



**INSTITUTO FEDERAL**

Norte de Minas Gerais

Campus Januária

# Estruturas de Dados I

## - *Linguagem C* -



## Breve Histórico...

- Criada por **Dennis Ritchie** em 1972, nos laboratórios da *AT&T*, para implementação do sistema **UNIX**.
- O UNIX permitiu uma grande difusão da linguagem.
- Uma das linguagens mais ensinadas no mundo, e compatível praticamente com todas as arquiteturas computacionais existentes.
- Base para criação do C++ (suporte à orientação a objetos) e influência para dezenas de outras linguagens...
  - PHP, C#, Java, JavaScript, ObjectiveC, Go, Python, ...



# Breve Histórico...

- Mas a linguagem C é tão antiga... Porque estudar?





**INSTITUTO FEDERAL**  
Norte de Minas Gerais  
Campus Januária

# Ranking

**TIOBE INDEX**



About us ▾

Knowledge

News

Coding Standards

TIOBE Index

Contact













Products ▾

Quality Models ▾

Markets ▾

Schedule a demo

Aug 2023	Aug 2022	Change	Programming Language		Ratings	Change
1	1			Python	13.33%	-2.30%
2	2			C	11.41%	-3.35%
3	4	▲		C++	10.63%	+0.49%
4	3	▼		Java	10.33%	-2.14%
5	5			C#	7.04%	+1.64%
6	8	▲		JavaScript	3.29%	+0.89%
7	6	▼		Visual Basic	2.63%	-2.26%
8	9	▲		SQL	1.53%	-0.14%
9	7	▼		Assembly language	1.34%	-1.41%
10	10			PHP	1.27%	-0.09%





**INSTITUTO FEDERAL**  
Norte de Minas Gerais  
Campus Januária

# Ranking

**TIOBE INDEX**



[About us](#) ▾
 [Knowledge](#)
[News](#)
[Coding Standards](#)
[TIOBE Index](#)
[Contact](#)

[Products](#) ▾
 [Quality Models](#) ▾
 [Markets](#) ▾
 [Schedule a demo](#)

Aug 2023	Aug 2022	Change	Programming Language	Ratings	Change
1	Home » <a href="#">TIOBE Index</a> <h1>The C Programming Language</h1> <p>Some information about C:</p> <ul style="list-style-type: none"> <li> <b>Highest Position (since 2001): #1 in Sep 2021</b> </li> <li> <b>Lowest Position (since 2001): #2 in Aug 2023</b> </li> <li> <b>Language of the Year: 2008, 2017, 2019</b> </li> </ul>				
2					
3					
4					
5					
6					
7					
8					
9					
10	10		PHP	1.27%	-0.09%



# Ranking

IEEE INDEX

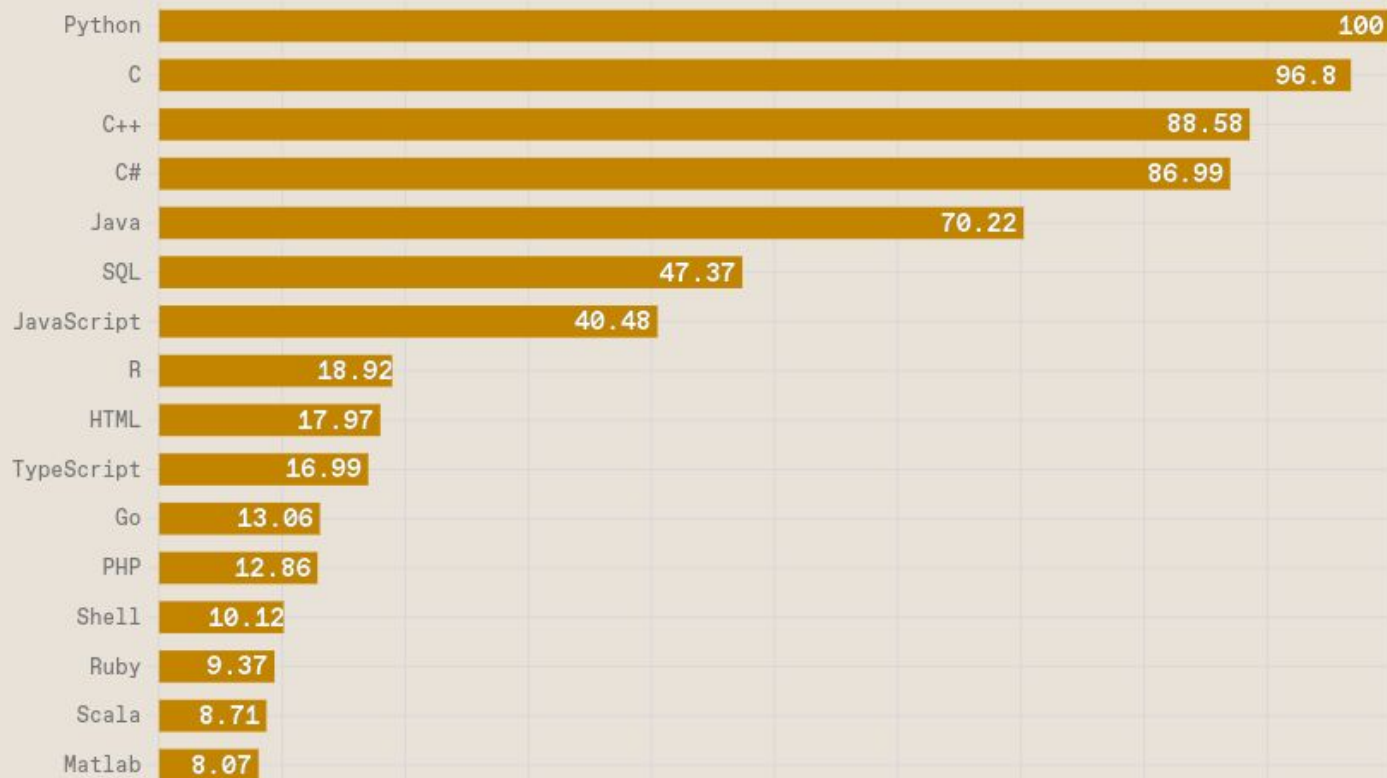
## Top Programming Languages 2022

Click a button to see a differently weighted ranking

Spectrum

Jobs

Trending





# Leia...

## Como a linguagem C se relaciona com outras tecnologias, como C++, Java e Python?

A linguagem C tem uma relação estreita com outras tecnologias, como C++, Java e Python. Aqui estão algumas maneiras como essas tecnologias se relacionam com C:

1. C++: C++ é uma extensão da linguagem C, adicionando recursos como programação orientada a objetos e coleta de lixo. O código C++ pode chamar funções escritas em C, e muitos projetos grandes incluem tanto código C quanto C++. Muitos dos conceitos aprendidos ao programar em C também se aplicam ao C++.
2. Java: A linguagem Java é frequentemente comparada a C em termos de velocidade e desempenho. O compilador Java é escrito em C e o código Java pode ser compilado em bytecode para ser executado em uma máquina virtual Java. Além disso, a API padrão do Java inclui várias funções C-like para trabalhar com entrada e saída de dados.
3. Python: Embora Python seja uma linguagem de programação de alto nível, os módulos Python podem ser escritos em C para melhorar o desempenho. O interpretador Python é escrito em C e muitas bibliotecas populares para processamento de dados, como NumPy e pandas, são escritas em C para obter melhor desempenho.

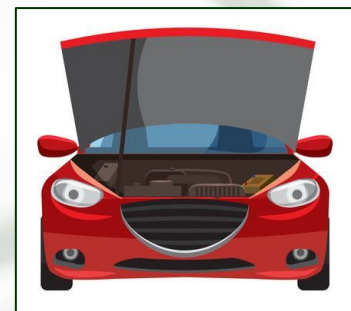




# Vantagens em Estudar C

- Vantagens em estudar a Linguagem C...
  - **Aprendizado teórico mais sólido sobre programação e arquitetura computacional.**
  - **Compreender na prática** aspectos como: a manipulação e gerência de memória, funcionamento de *buffers stdin* e *stdout*, operações em disco e arquivos, processos, redirecionamentos...

