

MCTA028-15: Programação Estruturada

Aula 7: Ponteiros (Primeira Parte)

Wagner Tanaka Botelho

wagner.tanaka@ufabc.edu.br / wagtanaka@gmail.com

Universidade Federal do ABC (UFABC)

Centro de Matemática, Computação e Cognição (CMCC)

Introdução

➤ **Ponteiros** são um tipo especial de **variáveis** que permitem armazenar **ENDEREÇOS DE MEMÓRIA** em vez de **dados numéricos**, como os tipos **int**, **float** e **double** ou caracteres (tipo **char**);

➤ Variável:

➤ É um espaço reservado de memória usado para **guardar** um **VALOR** que pode ser **modificado** pelo programa.

➤ Ponteiro:

➤ É um espaço reservado de memória usado para **guardar** um **ENDEREÇO** de **MEMÓRIA**.

MEMÓRIA

Endereço	Var	Conteúdo
61FE14	int cont	10
32XZ13	int *p	61FE14
11RE14		



ATENÇÃO!

Verifiquem a diferença do conteúdo do ponteiro ***p** e da variável **cont**.

Declaração

Declaração

- Na Linguagem C, um **ponteiro** pode ser declarado para qualquer **tipo de variável** (**char**, **int**, **float**, **double**, etc), inclusive para aquelas criadas pelo programador (**struct**, etc);
- A **declaração** de um **ponteiro** pelo programador segue a seguinte forma:

```
tipo_do_ponteiro *nome_do_ponteiro;
```

```
void main(){  
    //Declara um ponteiro para int  
    int *p;  
    //Declara um ponteiro para float  
    float *x;  
    //Declara um ponteiro para char  
    char *y;  
}
```

Um ponteiro do tipo **int*** **SÓ** pode **apontar** para uma variável do tipo **int** (ou seja, esse ponteiro só poderá **guardar** o **endereço** de uma variável **int**).

Inicialização e Atribuição

Inicialização

- **Ponteiros** apontam para uma **POSIÇÃO DE MEMÓRIA**. Sendo assim, a **simples** declaração de um ponteiro **NÃO** o faz útil ao programa:
 - Portanto, é necessário **INDICAR** para qual **endereço de memória** ele aponta.
- Um ponteiro declarado, não possui um endereço associado:

Ponteiros NÃO inicializados apontam para um lugar INDEFINIDO.

MEMÓRIA

#	Var	Conteúdo
119		
120	int *p	????
121		

O seu conteúdo **NÃO** é um **endereço válido**!

IMPORTANTE!!

Os ponteiros devem ser **INICIALIZADOS**, ou seja, **APONTADOS** para algum **LUGAR** conhecido!

Inicialização

- Um ponteiro pode ter um valor especial **NULL**, que é o endereço de **NENHUM** lugar;
- A constante **NULL** está definida na biblioteca `stdlib.h`:
 - Trata-se de um **valor reservado** que indica que o **ponteiro** está **APONTANDO** para uma posição de memória **INEXISTENTE**.

`int *p = NULL;`

MEMÓRIA		
#	Var	Conteúdo
119		
120	<code>int *p</code>	NULL
121		

Apontando para **NENHUM** lugar na **MEMÓRIA!!**

Atribuição

Ex_01.c X

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main() {
5      int count = 10;
6      int *p=NULL;
7
8      p = &count;
9
10     printf("Endereco de count: %p\n", &count);
11     printf("Endereco de p: %p", p);
12 }
```

Declara um ponteiro para int.

Atribui ao ponteiro o ENDEREÇO da variável count (int).

Para saber o endereço onde uma variável está guardada na memória, usa-se o operador **&** na frente do nome da variável.

D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\07\

Endereco de count: 000000000061FE14

Endereco de p: 000000000061FE14

Process returned 31 (0x1F) execution ti
Press any key to continue.

Endereços iguais!

Ex_01.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main() {
5      int count = 10;
6      int *p=NULL;
7
8      p = &count;
9
10     printf("Endereco de count: %p\n", &count);
11     printf("Endereco de p: %p", p);
12 }
```

```
D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\07\
Endereco de count: 000000000061FE14
Endereco de p: 000000000061FE14
Process returned 31 (0x1F)  execution ti
Press any key to continue.
```

MEMÓRIA

Endereço em hexadecimal

Endereço	Var	Conteúdo
61FE14	int count	10
34AW15	int *p	61FE14
18IX18		

Está apontando para um endereço de **MESMO** tipo de dado (**int**)

➤ Sabe-se que um **ponteiro** armazena um **endereço de memória**:

➤ Portanto, como saber o **valor** guardado dentro dessa posição?

