PMR 3100 – Introdução à Engenharia Mecatrônica

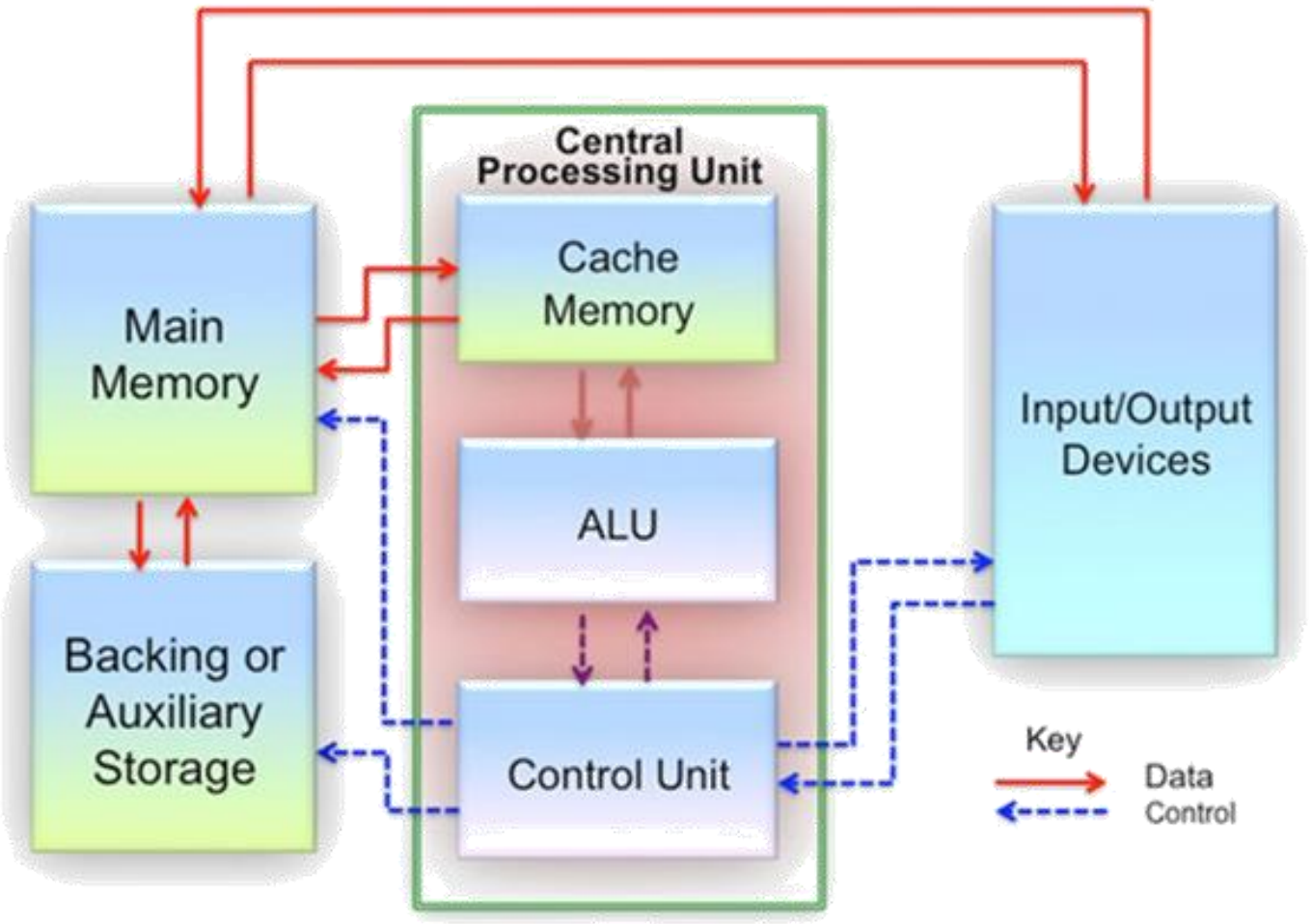
Módulo 04 – Meu Primeiro Robô

Aula 02 – Linguagem C, conversão entre tipos

Prof. Dr. Rafael Traldi Moura

Microprocessador

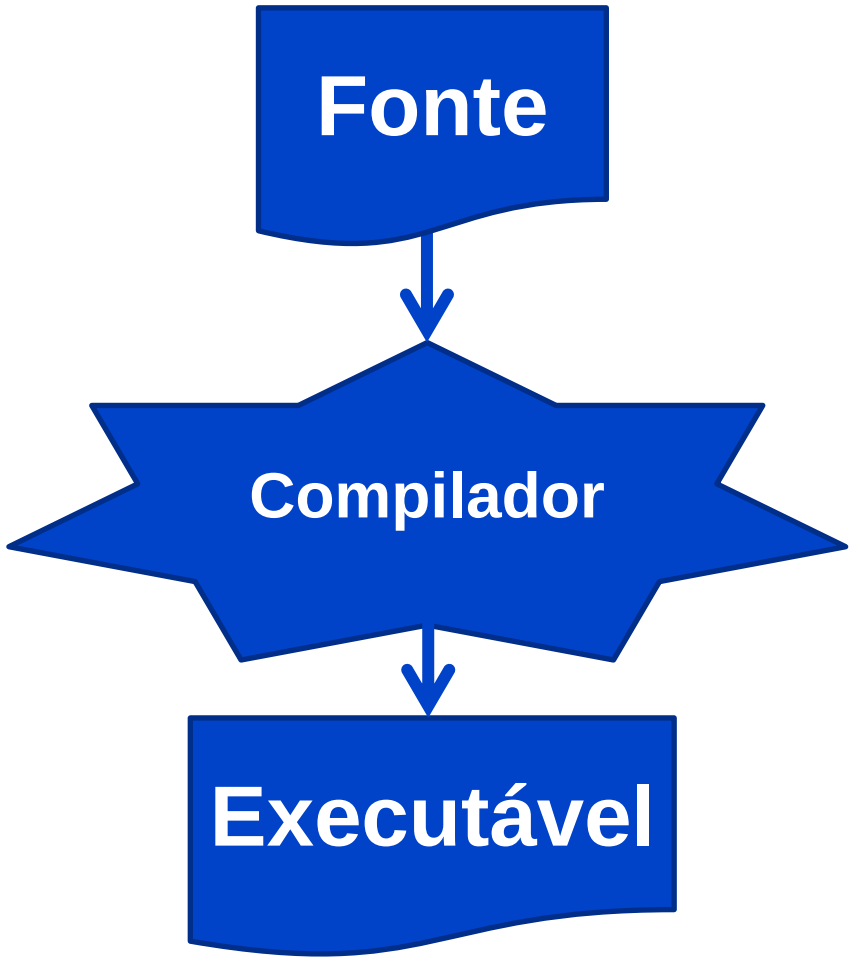
- Um microprocessador, MPU, é um CI (circuito integrado) composto por uma unidade de controle, uma ALU (Arithmetic Logic Unit) e memórias (registradores e cache).
