SSC0600 - Introdução à Ciência de Computação I **Provinha 3(a)**
Tópico: Recursão **25 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-multistructural`] ♣ Em relação ao trecho de código em Linguagem C apresentado na Listagem 1, Marque (X) nas afirmativas verdadeiras

- ☐ `foo` é uma função recursiva
- ■ `foo1` é uma função recursiva
- ☐ `foo2` é uma função recursiva
- ■ `foo3` é uma função recursiva
- ■ `foo4` é uma função recursiva
- ■ `foo5` é uma função recursiva
- ■ `foo6` é uma função recursiva
- ■ `foo` não é uma função recursiva
- ☐ `foo1` não é uma função recursiva
- ■ `foo2` não é uma função recursiva
- ☐ `foo3` não é uma função recursiva
- ☐ `foo4` não é uma função recursiva
- ☐ `foo5` não é uma função recursiva
- ☐ `foo6` não é uma função recursiva
- ☐ *Nenhuma das alternativas está correta*

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

Observações:

- `n` e `b` são inteiros positivos maiores que 0 para as funções `foobar1`, `foobar2` e `foobar3`.
- Os múltiplos de 3 são: $0 * 3$, $1 * 3$, $2 * 3$, $3 * 3$, ...

- [x] `foobar1` calcula a soma do n-ésimo termino da sequência fibonacci e o valor da variável `b`
- [ ] `foobar1` calcula o n-ésimo termino da sequência fibonacci
- [ ] `foobar1` retorna o valor da variável `b` incrementado em 1 quando `n` é 1 e ela retorna a soma do resultados das funções `foobar(n-1, b)` e `foobar(n-2, b)` em outros casos
- [ ] `foobar1` é a função que retorna o valor da variável `b` quando `n` é menor que 2 e retorna a soma do resultado da função `foobar(n-1, b)` e `foobar(n-2, b)` em outros casos
- [x] `foobar1` é a função que calcula a soma dos n+1 primeiros múltiplos de b
- [ ] `foobar1` é a função que calcula a soma dos n primeiros múltiplos de b
- [ ] `foobar1` é a função que calcula a soma dos n-1 primeiros múltiplos de b
- [x] `foobar2` é a função que calcula $b * 0 + b * 1 + b * 2 + ... + b * n$
- [ ] `foobar2` é a função que calcula $b * 0 + b * 1 + b * 2 + ... + b * n + b * (n + 1)$
- [ ] `foobar2` é a função que calcula $b * 0 + b * 1 + b * 2 + ... + b * (n - 2) + b * (n - 1)$
- [x] `foobar2` é a função que calcula o n+1 ésimo múltiplo de b
- [ ] `foobar2` é a função que calcula o n ésimo múltiplo de b
- [ ] `foobar2` é a função que calcula o n-1 ésimo múltiplo de b
- [x] Se n é impar então `foobar3` é a função que calcula a soma da sequência: $\frac{n}{foobar1(n,b)}$, $foobar2(n-1,b)$, ..., $foobar2(2,b)$, $\frac{1}{foobar1(1,b)}$; e se n é par então `foobar3` é a função que calcula a soma da sequência: $foobar2(n,b)$, $\frac{n-1}{foobar1(n-1,b)}$, ..., $foobar2(2,b)$, $\frac{1}{foobar1(1,b)}$
- [ ] Se n é impar então `foobar3` é a função que calcula a soma da sequência: $foobar2(n,b)$, $\frac{n-1}{foobar1(n-1,b)}$, ..., $foobar2(2,b)$, $\frac{1}{foobar1(1,b)}$; e se n é par então `foobar3` é a função que calcula a soma da sequência: $\frac{n}{foobar1(n,b)}$, $foobar2(n-1,b)$, ..., $foobar2(2,b)$, $\frac{1}{foobar1(1,b)}$
- [ ] Se n é par então `foobar3` é a função que calcula a soma da sequência: $\frac{n}{foobar1(n,b)}$, $foobar2(n-1,b)$, ..., $foobar2(2,b)$, $\frac{1}{foobar1(1,b)}$; e se n é impar então `foobar3` é a função que calcula a soma da sequência: $foobar2(n,b)$, $\frac{n-1}{foobar1(n-1,b)}$, ..., $foobar2(2,b)$, $\frac{1}{foobar1(1,b)}$
- [ ] *Nenhuma das alternativas está correta*

