# Estruturas definidas pelo programador

Prof. Bruno Travençolo

# Variáveis

- As variáveis vistas até agora eram:
  - simples: definidas por tipos **int**, **float**, **double** e **char**;
  - compostas homogêneas (ou seja, do mesmo tipo): definidas por **array**.

- No entanto, a linguagem C permite que se criem novas estruturas a partir dos tipos básicos.
  - Struct

- Mas antes, vamos rever alguns conceitos sobre a memória alocada por um programa

# Operador **sizeof**

- Traduzindo: *sizeof*: *size* (tamanho) *of* (de)
  - Retorna o tamanho em bytes ocupado por objetos ou tipos
  - Exemplo de uso
  - ```
    printf("\nTamanho em bytes de um char: %u", sizeof(char));
    ```
    - Retorna 1, pois o tipo char tem 1 byte
  - ```
    printf("\nTamanho em bytes de um char: %u", sizeof char);
    ```
    - Também funciona sem o parênteses

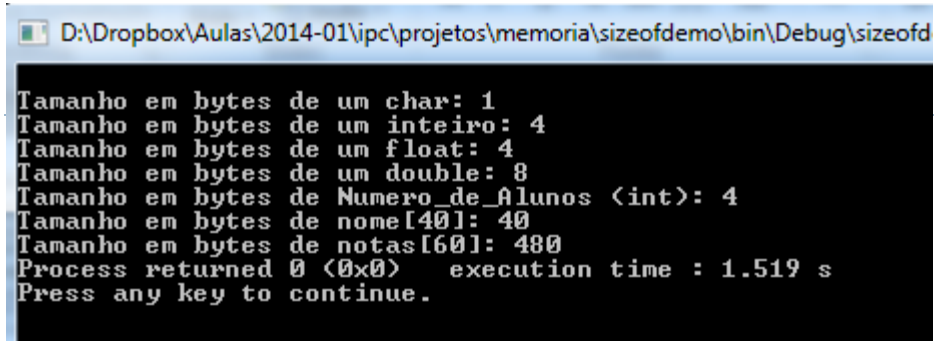
Retorna um tipo size_t, normalmente `unsigned int`, por isso o %u ao invés de %d

`unsigned int` - é um número inteiro sem sinal negativo

# Operador **sizeof**

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

int main()
{
    // descobrindo o tamanho ocupado por diferentes tipos de dados
    printf("\nTamanho em bytes de um char: %u", sizeof(char));
    printf("\nTamanho em bytes de um inteiro: %u", sizeof(int));
    printf("\nTamanho em bytes de um float: %u", sizeof(float));
    printf("\nTamanho em bytes de um double: %u", sizeof(double));

    // descobrindo o tamanho ocupado por uma variável
    int Numero_de_Alunos;
    printf("\nTamanho em bytes de Numero_de_Alunos (int): %u", sizeof Numero_de_Alunos );

    // também é possível obter o tamanho de vetores
    char nome[40];
    printf("\nTamanho em bytes de nome[40]: %u", sizeof(nome));

    double notas[60];
    printf("\nTamanho em bytes de notas[60]: %u", sizeof notas );

