

# Introdução à Linguagem C

Prof. Paulo Henrique Pisani

setembro/2021

# Tópicos

- Linguagens de programação
- Compilação
- Linguagem C:
  - Programa mínimo
  - Tipos de dados
  - Entrada/Saída
  - Operadores aritméticos, condicionais, lógicos
  - Estruturas de repetição

# Algoritmos

- É uma sequência de etapas bem definidas para realizar algo;
- Exemplos:
  - Algoritmo para verificar se um número é primo;
  - Algoritmo para realizar o somatório dos elementos de um vetor.

# CPU

- A unidade central de processamento (CPU - Central Processing Unit) executa sequências de instruções:
  - Cada CPU possui um conjunto finito de instruções;
  - Os **algoritmos** são então escritos utilizando esse conjunto de instruções.

# Linguagens de programação

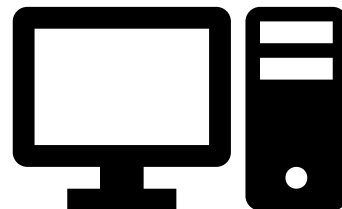
- Uma **linguagem de programação** é uma linguagem bem definida e sem ambiguidades utilizada para se comunicar com o computador.
- Inicialmente, era restrita às instruções da CPU;
  - Maior dificuldade para programar e entender o código (uma simples impressão na tela necessitava de várias instruções).



# Linguagens de programação

- Para lidar com esse problema, foram propostas linguagens de programação que serviriam como **intermediárias** entre a linguagem de máquina e o programador!

Linguagens de alto nível



# Linguagens de programação

- Podem ser compiladas ou interpretadas:
  - **Compilada**: o código-fonte é traduzido diretamente para código de máquina (executável).  
**Exemplo: C**
  - **Interpretada**: o código-fonte é traduzido para código de máquina apenas no momento da execução. Exemplo: Java

# Linguagens de programação

- Uma linguagem de programação pode ser classificada de acordo com o paradigma também:
  - Estruturado (e.g. C);
  - Orientado a Objetos (e.g. Java);
  - Funcional (e.g. Haskell);
  - etc.



# Compilação

- Traduz um código-fonte em uma dada linguagem para outra: traduz código-fonte em linguagem C para linguagem de máquina;
- O **programa objeto** pode então ser executado.



# Compilação

- Existem diversos compiladores para linguagem C: gcc (MinGW), Microsoft C/C++, Borland Turbo C, etc.