- A ALU faz operações aritméticas e lógicas (é igual? É maior? Etc...). A unidade de controle lê uma instrução e faz a ULA executá-la. O conjunto de instruções (*instruction set*) pode ser reduzido (RISC, ex.: ARM, Apple Silicon) ou complexo (CISC, ex: x86 da Intel e da AMD);

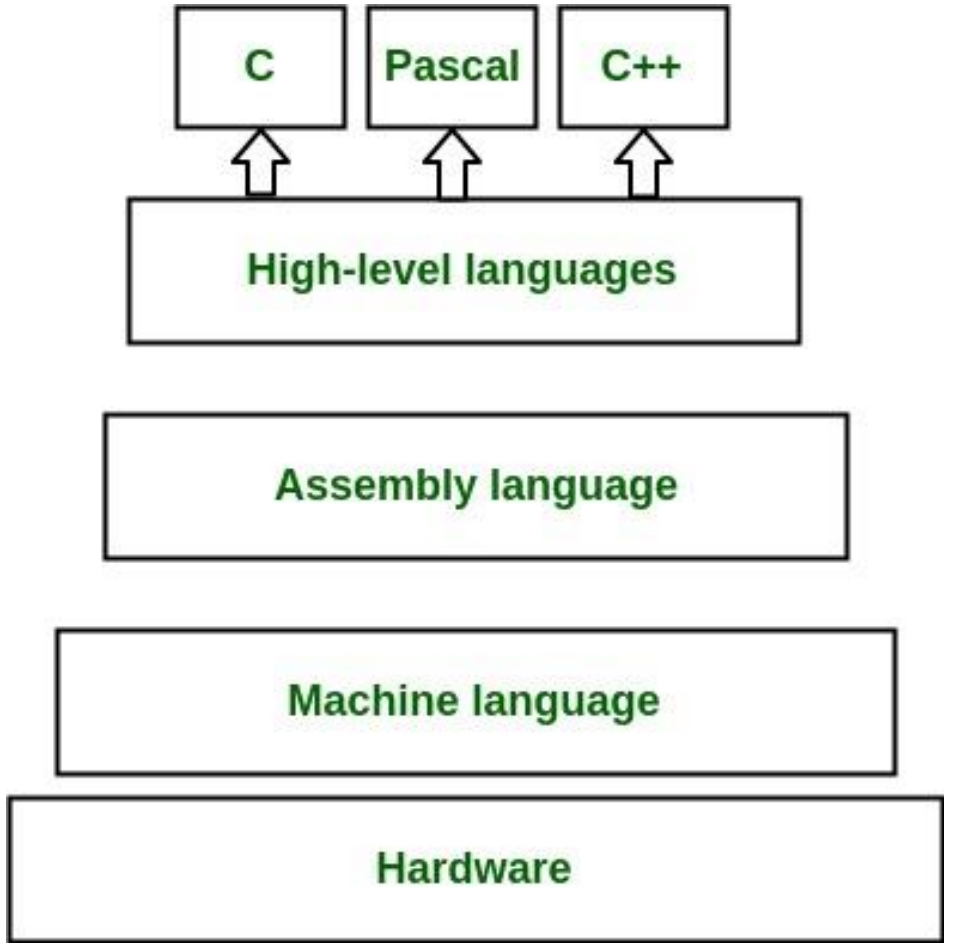


- Um programa de computador nada mais é do que um conjunto de instruções e valores. Essas instruções compõe a linguagem Assembly dos microprocessadores, ou Assembler;

