -1;
    int pos = (i+j)/2;
    int row = pos / m;
    int col = pos % m;
    if (matriz[row][col] == v) {
        return pos;
    } else {
        if (matriz[i/m][i%m] != matriz[j/m][j%m])
            if (matriz[row][col] <= v)
                return search_aux(matriz, m, v, i, pos-1);
            else
                return search_aux(matriz, m, v, pos+1, j);
        else
            if (matriz[row][col] >= v)
                return search_aux(matriz, m, v, i, pos-1);
            else
                return search_aux(matriz, m, v, pos+1, j);
    }
}

void search(int matriz[][100], int n, int m, int v) {
    int pos = search_aux(matriz, m, v, 0, (n*m)-1);
    if (pos < 0) printf ("Elemento %d não achado\n", v);
    else {
         printf("linha[0..n-1]: %d\n", pos/m);
         printf("coluna[0..m-1]: %d\n", pos%m);
    }
}
```

Listagem 6: `search` imprime a posição do elemento `v` na `matriz` de tamanho $n * m$ (o método funciona para matrizes com elementos ordenados de maneira ascendente ou com elementos ordenados de maneira descendente)