Matemática Concreta - Listas de exercícios

Helder Mateus dos Reis Matos - 201904940036

28 de outubro de 2020

Atenção: A linguagem C++ foi usada para implementação. Os passos importantes do programa são descritos e indicados pelas linhas `[inicial:final]`, com limites inclusivos.

Lista 1

1. **Escolha uma linguagem de programação, implemente as funções de recorrência e exiba os seis primeiros termos de cada sequência. Inclua o código fonte das funções na resposta.**

   (a) $a_1 = 5$ e $a_n = a_{n-1} + 3, \forall n > 1$

```cpp
#include <iostream> // std::cin e std::cout
#include <vector>   // std::vector

std::vector<float> recorrencia_1a(float a_1, int n) {
    std::vector<float> elementos = {a_1};

    while (n > 1) {
        elementos.push_back(elementos.back() + 3);
        --n;
    }

    return elementos;
}

int main() {
    int a_1, n;
    std::vector<float> resposta;

    std::cout << "Insira o valor de a_1: ";
    std::cin >> a_1;
    std::cout << "Insira o valor de n: ";
    std::cin >> n;
    resposta = recorrencia_1a(a_1, n);

    std::cout << n << " primeiros termos da sequencia: ";
    for (auto i: resposta) {
        std::cout << i << " ";
    }
}
```

`[4:5]` - Ao receber os valores de a_1 e n, a função `recorrencia_1a` cria um vetor `elementos`, contendo os elementos da sequência. Como o primeiro elemento é dado na entrada, o vetor já é inicializado com a_1.
`[7:10]` - Um laço de repetição adiciona ao vetor os elementos seguintes. O próximo elemento sempre será igual ao último elemento (retornado pela função `back()`) + 3.
`[12]` - Ao fim do laço, os n termos da sequência são retornados.

(b) $b_1 = 2$ e $b_n = b_{n-1}^2, \forall n > 1$

```cpp
#include <iostream> // std::cin and std::cout
#include <vector>   // std::vector
#include <cmath>    // pow()

std::vector<float> recorrencia_1b(float b_1, int n) {
    std::vector<float> elementos = {b_1};

    while (n > 1) {
        elementos.push_back(pow(elementos.back(), 2));
        --n;
    }

    return elementos;
}

int main() {
    int n;
    float b_1;
    std::vector<float> resposta;

    std::cout << "Insira o valor de b_1: ";
    std::cin >> b_1;
    std::cout << "Insira o valor de n: ";
    std::cin >> n;
    resposta = recorrencia_1b(b_1, n);

    std::cout << n << " primeiros termos da sequencia: ";
    for (auto i: resposta) {
        std::cout << i << " ";
    }
}
```

`[5:6]` - Similar a questão 1.a, a função `recorrencia_1b` cria um vetor inicializado com o primeiro elemento b_1.
`[8:11]` - O próximo elemento sempre será igual ao último elemento ao quadrado.
`[13]` - A função `recorrencia_1b` retorna os n elementos da sequência.

(c) $c_1 = 0$ e $c_n = 2c_{n-1} + n, \forall n > 1$

```cpp
#include <iostream> //std::cin e std::cout
#include <vector>   //std::vector

std::vector<float> recorrencia_1c(float c_1, int n) {
    std::vector<float> elementos = {c_1};

    for (int i = 2; i <= n; ++i) {
        elementos.push_back(2 * elementos.back() + i);
    }

    return elementos;
}

int main() {
    float c_1;
    int n;
    std::vector<float> resposta;

    std::cout << "Insira o valor de c_1: ";
    std::cin >> c_1;
    std::cout << "Insira o valor de n: ";
    std::cin >> n;
    resposta = recorrencia_1c(c_1, n);

    std::cout << n << " primeiros termos da sequencia: ";
    for (auto i: resposta) {
        std::cout << i << " ";
    }
}
```

`[4:5]` - Da mesma forma, `recorrencia_1c` começa com a declaração de um vetor contendo o primeiro elemento c_1.
`[7:9]` - O valor de n é usado na função de recorrência, portanto inicia-se um laço `for` onde o próximo elemento sempre será igual ao dobro do anterior + n.
`[12]` - A sequência resultante é retornada da função.