Question [`apply-unistructural`] ♣ Marque (X) nas afirmativas verdadeiras em relação à função `zoo` da Listagem 3

- [x] retorna 13 quando n é 17
- [ ] retorna 11 quando n é 17
- [ ] retorna 16 quando n é 17
- [x] retorna 16 quando n é 22
- [ ] retorna 15 quando n é 22
- [ ] retorna 12 quando n é 22
- [ ] retorna 12 quando n é 17
- [ ] retorna 15 quando n é 17
- [ ] retorna 17 quando n é 17
- [ ] retorna 17 quando n é 22
- [ ] retorna 11 quando n é 22
- [ ] retorna 13 quando n é 22
- [ ] *Nenhuma das alternativas está correta*

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

- [x] Depois que o código for executado, *v1* contém os valores: {5, 6, 8, 7, 9, 10, 4, 3, 2, 1}
- [ ] Depois que o código for executado, *v1* contém os valores: {5, 6, 7, 8, 4, 3, 2, 1}
- [ ] Depois que o código for executado, *v1* contém os valores: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- [ ] Depois que o código for executado, *v1* contém os valores: {1, 2, 3, 4, 10, 9, 7, 8, 6, 5}
- [ ] Depois que o código for executado, *v1* contém os valores: {1, 2, 3, 4, 8, 7, 6, 5}
- [ ] Depois que o código for executado, *v1* contém os valores: {10, 9, 8, 7, 6, 5, 4, 3, 2, 1}
- [x] Depois que o código for executado, *v2* contém os valores: {4, 65, 2, -31, 0, 1, 2, 83, 2, 99}
- [ ] Depois que o código for executado, *v2* contém os valores: {4, 65, 2, -31, 0, 1, 2, 2, 83, 99}
- [ ] Depois que o código for executado, *v2* contém os valores: {0, -31, 2, 1, 2, 2, 99, 4, 83, 65}
- [ ] Depois que o código for executado, *v2* contém os valores: {99, 2, 83, 2, 1, 0, -31, 2, 65, 4}
- [ ] Depois que o código for executado, *v2* contém os valores: {99, 83, 2, 2, 1, 0, -31, 2, 65, 4}
- [ ] Depois que o código for executado, *v2* contém os valores: {65, 83, 4, 99, 2, 2, 1, 2, -31, 0}
- [ ] *Nenhuma das alternativas está correta*

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

- [x] A chamada para a função `foo` é efetuada 24 vezes
- [ ] A chamada para a função `foo` é efetuada 23 vezes
- [ ] A chamada para a função `foo` é efetuada 22 vezes
- [ ] A chamada para a função `foo` é efetuada 21 vezes
- [ ] A chamada para a função `foo` é efetuada 20 vezes
- [x] A chamada para a função `bar` é efetuada 20 vezes
- [ ] A chamada para a função `bar` é efetuada 21 vezes
- [ ] A chamada para a função `bar` é efetuada 22 vezes
- [ ] A chamada para a função `bar` é efetuada 23 vezes
- [ ] A chamada para a função `bar` é efetuada 24 vezes
- [ ] *Nenhuma das alternativas está correta*

Question [analyse-relational-1] ♣ Marque (X) nas modificações que, de maneira independente umas das outras, façam com que a função `max_div_comum` apresentada na Listagem 5 calcule o máximo divisor comum de dois números `n1` e `n2` (maiores que 0).

