Matemática Concreta - Listas de exercícios

Helder Mateus dos Reis Matos - 201904940036

28 de outubro de 2020

Atenção: As questões de implementação foram feitas em C++. Passos importantes dos programas são descritos pelas indicações de linhas `[inicial:final]`, com limites inclusivos.

Atividade aula 1

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.**

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

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

#define a_1 5   // primeiro termo da sequencia
#define n   6   // tamanho da sequencia

int main() {
    float sequencia[n];                         // declarando array de n elementos

    sequencia[0] = a_1;                         // inserindo o primeiro termo
    for (int i = 1; i < n; ++i) {               // inserindo os demais termos
        sequencia[i] = sequencia[i-1] + 3;      // funcao de recorrencia
    }

    std::cout << n << " primeiros termos da sequencia: ";
    for (int i = 0; i < n; ++i) {
        std::cout << sequencia[i] << " ";
    }
}
```

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

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

#define a_1 2   // primeiro termo da sequencia
#define n   6   // tamanho da sequencia

int main() {
    float sequencia[n];                         // declarando arrays de n elementos

    sequencia[0] = a_1;                         // inserindo o primeiro termo
    for (int i = 1; i < n; ++i) {               // inserindo os demais termos
        sequencia[i] = pow(sequencia[i-1], 2);  // funcao de recorrencia
    }

    std::cout << n << " primeiros termos da sequencia: ";
    for (int i = 0; i < n; ++i) {
        std::cout << sequencia[i] << " ";
    }
}
```

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

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

#define a_1 0   // primeiro termo da sequencia
#define n   6   // tamanho da sequencia

int main() {
    float sequencia[n];                         // declarando array de n elementos

    sequencia[0] = a_1;                         // inserindo o primeiro termo
    for (int i = 1; i < n; ++i) {               // inserindo os demais termos
        sequencia[i] = 2 * sequencia[i-1] + i;  // funcao de recorrencia
    }

    std::cout << n << " primeiros termos da sequencia: ";
    for (int i = 0; i < n; ++i) {
        std::cout << sequencia[i] << " ";
    }
}
```

1.2. **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.**

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

bool checar_pa(float *sequencia, float razao, int n) {
    for (int i = 2; i < n ; ++i) {
        if (sequencia[i] - sequencia[i-1] != razao) {
            return false; // caso uma das razoes seja diferente
        }
    }
    return true; // caso todas as razoes sejam iguais
}

void categorizar_pa(float *sequencia, int n) {
    float razao = sequencia[1] - sequencia[0]; // a_2 - a_1

    if (checar_pa(sequencia, razao, n) == false) {
        std::cout << "A sequencia nao eh P.A.";
    }
    else {
        if (razao > 0) {
            std::cout << "A sequencia eh P.A. crescente";
        }
        else if (razao < 0) {
            std::cout << "A sequencia eh P.A. decrescente";
        }
        else {
            std::cout << "A sequencia eh P.A. constante";
        }
    }
}

int main() {
    int n;
    std::cout << "Insira o tamanho da sequencia: ";
    std::cin >> n;

    float sequencia[n];
    std::cout << "Insira os " << n << " elementos: ";
    for (int i = 0; i < n; ++i) {
        std::cin >> sequencia[i];
    }

    categorizar_pa(sequencia, n);
}
```

`[3:10]` - A função `checar_pa` recebe um array contendo a sequência, a razão entre os dois primeiros elementos e o seu tamanho n. Se a diferença entre um termo da sequência e o seu anterior for diferente da razão, a sequência não é P.A. Caso a razão se mantenha constante pra cada par de elementos, a sequência é P.A.
`[12:29]` - `categorizar_pa` recebe apenas a sequência e seu tamanho. Ao calcular a razão entre o segundo e primeiro termo, é chamada a função `checar_pa`, que dependendo do seu retorno e do valor da razão, classifica a sequência em "não P.A.", P.A. crescente, P.A. decrescente ou P.A. constante.

1.3. **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:

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

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

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

$$a_{17} = 83$$

1.4. **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:

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

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

$$r = 2$$

1.5. **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

void interpor_pa(float* pa, float a_1, float a_n, int k) {
    float razao = (a_n - a_1)/(k+1); // razao da pa

    for (int i = 0; i < k+2; ++i) {
        pa[i] = a_1;                 // inserindo elementos a partir de a_1
        a_1 += razao;                // incremento com base na razao
    }
}

int main() {
    float a_1, a_n;
    int k;

    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;

    float* pa;
    pa = (float*) malloc(sizeof(float) * (k+2)); // alocando memoria para k+2 termos reais
    interpor_pa(pa, a_1, a_n, k);

    std::cout << "P.A. interpolada: ";
    for (int i = 0; i < k+2; ++i) {
        std::cout << pa[i] << " ";
    }