C e Python - Interpretadores

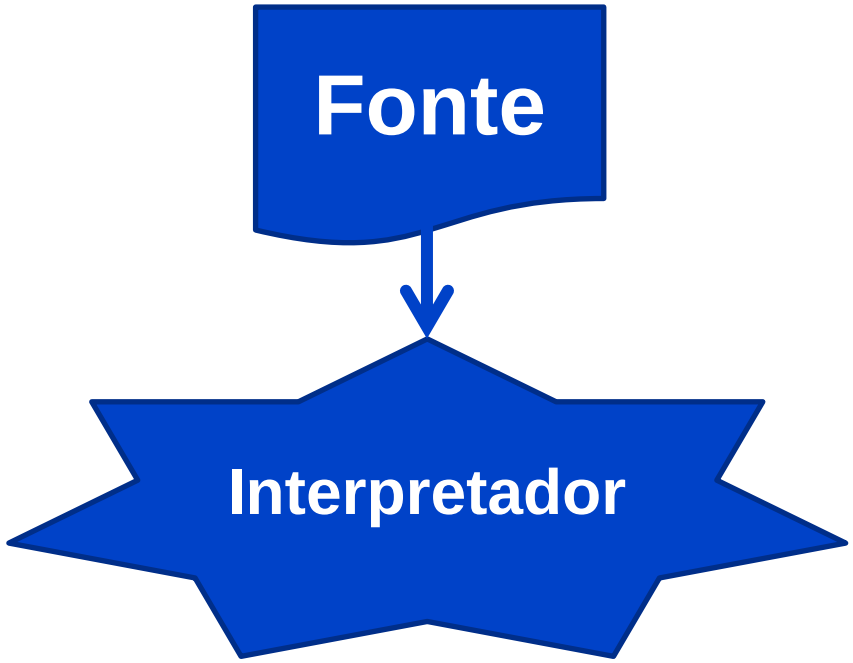
- C roda em linguagem de máquina;

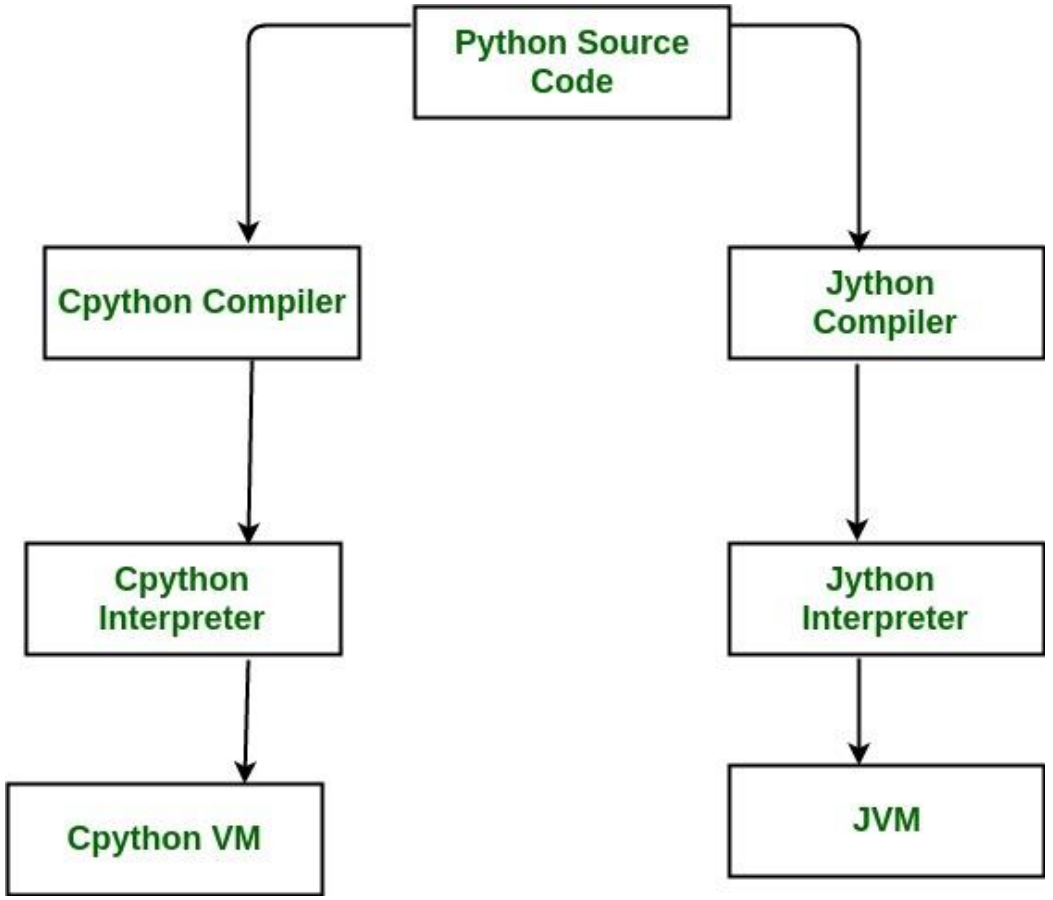




C e Python - Interpretadores

- Python roda em um interpretador;