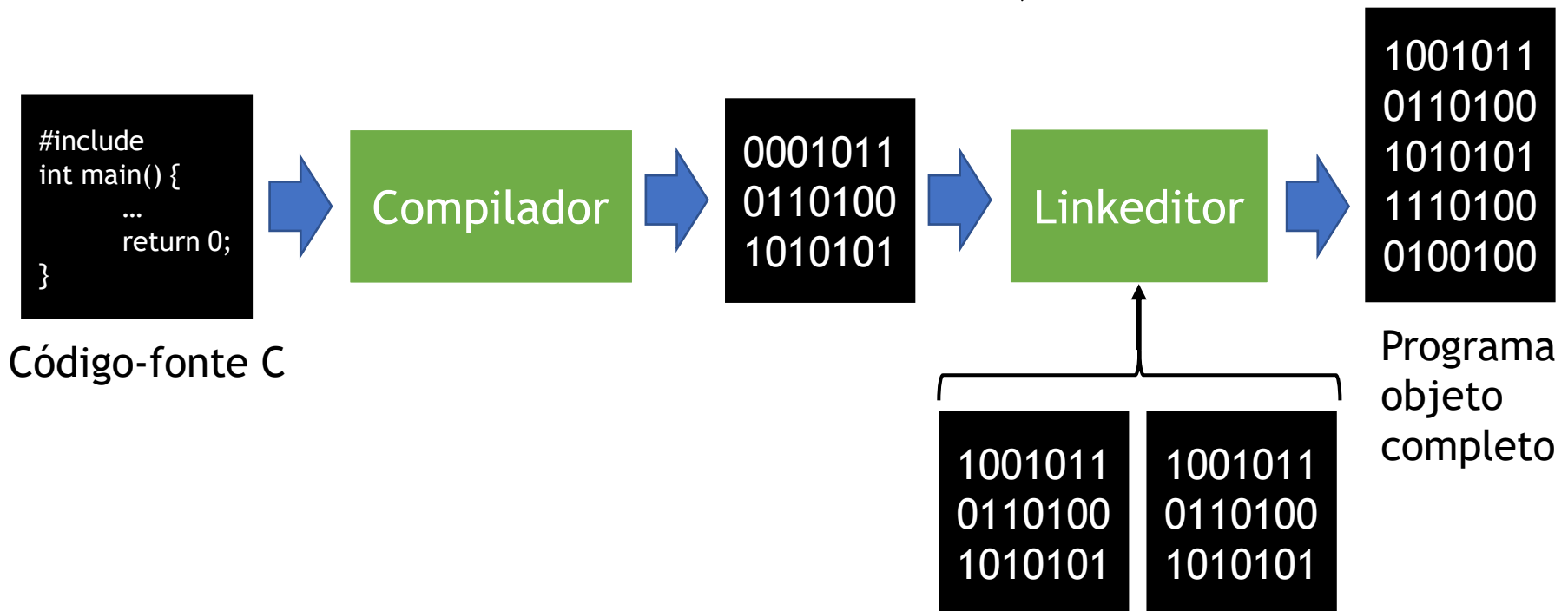
# Compilação

- O programa objeto é compilado para um sistema operacional específico: portanto, um programa compilado para MS Windows não executará no Linux diretamente.



# Compilação

- É possível compilar partes de um programa em C separadamente e depois juntá-las com o **linkeditor (processo de ligação)**: bastante usado em acesso a bibliotecas;



# Linguagem C

# Linguagem C

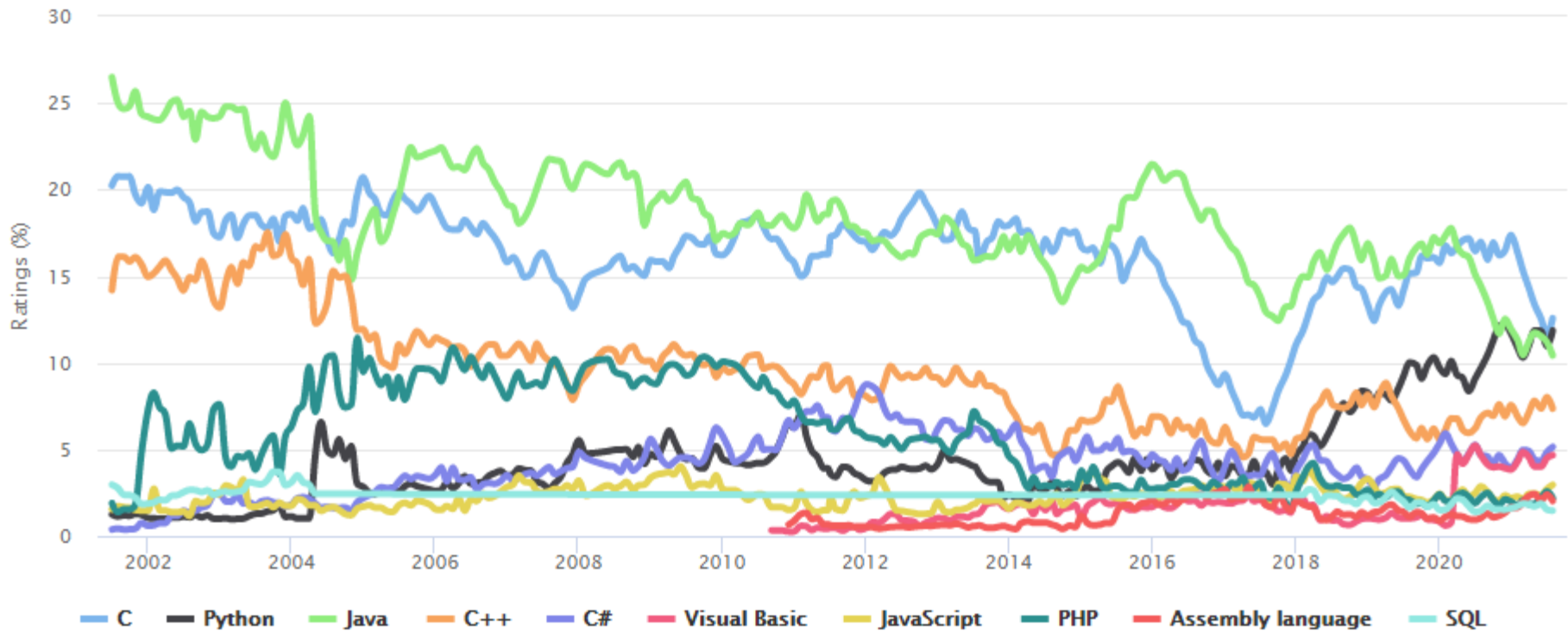
- Foi criada por Dennis Ritchie;
- Diversas variantes surgiram:
  - K & R (1978);
  - C89 (ANSI X3.159-1989 "Programming Language C.")
  - C99 (ISO/IEC 9899:1999)
  - C11 (ISO/IEC 9899:2011)
  - C18 (ISO/IEC 9899:2018)

