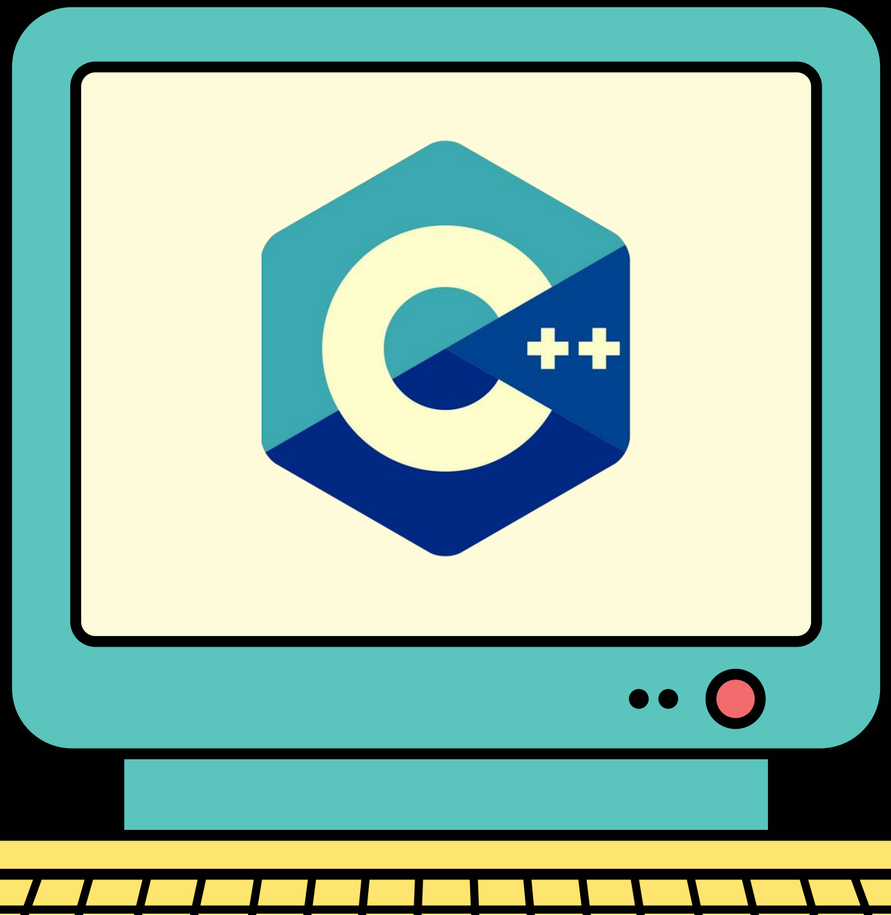
APRENDENDO



COMANDOS DE I/O

LINGUAGEM C

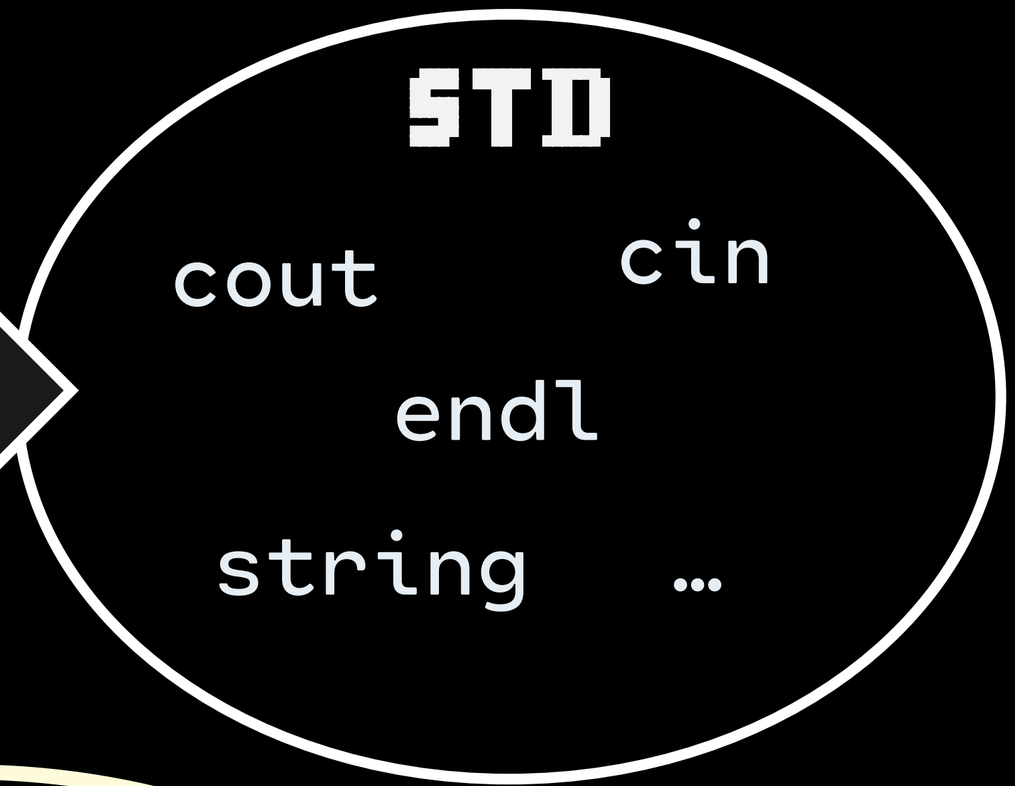
```c
#include <stdio.h>

int main()
{
  char nome[100];

  printf("Digite seu nome: ");
  scanf("%s", &nome);

  printf("Olá %s\n", &nome);
  return 0;
}
```

LINGUAGEM C++

```cpp
#include <iostream>

int main()
{
  std::string nome;

  std::cout << "Digite seu nome: ";
  std::cin >> nome;

  std::cout << "Ola " << nome << std::endl;
  return 0;
}
```

NAMESPACE STD

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



NAMESPACE STD

```cpp
#include <iostream>

using namespace std;

int main()
{
  string nome;

  cout << "Digite seu nome: ";
  cin >> nome;

  cout << "Ola " << nome << endl;
  return 0;
}
```

TIPOS

BOOLEANOS (LOGICOS)

```
bool maior = 5 > 6; // true ou false -> 1 byte
bool vivo = true;
```

CARACTERES/TEXTO

```
char caractere = 'A';        // 1 byte
string cadeia = "c mais mais"; // Depende da quantidade de caracteres