Para acessar o **CONTEÚDO** da posição de memória para a qual o ponteiro aponta, usa-se o operador **asterisco (*)** na frente do nome do ponteiro.

Ex_02.c X

```
1  #include <stdio.h>
2
3  void main() {
4      int count = 10; ➡ Declara count como int e atribui o
5                          valor 10
6      int *p=NULL; ➡ Declara um ponteiro (p) do tipo int
7
8      p = &count; ➡ Atribui ao ponteiro (p) o endereço de count (int)
9
10     printf("Conteudo apontado por p: %d \n", *p);
11
12     *p = 12; ➡ Atribui um NOVO valor à POSIÇÃO de MEMÓRIA apontada por p
13
14     printf("Conteudo apontado por p: %d \n", *p);
15     printf("Conteudo de count: %d \n", count);
16 }
```

Usa-se o ***** para acessar o **CONTEÚDO** de um **ENDEREÇO** de **MEMÓRIA**.

```
1  #include <stdio.h>
2
3  void main(){
4      int count = 10;
5
6      int *p=NULL;
7
8      p = &count;
9
10     printf("Conteudo apontado por p: %d \n", *p);
11
12     *p = 12;
13
14     printf("Conteudo apontado por p: %d \n", *p);
15     printf("Conteudo de count: %d \n", count);
16 }
```



D:\UFABC\Disciplinas\2021-2025\Q1\F

```
Conteudo apontado por p: 10
Conteudo apontado por p: 12
Conteudo de count: 12
```

Linha	count	p
4	10	
6		→ NULL
8		→ 61FE14
10		{10}
12	12	
14		{12}
15	{12}	

MEMÓRIA

Endereço	Var	Conteúdo
61FE14	int count	12
61FE10	int *p	NULL
...		

Aritmética

Valor do Endereço Armazenado por um Ponteiro

Aritmética - Endereço

- Apenas **DUAS** operações aritméticas podem ser utilizadas nos **ENDEREÇOS** armazenados pelo ponteiros:
 - **Adição** e **subtração**.
- As operações de **ADIÇÃO** e **SUBTRAÇÃO** no **ENDEREÇO** permitem **avançar** ou **retroceder** nas posições da **MEMÓRIA** do computador:
 - Esse tipo de operação é bastante **útil** quando trabalhamos com **arrays**, por exemplo:
 - Lembre-se: um *array* nada mais é do que um **conjunto** de **elementos** adjacentes na **memória**.
 - As **operações** devem ser **inteiras**, pois **NÃO** é possível para andar apenas **MEIA** posição na memória.

```
1  #include <stdio.h>
2
3  void main() {
4      int *p = 0x5DC;
5
6      printf("p = Hexadecimal: %p Decimal: %d \n", p, p);
7
8      p++;
9      printf("p = Hexadecimal: %p Decimal: %d \n", p, p);
10
11     p = p + 15;
12     printf("p = Hexadecimal: %p Decimal: %d \n", p, p);
13
14     p = p - 2;
15     printf("p = Hexadecimal: %p Decimal: %d \n", p, p);
16 }
```

Incrementa (p) em **UMA** posição.

Incrementa (p) em **QUINZE** posições.

Decrementa (p) em **DUAS** posições.

D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\07\Codigos\A

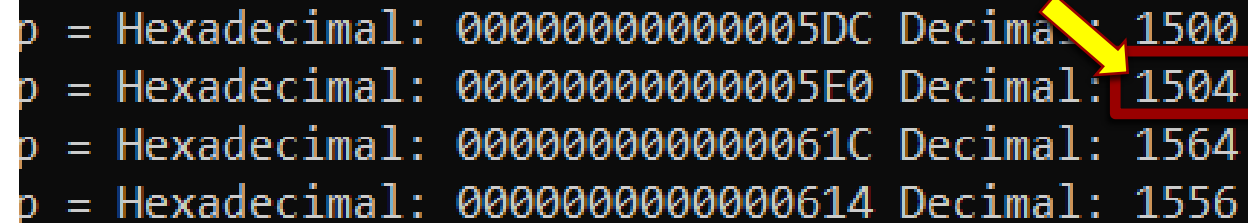
```
p = Hexadecimal: 00000000000005DC Decimal: 1500
p = Hexadecimal: 00000000000005E0 Decimal: 1504
p = Hexadecimal: 000000000000061C Decimal: 1564
p = Hexadecimal: 0000000000000614 Decimal: 1556
```

Você deve estar se perguntando, por que o **INCREMENTO** de p foi **QUATRO**? **NÃO** deveria ter sido **UM**?