    free(pa); // liberando memoria
}
```

`[3:10]` - O procedimento `interpor_pa` recebe uma sequência vazia, os extremos da P.A. e a quantidade k de elementos entre os extremos. De posse da razão, cada elemento da P.A. é inserido na sequência, começando do primeiro até o último, conforme o valor da razão.
`[23:33]` - Como o tamanho da sequência $(k+2)$ depende da entrada e não é mais alterado, optou-se pela alocação dinâmica do array, ao invés de estruturas mais sofisticadas.

1.6. **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 $S_1 = a_1 = 3$.

1.7. **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 <cmath>    // std::floor
#include <string>   // std::string
#include <sstream>  // std::isstringstream

float chao(float num) { // arredonda para baixo, 6 casas decimais
    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) || 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;
}

void split(float* sequencia, int n, std::string input) {
    std::istringstream iss(input);      // stream de entrada das strings
    std::string number;                 // string que representa um numero

    int i = 0;
    while (std::getline(iss, number, ' ')) {    // number armazena um termo
        sequencia[i] = parse_stof(number);      // parse de cada termo
        ++i;
    }
}

bool checar_pg(float* sequencia, float q, int n) {
    for (int i = 2; i < n; ++i) {
        if (chao(sequencia[i]/sequencia[i-1]) != q) {
            return false;
        }
    }
    return true;
}

void categorizar_pg(float* seq, int n) {
    float q = chao(seq[1] / seq[0]);

```

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

`[6:8]` - Excepcionalmente nessa solução é possível fazer a entrada de sequência de números reais fracionários, e.g. 1/3. Para solucionar o problema de precisão na representação de pontos flutuantes foi criada a função `chao` que arredonda um número real para baixo, com precisão de 6 casas decimais.
`[10:35]` - 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.
`[37:46]` - Não há função nativa em C++ para dividir uma string por caractere, como em outras linguagens. A função `split` foi criada com esse intuito, dividindo a stream de strings inserida pelo usuário em strings divididas por espaços simples. Em seguida a função `parse_stof` é chamada para a conversão das strings em floats.
`[48:55]` - É retornado de `checar_pg` um booleano indicando se a sequência informada é uma P.G., baseada na razão q entre os dois primeiros números.
`[57:81]` - A P.G. é classificada de acordo com o 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.

1.8. **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 <cmath>    // std::pow()

void interpor_pg(float* pg, float a_1, float a_n, int k) {
    float q = pow((a_n/a_1), (1.0/(k+1)));  //razao da pg

    for (int i = 0; i < k+2; ++i) {
        pg[i] = a_1;                        // inserindo elementos a partir de a_1
        a_1 *= q;                           // incremento com base em q
    }
}

int main() {
    float a_1, a_n;
    int k;

    std::cout << "Insira o primeiro termo da P.G.: ";
    std::cin >> a_1;
    std::cout << "Insira o ultimo termo da P.G.: ";
    std::cin >> a_n;
    std::cout << "Insira o valor de k: ";
    std::cin >> k;

    float* pg;
    pg = (float*) malloc(sizeof(float) * (k+2)); // alocando memoria para k+2 termos
    interpor_pg(pg, a_1, a_n, k);

    std::cout << "P.G. interpolada: ";
    for (int i = 0; i < k+2; ++i) {
        std::cout << pg[i] << " ";
    }

    free(pg); // liberando memoria
}
```

`[4:12]` - `interpor_pa` recebe a sequência vazia, o extremo inferior a_1, o extremo superior a_n e o valor de k termos intermediários. A razão entre elementos consecutivos $q = \sqrt[k+1]{\frac{a_n}{a_1}}$ é usada para incrementar geometricamente o valor de a_1 a cada iteração no array que armazena a sequência.

1.9. **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 chao(float num) {
    return std::floor(num*100000.0) / 100000.0;
}

bool checar_pg(float* sequencia, float q, int n) {
    for (int i = 2; i < n; ++i) {
        if (chao(sequencia[i]/sequencia[i-1]) != q) {
            return false;
        }
    }
    return true;
}

void soma_produto_pg(float* seq, int n) {
```

```
19      float q = round(seq[1] / seq[0]);
20      float soma, prod;
21
22      if (checar_pg(seq, q, n) == false) {
23          std::cout << "A sequencia nao eh uma P.G.";
24      }
25      else {
26          soma = (seq[0] * pow(q, n) - seq[0]) / (q-1);
27          prod = pow(seq[0], n) * pow(q, n*(n-1)/2.0);
28
29          std::cout << "A sequencia eh uma P.G.\n";
30          std::cout << "Sua soma eh: " << soma << "\n";
31          std::cout << "Seu produto eh: " << prod << "\n";
32      }
33  }
34
35  int main() {
36      int n;
37
38      std::cout << "Insira o tamanho da sequencia: ";
39      std::cin >> n;
40
41      float* sequencia;
42      sequencia = (float*) malloc(sizeof(float) * n);
43
44      std::cout << "Insira os " << n << " elementos da sequencia: ";
45      for (int i = 0; i < n; ++i){
46          std::cin >> sequencia[i];
47      }
48
49      soma_produto_pg(sequencia, n);
50      free(sequencia)
51  }
```

`[5:7]` - Desta vez não se fará o uso de entradas em forma de fração, mas manteve-se a função de arrendondamento para baixo, pois os resultados das divisões a seguir podem ser fracionários e não serem representados de forma precisa, ocasionando inconsistência ao compará-los com outros números fracionários.
`[9:16]` - `checar_pg` verifica se a razão entre cada elemento da sequência e seu antecessor é igual à razão entre os dois primeiros. `[18:33]` - O procedimento `soma_produto_pg` recebe a sequência e seu tamanho e verifica se é uma P.G. Se for, 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.