    return 0;
}
```

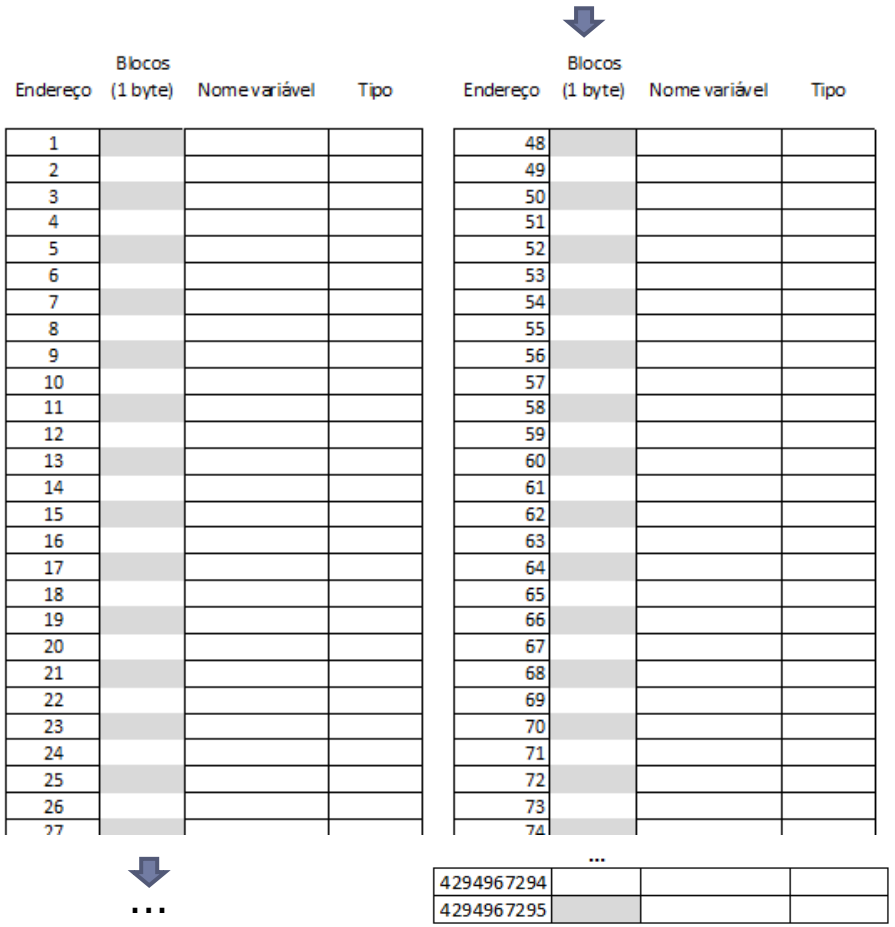
# Operador **sizeof**




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

int main()
{
    // descobrindo o tamanho ocupado por diferentes tipos de dados
    printf("\nTamanho em bytes de um char: %u", sizeof(char));
    printf("\nTamanho em bytes de um inteiro: %u", sizeof(int));
    printf("\nTamanho em bytes de um float: %u", sizeof(float));
    printf("\nTamanho em bytes de um double: %u", sizeof(double));

    // descobrindo o tamanho ocupado por uma variável
    int Numero_de_Alunos;
    printf("\nTamanho em bytes de Numero_de_Alunos (int): %u", sizeof Numero_de_Alunos );

    // também é possível obter o tamanho de vetores
    char nome[40];
    printf("\nTamanho em bytes de nome[40]: %u", sizeof(nome));

    double notas[60];
    printf("\nTamanho em bytes de notas[60]: %u", sizeof notas );

    return 0;
}
```

# Memória

- Podemos pensar na memória como uma sequência linear de bytes, sendo que cada byte possui um endereço.
- A memória é limitada
- O Sistema operacional (SO) gerencia a memória
- Vale observar que esse esquema é usado para entender alocação. A que ocorre de fato depende de vários fatores: SO, compilador, otimização, alinhamento, etc.
- Lembre também da arquitetura de Von Neumann (instruções e dados compartilham o mesmo endereçamento de memória)




Sistema 32 bits

# Exercício

- Indique, usando o mapa de memória do slide anterior, um possível estado da memória ao fim da execução do seguinte programa.
  - Indique quantos bytes as variáveis declaradas ocupam
  - Continue o programa para mostrar a soma da quantidade de memória ocupada (use a variável tamanho_total)

```
int  idade;
char nome[10] = "Maria";
double peso, altura;
int casada;
float grau_miopia[2];
unsigned int tamanho_total;

altura = 1.65;
peso = 70;
casada = 0; // false
grau_miopia[0] = 2.75; // olho esquerdo
grau_miopia[1] = 3; // olho direito
```