- [x] A linha 2 deve ser mudada para: `if (n2 == 0)`
  a linha 3 deve ser mudada para: `return n1;`
  a linha 5 deve ser mudada para: `return max_div_comum(n2, n1%n2);`
- [ ] A linha 2 deve ser mudada para: `if (n2 == 0)`
  a linha 3 deve ser mudada para: `return n1;`
  a linha 5 deve ser mudada para: `return max_div_comum(n2, n2%n1);`
- [ ] A linha 2 deve ser mudada para: `if (n2 == 0)`
  a linha 3 deve ser mudada para: `return n1;`
  a linha 5 deve ser mudada para: `return max_div_comum(n2, n1/n2);`
- [ ] A linha 2 deve ser mudada para: `if (n2 == 0)`
  a linha 3 deve ser mudada para: `return n1;`
  a linha 5 deve ser mudada para: `return max_div_comum(n2, n2/n1);`
- [x] A linha 2 deve ser mudada para: `if (n1 == 0)`
  a linha 3 deve ser mudada para: `return n2;`
  a linha 5 deve ser mudada para: `return max_div_comum(n2%n1, n1);`
- [ ] A linha 2 deve ser mudada para: `if (n1 == 0)`
  a linha 3 deve ser mudada para: `return n2;`
  a linha 5 deve ser mudada para: `return max_div_comum(n1%n2, n1);`
- [ ] A linha 2 deve ser mudada para: `if (n1 == 0)`
  a linha 3 deve ser mudada para: `return n2;`
  a linha 5 deve ser mudada para: `return max_div_comum(n2/n1, n1);`
- [ ] A linha 2 deve ser mudada para: `if (n1 == 0)`
  a linha 3 deve ser mudada para: `return n2;`
  a linha 5 deve ser mudada para: `return max_div_comum(n1/n2, n1);`
- [ ] *Nenhuma das alternativas está correta*

Question [analyse-relational-2] ♣ A função `count` apresentada na Listagem 6 tem sido proposta para efetuar a contagem do número de vezes que um elemento `e` aparece num vetor `v` de tamanho `n`. Os elementos do vetor `v` sempre estão em ordem ascendente ou descendente - e o algoritmo funciona para ambos casos. Marque (X) nas modificações que, de maneira independente umas das outras, façam a função `count` funcionar adequadamente para vetores em ordem ascendente ou descendente.

- [x] A linha 12 deve ser mudada para: `if (v[i] <= v[j])`
  a linha 13 deve ser mudada para: `if (v[k] > e)`
  a linha 18 deve ser mudada para: `if (v[k] < e)`
- [ ] A linha 12 deve ser mudada para: `if (v[i] <= v[j])`
  a linha 13 deve ser mudada para: `if (v[k] < e)`
  a linha 18 deve ser mudada para: `if (v[k] > e)`
- [ ] A linha 12 deve ser mudada para: `if (v[i] <= v[j])`
  a linha 13 deve ser mudada para: `if (v[k] > e)`
  a linha 18 deve ser mudada para: `if (v[k] > e)`
- [ ] A linha 12 deve ser mudada para: `if (v[i] <= v[j])`
  a linha 13 deve ser mudada para: `if (v[k] < e)`
  a linha 18 deve ser mudada para: `if (v[k] < e)`
- [x] A linha 12 deve ser mudada para: `if (v[i] >= v[j])`
  a linha 13 deve ser mudada para: `if (v[k] < e)`
  a linha 18 deve ser mudada para: `if (v[k] > e)`
- [ ] A linha 12 deve ser mudada para: `if (v[i] >= v[j])`
  a linha 13 deve ser mudada para: `if (v[k] > e)`
  a linha 18 deve ser mudada para: `if (v[k] < e)`
- [ ] A linha 12 deve ser mudada para: `if (v[i] >= v[j])`
  a linha 13 deve ser mudada para: `if (v[k] > e)`
  a linha 18 deve ser mudada para: `if (v[k] > e)`
- [ ] A linha 12 deve ser mudada para: `if (v[i] >= v[j])`
  a linha 13 deve ser mudada para: `if (v[k] < e)`
  a linha 18 deve ser mudada para: `if (v[k] < e)`
- [ ] *Nenhuma das alternativas está correta*

```c
int foo6(int a, int b) {
    ...
    if (b < a) {
        return foo4(a+b);
    } else {
        return 24;
    }
}

int foo5(int a, int b) {
    if (a > b) {
        return foo6(a+a, b*b);
    } else {
        return foo6(a/2, b/2);
    }
}

int foo4(int a) {
    ...
    return foo5(a, a*a);
}

int foo3(int a, int b) {
    ...
    if (a != 0) {
        return foo3(a-1, b+1);
    } else {
        return b+1;
    }
}

int foo2(int b) {
    return foo3(foo4(b), b+1);
}

int foo1(int a, int b) {
    ...
    if (a > b) {
        return foo1(b, a);
    } else {
        return foo2(b);
    }
}

int foo() {
    ...
    int v1 = foo1(23, 10);
    int v2 = foo2(10);
    int v3_v4 = foo3(34, 12) + foo4(34-12);
    return v1+v2+v3_v4;
}
```