```

TIPOS

NUMEROS INTEIROS

```
short curto = 1000;                  // -32.768 - 32.767 -> 2 bytes
int inteiro = 1000;                  // 4 bytes
long longo = 10000000;               // 4 ~ 8 bytes
long long muitoLongo = 1000000000; // 8 bytes
```

NUMEROS REAIS

```
float flutuante = 8.16;                  // 4 bytes
double dobroFlutuante = 8.1678283;       // 8 bytes
long double flutuanteLongo = 8.82347236; // 8 ~ 16 bytes
```

OPERADORES ARITMETICOS

FUNCIONAM SOMENTE COM NUMEROS

```
+ - * / %
```

RETORNAM UM NUMERO

OPERADORES ARITMETICOS

DIVISAO INTEIRA

```
int dividendo = 10, divisor = 3, quociente;
quociente = 10 / 3;
cout << quociente // quociente == 3;
```

DIVISAO REAL

```
float dividendo = 10, divisor = 3, quociente;
quociente = 10 / 3;
cout << quociente // quociente == 3.33333;
```

OPERADORES ARITMETICOS

MODULO

FUNCIONA SOMENTE COM NUMEROS INTEIROS

```
int dividendo = 10, divisor = 3, resto;
resto = 10 % 3;
cout << resto;
```

