

# REVISÃO

## TÉCNICAS DE PROGRAMAÇÃO

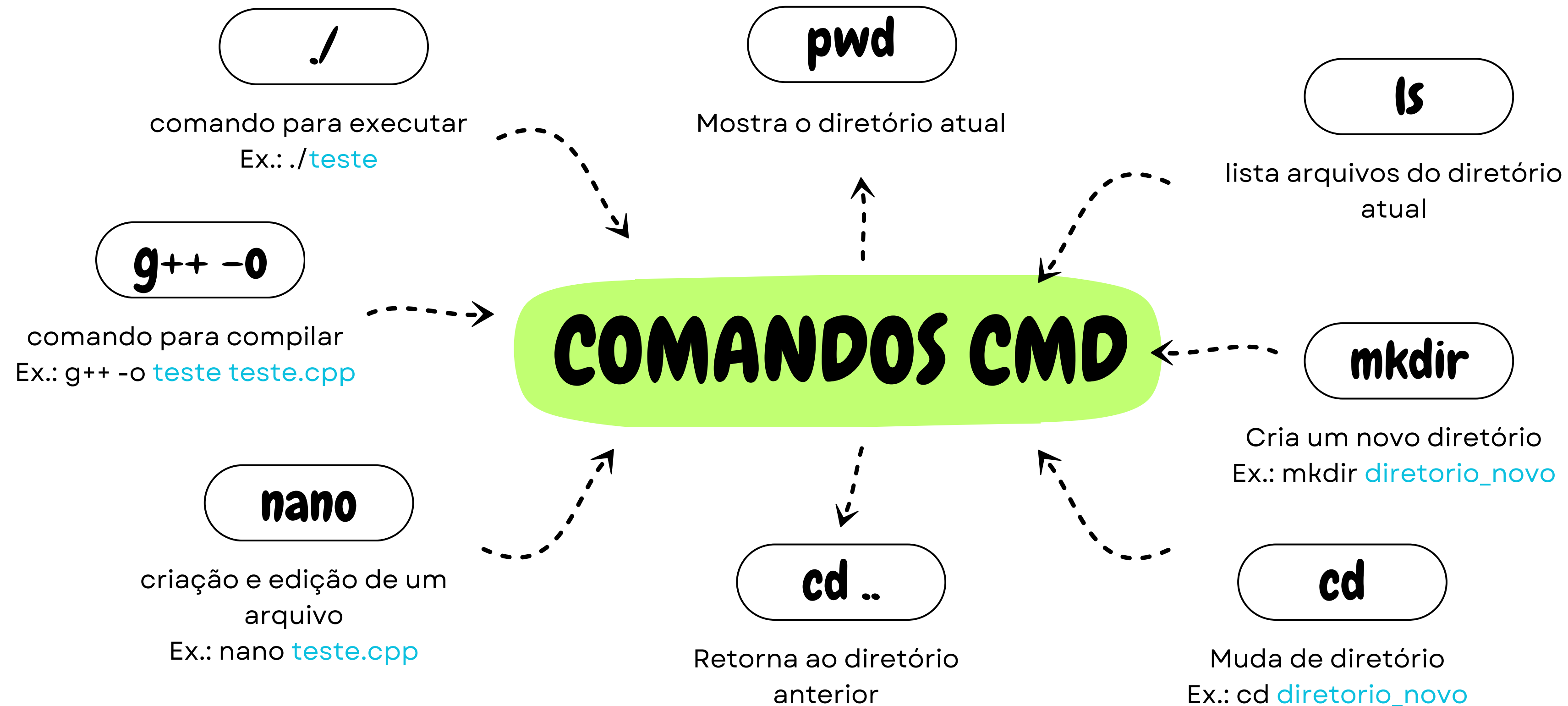
**Professor: Cleonilson Protásio**

**Monitora: Laryssa Paulino**

**Pós-Graduanda: Virgínia Vieira**

# Sumário

- Comandos no CMD
- Ambientes de Programação
- Bibliotecas
- Funções
- Espaços de nome
- Operadores
- if-else
- switch-case
- while
- do-while
- for



### AWS

ssh aluno@3.83.117.124 -p 22

senha: aluno00

### Máquina do lab

ssh aluno@150.165.162.14 -p 100

senha: aluno00

# Ambiente de Programação - CMD

```
aluno@microengenhariaAWS01: ~
Microsoft Windows [versão 10.0.19045.5608]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\Virginia Aires>ssh aluno@3.83.117.124 -p 22
aluno@3.83.117.124's password:
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 6.8.0-1024-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sat Mar 22 14:52:23 UTC 2025

System load:  1.072265625   Processes:           108
Usage of /:   84.8% of 28.89GB Users logged in:       0
Memory usage: 22%          IPv4 address for eth0: 172.31.84.73
Swap usage:   0%

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

97 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

New release '24.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sat Mar 22 14:51:02 2025 from 187.64.107.189
aluno@microengenhariaAWS01:~$

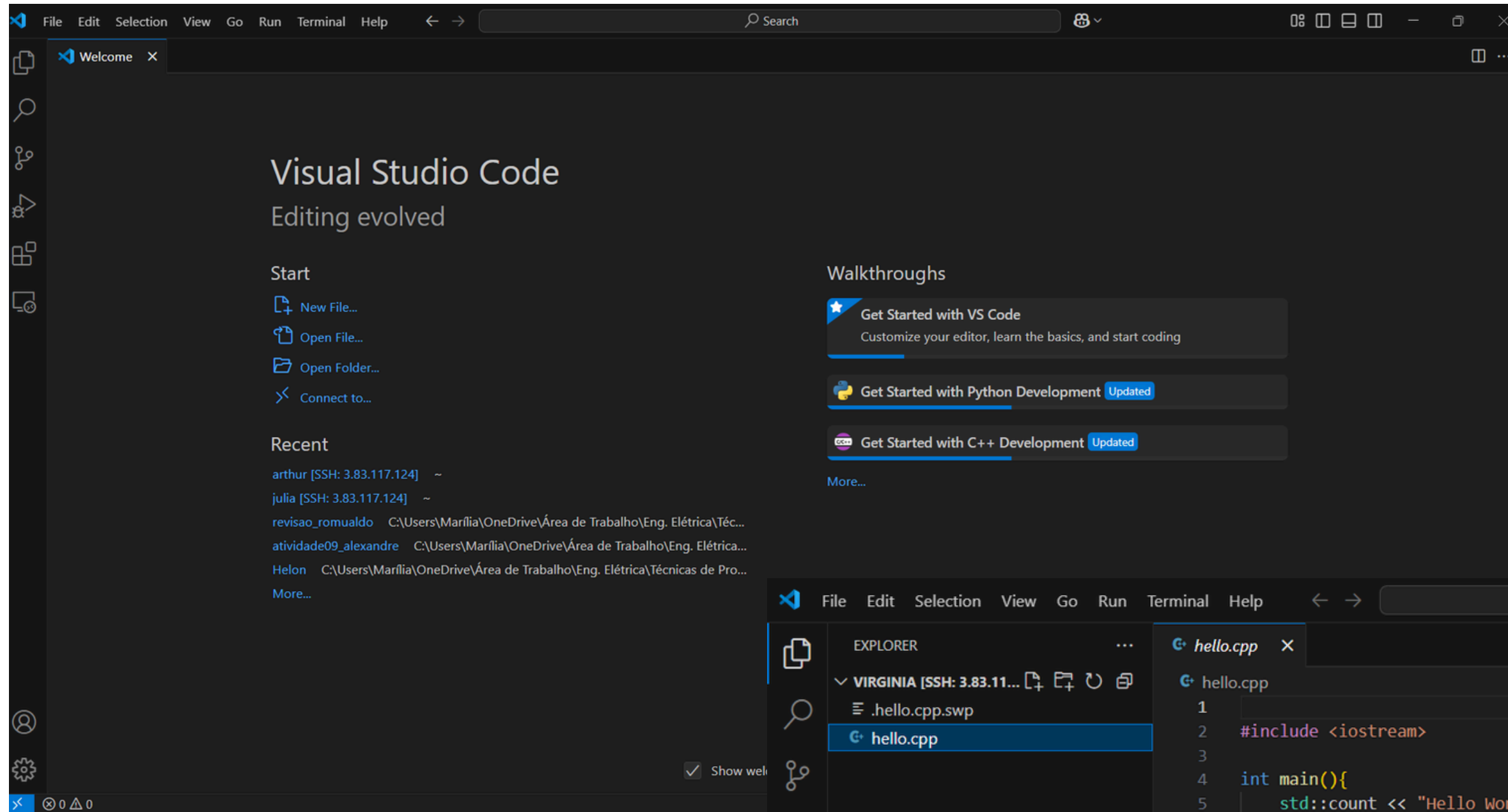
aluno@microengenhariaAWS01: ~/virginia

Last login: Sat Mar 22 14:51:02 2025 from 187.64.107.189
aluno@microengenhariaAWS01:~$ ls
Allysson      Milena  anna    chloe   diego    ferreira  joao_marcos  pierre  romualdofi  samuel1
Edward       Milenar arthur  daniel  eduardoc gabriel   julia        resolucao_exe1 samir    sebasthyan
Melquisedeque Raua    brunna  davi    fabiolahenrique henrique  lucas        rian     samuel      wagner
aluno@microengenhariaAWS01:~$ mkdir virginia
aluno@microengenhariaAWS01:~$ ls
Allysson      Milenar brunna  diego    gabriel   lucas      romualdofi  sebasthyan
Edward       Raua    chloe   eduardoc henrique  pierre     resolucao_exe1 samir    virginia
Melquisedeque anna    daniel  fabiolahenrique joao_marcos  rian       samuel      wagner
Milena       arthur  davi    ferreira julia      rian       samuel1
aluno@microengenhariaAWS01:~$ cd virginia
aluno@microengenhariaAWS01:~/virginia$

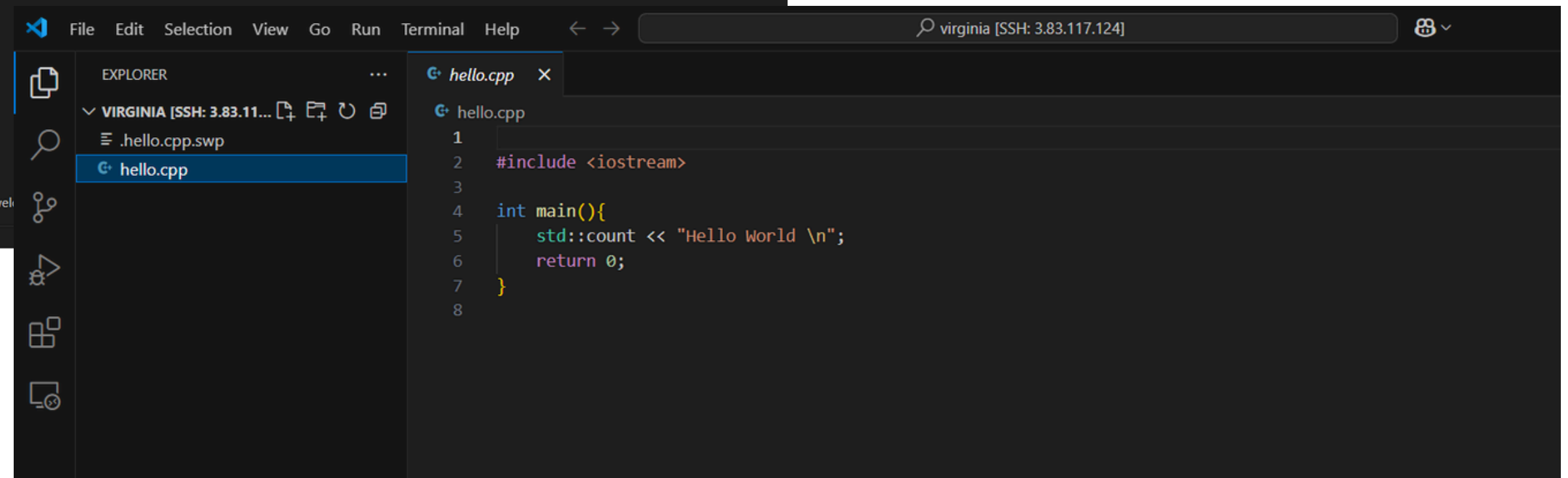
aluno@microengenhariaAWS01: ~/virginia
GNU nano 6.2                                hello.cpp
#include <iostream>

int main(){
    std::cout << "Hello World \n";
    return 0;
}
```