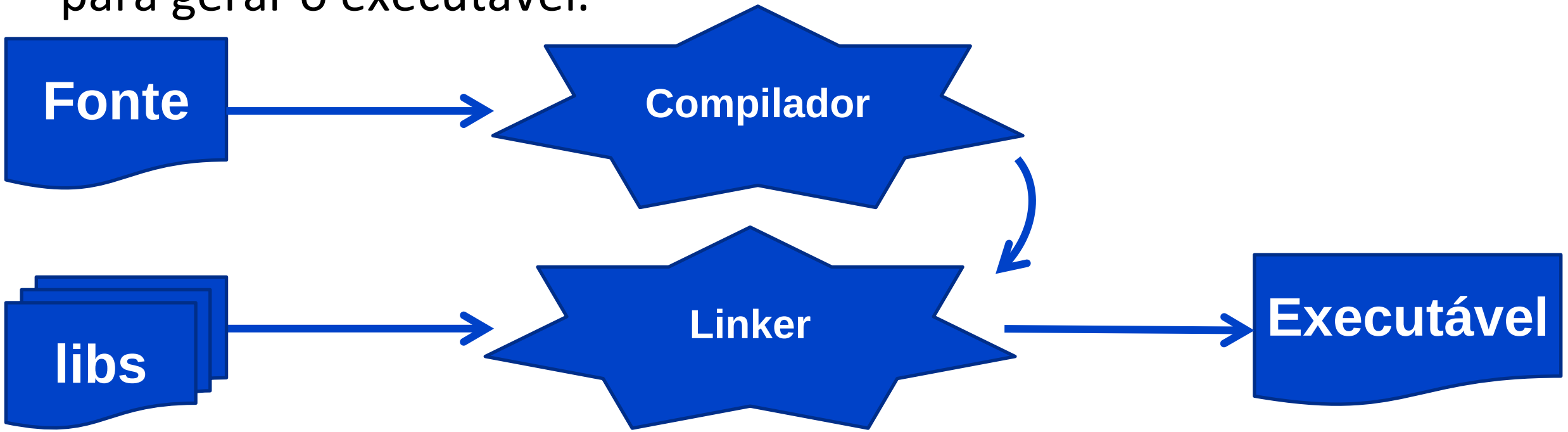


C e Python – Qual usar?

- A escolha da linguagem ideal para um programa depende muito da aplicação. Um sábio disse uma vez que a melhor linguagem é aquela que você sabe! :p
- Para elementos finitos, que é um software que simula a física a nossa volta e usamos, por exemplo, em testes virtuais de *crashtest*, usamos FORTRAN;
- Para problemas de Inteligência Artificial, geralmente usamos Python;
- Para microcontroladores, como o Arduino, usamos C ou Assembler.

C - bibliotecas

- Bibliotecas que são utilizadas são declaradas com #include. O #include inclui textos que são referencias para as chamas das funções das bibliotecas;
- Note que estas bibliotecas são **linkadas** (LINK) com o seu código para gerar o executável.



Preprocessador C

- Include

```
#include <xc.h>
#include <stdio.h>
#include "always.h"
#include "delay.h"
```

Inclusos no diretório do path

Inclusos no diretório do projeto

- Define

```
#define INPUT 1
#define OUTPUT 0
#define TRUE 1
#define FALSE 0
#define HIGH 1
#define LOW 0
#define hi 1
#define lo 0
#define OFF 0
#define ON 1
```

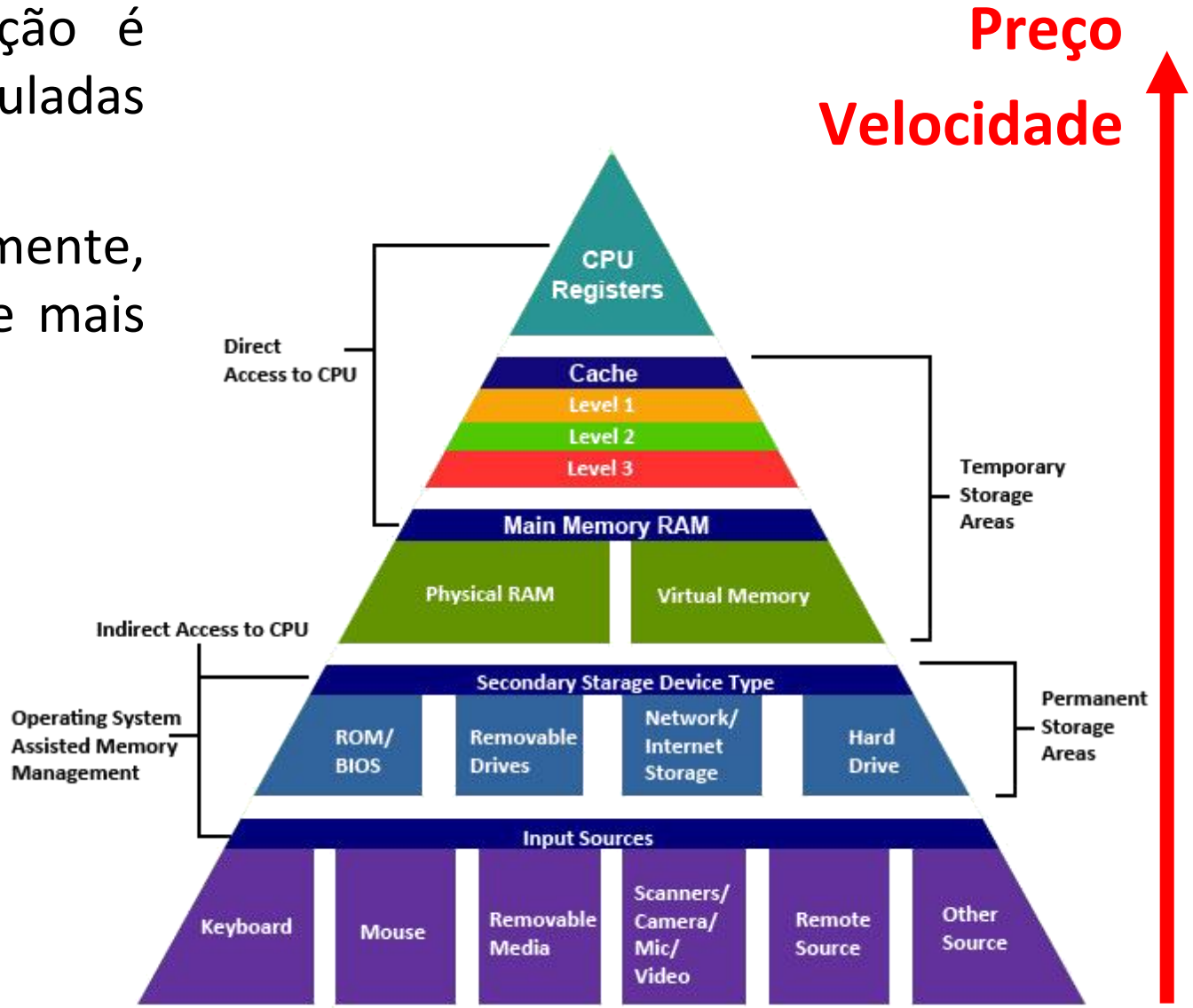
Memória

A memória é um componente cuja função é armazenar informações que serão manupuladas pelo sistema quando necessário.

