Laboratório de Fundamentos em TIC

Arrays e *Strings* em C

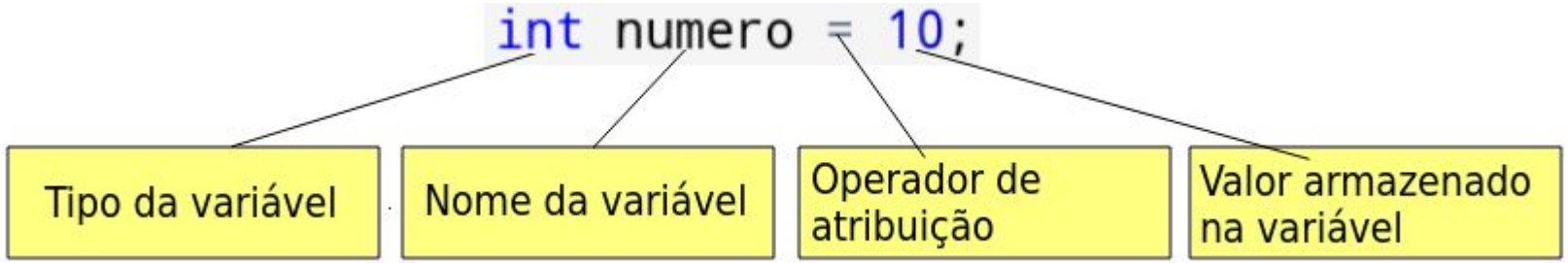
Prof. Gabriel Resende Machado

gabrielmachado@unifeso.edu.br

https://www.linkedin.com/in/machadogabriel

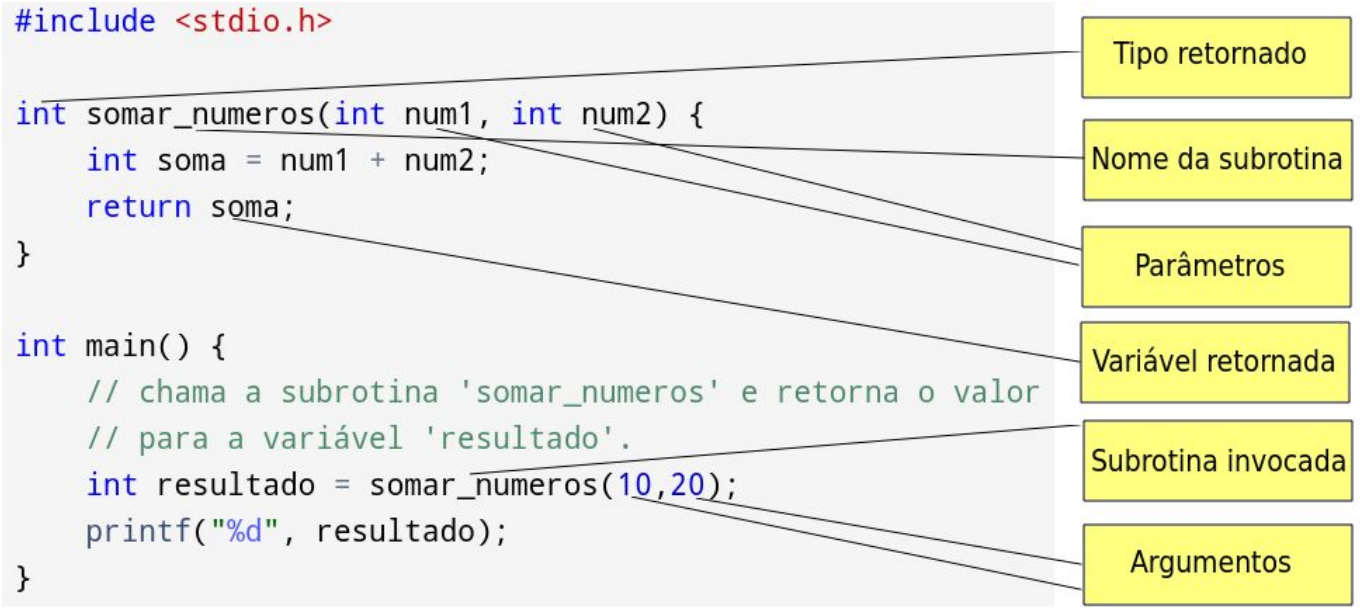
https://github.com/UNIFESO-Gabriel/fundamentos-em-tic

Relembrando: Variáveis em C



- Há variáveis especiais que não podem ter seus valores modificados, as constantes (**const**);
- **int** declara o tipo da variável: inteiro;
  - Há outros tipos, como **float, double, char, bool, short, long, unsigned**, etc.
- **int** também é conhecida como **palavra reservada**. Há 32 palavras reservadas em C (https://shorturl.at/flBLT).
- Nomes de variáveis não podem ter (i) espaços, (ii) começarem com números e (iii) pertencerem ao conjunto das palavras reservadas;

Relembrando: Subrotinas em C



- Subrotinas podem ser conhecidas também como (i) procedimentos, (ii) funções ou (iii) métodos;
- Quando uma subrotina não retorna um valor em C, seu tipo retornado é definido como **void**;
- Argumentos podem ser passados **por valor** ou **por referência**.