# Ambiente de Programação - VSCODE

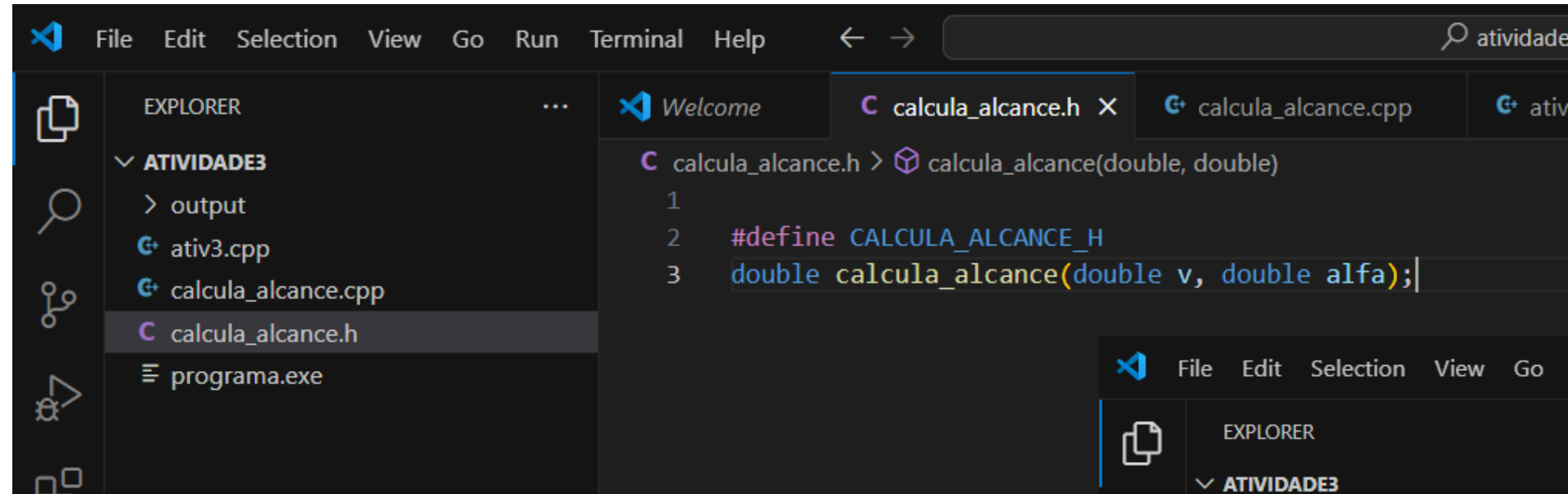


- Acesse a máquina remotamente para programar;
- Caso queira programar sem o acesso remoto, será necessário instalar o compilador MinGW.



