Aula 08 – Introdução a pilha

1) Introdução

Uma **pilha** (ou stack, em inglês) é uma estrutura de dados que segue o princípio **LIFO** (*Last In, First Out*), ou seja, o último elemento a ser inserido é o primeiro a ser removido. Imagine uma pilha de pratos: você coloca um prato em cima do outro, e o último prato que você colocou será o primeiro a ser retirado.

Características de uma pilha:

- **Push**: Adiciona um elemento no topo da pilha.
- **Pop**: Remove o elemento no topo da pilha.
- **Top (Peek)**: Acessa o elemento no topo sem removê-lo.
- **Empty**: Verifica se a pilha está vazia.

Onde as pilhas são usadas:

1. **Funções recursivas**: Toda vez que uma função é chamada, seu estado é salvo em uma pilha até que a função termine. Isso é usado para controlar a execução.
2. **Undo/Redo**: Em editores de texto, por exemplo, as ações são empilhadas para permitir desfazer (undo) e refazer (redo) ações.
3. **Análise de expressões**: Pilhas são usadas para avaliar expressões matemáticas ou converter entre notações (como infixa para pós-fixa).
4. **Controle de parênteses**: Usada para verificar se parênteses ou chaves em expressões estão balanceados.

Exemplo de código em C++ usando pilha

```
#include <iostream>
#include <stack>  // Incluímos a biblioteca de pilha

int main() {

    // Criando uma pilha de inteiros
    std::stack<int> pilha;

    // Inserindo elementos na pilha (push)
    pilha.push(10);
    pilha.push(20);
    pilha.push(30);

    // Mostrando o elemento no topo (top)
    std::cout << "Elemento no topo da pilha: " << pilha.top() << std::endl;

    // Removendo o elemento do topo (pop)
    pilha.pop();

    // Mostrando o novo elemento no topo
    std::cout << "Elemento no topo da pilha após o pop: " << pilha.top() << std::endl;

    // Verificando se a pilha está vazia
    if (pilha.empty()) {
        std::cout << "A pilha está vazia!" << std::endl;
    } else {
        std::cout << "A pilha não está vazia!" << std::endl;
    }

    return 0;
}
```

Explicação do código:

- Usamos a biblioteca stack da STL, que já implementa todas as operações básicas de uma pilha.
- Adicionamos três números à pilha com push().
- Utilizamos top() para acessar o último elemento adicionado (30, no caso).
- Após o comando pop(), o elemento no topo é removido, e o próximo elemento (20) se torna o topo.
- Por fim, verificamos se a pilha está vazia com empty().

2) Fazendo uma pilha manualmente

A implementação manual de uma pilha envolve a criação de uma estrutura de dados que simula o comportamento de uma pilha (com as operações de **push**, **pop**, **top**, etc.)

```
#include <iostream>

// Tamanho máximo da pilha
#define MAX 100

class Pilha {
private:

    // Índice do topo da pilha
    int topo;

    // Array para armazenar os elementos
    int itens[MAX];

public:
    // Construtor para inicializar a pilha
    Pilha() {
        topo = -1; // Indica que a pilha está vazia
    }

    // Método para verificar se a pilha está cheia
    bool cheia() {
        return topo == MAX - 1;
    }

    // Método para verificar se a pilha está vazia
    bool vazia() {
        return topo == -1;
    }

    // Método para inserir um elemento na pilha (push)
    void push(int valor) {
        if (cheia()) {
            std::cout << "Erro: Pilha cheia, não é possível adicionar o elemento!" << std::endl;
        } else {
            topo++;
            itens[topo] = valor;
        }
    }

    // Método para remover o elemento do topo da pilha (pop)
    int pop() {
        if (vazia()) {
            std::cout << "Erro: Pilha vazia, não é possível remover o elemento!" << std::endl;
```

```
            return -1; // Retorna um valor indicativo de erro
        } else {
            int valor = itens[topo];
            topo--;
            return valor;
        }
    }

    // Método para visualizar o elemento do topo (sem remover)
    int getTopo() {
        if (vazia()) {
            std::cout << "Erro: Pilha vazia!" << std::endl;
            return -1; // Retorna um valor indicativo de erro
        } else {
            return itens[topo];
        }
    }
};

int main() {

    // Criando uma instância da classe Pilha
    Pilha minhaPilha;

    int quantidade, valor;

    std::cout << "Quantos valores você deseja inserir na pilha? ";
    std::cin >> quantidade;

    // Inserindo elementos na pilha
    for (int i = 0; i < quantidade; i++) {
        std::cout << "Digite o valor " << i + 1 << ": ";
        std::cin >> valor;
        minhaPilha.push(valor);
    }

    std::cout << "\nElementos removidos da pilha em ordem (LIFO):" << std::endl;

    // Removendo e exibindo os elementos da pilha (em ordem inversa de inserção)
    while (!minhaPilha.vazia()) {
        std::cout << minhaPilha.pop() << std::endl;
    }

    return 0;
}
```

Explicação do código

1. **Classe Pilha**:

- Os atributos `topo` e `itens` agora são membros privados da classe.
- Os métodos foram implementados como membros públicos da classe, permitindo que as operações sejam realizadas de maneira encapsulada.

2. **Métodos**:
    - Os métodos cheia(), vazia(), push(), pop() e getTopo() foram definidos como métodos da classe Pilha, permitindo que o código que utiliza a classe interaja com a pilha de forma mais intuitiva.

3. **Construtor**:
    - Um construtor foi adicionado para inicializar o atributo `topo` no momento da criação da instância da classe.

4. **Uso da classe no `main`**:
    - O objeto minhaPilha é uma instância da classe Pilha, e as operações de pilha são realizadas chamando os métodos dessa instância.

3) Considerações da aula

Nesta aula tivemos o primeiro contato com herança. Nas próximas aulas aprofundaremos o estudo para tornar este conceito bem sedimentado.

Bons estudos.