

Faculdade de Computação - FACOM

Bacharelado em Sistemas de Informação

FACOM32201 - Algoritmos e Programação II

Prof. Thiago Pirola Ribeiro

Funcionamento da Memória

- 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.
- Existem outros tipos de variáveis que apontam para posições de memória
- Mas antes, vamos rever alguns conceitos sobre a memória alocada por um programa

Operador sizeof

- Traduzindo: `sizeof`: *size* (tamanho) e *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
- 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

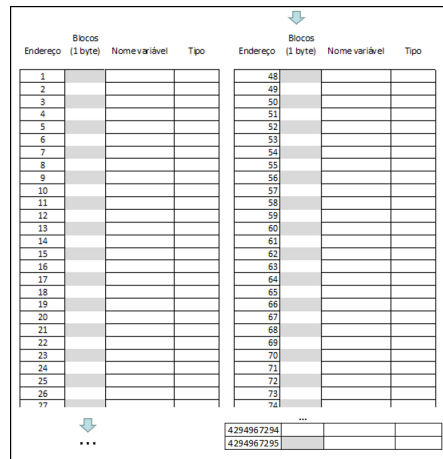
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main() {
4     // descobrindo o tamanho ocupado por diferentes tipos de dados
5     printf("\nTam. em bytes de um char: %u", sizeof(char));
6     printf("\nTam. em bytes de um inteiro: %u", sizeof(int));
7     printf("\nTam. em bytes de um float: %u", sizeof(float));
8     printf("\nTam. em bytes de um double: %u", sizeof(double));
9
10    // descobrindo o tamanho ocupado por uma variável
11    int Numero_de_Alunos;
12    printf("\nTam. bytes Numero_de_Alunos: %u", sizeof Numero_de_Alunos);
13
14    // também é possível obter o tamanho de vetores
15    char nome[40];
16    printf("\nTam. em bytes de nome[40]: %u", sizeof(nome));
17    double notas[60];
18    printf("\nTam. em bytes de notas[60]: %u", sizeof notas );
19
20    return 0;
21 }
```

```
Tamanho em bytes de um char: 1
Tamanho em bytes de um inteiro: 4
Tamanho em bytes de um float: 4
Tamanho em bytes de um double: 8
Tamanho em bytes de Numero_de_Alunos (int): 4
Tamanho em bytes de nome[40]: 40
Tamanho em bytes de notas[60]: 480
Process finished with exit code 0
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main() {
4      // descobrindo o tamanho ocupado por diferentes tipos de dados
5      printf("\nTam. em bytes de um char: %u", sizeof(char));
6      printf("\nTam. em bytes de um inteiro: %u", sizeof(int));
7      printf("\nTam. em bytes de um float: %u", sizeof(float));
8      printf("\nTam. em bytes de um double: %u", sizeof(double));
9
10     // descobrindo o tamanho ocupado por uma variável
11     int Numero_de_Alunos;
12     printf("\nTam. bytes Numero_de_Alunos: %u", sizeof Numero_de_Alunos);
13
14     // também é possível obter o tamanho de vetores
15     char nome[40];
16     printf("\nTam. em bytes de nome[40]: %u", sizeof(nome));
17     double notas[60];
18     printf("\nTam. em bytes de notas[60]: %u", sizeof notas );
19
20     return 0;
21 }
```

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;
 - 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).



Exercício

- Usando o mapa de memória do slide anterior:
 - Indique quantos bytes as variáveis declaradas ocupam;

```
1  int  idade;  
2  char nome[10] = "Maria";  
3  double peso, altura;  
4  int  casada;  
5  float grau_miopia[2];  
6  unsigned int tamanho_total;  
  
7  
8  altura = 1.65;  
9  peso = 70;  
10 casada = 0; // false  
11 grau_miopia[0] = 2.75; // olho esquerdo  
12 grau_miopia[1] = 3; // olho direito  
13
```


Exemplos de alocação