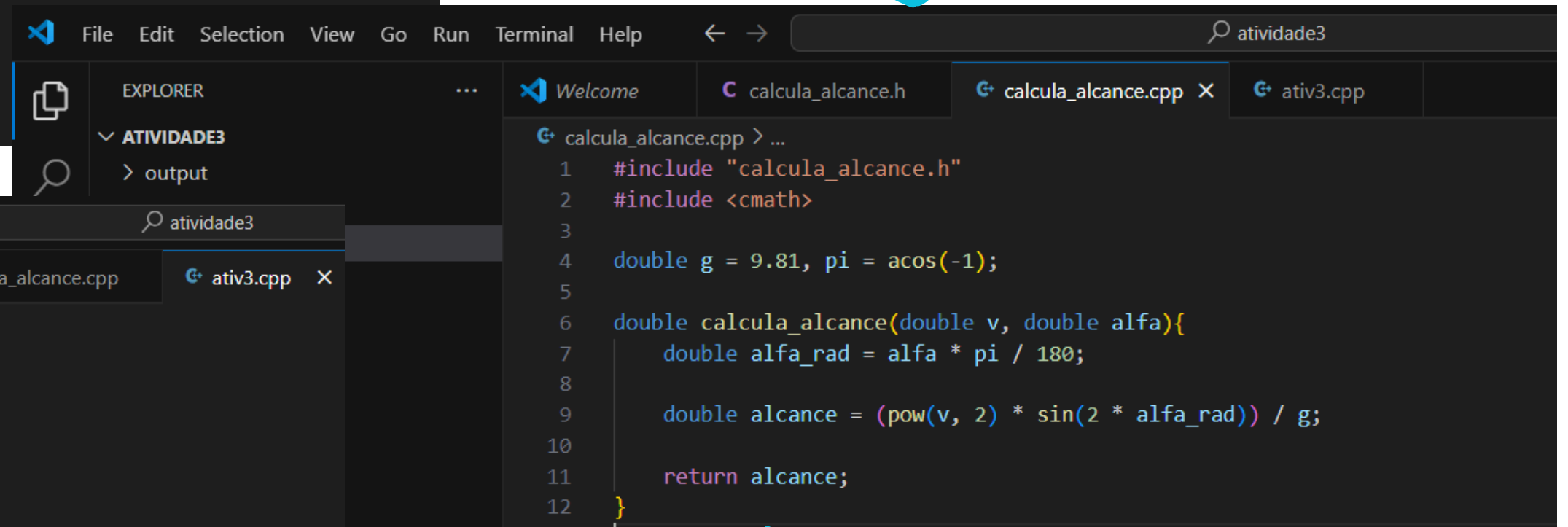
# Bibliotecas

- Bibliotecas são funções que incluímos no nosso código



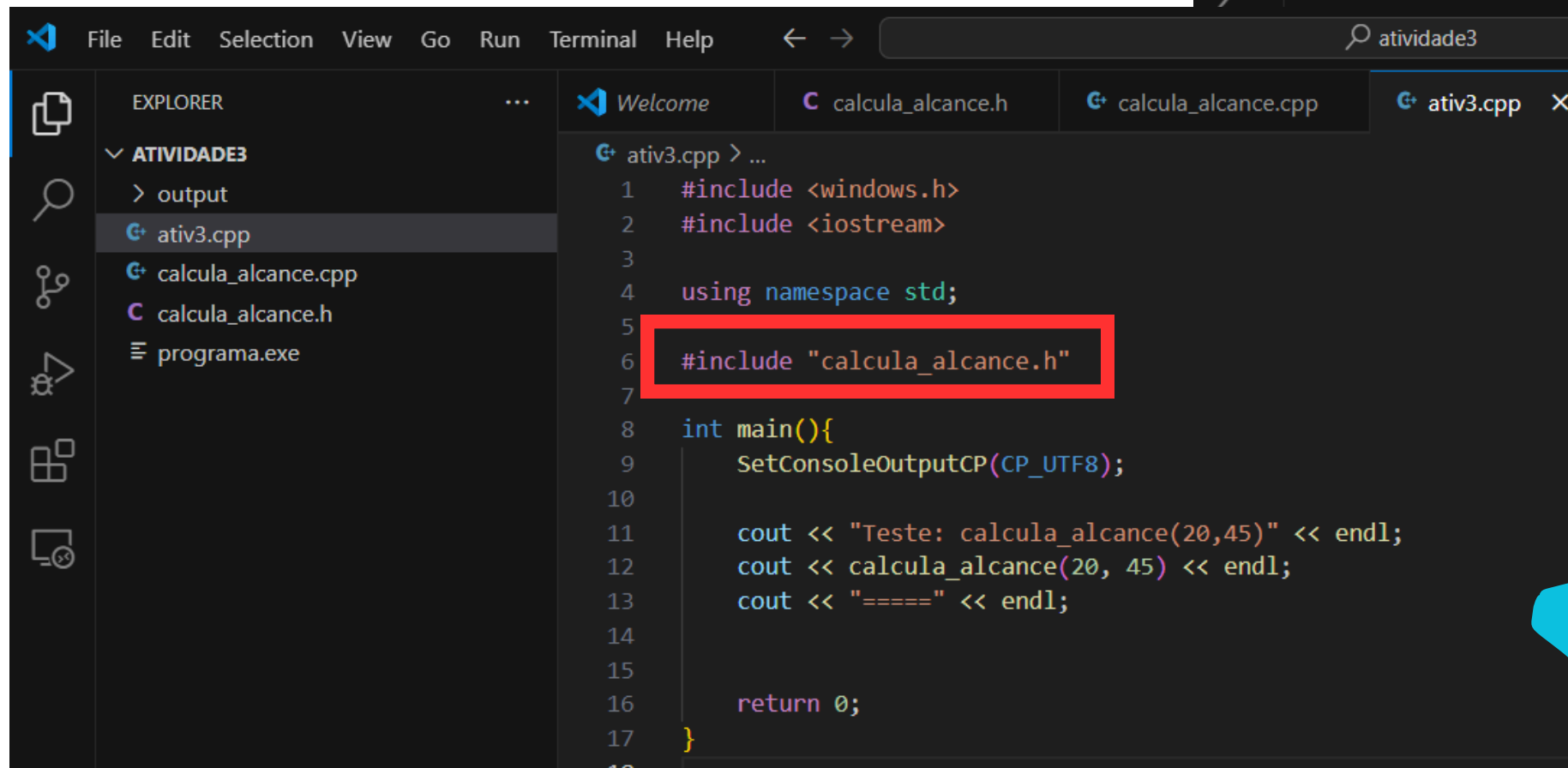
This screenshot shows the Visual Studio IDE with the file explorer on the left displaying a project named 'ATIVIDADE3'. The file 'calcula\_alcance.h' is selected. The main editor window shows the content of this header file, which includes a preprocessor directive to define the header and a function declaration for 'calcula\_alcance'.

```
1  
2 #define CALCULA_ALCANCE_H  
3 double calcula_alcance(double v, double alfa);
```



This screenshot shows the Visual Studio IDE with the file explorer on the left displaying the same project. The file 'calcula\_alcance.cpp' is selected. The main editor window shows the implementation of the 'calcula\_alcance' function, including necessary headers, constant definitions, and the function body.

```
1 #include "calcula_alcance.h"  
2 #include <cmath>  
3  
4 double g = 9.81, pi = acos(-1);  
5  
6 double calcula_alcance(double v, double alfa){  
7     double alfa_rad = alfa * pi / 180;  
8  
9     double alcance = (pow(v, 2) * sin(2 * alfa_rad)) / g;  
10  
11     return alcance;  
12 }
```



This screenshot shows the Visual Studio IDE with the file explorer on the left displaying the same project. The file 'ativ3.cpp' is selected. The main editor window shows the main function of the program, which includes the necessary headers, namespace declaration, and the call to the 'calcula\_alcance' function. The line '#include "calcula\_alcance.h"' is highlighted with a red box.

```
1 #include <windows.h>  
2 #include <iostream>  
3  
4 using namespace std;  
5 #include "calcula_alcance.h"  
6  
7  
8 int main(){  
9     SetConsoleOutputCP(CP_UTF8);  
10  
11     cout << "Teste: calcula_alcance(20,45)" << endl;  
12     cout << calcula_alcance(20, 45) << endl;  
13     cout << "=====" << endl;  
14  
15  
16     return 0;  
17 }
```

# Bibliotecas

- Bibliotecas padrões do C++

|              |            |           |                |             |
|--------------|------------|-----------|----------------|-------------|
| <algorithm>  | <iomanip>  | <list>    | <queue>        | <streambuf> |
| <bitset>     | <ios>      | <locale>  | <set>          | <string>    |
| <complex>    | <iosfwd>   | <map>     | <sstream>      | <typeinfo>  |
| <deque>      | <iostream> | <memory>  | <stack>        | <utility>   |
| <exception>  | <istream>  | <new>     | <stdexcept>    | <valarray>  |
| <fstream>    | <iterator> | <numeric> | <stringstream> | <vector>    |
| <functional> | <limits>   | <ostream> |                |             |

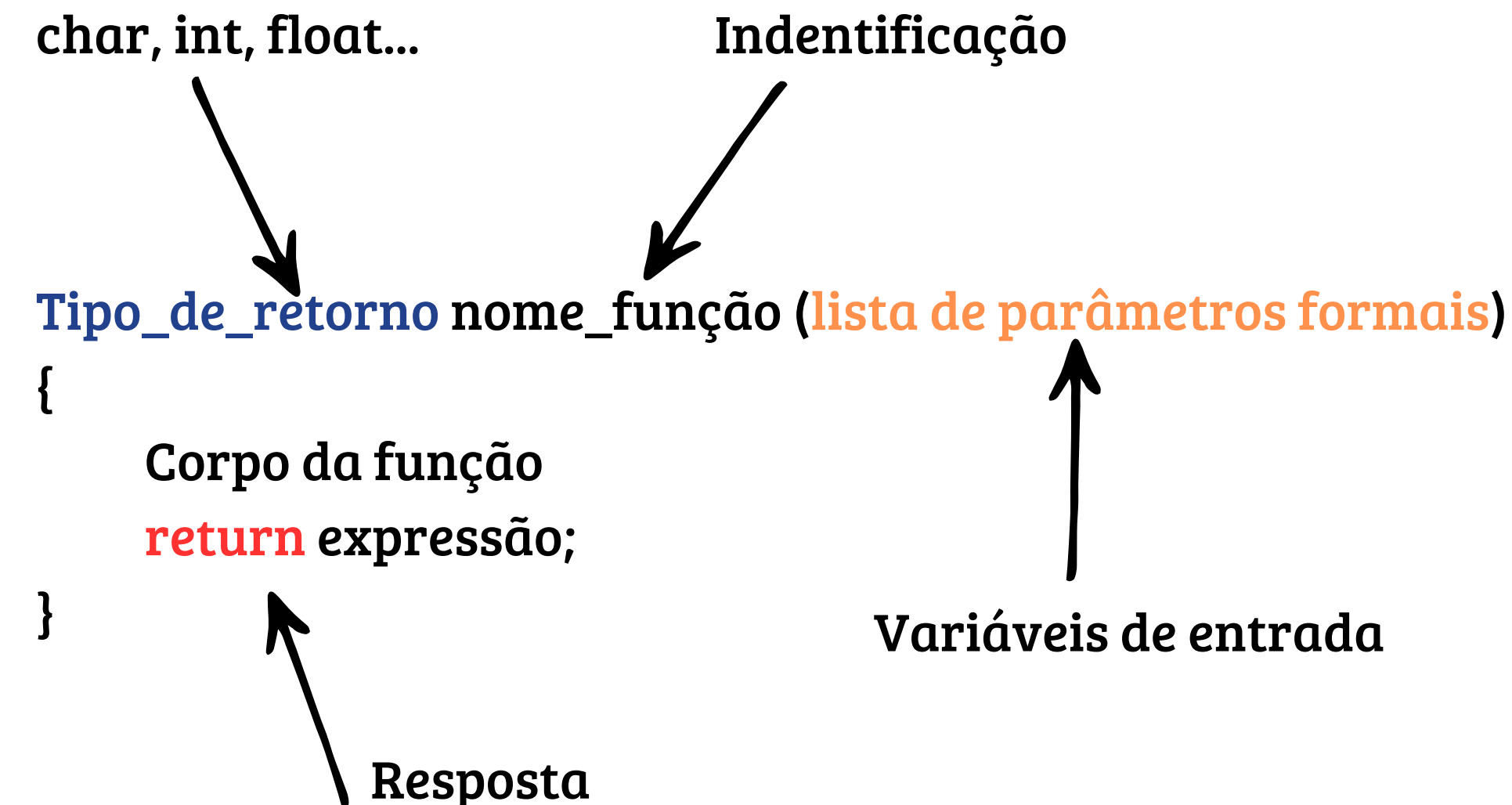
- Bibliotecas padrões do C que o C++ também tem acesso

