Introdução - AEDI

Prof. Paulo Henrique Pisani

fevereiro/2019

Algoritmos e Estruturas de Dados I

- Ementa:
  - Breve introdução à linguagem C;
  - Noções básicas de análise de complexidade de tempo de algoritmos;
  - Estruturas lineares: busca e ordenação;
  - Árvores de busca;
  - Árvores balanceadas.

Algoritmos e Estruturas de Dados I

- Detalhes sobre regras da disciplinas (cronograma, avaliação, material, etc) disponíveis em: https://folivetti.github.io/teaching/2019-summer-teaching-1

- A presença nas aulas será controlada por lista de presença;
- Plantão de atendimento (dias e horários no link acima);
- Cadastre-se no Piazza da disciplina (veja no link acima).

Algoritmos e Estruturas de Dados I

- Na aula de hoje, repassaremos alguns tópicos básicos sobre linguagem C, assim como uma introdução ao ambiente de programação.

Revisão C

Programa mínimo

```
#include <stdio.h>
int main() {
        printf("ABC");
        return 0;
}
```

- **Para compilar:**

gcc teste.c -o teste.exe

- **Para executar:**

./teste.exe

Programa (realmente) mínimo

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

- **Para compilar:**

gcc teste.c -o teste.exe

- **Para executar:**

./teste.exe

Tipos de dados

int	Atualmente, é o long int
short int	Inteiro de 2 bytes (-32.768 a 32.767)
long int	Inteiro de 4 bytes (-2^31 a 2^31-1)
long long int	Inteiro de 8 bytes (-2^63 a 2^63-1)
unsigned int	Versões dos tipos inteiro “sem sinal”
unsigned short int	
unsigned long int	
unsigned long long int	

float	Ponto flutuante de 4 bytes
double	Precisão dupla de 8 bytes

Tipos de dados

Importante! Atenção aos limites dos tipos!

int	Atualmente, é o long int
short int	Inteiro de 2 bytes (-32.768 a 32.767)
long int	Inteiro de 4 bytes (-2^31 a 2^31-1)
long long int	Inteiro de 8 bytes (-2^63 a 2^63-1)
unsigned int	Versões dos tipos inteiro “sem sinal”
unsigned short int	
unsigned long int	
unsigned long long int	

float	Ponto flutuante de 4 bytes
double	Precisão dupla de 8 bytes

Tipos de dados

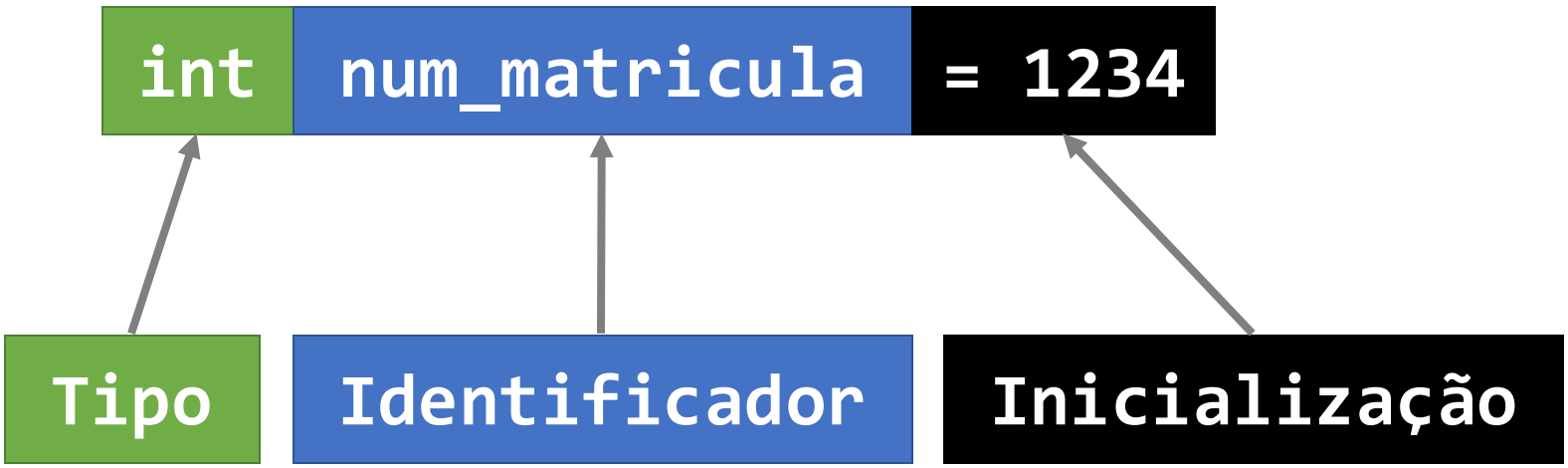
char	1 byte (-128 a 127)
unsigned char	1 byte (0 a 255)

Não há um tipo String. Strings são representadas por vetores de char (e o último char é o '\0').

