PLP-Lab-Aula 3 - Material Complementar

EXERCÍCIO 1 - Um professor deseja calcular a **média ponderada** de um aluno com base em três notas e seus respectivos pesos. O programa deve solicitar as três notas e os três pesos correspondentes, calcular a média ponderada e exibir o resultado formatado com duas casas decimais.

A fórmula da **média ponderada** é:

$$Média = \frac{(nota1 \times peso1) + (nota2 \times peso2) + (nota3 \times peso3)}{peso1 + peso2 + peso3}$$

Escreva o código correspondente abaixo:

RESPOSTA

```
#include <iostream>
#include <iomanip> // Para formatar a saída

using namespace std;

int main() {
   double nota1, nota2, nota3;
   double peso1, peso2, peso3;

   // Entrada de dados
   cout << "Digite a primeira nota e seu peso: ";
   cin >> nota1 >> peso1;

   cout << "Digite a segunda nota e seu peso: ";
   cin >> nota2 >> peso2;

   cout << "Digite a terceira nota e seu peso: ";
   cin >> nota3 >> peso3;

   // Cálculo da média ponderada
   double media = ((nota1 * peso1) + (nota2 * peso2) + (nota3 * peso3)) / (peso1 + peso2 + peso3);
```

```
    // Exibição do resultado formatado
    cout << fixed << setprecision(2);
    cout << "A média ponderada do aluno é: " << media << endl;

    return 0;
}
```

EXERCÍCIO 2 - Escreva um programa em C++ que solicite ao usuário um número inteiro positivo e verifique se ele é um **número primo**. Um número é considerado primo se for **maior que 1** e **divisível apenas por 1 e por ele mesmo**.

O programa deve exibir uma mensagem informando se o número é primo ou não. Escreva o código correspondente abaixo:

RESPOSTA

```
#include <iostream>

using namespace std;

int main() {
    int num;
    bool ehPrimo = true;

    // Entrada do número
    cout << "Digite um numero inteiro positivo: ";
    cin >> num;

    // Verificação de número primo
    if (num <= 1) {
        ehPrimo = false;
    } else {
        for (int i = 2; i < num; i++) {
            if (num % i == 0) {
                ehPrimo = false;
                break;
```

```
        }
      }
    }

    // Exibição do resultado
    if (ehPrimo) {
      cout << num << " é um número primo." << endl;
    } else {
      cout << num << " não é um número primo." << endl;
    }

    return 0;
}
```

EXERCÍCIO 3 – Escreva um programa em C++ que solicite ao usuário um número inteiro positivo e calcule o fatorial desse número. O fatorial de um número n (n!) é definido como o produto de todos os inteiros positivos de 1 até n.

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 1$$

O programa deve exibir o resultado do fatorial.

Exemplo de Entrada e Saída:

Digite um número inteiro positivo: 5

O fatorial de 5 é: 120

RESPOSTA

```
#include <iostream>

using namespace std;

int main() {
    int num;
    unsigned long long fatorial = 1; // Variável para armazenar o resultado
```

```
    // Entrada do número
    cout << "Digite um número inteiro positivo: ";
    cin >> num;

    // Validação de entrada
    if (num < 0) {
        cout << "O fatorial não é definido para números negativos." << endl;
    } else {
        // Cálculo do fatorial
        for (int i = 1; i <= num; i++) {
            fatorial *= i;
        }

        // Exibição do resultado
        cout << "O fatorial de " << num << " é: " << fatorial << endl;
    }

    return 0;
}
```

EXERCÍCIO 4 – Escreva um programa em C++ que solicite ao usuário uma temperatura em graus Celsius e a converta para Fahrenheit e Kelvin.

As fórmulas de conversão são:

$$F = (C \times 9/5) + 32$$

$$K = C + 273.15$$

O programa deve exibir os valores convertidos com duas casas decimais.

Exemplo de Entrada e Saída:

Digite a temperatura em Celsius: 25

Em Fahrenheit: 77.00

Em Kelvin: 298.15

RESPOSTA

```
#include <iostream>
```

```
#include <iomanip> // Para formatar a saída

using namespace std;

int main() {
   double celsius, fahrenheit, kelvin;

   // Entrada do usuário
   cout << "Digite a temperatura em graus Celsius: ";
   cin >> celsius;

   // Conversão para Fahrenheit e Kelvin
   fahrenheit = (celsius * 9.0 / 5.0) + 32;
   kelvin = celsius + 273.15;

   // Exibição dos resultados formatados
   cout << fixed << setprecision(2);
   cout << "Em Fahrenheit: " << fahrenheit << endl;
   cout << "Em Kelvin: " << kelvin << endl;

   return 0;
}
```

EXERCÍCIO 5 – Escreva um programa em C++ que solicite ao usuário uma frase e conte o número de vogais (a, e, i, o, u) presentes na string. O programa deve considerar tanto maiúsculas quanto minúsculas e exibir o total de vogais encontradas.

Exemplo de Entrada e Saída:

Digite uma frase: Programação é incrível!