|           |           |           |           |           |
|-----------|-----------|-----------|-----------|-----------|
| <cassert> | <ciso646> | <csetjmp> | <cstdio>  | <ctime>   |
| <cctype>  | <climits> | <csignal> | <cstdlib> | <cwchar>  |
| <cerrno>  | <locale>  | <cstdarg> | <cstring> | <cwctype> |
| <cfloat>  | <cmath>   | <cstddef> |           |           |



# Funções

- Funções são blocos de códigos reutilizáveis que executam tarefas específicas dentro do programa;
- Blocos de código que recebem parâmetros e retornam valores.



```
#include <iostream>
double g = 9.81, pi = acos(-1);

double calcula_alcance(double v, double alfa){
    double alfa_rad = alfa * pi / 180;

    double alcance = (pow(v, 2) * sin(2 * alfa_rad)) / g;

    return alcance;
}

int main(){
    float resultado = calcula_alcance(30, 30);
    std::cout << resultado << std::endl;
}
```




# Funções - Tipos

- Definidas pelo Usuário - Funções criadas pelo próprio programador para estruturar melhor o código;
- Biblioteca - Funções já definidas na linguagem, acessíveis por meio de biblioteca;
- Com Retorno - Funções que processam dados e devolvem um valor ao final da execução;
- Sem Retorno - Funções que executam uma ação, mas não retornam um valor específico.

```
GNU nano 6.2
#include <iostream>

// Função que apenas imprime uma mensagem
void mensagem() {
    std::cout << "Bem-vindo à aula de revisão!" << std::endl;
}

int main() {
    mensagem();
    return 0;
}
```



```
aluno@microengenhariaAWS01:~/virginia$ g++ -o hello hello.cpp
aluno@microengenhariaAWS01:~/virginia$ ./hello
Bem-vindo à aula de revisão!
```

# Espaços de nome (namespace)

- É um recurso que permite organizar e agrupar identificadores (como variáveis, funções e classes);
- Servem para evitar conflitos com nomes de variáveis iguais.
- Usamos, neste caso, `using namespace laryssa` para evitar usar `laryssa::altura`
- Usamos `using namespace std` pelo mesmo motivo;
- Às vezes usar um namespace pode causar um conflito de variáveis, então cuidado com o uso!

```
#include <iostream>

namespace laryssa{
    int idade;
    float altura;
}

using namespace laryssa;

int main(){
    int idade = 13;
    laryssa::idade = 21;

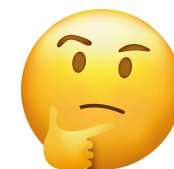
    altura = 1.71;
    float altura = 1.60;

    std::cout << "IDADE = " << idade << std::endl;
    std::cout << "IDADE = " << laryssa::idade << std::endl;

    std::cout << "ALTURA = " << altura << std::endl;
    std::cout << "ALTURA = " << altura << std::endl;
}
```

```
IDADE = 13
IDADE = 21
ALTURA = 1.6
ALTURA = 1.6
```

A estrutura do namespace te lembra alguma coisa?



# Espaços de nome (namespace)

```
namespace laryssa{  
    int idade;  
    float altura;  
}
```

```
struct laryssa{  
    int idade;  
    float altura;  
};
```

```
class laryssa{  
    int idade;  
    float altura;  
};
```

Então qual é a diferença? 🤔

TODA!

| Característica | Namespace                              | Struct                     | Class   |
|----------------|--|----------------------------|---|
| Objetivo       | Organização de código                  | Estrutura de dados simples | Orientação a objetos                          |
| Atributos      | ❌ Não pode armazenar dados diretamente | ✅ Sim                      | ✅ Sim   |
| Métodos        | ❌ Apenas agrupa funções                | ✅ Pode ter métodos         | ✅ Pode ter métodos                            |
| Instanciável?  | ❌ Não pode criar objetos               | ✅ Sim                      | ✅ Sim   |
| Acesso Padrão  | N/A                                    | Público                    | Privado                                       |
| Suporte a POO  | ❌ Não                                  | ❌ Limitado                 | ✅ Sim (Herança, Polimorfismo, Encapsulamento) |

# Operadores

- São símbolos especiais utilizados para realizar operações em variáveis e valores;
- Podem operar sobre:
  - Apenas um valor (Unário)

```
int x = 5;  
int y = -x;
```

- Dois operandos (Binário)

```
int a = 5, b = 10;  
int soma = a + b;  
int maior = (b > a);
```

- Três operandos (Ternário)

```
int nota = 8;  
resultado = (nota >= 7) ? "Aprovado" : "Reprovado";
```

# Operadores - Tipos

- **Atribuição** - Utilizados para atribuir valores a variáveis ;
- **Aritméticos** - Executam operações matemáticas básicas entre variáveis e valores numéricos, como adição, subtração, multiplicação e divisão;
- **Incremento e Decremento** - Utilizados para aumentar ou diminuir o valor de uma variável em uma unidade;
- **Relacionais** - Usados para comparar valores, retornando sempre um resultado verdadeiro ou falso.

```
aluno@microengenhariaAWS01: ~/virginia
GNU nano 6.2
#include <iostream>
using namespace std;

int main() {
    int a, b, soma, diferenca;

    // Atribuição
    a = 10;
    b = 5;

    // Operadores Aritméticos
    soma = a + b;           // soma = 15
    diferenca = a - b;      // diferenca = 5

    // Incremento e Decremento
    a++; // a agora é 11
    b--; // b agora é 4

    // Operadores Relacionais
    if (a > b) {
        cout << "a é maior que b." << endl;
    } else {
        cout << "a NAO é maior que b." << endl;
    }

    // Mostrar os valores finais
    cout << "Valor final de a: " << a << endl;
    cout << "Valor final de b: " << b << endl;
    cout << "Soma: " << soma << endl;
    cout << "Diferenca: " << diferenca << endl;

    return 0;
}
```

```
aluno@microengenhariaAWS01:~/virginia$ ./programa
a é maior que b.
Valor final de a: 11
Valor final de b: 4
Soma: 15
Diferenca: 5
```