Existem vários tipos de mémoria. Geralmente, quanto mais *próxima* da ULA, mais rápida e mais cara é a memória. Alguns tipos são:

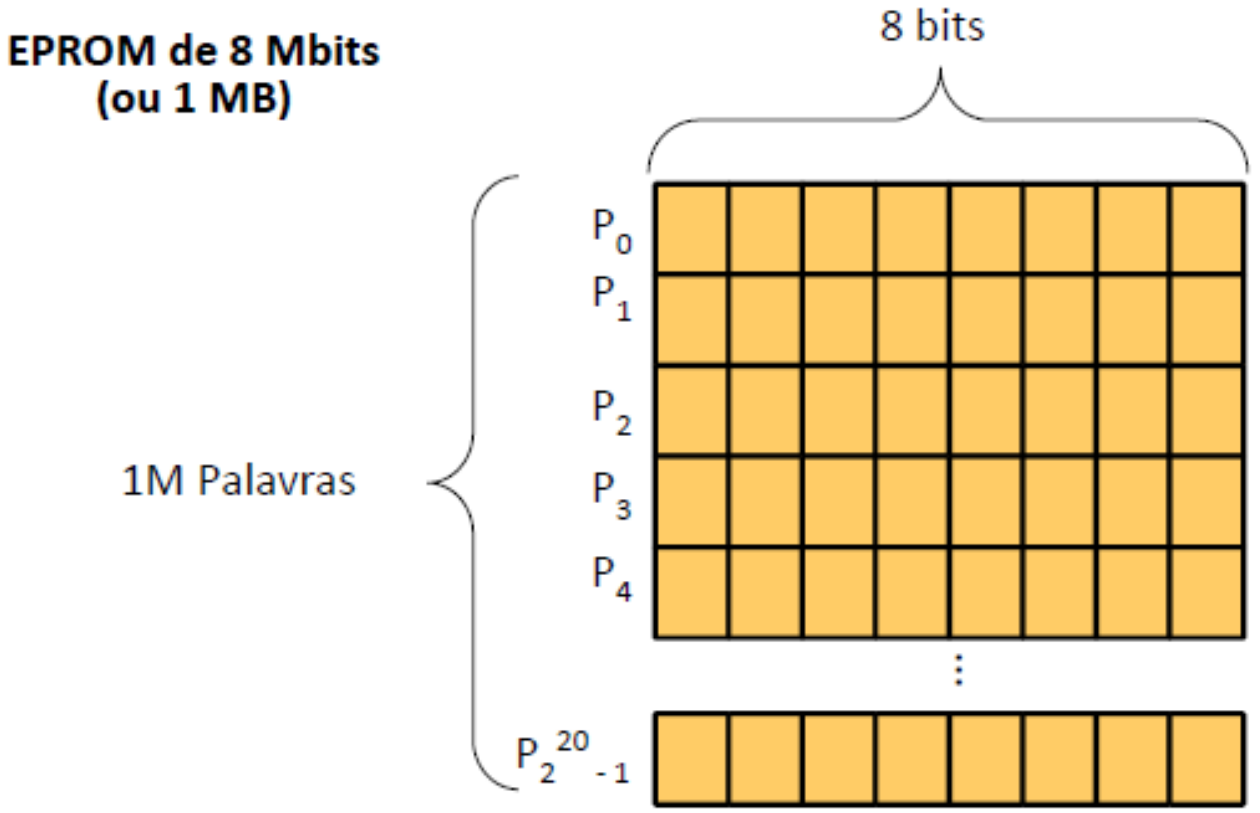
- RAM – Memória de acesso aleatório;
- ROM – Memória de apenas leitura;
- PROM – ROM programável;
- EPROM – PROM apagável;

- EEPROM – PROM eletricamente apagável;