# Exemplos de alocação

- `char nome[10] = "Maria"`

| Endereço | Blocos (1 byte) | Nome variável | Tipo |
|---|---|---|---|
| 0 / NULL | indefinido | ---- | ---- |
| 1 | 'M' | nome[0] | char |
| 2 | 'a' | nome[1] | char |
| 3 | 'r' | nome[2] | char |
| 4 | 'i' | nome[3] | char |
| 5 | 'a' | nome[4] | char |
| 6 | '\0' | nome[5] | char |
| 7 | lx | nome[6] | char |
| 8 | lx | nome[7] | char |
| 9 | lx | nome[8] | char |
| 10 | lx | nome[9] | char |
| 11 | | | |

obs

forma correta: são alocados 10 bytes de memória do tipo char (o tipo char ocupa 1 byte)

*** obs: na verdade as posições de 7 a 10 são inicializadas com \0, mas esse comportamento não é padrão em comandos como gets e strcpy

# Exemplos de alocação

- `char nome[10] = "Maria"` – Forma errada

| | | | |
|---|---|---|---|
| 11 | | | |
| 12 | 'M' | nome[10] | |
| 13 | 'a' | | |
| 14 | 'r' | | |
| 15 | 'i' | | |
| 16 | 'a' | | |
| 17 | '\0' | | |
| 18 | lx | | |
| 19 | lx | | |
| 20 | lx | | |
| 21 | lx | | |
| 22 | | | |
| 23 | | | |

erro: nome[10] representa a décima primeira posição do vetor nome, posição esta que não existe! (ele pegaria, neste caso, a posição 22)

# Exemplos de alocação

- `char nome[10] = "Maria"` – Forma errada

| | | | |
|---|---|---|---|
| 22 | | | |
| 23 | | | |
| 24 | 'M' | nome[0] | char |
| 25 | 'a' | nome[1] | char |
| 26 | 'r' | nome[2] | char |
| 27 | 'i' | nome[3] | char |
| 28 | 'a' | nome[4] | char |
| 29 | '\0' | nome[5] | char |
| 30 | | nome[6] | char |
| 31 | | nome[7] | char |
| 32 | | nome[8] | char |
| 33 | | nome[9] | char |
| 34 | | | |

erro: faltou colocar o lixo (lx) de nome[6] até nome[9]. Mesmo que "Maria" não ocupe todo o vetor, ele é alocado. Além disso, como não houve inicialização em parte do vetor, essa parte é lixo

# Exemplos de alocação

- `char nome[10] = "Maria"` – Forma errada

| Endereço | Blocos (1 byte) | Nome variável | Tipo |
|---|---|---|---|
| 47 | | | |
| 48 | 'M' | nome[0] | char |
| 49 | 'a' | nome[1] | char |
| 50 | 'r' | nome[2] | char |
| 51 | 'i' | nome[3] | char |
| 52 | 'a' | nome[4] | char |
| 53 | '\0' | nome[5] | char |
| 54 | lx | nome[6] | char |
| 55 | | nome[7] | char |
| 56 | | nome[8] | char |
| 57 | | nome[9] | char |
| 58 | | | |
| 59 | | | |

erro: tem que ser 'lx' para as quarto posições, pois são quatro 'char' distintos

# Exemplos de alocação

- `char nome[10] = "Maria"` – Forma errada

| | | | |
|---|---|---|---|
| 58 | | | |
| 59 | | | |
| 60 | 'M' | nome[1] | char |
| 61 | 'a' | nome[2] | char |
| 62 | 'r' | nome[3] | char |
| 63 | 'i' | nome[4] | char |
| 64 | 'a' | nome[5] | char |
| 65 | '\0' | nome[6] | char |
| 66 | | nome[7] | char |
| 67 | lx | nome[8] | char |
| 68 | | nome[9] | char |
| 69 | | nome[10] | char |
| 70 | | | |

erro: em C vetor sempre começa na posição zero (0)