Rank	Language	Type	Score
1	Python	  	100.0
2	Java	  	95.4
3	C	  	94.7
4	C++	  	92.4
5	JavaScript		88.1
6	C#	   	82.4
7	R		81.7
8	Go	 	77.7
9	HTML		75.4
10	Swift	 	70.4

<https://spectrum.ieee.org/top-programming-languages/> (31/08/2021)

## TIOBE Programming Community Index

Source: [www.tiobe.com](http://www.tiobe.com)



<https://www.tiobe.com/tiobe-index/> (31/08/2021)



# Programa mínimo

```
#include <stdio.h>

int main() {
    printf("ABC");
    return 0;
}
```

- Para compilar:

gcc teste.c -o teste.exe

Código-fonte

Programa objeto

- Para executar:

./teste.exe

# Programa (realmente) mínimo

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

- Para compilar:

gcc teste.c -o teste.exe

Código-fonte

Programa objeto

- 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)

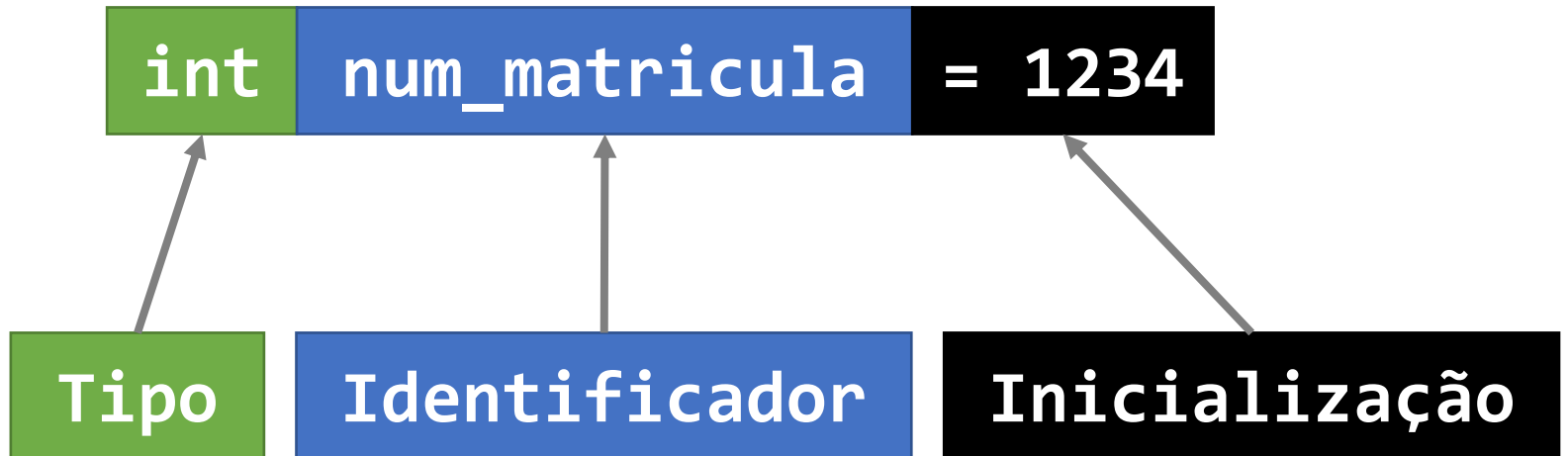
`unsigned char` 1 byte (0 a 255)