Ponteiros armazenam **ENDEREÇOS DE MEMÓRIA** em vez de **DADOS NUMÉRICOS**! Portanto, o **INCREMENTO** está relacionado com as **POSIÇÕES** de **MEMÓRIA** do computador.

D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\07\Codigos\A

```
p = Hexadecimal: 00000000000005DC Decimal: 1500  
p = Hexadecimal: 00000000000005E0 Decimal: 1504  
p = Hexadecimal: 000000000000061C Decimal: 1564  
p = Hexadecimal: 0000000000000614 Decimal: 1556
```



- Então, por que o **INCREMENTO** foi de **QUATRO** posições (1500 para 1504)?
 - O **ponteiro** é do tipo inteiro (**int *p**), ou seja, deve-se receber um **ENDEREÇO** de um **VALOR INTEIRO**;
 - Para a variável inteira (**int x**), o computador reserva uma espaço de **4 bytes** na memória:
 - Por isso, o **INCREMENTO** foi de **quatro** posições.
 - Portanto, se o ponteiro fosse para o tipo **double** (**double x**), as operações de **INCREMENTO/DECREMENTO** mudariam as posições de memória em **8 bytes**.

D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\07\Codigos\A

```
p = Hexadecimal: 000000000000005DC Decimal: 1500
p = Hexadecimal: 000000000000005E0 Decimal: 1504
p = Hexadecimal: 0000000000000061C Decimal: 1564
p = Hexadecimal: 00000000000000614 Decimal: 1556
```

Ex_04.c x

```
1  #include <stdio.h>
2
3  void main(){
4      int *p = 0x5DC;
5
6      printf("p = Hexadecimal: %p Decimal: %d \n",p,p);
7
8      p++;
9      printf("p = Hexadecimal: %p Decimal: %d \n",p,p);
10
11     p = p + 15;
12     printf("p = Hexadecimal: %p Decimal: %d \n",p,p);
13
14     p = p - 2;
15     printf("p = Hexadecimal: %p Decimal: %d \n",p,p);
16 }
```

printf("p = Hexadecimal: %p Decimal: %d \n",p,p); → 1500

p++; → 1500 + 4bytes = 1504

printf("p = Hexadecimal: %p Decimal: %d \n",p,p);

p = p + 15; → 1504 + (15*4bytes) = 1564

printf("p = Hexadecimal: %p Decimal: %d \n",p,p);

p = p - 2; → 1564 - (2*4bytes) = 1556

printf("p = Hexadecimal: %p Decimal: %d \n",p,p);

Double: 8 bytes

```
1  #include <stdio.h>
2
3  void main() {
4      double *p = 0x5DC;
5
6      printf("p = Hexadecimal: %p Decimal: %d \n", p, p); ➡ 1500
7
8      p++;
9      printf("p = Hexadecimal: %p Decimal: %d \n", p, p); ➡ 1500 + 8bytes = 1508
10
11     p = p + 15;
12     printf("p = Hexadecimal: %p Decimal: %d \n", p, p); ➡ 1508 + (15*8bytes) = 1628
13
14     p = p - 2;
15     printf("p = Hexadecimal: %p Decimal: %d \n", p, p); ➡ 1628 - (2*8bytes) = 1612
16 }
```

D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\07\Codigos\At

```
p = Hexadecimal: 00000000000005DC Decimal: 1500
p = Hexadecimal: 00000000000005E4 Decimal: 1508
p = Hexadecimal: 000000000000065C Decimal: 1628
p = Hexadecimal: 000000000000064C Decimal: 1612
```

Conteúdo Apontado pelo Ponteiro

Aritmética - Conteúdo

➤ Valem TODAS as operações aritméticas que o tipo do ponteiro suporta.

Ex_06.c x

D:\UFABC\Disciplinas\2021-2025\Q1\P

```
1  #include <stdio.h>
2
3  void main() {
4      int *p = NULL, x = 10;
5
6      p = &x;
7      printf("Conteudo apontado por p: %d \n", *p);
8
9      *p = *p + 1;
10     printf("Conteudo apontado por p: %d \n", *p);
11
12     *p = (*p) * 10;
13     printf("Conteudo apontado por p: %d \n", *p);
14 }
```

Conteudo apontado por p: 10
Conteudo apontado por p: 11
Conteudo apontado por p: 110

O * acessa o **CONTEÚDO** da posição de memória para a qual o **ponteiro** aponta.