Também não há tipo booleano! Podemos usar int ou char:
Valor = 0 -> FALSO
Valor != 0 -> VERDADEIRO

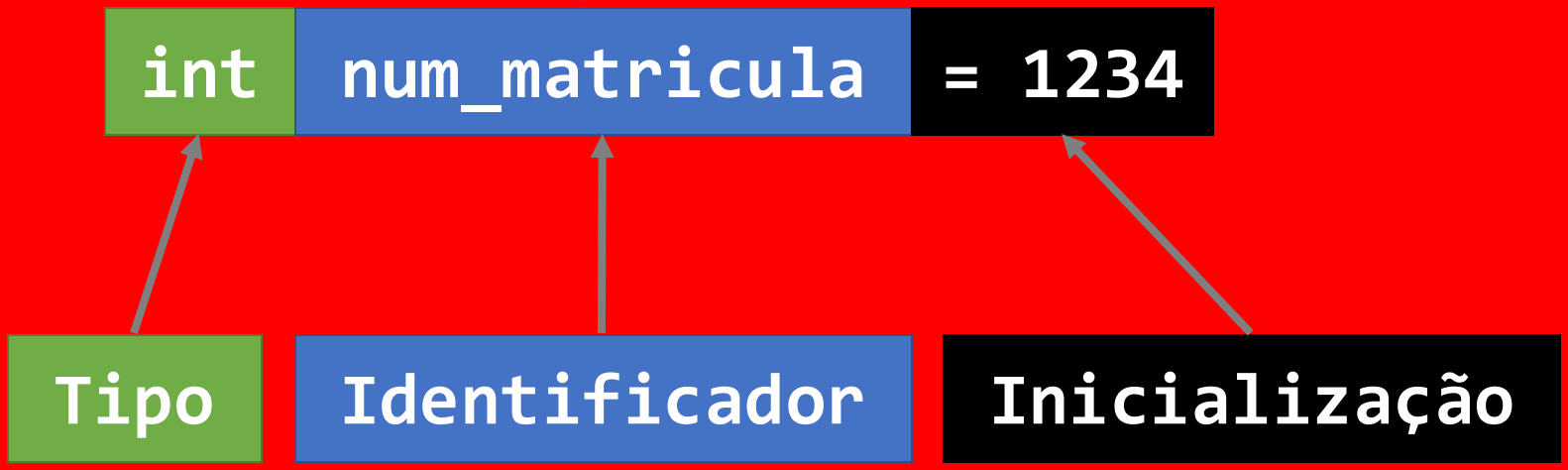
Declaração de variáveis

```
int main() {
        int num_matricula = 1234;
        return 0;
}
```



Cuidado! A linguagem C é *case-sensitive*, ou seja: A identificador "num_matricula" é diferente de "Num_matricula"!

```
int main() {
        int num_matricula = 1234;
        return 0;
}
```



Saída

- Exemplos:

%d	int
%ld	long int
%lld	long long int
%f	float
%lf	double
%c	char
%s	String (vetor de char)
%p	Ponteiro (endereço de memória)

```
#include <stdio.h>
int main() {
    printf("ABC");
    int num = 507;
    printf("%d", num);
    printf("%d\n", num);
    printf("A sala do professor eh a %d\n", num);
    printf("%c + %c = %d\n", 'A', 'B', num);
    return 0;
}
```

Entrada

- Exemplos:

```
#include<stdio.h>

int main() {
    printf("Digite um numero inteiro:\n");
    int num_int;
    scanf("%d", &num_int);
    return 0;
}
```

&num_int

Há um "&" antes do identificador da variável!!!

