



# Linguagem de Programação

## ECT2303

helton.maia@ect.ufrn.br

## **Problema:** Calculando a distância segura para veículos.

No trânsito, em ruas e estradas, é aconselhável aos motoristas manterem entre os veículos um distanciamento de segurança. Esta separação assegura, folgadoamente, o espaço necessário para que se possa, na maioria dos casos, parar sem risco de colidir com veículo que se encontra na frente. Pode-se calcular esse distanciamento de segurança mediante a seguinte regra prática:

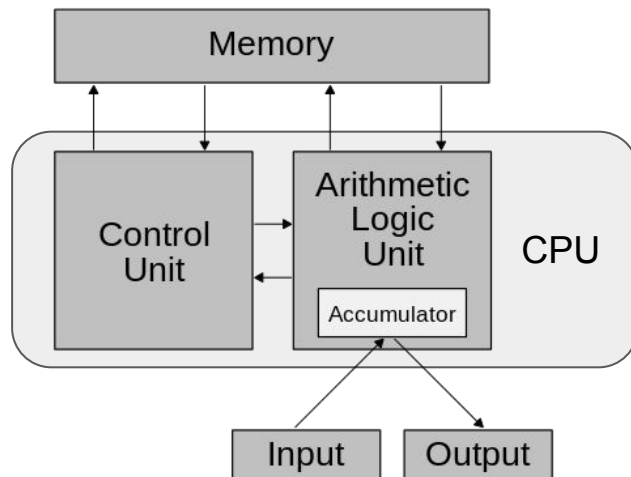
$$d(m) = \left( \frac{\textit{velocidade}(km/h)}{10} \right)^2$$

# Processos Básicos do Computador:



Estrutura interna:

[Fonte](#): wiki livros



**Variáveis:** Representam um espaço em memória, acessível através de um identificador, seu nome.

- Cada variável possui um tipo específico, que determina o tamanho em memória que será utilizado
- O nome da variável pode ser composto por sublinhado, letras e dígitos, devendo começar com uma letra.
- Letras maiúsculas e minúsculas são diferenciadas (*case-sensitive*)

## Variáveis: Tipos e descrição

- **char** - caractere  
Representa um número inteiro referente ao código ASCII de um determinado caractere.
- **int** – inteiro  
Representa um número inteiro.
- **float** - ponto flutuante.  
Representa um número real de precisão simples
- **double** - ponto flutuante de dupla precisão.  
Representa um número real de precisão dupla.
- **void** - sem valor.  
Utilizado em funções sem valor de retorno ou ponteiros genéricos.
- **bool**  
Armazena os valores lógicos verdadeiro e falso.
- **wchar\_t**  
Representam caracteres que exigem mais memória do que um simples char

# Tabela ASCII

Fonte:

<https://en.cppreference.com/w/cpp/language/ascii>

dec	oct	hex	ch
96	140	60	`
97	141	61	a

dec	oct	hex	ch	dec	oct	hex	ch	dec	oct	hex	ch	dec	oct	hex	ch
0	0	00	NUL (null)	32	40	20	(space)	64	100	40	@	96	140	60	`
1	1	01	SOH (start of header)	33	41	21	!	65	101	41	A	97	141	61	a
2	2	02	STX (start of text)	34	42	22	"	66	102	42	B	98	142	62	b
3	3	03	ETX (end of text)	35	43	23	#	67	103	43	C	99	143	63	c
4	4	04	EOT (end of transmission)	36	44	24	\$	68	104	44	D	100	144	64	d
5	5	05	ENQ (enquiry)	37	45	25	%	69	105	45	E	101	145	65	e
6	6	06	ACK (acknowledge)	38	46	26	&	70	106	46	F	102	146	66	f
7	7	07	BEL (bell)	39	47	27	'	71	107	47	G	103	147	67	g
8	10	08	BS (backspace)	40	50	28	(	72	110	48	H	104	150	68	h
9	11	09	HT (horizontal tab)	41	51	29	)	73	111	49	I	105	151	69	i
10	12	0a	LF (line feed - new line)	42	52	2a	*	74	112	4a	J	106	152	6a	j
11	13	0b	VT (vertical tab)	43	53	2b	+	75	113	4b	K	107	153	6b	k
12	14	0c	FF (form feed - new page)	44	54	2c	,	76	114	4c	L	108	154	6c	l
13	15	0d	CR (carriage return)	45	55	2d	-	77	115	4d	M	109	155	6d	m
14	16	0e	SO (shift out)	46	56	2e	.	78	116	4e	N	110	156	6e	n
15	17	0f	SI (shift in)	47	57	2f	/	79	117	4f	O	111	157	6f	o
16	20	10	DLE (data link escape)	48	60	30	0	80	120	50	P	112	160	70	p
17	21	11	DC1 (device control 1)	49	61	31	1	81	121	51	Q	113	161	71	q
18	22	12	DC2 (device control 2)	50	62	32	2	82	122	52	R	114	162	72	r
19	23	13	DC3 (device control 3)	51	63	33	3	83	123	53	S	115	163	73	s
20	24	14	DC4 (device control 4)	52	64	34	4	84	124	54	T	116	164	74	t
21	25	15	NAK (negative acknowledge)	53	65	35	5	85	125	55	U	117	165	75	u
22	26	16	SYN (synchronous idle)	54	66	36	6	86	126	56	V	118	166	76	v
23	27	17	ETB (end of transmission block)	55	67	37	7	87	127	57	W	119	167	77	w
24	30	18	CAN (cancel)	56	70	38	8	88	130	58	X	120	170	78	x
25	31	19	EM (end of medium)	57	71	39	9	89	131	59	Y	121	171	79	y
26	32	1a	SUB (substitute)	58	72	3a	:	90	132	5a	Z	122	172	7a	z
27	33	1b	ESC (escape)	59	73	3b	;	91	133	5b	[	123	173	7b	{
28	34	1c	FS (file separator)	60	74	3c	<	92	134	5c	\	124	174	7c	
29	35	1d	GS (group separator)	61	75	3d	=	93	135	5d	]	125	175	7d	}
30	36	1e	RS (record separator)	62	76	3e	>	94	136	5e	^	126	176	7e	~
31	37	1f	US (unit separator)	63	77	3f	?	95	137	5f	_	127	177	7f	DEL (delete)

# Declaração de variáveis

A definição da variável informa ao compilador, onde e a quantidade de memória necessária para criação da variável.

## Atribuição:

- A declaração de variáveis somente reserva espaço na memória para ser utilizado pelo programa.
- O comando de atribuição é responsável por armazenar as informações em uma variável.

## Nome    Tipo

```
int     i, j, k;
```

```
char    c, ch;
```

```
float   f, salary;
```

```
double x;
```

# Inicialização das variáveis

tipo nome = valor;

**exemplos:**

```
float f = 5.5;
```

```
bool b = true;
```

```
int z = 22;
```

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



## Lista de **palavras reservadas** para o C++

Não podem ser usadas como Constantes ou variáveis ou quaisquer outros nomes de identificadores.

and	decltype	new	
and_eq	default	noexcept	switch
	delete	not	template
alignas	double	not_eq	this
	dynamic_cast		thread_local
alignof	else	nullptr	throw
asm	enum	operator	true
auto		or	try
bitand	explicit	or_eq	typedef
bitor	export	private	typeid
bool		protected	typename
break	extern	public	union
case	false	register	unsigned
catch	float	reinterpret_cast	using
char	for	return	virtual
char16_t	friend	short	void
char32_t	goto	signed	volatile
class	if	sizeof	wchar_t
compl	inline	static	while
const	int	static_assert	xor
constexpr	long	static_cast	xor_eq
const_cast	mutable	struct	
continue	namespace		

## Exemplo: Declarando e inicializando variáveis:

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

int main () {
    // declarando variaveis
    int a, b;
    int c;
    float f;

