📚 Glossário Completo de C++ para Iniciantes

Este glossário foi criado especialmente para quem deseja começar a aprender **C++** e ainda não tem familiaridade com a linguagem. Explicações simples e exemplos práticos acompanham cada conceito para facilitar o seu entendimento.

📌 Variáveis

São usadas para armazenar valores na memória.

```
int idade = 25;
float altura = 1.75;
char letra = 'A';
```

📌 Constantes

Não podem ser alteradas após serem definidas.

```
const double PI = 3.1415;
```

📌 Tipos de Dados

Tipos primitivos mais usados em C++:

- `int` — números inteiros
- `float` — números decimais
- `double` — números decimais (mais preciso)
- `char` — caractere único
- `bool` — verdadeiro ou falso
- `string` — texto (*precisa incluir o cabeçalho* `<string>`)

```
#include <string>
string nome = "Lucas";
```

📌 Operadores

Operações matemáticas, relacionais e lógicas.

```
int resultado = 5 + 3;
bool condicao = (10 > 5) && (5 < 8);
```

📌 Entrada e Saída

Entrada com `cin` e saída com `cout`.

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

int idade;
cout << "Digite sua idade: ";
cin >> idade;
cout << "Idade: " << idade;
```

📌 Condicionais

Permite executar blocos diferentes dependendo de condições.

```
if (idade >= 18) {
    cout << "Maior de idade";
} else {
    cout << "Menor de idade";
}
```

📌 Loops

Estruturas para repetir comandos.

```
for (int i = 0; i < 5; i++) {
    cout << i << endl;
}
```

📌 Arrays

Armazenam vários valores do mesmo tipo.

```
int numeros[] = {1, 2, 3, 4, 5};
```

📌 Vetores Dinâmicos (vector)

Coleção flexível de elementos (*necessita* `<vector>`).

```
#include <vector>
vector<int> numeros = {10, 20, 30};
```

📌 Funções

Blocos de código reutilizáveis.

```
int soma(int a, int b) {
    return a + b;
}
```

📌 Função Principal

Todo programa em C++ começa pela função `main()`.

```
int main() {
    cout << "Olá, mundo!";
    return 0;
}
```

📌 Classes

Fundamento da programação orientada a objetos em C++.

```
class Pessoa {
public:
    string nome;

    void falar() {
        cout << "Olá, meu nome é " << nome << endl;
    }
};
```

📌 Objetos

Instâncias de uma classe.

```
Pessoa p;
p.nome = "João";
p.falar();
```

📌 Herança

Permite criar uma nova classe baseada em outra.

```
class Estudante : public Pessoa {
public:
    string curso;
};
```

📌 Encapsulamento

Controla o acesso aos atributos e métodos.

```
class Conta {
private:
    double saldo;
public:
    void setSaldo(double valor) {
        saldo = valor;
    }
    double getSaldo() {
        return saldo;
    }
};
```

📌 Ponteiros

Armazenam o endereço de memória de uma variável.

```
int num = 10;
int *ptr = &num;
cout << *ptr; // Acessa o valor através do ponteiro
```

📌 Referências

Apontam para o mesmo espaço de memória de outra variável.

```
int numero = 5;
int &ref = numero;
ref = 10; // numero agora também vale 10
```

📌 Struct

Estruturas para agrupar vários tipos de dados.

```
struct Carro {
    string modelo;
    int ano;
};

Carro c1 = {"Fusca", 1970};
```

📌 Templates

Permitem criar funções ou classes genéricas.

```
template <typename T>
T maior(T a, T b) {
    return (a > b) ? a : b;
}
```

📌 Namespaces

Evitam conflitos entre nomes de funções ou variáveis.

```
namespace exemplo {
    int valor = 10;
}
```

📌 Try-Catch (Tratamento de Exceções)

Permite capturar e tratar erros durante a execução.

```
try {
    throw 404;
} catch (int erro) {
    cout << "Erro capturado: " << erro;
}
```

📌 Bibliotecas Importantes

- `<iostream>` — entrada e saída
- `<string>` — manipulação de texto
- `<vector>` — vetores dinâmicos
- `<cmath>` — funções matemáticas
- `<algorithm>` — algoritmos úteis (ex.: sort)