**Técnico(a) Especialista em Tecnologias e Programação de Sistemas de Informação**

Sistema de Aprendizagem (APZ)

5412 - Programação de computadores - estruturada

|  |  |
| --- | --- |
| **FORMADOR/A** | **DATA** |
| Luis Cardoso | março de 2024 |

**Índice**

[Configurar o Visual Studio Code para compilar C++ 5](#_Toc161040971)

[Programação de computadores – estruturada 5](#_Toc161040972)

[1. Sequência 6](#_Toc161040973)

[2. Seleção 7](#_Toc161040974)

[3. Iteração 7](#_Toc161040975)

[4. Modularidade 7](#_Toc161040976)

[5. Estruturas de Dados 7](#_Toc161040977)

[6. Transparência de Código 7](#_Toc161040978)

[Exemplos em C++ 8](#_Toc161040979)

[1. Sequência 8](#_Toc161040980)

[2. Seleção 8](#_Toc161040981)

[3. Iteração 9](#_Toc161040982)

[**4. Modularidade** 10](#_Toc161040983)

[5. Estruturas de Dados 11](#_Toc161040984)

[Conceitos básicos Computadores, programas, dados e instruções 12](#_Toc161040985)

[1. Computadores 12](#_Toc161040986)

[2. Programas 13](#_Toc161040987)

[3. Dados 13](#_Toc161040988)

[4. Instruções 13](#_Toc161040989)

[Exemplo em C++ 13](#_Toc161040990)

[Estrutura de um programa 14](#_Toc161040991)

[Exemplo de um Programa C++ 15](#_Toc161040992)

[Operações básicas de entrada e de saída 16](#_Toc161040993)

[Operações de Saída: 16](#_Toc161040994)

[Operações de Entrada: 17](#_Toc161040995)

[Entrada e Saída de Strings: 17](#_Toc161040996)

[Compilação de um programa 18](#_Toc161040997)

[Primeiro exemplo 20](#_Toc161040998)

[Passo 1: Escrever o Programa 20](#_Toc161040999)

[Passo 2: Compilar o Programa 20](#_Toc161041000)

[Passo 3: Executar o Programa 20](#_Toc161041001)

[A linguagem C++ 21](#_Toc161041002)

[Sintaxe da linguagem 22](#_Toc161041003)

[Estrutura Básica de um Programa em C++ 22](#_Toc161041004)

[Inclusão de Bibliotecas 22](#_Toc161041005)

[Espaço de Nomes 23](#_Toc161041006)

[Declaração de Variáveis 23](#_Toc161041007)

[Funções 23](#_Toc161041008)

[Controle de Fluxo 23](#_Toc161041009)

[Comentários 24](#_Toc161041010)

[Ambiente de desenvolvimento 24](#_Toc161041011)

[1. Editor de Código ou IDE 24](#_Toc161041012)

[2. Compilador C++ 25](#_Toc161041013)

[3. Sistema de Build 25](#_Toc161041014)

[4. Depurador 25](#_Toc161041015)

[5. Sistema de Controle de Versão 26](#_Toc161041016)

[6. Bibliotecas e Gerenciadores de Dependências 26](#_Toc161041017)

[Constantes e variáveis 26](#_Toc161041018)

[Variáveis 26](#_Toc161041019)

[Constantes 27](#_Toc161041020)

[Exemplo de Uso 27](#_Toc161041021)

[Operadores 28](#_Toc161041022)

[Operadores Aritméticos 28](#_Toc161041023)

[Operadores de Comparação 28](#_Toc161041024)

[Operadores Lógicos 28](#_Toc161041025)

[Operadores de Atribuição 29](#_Toc161041026)

[Operadores de Incremento e Decremento 29](#_Toc161041027)

[Operadores Bit a Bit 29](#_Toc161041028)

[Operadores de Acesso a Membros 29](#_Toc161041029)

[Operador Ternário 29](#_Toc161041030)

[Tipos de dados 30](#_Toc161041031)

[Tipos de Dados Fundamentais 30](#_Toc161041032)

[Tipos de Dados Modificados 30](#_Toc161041033)

[Tipos de Dados Derivados 30](#_Toc161041034)

[Tipos de Dados Definidos pelo Usuário 30](#_Toc161041035)

[Exemplos de Declaração de Variáveis 31](#_Toc161041036)

[Exemplo 1: Cálculo da Média de Números 32](#_Toc161041037)

[Exemplo 2: Uso de Structs para Armazenar Informações 33](#_Toc161041038)

[Exemplo 3: Uso de Funções 33](#_Toc161041039)

[Exemplo 4: Uso Básico de Classes e Objetos 34](#_Toc161041040)

[Entrada e saída de dados 35](#_Toc161041041)

[Exemplo 1: Entrada e Saída Básica 35](#_Toc161041042)

[Exemplo 2: Trabalhando com Strings 36](#_Toc161041043)

[Exemplo 3: Formatação de Saída 36](#_Toc161041044)

[Exemplo 4: Entrada e Saída de Dados em Arquivos 37](#_Toc161041045)

[Instruções básicas 38](#_Toc161041046)

[Declaração de Variáveis 38](#_Toc161041047)

[Operações Aritméticas 38](#_Toc161041048)

[Estruturas de Controle de Fluxo 39](#_Toc161041049)

[Entrada e Saída 40](#_Toc161041050)

[Estruturas de controlo 41](#_Toc161041051)

[1. Estruturas de Decisão 41](#_Toc161041052)

[2. Estruturas de Loop 42](#_Toc161041053)

[3. Controle de Loop 43](#_Toc161041054)

[Exemplo 1: Uso de if-else 43](#_Toc161041055)

[Exemplo 2: Uso de switch-case 44](#_Toc161041056)

[Exemplo 3: Uso de for loop 45](#_Toc161041057)

[Exemplo 4: Uso de while loop 45](#_Toc161041058)

[Exemplo 5: Uso de do-while loop 46](#_Toc161041059)

Configurar o Visual Studio Code para compilar C++

<https://www.youtube.com/watch?v=DMWD7wfhgNY> – forma mais direta

<https://code.visualstudio.com/docs/languages/cpp>

<https://code.visualstudio.com/docs/cpp/config-mingw#_prerequisites>

Programação de computadores – estruturada

A programação de computadores estruturada é um paradigma de programação destinado a melhorar a clareza, qualidade e tempo de desenvolvimento de um programa de computador, utilizando exclusivamente subrotinas, blocos de código (como decisões e laços) e funções com retorno. Esse paradigma enfatiza a importância de dividir o programa em funções ou procedimentos menores para facilitar a compreensão e a manutenção do código.