EXERCICIOS 01

Fazer os exercícios de algoritmos “**SEQUÊNCIAS BÁSICAS**”, do 1 ao 16

OPERADORES RELACIONAIS

EXEMPLOS

```
int n1 = 1, n2 = 2;
5 > 6; // false
5 < 6; // true

n1 == n2; // false
n1 != n2 // true

n1 <= 5; // true
5 <= 5; // true
```

OPERADORES LOGICOS

RETORNAM UM VALOR BOOLEANO

APARECEM JUNTO COM OUTROS VALORES LOGICOS
OU RELACOES QUE RETORNEM VALORES LOGICOS

```
! && ||
```

TABELA VERDADE

AND - &&

v1	v2	&&
true	true	true
true	false	false
false	true	false
false	false	false

TABELA VERDADE

OR - ||

v1	v2	||
true	true	true
true	false	true
false	true	true
false	false	false

TABELA VERDADE

NOT - !

v	!
true	false
false	true

ESTRUTURAS CONDICIONAIS

IF - SE...

```
if (true)
{
  // executa o código aqui dentro
}
```

```
if (false)
{
  // não executa o código aqui dentro
}
```

ESCOPO

ESCOPO SERIA A VISIBILIDADE DE UMA VARIAVEL. NORMALMENTE, VEMOS DIFERENTES ESCOPOS SENDO REPRESENTADOS POR UM PAR DE CHAVES { }

EXERCICIOS 02

Fazer os exercícios de algoritmos **“CONDIÇÕES BÁSICAS”**, do 17 ao 25

ESTRUTURAS CONDICIONAIS

IF/ELSE - SE...SENÃO...

```
if (true)
{
  // executa o código aqui dentro
} else {
  // não executa o código aqui dentro
}
```

```
if (false)
{
  // não executa o código aqui dentro
} else {
  // executa o código aqui dentro
}
```

ESTRUTURAS CONDICIONAIS

IF/ELSE IF - SE...SENÃO SE...

```
if (true)
{
  // executa o código aqui dentro
}
else if (true /* ou false */)
{
  // não executa o código aqui dentro
}
else // opcional
{
  // não executa o código aqui dentro
}
```

```
if (false)
{
  // não executa o código aqui dentro
}
else if (true)
{
  // executa o código aqui dentro
}
else // opcional
{
  // não executa o código aqui dentro
}
```

ESTRUTURAS CONDICIONAIS

IF/ELSE IF - SE...SENÃO SE...

```
if (true)
{
  // não executa o código aqui dentro
}
else if (false)
{
  // não executa o código aqui dentro
}
else // opcional
{
  // executa o código aqui dentro
}
```

```
if (/*valor booleano*/) {}
else if (/*valor booleano*/){}
else if (/*valor booleano*/) {}
else if (/*valor booleano*/) {}
else if (/*valor booleano*/) {}
else if (/*valor booleano*/) {}
else if (/*valor booleano*/) {}
else if (/*valor booleano*/) {}
else if (/*valor booleano*/) {}
else {} // se precisar
```

EXERCICIOS 03

Fazer os exercícios de algoritmos **“CONDIÇÕES COMPOSTAS”**, do 26 ao 37

OPERADORES DE INCREMENTO

PRE INCREMENTO

```
int a = 5;
int b = ++a; // a == 6, b == 6
```

POS INCREMENTO

```
int a = 5;
int b = a++; // a == 6, b == 5
```

OPERADORES DE DECREMENTO

PRE DECREMENTO

```
int a = 5;
int b = --a; // a == 4, b == 4
```

POS DECREMENTO

```
int a = 5;
int b = a--; // a == 4, b == 5
```

OPERADORES DE ATRIBUICAO

FORMADOS POR:

OPERADOR ARITMETICO E "RECEBE"

```
int a = 10, b = 3;

a += b; // a = a + b -> a = 13
a -= b; // a = a - b -> a = 7
a *= b; // a = a * b -> a = 30
a /= b; // a = a / b -> a = 3, pois é divisão inteira
a %= b; // a = a % b -> a = 1
```