2. **Escolha uma linguagem de programação e escreva um programa para receber uma sequência numérica e informar se a sequência é um P.A ou não. Caso seja uma P.A, o programa deve informar se a P.A é crescente, constante ou decrescente.**

```
1  #include <iostream> // std::cin e std::cout
2  #include <vector>   // std::vector
3
4  void categorizar_pa(std::vector<float> sequencia, int n) {
5      float razao = sequencia[1] - sequencia[0];
6      bool pa = true;
7
8      for (int i = 2; i < n; ++i) {
9          if (sequencia[i] - sequencia[i-1] != razao) {
10             pa = false;
11             break;
12         }
13     }
14
15     if (pa == true) {
16         if (razao > 0) {
17             std::cout << "A sequencia eh P.A. crescente";
18         }
19         else if (razao < 0) {
20             std::cout << "A sequencia eh P.A. decrescente";
21         }
22         else {
23             std::cout << "A sequencia eh P.A. constante";
24         }
25     }
26     else {
27         std::cout << "A sequencia nao eh P.A.";
28     }
29 }
30
31 int main() {
32     int n;
33     float x;
34     std::vector<float> sequencia;
35
36     std::cout << "Insira o tamanho da sequencia: ";
37     std::cin >> n;
38     std::cout << "Insira os " << n << " elementos: ";
39     for (int i = 0; i < n; ++i) {
40         std::cin >> x;
41         sequencia.push_back(x);
42     }
43
44     categorizar_pa(sequencia, n);
45 }
```

`[4:6]` - O procedimento `categorizar_pa` recebe a sequência numérica e seu tamanho, ambos inseridos pelo usuário. Em seguida, a variável `razao` é declarada e armazena o valor da diferença entre o segundo e primeiro elemento. A booleana `pa` indica se a sequência é uma P.A., ao decorrer do algoritmo.
`[8:13]` - Ocorre a iteração sobre os elementos da sequência, a partir do terceiro, sempre verificando se a diferença entre o elemento atual e o anterior difere da `razao` entre os dois primeiros. Caso seja diferente em qualquer momento, `pa` se torna falsa e o loop é quebrado.

Caso contrário, `pa` sempre será verdadeira, já que a razão de uma P.A. é sempre constante.
`[15:29]` - Para exibir a informação da categoria da P.A., uma estrutura condicional aninhada é formada. Caso `pa` seja verdadeira, o sinal positivo, negativo ou nulo da `razao` indica se ela é crescente, decrescente ou constante, respectivamente.

3. **Sabendo que o primeiro termo é igual a 3 e a razão é igual a 5, calcule o 17 o termo de uma P.A.**
   O enésimo termo de uma P.A., onde o primeiro termo é a_1 e a razão é r, é dado por:

$$a_n = a_1 + (n-1)r$$

   Tomando $a_1 = 3$, $r = 5$ e $n = 17$, temos:

$$a_{17} = 3 + (17-1)5$$

$$a_{17} = 3 + 16 \cdot 5$$

$$a_{17} = 3 + 80$$

$$a_{17} = 83$$

4. **Sabendo que o primeiro termo é igual a -8 e o vigésimo igual a 30, calcule a razão da P.A.**
   Partindo da definição do enésimo termo de uma P.A., a razão pode ser encontrada por:

$$a_n = a_1 + (n-1)r$$

$$(n-1)r = a_n - a_1$$

$$r = \frac{a_n - a_1}{n-1}$$

   De posse de $a_1 = -8$ e $a_n = a_{20} = 30$, a razão desta P.A. é:

$$r = \frac{30 - (-8)}{20 - 1}$$

$$r = \frac{38}{19}$$

$$r = 2$$

5. **Escolha uma linguagem de programação e escreva um programa para receber os extremos de uma P.A, o valor de K e calcule a interpolação dessa P.A.**

```cpp
#include <iostream> // std::cin e std::cout
#include <vector>   // std::vector

std::vector<float> interpor_pa(float a_1, float a_n, int k) {
    float razao;
    std::vector<float> elementos;

    razao = (a_n - a_1)/(k+1);
    while (a_1 <= a_n) {
        elementos.push_back(a_1);
        a_1 += razao;
    }

    return elementos;
}

int main() {
    float a_1, a_n;
    int k;
    std::vector<float> resposta;

    std::cout << "Insira o primeiro termo da P.A.: ";
    std::cin >> a_1;
    std::cout << "Insira o ultimo termo da P.A.: ";
    std::cin >> a_n;
    std::cout << "Insira o valor de k: ";
    std::cin >> k;
    resposta = interpor_pa(a_1, a_n, k);

    std::cout << "P.A. interpolada: ";
    for (auto i: resposta) {
        std::cout << i << " ";
    }
}
```