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

```c
int foobar1(int n, int b) {
    if (n < 2) {
        return n + b;
    } else {
        return foobar1(n-1, b) + foobar1(n-2, b);
    }
}

float foobar2t(int n, int b, float resp) {
    if (n < 1) {
        return resp;
    } else {
        return foobar2t(n-1, b, resp + (b*n));
    }
}

float foobar2(int n, int b) {
    return foobar2t(n, b, 0);
}

float foobar3t(int n, int b, float resp) {
    if (n < 1) {
        return resp;
    } else {
        if (n % 2 != 0) {
            return foobar3t(n-1, b, resp + n/foobar1(n, b));
        } else {
            return foobar3t(n-1, b, resp + foobar2(n, b));
        }
    }
}

float foobar3(int n, int b) {
    return foobar3t(n, b, 0);
}
```

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

```c
int zoot(int n, int resp) {
    if (n == 1) {
        return resp+1;
    } else {
        if (n %2 == 0)
            return zoot(n/2, resp+1);
        else
            return zoot((n*3)+1 , resp+1);
    }
}

int zoo(int n) {
    return zoot(n, 0);
}
```

Listagem 3: Função `zoo` na Linguagem C

```c
#include <stdio.h>

void foo(int v[], int i, int j);
int bar(int v[], int value, int i, int j);

int bar(int v[], int value, int  i, int j) {
    if (i >= j) {
        return i;
    } else {
        if (v[i] > value && v[j] <= value) {
            int temp = v[i];
            v[i] = v[j];
            v[j] = temp;
        }
        if (v[i] <= value) i++;
        if (v[j] > value) j--;
        return bar(v, value, i, j);
    }
}

void foo(int v[], int i, int j) {
    if (i <= j) {
        int value = v[(i+j)/2];
        int k = bar(v, value, i, j);
        foo(v, i, k-1);
        foo(v, k+1, j);
    }
}

int main (void) {
    int v1[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
    int v2[] = {4, 65, 2, -31, 0, 99, 2, 83, 2, 1};

    foo(v1, 0, 5);
    foo(v2, 5, 9);

    return 0;
}
```

Listagem 4: Código de programa na Linguagem C

```c
int max_div_comum(int n1, int n2) {
    if (n1 == 0)
        return n2;
    else
        return max_div_comum(n1%n2, n2/n1);
}
```

Listagem 5: Função que calcula o máximo divisor comum de `n1` e `n2`

```c
int count_aux(int v[], int e, int i, int j) {
    if (i > j)
        return 0;
    int k = (i+j)/2;
    if (v[k] == e) {
        i = k;
        j = k;
        while (v[i] == e) i--;
        while (v[j] == e) j++;
        return j-(i+1);
    } else {
        if (v[i] != v[j])
            if (v[k] > e)
                return count_aux(v, e, i, k-1);
            else
                return count_aux(v, e, k+1, j);
        else
            if (v[k] > e)
                return count_aux(v, e, i, k-1);
            else
                return count_aux(v, e, k+1, j);
        }
    }
}

int count(int v[], int e, int n) {
    return count_aux(v, e, 0, n-1);
}
```

Listagem 6: Função `count` que calcula o número de vezes que o elemento `e` aparece no vetor `v[n]` (`v` pode estar ordenado de maneira ascendente ou de maneira descendente - o algoritmo funciona para ambos casos)