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

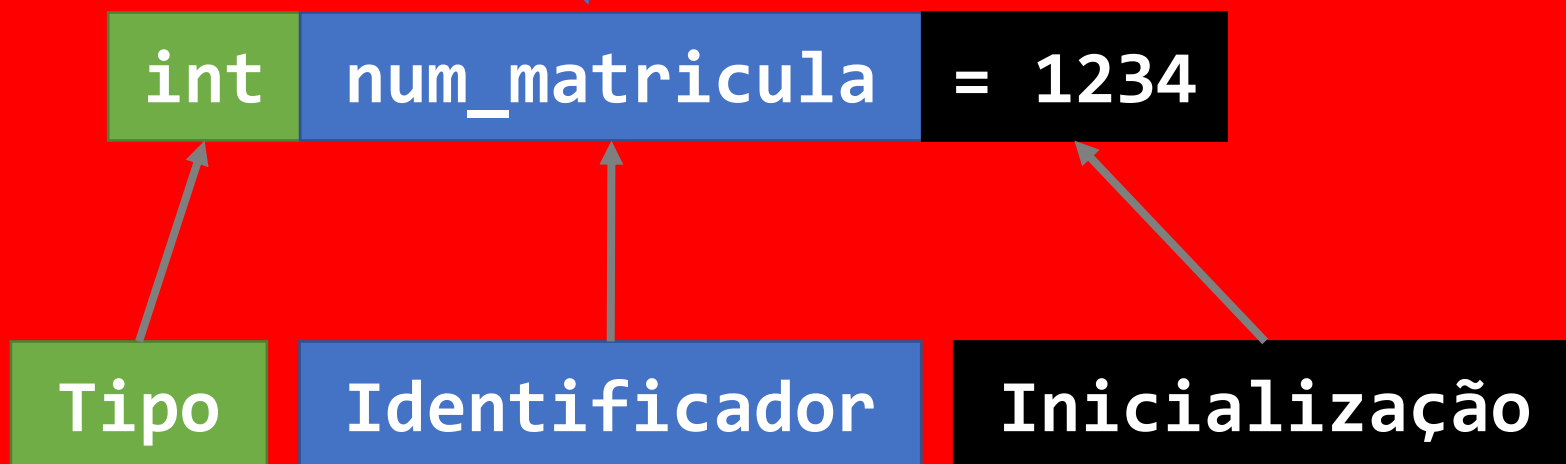
# Declaração de variáveis

```
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;  
}
```



# Entrada e Saída

- Na linguagem C, são utilizadas funções para as operações de entrada e saída;
- As funções básicas ficam na biblioteca padrão: a **stdio**!
- Para usar essa biblioteca, adicionamos a seguinte linha:




```
#include <stdio.h>
```



# Entrada e Saída

- Durante o curso, utilizaremos esta biblioteca em praticamente todos os programas, talvez em todos mesmo!
- Portanto, nosso programa mínimo torna-se:



```
#include <stdio.h>

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

# Saída

- Para imprimir um valor, usamos a função **printf**

```
int printf( const char * format, ... );
```

Formato



The diagram illustrates the components of the printf function signature. A blue arrow points from the word 'Formato' to the 'const char \* format' parameter. Another blue arrow points from the word 'Valores' to the '...' parameter.

Valores

# Saída

- Exemplos:

```
#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;
```

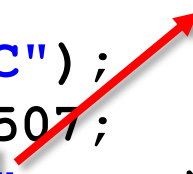
```
}
```

# Saída

- Exemplos:

```
#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;  
}
```



%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)

# Entrada

- Para ler um valor usamos a função **scanf**

```
int scanf( const char * format, ... );
```

Formato



Endereços  
das variáveis

# Entrada

- Exemplos:

```
#include<stdio.h>
```

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

# 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 “&”.



# Entrada

```
#include<stdio.h>
```

```
int main() {  
    printf("Digite um numero inteiro:\n");  
    int num_int;  
    scanf("%d", &num_int);  
  
    printf("Digite um numero fracionario:\n");  
    double num_frac;  
    scanf("%lf", &num_frac);  
  
    printf("INT=%d DOUBLE=%lf\n", num_int, num_frac);  
  
    int a, b;  
    printf("Digite dois numeros:\n");  
    scanf("%d %d", &a, &b);  
    printf("A=%d B=%d\n", a, b);  
  
    return 0;  
}
```

# Entrada e saída

Lembrar da incluir a biblioteca `stdio`!

- **printf**: recebe **valores** como argumento;
- **scanf**: recebe **endereços de memória** como argumento.