1.10. **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:

$$\frac{2 \cdot 2^n - 2}{2-1} = (2+4) + 4088$$

$$2 \cdot 2^n - 2 = 6 + 4088$$

$$2 \cdot 2^n = 4094 + 2$$

$$2^n = \frac{4096}{2}$$

$$2^n = 2048$$

$$2^n = 2^{11}$$

$$n = 11$$

Atividade aula 3

3.1. **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 << " ";
}
```

[4:12] - Nas soluções da lista 3 foi utilizada a classe de objetos do C++ `std::vector`, pela facilidade e flexibilidade ao armazenar, inserir e referenciar elementos. `subseq_impar` recebe uma sequência e retorna a subsequência de números ímpares.

3.2. **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)) primos.push_back(i);

    return primos;
}

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 << " ";
}
```

`[5:13]` - `primacidade` retorna um booleano indicando a primacidade de um número.
`[15:21]` - `subseq_primos` retorna a subsequência de números primos, a partir da sequência de entrada.

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

```
1  #include <iostream>
2  #include <vector>
3
4  std::vector<int> concatenar(std::vector<int> vec1, std::vector<int> vec2) {
5      std::vector<int> concatenacao;
6
7      concatenacao.insert(concatenacao.end(), vec1.begin(), vec1.end());
8      concatenacao.insert(concatenacao.end(), vec2.begin(), vec2.end());
9
10     return concatenacao;
11 }
12
13 int main() {
14     std::vector<int> input_A;
15     std::vector<int> input_B;
16     std::vector<int> ba;
17     int n, x;
18
19     std::cout << "Insira o tamanho da sequencia A: ";
20     std::cin >> n;
21     std::cout << "Insira os " << n << " elementos da sequencia A: ";
22     while (n > 0) {
23         std::cin >> x;
24         input_A.push_back(x);
25         --n;
26     }
27     std::cout << "Insira o tamanho da sequencia B: ";
28     std::cin >> n;
29     std::cout << "Insira os " << n << " elementos da sequencia B: ";
30     while (n > 0) {
31         std::cin >> x;
32         input_B.push_back(x);
33         --n;
34     }
35     ba = concatenar(input_B, input_A);
36
37     std::cout << ba.size() << " elementos BA: ";
38     for (auto i: ba) std::cout << i << " ";
39 }
```

`[4:11]` - `concatenar` faz a inserção dos vetores `vec1` e `vec2` em um vetor vazio `concatenacao`. Note que `vec1` é sempre inserido antes de `vec2`, logo a ordem dos vetores na chamada da função importa (linha `[35]`).

3.4. **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 << " ";
}
```

`[4:12]` - `segmentar` recebe a sequência de entrada e os limites do segmento. O segmento vazio é declarado com um tamanho $b - a + 1$. São declarados dois iteratores, que apontam para início (iterador `begin()`, apontando para o primeiro elemento da entrada + a) e fim (iterador `begin()`, apontando para o primeiro elemento da entrada, + b + 1) do segmento, respectivamente. Assim, os elementos que estão entre os limites indicados pelos iteradores são copiados para o segmento vazio.

3.5. **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 << " ";
}
```

`[4:12]` - Semelhante à solução anterior, os elementos que estão entre os limites apontados pelos iteradores de início (iterador `begin()`, apontando para o primeiro elemento da entrada) e fim (iterador `begin()`, apontando para o primeiro elemento da entrada + k) são copiados para um vetor `prefixo` de tamanho k.

3.6. **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.**

```cpp
#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 << " ";
}
```

`[4:12]` - Para encontrar o início do sufixo, o iterador `begin()` é acrescido da diferença entre o tamanho da entrada pelo tamanho do sufixo. O iterador `end()` aponta para o último elemento da sequência. Assim, os elementos entre os dois iteradores são copiados para o vetor `sufixo`.

Atividade aula 5

5.1. **Utilizando as propriedades do somatório, mostre que:**

$$\sum_{k=1}^{n}(x_{k+1} - x_k) = x_{n+1} - x_1$$

Através da propriedade associativa temos:

$$\sum_{k=1}^{n}(x_{k+1} - x_k) = \sum_{k=1}^{n} x_{k+1} - \sum_{k=1}^{n} x_k$$

Podemos incrementar os índices dos limites do primeiro somatório, ao decrementar na mesma proporção o índice da fórmula do mesmo.

$$\sum_{k=1}^{n}(x_{k+1} - x_k) = \sum_{k=2}^{n+1} x_k - \sum_{k=1}^{n} x_k$$

Podemos retirar as parcelas mais próximas dos extremos de um somatório. Vamos retirar a última parcela do primeiro e a primeira parcela do segundo.

$$\sum_{k=1}^{n}(x_{k+1} - x_k) = x_{n+1} + \sum_{k=2}^{n} x_k - \left(x_1 + \sum_{k=2}^{n} x_k\right)$$