# Operadores - Tipos

- Lógicos - Utilizados para combinar, inverter ou avaliar expressões booleanas. Retornam resultados verdadeiros ou falsos com base nas condições estabelecida;
- Bit a Bit - Operam diretamente sobre os bits individuais dos operandos, sendo úteis para manipulações de baixo nível;
- Condicional Ternário - Permite realizar decisões simples de forma compacta, retornando um valor com base em uma condição booleana.

```
GNU nano 6.2
#include <iostream>
using namespace std;

int main() {
    int a = 6;      // 00000110 em binário
    int b = 3;      // 00000011 em binário
    int resultado_bitwise;
    bool logico;

    // Operadores Lógicos
    if (a > 0 && b > 0) {
        logico = true;
    } else {
        logico = false;
    }

    // Operadores Bit a Bit
    resultado_bitwise = a & b; // AND bit a bit: 00000010 → 2

    // Operador Condicional Ternário
    string mensagem = (resultado_bitwise > 0) ? "Resultado é positivo" : "Resultado é zero ou negativo";

    // Exibir os resultados
    cout << "a = " << a << ", b = " << b << endl;
    cout << "Lógico (a > 0 && b > 0): " << logico << endl;
    cout << "Resultado Bitwise (a & b): " << resultado_bitwise << endl;
    cout << "Mensagem do Ternário: " << mensagem << endl;

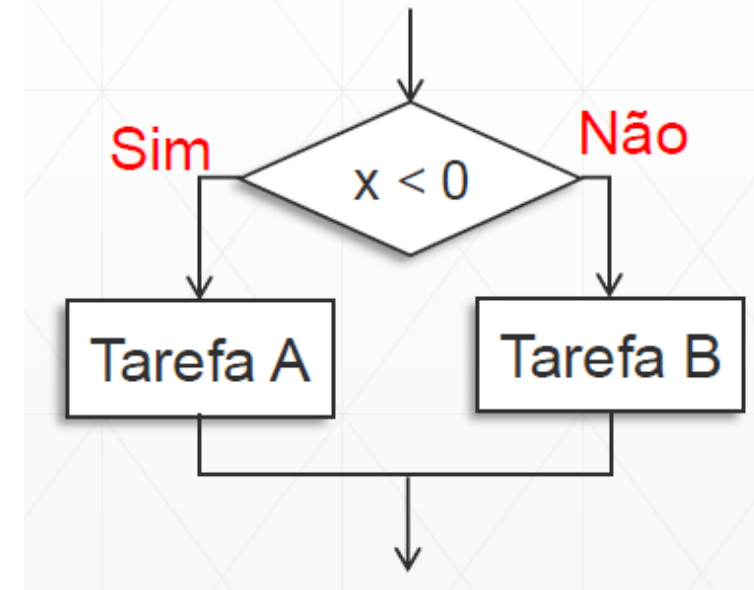
    return 0;
}
```

```
aluno@microengenhariaAWS01:~/virginia$ g++ -o hello hello.cpp
aluno@microengenhariaAWS01:~/virginia$ ./hello
a = 6, b = 3
Lógico (a > 0 && b > 0): 1
Resultado Bitwise (a & b): 2
Mensagem do Ternário: Resultado é positivo
```



# if-else

- Executa uma operação apenas SE a condição for verdadeira;
- SE NÃO for verdadeira, executa outra operação;



Exemplo:

```
#include <iostream>

using namespace std;

int main(){
    int idade = 13;

    if(idade < 18){
        cout << "Você é menor de idade!\n";
    }
    else cout << "Você é maior de idade!\n";

    float nota = 7.0;

    if(nota > 7)
        cout << "Você está aprovado!\n";
    else if(nota >= 4 && nota < 7)
        cout << "Prova final!\n";
    else
        cout << "Reprovado(a)!\n";

    return 0;
}
```

- Podemos utilizar a estrutura if-else aninhadas, testando mais de uma condição em cadeia

```
if (x > 0)
    cout << "x eh positivo";
else if (x < 0)
    cout << "x eh negativo";
else
    cout << "x eh 0";
```

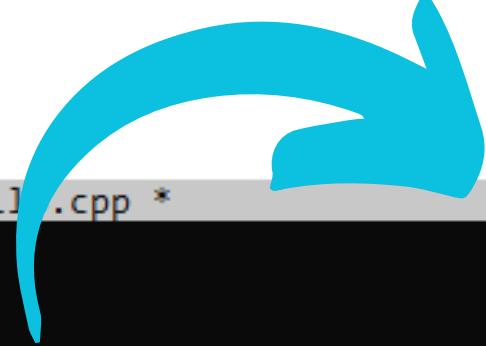


# Switch-Case

- Estrutura de controle de fluxo utilizada para tomar decisões com base no valor de uma variável;
- É recomendada quando a estrutura if-else se torna extensa e de difícil leitura;
- Geralmente, aceita variáveis dos tipos int, char ou equivalentes, dependendo da linguagem;
- Utiliza-se o comando break para encerrar a execução do switch após um caso ser atendido, evitando que os demais casos sejam executados em sequência.

- Estrutura:  
`switch (expressão) {`  
  
    `case valor1:`  
        `//instruções se expressão == valor1`  
        `break;`  
  
    `case valor2:`  
        `//instruções se expressão == valor2`  
        `break;`  
  
    `...`  
  
    `default:`  
        `// instruções caso nenhum dos casos anteriores for satisfeito`  
`}`

# Switch-Case



```
GNU nano 6.2      hell.cpp *
#include <iostream>
using namespace std;

int main() {
    int opcao;

    cout << "Menu de Opções:" << endl;
    cout << "1 - Mostrar mensagem de boas-vindas" << endl;
    cout << "2 - Somar dois números" << endl;
    cout << "3 - Sair" << endl;
    cout << "Digite a opção desejada: ";
    cin >> opcao;

    switch (opcao) {
        case 1:
            cout << "Bem-vindo ao programa!" << endl;
            break;

        case 2: {
            int a, b;
            cout << "Digite dois números: ";
            cin >> a >> b;
            cout << "Soma = " << (a + b) << endl;
            break;
        }

        case 3:
            cout << "Saindo do programa..." << endl;
            break;

        default:
            cout << "Opção inválida!" << endl;
    }

    return 0;
}
```

```
aluno@microengenhariaAWS01:~/virginia$ ./hello
Menu de Opções:
1 - Mostrar mensagem de boas-vindas
2 - Somar dois números
3 - Sair
Digite a opção desejada: 1
Bem-vindo ao programa!
aluno@microengenhariaAWS01:~/virginia$ ./hello
Menu de Opções:
1 - Mostrar mensagem de boas-vindas
2 - Somar dois números
3 - Sair
Digite a opção desejada: 2
Digite dois números: 5
3
Soma = 8
aluno@microengenhariaAWS01:~/virginia$ ./hello
Menu de Opções:
1 - Mostrar mensagem de boas-vindas
2 - Somar dois números
3 - Sair
Digite a opção desejada: 4
Opção inválida!
aluno@microengenhariaAWS01:~/virginia$ ./hello
Menu de Opções:
1 - Mostrar mensagem de boas-vindas
2 - Somar dois números
3 - Sair
Digite a opção desejada: 3
Saindo do programa...
```

- Se não colocar o break?

```
aluno@microengenhariaAWS01:~/virginia$ ./hello
Menu de Opções:
1 - Mostrar mensagem de boas-vindas
2 - Somar dois números
3 - Sair
Digite a opção desejada: 1
Bem-vindo ao programa!
Digite dois números: 2 3
Soma = 5
Saindo do programa...
Opção inválida!
```

# while/do-while

- Executa uma operação repetidamente ENQUANTO a condição for verdadeira;
- Testa ----> Executa
- **while**(condição){comando}

```
#include <iostream>

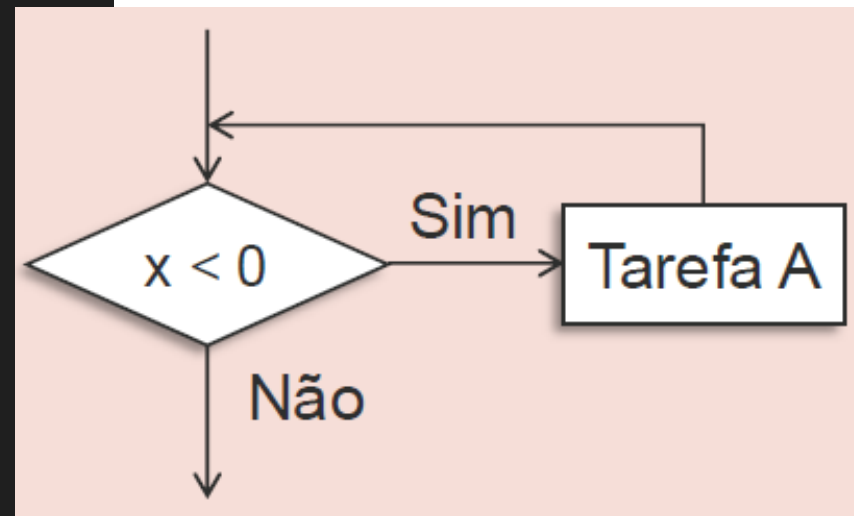
using namespace std;

int main(){

    int a = 10;

    while(a < 10){
        cout << a << endl;
        a++;
    }
    cout << "FIM!" << endl;

}
```



FIM!