Aqui estão alguns conceitos fundamentais da programação estruturada:

1. **Sequência**: A execução das instruções acontece de forma linear, uma após a outra, na ordem em que são escritas no programa.
2. **Seleção**: Instruções condicionais (como if, else, switch) permitem que o programa tome decisões e siga caminhos diferentes de execução com base em condições lógicas.
3. **Iteração**: Laços de repetição (como for, while, do-while) possibilitam a execução repetida de um bloco de código até que uma condição seja satisfeita.
4. **Modularidade**: O programa é dividido em módulos ou funções menores, cada um realizando uma tarefa específica. Isso facilita o entendimento individual de cada parte do programa e a reutilização de código.
5. **Estruturas de dados**: Utiliza-se de arrays, registros (structs), e outros tipos de dados agregados para organizar e manipular dados de forma eficiente.
6. **Transparência de código**: O código deve ser escrito de forma clara e compreensível, facilitando a leitura e manutenção por outros programadores.

Para exemplificar a programação estruturada em C++, considere um programa simples que calcula a média de três números:

#include <iostream>

using namespace std;

// Função para calcular a média de três números

double calcularMedia(double num1, double num2, double num3) {

return (num1 + num2 + num3) / 3.0;

}

int main() {

double a, b, c;

cout << "Digite três números: ";

cin >> a >> b >> c;

double media = calcularMedia(a, b, c);

cout << "A média é: " << media << endl;

return 0;

}

Neste exemplo, o programa é estruturado em uma função principal (**main**) que solicita a entrada do usuário e uma função auxiliar (**calcularMedia**) que realiza o cálculo da média. Isso demonstra a modularidade e a clareza do código na programação estruturada.

Claro, vamos aprofundar um pouco mais sobre os conceitos da programação estruturada e como eles são aplicados no desenvolvimento de software:

### 1. **Sequência**

Na programação estruturada, a execução sequencial de instruções é a base. Cada linha de código é processada uma após a outra, em ordem. Isso é fundamental para o fluxo de controle do programa, pois determina como as operações são realizadas e em que sequência. Por exemplo, em um programa que calcula a soma de números, primeiro você inicializa variáveis, depois lê os números, calcula a soma e, finalmente, exibe o resultado.

### 2. **Seleção**

As estruturas de seleção permitem que o programa tome decisões e escolha diferentes caminhos de execução. O uso de instruções **if**, **else** e **switch** permite que o programa reaja a diferentes condições e execute blocos de código específicos. Por exemplo, se você está desenvolvendo um programa que categoriza a idade de uma pessoa, você usará a seleção para determinar se a pessoa é um adulto, adolescente ou criança, com base em sua idade.

### 3. **Iteração**

Os laços de repetição são essenciais para realizar tarefas repetidas sem a necessidade de escrever o mesmo código várias vezes. **for**, **while**, e **do-while** são estruturas de iteração comuns que permitem executar um bloco de código repetidamente até que uma condição específica seja atendida. Por exemplo, se você deseja calcular a soma de todos os números de 1 a 100, você pode usar um laço para iterar sobre esses números e acumular a soma.

### 4. **Modularidade**

A modularidade é a chave para construir programas complexos de forma gerenciável. Ao dividir o programa em funções ou módulos menores, você pode focar em partes menores e mais gerenciáveis do problema. Cada módulo pode ser desenvolvido, testado e depurado de forma independente, aumentando a qualidade do software e reduzindo a complexidade. No exemplo do cálculo da média, a função **calcularMedia** é um módulo que pode ser reutilizado em diferentes partes do programa ou em outros programas.

### 5. **Estruturas de Dados**

Na programação estruturada, também é crucial escolher as estruturas de dados adequadas para representar e manipular informações. Arrays e registros (ou structs) são frequentemente usados para agrupar e organizar dados relacionados. Por exemplo, se você estiver desenvolvendo um sistema para gerenciar informações de estudantes, poderá usar uma struct para representar cada estudante e um array para armazenar todos os estudantes.

### 6. **Transparência de Código**

Escrever código que seja fácil de entender e manter é um aspecto fundamental da programação estruturada. Isso não apenas facilita a colaboração entre programadores, mas também simplifica a manutenção e a extensão do software ao longo do tempo. Comentários, nomes de variáveis ​​significativos e uma estrutura clara de código são práticas importantes para alcançar a transparência do código.

Implementar esses princípios na programação estruturada leva a um código mais robusto, fácil de entender e manter, o que é crucial para o sucesso de qualquer projeto de desenvolvimento de software.

Exemplos em C++

Claro, vamos explorar exemplos em C++ que demonstram os conceitos de programação estruturada que discutimos:

### 1. **Sequência**

Aqui está um exemplo simples que demonstra a execução sequencial das instruções em C++:

#include <iostream>

using namespace std;

int main() {

int a = 5;

int b = 10;

int soma = a + b;

cout << "A soma é: " << soma << endl;

return 0;

}

Neste programa, as instruções são executadas uma após a outra: inicialização de variáveis, cálculo da soma e exibição do resultado.

### 2. **Seleção**

Este exemplo usa uma estrutura de seleção para decidir qual mensagem exibir com base na idade do usuário:

#include <iostream>

using namespace std;

int main() {

int idade;

cout << "Digite sua idade: ";

cin >> idade;

if (idade >= 18) {

cout << "Você é adulto." << endl;

} else {

cout << "Você é menor de idade." << endl;

}

return 0;

}

Aqui, o programa usa a instrução **if-else** para determinar se o usuário é adulto ou menor de idade.

### 3. **Iteração**

Este exemplo mostra um loop que calcula a soma dos primeiros 10 números inteiros positivos:

#include <iostream>

using namespace std;

int main() {

int soma = 0;

for (int i = 1; i <= 10; i++) {

soma += i;

}

cout << "A soma dos primeiros 10 números inteiros é: " << soma << endl;

return 0;

}

#include <iostream>

using namespace std;

int main() {

int soma = 0;

for (int i = 1; i <= 10; i++) {

soma += i;

}

cout << "A soma dos primeiros 10 números inteiros é: " << soma << endl;

return 0;

}

O laço **for** é utilizado para iterar de 1 a 10, acumulando a soma dos números.

**4. Modularidade**

Este exemplo divide o programa em duas funções: a principal (**main**) e uma função para calcular a média de dois números:

#include <iostream>

using namespace std;

double calcularMedia(double num1, double num2) {

return (num1 + num2) / 2.0;

}

int main() {

double a, b;

cout << "Digite dois números: ";

cin >> a >> b;

double media = calcularMedia(a, b);

cout << "A média é: " << media << endl;

return 0;

}

A função **calcularMedia** é um módulo separado, que pode ser reutilizado para calcular a média de quaisquer dois números.