    // inicializacao das variaveis
    a = 10;
    b = 20;
    c = a + b;

    cout << c << endl ;

    f = 70.0/3.0;
    cout << f << endl ;

    return 0;
}
```

# Utilização de memória para diferentes tipos:

Modificadores

Tipo de dados

Tamanho (Bytes)

Faixa



short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
long long int	8	$-(2^{63})$ to $(2^{63})-1$
unsigned long long int	8	0 to 18,446,744,073,709,551,615
signed char	1	-128 to 127
unsigned char	1	0 to 255
float	4	
double	8	
long double	12	
wchar_t	2 or 4	1 wide character

## Utilização de memória para diferentes tipos:

- Toda variável deve ser declarada antes de ser utilizada

```
// Imprimir os tamanhos para cada tipo de dados

#include<iostream>
using namespace std;

int main()
{
    cout << "Size of char : " << sizeof(char)
        << " byte" << endl;
    cout << "Size of int : " << sizeof(int)
        << " bytes" << endl;
    cout << "Size of short int : " << sizeof(short int)
        << " bytes" << endl;
    return 0;
}
```

# Constantes

Utilizadas em comandos para representar valores fixos de um dado tipo, não sendo possível alterar seus valores durante a execução do programa.

Estágios do desenvolvimento:

- ***Compile time***: Ocorre no processo de compilação do seu programa, também chamado de tempo de compilação, o compilador garante que seu código esteja sintaticamente correto e o converte em arquivos executáveis.
- ***Runtime***: Etapa de execução (linha por linha) do programa.

**Constantes Inteiras**: números sem ponto decimal, precedidos ou não por um sinal.

Ex: 2123 -8 3200

**Constantes em ponto flutuante**: é requerido um ponto decimal, seguido da parte fracionária do número. Ex: 987.12 4.34e3 12.001

# Declarando Constantes

**Modificador `const`:** Garante que o valor salvo em um endereço de memória não seja alterado durante a execução do programa.

```
const <tipo> nome = <valor>;
```

```
const int A = 10;
```

```
const float GRAVIDADE = 9.8;
```

```
const double T = 1e-10;
```

```
const int G {9.8};
```

# Constantes Simbólicas definidas por difetivas ao pré-processador

Sintaxe: `#define <nome> <valor>`

```
#include <iostream>
#define GRAVIDADE 9.80665
using namespace std;
```

```
int main(){
cout << "Aceleracao da gravidade: " << GRAVIDADE<< endl;
return 0;
}
```

# Operadores

Um operador é um símbolo que informa ao compilador sobre a execução de manipulações matemáticas ou lógicas específicas. O C ++ é rico em operadores e fornece diversos tipos, são eles:

- Unário: ++, --
- Aritméticos: + - \* / %
- Relacionais: == != > < >= <=
- Lógicos: && || !
- *Bitwise*: & | ^ ~ << >>
- Comparativos: = += -= \*= /= %=
- Outros: sizeof ?:



# Operadores

**Importante:** As operações são realizadas considerando a precisão dos operandos, sempre o maior.

## Incremento e decremento:

Operador	Ação
++	Soma 1 ao seu operando
--	Subtrai 1 do seu operando

++x;  
x++;    Equivalente ao uso de  $x = x + 1$ ;

--x;  
x--;    Equivalente ao uso de  $x = x - 1$ ;

# Operadores

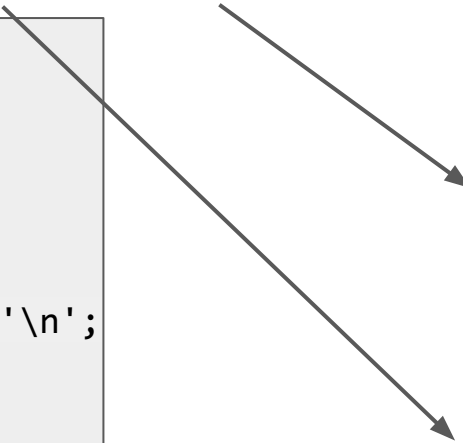
Incremento e decremento: Pré-fixado e Pós-fixado

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

int main()
{
    int n1 = 1;
    cout << "resultado inicial" << '\n';
    cout << "n1 = " << n1 << '\n';
    int n2 = ++n1*2;
    int n3 = n1++*2;

    cout << "resultado final" << endl;
    cout << "n1 = " << n1 << '\n'
         << "n2 = " << n2 << '\n'
         << "n3 = " << n3 << '\n';

    return 0;
}
```