# Exemplos de alocação

- `char nome[10] = "Maria"` – Forma errada

| | | | |
|---|---|---|---|
| 70 | | | |
| 71 | 'M' | nome[0] | char |
| 72 | 'a' | nome[1] | char |
| 73 | 'r' | nome[2] | char |
| 74 | 'i' | nome[3] | char |
| 75 | 'a' | nome[4] | char |
| 76 | lx | nome[5] | char |
| 77 | lx | nome[6] | char |
| 78 | lx | nome[7] | char |
| 79 | lx | nome[8] | char |
| 80 | lx | nome[9] | char |
| 81 | | | |

erro: faltou colocar o '\0' de fim de string

# Exemplos de alocação

- `double peso = 10;`

| Endereço | Blocos (1 byte) | Nome variável | Tipo |
|---|---|---|---|
| 0 / NULL | indefinido | ---- | ---- |
| 1 | | | |
| 2 | 10 | peso | double |
| 3 | | | |
| 4 | | | |
| 5 | | | |
| 6 | | | |
| 7 | | | |
| 8 | | | |
| 9 | | | |
| 10 | | | |

Correto. Um tipo double ocupa 8 bytes. Assim, independente do valor atribuído a variável peso (10, 20, 1milhão), serão 8 bytes ocupados

# Exemplos de alocação

- `double peso = 10;` - Forma errada

| | | | |
|---|---|---|---|
| 10 | | | |
| 11 | | | |
| 12 | 10 | peso | double |
| 13 | lx | | |
| 14 | lx | | |
| 15 | lx | | |
| 16 | lx | | |
| 17 | lx | | |
| 18 | lx | | |
| 19 | lx | | |
| 20 | | | |
| 21 | | | |

Erro. Todos os 8 bytes pertencem à variável. Uma vez atribuído o valor, ele ocupa todos os bits, e não só o primeiro, independente do valor (10, 20 ou 1 milhão)

# Exemplos de alocação

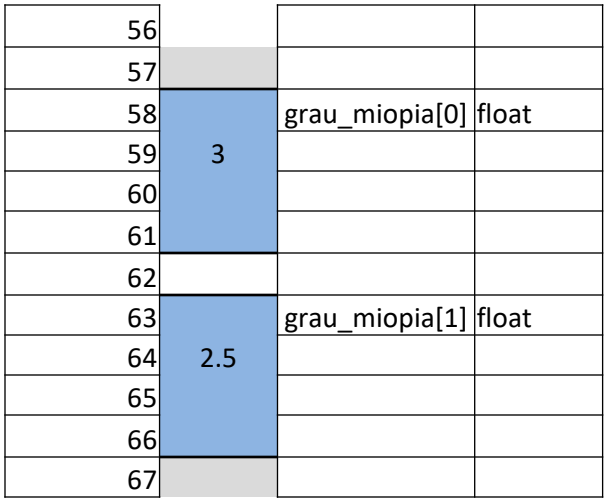
- `float grau_miopia[2];`
- `grau_miopia[0] = 3; grau_miopia[1]=2.5;`

| Endereço | Blocos (1 byte) | Nome variável | Tipo |
|---|---|---|---|
| 47 | 3 | grau_miopia[0] | float |
| 48 | | | |
| 49 | | | |
| 50 | | | |
| 51 | 2.5 | grau_miopia[1] | float |
| 52 | | | |
| 53 | | | |
| 54 | | | |
| 55 | | | |
| 56 | | | |

Correto. Cada elemento do vetor ocupa 4 bytes. O endereço de grau_miopia[0] é 47 e o de grau_miopia[1] é 51

# Exemplos de alocação

- `float grau_miopia[2]; - Forma errada`
- `grau_miopia[0] = 3; grau_miopia[1]=2.5;`




Erro. A alocação de dados de vetores é contínua, não há espaço entre um elemento e outro

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

int main()
{
    int  idade;
    char nome[10] = "Maria";
    double peso, altura;
    int casada;
    float grau_miopia[2];
    unsigned int tamanho_total;

    altura = 1.65;
    peso = 70;
    casada = 0; // false
    grau_miopia[0] = 2.75; // olho esquerdo
    grau_miopia[1] = 3; // olho direito

    // obs: o símbolo \ serve para continuar um comando em
    // uma outra linha.
    tamanho_total = sizeof(nome) + sizeof(altura) + sizeof(peso)+  \
                    sizeof(casada)+sizeof(grau_miopia)+sizeof(idade) + \
                    sizeof(tamanho_total);
    printf("\n Tamanho em bytes ocupado: %u", tamanho_total);


    return 0;
}
```