ESTRUTURAS DE REPETICAO

WHILE - ENQUANTO...

```
while (/* condição */)
{
  // executa o que está aqui dentro
} // quando a condição for falsa, volta ao código normal
```

EXERCICIOS 04

Fazer os exercícios de algoritmos **“REPETIÇÕES ENQUANTO”**, do 38 ao 55

ESTRUTURAS DE REPETICAO

FLAGS - BREAK

```
while (/* condição */)
{
  if (/* alguma condição */)
  {
    // código...
    break; // sai do loop
  }
}
```

ESTRUTURAS DE REPETICAO

FLAGS - CONTINUE

```
while (/* condição */)
{
  if (/* alguma condição */)
  {
    // código...
    continue; // pula para a próxima etapa do loop
  }
}
```

EXERCICIOS 05

Fazer os exercícios de algoritmos **“ENQUANTO COM FLAG”**, do 56 ao 60

ESTRUTURAS DE REPETICAO

DO...WHILE - FACA ENQUANTO

O CODIGO DENTRO DA ESTRUTURA VAI EXECUTAR UMA PRIMEIRA VEZ E SO VAI EXECUTAR NOVAMENTE SE DETERMINADA CONDICAO FOR VERDADEIRA, E VAI CONTINUAR EXECUTANDO ENQUANTO TAL CONDICAO PERMANECER VERDADEIRA

```
do {
  // código...
} while(/* condição */);
```

EXERCICIOS 06

Fazer os exercícios de algoritmos **“REPETIÇÃO COM FAÇA ENQUANTO”**, do 61 ao 63

ESTRUTURAS DE REPETICAO

FOR - PARA

```
for (int i = 0; i < 10; i++)
{
  // vai executar o que está aqui 10 vezes
}

for (int i=0; i<10; i+=2)
{
  // vai executar o que está aqui 5 vezes
}
```

EXERCICIOS 07

Fazer os exercícios de algoritmos **“REPETIÇÃO COM PARA”**, do 64 ao 70

ESTRUTURAS DE DADOS

ARRAYS - MATRIZES

ESTRUTURAS DE DADOS SAO VARIAVEIS QUE PODEM ARMAZENAR DIVERSAS OUTRAS VARIAVEIS.

A ESTUTURA MAIS BASICA PRESENTE EM QUASE TODA LINGUAGEM EH O **ARRAY** (EM PORTUGUES, MATRIZ).

NA PROGRAMACAO, CHAMAMOS MATRIZES DE 1 DIMENSAO DE **VETOR**, POR CONTA DE UMA ANALOGIA MATEMATICA.

ESTRUTURAS DE DADOS

ARRAYS - MATRIZES

ELES POSSUEM UMA ESTRUTURA DENOMINADA DE **CHAVE-VALOR**, ONDE CADA VALOR TEM UMA CHAVE CORRESPONDENTE.

OS **INDICES** (AS CHAVES) DE UM VETOR VAO DE **0 ATE N-1** (COM N SENDO O TAMANHO/QUANTIDADE DE VALORES DO VETOR).

ESTRUTURAS DE DADOS

ARRAYS - MATRIZES

OS **ELEMENTOS** (OS VALORES) DE UM ARRAY SAO TODOS DE UM MESMO TIPO, DETERMINADO NA DECLARACAO DO VETOR.

O **TAMANHO** DE UM ARRAY TAMBEM EH DETERMINADO EM SUA DECLARACAO, JAMAIS PODENDO SER ALTERADO DURANTE A EXECUCAO DO PROGRAMA.