```
1 char nome[10] = "Maria";
```

Blocos			
Endereço	(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:** 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

```
1 double peso = 10;
```

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

Exemplos de alocação

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

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int idade;
7      char nome[10] = "Maria";
8      double peso, altura;
9      int casada;
10     float grau_miopia[2];
11     unsigned int tamanho_total;
12
13     altura = 1.65;
14     peso = 70;
15     casada = 0; // false
16     grau_miopia[0] = 2.75; // olho esquerdo
17     grau_miopia[1] = 3; // olho direito
18
19     // obs: o símbolo \ serve para continuar um comando em
20     // uma outra linha.
21     tamanho_total = sizeof(nome) + sizeof(altura) + sizeof(peso)+ \
22     sizeof(casada)+sizeof(grau_miopia)+sizeof(idade) + \
23     sizeof(tamanho_total);
24     printf("\n Tamanho em bytes ocupado: %u", tamanho_total);
25
26     return 0;
27 }
28

```

Tamanho em bytes ocupado: 46
Process finished with exit code 0

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int idade;
7      char nome[10] = "Maria";
8      double peso, altura;
9      int casada;
10     float grau_miopia[2];
11     unsigned int tamanho_total;
12
13     altura = 1.65;
14     peso = 70;
15     casada = 0; // false
16     grau_miopia[0] = 2.75; // olho esquerdo
17     grau_miopia[1] = 3; // olho direito
18
19     // obs: o símbolo \ serve para continuar um comando em
20     // uma outra linha.
21     tamanho_total = sizeof(nome) + sizeof(altura) + sizeof(peso)+ \
22     sizeof(casada)+sizeof(grau_miopia)+sizeof(idade) + \
23     sizeof(tamanho_total);
24     printf("\n Tamanho em bytes ocupado: %u", tamanho_total);
25
26     return 0;
27 }
28
```

Observações sobre a Memória

Endereço	Blocos	Tamanho
1		(1 byte)
2		(1 byte)
3		(1 byte)
4		(1 byte)
5		(1 byte)
6		(1 byte)
7		(1 byte)
8		(1 byte)
9		(1 byte)
10		(1 byte)
11		(1 byte)
12		(1 byte)
13		(1 byte)
14		(1 byte)
....		

Observações sobre a Memória

```
char c;
```

Endereço	Blocos	Variável	tipo
1			
2			
3		c	char
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
....			

Observações sobre a Memória

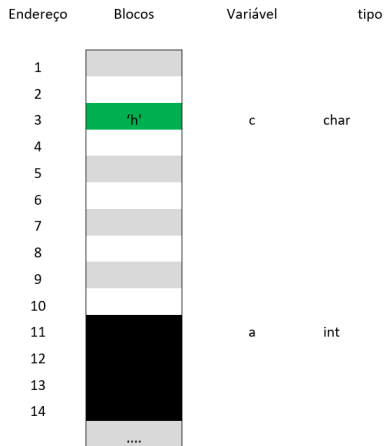
```
char c;  
c = 'h';
```

Endereço	Blocos	Variável	tipo
1			
2			
3	'h'	c	char
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			

Observações sobre a Memória

```
char c;  
c = 'h';
```

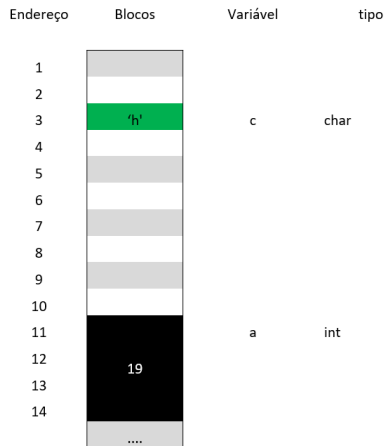
```
int a;
```



Observações sobre a Memória