O scanf recebe os **endereços de memória** das variáveis. O "&" serve para obter o endereço de memória da variável "num_int". Quando trabalharmos com ponteiros, veremos que nem sempre é necessário usar o "&".

Conversão de tipo

- Atenção com operações envolvendo tipos fracionários!

```
float num;

num = 5 / 2;
num = 5 / 2.0;
num = 5 / ((float) 2);
```

Inicialização de variáveis

- **Sempre inicialize variáveis em C!**

- Quando uma variável é declarada, o programa apenas reserva um espaço de memória para ela:
  - **Mas o espaço alocado não é inicializado!**
  - **Portanto, uma variável não inicializada pode ter qualquer valor!**

Sempre inicialize variáveis!

- Qual a saída deste programa?

```
#include<stdio.h>

int main() {

    int num;

    printf("%d\n", num);

    return 0;
}
```

Sempre inicialize variáveis!

- Qual a saída deste programa?

```
#include<stdio.h>

int main() {

    int num;

    printf("%d\n", num);

    return 0;
}
```

Variável não foi inicializada! A saída será o que estiver na área alocada! Pode ser qualquer valor!

Vetores

- É um conjunto de variáveis do mesmo tipo:
  - Referenciada por um mesmo identificador;
  - Cada elemento é acessado por meio de um índice.

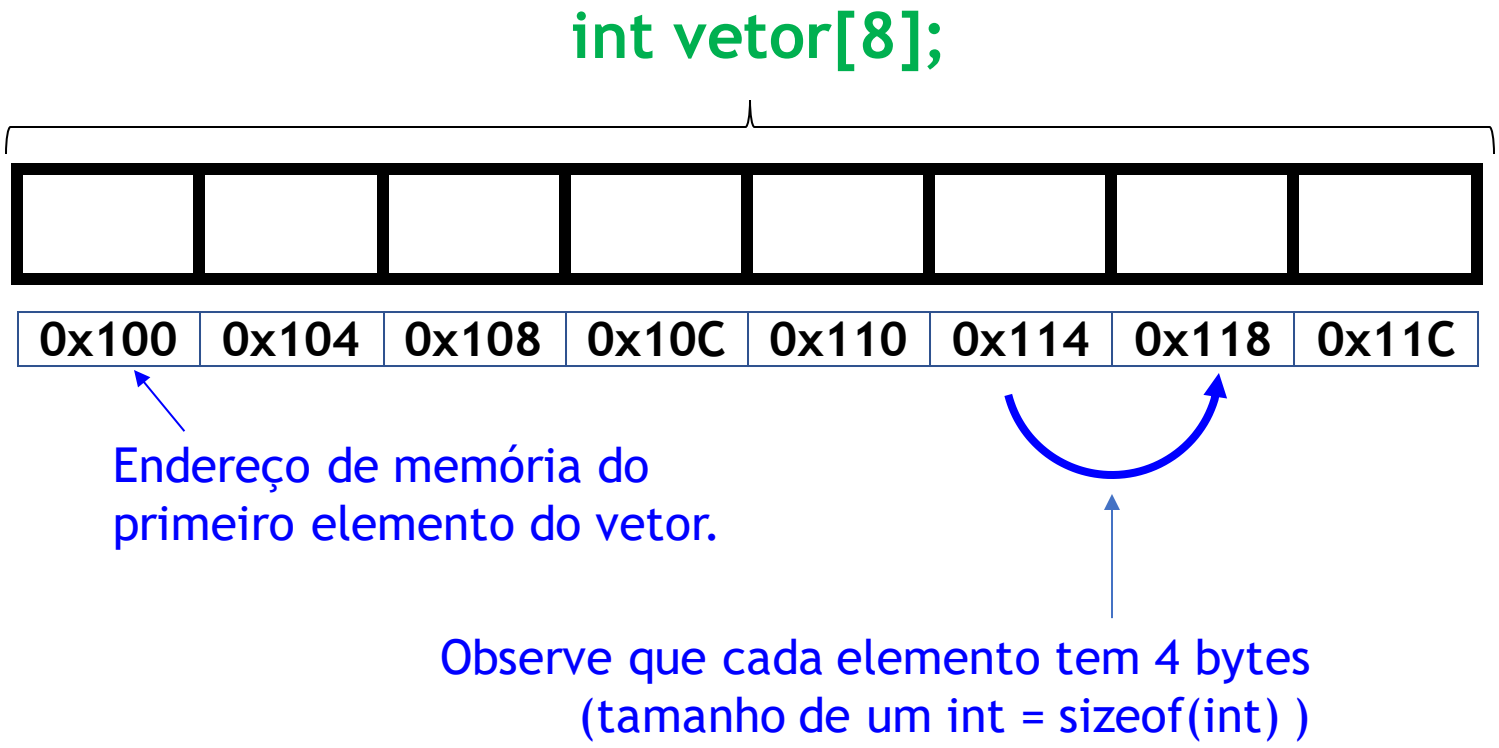
- **Declarar** vetor:

```
<tipo> <nome>[<tamanho>];
```