Saber dirigir (programar) é uma coisa...  
Mas conhecer e entender tudo que acontece por “debaixo do capô”  
te torna um profissional muito mais completo.







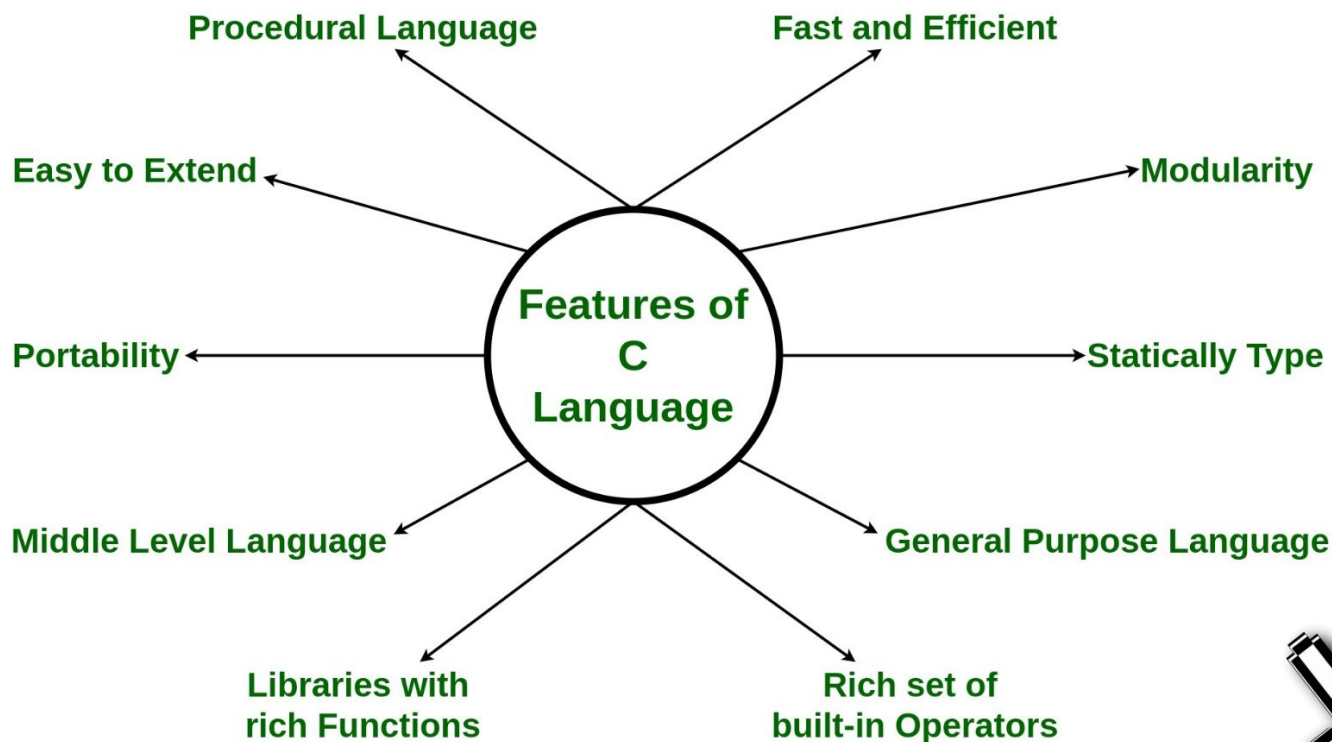
# Vantagens em Estudar C

- Vantagens em estudar a Linguagem C...
  - C ainda é muito utilizado para escrever *softwares* do mundo real em que **eficiência**, **flexibilidade** e **velocidade** são requisitos indispensáveis.
  - Sistemas embarcados e microcontroladores
    - *P.Ex.: Arduino, Controles Aéreos, Real-Time Apps*
  - Computação de alto desempenho
    - *P.Ex.: Softwares para grids, clusters e IaaS.*