- Executa ----> Testa
- **do**{comando}**while**(condição)

```
#include <iostream>

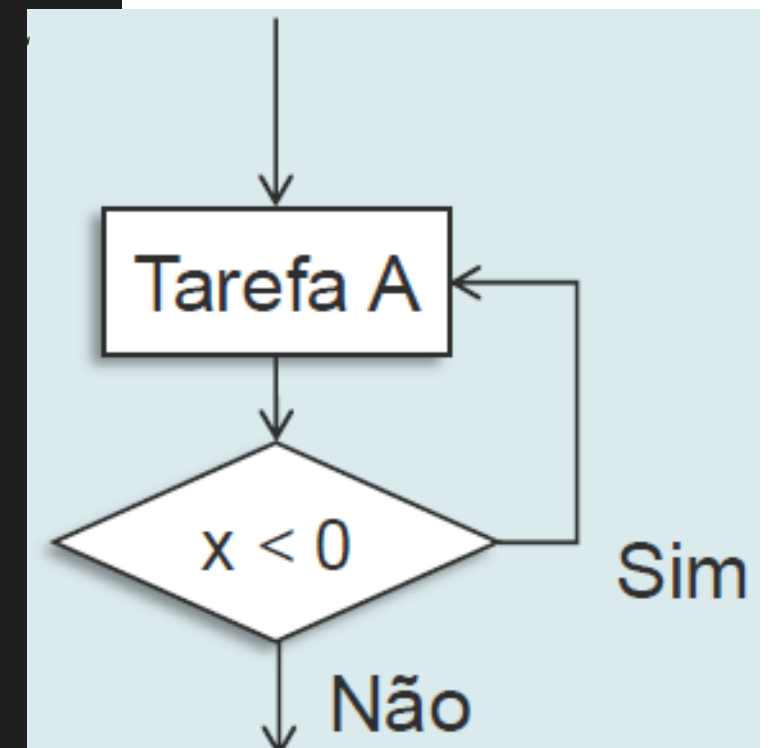
using namespace std;

int main(){

    int a = 10;

    do{
        cout << a << endl;
        a++;
    }
    while(a < 10);

}
```

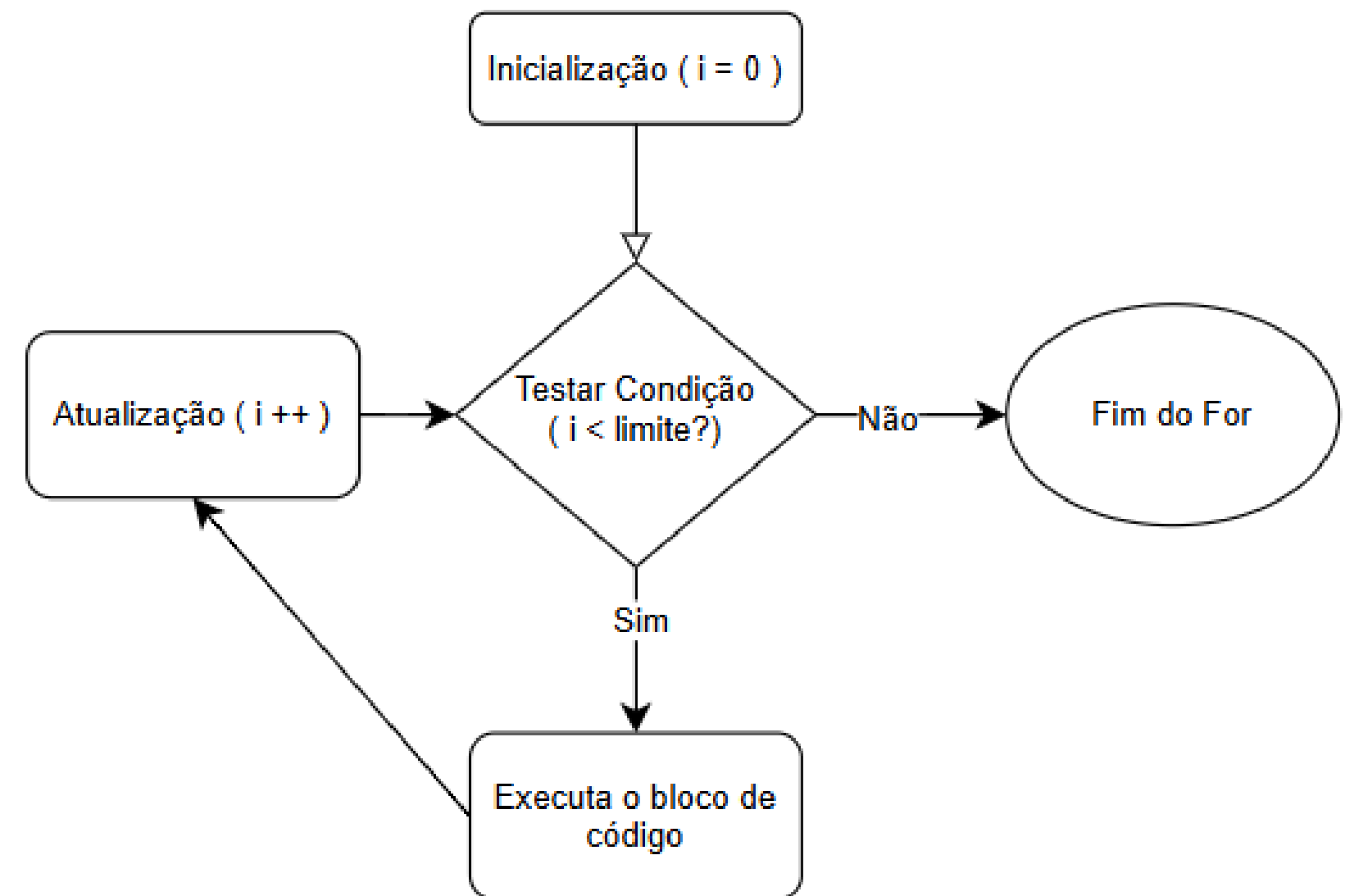


10

# For

- Estrutura de repetição for – Utilizada para executar um bloco de código um número conhecido e determinado de vezes;
- Estrutura:
  - Inicialização - Define a variável de controle, executada uma vez no início;
  - Condição - Expressão lógica que é avaliada antes de cada repetição;
  - Atualização - Altera a variável de controle após cada repetição.
- Ajuda a evitar esquecimentos comuns, como esquecer de atualizar a variável de controle.

- Estrutura:



# For

```
GNU nano 6.2      hello.cpp
#include <iostream>
using namespace std;

int main() {
    int n, numero, somaPares = 0, totalNumeros = 0;

    cout << "Quantos números você deseja digitar? ";
    cin >> n;

    for (int i = 1; i <= n; i++) {
        cout << "Digite o número " << i << ": ";
        cin >> numero;

        if (numero % 2 == 0) {
            somaPares += numero; // soma se for par
        }

        totalNumeros++; // contador geral
    }

    cout << "\nTotal de números digitados: " << totalNumeros << endl;
    cout << "Soma dos números pares: " << somaPares << endl;

    return 0;
}
```

```
aluno@microengenhariaAWS01:~/virginia$ g++ -o hello hello.cpp
aluno@microengenhariaAWS01:~/virginia$ ./hello
Quantos números você deseja digitar? 5
Digite o número 1: 2
Digite o número 2: 4
Digite o número 3: 5
Digite o número 4: 6
Digite o número 5: 7

Total de números digitados: 5
Soma dos números pares: 12
```



# Exercícios

# REVISÃO

## TÉCNICAS DE PROGRAMAÇÃO

**Professor: Cleonilson Protásio**

**Monitora: Laryssa Paulino**

**Pós-Graduanda: Virgínia Vieira**



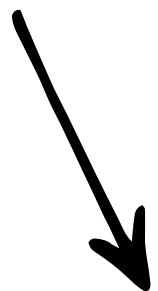
# Sumário

- **Array;**
- **String;**
- **Structs e Classes;**
- **Funções construtoras e destrutoras;**
- **Ponteiros;**

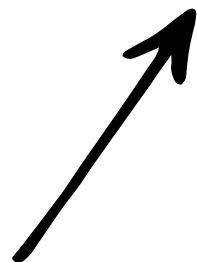
# Array

- Array é uma estrutura de dados que armazena uma sequência de elementos do mesmo tipo, em posições vizinhas de memória;
- Os elementos são acessados por índices (Começando em 0);
- Podem ser do tipo:
  - Unidimensional;
  - multidimensional.
- Declaração de uma Array:

char, int, float...



tipo nome\_array[tamanho]



Identificador

```
int numeros[10]; // array de 10 inteiros
float valores[5]; // array de 5 float
char letras[3]; // array de 3 caracteres
```

# Array

- Arrays podem ser inicializados diretamente no momento da declaração:

```
int numero[6] = {10,20,30,40,50,60}; //Tamanho explícito, com todos os elementos inicializados
int n[] = {3,4,5}; //Tamanho implícito, conta quantos elementos estão na lista
char c[] = {'U','F','P','B'} // Array de caracteres individuais, sem o caracter nulo \0
char s[] = "UFPB"; //string literal, inclui automaticamente o caracter nulo \0 ao final
```

- Pode acessar ou alterar elementos de uma array pelo seu índice:

```
numeros[0] = 15;
cout << numeros[0];
```

- Para saber o tamanho do array, utiliza-se sizeof:

```
sizeof(numeros); //retorna o total de bytes ocupados

int tamanho = sizeof(numeros) / sizeof(numeros[0]); //retorna o número de elementos
```