Exemplos

```
int idades[10];
double vetor2[5];
int valores[3] = {10, 20, 30};
```

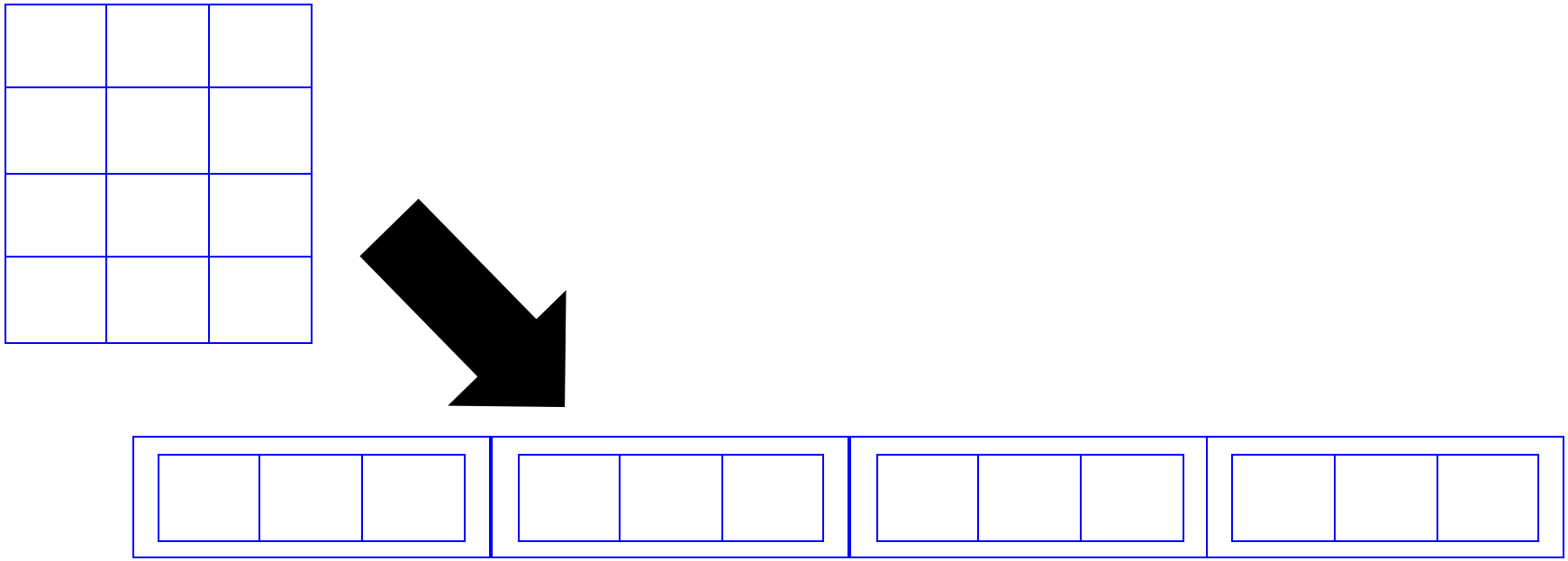
Vetores são armazenados em posições consecutivas na memória!



Este vetor ocupa 8 * 4 = 32 bytes
Ou seja, **`tamanho * sizeof(<tipo>)`**

Matriz é um vetor de vetores

- Internamente, a matriz é um vetor unidimensional, em que cada elemento é um vetor unidimensional.

