SSC0600 - Introdução à Ciência de Computação I **Provinha 3(c)**
Tópico: Recursão **27 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

- ■ `foo0` é recursivo
- ■ `foo1` é recursivo
- ■ `foo2` é recursivo
- ■ `foo3` é recursivo
- ■ `foo4` é recursivo
- ☐ `foo5` é recursivo
- ■ `foo6` é recursivo
- ☐ `foo0` não é recursivo
- ☐ `foo1` não é recursivo
- ☐ `foo2` não é recursivo
- ☐ `foo3` não é recursivo
- ☐ `foo4` não é recursivo
- ■ `foo5` não é recursivo
- ☐ `foo6` 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:

- A função `pow(a, b)` calcula a potência de base a e expoente b: a^b
- Os n primeiros números potência base a são: a^1, a^2, a^3, ..., a^{n-1}, a^n
- Um número quadrado é um número inteiro cuja raiz quadrada e um outro número inteiro

- [x] `bar1` calcula a soma dos n primeiros números potências de base 2
- [ ] `bar1` calcula a soma dos n primeiros números quadrados
- [x] `bar2` calcula b divido entre o produto dos n primeiros números quadrados
- [ ] `bar2` calcula b divido entre a soma dos n primeiros números quadrados
- [ ] `bar2` calcula b divido entre o produto dos n primeiros números potência de base 2
- [ ] `bar2` calcula b divido entre a soma dos n primeiros números potência de base 2
- [x] `bar1` é a função que calcula $0 + 2^0 + 2^1 + 2^2 + ... + 2^{n-1} + 2^n$
- [ ] `bar1` é a função que calcula $0 + 0^2 + 1^2 + 2^2 + ... + (n-1)^2 + n^2$
- [x] `bar2` é a função que calcula $\frac{b}{n^2*(n-1)^2*(n-2)^2*...*4*2*1}$
- [ ] `bar2` é a função que calcula $\frac{b}{n^2+(n-1)^2+(n-2)^2+...+4+2+1}$
- [ ] `bar2` é a função que calcula $\frac{b}{2^n*2^{n-1}*2^{n-2}*...*4*2*1}$
- [ ] `bar2` é a função que calcula $\frac{b}{2^n+2^{n-1}+2^{n-2}+...+4+2+1}$
- [x] Se n é impar então `bar3` calcula a soma da sequência: $bar2(1)$, $bar2(3)$, $bar2(5)$, ..., $bar2(n-2)$, $bar2(n)$
- [x] Se n é par então `bar3` calcula a soma da sequência: $bar1(2)$, $bar1(4)$, $bar1(6)$, ..., $bar1(n-2)$, $bar1(n)$
- [ ] Se n é impar então `bar3` calcula a soma da sequência: $bar1(1)$, $bar1(3)$, $bar1(5)$, ..., $bar1(n-2)$, $bar1(n)$
- [ ] Se n é par então `bar3` calcula a soma da sequência: $bar2(2)$, $bar2(4)$, $bar2(6)$, ..., $bar2(n-2)$, $bar2(n)$
- [ ] Se n é impar então `bar3` calcula a soma da sequência: $bar2(1)$, $bar1(2)$, $bar2(3)$, ..., $bar2(n-2)$, $bar1(n-1)$, $bar2(n)$
- [ ] Se n é par então `bar3` calcula a soma da sequência: $bar1(1)$, $bar2(2)$, $bar1(3)$, ..., $bar1(n-2)$, $bar2(n-1)$, $bar3(n)$
- [ ] *Nenhuma das alternativas está correta*

Question [apply-unistructural] ♣ (1.5 ponto) Seja o vetor `v[6]={1,7,3,9,7,5}`. Marque (X) nas afirmativas verdadeiras em relação à chamadas para a função `zoo(v,5)` e `zoo(v,4)` da Listagem 3.