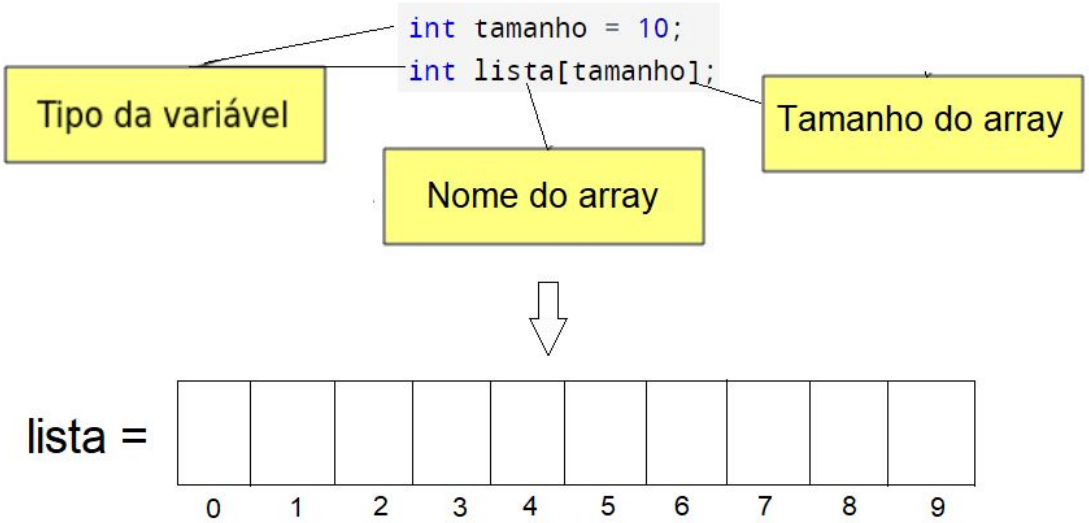
Tipos de dados em C

Tipo de Dados	Descrição	Intervalo de Valores	String de Formatação (printf / scanf)
char	Armazena caracteres.	-128 a 127 ou 0 a 255	%c / %c
signed char	Armazena caracteres com sinal.	-128 a 127	%hhd / %hhd
unsigned char	Armazena caracteres sem sinal.	0 a 255	%hhu / %hhu
int	Armazena números inteiros.	-32.768 a 32.767 ou -2.147.483.648 a 2.147.483.647	%d / %d
signed int	Sinônimo de int.	-32.768 a 32.767 ou -2.147.483.648 a 2.147.483.647	%d / %d
unsigned int	Armazena números inteiros sem sinal.	0 a 65.535 ou 0 a 4.294.967.295	%u / %u
short int (short)	Armazena inteiros curtos.	-32.768 a 32.767	%hd / %hd
signed short int (signed short)	Sinônimo de short int.	-32.768 a 32.767	%hd / %hd
unsigned short int (unsigned short)	Armazena inteiros curtos sem sinal.	0 a 65.535	%hu / %hu
long int (long)	Armazena inteiros longos.	-2.147.483.648 a 2.147.483.647	%ld / %ld
signed long int (signed long)	Sinônimo de long int.	-2.147.483.648 a 2.147.483.647	%ld / %ld
unsigned long int (unsigned long)	Armazena inteiros longos sem sinal.	0 a 4.294.967.295	%lu / %lu
long long int (long long)	Armazena inteiros longos longos.	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	%lld / %lld
signed long long int (signed long long)	Sinônimo de long long int.	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	%lld / %lld
unsigned long long int (unsigned long long)	Armazena inteiros longos longos sem sinal.	0 a 18.446.744.073.709.551.615	%llu / %llu
float	Armazena números de ponto flutuante de precisão simples.	1.2E-38 a 3.4E+38 (6 casas decimais)	%f / %f
double	Armazena números de ponto flutuante de precisão dupla.	2.3E-308 a 1.7E+308 (15 casas decimais)	%lf / %lf
long double	Armazena números de ponto flutuante de precisão estendida.	Depende da implementação específica	%Lf / %Lf
void	Indica a ausência de tipo.	N/A	N/A
_Bool	Armazena valores booleanos (verdadeiro ou falso).	0 ou 1	%d / %d
enum	Define um conjunto de constantes inteiras nomeadas.	Depende da enumeração definida	%d / %d
struct	Define um tipo de dado composto que pode armazenar diferentes tipos de dados.	N/A	N/A