Simplificando, obtemos a relação dada, como se queria demonstrar:

$$\sum_{k=1}^{n}(x_{k+1} - x_k) = x_{n+1} + \sum_{k=2}^{n} x_k - x_1 - \sum_{k=2}^{n} x_k$$

$$\sum_{k=1}^{n}(x_{k+1} - x_k) = x_{n+1} - x_1$$

5.2. **Utilizando as propriedades do somatório, mostre que:**

$$\sum_{k=1}^{n} k(k+1) = \frac{n(n+1)(n+2)}{3}$$

Expandindo o somatório original:

$$\sum_{k=1}^{n} k(k+1) = \sum_{k=1}^{n} k^2 + k$$

Usando a propriedade associativa:

$$\sum_{k=1}^{n} k(k+1) = \sum_{k=1}^{n} k^2 + \sum_{k=1}^{n} k$$

Ambos os somatórios são conhecidos, e ao substituirmos:

$$\sum_{k=1}^{n} k(k+1) = \frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2}$$

Simplificando, obtemos a relação dada, como se queria demonstrar:

$$\sum_{k=1}^{n} k(k+1) = \frac{(n^2+n)(2n+1)}{6} + \frac{n^2+n}{2}$$

$$\sum_{k=1}^{n} k(k+1) = \frac{2n^3+n^2+2n^2+n}{6} + \frac{n^2+n}{2}$$

$$\sum_{k=1}^{n} k(k+1) = \frac{2n^3+n^2+2n^2+n+3n^2+3n}{6}$$

$$\sum_{k=1}^{n} k(k+1) = \frac{2n^3+6n^2+4n}{6}$$

$$\sum_{k=1}^{n} k(k+1) = \frac{n^3+3n^2+2n}{3}$$

$$\sum_{k=1}^{n} k(k+1) = \frac{n(n^2+3n+2)}{3}$$

$$\sum_{k=1}^{n} k(k+1) = \frac{n(n+1)(n+2)}{3}$$

5.3. **Utilizando as propriedades do somatório, mostre que:**

$$\sum_{k=0}^{n-1} 2^k = 2^n - 1$$

O somatório original é uma parcela de um somatório já conhecido, $\sum_{k=0}^{n} 2^k$. Note que o somatório dado compreende as somas entre $k = 0$ e $k = n - 1$, restando apenas a adição da última parcela, 2^n, para obtermos o somatório de 2^k, mencionado acima.

$$\sum_{k=0}^{n-1} 2^k = \left(\sum_{k=0}^{n} 2^k\right) - 2^n$$

Substituindo $\sum_{k=0}^{n} 2^k = 2^{n+1} - 1$:

$$\sum_{k=0}^{n-1} 2^k = 2^{n+1} - 1 - 2^n$$

Simplificando, obtemos a relação dada, como se queria demonstrar:

$$\sum_{k=0}^{n-1} 2^k = (2^n 2^1) - 1 - 2^n$$

$$\sum_{k=0}^{n-1} 2^k = 2^n(2-1) - 1$$

$$\sum_{k=0}^{n-1} 2^k = 2^n - 1$$

5.4. **Utilizando as propriedades do somatório, mostre que:**

$$\sum_{k=1}^{n} k2^{k-1} = 2^n(n-1) + 1$$

Observe que: $2^{k-1} = 2^k - 2^{k-1}$
Analisando a expressão do somatório, é possível observar que a mesma é a derivada de outro somatório conhecido, $\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$:

$$\sum_{k=1}^{n} k2^{k-1} = \frac{d}{dx}\left(\sum_{k=0}^{n} x^k\right) = \frac{d}{dx}\left(\frac{x^{n+1}-1}{x-1}\right)$$

Tomando apenas as duas últimas igualdades, podemos generalizar essas relações:

$$\frac{d}{dx}\left(\sum_{k=0}^{n} x^k\right) = \frac{d}{dx}\left(\frac{x^{n+1}-1}{x-1}\right)$$

Resolvendo ambas as derivadas, obtemos um expressão geral para qualquer valor de x:

$$\sum_{k=1}^{n} kx^{k-1} = \frac{[(x^{n+1}-1)'(x-1)] - [(x^{n+1}-1)(x-1)']}{(x-1)^2}$$

$$\sum_{k=1}^{n} kx^{k-1} = \frac{(n+1)x^n(x-1) - (x^{n+1}-1)}{(x-1)^2}$$

$$\sum_{k=1}^{n} kx^{k-1} = \frac{(n+1)x^n(x-1) - x^{n+1}+1}{(x-1)^2}$$

Tomando $x = 2$, obtemos a relação dada, como se queria demonstrar:

$$\sum_{k=1}^{n} k2^{k-1} = \frac{(n+1)2^n(2-1) - 2^{n+1}+1}{(2-1)^2}$$

$$\sum_{k=1}^{n} k2^{k-1} = 2^n(n+1) - 2^{n+1} + 1$$

$$\sum_{k=1}^{n} k2^{k-1} = 2^n(n+1) - 2^n 2^1 + 1$$

$$\sum_{k=1}^{n} k2^{k-1} = 2^n(n+1-2) + 1$$

$$\sum_{k=1}^{n} k2^{k-1} = 2^n(n-1) + 1$$

5.5. **Utilizando as propriedades do somatório, mostre que:**

$$\sum_{k=1}^{100}(3-2k)^2 = 1293700$$

Expandindo a fórmula do somatório:

$$\sum_{k=1}^{100}(3-2k)^2 = \sum_{k=1}^{100} 4k^2 - 12k + 9$$

Pela propriedade da distributividade:

$$\sum_{k=1}^{100}(3-2k)^2 = \sum_{k=1}^{100} 4k^2 - \sum_{k=1}^{100} 12k + \sum_{k=1}^{100} 9$$