- [x] retorna 3 quando é `zoo(v,5)`
- [ ] retorna 1 quando é `zoo(v,5)`
- [x] retorna 3 quando é `zoo(v,4)`
- [ ] retorna 2 quando é `zoo(v,4)`
- [ ] retorna 2 quando é `zoo(v,5)`
- [ ] retorna 4 quando é `zoo(v,5)`
- [ ] retorna 4 quando é `zoo(v,4)`
- [ ] retorna 1 quando é `zoo(v,4)`
- [ ] *Nenhuma das alternativas está correta*

Question [`apply-relational`] ♣ (2 pontos) Marque (X) nas afirmativas verdadeiras em relação ao conteúdo dos vetores `v1` e `v2` após a execução do programa da Listagem 4.

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

Question [`evaluate-multistructural`] ♣ (1 ponto) Marque (X) nas afirmativas verdadeiras em relação a chamadas às funções `foo` e `bar` no programa da Listagem 4.

- [x] A chamada para a função `foo` é efetuada 9 vezes
- [ ] 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 10 vezes
- [ ] A chamada para a função `foo` é efetuada 11 vezes
- [x] A chamada para a função `bar` é efetuada 15 vezes
- [ ] A chamada para a função `bar` é efetuada 14 vezes
- [ ] A chamada para a função `bar` é efetuada 13 vezes
- [ ] A chamada para a função `bar` é efetuada 12 vezes
- [ ] A chamada para a função `bar` é efetuada 11 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 `parcheck` apresentada na Listagem 5 funcione adequadamente. A função devolve 0 se a cadeia de caracteres `s[]` de comprimento `n` tiver a mesma quantidade de parênteses de abertura e de fechamento (balanceamento de parênteses). Caso contrario, ele retorna um outro número diferente de 0.

- [x] A linha 2 deve ser mudada para: `if (n < 1) {`
  a linha 6 deve ser mudada para: `return parcheck_t(s, n-1, n_opens+1);`
  a linha 8 deve ser mudada para: `return parcheck_t(s, n-1, n_opens-1);`
  a linha 10 deve ser mudada para: `return parcheck_t(s, n-1, n_opens);`
- [x] A linha 2 deve ser mudada para: `if (n < 1) {`
  a linha 6 deve ser mudada para: `return parcheck_t(s, n-1, n_opens-1);`
  a linha 8 deve ser mudada para: `return parcheck_t(s, n-1, n_opens+1);`
  a linha 10 deve ser mudada para: `return parcheck_t(s, n-1, n_opens);`
- [ ] A linha 2 deve ser mudada para: `if (n < 1) {`
  a linha 6 deve ser mudada para: `return parcheck_t(s, n-1, n_opens);`
  a linha 8 deve ser mudada para: `return parcheck_t(s, n-1, n_opens+1);`
  a linha 10 deve ser mudada para: `return parcheck_t(s, n-1, n_opens-1);`
- [ ] A linha 2 deve ser mudada para: `if (n < 1) {`
  a linha 6 deve ser mudada para: `return parcheck_t(s, n-1, n_opens);`
  a linha 8 deve ser mudada para: `return parcheck_t(s, n-1, n_opens-1);`
  a linha 10 deve ser mudada para: `return parcheck_t(s, n-1, n_opens+1);`
- [ ] A linha 2 deve ser mudada para: `if (n < 0) {`
  a linha 6 deve ser mudada para: `return parcheck_t(s, n-1, n_opens+1);`
  a linha 8 deve ser mudada para: `return parcheck_t(s, n-1, n_opens-1);`
  a linha 10 deve ser mudada para: `return parcheck_t(s, n-1, n_opens);`
- [ ] A linha 2 deve ser mudada para: `if (n < 0) {`
  a linha 6 deve ser mudada para: `return parcheck_t(s, n-1, n_opens-1);`
  a linha 8 deve ser mudada para: `return parcheck_t(s, n-1, n_opens+1);`
  a linha 10 deve ser mudada para: `return parcheck_t(s, n-1, n_opens);`
- [ ] A linha 2 deve ser mudada para: `if (n < 0) {`
  a linha 6 deve ser mudada para: `return parcheck_t(s, n-1, n_opens);`
  a linha 8 deve ser mudada para: `return parcheck_t(s, n-1, n_opens+1);`
  a linha 10 deve ser mudada para: `return parcheck_t(s, n-1, n_opens-1);`
- [ ] A linha 2 deve ser mudada para: `if (n < 0) {`
  a linha 6 deve ser mudada para: `return parcheck_t(s, n-1, n_opens);`
  a linha 8 deve ser mudada para: `return parcheck_t(s, n-1, n_opens-1);`
  a linha 10 deve ser mudada para: `return parcheck_t(s, n-1, n_opens+1);`
- [ ] *Nenhuma das alternativas está correta*