# Array

```
GNU nano 6.2                                hello.cpp *
#include <iostream>
using namespace std;

int main() {
    // 1. Inicialização do array
    int numeros[] = {10, 20, 30, 40, 50};


    // 2. Cálculo do tamanho total em bytes
    int tamanhoEmBytes = sizeof(numeros);

    // 3. Cálculo do número de elementos
    int quantidade = sizeof(numeros) / sizeof(numeros[0]);

    // Exibir informações
    cout << "Tamanho total do array em bytes: " << tamanhoEmBytes << endl;
    cout << "Quantidade de elementos: " << quantidade << endl;

    // 4. Acesso e impressão dos elementos
    cout << "Elementos do array:" << endl;
    for (int i = 0; i < quantidade; i++) {
        cout << "numeros[" << i << "] = " << numeros[i] << endl;
    }

    return 0;
}
```



```
aluno@microengenhariaAWS01:~/virginia$
Tamanho total do array em bytes: 20
Quantidade de elementos: 5
Elementos do array:
numeros[0] = 10
numeros[1] = 20
numeros[2] = 30
numeros[3] = 40
numeros[4] = 50
```

# Array - Multidimensional


- Um Array multidimensional é uma matriz ou tabela, com linhas e colunas.
- Bidimensional:

```
GNU nano 6.2
#include <iostream>
using namespace std;

int main() {
    // Declarando e inicializando uma matriz 3x2
    int matriz[3][2] = {
        {1, 2},
        {3, 4},
        {5, 6}
    };

    // Acessando e imprimindo os elementos
    cout << "Matriz 3x2:" << endl;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 2; j++) {
            cout << matriz[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```




```
aluno@microengent
Matriz 3x2:
1 2
3 4
5 6
```

- Multidimensional:

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

int main() {
    // Declarando um cubo 2x2x2 e inicializando os valores
    int cubo[2][2][2] = {
        {
            {1, 2},
            {3, 4}
        },
        {
            {5, 6},
            {7, 8}
        }
    };

    // Imprimindo os elementos
    cout << "Array tridimensional 2x2x2:" << endl;
    for (int i = 0; i < 2; i++) {
        cout << "Plano " << i << ":" << endl;
        for (int j = 0; j < 2; j++) {
            for (int k = 0; k < 2; k++) {
                cout << cubo[i][j][k] << " ";
            }
            cout << endl;
        }
    }
}
```



```
aluno@microengenhariaAWS01:~
Array tridimensional 2x2x2:
Plano 0:
1 2
3 4

Plano 1:
5 6
7 8
```

# Array

- Ao passar um array para uma função, o que é realmente transmitido é o endereço de memória do seu primeiro elemento, permitindo o acesso aos seus valores por referência, em vez de copiar todo o array.

```
GNU nano 6.2
#include <iostream>
using namespace std;

// Função para imprimir os elementos do array
void imprimirArray(int v[], int tamanho) {
    cout << "Elementos do array: ";
    for (int i = 0; i < tamanho; i++) {
        cout << v[i] << " ";
    }
    cout << endl;
}

// Função para dobrar os valores do array
void dobrarElementos(int v[], int tamanho) {
    for (int i = 0; i < tamanho; i++) {
        v[i] *= 2; // dobra o valor
    }
}

int main() {
    int numeros[] = {1, 2, 3, 4, 5};
    int tamanho = sizeof(numeros) / sizeof(numeros[0]);

    imprimirArray(numeros, tamanho); // mostra valores originais
    dobrarElementos(numeros, tamanho); // modifica os valores
    imprimirArray(numeros, tamanho); // mostra após modificação

    return 0;
}
```

```
aluno@microengenhariaAWS01:~/virg
Elementos do array: 1 2 3 4 5
Elementos do array: 2 4 6 8 10
```



# Strings

- É uma sequência de caracteres que possui duas formas de representação:

- Como uma Array de char:

```
char nome[] = "revisão";
```

- Utilizando a biblioteca <string>

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

using namespace std;

int main() {

    string nome = "revisao";
    cout << nome << endl;

    return 0;
}
```

- Também é possível utilizar a biblioteca <cstring>, que contém funções para manipulação de C-strings, que são arrays de caracteres terminados pelo caractere nulo (\0).
- Algumas delas são:
  - **strlen**(arg): retorna o tamanho;
  - **strcpy**(destino, origem): copia uma string em outra;
  - **strcmp**(arg1, arg2): compara;
  - **strcat**(arg1, arg2): concatena uma string na outra



# Strings

- Exemplo:

```
#include <iostream>
#include <cstring>

using namespace std;

int main() {
    char a[8] = "REVISA0";
    char b[8];

    strcpy(b, a);

    cout << "Tamanho de a: " << strlen(a) << endl;
    cout << "Tamanho de b: " << strlen(b) << endl;

    int comp = strcmp(a, b);
    cout << comp << endl;
    //Retorna 0 se a == b
    //Retorna 1 se a > b. Compara caracter por caracter em ASCII até encontrar o maior
    //Retorna -1 se a < b. Compara caracter por caracter em ASCII até encontrar o menor
    char c[40];
    strcpy(c, "string");
    strcat(c, " concatenado");
    cout << c << endl;

    return 0;
}
```

Tamanho de a: 7

Tamanho de b: 7

0

string concatenado

# A nível de conhecimento...

- Há uma biblioteca em C++ chamada `<ctype>` que possibilita a verificação de caracteres de uma string.

- Funções de caracteres: `<ctype>`

- `int isalpha (int c)` // teste se c é letra
- `int isdigit (int c)` // teste se c é dígito decimal
- `int isupper (int c)` // teste se c é letra maiúscula
- `int islower (int c)` // teste se c é letra minúscula
- `int isalnum (int c)` // teste se c é letra ou número
- `int iscntrl (int c)` // teste se c é caractere de controle
- `int isxdigit (int c)` // teste se c é dígito hexadecimal
- `int isspace (int c)` // teste se c é espaço, avanço de página, nova linha, retorno de carro, tabulação.
- `int toupper (int c)` // converte para letra maiúscula
- `int tolower (int c)` // converte para letra minúscula

# Struct

- De maneira simplificada, podemos associar como sendo “um array que armazena mais de um tipo de dado”.

```
#include <iostream>
#include <cstring>

using namespace std;

int main() {
    int meu_array[5] = {2, 3, 4, 5, 6};
}

struct estrutura{
    int a = 5;
    float b = 10.0;
    char c = 'x';
};
```

- Muito útil para organizar vários tipos de dados em uma só estrutura.
- Podemos acessar esses dados e setar com quaisquer valores que quisermos;

```
#include <iostream>
#include <cstring>

using namespace std;

struct pessoa{
    int idade;
    float altura;
    char nome[50];
};

int main() {
    pessoa virginia;
    strcpy(virginia.nome, "virginia");
    virginia.idade = 24;
    virginia.altura = 1.60;
}
```

# Struct

- Posso, inclusive, criar uma função que receba como parâmetro uma estrutura!

```
#include <iostream>
#include <cstring>

using namespace std;

struct pessoa{
    int idade;
    float altura;
    char nome[50];
};

void validar_altura(pessoa & individuo){
    if(individuo.idade < 0){
        cout << "Idade Inválida!" << endl;
    }
}

int main() {
    pessoa virginia;
    strcpy(virginia.nome, "virginia");
    virginia.idade = 24;
    virginia.altura = 1.60;
    validar_altura(virginia);
}
```

# Struct

- Posso, inclusive, criar uma função que receba como parâmetro uma estrutura!

```
#include <iostream>
#include <cstring>

using namespace std;

struct pessoa{
    int idade;
    float altura;
    char nome[50];
};

void validar_altura(pessoa & individuo){
    if(individuo.idade < 0){
        cout << "Idade Inválida!" << endl;
    }
}

int main() {
    pessoa virginia;
    strcpy(virginia.nome, "virginia");
    virginia.idade = 24;
    virginia.altura = 1.60;
    validar_altura(virginia);
}
```

Agora imagina combinar funções e estruturas em um lugar só?

**SÃO AS NOSSAS CLASSES!**

# Class

- É uma maneira de conectar estruturas e funções, associadas umas às outras!

dados

```
struct pessoa{  
    int idade;  
    float altura;  
    char nome[50];  
};
```



funções

```
void validar_idade(pessoa & individuo){  
    if(individuo.idade < 0){  
        cout << "Idade Inválida!" << endl;  
    }  
}
```

```
class pessoa{  
    private:  
        int idade;  
        float altura;  
        char nome[50];  
  
    public:  
        void validar_idade(){  
            if(idade < 0){  
                cout << "Idade Inválida!" << endl;  
            }  
        };  
};
```

Combinação de dados e  
funções

# P00

- Programação Orientada a Objetos (POO) é um paradigma de programação baseado no conceito de objetos, que representam entidades do mundo real. Esses objetos possuem atributos (dados) e métodos (funções que manipulam esses dados).

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

using namespace std;

class carro{
private:
    int cor;
    string marca;
    string modelo;
    float velocidade;
    float aceleracao;

public:
    void acelerar();
    void frear();
};

int main() {
    carro meu_carro;
}
```

- Foi criado um OBJETO da CLASSE carro
- O objeto é uma instância da classe



# PОО

- Programação Orientada a Objetos (POO) é um paradigma de programação baseado no conceito de objetos, que representam entidades do mundo real. Esses objetos possuem atributos (dados) e métodos (funções que manipulam esses dados).