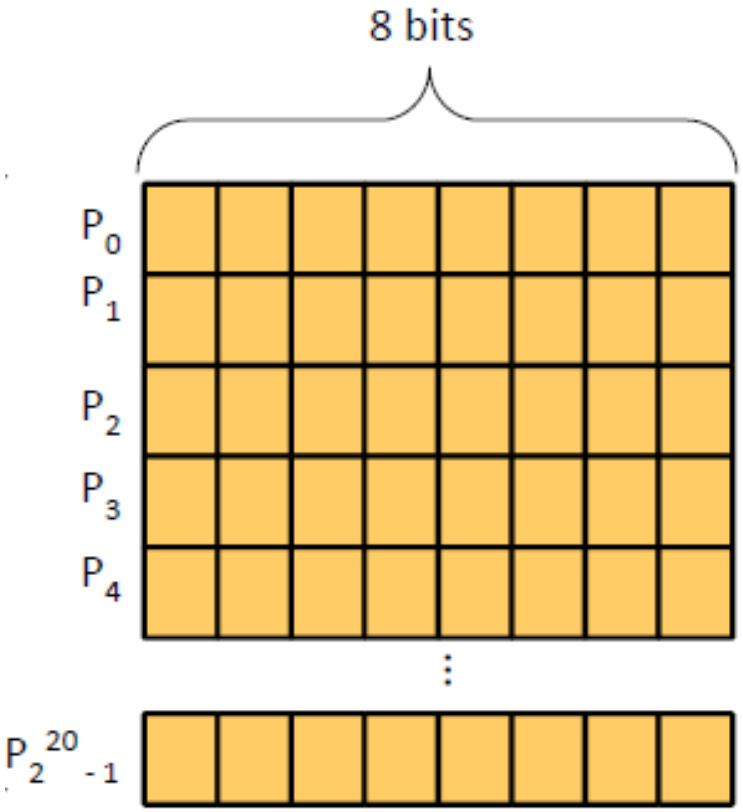
Memória – Arranjo Estrutural

- O arranjo físico do armazenamento de dados na memória está ilustrado na figura ao lado.
- Cada *espaço* para armazenamento recebe um valor de identificação, como o número da sua casa serve para identificar a posição da rua em que você mora. Esse valor de identificação é conhecido como **endereço**.
- Assim como o maior número que se pode representar com 2 algarismos é 99, repare que a largura da matriz ao lado, dada em bits, determina o maior inteiro a ser armazenado, sendo neste caso 2^8-1=255.



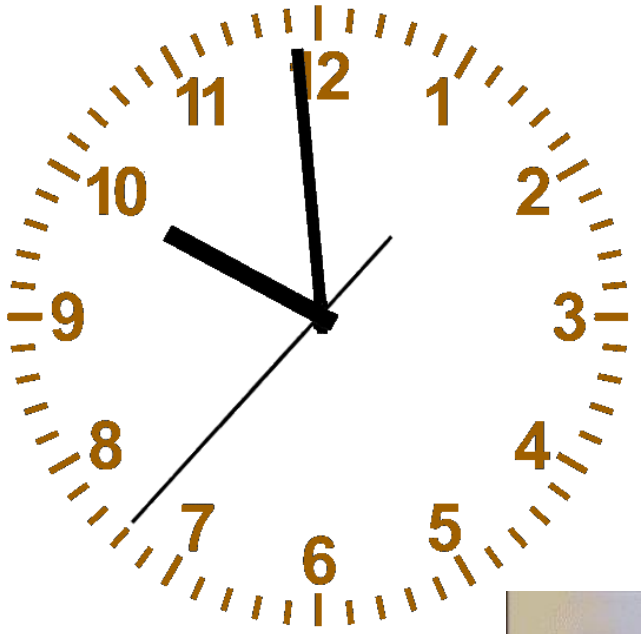
Memória – Arranjo Estrutural

- Como vimos anteriormente, o programa em C é compilado diretamente em linguagem de máquina;
- Por isso, quando você dá um duplo clique no executavel *NomeDoPrograma.exe*, o Sistema Operacional já analisa memória ram e reserva o espaço necessário para as variáveis do programa;
- Desta forma, é importante pensar que um boolen poderia ocupar apenas 1 bit, mas acaba ocupando um byte (8 bits).
- Além disso, 255 pode ser um valor inteiro muito pequeno para usarmos. Então temos inteiros que ocupam dois endereços de memóra, ou seja, são de 16 bits (0 a 65535);
- Ou seja, C tem diferentes tipos de variável que ocupam espaços diferentes na memória. Para entender melhor, vamos analisar diferentes sistemas de numeração e conversões.



Sistemas de numeração

- Nós estamos acostumados a usar o sistema decimal de numeração, ou base 10. Em outras palavras, temos 10 algarismos para representar os números.
- Entretanto, temos outros 2 sistemas muito utilizados em nosso dia a dia:
  - Sistema sexagesimal (base 60): segundos em um minuto e minutos em uma hora;
  - Sistema duodecimal (base 12): meses do ano;



Sistemas de numeração

- Entretanto, existe um outro tipo de sistema muito conhecido: **o binário (base 2)**. Neste, usamos apenas os algarismos 0 e 1 para representar qualquer número inteiro;
- Para converter de decimal para binário, fazemos **divisões por** 2 e olhamos os restos;
- Para converter de decimal para binário, fazemos as somas das potências de 2.

$$1011_{(2)} = 11_{(10)}$$

11	2
1	5
	2
1	2
	2
0	1
	2
1	0

Menos significativo

Mais significativo

$$1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 =$$

$$1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 =$$

$$8 + 0 + 2 + 1 = 11$$

Sistemas de numeração

- No sistema binário, como temos apenas 0 e 1, precisamos de muitos algarismos para representar valores grandes. Por exemplo, são necessários 4 algarismos binários para representar o decimal 11;
- Para facilitar, temos o costume de representar um binário de 4 algarismos condensado em apenas um algarismo hexadecimal;
- Do 0 ao 9, não há diferenças explicitas entre o sistema decimal e o hexadecimal;
- Entretanto, do 10 ao 15, usamos as letras maiúsculas A a F como algarismos.

$$1011_{(2)} = 11_{(10)} = B_{(16)}$$

$0_{(10)} = 0_{(16)}$
$1_{(10)} = 1_{(16)}$
$2_{(10)} = 2_{(16)}$
$3_{(10)} = 3_{(16)}$
$4_{(10)} = 4_{(16)}$
$5_{(10)} = 5_{(16)}$
$6_{(10)} = 6_{(16)}$
$7_{(10)} = 7_{(16)}$
$8_{(10)} = 8_{(16)}$
$9_{(10)} = 9_{(16)}$