Operadores em C

Operador	Descrição	Propósito
Operadores Aritméticos	Adição (+) Subtração (-) Multiplicação (*) Divisão (/) Módulo (%)	Adiciona dois operandos. Subtrai o segundo operando do primeiro. Multiplica dois operandos. Divide o numerador pelo denominador. Retorna o resto da divisão.
Operadores Booleanos	Igual a (==) Diferente de (!=) Maior que (>) Menor que (<) Maior ou igual a (>=) Menor ou igual a (<=) E lógico (&&) OU lógico (||) NÃO lógico (!)	Verifica se dois operandos são iguais. Verifica se dois operandos são diferentes. Verifica se o operando da esquerda é maior que o da direita. Verifica se o operando da esquerda é menor que o da direita. Verifica se o operando da esquerda é maior ou igual ao da direita. Verifica se o operando da esquerda é menor ou igual ao da direita. Retorna verdadeiro se ambos os operandos forem verdadeiros. Retorna verdadeiro se pelo menos um dos operandos for verdadeiro. Retorna verdadeiro se o operando for falso e vice-versa.
Operadores de Atribuição	Atribuição (=) Atribuição composta (+=, -=, *=, /=, %=)	Atribui o valor da expressão à direita ao operando da esquerda. Realiza a operação aritmética nos dois operandos e atribui o resultado ao operando da esquerda.
Operadores de Incremento e Decremento	Incremento (++) Decremento (--)	Aumenta o valor do operando em 1. Diminui o valor do operando em 1.
Operadores Bit a Bit	E bit a bit (&) OU bit a bit (|) XOR bit a bit (^) NÃO bit a bit (~) Deslocamento à esquerda (<<) Deslocamento à direita (>>)	Realiza a operação AND bit a bit. Realiza a operação OR bit a bit. Realiza a operação XOR bit a bit. Realiza a operação complemento bit a bit (complemento de um). Desloca os bits do operando da esquerda para a esquerda pelo número de posições especificado pelo operando da direita. Desloca os bits do operando da esquerda para a direita pelo número de posições especificado pelo operando da direita.
Operador Condicional (Operador Ternário)	Operador Condicional (?)	Avalia uma condição e retorna um dos dois valores dependendo se a condição é verdadeira ou falsa.
Operador de Vírgula	Operador de Vírgula (,)	Avalia duas expressões e retorna o resultado da segunda expressão.
Operadores de Memória	Asterisco (*) E Comercial (&)	Faz a desferenciação de um endereço de memória, i.e. acessa o valor armazenado nesse endereço. Retorna o endereço de memória de uma variável.

Arrays em C



- Um *array* em C é um agrupamento de variáveis do mesmo tipo (estrutura homogênea);
- Arrays utilizam colchetes em sua declaração e acesso aos elementos;
- Sempre é preciso declarar o tamanho dos vetores;
- O primeiro índice em C sempre é 0 (*zero-based*).

Passagem de Arrays para Subrotinas

```c
#include <stdio.h>

void exibe_array(int valores[], int tamanho) {

    for (int i = 0; i < tamanho; i++) {
        printf("%d ", valores[i]);
    }
}

int main() {

    int notas[5] = {70, 80, 90, 100, 60};
    exibe_array(notas, sizeof(notas) / sizeof(int));

    return 0;
}
```

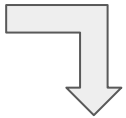
Arrays Multidimensionais

```c
void preenche_matriz(int matriz[][3], int linhas, int colunas)
{
    int num = 0;
    for (int i = 0; i < linhas; i++) {
        for (int j = 0; j < colunas; j++) {
            matriz[i][j] = ++num;
        }
    }
}

void exibe_matriz(int matriz[][3], int linhas, int colunas)
{
    for (int i = 0; i < linhas; i++) {
        if (i > 0)
            printf("\n");
        for (int j = 0; j < colunas; j++) {
            printf("%d ", matriz[i][j]);
        }
    }
}
```

```c
int main() {

    int matriz[3][3] = {0};
    preenche_matriz(matriz, 3, 3);
    exibe_matriz(matriz, 3, 3);
    return 0;
}
```