### 5. **Estruturas de Dados**

Aqui está um exemplo que usa uma **struct** para representar um estudante e um array para armazenar vários estudantes:

#include <iostream>

#include <string>

using namespace std;

struct Estudante {

string nome;

int idade;

float media;

};

int main() {

Estudante turma[3] = {

{"João", 20, 8.5},

{"Maria", 19, 9.0},

{"Pedro", 21, 7.5}

};

for (int i = 0; i < 3; i++) {

cout << "Estudante: " << turma[i].nome << " - Idade: " << turma[i].idade << " - Média: " << turma[i].media << endl;

}

return 0;

}

Esse código define uma **struct** para um estudante e usa um array para armazenar informações sobre uma turma de estudantes, demonstrando como estruturas de dados podem ser usadas para organizar informações relacionadas.

Esses exemplos cobrem os conceitos básicos da programação estruturada em C++ e demonstram como aplicá-los em programas reais.

Conceitos básicos Computadores, programas, dados e instruções

Claro, vamos detalhar os conceitos básicos de computadores, programas, dados e instruções, especialmente no contexto da programação em C++.

### 1. **Computadores**

Um computador é uma máquina capaz de realizar uma série de tarefas ou operações de forma automática. Ele pode processar dados, executar cálculos, armazenar informações e interagir com outros dispositivos. No coração de um computador, temos a CPU (Unidade Central de Processamento), que executa as instruções dos programas, e a memória, que armazena essas instruções e os dados.

### 2. **Programas**

Um programa é um conjunto de instruções que diz ao computador como realizar uma tarefa específica. Em C++, um programa é geralmente composto por funções, variáveis, e outras declarações escritas seguindo a sintaxe da linguagem. Quando o programa é executado, o compilador ou interpretador converte o código fonte em código de máquina que a CPU pode entender e executar.

### 3. **Dados**

Dados são as informações que os programas processam. Em C++, os dados são representados através de variáveis e constantes, que podem ser de diferentes tipos, como **int** (números inteiros), **double** (números de ponto flutuante), **char** (caracteres), **string** (sequências de caracteres), entre outros. Os programas manipulam esses dados através de operações, atribuições e expressões.

### 4. **Instruções**

As instruções são os comandos que um programa executa. Em C++, existem vários tipos de instruções, como:

* **Declarações de variáveis**: Para armazenar dados.
* **Instruções de atribuição**: Para atribuir valores a variáveis.
* **Estruturas de controle**: Como **if**, **switch**, **for**, **while**, que controlam o fluxo do programa.
* **Chamadas de função**: Que executam tarefas específicas definidas em funções.

### Exemplo em C++

Aqui está um exemplo simples em C++ que ilustra esses conceitos:

#include <iostream>

using namespace std;

// Função principal do programa

int main() {

// Declaração de variáveis (Dados)

int numero1 = 10;

int numero2 = 20;

// Instrução de atribuição

int soma = numero1 + numero2;

// Estrutura de controle e instrução para exibir dados

if (soma > 0) {

cout << "A soma é: " << soma << endl;

}

// Retorna 0 indicando que o programa foi executado com sucesso

return 0;

}

Neste programa, as variáveis **numero1** e **numero2** são declaradas e inicializadas com valores. A variável **soma** é usada para armazenar o resultado da adição desses dois números. O programa verifica se **soma** é maior que 0 e, em caso afirmativo, imprime o resultado. Cada linha do código é uma instrução que o computador executará sequencialmente quando o programa for rodado.

Estrutura de um programa

A estrutura de um programa em C++ é composta por várias partes fundamentais que organizam o código e definem como ele será executado. Um programa C++ típico pode ser dividido nas seguintes partes principais:

1. **Preprocessor Directives (#include)**: As diretivas do pré-processador começam com **#** e são instruções para o pré-processador do C++. Por exemplo, **#include <iostream>** é uma diretiva que diz ao pré-processador para incluir o arquivo de cabeçalho da biblioteca de entrada/saída padrão antes da compilação do programa.
2. **Namespace**: Um namespace em C++ é uma forma de organizar o código e evitar conflitos de nome. **using namespace std;** permite acessar membros do namespace **std** (como **cout** e **cin**) sem precisar prefixá-los com **std::**.
3. **Function Definitions**: As funções são blocos de código que realizam tarefas específicas. Em C++, toda execução de programa começa com a função **main()**. Essa função é obrigatória em todos os programas C++.
4. **Main Function**: A função **main()** é o ponto de entrada de um programa C++. É aqui que a execução do programa começa. A função pode retornar um valor (geralmente **0** para indicar que o programa foi executado com sucesso) e pode aceitar argumentos da linha de comando.
5. **Variable Declarations**: As variáveis ​​são espaços de armazenamento nomeados usados ​​para reter valores. Elas devem ser declaradas com um tipo antes de serem usadas.
6. **Statements and Expressions**: Instruções são as 'ações' que o programa executa, como atribuir valores a variáveis, chamar funções e executar operações, enquanto expressões produzem valores.
7. **Control Structures**: Estruturas de controle dirigem o fluxo de execução do programa. Elas incluem decisões (**if**, **else**, **switch**) e loops (**for**, **while**, **do-while**).
8. **Comments**: Os comentários são usados ​​para explicar o código e são ignorados pelo compilador. Em C++, os comentários de linha única são precedidos por **//**, e os comentários de várias linhas são cercados por **/\*** e **\*/**.

### Exemplo de um Programa C++

#include <iostream> // Inclui a biblioteca de entrada/saída

using namespace std; // Permite usar elementos do namespace std sem prefixo

// Definição da função principal

int main() {

// Declaração de variáveis

int a = 5;

int b = 10;

// Instrução de saída

cout << "A soma de " << a << " e " << b << " é " << (a + b) << endl;

// Retorno de status

return 0;

}

Este programa inclui a biblioteca **iostream**, usa o namespace **std**, declara duas variáveis **a** e **b**, calcula a soma e a exibe, e finalmente retorna 0, indicando a conclusão bem-sucedida do programa.

Operações básicas de entrada e de saída

Em C++, as operações básicas de entrada e saída (E/S) são geralmente realizadas usando o fluxo de entrada padrão (**cin**) para a entrada e o fluxo de saída padrão (**cout**) para a saída. Essas operações são fundamentais para interagir com o usuário, permitindo que o programa receba dados e apresente resultados. Aqui está como você pode realizar essas operações básicas em C++:

### Operações de Saída:

* **cout**: **cout** é usado para enviar dados para a saída padrão (geralmente a tela). Ele é usado junto com o operador de inserção **<<** para enviar a saída.

#include <iostream>

using namespace std;

int main() {

int numero = 10;

cout << "O valor do número é: " << numero << endl;

return 0;

}

No exemplo acima, **cout** exibe a string "O valor do número é: " seguida pelo valor da variável **numero** e, em seguida, uma quebra de linha (**endl**).

### Operações de Entrada:

* **cin**: **cin** é usado para receber dados do usuário através da entrada padrão (geralmente o teclado). Ele é usado junto com o operador de extração **>>** para receber a entrada.