$10_{(10)} = A_{(16)}$
$11_{(10)} = B_{(16)}$
$12_{(10)} = C_{(16)}$
$13_{(10)} = D_{(16)}$
$14_{(10)} = E_{(16)}$
$15_{(10)} = F_{(16)}$

Tipos de variáveis - radicais numéricos

Radix	Format	Example
binary	`0b` *number* or `0B` *number*	0b10011010
octal	`0` *number*	0763
decimal	*number*	129
hexadecimal	`0x` *number* or `0X` *number*	0x2F

- Exemplo:

```
char x, y, z, c;      // variáveis de 8-bits sem sinal (0 e 255)
x = 65;
y = 0x41;             // (x == y) é True ou False ?
z = 0b01000001;       // (x == z) é True ou False ?
c = 'A' ;             // (x == z) é True ou False ?
```

Programa exemplo em C

- Todo programa em C se inicia pela execução **main**;
- Em C, tudo deve ser declarado antes de ser usado. Isso inclui declarar variáveis, constantes, funções, etc.

```
#include <stdio.h>
int main(){
    int teste;
    teste = 129;
    //teste = 0x81;
    //teste = 0b10000001;
    //teste = 'A';
    printf("%d", teste);
    return 0;
}
```

USAR

este compilador online

Tipos de variáveis – códigos para texto ASCII

- American Standard Code for Information Interchange;
- Publicado em 1963, revisto em 1967;
- Código com 7-bits para representar texto em computadores e equipamentos de comunicação (telégrafo na época);

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

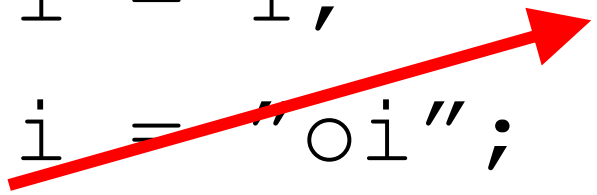
Tipos de variáveis

- Em Python, podemos fazer assim:

```
i = 1
i = "oi"
```

- Já em C, o tipo de variável tem a ver com o seu tamanho e não com o seu conteúdo. **O tamanho do tipo de variável tem a ver com o compilador**.

```
int i;
i = 1;
i = "oi";
```



Nunca atribuir um valor diferente do indicado na declaração da variável.
Devido ao tamanho da variável.

Tipos de variáveis no compilador XC8 (PIC)

- Em C, o tipo de variável tem a ver com o seu tamanho e não com o seu conteúdo. **O tamanho do tipo de variável tem a ver com o compilador**.

Tipo	Standard ISO types	Tamanho (bits)	Tipo aritmético
bit		1	unsigned integer
signed char	int8_t	8	signed integer
unsigned char	uint8_t	8	unsigned integer
signed short		16	signed integer
unsigned short		16	unsigned integer
signed int	int16_t	16	signed integer
unsigned int	uint16_t	16	unsigned integer
signed short long	int24_t	24	signed integer
unsigned short long	uint24_t	24	unsigned integer
signed long	int32_t	32	signed integer
unsigned long	uint32_t	32	unsigned integer
float		24 ou 32	real
double		24 ou 32	real

OBS: O tipo default está sublinhado. ISO Types em <stdint.h>.

Tipos de variáveis em XC8 - Exemplos

Exemplos:

```
// C types
char a; // variável de 8-bits sem sinal, pode armazenar valores
        // entre 0 a 255
int b;  // variável de 16-bits com sinal, pode armazenar valores
        // entre -32.768 e 32.767 (em complemento de 2)

long c; // variável de 32-bits com sinal, pode armazenar valores
        // entre -2.147.483.648 e 2.147.483.647 (em complemento de 2)

// ISO standard types
int8_t x;    //variável de 8-bits com sinal, pode armazenar valores
             // entre -128 a 127
uint16_t y; // variável de 16-bits sem sinal, pode armazenar valores
             // entre 0 e 65.535
uint32_t z; // variável de 32-bits com sinal, pode armazenar valores
             // entre 0 e 4.294.967.295
```

Tipos de variáveis - **printf**

- Em Python, podemos fazer assim:

```
i = 1
print i
```

- Já em C, usamos a função printf:

```
int i;
i = 1;
printf("%d", i);
```

Tipos de variáveis - **printf**

- Mas o que é "%d"? Como explicado, cada variável tem um tamanho diferente, então a função tem que saber qual o tipo e como imprimir (inteiro pode ser impresso em binário, hexadecima, ASCII ou decimal).
- Usamos esta tabela:

Character	Argument type; Printed As
d,i	int; decimal number
o	int; unsigned octal number (without a leading zero)
x,X	int; unsigned hexadecimal number (without a leading 0x or 0X), using abcdef or ABCDEF for 10, ...,15.
u	int; unsigned decimal number
c	int; single character
s	char *; print characters from the string until a '\0' or the number of characters given by the precision.
f	double; [-]*m.dddddd*, where the number of *d*'s is given by the precision (default 6).
e,E	double; [-]*m.ddddddE+/-xx* or [-]*m.ddddddE+/-xx*, where the number of *d*'s is given by the precision (default 6).
g,G	double; use %e or %E if the exponent is less than -4 or greater than or equal to the precision; otherwise use %f. Trailing zeros and a trailing decimal point are not printed.