# Principais Características

## Features of C Programming Language

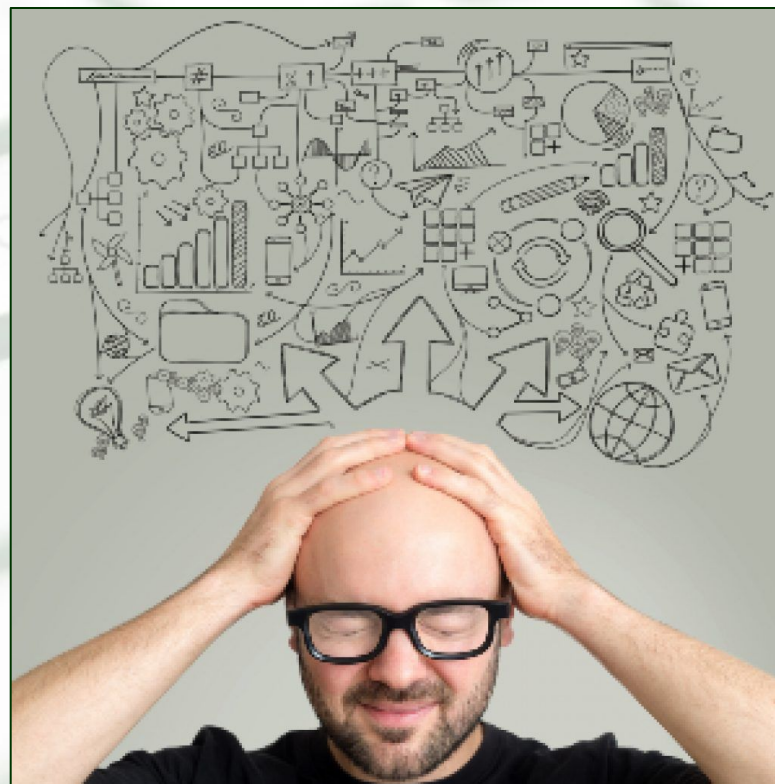


**VOCÊ  
ESTUDA  
E  
EXPLICA!**





***O que fazer para resolver problemas  
muito complexos?***





# Filosofia do C

*O que fazer para resolver problemas muito complexos?*

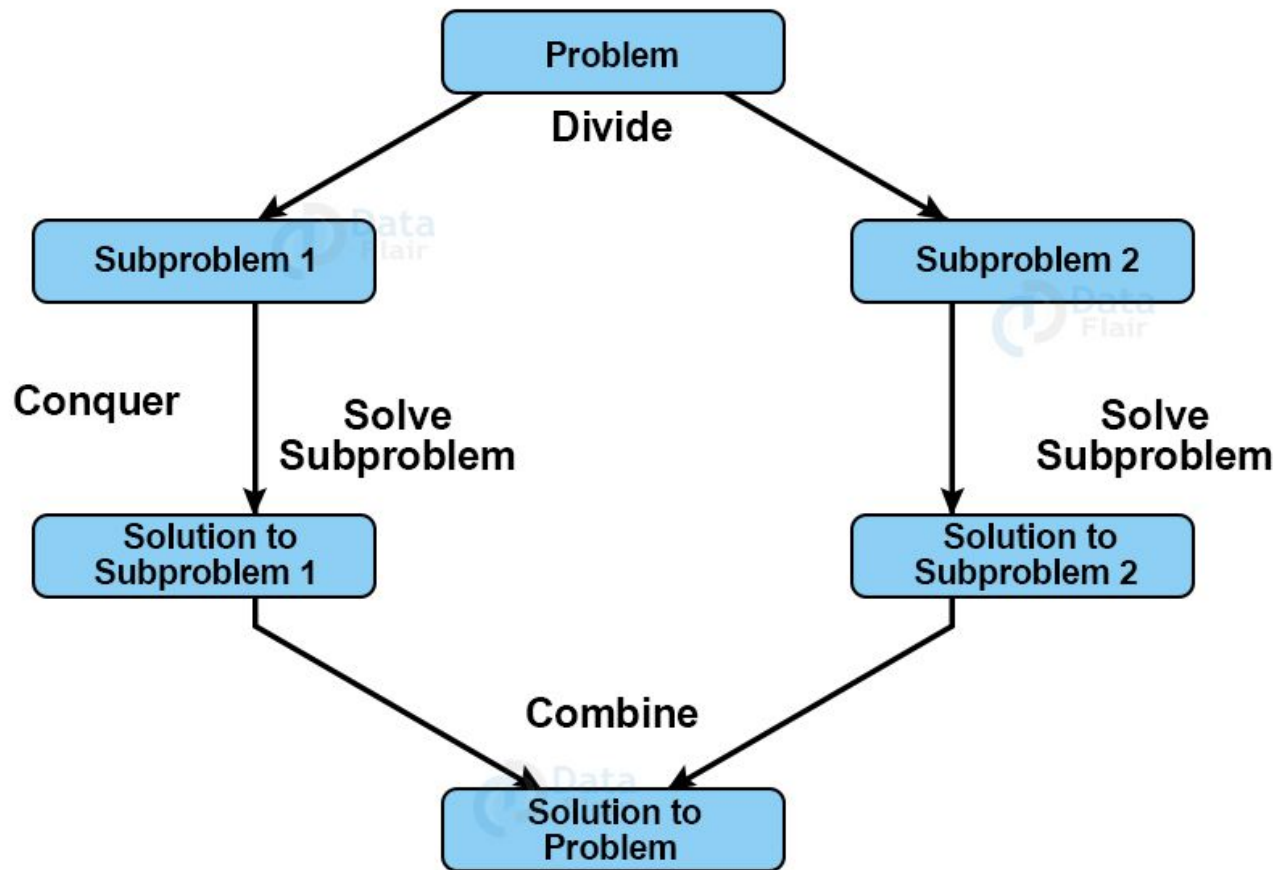
**Dividir para Conquistar**

**É mais fácil implementar pequenos pedaços de código que realizem corretamente uma única função do que fazer extensos códigos, com muitas variáveis, condições e exceções de forma a atingir o mesmo resultado.**





# Filosofia do C



Divide and Conquer Approach



É mais fácil  
que fazer  
e

digito  
que  
ções  
o.



# Filosofia do C





## Primeiro programa em C...

```
#include <stdio.h>

int main(){
    //meu primeiro programa em c
    printf("hello world!");
}
```



## Primeiro programa em C...

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

```
int main(){
```

```
//meu primeiro programa em c
```

```
printf("hello world!");
```

```
}
```

*Todo programa em C consiste  
em uma ou mais funções*





## Primeiro programa em C...

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

```
int main(){
```

*Mas todo programa sempre inicia a execução a partir da função main()*

```
//meu primeiro programa em c
```

```
printf("hello world!");
```

```
}
```



## Primeiro programa em C...

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

```
int main(){
```

*Os caracteres { } determinam o início e fim de blocos de execução*

```
//meu primeiro programa em c
```

```
printf("hello world!");
```

```
}
```



## Primeiro programa em C...

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

```
int main(){
```

*Toda instrução deve terminar em ;*

```
//meu primeiro programa em c
```

```
printf("hello world!");
```

```
}
```



## Primeiro programa em C...

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

```
int main(){
```

*// inicia uma linha de comentário*

```
//meu primeiro programa em c
```

```
printf("hello world!");
```

```
}
```





## Primeiro programa em C...

```
/* #include <stdio.h>  
  
int main(){  
    //meu primeiro programa em c  
    printf("hello world!");  
}*/
```

*/\* para comentários multi-linhas \*/*



## Primeiro programa em C...

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

```
int main(){
```

```
//meu primeiro programa em c
```

```
printf("hello world!");
```

```
}
```

*#include permite a importação de bibliotecas de funções prontas*



# Uso de Bibliotecas

- A linguagem C permite a importação de diversas **bibliotecas**, possibilitando o uso de uma infinidade de funções pré-programadas.
- Por exemplo, as operações de **leitura em teclado** e **impressão em monitor** são feitas através de funções prontas da biblioteca **stdio**.

STanDart In/Out

```
#include <nome_da_biblioteca.h>
```



# Estrutura Básica

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

```
int main(){
```

```
//meu primeiro programa em c
```

```
printf("hello world!");
```

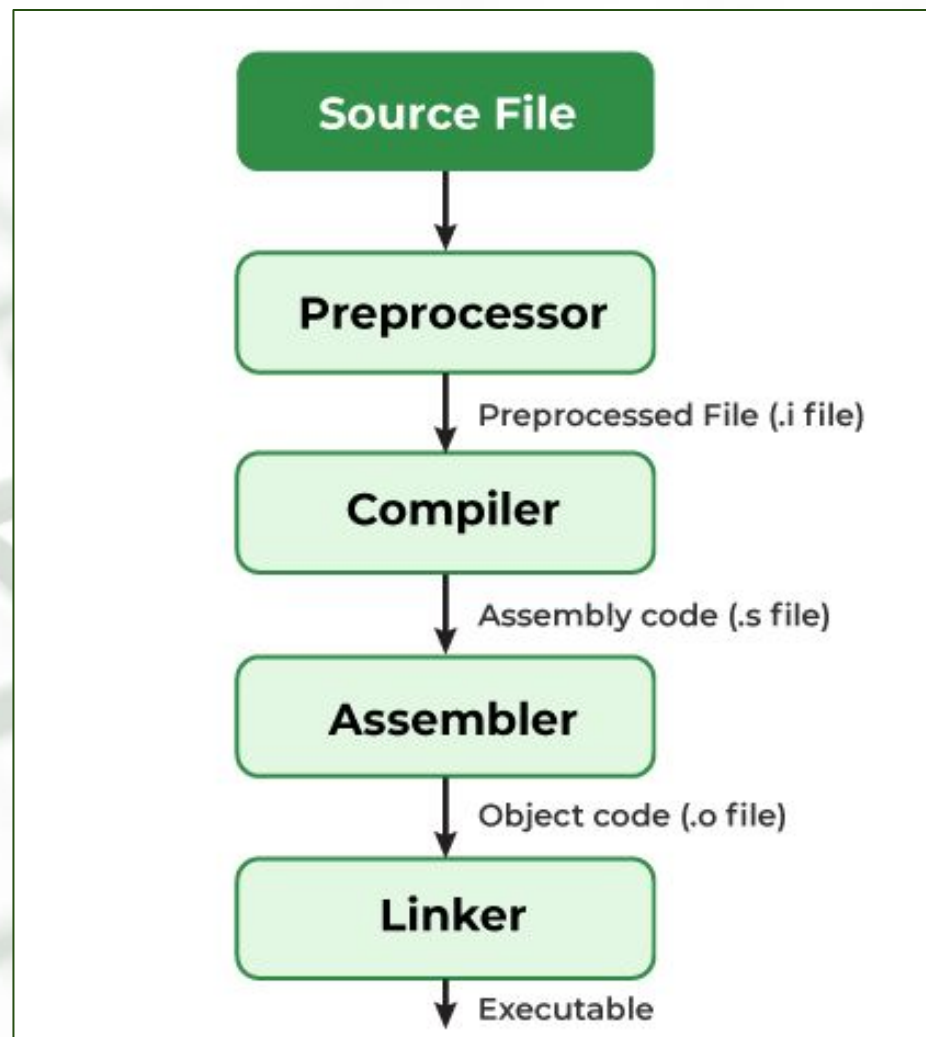
```
}
```





# Etapas do Desenvolvimento

- Editor
  - Código-Fonte
- Pré-Processador
  - Código Expandido
- Linter
  - Verificação de Erros
- Compilador
  - Código Objeto
- Linker
  - Código Executável





# Estrutura Básica

## Algumas bibliotecas que iremos utilizar durante o curso

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
```

## DICA ÚTIL

Ajuste o template padrão utilizado pelo Geany em:

```
sudo nano /usr/share/geany/templates/files/main.c
```



# Estrutura Básica

```

sudo nano /usr/share/geany/templates/files/main.c

Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda

/usr/share/geany/templates/files/main.c *

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

int main(){

}

^G Ajuda      ^O Gravar     ^W Onde está ^K Recortar   ^T Executar
^X Sair       ^R Ler o arq ^\ Substitui  ^U Colar    ^J Justificar
  
```



# Variáveis

**Variável é a representação de um espaço na memória principal (RAM) capaz de reter/armazenar algum dado.**

*Existem 05 informações fundamentais associadas a toda variável, são elas:*

**TIPO DE DADO ARMAZENADO**

**NOME DA VARIÁVEL**

**VALOR DA VARIÁVEL**

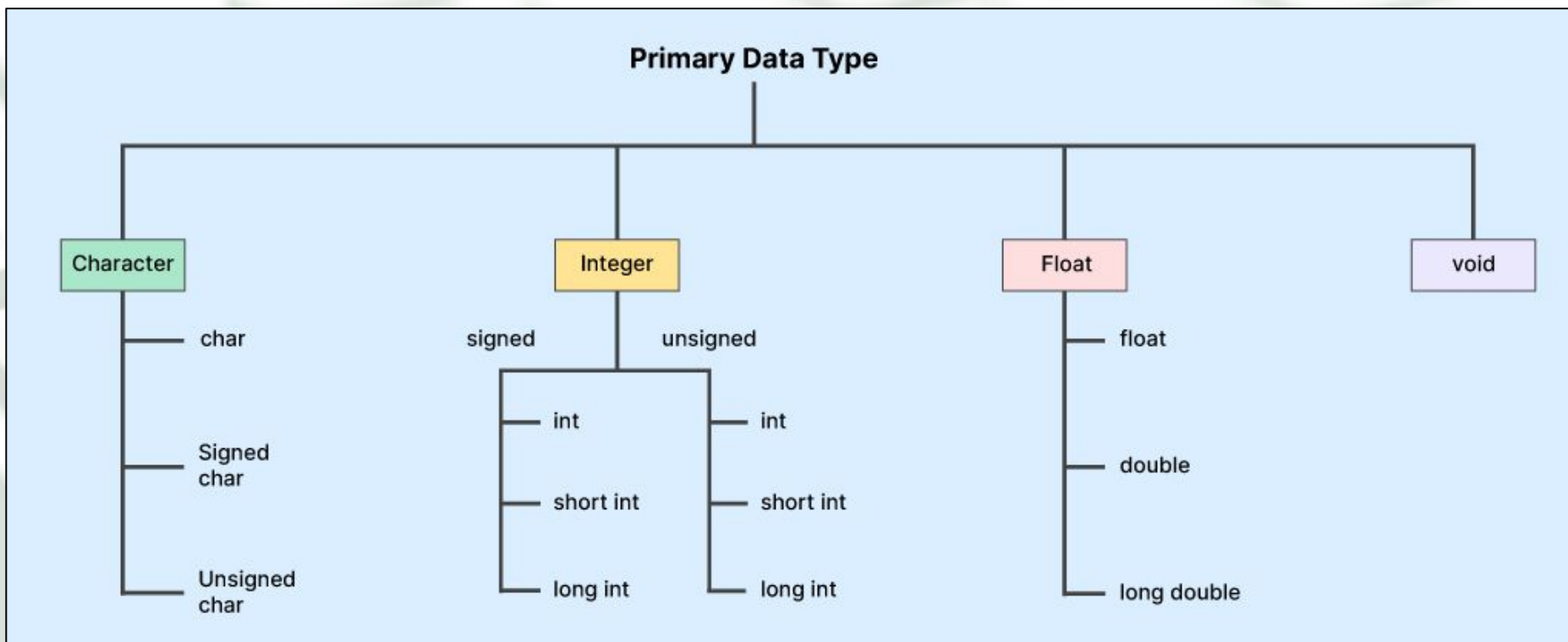
**ENDEREÇO DA VARIÁVEL**

**ESCOPO DA VARIÁVEL**



# Tipos Básicos da Linguagem

- O **Tipo da variável** irá determinar o tamanho do espaço em memória reservado para armazená-la.







# Tipos Básicos da Linguagem

INFORMAÇÃO	TIPO	ESPAÇO RAM*	FLAG	PRECISÃO	
Valor Inteiro	int	4 Bytes	%d ou %i	-2.147.483.648	2.147.483.647
	unsigned int	4 Bytes	%u	0	4.294.967.295
	short int	2 Bytes	%hd	-32.768	32.767
	unsigned short int	2 Bytes	%hu	0	65.535
	long int	8 Bytes	%ld	-2 <sup>63</sup>	2 <sup>63</sup> -1
	unsigned long int	8 Bytes	%lh	0	2 <sup>64</sup> -1
Valor Fracionado	float	4 Bytes	%f	~ 7 Dígitos de precisão	
	double	8 Bytes	%lf	~ 12 Dígitos de precisão	
	long double	16 Bytes	%Lf	~ 16 Dígitos de precisão	
Símbolo ou Caractere	char	1 Byte	%c	-128	127
	unsigned char	1 Byte	%d	0	255

\*CONSIDERANDO ARQUITETURA CPU-64 BITS (x64)



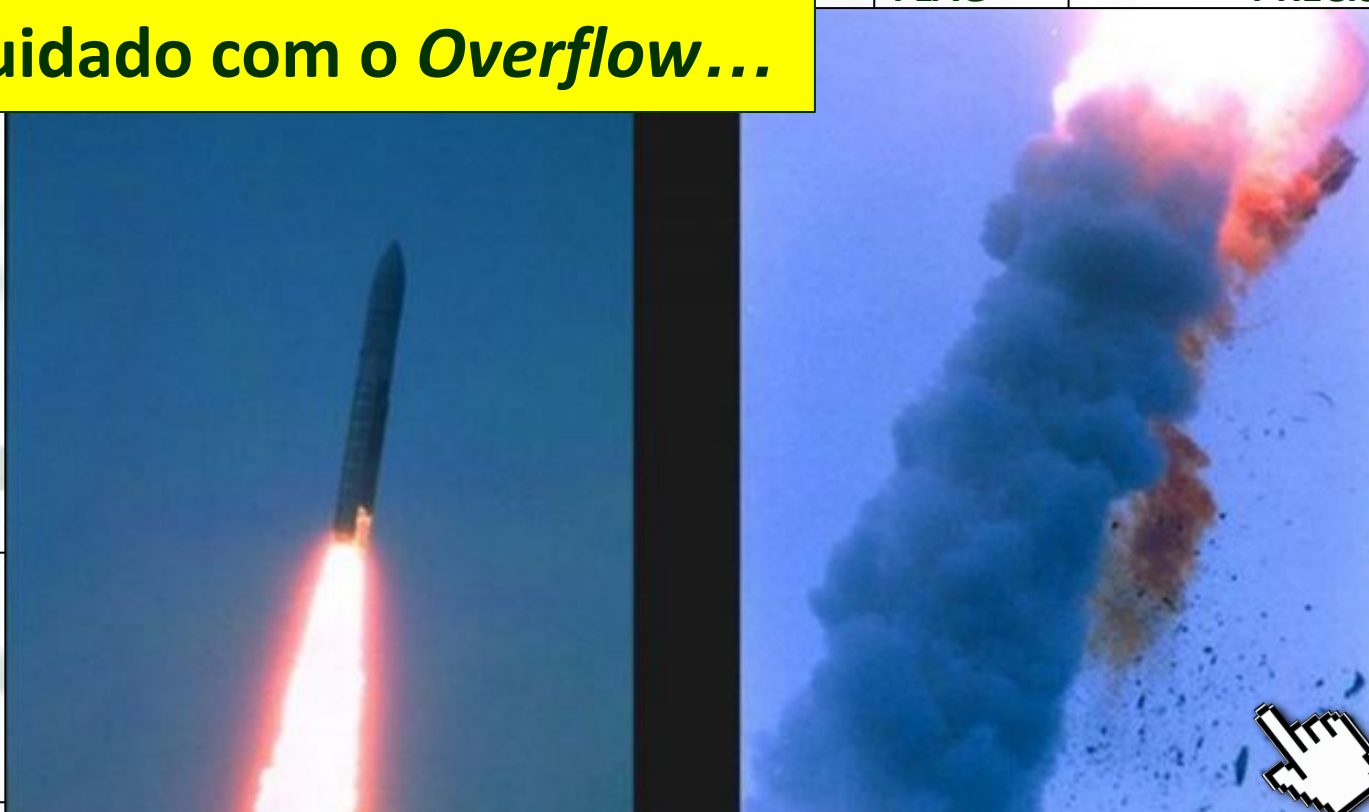
# Tipos Básicos da Linguagem

**Cuidado com o *Overflow*...**

Valor Inteiro

Valor Fracionado

Símbolo ou Caractere



**ARIANE-5: THE 7 BILLION DOLLAR OVERFLOW**

unsigned char

1 Byte

%d

0

127

255

2.147.483.647

1.294.967.295

32.767

65.535

$2^{63}-1$

$2^{64}-1$

precisão

de precisão

de precisão

\*CONSIDERANDO ARQUITETURA CPU-64 BITS (x64)



# Declaração de Variáveis

- Declaramos uma variável atribuindo o **TIPO DE DADO** e o **NOME** ao qual será identificada.