#include <iostream>

using namespace std;

int main() {

int numero;

cout << "Digite um número: ";

cin >> numero;

cout << "O número digitado é: " << numero << endl;

return 0;

}

Neste exemplo, o programa solicita ao usuário que digite um número. O **cin** captura o valor inserido e o armazena na variável **numero**. Depois, o programa exibe o valor armazenado em **numero**.

### Entrada e Saída de Strings:

Para strings, você pode usar **cin** e **cout** de forma similar:

#include <iostream>

#include <string> // Inclui o suporte para o tipo string

using namespace std;

int main() {

string nome;

cout << "Digite seu nome: ";

cin >> nome; // Isso lê a string até o primeiro espaço

cout << "Olá, " << nome << "!" << endl;

return 0;

}

No entanto, se você quiser ler uma linha inteira de texto, incluindo espaços, é melhor usar **getline(cin, string)**:

#include <iostream>

#include <string>

using namespace std;

int main() {

string nomeCompleto;

cout << "Digite seu nome completo: ";

getline(cin, nomeCompleto); // Lê a linha inteira até a quebra de linha

cout << "Olá, " << nomeCompleto << "!" << endl;

return 0;

}

Estes são os conceitos básicos de entrada e saída em C++. Com eles, você pode começar a interagir com o usuário, coletando dados e apresentando informações.

Compilação de um programa

A compilação de um programa é o processo de converter o código-fonte escrito em uma linguagem de programação de alto nível, como C++, para uma linguagem de máquina que o computador pode entender e executar. O processo de compilação em C++ geralmente envolve várias etapas:

1. **Pré-processamento**: Durante esta fase, o pré-processador manipula o código-fonte antes da compilação real. Isso inclui a expansão de diretivas de pré-processador (como **#include** e **#define**), remoção de comentários e realização de substituições macro.
2. **Compilação**: O compilador converte o código-fonte pré-processado em um código intermediário chamado código de objeto. Nesta etapa, o código é analisado (análise léxica, sintática e semântica) e convertido em uma forma intermediária ou código de máquina específico para a plataforma, mas ainda não é um programa executável.
3. **Linkage (Linker)**: O linker pega um ou mais arquivos de código de objeto gerados pelo compilador e os combina em um único arquivo executável. Durante esse processo, o linker resolve referências a funções e variáveis entre os arquivos de objeto e bibliotecas externas.

Para ilustrar, vamos usar um exemplo simples em C++. Suponha que você tenha o seguinte código em um arquivo chamado **main.cpp**:

#include <iostream>

int main() {

std::cout << "Olá, mundo!" << std::endl;

return 0;

}

Aqui estão os passos básicos para compilar este programa usando o GCC, um compilador popular para C++:

1. **Abra um terminal** no seu sistema operacional.
2. **Navegue até o diretório** onde **main.cpp** está localizado.
3. **Execute o comando de compilação**: Para compilar o arquivo **main.cpp** para um executável chamado **programa**, você usaria o seguinte comando no terminal:

g++ main.cpp -o programa

* + **g++** é o comando para chamar o compilador do GCC para C++.
  + **main.cpp** é o arquivo de entrada para o compilador.
  + **-o programa** especifica o nome do arquivo de saída (executável). Se você não usar **-o**, o GCC geralmente produzirá um arquivo executável chamado **a.out** por padrão (em sistemas Unix-like).

1. **Execute o programa**: Após a compilação, você pode executar o programa usando:

./programa

1. Isso deverá exibir "Olá, mundo!" no terminal.

Este é o fluxo básico de compilação e execução de um programa C++ simples em um ambiente Unix-like, usando o GCC como compilador. O processo é semelhante em outros sistemas operacionais e compiladores, embora os comandos específicos possam variar.

Primeiro exemplo

Vamos criar um exemplo simples de um programa em C++ que imprime "Olá, Mundo!" no console. Este exemplo ajudará você a entender como escrever, compilar e executar um programa básico em C++ usando o Visual Studio Code.

### Passo 1: Escrever o Programa

1. Abra o Visual Studio Code.
2. Crie um novo arquivo clicando em **File > New File**.
3. Cole o seguinte código no arquivo:

#include <iostream>

int main() {

std::cout << "Olá, Mundo!" << std::endl;

return 0;

}

1. Salve o arquivo com a extensão **.cpp**, por exemplo, **hello.cpp**.

### Passo 2: Compilar o Programa

Assumindo que você já configurou o arquivo **tasks.json** como descrito anteriormente, você pode compilar o programa da seguinte maneira:

1. Abra a paleta de comandos com **Ctrl+Shift+P**.
2. Digite 'Run Build Task' e selecione-a.
3. Isso compilará seu programa usando a tarefa de compilação definida no **tasks.json**.

Se você não configurou a tarefa de compilação, siga as instruções da mensagem anterior para configurar o **tasks.json** adequadamente.

### Passo 3: Executar o Programa

Depois de compilar o programa, você pode executá-lo diretamente do terminal integrado do VS Code.

1. Abra um novo terminal no VS Code (**Terminal > New Terminal**).
2. Navegue até o diretório onde seu arquivo executável está localizado.
3. No Windows, digite **.\hello.exe** (ou o nome que você deu ao arquivo executável). No Linux ou macOS, digite **./hello** (ou o nome que você deu ao arquivo executável).

Quando você executar o programa, ele deve exibir "Olá, Mundo!" no terminal, demonstrando que seu programa foi escrito, compilado e executado com sucesso.

A linguagem C++

Vamos criar um exemplo simples de programa em C++ para ilustrar o básico da linguagem. O programa que vamos desenvolver será um clássico "Hello, World!", que simplesmente exibe essa mensagem no console. Esse exemplo ajudará você a entender a estrutura básica de um programa em C++, incluindo como definir a função **main**, usar a biblioteca de entrada/saída padrão e compilar e executar o código.

Aqui está o código para o programa "Hello, World!" em C++:

#include <iostream> // Inclui a biblioteca de entrada e saída

// O ponto de entrada principal para o programa

int main() {

// Usa cout para exibir a mensagem "Hello, World!" na tela

std::cout << "Hello, World!" << std::endl;

return 0; // Retorna 0 para indicar que o programa terminou com sucesso

}

Aqui está o que cada parte do programa faz:

* **#include <iostream>**: Esta linha diz ao pré-processador para incluir a biblioteca de entrada/saída padrão (**iostream**) que é usada para operações de entrada e saída, como imprimir texto na tela.
* **int main() { ... }**: Define a função **main**, que é o ponto de entrada de todos os programas em C++. Quando o programa é executado, a execução começa pela função **main**.
* **std::cout << "Hello, World!" << std::endl;**: Esta linha usa o objeto **cout** (parte do namespace **std**) para enviar a string "Hello, World!" para a saída padrão (geralmente a tela). **std::endl** é usado para inserir uma quebra de linha e limpar o buffer de saída.
* **return 0;**: Indica que o programa foi concluído com sucesso. O valor **0** é geralmente usado para indicar uma terminação bem-sucedida.