```
#include<stdio.h>
```

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

# Tamanhos dos tipos de dados (sizeof)

- Para saber quantos bytes um tipo de dados ocupa, usamos **sizeof** (o retorno é do tipo **long int**).

```
#include<stdio.h>
```

```
int main() {  
    int a;  
    long int b;  
    long long int c;  
    float d;  
    double e;  
    char f;
```

```
    printf("%ld %ld %ld %ld %ld %ld",  
           sizeof(a),  
           sizeof(b),  
           sizeof(c),  
           sizeof(d),  
           sizeof(e),  
           sizeof(f));
```

```
    return 0;
```

```
}
```

# Conversão de tipo

- Expressões com diversos tipos de dados:

```
#include<stdio.h>
```

```
int main() {
```

```
    float num;
```

```
    num = 4 / 2;
```

```
    printf("%.2f\n", num);
```

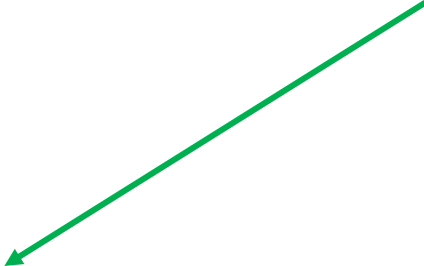
```
    num = 10 / 2.5;
```

```
    printf("%.2f\n", num);
```

```
    return 0;
```

```
}
```

Formata o número com  
duas casas decimais



Saída

?

# Conversão de tipo

- Expressões com diversos tipos de dados:

```
#include<stdio.h>
```

```
int main() {
```

```
    float num;
```

```
    num = 4 / 2;
```

```
    printf("%.2f\n", num);
```

```
    num = 10 / 2.5;
```

```
    printf("%.2f\n", num);
```

```
    return 0;
```

```
}
```

Saída

2.00

4.00

# Conversão de tipo

- Expressões com diversos tipos de dados:

```
#include<stdio.h>

int main() {

    float num, a=5, b=2;

    num = a / b;

    printf("%.2f\n", num);

    return 0;
}
```

Saída

?

# Conversão de tipo

- Expressões com diversos tipos de dados:

```
#include<stdio.h>

int main() {

    float num, a=5, b=2;

    num = a / b;

    printf("%.2f\n", num);

    return 0;
}
```

Saída

2.50

# Conversão de tipo

- Expressões com diversos tipos de dados:

```
#include<stdio.h>
```

```
int main() {
```

```
    float num;
```

```
    num = 4 / 2;
```

```
    printf("%.2f\n", num);
```

```
    num = 5 / 2;
```

```
    printf("%.2f\n", num);
```

```
    return 0;
```

```
}
```

Saída

?



# Conversão de tipo

- Expressões com diversos tipos de dados:

```
#include<stdio.h>
```

```
int main() {
```

```
    float num;
```

```
    num = 4 / 2;
```

```
    printf("%.2f\n", num);
```

```
    num = 5 / 2;
```

```
    printf("%.2f\n", num);
```

```
    return 0;
```

```
}
```

Saída

2.00

2.00

# Conversão de tipo

- Atenção com operações envolvendo tipos fracionários!

```
float num;
```

<code>num = 5 / 2;</code>	→	2
<code>num = 5 / 2.0;</code>	→	2.5
<code>num = 5 / ((float) 2);</code>	→	2.5

# Conversão de tipo

- Conversão entre tipos:

```
#include<stdio.h>
```

```
int main() {
```

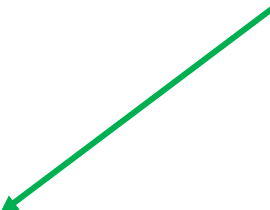
```
    int n1 = 507;  
    long long int n2 = n1;
```

```
    printf("%lld\n", n2);
```

```
    return 0;
```

```
}
```

Ok! Armazena valor (int) em  
variável com tipo de dados que  
possui faixa maior (long long int)



Saída

507

# Conversão de tipo

- Conversão entre tipos:

```
#include<stdio.h>
```

```
int main() {
```

```
    int n1 = 507;
```

```
    short int n2 = n1;
```

```
    printf("%d\n", n2);
```

```
    return 0;
```

```
}
```

**Perigoso! Armazena valor (int) em variável com tipo de dados que possui faixa MENOR (short int)**



Saída

507

# Conversão de tipo

- Conversão entre tipos:

```
#include<stdio.h>
```

```
int main() {
```

```
    int n1 = 50000;  
    short int n2 = n1;
```

```
    printf("%d\n", n2);
```

```
    return 0;
```

```
}
```

**Perigoso! Armazena valor (int) em variável com tipo de dados que possui faixa MENOR (short int)**



Saída

-15536

# Conversão de tipo

- Conversão entre tipos:

```
#include<stdio.h>
```

```
int main() {
```

```
    int n1 = 50000;  
    short int n2 = n1;
```

```
    printf("%hd\n", n2);
```

```
    return 0;
```

```
}
```

**Perigoso! Armazena valor (int) em variável com tipo de dados que possui faixa MENOR (short int)**



Saída

-15536

# Conversão de tipo

- Conversão entre tipos:

```
#include<stdio.h>
```

```
int main() {
```

```
    float f1 = 2.782;
```

```
    int n2 = f1;
```

```
    printf("%d\n", n2);
```

```
    return 0;
```

```
}
```

Descarta parte fracionária.



Saída

2

# Conversão de tipo

- Conversão entre tipos usando **cast**:

```
float n = (float) 507;  
int t = (int) n;
```



Força a conversão de tipo



# Conversão de tipo

- Conversão entre tipos usando **cast**:

```
#include <stdio.h>
int main() {
    float r1 = 7 / 2;
    float r2 = ((float) 7) / 2;
    float r3 = 7 / ((float) 2);
    float r4 = ((float) 7) / ((float) 2);
    float r5 = (float) 7 / 2;
    float r6 = (float) (7 / 2);

    printf("%.1f %.1f %.1f\n", r1, r2, r3);
    printf("%.1f %.1f %.1f\n", r4, r5, r6);

    return 0;
}
```

Saída

?

# Conversão de tipo

- Conversão entre tipos usando **cast**:

```
#include <stdio.h>
int main() {
    float r1 = 7 / 2;
    float r2 = ((float) 7) / 2;
    float r3 = 7 / ((float) 2);
    float r4 = ((float) 7) / ((float) 2);
    float r5 = (float) 7 / 2;
    float r6 = (float) (7 / 2);
    printf("%.1f %.1f %.1f\n", r1, r2, r3);
    printf("%.1f %.1f %.1f\n", r4, r5, r6);

    return 0;
}
```

Saída

3.0 3.5 3.5  
3.5 3.5 3.0

# Inicialização de variáveis

- **Sempre inicialize variáveis em C!**
- Quando uma variável local 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!**

# Inicialização de variáveis

- **Sempre inicie variáveis em C!**
- Quando uma variável local 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!**



## Resultado da compilação usando -Wall:

```
teste.c: In function 'main':  
teste.c:9:5: warning: 'num' is used uninitialized in this function  
[-Wuninitialized]  
    printf("%d\n", num);
```

```
#include<stdio.h>
```

```
int main() {
```

```
    int num;
```

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

```
    return 0;
```

```
}
```

# Operadores aritméticos

+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão



# Operadores relacionais

< Menor que

> Maior que

<= Menor ou igual a

>= Maior ou igual a

== Igual a

!= Diferente de

# Operadores lógicos

&&	E
	Ou
!	Negação

# Operadores de atribuição

++	Incremento unitário
--	Decremento unitário
+=	Atribuição por soma
-=	Atribuição por subtração
*=	Atribuição por multiplicação
/=	Atribuição por divisão
%=	Atribuição por resto da divisão

# Operadores de atribuição

**Não use esses operadores mais de uma vez na mesma linha!**

++	Incremento unitário
--	Decremento unitário
+=	Atribuição por soma
-=	Atribuição por subtração
*=	Atribuição por multiplicação
/=	Atribuição por divisão
%=	Atribuição por resto da divisão

# Estruturas condicionais

```
if (<condicao>) {  
  
}
```

```
if (<condicao>) {  
  
} else {  
  
}
```

```
<condicao> ? <retorno verdadeiro> : <retorno falso>
```

# Atenção!

= é diferente de ==

- Qual a saída deste programa?