`[4:6]` - A função `interpor_pa` recebe os três dados do usuário, a_1, a_n e k. São declarados a variável `q`, que armazena a razão da sequência, e o vetor `elementos`.
`[8:12]` - A razão $r = \frac{a_n - a_1}{k+1}$ da P.A. é calculada. Em seguida um laço de repetição insere o valor atual de a_1 no vetor `elementos` e soma o mesmo com o valor da `razao`, até que $a_1 = a_n$.
`[14]` - A P.A. interpolada é retornada.

6. **Calcule a P.A em que a soma dos n primeiros termos é igual a $n^2 + 2n$.**
Conhecendo a expressão da soma dos n termos, podemos encontrar a soma S_1 do primeiro termo, que nada mais é do que o primeiro termo:

$$S_n = n^2 + 2n$$

$$S_1 = 1^2 + 2 \cdot 1 = 3 \quad \therefore \quad a_1 = 3$$

Encontrando a soma S_2 dos dois primeiros termos, podemos encontrar o segundo termo, pois ele é a diferença entre S_2 e S_1. Para os demais termos, o procedimento é semelhante:

$$S_2 = 2^2 + 2 \cdot 2 = 8 \quad \therefore \quad a_2 = S_2 - S_1 = 8 - 3 = 5$$

$$S_3 = 3^2 + 2 \cdot 3 = 15 \quad \therefore \quad a_3 = S_3 - S_2 = 15 - 8 = 7$$

$$S_4 = 4^2 + 2 \cdot 4 = 24 \quad \therefore \quad a_4 = S_4 - S_3 = 24 - 15 = 9$$

$$S_5 = 5^2 + 2 \cdot 5 = 35 \quad \therefore \quad a_5 = S_5 - S_4 = 35 - 22 = 11$$

Pode-se verificar a formação de uma progressão aritmética de razão 2, portanto os demais elementos sempre aumentarão com esta razão.
Generalizando, $a_n = S_n - S_{n-1} = (n^2 + 2n) - ((n-1)^2 + 2(n-1))$, para $S_1 = a_1 = 3$.