Para compilar e executar este programa, você precisaria salvar o código em um arquivo com a extensão **.cpp**, como **hello\_world.cpp**, e então usar um compilador C++, como o g++, para compilar o código. Após compilar, você pode executar o arquivo resultante para ver a mensagem "Hello, World!" impressa no console.

Parte superior do formulário

Sintaxe da linguagem

A sintaxe da linguagem C++ é o conjunto de regras que define como escrever instruções que podem ser compreendidas e executadas pelo compilador. Vou detalhar alguns aspectos fundamentais da sintaxe em C++ para lhe dar uma compreensão básica de como programas em C++ são estruturados.

### Estrutura Básica de um Programa em C++

Todo programa em C++ começa com uma função principal chamada **main()**. Essa é a função de entrada do programa, onde a execução começa.

int main() {

// O código vai aqui

return 0;

}

### Inclusão de Bibliotecas

As bibliotecas em C++ fornecem funcionalidades adicionais, como operações de entrada e saída, manipulação de strings, entre outras. Para usar uma biblioteca, você a inclui no início do seu programa com a diretiva **#include**.

#include <iostream> // Permite usar std::cout e std::cin

### Espaço de Nomes

O espaço de nomes **std** é onde muitos dos recursos padrão do C++ estão definidos. Para evitar prefixar elementos do espaço de nomes **std** com **std::**, você pode usar a declaração **using namespace std;**.

using namespace std;

### Declaração de Variáveis

Em C++, cada variável tem um tipo, que define o tamanho e o layout da memória da variável, os valores que ela pode armazenar e as operações que podem ser realizadas sobre ela.

int numero = 10;

float decimal = 5.5;

char letra = 'A';

### Funções

Uma função é um bloco de código que realiza uma tarefa específica. As funções são definidas especificando o tipo de retorno, o nome da função e os parâmetros entre parênteses.

int somar(int a, int b) {

return a + b;

}

### Controle de Fluxo

C++ fornece várias estruturas de controle de fluxo, como condicionais e laços.

* **Condicionais**: **if** e **else** são usados para executar diferentes blocos de código com base em determinadas condições.

if (numero > 5) {

cout << "Número é maior que 5" << endl;

} else {

cout << "Número é 5 ou menor" << endl;

}

**Laços**: **for**, **while** e **do-while** são usados para repetir um bloco de código.

**for (int i = 0; i < 10; i++) {**

**cout << i << " ";**

**}**

### Comentários

Comentários são usados para explicar o código e são ignorados pelo compilador. Comentários de linha única começam com **//**, e comentários de várias linhas são cercados por **/\*** e **\*/**.

**// Este é um comentário de linha única**

**/\***

**Este é um comentário**

**de várias linhas**

**\*/**

Entender esses aspectos fundamentais da sintaxe de C++ ajudará você a começar a escrever programas simples e a se familiarizar com a linguagem.

Ambiente de desenvolvimento

O ambiente de desenvolvimento em C++ consiste em várias ferramentas e componentes que trabalham juntos para facilitar a escrita, compilação e depuração de programas. Aqui estão os principais componentes de um ambiente de desenvolvimento para C++:

### 1. Editor de Código ou IDE

Um editor de código é onde você escreve seu código-fonte. Ele pode ser simples como um editor de texto ou mais avançado como uma IDE (Integrated Development Environment), que oferece recursos adicionais para facilitar o desenvolvimento.

* **Editores de texto**: Ferramentas como VS Code, Sublime Text e Atom são populares entre os desenvolvedores C++ para escrever código. Eles podem ser personalizados com extensões e plug-ins para suportar recursos de desenvolvimento C++.
* **IDEs**: Ambientes de desenvolvimento integrado, como Visual Studio, CLion e Qt Creator, fornecem uma suíte completa de ferramentas para desenvolvimento em C++, incluindo editor de código, compilador, depurador e, às vezes, um designer de interface gráfica. Eles facilitam a visualização de estruturas de projeto, gerenciamento de dependências e depuração.

### 2. Compilador C++

O compilador é a ferramenta que converte seu código-fonte em um programa executável. Os compiladores mais comuns para C++ são:

* **GCC (GNU Compiler Collection)**: Muito usado em ambientes Linux e disponível para outros sistemas operacionais.
* **MSVC (Microsoft Visual C++)**: O compilador padrão no Windows, parte do Visual Studio.
* **Clang**: Conhecido por suas mensagens de erro amigáveis e suporte a modernas funcionalidades de C++.

### 3. Sistema de Build

Um sistema de build automatiza o processo de compilação, vinculação e, às vezes, teste do seu código. Alguns sistemas populares incluem:

* **Make**: Uma das ferramentas de automação de compilação mais antigas, que usa arquivos **Makefile** para definir como construir o programa.
* **CMake**: Um sistema de build multiplataforma que gera arquivos de projeto ou makefiles em várias plataformas.
* **Ninja**: Focado em velocidade, é frequentemente usado em combinação com o CMake.

### 4. Depurador

Um depurador permite que você execute um programa passo a passo, inspecione variáveis e controle o fluxo de execução. Isso é crucial para identificar e corrigir bugs. GDB (GNU Debugger) é um depurador popular em sistemas Linux, enquanto o Visual Studio fornece um depurador integrado no Windows.

### 5. Sistema de Controle de Versão

Embora não exclusivo para o desenvolvimento em C++, um sistema de controle de versão como Git é essencial para gerenciar as mudanças no código, colaborar com outros desenvolvedores e manter um histórico do desenvolvimento do projeto.

### 6. Bibliotecas e Gerenciadores de Dependências

As bibliotecas fornecem código reutilizável para funcionalidades comuns, e os gerenciadores de dependências ajudam a gerenciar e instalar essas bibliotecas. No C++, o Conan é um gerenciador de dependências popular, e o vcpkg da Microsoft também é uma opção no Windows.

Ao configurar seu ambiente de desenvolvimento em C++, você pode escolher as ferramentas que melhor se adaptam ao seu fluxo de trabalho e às necessidades do projeto. A escolha das ferramentas pode depender do sistema operacional, das preferências pessoais e da natureza do projeto que você está desenvolvendo.

Parte superior do formulário

Constantes e variáveis

No contexto da programação em C++, é fundamental entender a diferença entre constantes e variáveis, pois elas são elementos básicos para armazenar dados dentro de um programa.

### Variáveis

Uma variável em C++ é um espaço de armazenamento nomeado na memória do computador onde você pode guardar valores que podem ser modificados durante a execução do programa. Cada variável tem um tipo associado, que determina o tamanho e o layout da memória da variável, os valores que ela pode armazenar e as operações que podem ser realizadas com ela.