Retirando as constantes dos somatórios:

$$\sum_{k=1}^{100}(3-2k)^2 = \left(4\sum_{k=1}^{100} k^2\right) - \left(12\sum_{k=1}^{100} k\right) + \left(9\sum_{k=1}^{100}\right)$$

Substituindo os somatórios já conhecidos:

$$\sum_{k=1}^{100}(3-2k)^2 = \left(4\frac{n(n+1)(2n+1)}{6}\right) - \left(12\frac{n(n+1)}{2}\right) + (9\cdot n)$$

Substituindo $n = 100$, obtemos a relação dada, como se queria demonstrar:

$$\sum_{k=1}^{100}(3-2k)^2 = \left(4\frac{100(100+1)(2\cdot 100+1)}{6}\right) - \left(12\frac{100(100+1)}{2}\right) + (9\cdot 100)$$

$$\sum_{k=1}^{100}(3-2k)^2 = (4\cdot 338350) - (12\cdot 5050) + (9\cdot 100)$$

$$\sum_{k=1}^{100}(3-2k)^2 = 1353400 - 60600 + 900$$

$$\sum_{k=1}^{100}(3-2k)^2 = 1293700$$

5.6. **Utilizando uma linguagem de programação, codifique o seguinte somatório em que n, x_i e y_i são valores digitados pelo usuário:**

$$\sum_{i=1}^{n} x_i y_i$$

```cpp
#include <iostream>

int main() {
    int n;
    float soma = 0;
    float x_i, y_i;

    std::cout << "Insira o valor de n: ";
    std::cin >> n;

    for (int i = 1; i <= n; ++i) {
        std::cout << "Insira o valor de x_" << i << ": ";
        std::cin >> x_i;
        std::cout << "Insira o valor de y_" << i << ": ";
        std::cin >> y_i;
        soma += (x_i * y_i);
    }

    std::cout << "Valor do somatorio: " << soma;
}
```

5.7. **Utilizando uma linguagem de programação, codifique o seguinte somatório em que n é informado pelo usuário:**

$$\sum_{i=1}^{n} i$$

```cpp
#include <iostream>

int main() {
    int n;
    int soma = 0;

    std::cout << "Insira o valor de n: ";
    std::cin >> n;

    for (int i = 1; i <= n; ++i) {
        soma += i;
    }

    std::cout << "Valor do somatorio: " << soma;
}
```

5.8. **Utilizando uma linguagem de programação, codifique o seguinte somatório em que n e b_i são valores digitados pelo usuário:**

$$\sum_{i=1}^{n} b_i^2$$

```cpp
#include <iostream>

int main() {
    int n;
    float soma = 0;
    float b_i;
```

```
7
8       std::cout << "Insira o valor de n: ";
9       std::cin >> n;
10
11      for (int i = 1; i <= n; ++i) {
12          std::cout << "Insira o valor de x_" << i << ": ";
13          std::cin >> b_i;
14          soma += (b_i * b_i);
15      }
16
17      std::cout << "Valor do somatorio: " << soma;
18  }
```

5.9. **Utilizando uma linguagem de programação, codifique o seguinte somatório em que n é informado pelo usuário:**

$$\sum_{i=0}^{n} 2^i$$

```
1   #include <iostream>
2   #include <cmath>
3
4   int main() {
5       int n;
6       float soma = 0;
7
8       std::cout << "Insira o valor de n: ";
9       std::cin >> n;
10
11      for (int i = 0; i <= n; ++i) {
12          soma += pow(2, i);
13      }
14
15      std::cout << "Valor do somatorio: " << soma;
16  }
```

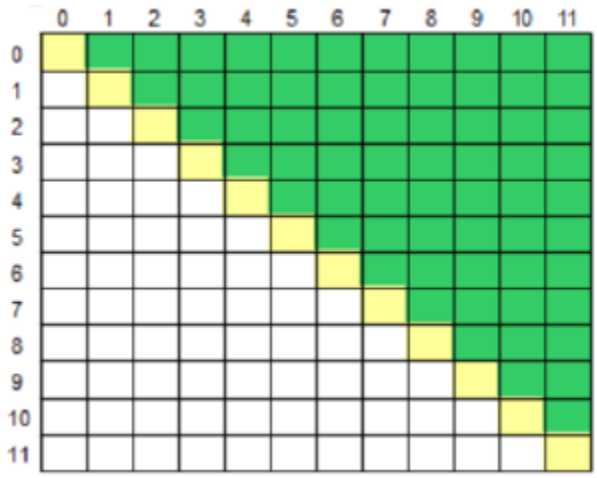
5.10. **Utilizando uma linguagem de programação, codifique o seguinte somatório em que n, x_i e y_i são valores digitados pelo usuário:**

$$\sum_{i=1}^{n} \frac{1}{x_i} + \frac{1}{y_i}$$

```
1   #include <iostream>
2
3   int main() {
4       int n;
5       float soma = 0;
6       float x_i, y_i;
7
8       std::cout << "Insira o valor de n: ";
9       std::cin >> n;
10
11      for (int i = 1; i <= n; ++i) {
12          std::cout << "Insira o valor de x_" << i << ": ";
13          std::cin >> x_i;
14          std::cout << "Insira o valor de y_" << i << ": ";
15          std::cin >> y_i;
16          soma += (1.0/x_i + 1.0/y_i)*n;
17      }
18
19      std::cout << "Valor do somatorio: " << soma;
20  }
```