```
char c;  
c = 'h';
```

```
int a;  
a = 19;
```

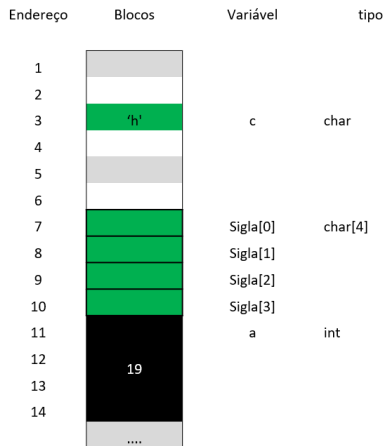


Observações sobre a Memória

```
char c;  
c = 'h';
```

```
int a;  
a = 19;
```

```
char Sigla[4];
```

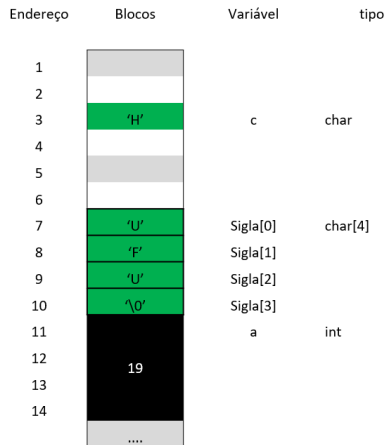


Observações sobre a Memória

```
char c;  
c = 'h';
```

```
int a;  
a = 19;
```

```
char Sigla[4];  
Sigla[0] = 'U';  
Sigla[1] = 'F';  
Sigla[2] = 'U';  
Sigla[3] = '\0';
```



Endereço de variáveis

- Para descobrir o endereço de uma variável em C, use o operador &

```
int a = 5;  
int b = 10;  
char c[5] = {'A', 'b', '8', 'd', '|'};
```

```
printf("Valor de a: %d \n", a);  
printf("Endereco de a: %u \n", &a);  
printf("Endereco de a (em hexadecimal): %p \n\n", &a);
```

```
printf("Valor de b: %d \n", b);  
printf("Endereco de b: %u \n", &b);  
printf("Endereco de b (em hexadecimal): %p \n\n", &b);
```

```
for (int i=0; i < 5; i++){  
    printf("Valor de c[%d]: %c \n", i, c[i]);  
    printf("Endereco de c[%d]: %u \n", i, &c[i]);  
    printf("Endereco de c[%d] (em hexadecimal): %p \n\n", i, &c[i]);  
}
```

Endereço de variáveis

Valor de a: 5
Endereço de a: 3690985512
Endereço de a (em hexadecimal): 0000001CDBFFF828

Valor de b: 10
Endereço de b: 3690985508
Endereço de b (em hexadecimal): 0000001CDBFFF824

Valor de c[0]: A
Endereço de c[0]: 3690985503
Endereço de c[0] (em hexadecimal): 0000001CDBFFF81F

Valor de c[1]: b
Endereço de c[1]: 3690985504
Endereço de c[1] (em hexadecimal): 0000001CDBFFF820

Valor de c[2]: 8
Endereço de c[2]: 3690985505
Endereço de c[2] (em hexadecimal): 0000001CDBFFF821

O comando scanf()

Comando de entrada

- Em C, o comando que permite ler dados da entrada padrão (no caso o teclado) é o `scanf()`
- Sintaxe: `scanf("format",&name1,...)`
 - `format` – especificador de formato da entrada que será lida
 - `&name1, &name2, ...` – endereços das variáveis que receberão os valores lidos

Comando de entrada

- Temos, igual ao comando printf, que especificar o tipo (formato) do dado que será lido
 - `scanf("tipo de entrada", lista de variáveis)`
- Alguns “tipos de entrada”
 - `%c` – leitura de um caractere
 - `%d` – leitura de números inteiros
 - `%f` – leitura de número reais
 - `%s` – leitura de vários caracteres

Comando scanf() - Exemplo