Para declarar uma variável em C++, você precisa especificar o tipo da variável seguido pelo nome que deseja dar a ela. Você também pode inicializar a variável com um valor.

int idade = 30; // Declara uma variável do tipo int chamada idade e inicializa com 30

double salario = 4578.50; // Declara uma variável do tipo double chamada salario e inicializa com 4578.50

char letra = 'A'; // Declara uma variável do tipo char chamada letra e inicializa com 'A'

### Constantes

Uma constante é um valor que não pode ser alterado durante a execução do programa. No C++, você pode criar constantes de duas maneiras principais: usando a palavra-chave **const** ou **#define**.

* **Usando const**: Ao declarar uma variável, você pode adicionar a palavra-chave **const** para torná-la uma constante. Isso significa que seu valor não pode ser modificado após a inicialização.

const int MAX\_USUARIOS = 100; // Declara uma constante do tipo int

const double PI = 3.14159; // Declara uma constante do tipo double

**Usando #define**: Você também pode definir constantes usando a diretiva de pré-processamento **#define**, que substitui todas as ocorrências do identificador no código pelo valor correspondente. **#define** não reserva espaço na memória; é simplesmente uma substituição feita pelo pré-processador.

#define MAX\_USUARIOS 100

#define PI 3.14159

É importante destacar que, ao usar **#define**, você não especifica o tipo da constante, e não há verificação de tipo em tempo de compilação. Por isso, o uso de **const** é geralmente preferido por proporcionar segurança de tipo.

### Exemplo de Uso

#include <iostream>

int main() {

const int maxIdade = 100; // Constante

int idade = 25; // Variável

// maxIdade = 150; // Isso geraria um erro de compilação, pois maxIdade é uma constante

idade = 30; // OK: o valor da variável idade pode ser alterado

std::cout << "Idade: " << idade << "\n";

std::cout << "Idade máxima: " << maxIdade << "\n";

return 0;

}

Neste exemplo, **maxIdade** é uma constante e seu valor não pode ser modificado após a inicialização, enquanto **idade** é uma variável cujo valor pode ser alterado. Tentar modificar o valor de **maxIdade** resultaria em um erro de compilação.

Operadores

Operadores em C++ são símbolos especiais ou palavras-chave que dizem ao compilador para realizar operações matemáticas, lógicas ou outras operações em variáveis e valores. Aqui está uma visão geral dos tipos principais de operadores em C++:

### Operadores Aritméticos

Estes operadores são usados para realizar operações matemáticas básicas:

* **Adição (+)**: Soma valores.
* **Subtração (-)**: Subtrai valores.
* **Multiplicação (\*)**: Multiplica valores.
* **Divisão (/)**: Divide valores. A divisão entre inteiros descarta a parte fracionária.
* **Módulo (%)**: Retorna o resto de uma divisão entre inteiros.

### Operadores de Comparação

Estes operadores comparam dois valores e retornam um valor booleano (**true** ou **false**):

* **Igual (==)**: Verifica se dois valores são iguais.
* **Diferente (!=)**: Verifica se dois valores são diferentes.
* **Maior que (>)**: Verifica se o valor à esquerda é maior que o valor à direita.
* **Menor que (<)**: Verifica se o valor à esquerda é menor que o valor à direita.
* **Maior ou igual (>=)**: Verifica se o valor à esquerda é maior ou igual ao valor à direita.
* **Menor ou igual (<=)**: Verifica se o valor à esquerda é menor ou igual ao valor à direita.

### Operadores Lógicos

Usados principalmente em expressões condicionais, eles retornam valores booleanos:

* **AND lógico (&&)**: Retorna **true** se ambos os operandos forem verdadeiros.
* **OR lógico (||)**: Retorna **true** se pelo menos um dos operandos for verdadeiro.
* **NOT lógico (!)**: Inverte o valor booleano do operando.

### Operadores de Atribuição

Estes operadores atribuem valores a variáveis:

* **Atribuição (=)**: Atribui o valor da direita ao operando da esquerda.
* **Atribuição com adição (+=)**: Soma o valor da direita ao valor da esquerda e atribui o resultado à esquerda.
* **Atribuição com subtração (-=)**: Subtrai o valor da direita do valor da esquerda e atribui o resultado à esquerda.
* **Outros operadores de atribuição incluem \*=, /=, e %=**.

### Operadores de Incremento e Decremento

* **Incremento (++)**: Aumenta o valor de uma variável em um.
* **Decremento (--)**: Diminui o valor de uma variável em um.

Estes podem ser prefixados ou sufixados (ex: **++x** ou **x++**), o que pode alterar o momento em que a alteração do valor ocorre em relação ao uso do valor na expressão.

### Operadores Bit a Bit

Estes operadores são usados para manipular dados ao nível de bits:

* **AND bit a bit (&)**: Realiza a operação AND em cada par de bits.
* **OR bit a bit (|)**: Realiza a operação OR em cada par de bits.
* **XOR bit a bit (^)**: Realiza a operação XOR em cada par de bits.
* **NOT bit a bit (~)**: Inverte os bits.
* **Deslocamento à esquerda (<<)**: Desloca os bits para a esquerda.
* **Deslocamento à direita (>>)**: Desloca os bits para a direita.

### Operadores de Acesso a Membros

* **Acesso a membro (.)**: Acessa membros de uma struct ou classe.
* **Acesso a membro através de ponteiro (->)**: Acessa membros de um objeto apontado por um ponteiro.

### Operador Ternário

* **Ternário (? :)**: Retorna um valor baseado em uma condição (**condição ? valor\_se\_verdadeiro : valor\_se\_falso**).

Estes operadores são essenciais para a criação de expressões em C++, permitindo realizar cálculos, fazer comparações e controlar o fluxo do programa.

Tipos de dados

Em C++, os tipos de dados definem a natureza dos valores que uma variável pode armazenar. Aqui estão os principais tipos de dados em C++:

### Tipos de Dados Fundamentais

1. **Char**: Armazena um único caractere (como 'a' ou 'B'). Normalmente tem o tamanho de um byte.
2. **Int**: Usado para armazenar números inteiros. O tamanho pode variar dependendo do sistema, mas geralmente é de 4 bytes em máquinas de 32 ou 64 bits.
3. **Float**: Armazena números de ponto flutuante (números com casas decimais). Normalmente é de 4 bytes e tem precisão de 6 a 7 dígitos decimais.
4. **Double**: Similar ao float, mas com o dobro do tamanho (geralmente 8 bytes) e precisão de aproximadamente 15 dígitos decimais.
5. **Void**: Um tipo especial que representa a ausência de valor. É usado em funções que não retornam um valor.
6. **Bool**: Representa valores booleanos (**true** ou **false**).