Número de vogais na frase: 10

RESPOSTA

```
#include <iostream>
#include <string> // Para manipulação de strings

using namespace std;
```

```
int main() {
   string frase;
   int contadorVogais = 0;

   // Entrada da frase
   cout << "Digite uma frase: ";
   getline(cin, frase); // Lê a linha inteira, incluindo espaços

   // Percorre a string para contar vogais
   for (char c : frase) {
      c = tolower(c); // Converte para minúscula para facilitar a verificação
      if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {
         contadorVogais++;
      }
   }

   // Exibição do resultado
   cout << "Número de vogais na frase: " << contadorVogais << endl;

   return 0;
}
```

EXERCÍCIO 6 – Problema da Mochila (Knapsack 0/1) usando Programação Dinâmica

Uma empresa de importação precisa otimizar o carregamento de um container. Cada produto tem um **peso** e um **valor** de mercado, e o container tem uma capacidade máxima de carga. O objetivo é selecionar quais produtos devem ser colocados no container para **maximizar o valor total**, sem ultrapassar a capacidade máxima de carga.

Você deve implementar um programa em **C++ usando programação dinâmica** para resolver esse problema.

Entrada:

- Um número inteiro **N** representando a quantidade de produtos disponíveis.
- Um número inteiro **W** representando a capacidade máxima do container (peso máximo suportado).

- Dois vetores de tamanho **N**, onde:
    - O primeiro vetor contém os **pesos dos produtos**.
    - O segundo vetor contém os **valores dos produtos**.

Saída:

- O **maior valor possível** que pode ser obtido sem ultrapassar a capacidade do container.

📌 Exemplo de Entrada e Saída

Entrada:
Digite o número de produtos: 4
Digite a capacidade máxima do container: 7
Digite os pesos dos produtos: 1 3 4 5
Digite os valores dos produtos: 10 40 50 70

Saída esperada:
O maior valor possível que pode ser carregado é: 90
(Corresponde aos itens de peso 3 e 4, que totalizam peso 7 e valor 90)

O **problema da mochila 0/1** (**0/1 Knapsack Problem**) é um problema clássico de **otimização combinatória** na ciência da computação. Ele é chamado assim porque simula a escolha de itens para colocar em uma **mochila (knapsack)**, onde cada item tem um **peso** e um **valor**, e o objetivo é maximizar o valor total sem ultrapassar o limite de peso.

📌 Por que "0/1"?

O "0/1" no nome significa que **cada item pode ser incluído (1) ou não (0)** na mochila. Não podemos pegar um item **parcialmente**, como em outros tipos de problemas de mochila (exemplo: **mochila fracionária**, onde podemos pegar frações de um item).

📌 Exemplo

Imagine que temos os seguintes itens:

Item	Peso	Valor
1	2 kg	$3
2	3 kg	$4
3	4 kg	$5
4	5 kg	$8

Se a **mochila suporta até 5 kg**, quais itens devemos escolher para **maximizar o valor total**?

Opções Possíveis

1. Levar **apenas o item 4** (peso 5 kg, valor **$8**).

2. Levar **os itens 1 e 3** (peso 2 + 3 = 5 kg, valor **$3 + $4 = $7**).
3. Levar **os itens 2 e 3** (peso 3 + 2 = 5 kg, valor **$4 + $3 = $7**).

🔷 **A melhor solução é escolher o item 4 sozinho, que dá o maior valor ($8).**

📌 **Como Resolver?**

O problema da mochila 0/1 pode ser resolvido de várias formas, como: ✅ **Força bruta** (testando todas as combinações possíveis, mas é ineficiente). ✅ **Programação dinâmica** (mais eficiente, resolve em tempo **O(N × W)**, onde N é o número de itens e W a capacidade da mochila).

A solução dada a seguir, e implementada com o código C++, utiliza **programação dinâmica** para resolver o problema, garantindo encontrar a melhor solução sem testar todas as combinações possíveis de forma ineficiente.

📌 **Resumo:** O **knapsack 0/1** é um problema onde cada item pode ser **totalmente incluído ou excluído**, e o objetivo é maximizar o valor total respeitando uma capacidade de peso limitada. Ele é amplamente usado em **logística, finanças, inteligência artificial** e outros domínios de otimização. 🚀

RESPOSTA

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// Função para resolver o problema da Mochila 0/1 usando Programação Dinâmica
int knapsack(int capacidade, vector<int>& pesos, vector<int>& valores, int N) {
  // Criamos uma matriz dp[N+1][capacidade+1] inicializada com 0
  vector<vector<int>> dp(N + 1, vector<int>(capacidade + 1, 0));

  // Preenchendo a matriz
  for (int i = 1; i <= N; i++) {
    for (int w = 1; w <= capacidade; w++) {
      if (pesos[i - 1] <= w) {
        // Escolhemos entre pegar ou não o item atual
        dp[i][w] = max(valores[i - 1] + dp[i - 1][w - pesos[i - 1]], dp[i - 1][w]);
```

```
            } else {
                // Se o item não cabe, mantemos o valor anterior
                dp[i][w] = dp[i - 1][w];
            }
        }
    }
    return dp[N][capacidade];
}

int main() {
    int N, capacidade;

    // Entrada do número de produtos e capacidade máxima
    cout << "Digite o número de produtos: ";
    cin >> N;

    cout << "Digite a capacidade máxima do container: ";
    cin >> capacidade;

    vector<int> pesos(N), valores(N);

    // Entrada dos pesos dos produtos
    cout << "Digite os pesos dos produtos: ";
    for (int i = 0; i < N; i++) {
        cin >> pesos[i];
    }

    // Entrada dos valores dos produtos
    cout << "Digite os valores dos produtos: ";
    for (int i = 0; i < N; i++) {
        cin >> valores[i];
    }
```

```
    // Chamada da função Knapsack
    int resultado = knapsack(capacidade, pesos, valores, N);

    // Exibição do resultado
    cout << "O maior valor possível que pode ser carregado é: " << resultado << endl;

    return 0;
}
```