```
1 // declaração das variáveis
2 float peso;
3 float altura;
4 float IMC;
5
6 // Obtendo os dados do usuário
7 printf("Informe o peso: ");
8 scanf("%f", &peso);
9 printf("Informe a altura: ");
10 scanf("%f", &altura);
11
12 // calculando o IMC e mostrando o resultado
13 IMC = peso / (altura*altura);
14 printf("Peso: %f, Altura: %f, IMC: %f", peso, altura, IMC);
```

Comando scanf() - Exemplo

- ```
// leia um valor real (float ou double) e
// armazene no endereço reservado para a variável peso

scanf ("%f", &peso);
```
- O símbolo & indica qual é o endereço da variável que vai receber os dados lidos
  - peso – variável peso
  - &peso – endereço da variável peso

```
1 int k;
2 unsigned int endereco_de_k;
3
4 // inicializando k
5 k = 10;
6 printf("\n Valor da variavel 'k': %d \n",k);
7
8 // obtendo o endereço da variável 'k' // usando o operador &
9 endereco_de_k = &k;
10
11 printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
12 printf("\n Endereco da variavel 'k': %u \n",&k);
13 printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
14
15 // sabemos que o scanf pede um endereço de memória
16 // o que acontece se passarmos o endereço da variável k?
17 printf("\n Digite o valor novo valor, a ser armazenado no endereço da
variavel k: ");
18 scanf("%d",endereco_de_k);
```

```
1 // mostrando o novo valor de 'k'
2 printf("\n\n Valor da variavel k, apos scanf de 'endereco_de_k': %d \n
",k);
3
4 // mostrando o endereço de 'k' que deve permanecer o mesmo de antes
5 printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
6 printf("\n Endereco da variavel 'k': %u \n",&k);
7 printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
```

```

1 int k;
2 unsigned int endereco_de_k;
3
4 // inicializando k
5 k = 10;
6 printf("\n Valor da variavel 'k': %d \n",k);
7
8 // obtendo o endereço da variável 'k' // usando o operador &
9 endereco_de_k = &k;
10
11 printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
12 printf("\n Endereco da variavel 'k': %u \n",&k);
13 printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
14
15 // sabemos que o scanf pede um endereço de memória
16 // o que acontece se passarmos o endereço da
17 // variável k?
18 printf("\n Digite o valor novo valor, a ser armazenado no endereço da variavel k: ");
19 // OBSERVE que não estamos usando & no scanf! Isso porque já temos o endereço
20 scanf("%d",endereco_de_k);
21
22 // mostrando o novo valor de 'k'
23 printf("\n\n Valor da variavel k, apos scanf de 'endereco_de_k': %d \n",k);
24
25 // mostrando o endereço de 'k' que deve permanecer o mesmo de antes
26 printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
27 printf("\n Endereco da variavel 'k': %u \n",&k);
28 printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);

```

| Endereço | Blocos<br>(1 byte) | Nome variável        | Tipo         |
|----------|--------------------|----------------------|--------------|
| 0 / NULL | indefinido         | ---                  | ---          |
| 1        | lx                 | <b>k</b>             | int          |
| 2        |                    |                      |              |
| 3        |                    |                      |              |
| 4        |                    |                      |              |
| 5        |                    |                      |              |
| 6        | lx                 | <b>endereco_de_k</b> | unsigned int |
| 7        |                    |                      |              |
| 8        |                    |                      |              |
| 9        |                    |                      |              |
| 10       |                    |                      |              |
| 11       |                    |                      |              |

```

1 int k;
2 unsigned int endereco_de_k;

```

|   |    |               |              |
|---|----|---------------|--------------|
| 1 |    | k             | int          |
| 2 | 10 |               |              |
| 3 |    |               |              |
| 4 |    |               |              |
| 5 |    |               |              |
| 6 |    | endereco_de_k | unsigned int |
| 7 | lx |               |              |
| 8 |    |               |              |
| 9 |    |               |              |