Atividade aula 7

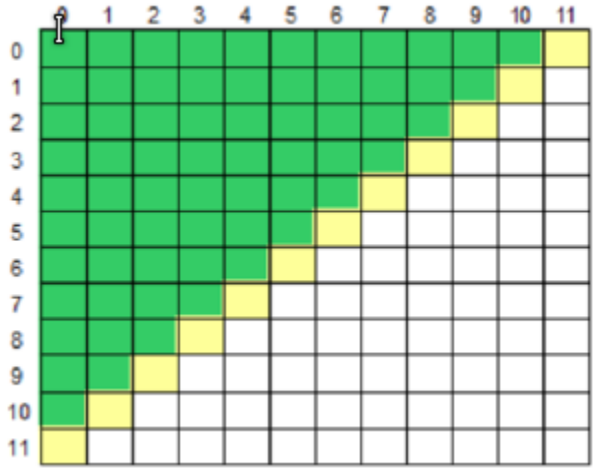
7.1. **Utilizando uma linguagem de programação, codifique um somatório para somar os valores as posições de cor verde da matriz. Os valores dessa posições devem ser informados pelo usuário.**




```cpp
#include <iostream>

void preencher_matriz(int matriz[12][12]) {
    std::cout << "Insira os " << 12*12 << "elementos da matriz: ";
    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            std::cin >> matriz[i][j];
        }
    }
}

int soma_acima_diagonal_principal(int matriz[12][12]) {
    int soma = 0;

    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            if (i < j) {
                soma += matriz[i][j];
            }
        }
    }

    return soma;
}

int main() {
    int matriz[12][12];

    preencher_matriz(matriz);

    std::cout << "\nSoma acima da diagonal principal: " << soma_acima_diagonal_principal(matriz);
}
```

`[12:24]` - A soma dos valores nas posições verdes (acima da diagonal principal) é acumulada sempre que o índice das linhas for menor que o índice das colunas.

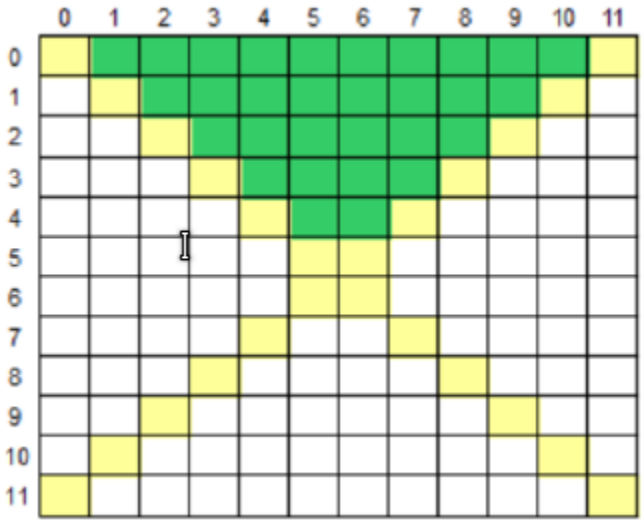
7.2. **Utilizando uma linguagem de programação, codifique um somatório para somar os valores as posições de cor verde da matriz. Os valores dessa posições devem ser informados pelo usuário.**




```cpp
#include <iostream>

void preencher_matriz(int matriz[12][12]) {
    std::cout << "Insira os " << 12*12 << "elementos da matriz: ";
    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            std::cin >> matriz[i][j];
        }
    }
}

int soma_acima_diagonal_secundaria(int matriz[12][12]) {
    int soma = 0;

    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            if (i+j < 11) {
                soma += matriz[i][j];
            }
        }
    }

    return soma;
}

int main() {
    int matriz[12][12];

    preencher_matriz(matriz);

    std::cout << "\nSoma acima da diagonal secundaria: " << soma_acima_diagonal_secundaria(matriz);
}
```

`[12:24]` - A soma dos valores nas posições verdes (acima da diagonal secundária) é acumulada sempre que a soma do índice das linhas com o índice das colunas for menor que o tamanho da matriz -1.

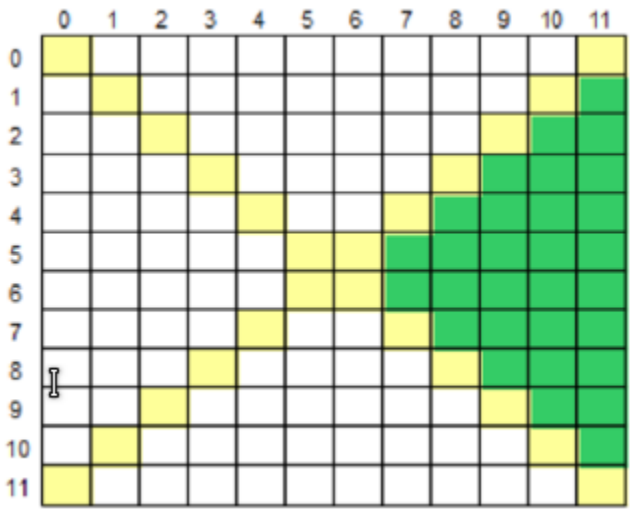
7.3. **Utilizando uma linguagem de programação, codifique um somatório para somar os valores as posições de cor verde da matriz. Os valores dessa posições devem ser informados pelo usuário.**




```cpp
#include <iostream>

void preencher_matriz(int matriz[12][12]){
    std::cout << "Insira os " << 12*12 << "elementos da matriz: ";
    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            std::cin >> matriz[i][j];
        }
    }
}

int soma_acima_ambas_diagonais(int matriz[12][12]){
    int soma = 0;

    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            if (i < j && i+j < 11) {
                soma += matriz[i][j];
            }
        }
    }

    return soma;
}

int main() {
    int matriz[12][12];

    preencher_matriz(matriz);

    std::cout << "\nSoma acima de ambas as diagonais: " << soma_acima_ambas_diagonais(matriz);
}
```