ESTRUTURAS DE DADOS

ARRAYS - MATRIZES

STRINGS SAO ARRAYS DE CHAR, COMO DEMONSTRADO NESSE CODIGO FEITO EM C:

```c
#include <stdio.h>

int main()
{
  char nome[100];

  printf("Digite seu nome: ");
  scanf("%s", &nome);

  printf("Olá %s\n", &nome);
  return 0;
}
```

EM C++, O TIPO **STRING** EH UMA ABSTRACAO DE UM VETOR DE CARACTERES, MAS ISSO EH ASSUNTO PARA OUTRO MOMENTO...

ESTRUTURAS DE DADOS

ARRAYS - MATRIZES

```
int tamanho = 5;
int numeros[tamanho] = {10, 20, 30, 40, 50};

for (int i = 0; i < 5; i++)
{
  cout << numeros[i] << "\n";
} // os indices de um array vão de 0 até n-1, com n sendo o
                        tamanho do vetor
```

ESTRUTURAS DE DADOS

ARRAYS - MATRIZES

```
int tamanho = 5;
int numeros[tamanho];

for (int i = 0; i < 5; i++)
{
    cin >> numeros[i];
}
```

ESTRUTURAS DE DADOS

ARRAYS - MATRIZES

ARRAYS DE ARRAYS TAMBEM SAO POSSIVEIS, E ESTES NOS CHAMAMOS DE MATRIZES MESMO.

```
int qtdLinhas = 3, qtdColunas = 2;
int matriz[qtdLinhas][qtdColunas] = {
  {1, 2},
  {3, 4},
  {5, 6}
};
```

ESTRUTURAS DE DADOS

ARRAYS - MATRIZES

ARRAYS DE ARRAYS TAMBEM SAO POSSIVEIS, E ESTES NOS CHAMAMOS DE MATRIZES MESMO.

```
int tamanhoVetorzao = 3, tamanhoVetorzinhos = 2;
int matriz[tamanhoVetorzao][tamanhoVetorzinhos] = {
    {1, 2},
    {3, 4},
    {5, 6}
};
```

EXERCICIOS 08

Fazer os exercícios de algoritmos “**VETORES**”, do 71 ao 85

NOVO TIPO - VOID

O TIPO VOID REPRESENTA, NA PRATICA, **NADA**.

NO EXEMPLO ABAIXO, ESTOU PASSANDO "NADA" PARA A FUNCAO MAIN.

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

```c
int main(void)
{
  return 0;
}
```

SAO A MESMA COISA NA PRATICA!

FUNCOES

AS FUNCOES SAO BLOCOS DE CODIGO REUTILIZAVEIS.

```cpp
#include <iostream>

using namespace std;

void escreverOlaMundo()
{
  cout << "Ola Mundo";
}
```

```cpp
int main()
{
  escreverOlaMundo();
  return 0;
}
```

FUNCOES

ELAS SAO FORMADAS POR:

```
tipoDeRetorno nomeDaFuncao(parametros)
{
  /* bloco de código */
}
```

O **RETORNO** DE UMA FUNCAO EH UM VALOR QUE ELA CRIA PARA A GENTE.

FUNCOES

EXEMPLOS:

```c
int somarDobroDoisInteiros(int n1, int n2)
{
  int dobro1 = n1 * 2, dobro2 = n2 * 2;
  return dobro1 + dobro2;
}

int main()
{
  int v = somarDobroDoisInteiros(4, 9); // 26
  return 0;
}
```

FUNCOES

EXEMPLOS:

```
void escreverMensagem(string mensagem) // função sem retorno
{
  cout << mensagem << endl;
}

int main()
{
  escreverMensagem("Hello World!"); // só precisa executar código
  // não tem motivo para devolver um valor
  return 0;
}
```

FUNCOES

EXEMPLOS:

```
int main()
{
  escreverMensagem("Hello World!");
  escreverMensagem("Ola mundo!");
  escreverMensagem("Ola C++!");
  escreverMensagem("Posso usar quantas vezes eu quiser!")

  return 0;
}
```

EXERCICIOS 09 E 10

Fazer os exercícios de algoritmos **“PROCEDIMENTOS”**, e **“FUNÇÕES”**, do 86 ao 100