```
tipo nomeDaVariavel;
```

```
#include <stdio.h>
int main(){
    float altura;
    int idade, soma;
    int qtdePessoas;
}
```



# Declaração de Variáveis

```
#include <stdio.h>
int main(){
    float altura;
    int idade, soma;
    int qtdePessoas;
}
```

Pode-se declarar duas ou mais variáveis de um mesmo tipo em uma única instrução



# Declaração de Variáveis

```
#include <stdio.h>
int main(){
    float altura;
    int idade, soma=0;
    int qtdePessoas=0;
}
```

Também é possível  
inicializar a variável com  
um valor determinado





# Declaração de Variáveis

```
#include <stdio.h>
int main(){
    float altura;
    int idade, soma=0;
    int qtdePessoas=0;
}
```

## BOA PRÁTICA!

O nome de uma variável deve ser escolhido de modo a **facilitar** a compreensão da sua utilidade/necessidade



# Declaração de Variáveis

```
#include <stdio.h>
int main(){
    float altura;
    int idade, soma=0;
    int qtdePessoas=0;
    float mediaAlturas;
    int opcaoUsuario;
}
```

## BOA PRÁTICA!

Quando nomes compostos, a primeira inicial em minúsculo e as seguintes em maiúsculo.  
(**camelCase**)



# Declaração de Variáveis

```
#include <stdio.h>
int main(){
    float altura;
    int idade, soma=0;
    int qtde_pessoas=0;
    float media_alturas;
    int opcao_usuario;
}
```

**BOA PRÁTICA!**  
...ou separar as palavras  
por símbolo underline.  
(snake\_case)



# Endereço de uma Variável

- O nome de uma variável **pouco importa para o programa em si.**
- Na verdade, **o nome é útil apenas para nós: Programadores**, termos mais facilidade na hora de desenvolver nossos códigos.

**Afinal... É muito mais fácil lembrar  
"qtde\_pessoas" do que 0x7ffd3c0cc624**

*Endereço de memória onde está  
localizado "qtde\_pessoas"*



# Alocação de Memória

## Compreendendo Alocação de Memória

Todo espaço disponível na memória RAM do seu dispositivo é mapeado em BYTES (conjunto de 8 bits).

**Cada Byte possui um endereço específico.**

1026 <= Endereço do Byte

<= Conteúdo/Valor do Byte

MEMÓRIA RAM							
1000	1001	1002	1003	1004	1005	1006	1007
1008	1009	1010	1011	1012	1013	1014	1015
1016	1017	1018	1019	1020	1021	1022	1023
1024	1025	1026	1027	1028	1029	1030	1031
1032	1033	1034	1035	1036	1037	1038	1039
1040	1041	1042	1043	1044	1045	1046	1047





# Alocação de Memória

## Compreendendo Alocação de Memória

A memória RAM é compartilhada por todos os processos (programas) em execução no dispositivo, e muitos destes “espaços” já podem estar ocupados.

1026 <= Endereço do Byte

<= Conteúdo/Valor do Byte

MEMÓRIA RAM							
1000	1001	1002	1003	1004	1005	1006	1007
1008	1009	1010	1011	1012	1013	1014	1015
1016	1017	1018	1019	1020	1021	1022	1023
1024	1025	1026	1027	1028	1029	1030	1031
1032	1033	1034	1035	1036	1037	1038	1039
1040	1041	1042	1043	1044	1045	1046	1047



# Alocação de Memória

## Compreendendo Alocação de Memória

Quando você declara:  
**int** qtde\_pessoas;  
**char** letra;

Seu compilador sabe  
(pelos tipos) que:

**int** ocupa **4 Bytes**  
**char** ocupa **1 Byte**

Procura-se então os  
espaços livres para  
armazenar as variáveis...

MEMÓRIA RAM							
1000	1001	1002	1003	1004	1005	1006	1007
1008	1009	1010	1011	1012	1013	1014	1015
1016	1017	1018	1019	1020	1021	1022	1023
1024	1025	1026	1027	1028	1029	1030	1031
1032	1033	1034	1035	1036	1037	1038	1039
1040	1041	1042	1043	1044	1045	1046	1047



# Alocação de Memória

## Compreendendo Alocação de Memória

Nessa ilustração didática  
por exemplo...

**qtde\_pessoas** iniciará no  
endereço de memória **1024**.

**letra** iniciará no endereço  
de memória **1004**.

RAM é acrônimo para  
*Random Access Memory*

MEMÓRIA RAM							
1000	1001	1002	1003	1004	1005	1006	1007
1008	1009	1010	1011	1012	1013	1014	1015
1016	1017	1018	1019	1020	1021	1022	1023
1024	1025	1026	1027	1028	1029	1030	1031
1032	1033	1034	1035	1036	1037	1038	1039
1040	1041	1042	1043	1044	1045	1046	1047



# Alocação de Memória

## Compreendendo Alocação de Memória

Nessa ilustração didática por exemplo...

`qtde_pessoas` iniciará no endereço de memória **1024**.

`letra` iniciará no endereço de memória **1004**.

MEMÓRIA RAM			
1004	1005	1006	1007
11001101	01001101	10110111	11010101
////////////////////////////////////			
1024	1025	1026	1027
00001001	11101011	01101110	00011110

Acontece que, nestes espaços de memória, pode haver resíduos de conteúdos de outros processos encerrados anteriormente...

Chamamos isso na programação de:

**Lixo de Memória**

**Dependendo da operação que fizermos, se não tratarmos, podemos ter um erro de execução!**



# Alocação de Memória

## Compreendendo Alocação de Memória

Nessa ilustração didática por exemplo...

`qtde_pessoas` iniciará no endereço de memória **1024**.

`letra` iniciará no endereço de memória **1004**.

MEMÓRIA RAM			
1004	1005	1006	1007
01011111	01001001	00110111	00010101
////////////////////////////////////			
1024	1025	1026	1027
00000000	00000000	00000000	00001010

Tratamos o problema garantindo que o valor do conteúdo corresponde a um valor válido, seja:

*(i) inicializando o valor na criação da variável...*

```
int qtde_pessoas = 10;
char letra = 'a';
```

*(ii) lendo o valor da variável diretamente do teclado (veremos logo a seguir...)*





# Alocação de Memória

Veja que interessante...