7. **Escolha uma linguagem de programação e escreva um programa para receber uma sequência numérica e informar se a sequência é um P.G ou não. Caso seja uma P.G, o programa deve informar se a P.G é crescente, constante, decrescente, alternante ou estacionária.**

```cpp
#include <iostream> // std::cin e std::cout
#include <vector>   // std::vector
#include <cmath>    // std::floor
#include <string>   // std::string
#include <sstream>  // std::isstringstream

float round(float num) {
    return std::floor(num*100000.0) / 100000.0;
}

float parse_stof(std::string str) {
    float parsed_float;
    std::string num = "";
    bool divisao = false;
    int numerador, denominador;

    for (auto c: str) {
        if (std::isdigit(c)) {
            num += c;
        }
        else if (c == '/') {
            numerador = stoi(num);
            num = "";
            divisao = true;
        }
        if (c == str.back() && divisao == true) {
            denominador = stoi(num);
            parsed_float = 1.0*numerador/denominador;
        }
        else if (c == str.back() && divisao == false) {
            parsed_float = stoi(num);
        }
    }

    return parsed_float;
}

std::vector<float> split_and_parse(std::string input) {
    std::vector<float> sequencia;
    std::vector<std::string> splitted_strings;
    std::istringstream iss(input);
    std::string number;

    while (std::getline(iss, number, ' ')) {
        splitted_strings.push_back(number);
    }
    for (auto str: splitted_strings) {
        sequencia.push_back(parse_stof(str));
    }

    return sequencia;
}

void categorizar_pg(std::vector<float> seq, int n) {
    float q = round(seq[1] / seq[0]);
    bool pg = false;

    for (int i = 2; i < n; ++i) {
        if (round(seq[i] / seq[i-1]) == q || q == 0 && seq[i-1] == 0) {
```

```
60              pg = true;
61          }
62          else {
63              pg = false;
64              break;
65          }
66      }
67
68      if (pg == true) {
69          if ((q > 1 && seq[0] > 0) || (q > 0 && q < 1 && seq[0] < 0)) {
70              std::cout << "A sequencia eh P.G. crescente";
71          }
72          else if ((q > 0 && q < 1 && seq[0] > 0) || (q > 1 && seq[0] < 0)) {
73              std::cout << "A sequencia eh P.G. decrescente";
74          }
75          else if (q == 1 && seq[0] != 0) {
76              std::cout << "A sequencia eh P.G. constante";
77          }
78          else if (q < 0) {
79              std::cout << "A sequencia eh P.G. alternante";
80          }
81          else if (q == 0) {
82              std::cout << "A sequencia eh P.G. estacionaria";
83          }
84      }
85      else {
86          std::cout << "A sequencia nao eh P.G.";
87      }
88  }
89
90  int main() {
91      int n;
92      float x;
93      std::vector<float> sequencia;
94      std::string input;
95
96      std::cout << "Insira o tamanho da sequencia: ";
97      std::cin >> n;
98      std::cout << "Insira os " << n << " primeiros termos da sequencia: ";
99      std::cin.ignore();
100     std::getline(std::cin, input);
101     sequencia = split_and_parse(input);
102
103     categorizar_pg(sequencia, n);
104 }
```

`[7:9]` - Verificou-se que seria interessante fazer a entrada de sequência de números reais fracionários para esta questão, para solucionar o problema de precisão de representação de pontos flutuantes. Foi criada a função `round` que "arredonda para baixo"(através da `std::floor`) um número real, com precisão de 6 casas decimais.
`[11:36]` - Para a entrada de números reais fracionários, foi criada a função `parse_stof` que converte strings em valores numéricos, além de verificar se há sinal de divisão `'/'` dentre os números da sequência inserida. Caso exista, é efetuada a divisão entre o numerador e o denominador. Desse modo, é possível representar com mais exatidão dízimas periódicas, e.g. $1/3 = 0.3333...$
`[38:52]` - Não há função nativa equivalente em C++ para dividir uma string por caractere, como em outras linguagens. A função `split_and_parse` foi criada com esse intuito, analisando uma stream que contém a string inserida pelo usuário e criando um vetor `splitted_strings`, contendo as substrings divididas por espaços simples. Em seguida a função `parse_stof` é chamada para a conversão das strings em floats.
`[54:55]` - O procedimento `categorizar_pg` recebe a sequência `seq` e seu tamanho n. Para encontrar o valor de q, bastaria dividir qualquer um dos termos da sequência pelo anterior,

sendo escolhidos os dois primeiros. Note que isso traria um problema de precisão decimal, que é rapidamente solucionado pela função `round`, definida em `[7:9]`.
`[58:66]` - É verificado para cada termo a partir do terceiro, se a razão entre ele e o anterior é igual a razão dos dois primeiros, calculada em `[55]`. Não somente isso, caso a sequência venha a ser estacionária ($q = 0$), em algum momento ocorreria uma divisão por zero, fato mencionado na segunda operação lógica da condição. Se uma das condições se satisfazerem para todos os elementos, a sequência é P.G. Caso contrário, ou seja, se pelo menos uma das razões entre dois termos for diferente ou a P.G. não for estacionária, a sequência não é P.G.
`[68:88]` - O último passo do procedimento é categorizar, dependendo do valor de q, e, quando necessário, do sinal dos termos da sequência. Caso $q > 1$ para termos positivos ou $0 < q < 1$ para termos negativos, a P.G. é crescente. Caso $0 < q < 1$ para termos negativos ou $q > 1$ para termos positivos, a P.G. é decrescente. Se $q = 1$ para termos não nulos, a P.G. é constante. Se $q < 0$, os termos alternam o sinal. Para $q = 0$, a P.G. é estacionária.