```
#include<stdio.h>
```

```
int main() {  
    int n = 507;  
  
    if (n = 5)  
        printf("n eh 5.\n");  
    else  
        printf("n nao eh 5.\n");  
  
    return 0;  
}
```

# Atenção!


= é diferente de ==

- Qual a saída deste programa?

```
#include<stdio.h>
```

```
int main() {  
    int n = 507;  
    if (n = 5)  
        printf("n eh 5.\n");  
    else  
        printf("n nao eh 5.\n");  
  
    return 0;  
}
```

Deveria ser:  
if (n == 5)



# Estruturas de repetição

- while

```
while (<condicao>) {  
  
}
```

## Exemplo

```
int i = 0;  
while (i < 3) {  
    i++;  
    printf("%d\n", i);  
}
```

## Saída





# Estruturas de repetição

- while

```
while (<condicao>) {  
  
}
```

## Exemplo

```
int i = 0;  
while (i < 3) {  
    i++;  
    printf("%d\n", i);  
}
```

## Saída

```
1  
2  
3
```

# Estruturas de repetição

- while

```
while (<condicao>) {  
  
}
```

## Exemplo

```
int i = 0;  
while (i < 3) {  
    printf("%d\n", i++);  
}
```

## Saída



# Estruturas de repetição

- while

```
while (<condicao>) {  
  
}
```

## Exemplo

```
int i = 0;  
while (i < 3) {  
    printf("%d\n", i++);  
}
```

## Saída

```
0  
1  
2
```

# Estruturas de repetição

- while

```
while (<condicao>) {  
  
}
```

## Exemplo

```
int i = 0;  
while (i < 3) {  
    printf("%d\n", ++i);  
}
```

## Saída



# Estruturas de repetição

- while

```
while (<condicao>) {  
  
}
```

## Exemplo

```
int i = 0;  
while (i < 3) {  
    printf("%d\n", ++i);  
}
```

## Saída

```
1  
2  
3
```

# Estruturas de repetição

- do-while

```
do {  
  
} while (<condicao>);
```

## Exemplo

```
int i = 0;  
do {  
    printf("1\n");  
    i++;  
} while (i < 3);
```

## Saída



# Estruturas de repetição

- do-while

```
do {  
  
} while (<condicao>);
```

## Exemplo

```
int i = 0;  
do {  
    printf("1\n");  
    i++;  
} while (i < 3);
```

## Saída

```
1  
1  
1
```

# Estruturas de repetição

- for

```
for (<inicializacao>; <condicao>; <passo>) {  
  
}
```

## Exemplo

```
int i;  
for (I = 0; I < 3; I++) {  
    printf("%d\n", i);  
}
```

## Saída





# Estruturas de repetição

- for

```
for (<inicializacao>; <condicao>; <passo>) {  
  
}
```

## Exemplo

```
int i;  
for (I = 0; I < 3; I++) {  
    printf("%d\n", i);  
}
```

## Saída

Não  
compila!

# Estruturas de repetição

- for

```
for (<inicializacao>; <condicao>; <passo>) {  
  
}
```

## Exemplo

```
int i;  
for (i = 0; i < 3; i++) {  
    printf("%d\n", i);  
}
```

## Saída



# Estruturas de repetição

- for

```
for (<inicializacao>; <condicao>; <passo>) {  
  
}
```

## Exemplo

```
int i;  
for (i = 0; i < 3; i++) {  
    printf("%d\n", i);  
}
```

## Saída

```
0  
1  
2
```

# 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.

**Alguns exemplos**

# Revisão

- Padrões:
  - A. Programa mínimo
  - B. Leitura de  $n$  elementos ( $n$  pré-definido)
  - C. Leitura de  $n$  elementos ( $n$  desconhecido)
  - D. Acumulador (e.g. soma)
  - E. Valor referência (e.g. menor, maior) - métodos de inicialização

# Leitura de $n$ elementos ( $n$ pré-definido)

```
#include <stdio.h>

int main() {

    int qtd;
    scanf("%d", &qtd);

    int i, numero;
    for (int i = 0; i < qtd; i++) {

        scanf("%d", &numero);