```
#include<stdio.h>

int main(){
    int a = 1;
    int b;
    float c = 3.14;
    char d = 'x';
    printf("a => endereço: %p valor: %i\n",&a,a);
    printf("b => endereço: %p valor: %i\n",&b,b);
    printf("c => endereço: %p valor: %f\n",&c,c);
    printf("d => endereço: %p valor: %c\n",&d,d);
}
```

```
Terminal
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
a => endereço: 0x7ffe033e9530 valor: 1
b => endereço: 0x7ffe033e9534 valor: 57129
c => endereço: 0x7ffe033e9538 valor: 3.140000
d => endereço: 0x7ffe033e952f valor: X
```



# Alocação de Memória

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

```
int main(){
```

```
    int a = 1;
```

```
    int b;
```

```
    float c = 3.14;
```

```
    char d = 'x';
```

```
    printf("a => endereço: %p valor: %i\n",&a,a);
```

```
    printf("b => endereço: %p valor: %i\n",&b,b);
```

```
    printf("c => endereço: %p valor: %f\n",&c,c);
```

```
    printf("d => endereço: %p valor: %c\n",&d,d);
```

```
}
```

Veja que interessante...

*& antes do nome da variável recupera o endereço de memória da variável*

```
Terminal
```

Arquivo Editar Ver Pesquisar Terminal Ajuda

```
a => endereço: 0x7ffe033e9530 valor: 1
b => endereço: 0x7ffe033e9534 valor: 57129
c => endereço: 0x7ffe033e9538 valor: 3.140000
d => endereço: 0x7ffe033e952f valor: X
```



# Saída Formatada: `printf()`

```
int printf(const char* str_contr [, lista_args]);
```

- Biblioteca `<stdio.h>`
- A função `printf()` é o comando padrão em C para impressão no dispositivo de saída *default* (monitor);
- A string de controle (*str\_contr*) é a máscara que especifica o que, e como, serão impressos os dados e as informações da *lista\_args*.



# Exemplos

- `printf("Olá Mundo");`
- `printf("Linha 1\nLinha 2");`
- `printf("Coluna1\tColuna 2");`
- `printf("Tenho %d Anos de Vida", idade);`
- `printf("Média de Idades: %f", media);`
- `printf("Total da Conta: %.2f", total);`
- `printf("Letra Inicial: %c", letra);`





# Caracteres de Escape

<code>\n</code>	Nova Linha (ENTER)
<code>\t</code>	Tabulação Horizontal (TAB)
<code>\r</code>	Retorno de Linha
<code>\b</code>	Backspace
<code>\'</code>	Apóstrofo
<code>\"</code>	Aspas
<code>\\</code>	Barra Invertida
<code>\0</code>	Caracter Nulo





# Flags de Formato

## ■ Principais flags para impressão de variáveis

<b>%c</b>	Caractere Simples
<b>%d</b> <b>%i</b>	Inteiro na Base Decimal
<b>%f</b>	Ponto Flutuante / Valor Fracionado
<b>%e</b>	Notação Científica
<b>%u</b>	Inteiro sem Sinal (Unsigned)
<b>%p</b>	Endereço de Memória (Ponteiro)



# Modificadores de Flags

## ■ Modificadores de flags para impressão.

<code>%5d</code>	Usar 5 espaços para imprimir o conteúdo. (Alinhamento à Direita é default)
<code>%-5d</code>	Usar 5 espaços, mas alinhado à Esquerda.
<code>%05d</code>	Usar 5 espaços, completando à esquerda com '0'.
<code>%.2f</code>	Arredondar o float para 2 casas decimais.



# Recomendação

*Salvo em casos excepcionais,  
é altamente recomendável finalizar  
toda impressão com `\n`  
Ao final explico porquê...*

- `printf("Olá Mundo\n");`
- `printf("Linha 1\nLinha 2\n");`
- `printf("Coluna1\tColuna 2\n");`
- `printf("Tenho %d Anos de Vida\n", idade);`
- `printf("Média de Idades: %.2f\n", media);`
- `printf("Total da Conta: %f\n", total);`
- `printf("Letra Inicial: %c\n", letra);`



# Operadores Aritméticos

+	Soma
-	Subtração
/	Divisão
*	Multiplicação
%	Resto de Divisão - Módulo
+=	Atribuição Aritmética $x += 1 \Rightarrow x = x + 1$
-=	Atribuição Aritmética $x -= 2 \Rightarrow x = x - 2$
++	Incremento $i++$ ou $++i \Rightarrow i = i + 1$
--	Decremento $i--$ ou $--i \Rightarrow i = i - 1$



- A partir dos valores abaixo, programe um único `printf()`, que gere a saída ao lado:

```
int main(){  
    int n = 50;  
    float pi = 3.14159;  
    int k = 2;  
}
```

```
3.14  
6.3  
157.1  
0002  
025  
100{20 espaços}50
```





# Indentação é Obrigatória!

## ATENÇÃO

A **Indentação**, além de boa prática, facilita a leitura, organização e correção dos algoritmos!

Você só tem a ganhar fazendo-a corretamente.

*Any fool can write code that a computer can understand.*

*Good programmers write code that HUMANS CAN UNDERSTAND.*

*- Martin Fowler,*

*Refactoring: Improving the Design of Existing Code*



# Exemplo

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

typedef struct{
int i;
num* prox; }num;

num* setNum(){
num* n = (num*)malloc(sizeof(num));
printf("Digite Num: ");
scanf(" %d",&n->i);
n->prox = NULL;
return n; }

num* getUltimoNum(num* list){
if(list->prox)
return getUltimoNum(list->prox);
else
return list; }
```

```
void getNums(num* list){
if(list){
printf("\n%d",list->i);
getNums(list->prox); } }

int main(){
num* fila = NULL;
int opt;
do{
if(!fila)
fila = setNum();
else{
num* fim = getUltimoNum(fila);
fim->prox = setNum(); }
printf("Continua?");
scanf(" %d",&opt);
}while(opt);
getNums(fila);
getch();
}
```



# Exemplo

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

typedef struct{
    int i;
    num* prox;
}num;

num* setNum(){
    num* n = (num*)malloc(sizeof(num));
    printf("Digite Num: ");
    scanf(" %d",&n->i);
    n->prox = NULL;
    return n;
}

num* getUltimoNum(num* list){
    if(list->prox)
        return getUltimoNum(list->prox);
    else
        return list;
}
```

```
void getNums(num* list){
    if(list){
        printf("\n%d",list->i);
        getNums(list->prox);
    }
}

int main(){
    num* fila = NULL;
    int opt;
    do{
        if(!fila)
            fila = setNum();
        else{
            num* fim = getUltimoNum(fila);
            fim->prox = setNum();
        }
        printf("Continua?");
        scanf(" %d",&opt);
    }while(opt);
    getNums(fila);
    getch();
}
```



# Exemplo

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

typedef struct{
```

```
void getNums(num* list){
    if(list){
        printf("\n%d",list->i);
        getNums(list->prox);
    }
```

**NÃO ESCREVEMOS CÓDIGO PARA O COMPUTADOR!**

```
num* setNum(){
```

```
num* fila = NULL;
```

**ESCREVEMOS CÓDIGO PARA OUTRA PESSOA**

```
    return n;
}

num* getUltimoNum(num* list){
    if(list->prox)
        return getUltimoNum(list->prox);
    else
        return list;
}
```

```
    else{
        num* fim = getUltimoNum(fila);
        fim->prox = setNum();
    }
    printf("Continua?");
    scanf(" %d",&opt);
}while(opt);
getNums(fila);
getch();
}
```



# Exemplo

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

typedef struct{
```

```
void getNums(num* list){
    if(list){
        printf("\n%d",list->i);
        getNums(list->prox);
    }
```

**NÃO ESCREVEMOS CÓDIGO PARA O COMPUTADOR!**

```
num* setNum(){
```

```
num* fila = NULL;
```

**ESCREVEMOS CÓDIGO PARA OUTRA PESSOA**

```
return n;
```

```
else{
```

**INCLUSIVE O “VOCÊ” DO FUTURO...**

```
return getUltimoNum(list->prox);
else
return list;
}
```

```
scanf(" %d",&opt);
}while(opt);
getNums(fila);
getch();
}
```





# Precedência de Operadores

```
int a, b, c, i;  
i = 3;      // a == ?    b == ?    c == ?    i == 3  
a = i++;    //...?  
b = ++i;  
c = --i;
```

Os operadores incrementais serão bastante utilizados nos **laços de repetição**.



# Precedência de Operadores

```
int a, b, c, i;  
i = 3;      // a == ?    b == ?    c == ?    i == 3  
a = i++;    // a == 3    b == ?    c == ?    i == 4  
b = ++i;    //...?  
c = --i;
```

Os operadores incrementais serão bastante utilizados nos **laços de repetição**.



# Precedência de Operadores

```
int a, b, c, i;  
i = 3;      // a == ?    b == ?    c == ?    i == 3  
a = i++;    // a == 3    b == ?    c == ?    i == 4  
b = ++i;    // a == 3    b == 5    c == ?    i == 5  
c = --i;    //...?
```

Os operadores incrementais serão bastante utilizados nos **laços de repetição**.



# Precedência de Operadores

```
int a, b, c, i;  
i = 3;      // a == ?    b == ?    c == ?    i == 3  
a = i++;    // a == 3    b == ?    c == ?    i == 4  
b = ++i;    // a == 3    b == 5    c == ?    i == 5  
c = --i;    // a == 3    b == 5    c == 4    i == 4
```

Os operadores incrementais serão bastante utilizados nos **laços de repetição**.



# Typecast

- Qual será a saída gerada pelo código abaixo?