8. **Escolha uma linguagem de programação e escreva um programa para receber os extremos de uma P.G, o valor de K e calcule a interpolação dessa P.G.**

```cpp
#include <iostream> // std::cin e std::cout
#include <vector>   // std::vector
#include <cmath>    // std::pow()

std::vector<float> interpolar_pg(float a_1, float a_n, int k) {
    std::vector<float> elementos;
    float q;

    q = pow((a_n/a_1), (1.0/(k+1)));
    while (a_1 <= a_n){
        elementos.push_back(a_1);
        a_1 *= q;
    }

    return elementos;
}

int main() {
    float a_1, a_n;
    int k;
    std::vector<float> resposta;

    std::cout << "Insira o valor de a_1: ";
    std::cin >> a_1;
    std::cout << "Insira o valor de a_n: ";
    std::cin >> a_n;
    std::cout << "Insira o valor de k: ";
    std::cin >> k;
    resposta = interpolar_pg(a_1, a_n, k);

    std::cout << "P.G. interpolada: ";
    for (auto i: resposta) {
        std::cout << i << " ";
    }
}
```

`[5:7]` - `interpor_pa` recebe o extremo inferior a_1, o extremo superior a_n e o valor de k termos intermediários. O float q é a razão entre elementos consecutivos, enquanto o vetor `elementos` armazena os elementos da interpolação.
`[9:13]` - É definido $q = \frac{a_n - a_1}{k+1}$. De posse do valor de q basta usá-lo para incrementar geometricamente o valor de a_1 e inserir os resultados no vetor `elementos`, até que $a_1 = a_n$.

9. **Escolha uma linguagem de programação e escreva um programa para receber uma sequência numérica. Se a sequência numérica for uma P.G, informe a produto e a soma dos termos dessa P.G. Caso contrário, informe que a sequência não é uma P.G.**

```cpp
#include <iostream> // std::cin e std::cout
#include <vector>   // std::vector
#include <cmath>    // std::pow()

float round(float num) {
    return std::floor(num*100000.0) / 100000.0;
}

void soma_produto_pg(std::vector<float> seq, int n) {
    float q = round(seq[1] / seq[0]);
    bool pg = false;
    float soma, prod;

    for (int i = 2; i < n; ++i) {
        if (round(seq[i] / seq[i-1]) == q || q == 0 && seq[i-1] == 0) {
            pg = true;
        }
        else {
            pg = false;
            break;
        }
    }

    if (pg == true){
        soma = (seq[0] * pow(q, n) - seq[0]) / (q-1);
        prod = pow(seq[0], n) * pow(q, n*(n-1)/2.0);

        std::cout << "A sequencia eh uma P.G.\n";
        std::cout << "Sua soma eh: " << soma << "\n";
        std::cout << "Seu produto eh: " << prod << "\n";
    }
    else std::cout << "A sequencia nao eh uma P.G.";
}

int main() {
    int n;
    float x;
    std::vector<float> sequencia;

    std::cout << "Insira o tamanho da sequencia: ";
    std::cin >> n;
    std::cout << "Insira os " << n << " elementos da sequencia: ";
    for (int i = 0; i < n; ++i){
        std::cin >> x;
        sequencia.push_back(x);
    }
    soma_produto_pg(sequencia, n);
}
```

`[5:7]` - Desta vez não se fará o uso de entradas fracionárias, mas manteve-se a função de arrendondamento para baixo.
`[5:8]` - O procedimento `soma_produto_pg` recebe a sequência e seu tamanho. Em seguida, q é declarada como a razão entre o segundo e primeiro elemento, junto a uma booleana que indica se a sequência é P.G. e variáveis para armazenar soma e produto. Note que não há necessidade de iniciar essas variáveis, pois existem relações já definidas para calculá-las, em detrimento da acumulação causada por uma iteração.
`[14:22]` - Para cada termo em diante, verifica-se se o quociente entre o mesmo e o seu anterior é igual à razão q, evitando também o caso especial em que a P.G. é estacionária.
`[24:33]` - Como já mencionado acima, ocorre a aplicação da soma da P.G, $S_{P.G.} = \frac{a_1 q^n - a_1}{q-1}$,

e do produto da P.G., $P_{P.G.} = a_1^n \cdot q^{\frac{n(n-1)}{2}}$. Em seguida, os resultados são apresentados.