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

using namespace std;
```

```
class carro{
private:
int cor;
string marca;
string modelo;
float velocidade;
float aceleracao;

public:
void acelerar();
void frear();
};
```

```
int main() {
    carro meu_carro;
}
```

Variáveis-membro

Funções-membro

# P OO

- Programação Orientada a Objetos (P OO) é um paradigma de programação baseado no conceito de objetos, que representam entidades do mundo real. Esses objetos possuem atributos (dados) e métodos (funções que manipulam esses dados).

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

using namespace std;

class carro{
    private:
        int cor;
        string marca;
        string modelo;
        float velocidade;
        float aceleracao;
    public:
        void acelerar();
        void frear();
};

int main() {
    carro meu_carro;
}
```

## ▪ Especificador\_de\_acesso

Existe o especificador *protected*.

### ▪ *private*:

- **Membros privados** podem ser acessados somente por membros da mesma classe. (**NÃO VISÍVEIS FORA DA CLASSE**)

### ▪ *public*:

- **Membros públicos** podem ser acessados em qualquer escopo em que a classe é visível. (**VISÍVEIS FORA DA CLASSE**)

- Tornar dados públicos os deixam desprotegidos
- A melhor forma de acessar dados-membro é por meio de funções `get()` e `set()`

# P00

```
#include <iostream>
#include <cstring>

using namespace std;

class pessoa{
    int idade;
    char nome[50];
    float altura;

    public:
    void set_idade(int);
    void set_nome(char[50]);
    void set_altura(float);
};

void pessoa::set_idade(int x){idade = x;}
void pessoa::set_nome(char a[50]){strcpy(nome, a);}
void pessoa::set_altura(float b){altura = b;}

int main() {
    pessoa pessoa1;
}
```

# Funções Construtoras

- As funções construtoras são chamadas na criação de um objeto e serve para:
  - Inicializar variáveis-membros,
  - Alocar memória dinamicamente, e
  - Evitar o uso de valores inesperados.
- As funções construtoras, um tipo de função especial de uma classe, são automaticamente chamadas quando um novo objeto é criado.
- Uma função construtora:
  - Tem o mesmo nome da classe
  - E não deve ter nenhum tipo de retorno (nem mesmo void)

# Funções Construtoras

```
#include <iostream>
#include <cstring>

using namespace std;

class pessoa{
    int idade;
    char nome[50];
    float altura;

public:
    pessoa(int a, float b){
        idade = a; altura = b;
    }
    void set_idade(int);
    void set_nome(char[50]);
    void set_altura(float);
};

void pessoa::set_nome(char a[50]){strcpy(nome, a);}

int main() {
    pessoa pessoa1(20, 1.65);
    char nome[50];
    pessoa1.set_nome(nome);
}
```

# Funções Destrutoras

- Fazem o oposto das funções construtoras
- São automaticamente chamadas quando um objeto é destruído:
  - O escopo do objeto é finalizado
    - Exemplo: um objeto é definido localmente dentro de uma função e a função termina
- O objeto é alocado dinamicamente na memória e é usado um operador de deleção do objeto
- O destrutor tem o mesmo nome da classe, mas:
  - é precedido de ~
  - Não retorna nenhum tipo

# Funções Destrutoras

```
class pessoa{
    int idade;
    char nome[50];
    float altura;

public:
    pessoa(int a, float b){
        idade = a; altura = b;
    }
    ~pessoa(){
        cout << "Objeto destruído!" << endl;
    }
    void set_nome(char[50]);
};

void pessoa::set_nome(char a[50]){strcpy(nome, a);}
}
```

# Ponteiros

- Ponteiros armazenam o endereço de uma variável;
- Operador & obtém o endereço:
  - `int* p = &x;`
- Operador \* acessa o valor armazenado no endereço:
  - `*p;`
- Você pode acessar:
  - O endereço do ponteiro com `&p;`
  - O conteúdo apontado com `*p;`

```
GNU nano 6.2
#include <iostream>
using namespace std;

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

    cout << "Valor de x: " << x << endl;
    cout << "Endereco de x (&x): " << &x << endl;
    cout << "Valor de p (endereco): " << p << endl;
    cout << "Valor apontado por p (*p): " << *p << endl;

    return 0;
}
```

```
aluno@microengenhariaAWS01:~/virginia$
Valor de x: 10
Endereco de x (&x): 0x7ffda2a37bec
Valor de p (endereco): 0x7ffda2a37bec
Valor apontado por p (*p): 10
```





# Ponteiros


- Declaração:
  - **Tipo**\* **nome**
- Inicialização:
  - `p = &variavel;`
- O ponteiro deve ser do mesmo tipo da variável apontada:
  - Ex.: `float* pf;` só pode apontar para float;
- A verificação de tipo evita erros de compilação.

```
GNU nano 6.2
#include <iostream>
using namespace std;

int main() {
    float valor = 3.14;
    float* pf;      // declaração
    pf = &valor;     // inicialização

    cout << "Endereco de valor: " << &valor << endl;
    cout << "Valor via ponteiro (*pf): " << *pf << endl;

    return 0;
}
```



```
aluno@microengenhariaAWS01:~/virgi
Endereco de valor: 0x7fff48c74c3c
Valor via ponteiro (*pf): 3.14
```

# Ponteiros

- **Ponteiro para ponteiro:**
  - Um ponteiro que armazena o endereço de outro ponteiro;
  - `int** pp = &p;`
- **Ponteiro para objetos:**
  - Permitem acessar membros de um objeto dinamicamente criado;
  - `Pessoa* obj = new Pessoa();`  
`obj->nome = "João";`
- **Ponteiro e Array:**
  - É um array onde cada elemento é um ponteiro;
  - `int* vet[5];`
- **Acesso a objetos via ponteiro:**
  - Quando usar ponteiros para objetos, o operador `->` é necessário para acessar membros
  - `obj->atributo;`

```
GNU nano 6.2
#include <iostream>
using namespace std;

int main() {
    int a = 5;
    int* p = &a;
    int** pp = &p;


    cout << "Valor de a: " << a << endl;
    cout << "Via ponteiro: " << *p << endl;
    cout << "Via ponteiro para ponteiro: " << **pp << endl;

    int arr[] = {1, 2, 3};
    int* pArr = arr;

    cout << "Primeiro elemento do array via ponteiro: " << *pArr << endl;
    cout << "Segundo elemento: " << *(pArr + 1) << endl;

    return 0;
}
```

```
aluno@microengenhariaAWS01:~/virginia$ ./he
Valor de a: 5
Via ponteiro: 5
Via ponteiro para ponteiro: 5
Primeiro elemento do array via ponteiro: 1
Segundo elemento: 2
```



# Ponteiros

- Exemplo com objeto

```
GNU nano 6.2
#include <iostream>
using namespace std;

class Pessoa {
public:
    string nome;
};

int main() {
    Pessoa* p = new Pessoa;
    p->nome = "Maria";

    cout << "Nome: " << p->nome << endl;

    delete p;
    return 0;
}
```



```
aluno@microengenhariaAWS01:~/
Nome: Maria
```

# Ponteiros

- Passagem por endereço:
  - A função recebe o endereço da variável;
  - A função acessa o conteúdo de x através do ponteiro p;
  - A variável original é modificada;
  - É necessário usar o operador \* para acessar o valor e & para passar o endereço.
- Passagem por referência:
  - A variável é passada como apelido direto, sem precisar de ponteiro ou operadores \* e & dentro da função;
  - A função age como se estivesse trabalhando diretamente com a variável x;
  - Mais simples e segura para uso geral;
  - Não precisa usar \* nem &.

```
GNU nano 6.2
#include <iostream>
using namespace std;

void porEndereco(int* p) {
    *p = *p + 10;
}

void porReferencia(int& x) {
    x = x + 10;
}

int main() {
    int a = 5, b = 5;

    porEndereco(&a);
    porReferencia(b);

    cout << "Resultado por endereco: " << a << endl;
    cout << "Resultado por referencia: " << b << endl;

    return 0;
}
```

```
[[Aadun@milic10engennat1:AW301.
Resultado por endereco: 15
Resultado por referencia: 15
]]
```



# Ponteiros

- **Ponteiro para função:**
  - É uma variável que armazena o endereço de uma função, permitindo chamar funções dinamicamente;
  - Chamar funções de forma indireta;
  - Passar funções como parâmetro para outras funções;
  - Criar estruturas flexíveis e reutilizáveis, como menus, filtros e callbacks.

```
GNU nano 6.2
#include <iostream>
using namespace std;

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

int main() {
    int (*operacao)(int, int);
    operacao = &soma;

    cout << "Resultado da funcao via ponteiro: " << operacao(4, 6) << endl;

    return 0;
}
```

```
aluno@microengenhariaAWS01:~/virginia$
Resultado da funcao via ponteiro: 10
```