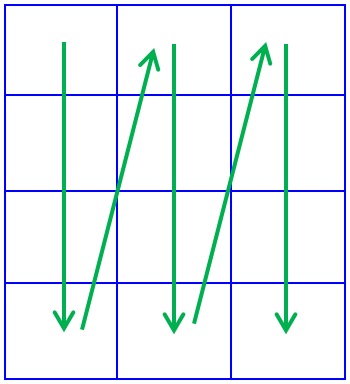


Percurso em Matrizes

Percorre colunas

```
int i, j;
int matriz[4][3];
for (int j = 0; j < 3; j++)
    for (int i = 0; i < 4; i++)
        matriz[i][j];
```

Percorre cada elemento na coluna



Dicas gerais

Escopo de variáveis

- O **escopo** de uma variável determina de onde ela pode ser acessada;
- Em C, existem dois escopos principais:
  - **Variáveis locais**: acessíveis apenas localmente (pela função);
  - **Variáveis globais**: acessíveis a partir de qualquer parte do código.

```c
#include<stdio.h>

double resultado;

void calcula_quadrado(double num) {
    resultado = num * num;
}

double calcula_soma(double n1, double n2) {
    double r;
    r = n1 + n2;
    return r;
}

int main() {
    int a = 2, b = 3;
    resultado = calcula_soma(a, b);
    printf("%.2lf\n", resultado);
    calcula_quadrado(resultado);
    printf("%.2lf\n", resultado);
    calcula_quadrado(resultado);
    printf("%.2lf\n", resultado);

    return 0;
}
```

Variável global

A variável resultado pode ser acessada a partir de qualquer função!

Variável local da função calcula_soma

Variáveis locais da função main

Variáveis locais são acessíveis apenas da função onde forma declaradas.

```c
#include<stdio.h>

double resultado;

void calcula_quadrado(double num) {
    resultado = num * num;
}

double calcula_soma(double n1, double n2) {
    double r;
    r = n1 + n2;
    return r;
}

int main() {
    int a = 2, b = 3;
    resultado = calcula_soma(a, b);
    printf("%.2lf\n", resultado);
    calcula_quadrado(resultado);
    printf("%.2lf\n", resultado);
    calcula_quadrado(resultado);
    printf("%.2lf\n", resultado);

    return 0;
}
```

O que será impresso?

```c
#include<stdio.h>

double resultado;

void calcula_quadrado(double num) {
    resultado = num * num;
}

double calcula_soma(double n1, double n2) {
    double r;
    r = n1 + n2;
    return r;
}

int main() {
    int a = 2, b = 3;
    resultado = calcula_soma(a, b);
    printf("%.2lf\n", resultado);
    calcula_quadrado(resultado);
    printf("%.2lf\n", resultado);
    calcula_quadrado(resultado);
    printf("%.2lf\n", resultado);

    return 0;
}
```

O que será impresso?

```
5.00
25.00
625.00
```

Veja que o uso de variáveis globais dificulta a leitura do código (e pode levar a erros de programação).

Portanto, evite variáveis globais ao máximo!

Inicialização de variáveis

- **Sempre inicialize variáveis em C!**

- Quando uma variável é declarada, o programa apenas reserva um espaço de memória para ela:
  - **Mas o espaço alocado não é inicializado!**
  - **Portanto, uma variável não inicializada pode ter qualquer valor!**

Importante!

Observe o **estilo de codificação** adotado nos slides:

- Indentação;
- Posição das chaves;
- Nomenclatura de variáveis.

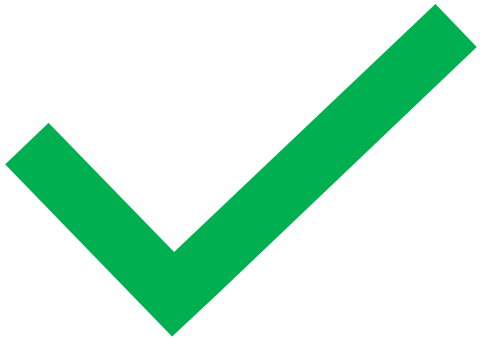
Também use nomes representativos para variáveis!

Evite usar tmp1, tmp2, aux1, aux2, aux50, etc.

Indentação

- Cuidado ao indentar vários blocos de código!!!