10. **Determine o valor de n tal que $\sum_{i=3}^{n} 2^i = 4088$** Para encontrar a soma de $i = 3$ até n, basta notar que a soma de $i = 1$ até n pode ser representada da seguinte forma:

$$\sum_{i=1}^{n} 2^i = \sum_{i=1}^{2} 2^i + \sum_{i=3}^{n} 2^i$$

O lado esquerdo representa a soma de uma progressão geométrica.

$$\frac{a_1 q^n - a_1}{q - 1} = \sum_{i=1}^{2} 2^i + \sum_{i=3}^{n} 2^i$$

Tomando $a_1 = 2^1 = 2$ e $q = 2$, e substituindo os valores que já conhecemos do lado direito obtemos:

$$\frac{2 \cdot 2^n - 2}{2 - 1} = (2 + 4) + 4088$$
$$2 \cdot 2^n - 2 = 6 + 4088$$
$$2 \cdot 2^n = 4094 + 2$$
$$2^n = \frac{4096}{2}$$
$$2^n = 2048$$
$$2^n = 2^{11}$$
$$n = 11$$

Lista 2

1. **Escolha uma linguagem de programação e escreva um programa que receba uma sequência digitada pelo usuário e exiba a subsequência de números ímpares.**

```cpp
#include <iostream>
#include <vector>

std::vector<int> subseq_impar(std::vector<int> input) {
    std::vector<int> impares;

    for (auto i: input) {
        if (i % 2 != 0) {
            impares.push_back(i);
        }
    }

    return impares;
}

int main() {
    std::vector<int> input;
    std::vector<int> impares;
    int n, x;

    std::cout << "Insira o tamanho da sequencia: ";
    std::cin >> n;
    std::cout << "Insira os " << n << " elementos da sequencia: ";
    while (n > 0) {
        std::cin >> x;
        input.push_back(x);
        --n;
    }
    impares = subseq_impar(input);

    std::cout << impares.size() << " elementos impares: ";
    for (auto i: impares) {
        std::cout << i << " ";
    }
}
```

2. **Escolha uma linguagem de programação e escreva um programa que receba uma sequência digitada pelo usuário e exiba a subsequência de números primos.**

```cpp
#include <iostream>
#include <vector>
#include <cmath>

bool primacidade(int num) {
    if (num <= 2) {
        return (num == 2) ? true : false;
    }
    for (int i = 2; i <= std::sqrt(num); ++i) {
        if (num % i == 0) return false;
    }
    return true;
}

std::vector<int> subseq_primos(std::vector<int> input) {
    std::vector<int> primos;

    for (auto i: input) {
        if (primacidade(i) == true) primos.push_back(i);
    }

    return primos;
}

```

```cpp
int main() {
    std::vector<int> input;
    std::vector<int> primos;
    int n, x;

    std::cout << "Insira o tamanho da sequencia: ";
    std::cin >> n;
    std::cout << "Insira os " << n << " elementos da sequencia: ";
    while (n > 0) {
        std::cin >> x;
        input.push_back(x);
        --n;
    }
    primos = subseq_primos(input);

    std::cout << primos.size() << " elementos primos: ";
    for (auto i: primos) {
        std::cout << i << " ";
    }
}
```

3. **Escolha uma linguagem de programação e escreva um programa que receba duas sequências A e B digitada pelo usuário e exiba a concatenação BA.**

```cpp
#include <iostream>
#include <vector>

std::vector<int> concatenar(std::vector<int> input_A, std::vector<int> input_B) {
    std::vector<int> ba;

    ba.insert(ba.end(), input_B.begin(), input_B.end());
    ba.insert(ba.end(), input_A.begin(), input_A.end());

    return ba;
}

int main() {
    std::vector<int> input_A;
    std::vector<int> input_B;
    std::vector<int> ba;
    int n, x;

    std::cout << "Insira o tamanho da sequencia A: ";
    std::cin >> n;
    std::cout << "Insira os " << n << " elementos da sequencia A: ";
    while (n > 0) {
        std::cin >> x;
        input_A.push_back(x);
        --n;
    }
    std::cout << "Insira o tamanho da sequencia B: ";
    std::cin >> n;
    std::cout << "Insira os " << n << " elementos da sequencia B: ";
    while (n > 0) {
        std::cin >> x;
        input_B.push_back(x);
        --n;
    }
    ba = concatenar(input_A, input_B);

    std::cout << ba.size() << " elementos BA: ";
    for (auto i: ba) {
        std::cout << i << " ";
    }
}
```