X++;  
X--;

++X;  
--X;

Output ?

# Operadores: `sizeof`

- É um operador de tempo de compilação (compile-time) que retorna o tamanho em bytes de uma variável, tipo ou valor de expressão;
- Sintaxe:  
`sizeof (<tipo>);`  
`sizeof (<expressão>);`  
`sizeof (<variavel>);`

# Operadores (exemplo `sizeof`)

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

int main() {
    cout << "Tamanho do char : " << sizeof(char) << endl;
    cout << "Tamanho do int : " << sizeof(int) << endl;
    cout << "Tamanho do short int : " << sizeof(short int) << endl;
    cout << "Tamanho do long int : " << sizeof(long int) << endl;
    cout << "Tamanho do float : " << sizeof(float) << endl;
    cout << "Tamanho do double : " << sizeof(double) << endl;
    cout << "Tamanho do wchar_t : " << sizeof(wchar_t) << endl;

    return 0;
}
```

# Expressões (precedência dos operadores)

Operadores de mesmo nível de precedência, isto é, na mesma linha, são avaliados da esquerda para a direita.

Mais alta



Mais baixa

++ --  
/ % \*  
+ -

**Obs:** Use parênteses para forçar uma ou mais operações a terem precedência maior, ou no caso de não ter certeza de qual operador tem maior precedência.

# Expressões com Operadores lógicos e relacionais

## Relacionais

Operador	Ação
>	Maior que
>=	Maior ou igual que
<	Menor que
<=	Menor ou igual que
==	Igual
!=	Diferente

## Lógicos

Operador	Ação
&&	AND
	OR
!	NOT

# Expressões com Operadores lógicos e relacionais

Operadores lógicos e relacionais possuem menor precedência do que operadores aritméticos.

Precedência:

**Mais alta**  
↓  
**Mais baixa**

!  
> >= < <=  
== !=  
&&  
||

20 > 2 + 1

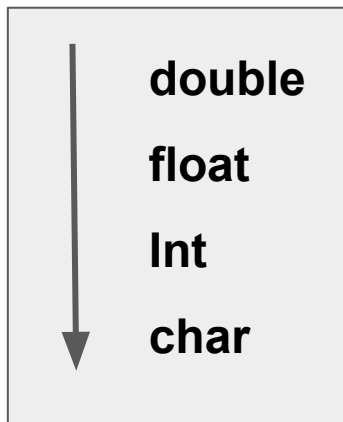
É avaliado como

20 > ( 2+1 )

# Conversões

Quando operandos de tipos diferentes são misturados em uma expressão, os valores são convertidos no tipo do maior operando.

Prioridade para converter:





# Conversões implícitas de tipos em expressões

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

int main(){
    int x = 10;    // inteiro x
    char y = 'a';  // caracter c

    // y convertido em int. ASCII
    // valor de 'a' = 97
    x = x + y;

    // x implicitamente convertido em float
    float z = x + 1.1;

    cout << "x: " << x << " y: " << y << " z: " << z << endl;

    return 0;
}
```

**Output ?**

# Conversões explícitas de tipos utilizando *Casting*

**Sintaxe:** (tipo) expressão

**Exemplo:**

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

```
int main(){
    double x = 1.2;

    // conversao explicita double para int
    int sum = (int)x + 1;

    cout << sum << endl;

    return 0;
}
```

**Output ?**