```
int main() {  
    int x,y;  
    float f;  
    x = 3;  
    y = 2;  
    f = x/y;  
    printf("%.2f", f);  
    return 0;  
}
```





# Typecast

## ! Atenção !

- Operações aritméticas realizadas entre variáveis de um mesmo tipo resultam em valor do mesmo tipo.
- *Exemplo:*

`int/int == int`

***Mesmo que o resultado não seja, necessariamente, um inteiro***



# Typecast

- Para resolver essas situações, utiliza-se a conversão explícita de tipos, chamado **TYPECAST**
- *Exemplo:*

**(float)**int/int == float

A instrução **(float)** antes da operação aritmética indica que se espera (e converte) o resultado para um tipo **float**.



# Typecast

## ■ Tente novamente...

```
int main() {  
    int x,y;  
    float f;  
    x = 3;  
    y = 2;  
    f = (float)x/y;  
    printf("%.2f", f);  
    return 0;  
}
```



# Leitura Formatada: `scanf()`

```
int scanf(const char* str_contr [, end_var, ...]);
```

- Biblioteca `<stdio.h>`
- A função **`scanf()`** é a instrução padrão para leitura do `stdin` - (*standart in*) dispositivo de entrada *default* (teclado).
- A lista de argumentos (*end\_var*) consiste nos **endereços das variáveis**, que é obtido através do **operador `&` seguido do nome da variável**.



# Exemplos

- `scanf("%d", &num);` // lê num como **int**
- `scanf("%f", &num);` // lê num como **float**
- `scanf("%c", &num);` // lê num como **char**
  
- `scanf("%d %d", &a, &b);` // lê **dois inteiros**  
// em sequência
  
- `scanf("%d %f", &a, &b);` /\* lê um **int** e um **float**  
em sequência \*/
  
- `scanf("%c %c %c", &a, &b, &c);` /\* lê **três char**  
em sequência \*/





# Atenção!

- Analise o código abaixo:

```
int main(){  
    char a;  
    scanf("%c",&a);  
    printf("%d\n",a);  
}
```

*O que acontecerá?*



# Regras de Sintaxe

- Em C, **tudo** é armazenado “como número”, inclusive letras e caracteres especiais.

```
int main(){  
    char a,b,c;  
    a = 15;  
    b = 'a';  
    c = a+b;  
    printf("%d %c\n",c,c);  
}
```



# Regras de Sintaxe

- Em C, **tudo** é armazenado “como número”, inclusive letras e caracteres especiais.

```
int main(){  
    char a,b,c;  
    a = 15;  
    b = 'a';  
    c = a+b;  
    printf("%d %c\n",c,c);  
}
```

```
Terminal  
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda  
112 p  
-----  
(program exited with code: 0)  
Press return to continue
```



- A partir do valor da variável `c` abaixo, programe um único `printf()`, que gere a saída ao lado:

```
int main(){  
    char c = 'A';  
}
```

B  
C  
D  
a  
65



# Atenção!

- Execute o código abaixo:

```
int main() {  
    char a,b;  
  
    scanf("%c",&a);  
    scanf("%c",&b);  
    printf("%c e %c foram lidos.\n",a,b);  
    return 0;  
}
```

***Por que não funciona como esperado?***





# Buffer stdin

- É o **conhecimento técnico** que distingue os melhores profissionais...

*Input*

Buffer stdin

*Buffer: Região de memória para armazenamento temporário de dados, antes de serem efetivamente consumidos.*

*scanf()*



Aplicação



# Buffer stdin

- É o **conhecimento técnico** que distingue os melhores profissionais...

*Input*

Buffer stdin

*Buffer: Região de memória para armazenamento temporário de dados, antes de serem efetivamente consumidos.*

*scanf()*



É graças ao esquema de **bufferização** que você, enquanto programador, não precisa se preocupar em tratar eventos de teclado como: backspace, delete, shift, caps-lock...



**Aplicação**



# Alternativa 1

*Um espaço em branco no início do `scanf()` indica para que se ignore caracteres como: 'espaço em branco' ou '\n' existentes no buffer `stdin`.*

```
int main() {  
    char a,b;  
  
    scanf(" %c",&a);  
    scanf(" %c",&b);  
    printf("%c e %c foram lidos.\n",a,b);  
    return 0;  
}
```



# Alternativa 1

```
int main() {  
    char a,b;  
  
    scanf(" %c",&a);  
    scanf(" %c",&b);  
    printf("%c e %c foram lidos.\n",a,b);  
    return 0;  
}
```

*Um espaço em branco no início do `scanf()` indica para que se ignore caracteres como: 'espaço em branco' ou '\n' existentes no buffer `stdin`.*

*...tente escrever "algo" para variável a*





## Alternativa 2

*Consome todos os caracteres do buffer stdin até que encontre-se a tecla ENTER (\n)*

```
int main() {  
    char a,b;  
  
    scanf("%c",&a);  
    while(getchar()!='\n');  
    scanf("%c",&b);  
    while(getchar()!='\n');  
    printf("%c e %c foram lidos.\n",a,b);  
    return 0;  
}
```





# Buffer stdout

- É o **conhecimento técnico** que distingue os melhores profissionais...