4. **Escolha uma linguagem de programação e escreva um programa que receba uma sequência, os índices a e b digitados pelo usuário, e exiba o segmento com os**

extremos x_a e x_b. Considere que sempre $a \leq b$.

```cpp
#include <iostream>
#include <vector>

std::vector<int> segmentar(std::vector<int> input, int a, int b) {
    std::vector<int> segmento(b - a + 1);
    std::vector<int>::iterator inicio = input.begin() + a;
    std::vector<int>::iterator fim = input.begin() + b + 1;

    std::copy(inicio, fim, segmento.begin());

    return segmento;
}

int main() {
    std::vector<int> input;
    std::vector<int> segmento;
    int n, x, a, b;

    std::cout << "Insira o tamanho da sequencia: ";
    std::cin >> n;
    std::cout << "Insira os " << n << " elementos da sequencia: ";
    while (n > 0) {
        std::cin >> x;
        input.push_back(x);
        --n;
    }
    std::cout << "Insira o indice para o extremo inferior a: ";
    std::cin >> a;
    std::cout << "Insira o indice para o extremo superior b: ";
    std::cin >> b;
    segmento = segmentar(input, a, b);

    std::cout << "Segmento entre x["  << a << "] e x[" << b << "]: ";
    for (auto i: segmento) {
        std::cout << i << " ";
    }
}
```

5. **Escolha uma linguagem de programação e escreva um programa que receba uma sequência, o valor k digitado pelo usuário, e exiba o prefixo de comprimento k.**

```cpp
#include <iostream>
#include <vector>

std::vector<int> prefixar(std::vector<int> input, int k) {
    std::vector<int> prefixo(k);
    std::vector<int>::iterator inicio = input.begin();
    std::vector<int>::iterator fim = input.begin() + k;

    std::copy(inicio, fim, prefixo.begin());

    return prefixo;
}

int main() {
    std::vector<int> input;
    std::vector<int> prefixo;
    int n, x, k;

    std::cout << "Insira o tamanho da sequencia: ";
    std::cin >> n;
    std::cout << "Insira os " << n << " elementos da sequencia: ";
    while (n > 0) {
        std::cin >> x;
        input.push_back(x);
        --n;
    }
    std::cout << "Insira o valor k: ";
```

```
    std::cin >> k;
    prefixo = prefixar(input, k);

    std::cout << "Prefixo de comprimento " << k << ": ";
    for (auto i: prefixo) {
        std::cout << i << " ";
    }
}
```

6. **Escolha uma linguagem de programação e escreva um programa que receba uma sequência, o valor k digitado pelo usuário, e exiba o sufixo de comprimento k.**

```
#include <iostream>
#include <vector>

std::vector<int> sufixar(std::vector<int> input, int k) {
    std::vector<int> sufixo(k);
    std::vector<int>::iterator inicio = input.begin() + input.size() - k;
    std::vector<int>::iterator fim = input.end();

    std::copy(inicio, fim, sufixo.begin());

    return sufixo;
}

int main() {
    std::vector<int> input;
    std::vector<int> sufixo;
    int n, x, k;

    std::cout << "Insira o tamanho da sequencia: ";
    std::cin >> n;
    std::cout << "Insira os " << n << " elementos da sequencia: ";
    while (n > 0) {
        std::cin >> x;
        input.push_back(x);
        --n;
    }
    std::cout << "Insira o valor de k elementos do sufixo: ";
    std::cin >> k;
    sufixo = sufixar(input, k);

    std::cout << "Sufixo de comprimento " << k << ": ";
    for (auto i: sufixo) {
        std::cout << i << " ";
    }
}
```