```

1 int k;
2 unsigned int endereco_de_k;
3
4 // inicializando k
5 k = 10;
6 printf("\n Valor da variavel 'k': %d \n",k);

```



|   |    |                      |              |
|---|----|----------------------|--------------|
| 1 |    | <b>k</b>             | int          |
| 2 | 10 |                      |              |
| 3 |    |                      |              |
| 4 |    |                      |              |
| 5 |    |                      |              |
| 6 |    | <b>endereco_de_k</b> | unsigned int |
| 7 | 1  |                      |              |
| 8 |    |                      |              |
| 9 |    |                      |              |

```

1 int k;
2 unsigned int endereco_de_k;
3
4 // inicializando k
5 k = 10;
6 printf("\n Valor da variavel 'k': %d \n",k);
7
8 // obtendo o endereço da variável 'k' // usando o operador &
9 endereco_de_k = &k;

```

|   |    |               |              |
|---|----|---------------|--------------|
| 1 |    | k             | int          |
| 2 | 10 |               |              |
| 3 |    |               |              |
| 4 |    |               |              |
| 5 |    |               |              |
| 6 |    | endereco_de_k | unsigned int |
| 7 | 1  |               |              |
| 8 |    |               |              |
| 9 |    |               |              |

```

1 int k;
2 unsigned int endereco_de_k;
3
4 // inicializando k
5 k = 10;
6 printf("\n Valor da variavel 'k': %d \n",k);
7
8 // obtendo o endereço da variável 'k' // usando o operador &
9 endereco_de_k = &k;
10
11 printf("\n Endereco da variavel 'k': %u \n",endereco_de_k); // Valor

```

1

|   |    |               |              |
|---|----|---------------|--------------|
| 1 |    | k             | int          |
| 2 | 10 |               |              |
| 3 |    |               |              |
| 4 |    |               |              |
| 5 |    |               |              |
| 6 |    | endereco_de_k | unsigned int |
| 7 | 1  |               |              |
| 8 |    |               |              |
| 9 |    |               |              |

```

1 int k;
2 unsigned int endereco_de_k;
3
4 // inicializando k
5 k = 10;
6 printf("\n Valor da variavel 'k': %d \n",k);
7
8 // obtendo o endereço da variável 'k' // usando o operador &
9 endereco_de_k = &k;
10
11 printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
12 printf("\n Endereco da variavel 'k': %u \n",&k); // Endereço 1

```

|   |    |               |              |
|---|----|---------------|--------------|
| 1 |    | k             | int          |
| 2 | 10 |               |              |
| 3 |    |               |              |
| 4 |    |               |              |
| 5 |    |               |              |
| 6 |    | endereco_de_k | unsigned int |
| 7 | 1  |               |              |
| 8 |    |               |              |
| 9 |    |               |              |

```

1 int k;
2 unsigned int endereco_de_k;
3
4 // inicializando k
5 k = 10;
6 printf("\n Valor da variavel 'k': %d \n",k);
7
8 // obtendo o endereço da variável 'k' // usando o operador &
9 endereco_de_k = &k;
10
11 printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
12 printf("\n Endereco da variavel 'k': %u \n",&k);
13 printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
14 // Endereço 0x1

```

|   |    |               |              |
|---|----|---------------|--------------|
| 1 |    | k             | int          |
| 2 | 10 |               |              |
| 3 |    |               |              |
| 4 |    |               |              |
| 5 |    |               |              |
| 6 |    | endereco_de_k | unsigned int |
| 7 | 1  |               |              |
| 8 |    |               |              |
| 9 |    |               |              |