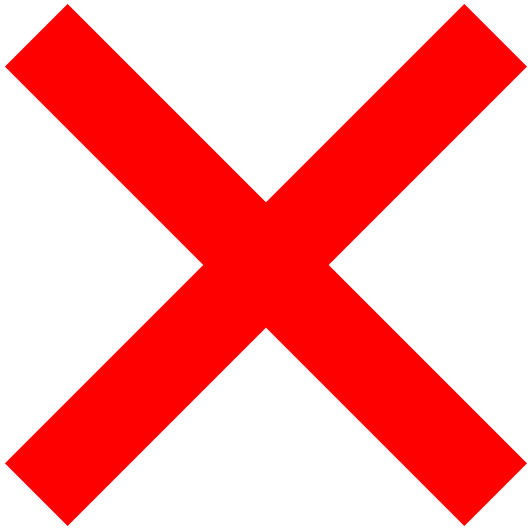
```
void funcao_teste(int param1) {
    int a = param1;
    if (param1 > 0) {
        int b = 0;
        int i;
        for (i = 0; i < 10; i++) {
            int c = i * i;
            b += c;
            printf("%d %d %d", a, b, c);
        }
        printf("%d %d %d", a, b, c);
    }
    printf("%d %d %d", a, b, c);
}
```



Indentação

- **Cuidado ao indentar vários blocos de código!!!**

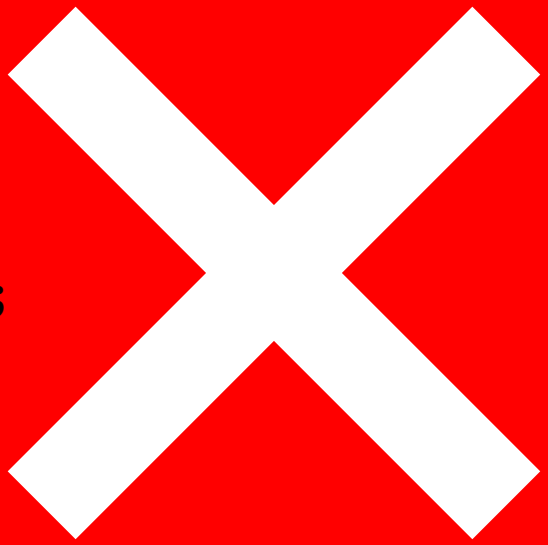
```
void funcao_teste(int param1) {
int a = param1;
if (param1 > 0) {
int b = 0;
int i;
for (i = 0; i < 10; i++) {
int c = i * i;
b += c;
printf("%d %d %d", a, b, c);
}
printf("%d %d %d", a, b, c);
}
printf("%d %d %d", a, b, c);
}
```



Indentação

- **Cuidado ao indentar vários blocos de código!!!**

```
void funcao_teste(int param1) {
    int a = param1;
    if (param1 > 0) {
        int b = 0;
        int i;
        for (i = 0; i < 10; i++) {
            int c = i * i;
            b += c;
            printf("%d %d %d", a, b, c);
    }
    printf("%d %d %d", a, b, c);
    }
    printf("%d %d %d", a, b, c);
}
```



Depuração

- E quando meu programa não funcionar?

Depuração

- E quando meu programa não funcionar?

- Esta é uma excelente oportunidade para encontrar o erro e aprender como evitá-lo!
  - Aplique técnicas de depuração (debug).
- Quando o programa segue algumas simples boas práticas (e.g. indentação correta, nomes significativos para variáveis, etc), o código fica mais legível:
  - **Ocorrem menos erros de codificação;**
  - **É mais fácil encontrar os erros.**

Ambiente de programação

Ambiente de programação

- Utilizaremos o gcc
- IDE:
  - Editor de texto (e.g. gedit, vim, etc)
  - VS Code
  - Codeblocks

Material Complementar

- Kernighan & Ritchie. C A Linguagem de Programação Padrão ANSI. Ed. Campus

Referências

- Slides do Prof. Paulo H. Pisani: Programação Estruturada (Q3/2018)
- Slides do Prof. Fabrício Olivetti:
  - http://folivetti.github.io/courses/ProgramacaoEstruturada/
- Slide do Prof. Monael Pinheiro Ribeiro:
  - https://sites.google.com/site/aed2018q1/
- PINHEIRO, F. A. C. Elementos de programação em C. Porto Alegre, RS: Bookman, 2012.
- CELES, W.; CERQUEIRA, R.; RANGEL, J. L. Introdução a Estruturas de Dados. Elsevier/Campus, 2004.