Question [analyse-relational-2] ♣ (2 pontos) Seja `v[]` o vetor utilizado para representar um conjunto ordenado de `n` números inteiros (a ordem dos elementos no conjunto é ascendente). A função `lower_bound` apresentado na Listagem 6 tem sido proposto para calcular a posição do limitante inferior dos números maiores que um número inteiro `value`. Isso é calcular a posição do menor número entre os números maiores que `value`. Por exemplo:

- Para `v[7] = {1,2,4,5,8,9,10}`, a posição do límitante inferior de `value=7` é 4 devido a que o `v[4]=8` é o menor entre os números maiores que 7 (`v[4]=8`, `v[5]=8` e `v[6]=8`).
- Para `v[7] = {1,2,4,5,8,9,10}`, a posição do límitante inferior de `value=10` é -1 devido a que não há elemento maior que 10 no conjunto `v[]`.

Marque (X) nas modificações que, de maneira independente umas das outras, façam a função `lower_bound` funcionar adequadamente.

- [x] A linha 9 deve ser mudada para: `if (v[m] <= value)`
  a linha 10 deve ser mudada para: `return lower_bound_t(v, m+1, j, value);`
  a linha 12 deve ser mudada para: `return lower_bound_t(v, i, m-1, value);`
- [ ] A linha 9 deve ser mudada para: `if (v[m] <= value)`
  a linha 10 deve ser mudada para: `return lower_bound_t(v, i, m-1, value);`
  a linha 12 deve ser mudada para: `return lower_bound_t(v, m+1, j, value);`
- [ ] A linha 9 deve ser mudada para: `if (v[m] < value)`
  a linha 10 deve ser mudada para: `return lower_bound_t(v, m+1, j, value);`
  a linha 12 deve ser mudada para: `return lower_bound_t(v, i, m-1, value);`
- [ ] A linha 9 deve ser mudada para: `if (v[m] < value)`
  a linha 10 deve ser mudada para: `return lower_bound_t(v, i, m-1, value);`
  a linha 12 deve ser mudada para: `return lower_bound_t(v, m+1, j, value);`
- [x] A linha 9 deve ser mudada para: `if (v[m] > value)`
  a linha 10 deve ser mudada para: `return lower_bound_t(v, i, m-1, value);`
  a linha 12 deve ser mudada para: `return lower_bound_t(v, m+1, j, value);`
- [ ] A linha 9 deve ser mudada para: `if (v[m] > value)`
  a linha 10 deve ser mudada para: `return lower_bound_t(v, m+1, j, value);`
  a linha 12 deve ser mudada para: `return lower_bound_t(v, i, m-1, value);`
- [ ] A linha 9 deve ser mudada para: `if (v[m] >= value)`
  a linha 10 deve ser mudada para: `return lower_bound_t(v, i, m-1, value);`
  a linha 12 deve ser mudada para: `return lower_bound_t(v, m+1, j, value);`
- [ ] A linha 9 deve ser mudada para: `if (v[m] >= value)`
  a linha 10 deve ser mudada para: `return lower_bound_t(v, m+1, j, value);`
  a linha 12 deve ser mudada para: `return lower_bound_t(v, i, m-1, value);`
- [ ] *Nenhuma das alternativas está correta*

```c
int foo6(int a, int b) {
    if (foo4(a, b) > a+b) {
        return b*a;
    } else {
        return a+b;
    }
}

int foo5(int a, int b) {
    if (a > b) {
        return foo6(b, a);
    } else {
        return foo6(a, b);
    }
}

int foo4(int a, int b) {
    return foo6(b*a, a/b);
}

int foo3(int a, int b) {
    int a = foo5(b, a);
    int b = foo5(a, foo1(a-b, b-a))
    return a+b;
}

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

int foo1(int a, int b) {
    if (a - b > 0) {
        return foo2(b);
    }
    return foo3(a, b);
}

int foo0(int a, int b) {
    if (a != 0) {
        return foo1(a*a, foo0(a%b, b-1));
    } else {
        return b;
    }
}
```

