Parte I

Estruturas de Dados e Manipulação em C++

Introdução

A linguagem de programação C++ é uma das mais utilizadas no desenvolvimento de software moderno, destacando-se por sua flexibilidade e eficiência. Uma parte fundamental da programação em C++ é o entendimento de estruturas de dados, que permitem armazenar e manipular informações de maneira eficiente. Este trabalho abordará quatro tópicos essenciais: vetores, registros, ponteiros e ficheiros, explicando suas definições, sintaxes, usos práticos e suas inter-relações. Também serão incluídas fichas de exercícios para cada tópico, juntamente com suas soluções.

---

1. Vetores

Definição e Conceito

Vetores são coleções de elementos do mesmo tipo, acessíveis por meio de índices. Eles são usados para armazenar listas de dados de forma organizada.

Declaração e Inicialização

A declaração de um vetor em C++ é feita da seguinte forma:

```
int notas[5]; // Vetor de inteiros com 5 elementos
```

A inicialização pode ser realizada durante a declaração:

```
int notas[5] = {9, 7, 8, 6, 10};
```

Acesso e Manipulação de Elementos

Os elementos de um vetor podem ser acessados e manipulados usando seus índices. Por exemplo:

```
int primeiraNota = notas[0]; // Acesso à primeira nota
notas[1] = 10; // Modificando a segunda nota
```

Exercícios sobre Vetores

Exercício 1

Crie um vetor que armazene 10 números inteiros e calcule a soma de todos os elementos.

Solução:

```
#include <iostream>

int main() {
    int numeros[10], soma = 0;

    // Entrada de dados
    for (int i = 0; i < 10; i++) {
        std::cout << "Digite o número " << (i + 1) << ": ";
        std::cin >> numeros[i];
    }

    // Cálculo da soma
    for (int i = 0; i < 10; i++) {
        soma += numeros[i];
    }

    std::cout << "A soma dos números é: " << soma << std::endl;
    return 0;
}
```

Exercício 2

Escreva um programa que encontre o maior elemento de um vetor de inteiros de tamanho 5.

Solução:

```
#include <iostream>

int main() {
    int numeros[5], maior;

    // Entrada de dados
    for (int i = 0; i < 5; i++) {
        std::cout << "Digite o número " << (i + 1) << ": ";
        std::cin >> numeros[i];
    }

    maior = numeros[0]; // Inicializa com o primeiro elemento

    // Encontrar o maior elemento
    for (int i = 1; i < 5; i++) {
        if (numeros[i] > maior) {
            maior = numeros[i];
        }
    }

    std::cout << "O maior número é: " << maior << std::endl;
    return 0;
}
```

---

2. Registros (Structs)

Definição e Conceito

Registros, ou structs, permitem agrupar diferentes tipos de dados sob um único nome, representando entidades complexas.

Sintaxe de Declaração

A declaração de um registro é feita da seguinte forma:

```
struct Aluno {
    std::string nome;
    int idade;
    float nota;
};
```

Uso de Registros para Modelar Objetos Complexos

Registros são ideais para modelar dados que possuem múltiplos atributos. Por exemplo, um sistema de cadastro de alunos.

Exercícios sobre Registros

Exercício 1

Crie um registro chamado `Produto` que armazene o nome, preço e quantidade de um produto. Em seguida, escreva um programa que permita ao usuário inserir os dados de um produto e imprima as informações.

Solução:

```
#include <iostream>
#include <string>

struct Produto {
    std::string nome;
    float preco;
    int quantidade;
};

int main() {
    Produto produto;

    // Entrada de dados
    std::cout << "Digite o nome do produto: ";
    std::cin >> produto.nome;
    std::cout << "Digite o preço do produto: ";
    std::cin >> produto.preco;
    std::cout << "Digite a quantidade do produto: ";
    std::cin >> produto.quantidade;

    // Saída de dados
    std::cout << "Produto: " << produto.nome << ", Preço: " <<
produto.preco << ", Quantidade: " << produto.quantidade << std::endl;
    return 0;
}
```

Exercício 2

Escreva um programa que utilize um registro `Aluno` e uma função para imprimir as informações do aluno.

Solução:

```
#include <iostream>
#include <string>

struct Aluno {
    std::string nome;
    int idade;
    float nota;
};

void imprimirAluno(const Aluno& aluno) {
    std::cout << "Nome: " << aluno.nome << ", Idade: " << aluno.idade << ",
Nota: " << aluno.nota << std::endl;
}

int main() {
    Aluno aluno;

    // Entrada de dados
    std::cout << "Digite o nome do aluno: ";
    std::cin >> aluno.nome;
    std::cout << "Digite a idade do aluno: ";
    std::cin >> aluno.idade;
    std::cout << "Digite a nota do aluno: ";
    std::cin >> aluno.nota;

    // Imprimir informações do aluno
    imprimirAluno(aluno);
    return 0;
}
```

---

3. Ponteiros

Definição e Conceito

Ponteiros são variáveis que armazenam endereços de memória de outras variáveis, permitindo manipulação direta da memória.

Declaração e Inicialização de Ponteiros

Para declarar um ponteiro, usamos:

```
int x = 10;
int* p = &x; // p armazena o endereço de x
```

Operações Básicas com Ponteiros

Os ponteiros permitem acesso e modificação de dados:

```
std::cout << *p; // Imprime o valor de x
```

```
*p = 20; // Modifica o valor de x
```

Exercícios sobre Ponteiros

Exercício 1

Crie um programa que declare um inteiro, utilize um ponteiro para armazenar seu endereço e imprima o valor original e o valor modificado.