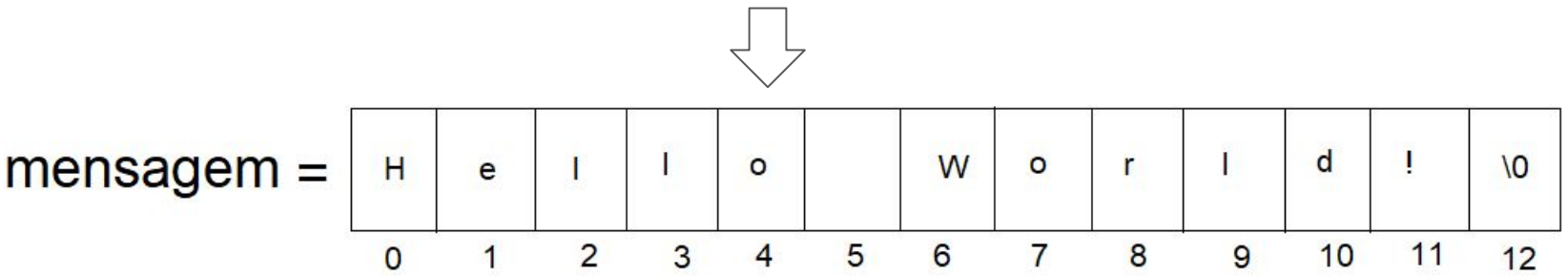


```
1 2 3
4 5 6
7 8 9
```

- Há a possibilidade de criar *arrays* dinamicamente a partir da função *malloc;*
- Esse tema será visto com mais detalhes quando falarmos de **ponteiros**.

Strings em C

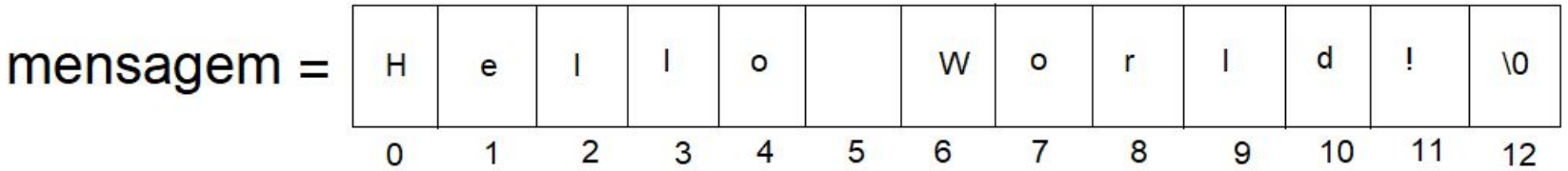
```c
char mensagem[13] = "Hello World!";
printf("%s", mensagem);
return 0;
```



- Uma *string* é representada por um *array* de caracteres;
- São utilizadas para armazenar e imprimir texto;
- Toda *string* em C termina com o caractere nulo \0;
- Uma *string* pode ser impressa no terminal via a *string* de formatação %s.

Strings em C

```c
char mensagem[13] = "Hello World!";
printf("%s", mensagem);
return 0;
```



- Uma *string* é representada por um *array* de caracteres;
- São utilizadas para armazenar e imprimir texto;
- Toda *string* em C termina com o caractere nulo \0;
- Strings podem ser lidas via *scanf*. Porém, é preferível o uso da função *fgets;*
- Uma *string* pode ser impressa no terminal via a *string* de formatação %s.

Strings em C

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

#define MAX_LENGTH 100

int main() {
    char msg[MAX_LENGTH];

    printf("Digite um texto com espacos: ");
    fgets(msg, sizeof(msg), stdin);
    printf("%s\n", msg);

}
```

- *Strings* podem ser lidas via *scanf*. Porém, é preferível o uso da função *fgets*.

Strings em C

- Há diversas funções úteis na biblioteca <*string.h*>:

```
#include <string.h>
int main()
{
      char test[100];
      char test2[]= "World!\n";
      strcpy(test,"Hello"); /*copia*/
      strcat(test,test2); /* concatenacao*/
      if (strcmp(test,"david") == 0)
            printf ("Test é o mesmo que David\n");
      printf ("comprimento de test é is %d\n",
      strlen (test));
}
```

Strings em C

- Se a *string* contém um número, podemos usar as funções **atoi** e **atof** para convertê-la em um número;
- Estas funções estão declaradas na biblioteca ***<stdlib.h>*** e **retornam 0 em caso de erro**.

```
char numberstring[]= "3.14";
int i;
double pi;
pi= atof (numberstring);
i= atoi ("12");
```

Antes de tudo...

- Refazer todas as listas de exercícios da Aula 02 em linguagem C;
- Resolver problemas do portal *Beecrowd:* https://judge.beecrowd.com.

Laboratório de Fundamentos em TIC

Arrays e *Strings* em C

Prof. Gabriel Resende Machado

gabrielmachado@unifeso.edu.br

https://www.linkedin.com/in/machadogabriel

https://github.com/UNIFESO-Gabriel/fundamentos-em-tic