```

1 ...
2 // sabemos que o scanf pede um endereço de memória
3 // o que acontece se passarmos o endereço da
4 // variável k?
5 printf("\n Digite o valor novo valor, a ser armazenado no endereço da
variavel k: ");
6
7 // OBSERVE que não estamos usando & no scanf! Isso porque já temos o
endereço
8 scanf("%d", endereco_de_k);

```

|   |    |               |              |
|---|----|---------------|--------------|
| 1 | 50 | k             | int          |
| 2 |    |               |              |
| 3 |    |               |              |
| 4 |    |               |              |
| 5 |    |               |              |
| 6 | 1  | endereco_de_k | unsigned int |
| 7 |    |               |              |
| 8 |    |               |              |
| 9 |    |               |              |

```

1 ...
2 // sabemos que o scanf pede um endereço de memória
3 // o que acontece se passarmos o endereço da
4 // variável k?
5 printf("\n Digite o valor novo valor, a ser armazenado no endereço da
variavel k: ");
6
7 // OBSERVE que não estamos usando & no scanf! Isso porque já temos o
endereço
8 scanf("%d", endereco_de_k);
9
10 // Suponha que foi digitado o valor 50

```

|   |    |               |              |
|---|----|---------------|--------------|
| 1 | 50 | k             | int          |
| 2 |    |               |              |
| 3 |    |               |              |
| 4 |    |               |              |
| 5 |    |               |              |
| 6 | 1  | endereco_de_k | unsigned int |
| 7 |    |               |              |
| 8 |    |               |              |
| 9 |    |               |              |

```

1 ...
2 // sabemos que o scanf pede um endereço de memória
3 // o que acontece se passarmos o endereço da
4 // variável k?
5 printf("\n Digite o valor novo valor, a ser armazenado no endereço da
 variavel k: ");
6
7 // OBSERVE que não estamos usando & no scanf! Isso porque já temos o
 endereço
8 scanf("%d", endereco_de_k);
9
10 // mostrando o novo valor de 'k'
11 printf("\n Valor da variavel k, apos scanf de 'endereco_de_k': %d\n", k);
 // Valor 50

```

|   |    |               |              |
|---|----|---------------|--------------|
| 1 | 50 | k             | int          |
| 2 |    |               |              |
| 3 |    |               |              |
| 4 |    |               |              |
| 5 |    |               |              |
| 6 | 1  | endereco_de_k | unsigned int |
| 7 |    |               |              |
| 8 |    |               |              |
| 9 |    |               |              |

```

1 ...
2
3 // mostrando o novo valor de 'k'
4 printf("\n\n Valor da variavel k, apos scanf de 'endereco_de_k': %d\n",k)
 ; // Valor 50
5
6 // mostrando o endereco de 'k' que deve permanecer o mesmo de antes
7 printf("\n Endereco da variavel 'k': %u \n",endereco_de_k); // Valor 1

```



|   |    |               |              |
|---|----|---------------|--------------|
| 1 | 50 | k             | int          |
| 2 |    |               |              |
| 3 |    |               |              |
| 4 |    |               |              |
| 5 |    |               |              |
| 6 | 1  | endereco_de_k | unsigned int |
| 7 |    |               |              |
| 8 |    |               |              |
| 9 |    |               |              |

```

1 ...
2
3 // mostrando o novo valor de 'k'
4 printf("\n\n Valor da variavel k, apos scanf de 'endereco_de_k': %d\n"
,k); // Valor 50
5
6 // mostrando o endereço de 'k' que deve permanecer o mesmo de antes
7 printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
8 printf("\n Endereco da variavel 'k': %u \n",&k); // Endereço 1

```

|   |    |               |              |
|---|----|---------------|--------------|
| 1 | 50 | k             | int          |
| 2 |    |               |              |
| 3 |    |               |              |
| 4 |    |               |              |
| 5 |    |               |              |
| 6 | 1  | endereco_de_k | unsigned int |
| 7 |    |               |              |
| 8 |    |               |              |
| 9 |    |               |              |