### Tipos de Dados Modificados

Estes são modificadores que podem ser aplicados aos tipos de dados fundamentais para alterar seu tamanho ou propriedades:

1. **Signed**: Pode armazenar números positivos e negativos.
2. **Unsigned**: Só pode armazenar números não negativos, aumentando assim o intervalo de valores positivos que podem ser armazenados.
3. **Short**: Reduz o tamanho do tipo de dados (por exemplo, **short int**).
4. **Long**: Aumenta o tamanho do tipo de dados (por exemplo, **long int**).

### Tipos de Dados Derivados

1. **Array**: Uma coleção de elementos de um tipo, armazenados sequencialmente na memória.
2. **Pointer**: Armazena o endereço de memória de outro tipo de dados.
3. **Struct**: Permite combinar diferentes tipos de dados em um único tipo.
4. **Union**: Permite armazenar diferentes tipos de dados no mesmo local de memória.
5. **Enum**: Define um tipo de dados que consiste em um conjunto de constantes nomeadas.

### Tipos de Dados Definidos pelo Usuário

1. **Class**: Define um tipo de dados que pode conter membros de dados e funções membro.
2. **Typedef**: Permite definir um novo nome para um tipo existente.

### Exemplos de Declaração de Variáveis

char letra = 'A';

int idade = 30;

float altura = 1.75;

double precisao = 0.123456789;

bool isOnline = true;

// Array

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

// Pointer

int\* ptr = &idade;

// Struct

struct Pessoa {

std::string nome;

int idade;

};

// Union

union Dados {

int i;

float f;

};

// Enum

enum Cores { VERMELHO, VERDE, AZUL };

Compreender esses tipos de dados é fundamental para o desenvolvimento em C++, pois determina como os dados são armazenados, manipulados e interagem entre si no programa.

### Exemplo 1: Cálculo da Média de Números

Este exemplo demonstra como calcular a média de um conjunto de números armazenados em um array.

#include <iostream>

using namespace std;

int main() {

double numeros[] = {10.5, 20.5, 30.5, 40.5, 50.5};

double soma = 0.0;

int tamanho = sizeof(numeros) / sizeof(numeros[0]);

for (int i = 0; i < tamanho; ++i) {

soma += numeros[i];

}

double media = soma / tamanho;

cout << "A média é: " << media << endl;

return 0;

}

### Exemplo 2: Uso de Structs para Armazenar Informações

Este exemplo mostra como usar **structs** para agrupar diferentes tipos de dados em uma única unidade lógica.

#include <iostream>

#include <string>

using namespace std;

struct Pessoa {

string nome;

int idade;

};

int main() {

Pessoa pessoa1;

pessoa1.nome = "João";

pessoa1.idade = 30;

cout << "Nome: " << pessoa1.nome << ", Idade: " << pessoa1.idade << endl;

return 0;

}

### Exemplo 3: Uso de Funções

Este exemplo ilustra a definição e a chamada de uma função em C++.

#include <iostream>

using namespace std;

// Declaração da função

int somar(int a, int b) {

return a + b;

}

int main() {

int resultado = somar(10, 20);

cout << "A soma é: " << resultado << endl;

return 0;

}

### Exemplo 4: Uso Básico de Classes e Objetos

Este exemplo demonstra a definição de uma classe simples e a criação de um objeto.

#include <iostream>

using namespace std;

class Carro {

public:

string marca;

int ano;

void buzinar() {

cout << "Beep Beep!" << endl;

}

};

int main() {

Carro meuCarro;

meuCarro.marca = "Toyota";

meuCarro.ano = 2020;

cout << meuCarro.marca << " " << meuCarro.ano << endl;

meuCarro.buzinar();

return 0;

}

Esses exemplos cobrem alguns dos conceitos básicos de C++, como o uso de arrays, structs, funções, e classes. Experimentar e modificar esses exemplos pode ser uma ótima maneira de aprender mais sobre a linguagem.

Entrada e saída de dados

A entrada e saída de dados (E/S) em C++ são fundamentais para a interação com o usuário e para o processamento de informações. Vamos explorar como realizar operações básicas de E/S usando os streams **cin** para entrada e **cout** para saída.

### Exemplo 1: Entrada e Saída Básica

#include <iostream>

using namespace std;

int main() {

int numero;

cout << "Digite um número: ";

cin >> numero;

cout << "O número digitado foi: " << numero << endl;

return 0;

}

Neste exemplo, o programa pede ao usuário para digitar um número, lê o número usando **cin** e o exibe de volta ao usuário usando **cout**.

### Exemplo 2: Trabalhando com Strings

#include <iostream>

#include <string> // Necessário para usar objetos do tipo string

using namespace std;

int main() {

string nome;

cout << "Digite seu nome: ";

getline(cin, nome); // Lê uma linha inteira, incluindo espaços

cout << "Olá, " << nome << "!" << endl;

return 0;

}

Neste exemplo, o programa solicita o nome do usuário e lida com a possibilidade de o nome conter espaços, utilizando a função **getline**.

### Exemplo 3: Formatação de Saída

#include <iostream>

#include <iomanip> // Necessário para manipular a formatação de saída

using namespace std;

int main() {

double pi = 3.141592653589793;

cout << "Pi sem formatação: " << pi << endl;

// Definindo a precisão da saída para 2 casas decimais

cout << fixed << setprecision(2);

cout << "Pi com duas casas decimais: " << pi << endl;

return 0;

}

Este exemplo mostra como formatar a saída para limitar o número de casas decimais de um valor de ponto flutuante usando **fixed** e **setprecision** da biblioteca **<iomanip>**.

### Exemplo 4: Entrada e Saída de Dados em Arquivos

#include <iostream>

#include <fstream> // Necessário para manipular arquivos

using namespace std;

int main() {

string linha;

ifstream meuArquivo("exemplo.txt");

if (meuArquivo.is\_open()) {

while (getline(meuArquivo, linha)) {

cout << linha << '\n';

}

meuArquivo.close();

} else {

cout << "Não foi possível abrir o arquivo" << endl;

}

return 0;

}

Este exemplo lê um arquivo chamado **exemplo.txt** e imprime seu conteúdo linha por linha. Ele usa **ifstream** para representar o stream de entrada de um arquivo.

Esses exemplos abrangem os aspectos básicos da entrada e saída de dados em C++, mostrando como ler e escrever tanto no console quanto em arquivos.

Parte superior do formulário

Instruções básicas

As instruções básicas em C++ são os fundamentos que permitem a execução de operações essenciais no seu programa. Vamos discutir algumas das instruções mais fundamentais que você usará regularmente ao escrever programas em C++.

