- Raul Costa

- Pedro Oliveira

➔ *Usando STL*

```cpp
#include <iostream>
#include <map>

using namespace std;

int main() {
    map<string, int> idade;

    idade["Pedro"] = 25;
    idade["João"] = 30;
    idade["Maria"] = 22;

    // Obtendo o tamanho do mapa
    cout << "O mapa contém " << idade.size() << "
elementos." << endl;

    return 0;
}
```

➔ Sem usar STL

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

using namespace std;

// Definindo o tipo de dado para chave e valor
struct Node {
    string chave;
    int valor;
    Node* prox;
```

```
    // Construtor
    Node(const string& chave, int valor) : chave(chave),
valor(valor), prox(nullptr) {}
};

class Map {
private:
    Node* head;  // Ponteiro para o início da lista
encadeada

public:
    // Construtor
    Map() : head(nullptr) {}

    // Destrutor
    ~Map() {
        while (head != nullptr) {
            Node* temp = head;
            head = head->prox;
            delete temp;
        }
    }

    // Adiciona chave-valor (sem duplicar chave)
    void insert(const string& chave, int valor) {
        // Se a lista estiver vazia
        if (head == nullptr) {
            head = new Node(chave, valor);
            return;
        }

        // Se a chave já existir, apenas atualize o valor
        Node* current = head;
        while (current != nullptr) {
            if (current->chave == chave) {
                current->valor = valor;
                return;
            }
            current = current->prox;
        }
```

```
        // Se não encontrou a chave, insira um novo nó
ordenado
        Node* novo = new Node(chave, valor);
        if (chave < head->chave) {  // Inserir no início
            novo->prox = head;
            head = novo;
            return;
        }

        // Procurando pela posição certa para inserir
        current = head;
        while (current->prox != nullptr && current->prox-
>chave < chave) {
            current = current->prox;
        }

        // Inserir o novo nó depois de current
        novo->prox = current->prox;
        current->prox = novo;
    }

    // Busca um valor pela chave
    bool search(const string& chave, int& valor) {
        Node* current = head;
        while (current != nullptr) {
            if (current->chave == chave) {
                valor = current->valor;
                return true;
            }
            current = current->prox;
        }
        return false;  // Chave não encontrada
    }

    // Remove um par chave-valor
    void remove(const string& chave) {
        if (head == nullptr) return;

        // Se a chave estiver no início
```

```
        if (head->chave == chave) {
            Node* temp = head;
            head = head->prox;
            delete temp;
            return;
        }

        // Procurar pela chave
        Node* current = head;
        while (current->prox != nullptr && current->prox-
>chave != chave) {
            current = current->prox;
        }

        // Se a chave foi encontrada
        if (current->prox != nullptr) {
            Node* temp = current->prox;
            current->prox = current->prox->prox;
            delete temp;
        }
    }

    // Retorna o tamanho do mapa
    int size() const {
        int count = 0;
        Node* current = head;
        while (current != nullptr) {
            count++;
            current = current->prox;
        }
        return count;
    }

    // Imprime o mapa
    void print() const {
        Node* current = head;
        while (current != nullptr) {
            cout << current->chave << " => " << current-
>valor << endl;
            current = current->prox;
```

```
        }
    }
};

int main() {
    Map Mapa;

    // Inserir pares chave-valor
    Mapa.insert("Pedro", 25);
    Mapa.insert("João", 30);
    Mapa.insert("Maria", 22);

    // Exibir o conteúdo do mapa
    cout << "Conteúdo do mapa:" << endl;
    Mapa.print();

    // Buscar um valor
    int idade;
    if (Mapa.search("João", idade)) {
        cout << "Idade de João: " << idade << endl;
    }

    // Remover uma chave
    Mapa.remove("Pedro");

    // Exibir o mapa após remoção
    cout << "Após remoção de Pedro:" << endl;
    Mapa.print();

    // Tamanho do mapa
    cout << "Tamanho do mapa: " << Mapa.size() << endl;

    return 0;
}
```

Utilizando STL, podemos aproveitar os contêiners e os algoritmos contidos nele, por isso, utilizá-lo torna o programa mais rápido de ser desenvolvido e mais legível. Sem STL, temos que criar do zero classes para armazenar tipos específicos de dados – nesse caso, utilizamos a classe Map -, assim como suas funções,

aumentando muito o tempo de desenvolvimento e tamanho do código, o que torna a manutenção mais difícil e piora a eficiência do programa.