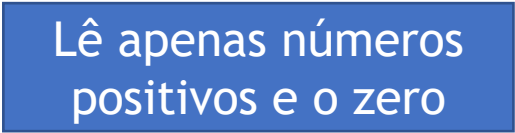
        // Faz alguma coisa
        printf("Leu numero!\n");
    }

    printf("Saiu!\n");

    return 0;
}
```

# Leitura de $n$ elementos ( $n$ desconhecido)

Lê apenas números  
positivos e o zero



```
#include <stdio.h>

int main() {

    int numero;
    scanf("%d", &numero);
    while (numero >= 0) {

        scanf("%d", &numero);

        // Faz alguma coisa
        printf("Leu numero!\n");
    }

    printf("Saiu!\n");

    return 0;
}
```



# Exercício 1

- **Faça um programa que leia um inteiro e imprima todos os divisores deste número inteiro (neste exercício, um divisor é um número que divide o outro sendo o resultado um número inteiro).**

# Exercício 2 - Lista de Números 1

- Faça um programa que leia um número  $n$ ;
- Depois o programa solicitará ao usuário a entrada de  $n$  números;
- Ao final, considerando os  $n$  números, o programa mostra:
  - Maior número;
  - Menor número;
  - Soma de todos os números.
- **Observação: não use vetor/array!**

# Exercício 3 - Lista de Números 2

- Modifique o programa anterior para não solicitar mais a quantidade *n* no início;
- Agora **o programa permanecerá lendo números até que o usuário digite um número negativo.**

# Exercício 4

- Faça um programa que leia um número  $n$ ;
- Depois o programa solicitará ao usuário a entrada de  $n$  números;
- Ao final, considerando o  $n$  números, o programa mostra **quantos números são ímpares e quantos são pares**.

# Exercício 5 - Menu operações

Faça um programa conforme descrito a seguir:

1. Leia um código de operação:
  - 1: soma dois números ( $a + b$ )
  - 2: soma três números ( $a + b + c$ )
  - 3: multiplicação de dois números ( $a * b$ )
  - 0: sair
2. Se o usuário digitar a operação sair, o programa **mostra o menor resultado obtido e encerra**;
3. Caso contrário, solicitará a entrada dos números ( $a$  e  $b$  ou  $a$ ,  $b$  e  $c$ ) e imprimirá o resultado da operação (soma ou multiplicação); Após isso, o programa volta ao item 1 (ler código de operação).

# Exercício 5 - Menu Operações

## Exemplo 1

```
Operacao? 1
300
207
507
Operacao? 0
507
```

## Exemplo 2

```
Operacao? 1
-2
1
-1
Operacao? 3
2
90
180
Operacao? 2
80
2
5
87
Operacao? 0
-1
```

# Referências

- Slides do Prof. Fabrício Olivetti:
  - <http://folivetti.github.io/courses/ProgramacaoEstruturada/>
- CELES, W.; CERQUEIRA, R.; RANGEL, J. L. Introdução a Estruturas de Dados. Elsevier/Campus, 2004.

# Bibliografia básica

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: teoria e prática. 2. ed. Rio de Janeiro, RJ: Campus, 2002.
- FORBELLONE, A. L. V.; EBERSPACHER, H. F. Lógica de programação: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo, SP: Prentice Hall, 2005.
- PINHEIRO, F. A. C. Elementos de programação em C. Porto Alegre, RS: Bookman, 2012.



# Bibliografia complementar

- AGUILAR, L. J. Programação em C++: algoritmos, estruturas de dados e objetos. São Paulo, SP: McGraw-Hill, 2008.
- DROZDEK, A. Estrutura de dados e algoritmos em C++. São Paulo, SP: Cengage Learning, 2009.
- KNUTH D. E. The art of computer programming. Upper Saddle River, USA: Addison- Wesley, 2005.
- SEDGEWICK, R. Algorithms in C++: parts 1-4: fundamentals, data structures, sorting, searching. Reading, USA: Addison-Wesley, 1998.
- SZWARCFITER, J. L.; MARKENZON, L. Estruturas de dados e seus algoritmos. 3. ed. Rio de Janeiro, RJ: LTC, 1994.
- TENENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. Estruturas de dados usando C. São Paulo, SP: Pearson Makron Books, 1995.