### Declaração de Variáveis

A declaração de variáveis é uma das primeiras instruções que você encontrará. Variáveis são espaços nomeados na memória que podem armazenar dados que podem ser modificados pelo programa.

int numero; // Declaração de uma variável do tipo inteiro

double saldo = 100.0; // Declaração e inicialização de uma variável do tipo double

char letra = 'A'; // Declaração e inicialização de uma variável do tipo char

### Operações Aritméticas

Operações aritméticas são usadas para realizar cálculos matemáticos. Os operadores aritméticos básicos incluem adição (**+**), subtração (**-**), multiplicação (**\***), divisão (**/**) e módulo (**%**).

int soma = 5 + 3; // Adição

int diferenca = 5 - 3; // Subtração

int produto = 5 \* 3; // Multiplicação

int quociente = 5 / 3; // Divisão

int resto = 5 % 3; // Módulo (resto da divisão)

### Estruturas de Controle de Fluxo

As estruturas de controle de fluxo determinam a ordem em que as instruções são executadas.

* **If-Else**: Usado para executar blocos de código com base em uma condição.

if (numero > 0) {

cout << "O número é positivo." << endl;

} else {

cout << "O número é zero ou negativo." << endl;

}

**Switch**: Seleciona um dos muitos blocos de código a ser executado.

switch (numero) {

case 1:

cout << "Número é 1" << endl;

break;

case 2:

cout << "Número é 2" << endl;

break;

default:

cout << "Número é outro valor" << endl;

break;

}

**Loops**: Permite executar um bloco de código repetidamente.

* **For**: Executa um bloco de código um número específico de vezes.

for (int i = 0; i < 5; i++) {

cout << i << " ";

}

**While**: Executa um bloco de código enquanto uma condição é verdadeira.

int i = 0;

while (i < 5) {

cout << i << " ";

i++;

}

**Do-While**: Semelhante ao loop while, mas verifica a condição após a execução do bloco de código.

int i = 0;

do {

cout << i << " ";

i++;

} while (i < 5);

### Entrada e Saída

* **Entrada (cin)**: Usado para obter a entrada do usuário.

int numero;

cout << "Digite um número: ";

cin >> numero;

**Saída (cout)**: Usado para exibir a saída para o usuário.

cout << "Você digitou: " << numero << endl;

Essas instruções básicas formam a espinha dorsal da maioria dos programas em C++. À medida que você se torna mais familiarizado com C++, começará a combinar essas instruções de maneiras mais complexas e poderosas para construir programas sofisticados.

Estruturas de controlo

As estruturas de controle em C++ são usadas para direcionar o fluxo de execução do programa. Elas permitem que o programa tome decisões, execute código repetidamente ou escolha entre várias opções de código a ser executado. Aqui estão as principais estruturas de controle em C++:

### 1. Estruturas de Decisão

#### **if, else-if, else**

A instrução **if** permite que o programa execute um bloco de código apenas se uma determinada condição for verdadeira.

if (condição) {

// Bloco de código executado se a condição for verdadeira

} else if (outra\_condição) {

// Bloco de código executado se a condição anterior for falsa e essa for verdadeira

} else {

// Bloco de código executado se todas as condições anteriores forem falsas

}

#### **switch-case**

A instrução **switch** permite que um programa execute diferentes blocos de código com base no valor de uma variável ou expressão.

switch (expressão) {

case constante1:

// Bloco de código executado se expressão == constante1

break;

case constante2:

// Bloco de código executado se expressão == constante2

break;

// Pode ter mais casos aqui

default:

// Bloco de código executado se nenhum caso corresponder

}

### 2. Estruturas de Loop

#### **for**

O loop **for** é usado para repetir um bloco de código um número específico de vezes. Ele é comumente usado quando o número de iterações é conhecido.

for (inicialização; condição; incremento) {

// Bloco de código a ser repetido

}

#### **while**

O loop **while** repete um bloco de código enquanto uma condição especificada for verdadeira. O número de iterações pode não ser conhecido antes de executar o loop.

while (condição) {

// Bloco de código a ser repetido

}

#### **do-while**

O loop **do-while** é semelhante ao loop **while**, mas a condição é avaliada após a execução do bloco de código. Isso garante que o bloco de código seja executado pelo menos uma vez.

do {

// Bloco de código a ser repetido

} while (condição);

### 3. Controle de Loop

* **break**: Usado para sair prematuramente de um loop.
* **continue**: Pula a iteração atual de um loop e continua com a próxima iteração.

Essas estruturas de controle são fundamentais para a construção de programas em C++, permitindo a criação de códigos mais dinâmicos e interativos.

### Exemplo 1: Uso de if-else

Este programa pede ao usuário que insira um número e informa se o número é positivo, negativo ou zero.

#include <iostream>

using namespace std;

int main() {

int numero;

cout << "Digite um número: ";

cin >> numero;

if (numero > 0) {

cout << "O número é positivo." << endl;

} else if (numero < 0) {

cout << "O número é negativo." << endl;

} else {

cout << "O número é zero." << endl;

}

return 0;

}

### Exemplo 2: Uso de switch-case

Este programa pede ao usuário para escolher uma bebida de um menu e exibe o preço.

#include <iostream>

using namespace std;

int main() {

int escolha;

cout << "Escolha uma bebida:" << endl;

cout << "1. Água" << endl;

cout << "2. Refrigerante" << endl;

cout << "3. Suco" << endl;

cin >> escolha;

switch (escolha) {

case 1:

cout << "Água: R$1,00" << endl;

break;

case 2:

cout << "Refrigerante: R$3,00" << endl;

break;

case 3:

cout << "Suco: R$2,50" << endl;

break;

default:

cout << "Escolha inválida." << endl;

}

return 0;

}

### Exemplo 3: Uso de for loop

Este programa exibe os primeiros 10 números naturais usando um loop for.

#include <iostream>

using namespace std;

int main() {

for (int i = 1; i <= 10; i++) {

cout << i << " ";

}

cout << endl;

return 0;

}

### Exemplo 4: Uso de while loop

Este programa calcula a soma de números digitados pelo usuário até que o usuário digite zero.

#include <iostream>

using namespace std;

int main() {

int numero, soma = 0;

cout << "Digite números para somar (0 para terminar): ";

while (true) {

cin >> numero;

if (numero == 0) {

break;

}

soma += numero;

}

cout << "A soma é: " << soma << endl;

return 0;

}

### Exemplo 5: Uso de do-while loop

Este programa pede ao usuário para digitar um número. Se o número não estiver no intervalo de 1 a 10, o usuário é solicitado a tentar novamente.

#include <iostream>

using namespace std;

int main() {

int numero;

do {

cout << "Digite um número entre 1 e 10: ";

cin >> numero;

} while (numero < 1 || numero > 10);

cout << "Você digitou: " << numero << endl;

return 0;

}

Estes exemplos abrangem as estruturas de controle básicas em C++ e mostram como elas são usadas para controlar o fluxo de execução de um programa.