`[12:24]` - A soma dos valores nas posições verdes (acima de ambas as diagonais) é acumulada sempre que os índices forem parte da interseção entre os valores acima da diagonal principal e acima da diagonal secundária.

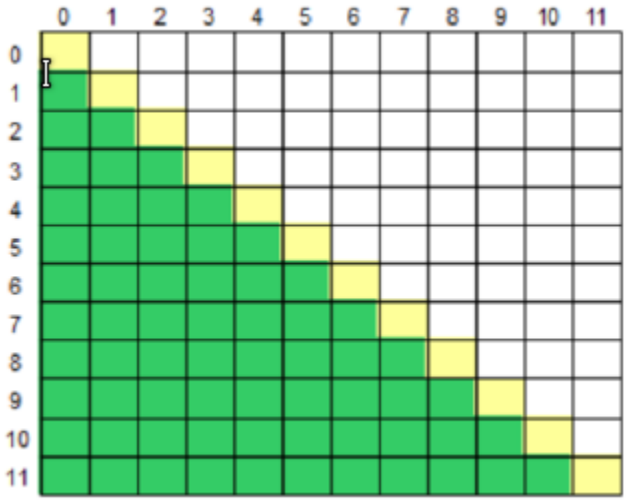
7.4. **Utilizando uma linguagem de programação, codifique um somatório para somar os valores as posições de cor verde da matriz. Os valores dessa posições devem ser informados pelo usuário.**




```cpp
#include <iostream>

void preencher_matriz(int matriz[12][12]){
    std::cout << "Insira os " << 12*12 << "elementos da matriz: ";
    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            std::cin >> matriz[i][j];
        }
    }
}

int soma_entre_diagonais_direita(int matriz[12][12]){
    int soma = 0;

    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            if (i < j && i+j > 11) {
                soma += matriz[i][j];
            }
        }
    }

    return soma;
}

int main() {
    int matriz[12][12];

    preencher_matriz(matriz);

    std::cout << "\nSoma entre as diagonais pela direita: " << soma_entre_diagonais_direita(matriz);
}
```

[12:24] - A soma dos valores nas posições verdes (entre ambas as diagonais, pela direita) é acumulada sempre que os índices forem parte da interseção entre os valores acima da diagonal principal e abaixo da diagonal secundária.

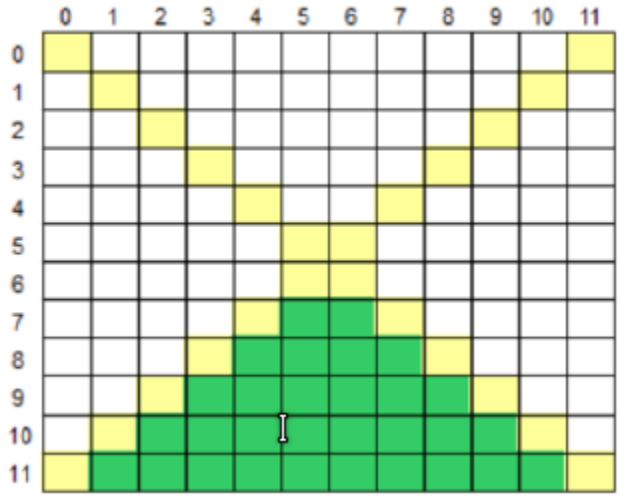
7.5. **Utilizando uma linguagem de programação, codifique um somatório para somar os valores as posições de cor verde da matriz. Os valores dessa posições devem ser informados pelo usuário.**




```cpp
#include <iostream>

void preencher_matriz(int matriz[12][12]){
    std::cout << "Insira os " << 12*12 << "elementos da matriz: ";
    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            std::cin >> matriz[i][j];
        }
    }
}

int soma_abaixo_diagonal_principal(int matriz[12][12]){
    int soma = 0;

    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            if (i > j) {
                soma += matriz[i][j];
            }
        }
    }

    return soma;
}

int main() {
    int matriz[12][12];

    preencher_matriz(matriz);

    std::cout << "\nSoma abaixo da diagonal principal: " << soma_abaixo_diagonal_principal(matriz);
}
```

[12:24] - A soma dos valores nas posições verdes (abaixo da diagonal principal) é acumulada sempre que os índices das linhas for maior que o índice das colunas.