Programa exemplo em C

- Execute nosso mesmo exemplo mas mudando apenas a forma de imprimir na tela entre decima, binário, hexadecimal e ASCII.

```
#include <stdio.h>
int main(){
    int teste;
    teste = 129;
    //teste = 0x81;
    //teste = 0b10000001;
    //teste = 'A';
    printf("%d", teste);
    return 0;
}
```

USAR

este compilador online

Tipos de variáveis – vetores em C

- Em Python existem “lists”, que podem conter quaisquer dados.

```
data = [["name1", "long name1", 1, 2, 3],
        ["name2", "long name2", 5, 6, 7]]
```

- Já em C, existem arrays (vetores), que podem ser declarados e usado da mesma forma. O índice começa em zero.

```
int valores[3];
valores[0] = 1;
x = valores[2];
```

Tipos de variáveis – Códigos para texto

```
char z;          // variável de 8-bits (1 byte) sem sinal
                 // (0 e 255)
...

z = "A";         // Está correto?
```

NÃO! “A” representa uma string e necessita de 2 bytes para ser armazenado em C

Tipos de variáveis – Strings em C

- String em C é um vetor de char terminado por zero. Zero significa:
  - `0    // em decimal`
  - `0x0  // em hexadecimal`
  - `0b0  // em binário`
  - `'\0' // em escape sequence (caracter nulo)`
- Assim, o espaço para armazenar um string deve ser o número de caracteres mais um
  - `char myString[6];    // armazena 5 caracteres e um // terminador`
- Pode-se atribuir uma string em C com a função
  - `sprintf(myString, "vel = %d", vel);//o uso é igual ao //printf(), mas o string é armazenado no vetor`

Sintaxe do if

- Python usa *:* e *tab*;

```
if x == 0:
      x = x + 1
else
     x = x + 2
```

- C usa *{* , *}* e termina comando com *;*;

```
 if (x == 0) {
     x = x + 1;
} else (x == 1) {
     x = x + 2;
}
```

Sintaxe do while

- Python usa *:* e *tab*;

```
nota = 10
while nota > 1:
    nota = nota / 2
    print(nota)
```

- C usa *{* , *}* e termina comando com *;*;

```
float nota;
nota = 10.0;
while(nota > 1.0) {
    nota = nota / 2.0;
    printf("%f", nota);
}
```

Funções

Python usa *def* para funções;

```
def fat(n):
    if n==0:
        return 1
    else:
        return n * fat(n-1)
x = fat(2)
```

C usa uma declaração de tipo para as funções;

Funções do tipo *void* não tem retorno;

```
int fat(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * fat(n -1);
    }
}
x = fat(2);
```

Funções – passagem de parâmetros

- Em C, todos os parâmetros são passados “por valor”, ou seja, não modifica o valor da variável passada como parâmetro;

```
/* power: base elevada à n-ésima potência; n >= 0 */
int power(int base, int n) {
 int p;
 p = 1;
 while (n > 0;){
       p = p * base;
       n = n - 1;
 return p;
}
```

n é usado como variável na função, mas seu valor original não é modificado

Programa exemplo com funções em C

- Todo programa em C se inicia pela execução **MAIN**;
- Em C, tudo deve ser declarado antes de ser usado. Isso inclui declarar variáveis, constantes, funções, etc.

```
#include <stdio.h>
int fat (int n) {
  if (n == 0)
    return 1;
  else
    return n * fat(n-1);}
void main(void) {
  int resultados[3];
  int i, j;
  j = 0;
  i = 2;
  while (i < 5) {
    resultados[j] = fat(i);
    i = i + 1;
    j = j + 1;
  }
  printf("%d %d %d", resultados[0], resultados[1], resultados[2]);
}
```

USAR

este compilador online

Operadores

- Aritméticos: `+ - * / % ;`
- Lógicos: `> >= < <= == != && ||;`
- Incremento e decremento: `++ --`:
  - Pré-incremento
    - `y = ++x;  // incrementa antes e usa o valor //depois`
  - Pós-incremento
    - `y = x++;  // usa o valor primeiro e incrementa //depois`

Operadores

- Exemplos de pós e pré incremento:

```
char x, y;              // variáveis de 8-bits sem sinal
                        //(0 e 255)
char myString[8]; // String com "PMR3406"
int i;                  // contador
...
i = 2;
x = myString[i++];// x = ?, i antes? , i depois ?
y = myString[++i];// y = ?, i antes? , i depois ?
```

x = 'R',
i antes = 2,
i depois = 3

y = '4',
i antes = 3,
i depois = 4

Operadores

- Atribuição: `op= ;`

no qual `op` pode ser `{ + - * / % << >> & | ^ };`

`expr1 op= expr2` é equivalente a

`expr1 = (expr1) op (expr2)`

- Exemplos
  - `i = 2;` `// ok`
  - `i += 2;` `// i = i + 2`
  - `x *= y + 1;` `// x = x* (y + 1) e não x = x* y +1`

Sua vez!!!

Você foi contratado para escrever uma parte do programa que rodará nos caixas eletrônicos do BANUSP. A parte do programa que ficou sob sua responsabiliade é a que expressa o valor de um saque em termos de cédulas de R$ 100,00, R$ 50,00, R$ 20,00, R$ 10,00, R$ 5,00, R$ 2,00 e R$ 1,00. Sua tarefa é escrever um programa em C que leia uma sequência de valores inteiros e positivos, e expresse cada valor como uma série de cédulas.

```
Saque de 394 reais:
        3 notas de 100 reais
        1 nota  de  50 reais
        2 notas de  20 reais
        2 notas de   2 reais
```

Para ler uma valor inteiro digitado pelo usuário use:

```
scanf("%d",&nomeDaVariavel);
```

MAC0115 – Introdução à Computação para Ciências Exatas e Tecnologia

INSTITUTO DE FÍSICA — TURMA 22 — SEGUNDO SEMESTRE DE 2011