```

1 ...
2
3 // mostrando o novo valor de 'k'
4 printf("\n\n Valor da variavel k, apos scanf de 'endereco_de_k': %d\n"
,k); // Valor 50
5
6 // mostrando o endereço de 'k' que deve permanecer o mesmo de antes
7 printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
8 printf("\n Endereco da variavel 'k': %u \n",&k);
9 printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
10 // Endereço 0x1

```

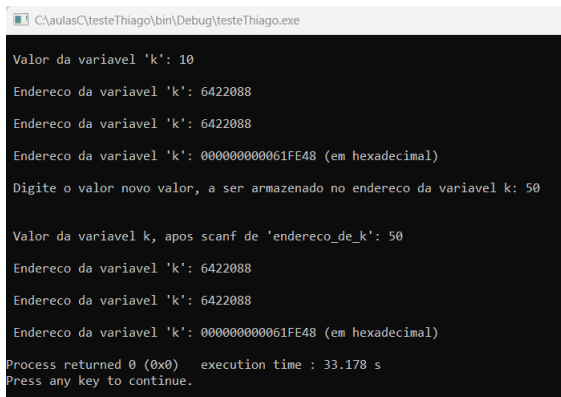
```

1 int k;
2 unsigned int endereco_de_k;
3
4 // inicializando k
5 k = 10;
6 printf("\n Valor da variavel 'k': %d \n",k);
7
8 // obtendo o endereço da variável 'k' // usando o operador &
9 endereco_de_k = &k;
10
11 printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
12 printf("\n Endereco da variavel 'k': %u \n",&k);
13 printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
14
15 // sabemos que o scanf pede um endereço de memória
16 // o que acontece se passarmos o endereço da
17 // variável k?
18 printf("\n Digite o valor novo valor, a ser armazenado no endereço da variavel k: ");
19 // OBSERVE que não estamos usando & no scanf! Isso porque já temos o endereço
20 scanf("%d",endereco_de_k);
21
22 // mostrando o novo valor de 'k'
23 printf("\n\n Valor da variavel k, apos scanf de 'endereco_de_k': %d \n",k);
24
25 // mostrando o endereço de 'k' que deve permanecer o mesmo de antes
26 printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
27 printf("\n Endereco da variavel 'k': %u \n",&k);
28 printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);

```

# Execução Real

- Saída



```
C:\aulas\testeThiago\bin\Debug\testeThiago.exe

Valor da variavel 'k': 10

Endereco da variavel 'k': 6422088

Endereco da variavel 'k': 6422088

Endereco da variavel 'k': 00000000061FE48 (em hexadecimal)

Digite o valor novo valor, a ser armazenado no endereco da variavel k: 50

Valor da variavel k, apos scanf de 'endereco_de_k': 50

Endereco da variavel 'k': 6422088

Endereco da variavel 'k': 6422088

Endereco da variavel 'k': 00000000061FE48 (em hexadecimal)

Process returned 0 (0x0) execution time : 33.178 s
Press any key to continue.
```

- Observe no código que não foi usado `&` para passar o endereço da variável `scanf("%d", endereco_de_k);`

C:\aulasC\testeThiago\bin\Debug\testeThiago.exe

Valor da variavel 'k': 10

Endereco da variavel 'k': 6422088

Endereco da variavel 'k': 6422088

Endereco da variavel 'k': 000000000061FE48 (em hexadecimal)

Digite o valor novo valor, a ser armazenado no endereco da variavel k: 50

Valor da variavel k, apos scanf de 'endereco\_de\_k': 50

Endereco da variavel 'k': 6422088

Endereco da variavel 'k': 6422088

Endereco da variavel 'k': 000000000061FE48 (em hexadecimal)

Process returned 0 (0x0) execution time : 33.178 s

Press any key to continue.

## Faculdade de Computação - FACOM

### Bacharelado em Sistemas de Informação

**Prof. Thiago Pirola Ribeiro**