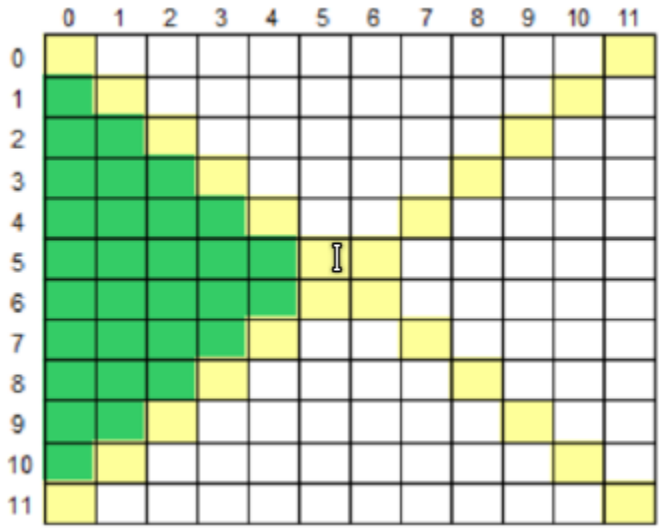
7.6. **Utilizando uma linguagem de programação, codifique um somatório para somar os valores as posições de cor verde da matriz. Os valores dessa posições devem ser informados pelo usuário.**




```cpp
#include <iostream>

void preencher_matriz(int matriz[12][12]) {
    std::cout << "Insira os " << 12*12 << "elementos da matriz: ";
    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            std::cin >> matriz[i][j];
        }
    }
}

int soma_abaixo_ambas_diagonais(int matriz[12][12]) {
    int soma = 0;

    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            if (i > j && i+j > 11) {
                soma += matriz[i][j];
            }
        }
    }

    return soma;
}

int main() {
    int matriz[12][12];

    preencher_matriz(matriz);

    std::cout << "\nSoma abaixo de ambas as diagonais: " << soma_abaixo_ambas_diagonais(matriz);
}
```

[12:24] - A soma dos valores nas posições verdes (abaixo de ambas as diagonais) é acumulada sempre que os índices forem parte da interseção entre os valores abaixo da diagonal principal e abaixo da diagonal secundária.

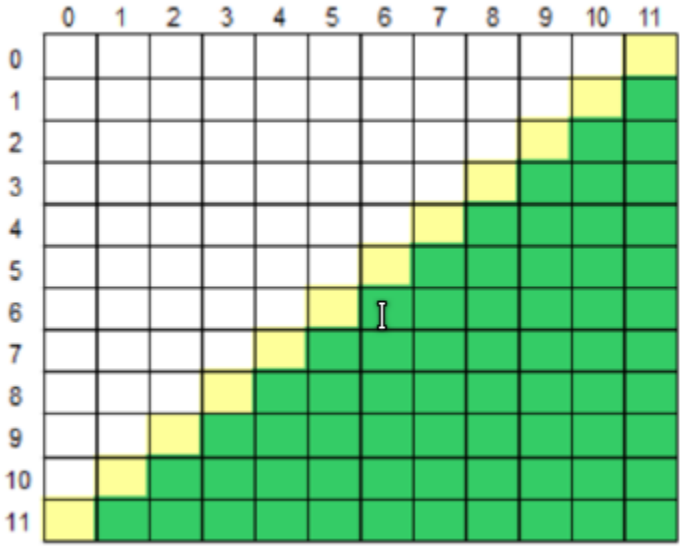
7.7. **Utilizando uma linguagem de programação, codifique um somatório para somar os valores as posições de cor verde da matriz. Os valores dessa posições devem ser informados pelo usuário.**




```cpp
#include <iostream>

void preencher_matriz(int matriz[12][12]){
    std::cout << "Insira os " << 12*12 << "elementos da matriz: ";
    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            std::cin >> matriz[i][j];
        }
    }
}

int soma_entre_diagonais_esquerda(int matriz[12][12]){
    int soma = 0;

    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            if (i > j && i+j < 11) {
                soma += matriz[i][j];
            }
        }
    }

    return soma;
}

int main() {
    int matriz[12][12];

    preencher_matriz(matriz);

    std::cout << "\nSoma entre as diagonais pela esquerda: " << soma_entre_diagonais_esquerda(matriz);
}
```

[12:24] - A soma dos valores nas posições verdes (entre ambas as diagonais, pela esquerda) é acumulada sempre que os índices forem parte da interseção entre os valores abaixo da diagonal principal e acima da diagonal secundária.

7.8. **Utilizando uma linguagem de programação, codifique um somatório para somar os valores as posições de cor verde da matriz. Os valores dessa posições devem ser informados pelo usuário.**




```cpp
#include <iostream>

void preencher_matriz(int matriz[12][12]){
    std::cout << "Insira os " << 12*12 << "elementos da matriz: ";
    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            std::cin >> matriz[i][j];
        }
    }
}

int soma_abaixo_diagonal_secundaria(int matriz[12][12]){
    int soma = 0;

    for (int i = 0; i < 12; ++i) {
        for (int j = 0; j < 12; ++j) {
            if (i+j > 11) {
                soma += matriz[i][j];
            }
        }
    }

    return soma;
}

int main() {
    int matriz[12][12];

    preencher_matriz(matriz);

    std::cout << "\nSoma abaixo da diagonal secundaria: " << soma_abaixo_diagonal_secundaria(matriz);
}
```

[12:24] - A soma dos valores nas posições verdes (abaixo da diagonal secundária) é acumulada sempre que a soma do índice das linhas com o índice das colunas for maior que o tamanho da matriz -1.