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

```c
int bar1t(int n, int resp) {
    if (n < 0) {
        return resp;
    } else {
        return bar1t(n-1, pow(2,n)+resp);
    }
}

int bar1(int n) {
    return bar1t(n, 0);
}

float bar2(float n, float b) {
    if (n < 1) {
        return b;
    } else {
        return (1/pow(n, 2)) * bar2(n-1, b);
    }
}

float bar3(float n) {
    if (n >= 1) {
        if (n % 2 == 0) {
            return bar1(n) + bar3(n-2);
        } else {
            return bar2(n) + bar3(n-2);
        }
    }
    return 0;
}
```

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

```c
int zoo(int v[], int n) {
    int k;
    int c = 1;
    for (k = n-1 ; k >= 0 ; k--) {
        if (v[k] <= v[n]) {
            int d = zoo(v, k);
            if (d+1 > c) {
                c = d+1;
            }
        }
    }
    return c;
}
```

Listagem 3: Função `zoo` na Linguagem C

```c
#include<stdio.h>

void bar(int v[], int i, int n) {
    int l = (2*i)+1;
    int r = (2*i)+2;
    int m = i;
    if (l < n && v[l] > v[m]) {
        m = l;
    }
    if (r < n && v[r] > v[m]) {
        m = r;
    }
    if (m != i) {
        int aux = v[i];
        v[i] = v[m];
        v[m] = aux;
        bar(v, m, n);
    }
}

void foo(int v[], int k, int n) {
    if (k >= 0) {
        bar(v, k, n);
        foo(v, k-1, n);
    }
}

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

    foo(v1, 3, 8);
    foo(v2, 2, 8);

    return 0;
}
```

Listagem 4: Código de programa na Linguagem C

```c
int parcheck_t(char s[], int n, int n_opens) {
    if (n > 0) {
        return n_opens;
    } else {
        if (s[n-1] == '(') {
            return parcheck_t(s, n+1, n_opens);
        } else if (s[n-1] == ')') {
            return parcheck_t(s, n+1, n_opens);
        } else {
            return parcheck_t(s, n+1, n_opens);
        }
    }
}

int parcheck(char s[], int n) {
    return parcheck_t(s, n, 0);
}
```

Listagem 5: Calcula se uma string `s[]` tem balanceamento de parênteses

.
.

```c
int lower_bound_t(int v[], int i, int j, int value) {
    if (i > j) {
        return -1;
    } else {
        int m = (i+j)/2;
        if (v[m-1] <= value && v[m] > value) {
            return m;
        } else {
            if (v[m] == value)
                return lower_bound_t(v, i, j, value);
            else
                return lower_bound_t(v, i, j, value);
        }
    }
}

int lower_bound(int v[], int n, int value) {
    lower_bound_t(v, 0, n-1, value);
}
```

Listagem 6: Calcula a posição do limitante inferior dos números maiores que `value` para os números inteiros ordenados no vetor `v[]` de comprimento `n`