```
int main() {  
    printf("ola mundo");  
    system("clear");  
    printf("OLA MUNDO");  
}
```

**system("clear")**

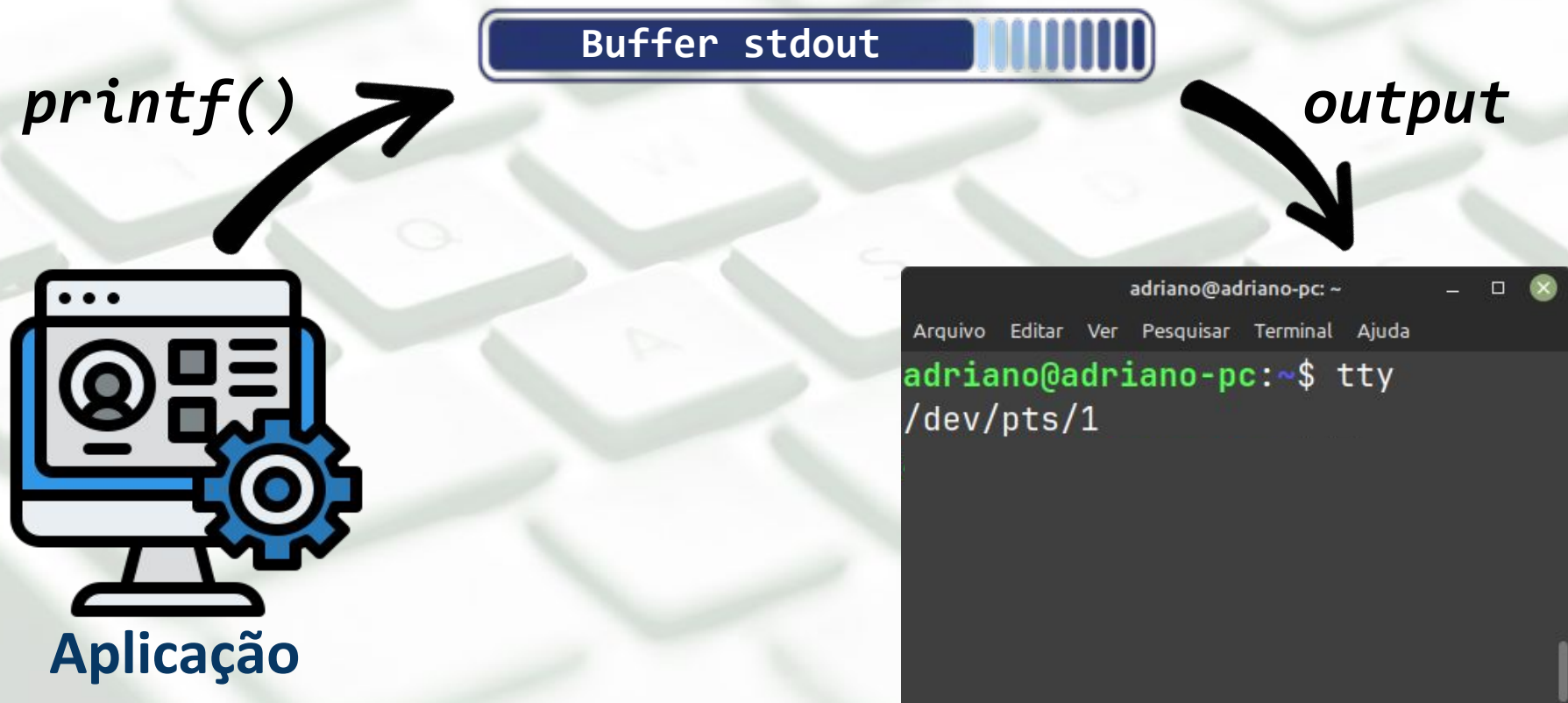
*é uma instrução para limpar (apagar) todo o conteúdo apresentado pelo terminal*

```
Terminal  
Arquivo Editar Ver Pesquisar Terminal Ajuda  
ola mundoOLA MUNDO  
-----  
(program exited with code: 0)  
Press return to continue
```



# Buffer stdout

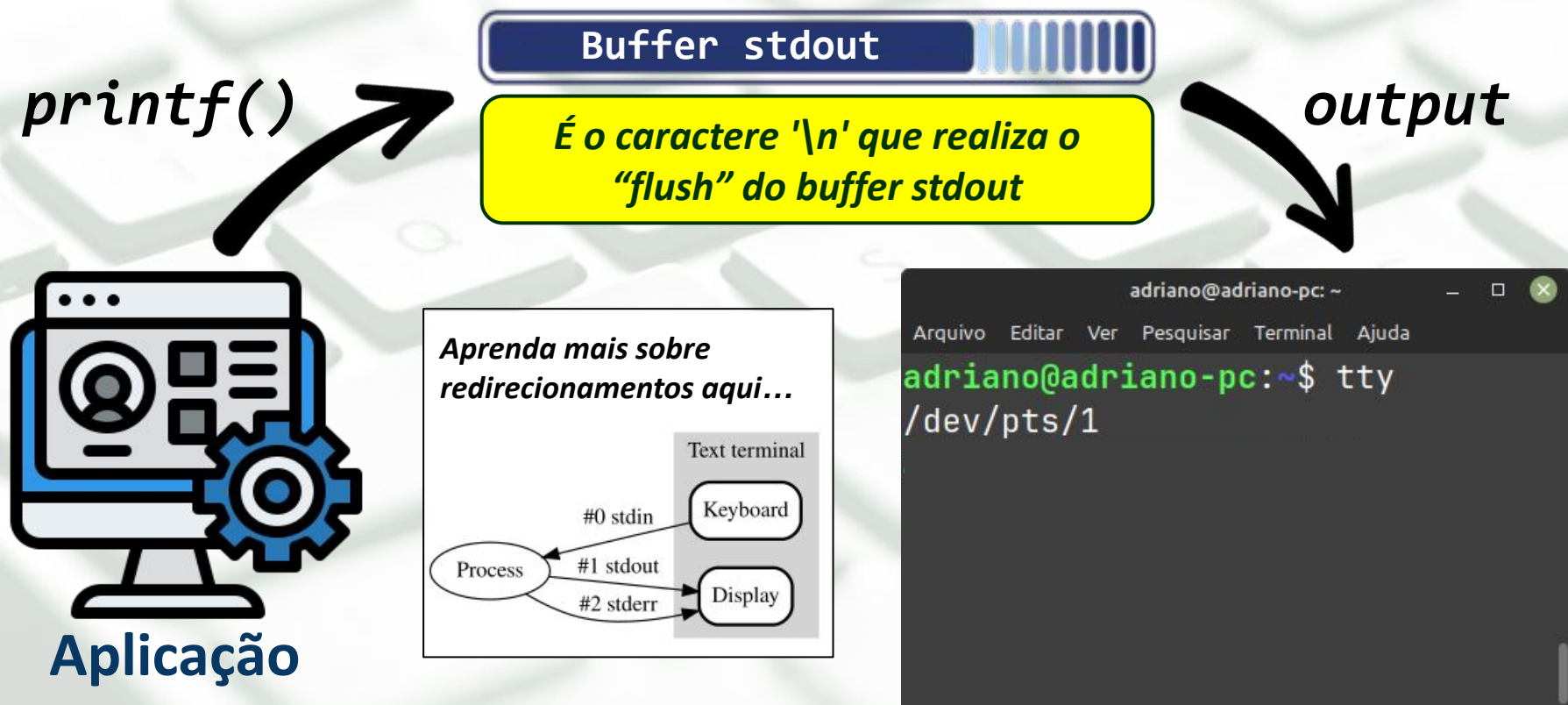
- É o **conhecimento técnico** que distingue os melhores profissionais...





# Buffer stdout

- É o **conhecimento técnico** que distingue os melhores profissionais...





# Buffer stdout

- É o **conhecimento técnico** que distingue os melhores profissionais...

```
int main() {  
    printf("ola mundo\n");  
    system("clear");  
    printf("OLA MUNDO\n");  
}
```

*É o caractere '\n' que realiza o "flush" do buffer stdout*

```
Terminal  
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda  
OLA MUNDO  
-----  
(program exited with code: 0)  
Press return to continue
```





# Bora CODAR!!!



1. É possível trocar o valor de duas variáveis sem utilizar uma terceira como apoio? Pense numa solução viável para esse problema.
2. Faça um programa em C que leia três números inteiros e imprima a média simples com três casas decimais.
3. Faça um programa que leia o tempo da duração de um evento expressa em segundos e mostre-o expresso no formato HH:MM:SS
4. Faça um programa que leia do usuário um símbolo (caractere) e retorne o código ASCII correspondente.
5. Faça um programa que leia dois símbolos numéricos ('0' a '9') no formato char e imprima o resultado (correto) da multiplicação destes valores.
6. Faça um programa que recebe um caractere minúsculo ('a' até 'z') e imprima o mesmo caractere, mas em grafia maiúscula.





# Bora CODAR!!!



7. Programe uma bomba de combustível: o usuário informa o preço do litro de combustível e o valor que o motorista deseja abastecer. Informe a quantidade de combustível que a bomba irá dispensar.
8. Joãozinho tem um cofre com muitas moedas, e deseja saber quantos reais conseguiu poupar. Faça um programa para ler a quantidade de cada tipo de moeda, e imprimir o valor total economizado, em reais.
9. Programe um caixa eletrônico. O usuário deve informar o valor que deseja sacar e o programa emite a menor quantidade de notas possíveis, totalizando o valor (Notas disponíveis: 100, 50, 20, 10, 5, 2 e 1).
10. O custo final de um produto numa fábrica qualquer é a soma do custo de produção (float), acrescido de X% de impostos e sobre isso e Y% do distribuidor. Sendo X e Y valores inteiros, faça um programa em C que imprima o custo final de um produto.