Solução:

```
#include <iostream>

int main() {
    int x = 10;
    int* p = &x;

    std::cout << "Valor original: " << x << std::endl;
    *p = 20; // Modificando o valor através do ponteiro
    std::cout << "Valor modificado: " << x << std::endl;

    return 0;
}
```

Exercício 2

Escreva uma função que receba um ponteiro para um vetor de inteiros e o tamanho do vetor, e que imprima os elementos do vetor.

Solução:

```
#include <iostream>

void imprimirVetor(int* vetor, int tamanho) {
    for (int i = 0; i < tamanho; i++) {
        std::cout << vetor[i] << " ";
    }
    std::cout << std::endl;
}

int main() {
    int numeros[5] = {1, 2, 3, 4, 5};
    imprimirVetor(numeros, 5); // Passando o vetor e seu tamanho

    return 0;
}
```

---

4. Ficheiros

Definição e Importância dos Ficheiros

Ficheiros são usados para armazenar dados de forma permanente, permitindo que os dados sejam lidos e escritos fora da memória do programa.

Manipulação de Ficheiros em C++

C++ oferece suporte a operações de leitura e escrita em ficheiros usando bibliotecas como `<fstream>`:

```
#include <fstream>
```

Exercícios sobre Ficheiros

Exercício 1

Escreva um programa que crie um ficheiro chamado "notas.txt" e grave três notas de alunos.

Solução:

```
#include <iostream>
#include <fstream>

int main() {
    std::ofstream outfile("notas.txt");

    if (outfile.is_open()) {
        outfile << "Nota 1: 8.5\n";
        outfile << "Nota 2: 9.0\n";
        outfile << "Nota 3: 7.5\n";
        outfile.close();
    } else {
        std::cerr << "Erro ao abrir o ficheiro!" << std::endl;
    }

    return 0;
}
```

Exercício 2

Escreva um programa que leia dados de um ficheiro "notas.txt" e imprima os dados no console.

Parte II

Desafio de Projeto: Sistema de Gerenciamento

Data de entrega: ***ultima semana de aulas***

Objetivo:

Desenvolver um sistema que resolva um problema real ou fictício utilizando C++. Os alunos devem aplicar os conceitos de registros, ficheiros, vetores e ponteiros, tendo liberdade para escolher o problema e o domínio da aplicação.

Diretrizes do Projeto:

1. **Escolha do Problema:**
    - Os alunos podem escolher um problema de qualquer área, como:
        - **Gerenciamento de Biblioteca** (como mencionado anteriormente)
        - **Controle de Estoque** de produtos
        - **Sistema de Registro de Alunos** em uma escola
        - **Gerenciamento de Tarefas** em um aplicativo de produtividade
        - **Registro de Eventos** para um calendário
        - **Simulação de Jogo** (ex: jogo de cartas ou tabuleiro)
2. **Requisitos do Sistema:** Cada projeto deve incluir os seguintes elementos:
    - **Estruturas de Dados:**
        - Definição de registros (`structs`) para armazenar dados relevantes ao problema escolhido. Por exemplo:
            - Para o gerenciamento de biblioteca: `Livro`, `Usuario`
            - Para controle de estoque: `Produto`, `Fornecedor`
            - Para registro de alunos: `Aluno`, `Disciplina`
    - **Uso de Vetores:**
        - Implementação de vetores dinâmicos para armazenar coleções de registros.
    - **Uso de Ponteiros:**
        - Manipulação de ponteiros para trabalhar com dados, como listas encadeadas ou referências a registros.
    - **Leitura e Escrita em Arquivos:**

        - Implementação de funcionalidades para salvar e carregar dados em arquivos, permitindo a persistência das informações entre execuções do programa.

3. **Funcionalidades:**
    - As funcionalidades devem ser adaptadas ao problema escolhido, mas devem incluir:
        - **Cadastro de Dados:** Permitir que o usuário insira novas informações.
        - **Listagem de Dados:** Mostrar informações cadastradas.
        - **Atualização de Dados:** Permitir modificar registros existentes.
        - **Exclusão de Dados:** Permitir a remoção de registros.
        - **Salvar e Carregar Dados:** Implementar leitura e gravação em arquivos.
4. **Interface do Usuário:**
    - Criação de um menu interativo para que o usuário escolha as operações desejadas. A interface pode ser baseada em linha de comando.
5. **Validações:**
    - Implementação de validações para entradas do usuário e operações, garantindo que o sistema funcione corretamente mesmo com entradas inesperadas.

Fichas de Exercícios:

Para cada seção do projeto, forneça fichas de exercícios que envolvam:

- Criação e manipulação de vetores e registros.
- Implementação de funções que operem sobre ponteiros.
- Leitura e gravação de arquivos.

Sugestões de Implementação:

- **Documentação:** Os alunos devem documentar seu projeto, incluindo a descrição do problema escolhido, a estrutura do código, e as decisões tomadas durante o desenvolvimento.
- **Critérios de Avaliação:**
    - Funcionalidade: O sistema atende aos requisitos especificados?
    - Estrutura do Código: O código é organizado e segue boas práticas?
    - Criatividade: O aluno demonstrou inovação na escolha do problema e na implementação da solução?
    - Apresentação: O projeto é bem documentado e apresentado?