Operações com Ponteiros

`== e !=`

Operações com Ponteiros

- Os operadores `==` e `!=` são usados para saber se **DOIS** ponteiros são **IGUAIS** ou **DIFERENTES**:
 - Dois ponteiros são considerados **IGUAIS** se eles apontam para a **MESMA** posição de **memória**.

```
Ex_07.c x
1  #include <stdio.h>
2
3  void main(){
4      int *p=NULL, *p1=NULL, x=0, y=0;
5
6      p = &x;
7      p1 = &y;
8
9      printf("End. de p: %p\n", p);
10     printf("End. de p1: %p\n", p1);
11
12     if(p == p1){
13         printf("Ponteiros IGUAIS!");
14     }
15     else{
16         printf("Ponteiros DIFERENTES!");
17     }
18 }
```

```
D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\07\Codigos\Aula07\bi
End. de p: 000000000061FE0C
End. de p1: 000000000061FE08
Ponteiros DIFERENTES!
Process returned 21 (0x15)   execution time : 0.667 s
Press any key to continue.
```

Operações com Ponteiros

*Ex_08.c X

```
1  #include <stdio.h>
2
3  void main() {
4      int *p=NULL, *p1=NULL, x=0;
5
6      p = &x;
7      p1 = &x;
8
9      printf("End. de p: %p\n", p);
10     printf("End. de p1: %p\n", p1);
11
12     if(p == p1){
13         printf("Ponteiros IGUAIS!");
14     }
15     else{
16         printf("Ponteiros DIFERENTES!");
17     }
18 }
```

D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\07\Codigos\Aula07\bin\I

End. de p: 000000000061FE0C

End. de p1: 000000000061FE0C

Ponteiros IGUAIS!

Process returned 17 (0x11) execution time : 0.354 s

Press any key to continue.

$>, <, \geq, \leq$

Operações com Ponteiros: Endereço

- Os operadores `>`, `<`, `>=` e `<=` são usados para saber se um **ponteiro APONTA** para uma **POSIÇÃO** mais **adiante** na memória do que o outro:
- Tipo de operação bastante útil para **arrays**.

Ex_09.c x

```
1  #include <stdio.h>
2
3  int main(){
4      int *p=NULL, *p1=NULL, x=0, y=0;
5
6      p = &x; End de p: 6422028
7      p1 = &y; End de p1: 6422024
8
9      printf("End. de p em decimal: %d\n", p);
10     printf("End. de p1 em decimal: %d\n", p1);
11
12     if(p > p1){
13         printf("p aponta para uma posicao a frente de p1\n");
14     }
15     else{
16         printf("p NAO aponta para uma posicao a frente de p1\n");
17     }
18 }
```

Selecionar D:\UFABC\Disciplinas\2021-2025\Q1\PEV\

End. de p em decimal: 6422028

End. de p1 em decimal: 6422024

p aponta para uma posicao a frente de p1

Operações com Ponteiros: Conteúdo

- O operador asterisco (*) na frente do nome do ponteiro indica que está sendo acessado o valor guardado na variável para a qual o ponteiro aponta::

Ex_10.c X

```
1  #include <stdio.h>
2
3  void main(){
4      int *p=NULL, *p1=NULL, x = 10, y = 20;
5
6      p = &x;
7      p1 = &y;
8
9      printf("Conteudo de p: %i\n", *p);
10     printf("Conteudo de p1: %i\n", *p1);
11
12     if(*p > *p1){
13         printf("Conteudo de p > conteudo de p1.\n");
14     }
15     else
16     {
17         printf("Conteudo de p < conteudo de p1.\n");
18     }
19 }
```

D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\07

Conteudo de p: 10

Conteudo de p1: 20

Conteudo de p < conteudo de p1.

Ponteiros Genéricos

Ponteiros Genéricos

- Normalmente, um **ponteiro** aponta para um **tipo** específico **de dado**:
 - Porém, pode-se criar um **PONTEIRO GENÉRICO** que aponta para **TODOS** os **TIPOS DE DADOS** existentes ou que ainda serão criados.
- A **declaração** do ponteiro genérico é:

```
void *nome_do_ponteiro;
```

Ponteiro genérico (`void *pp`)
permite guardar o endereço de
qualquer tipo de dado.

Ex_11.c x

```

1  #include <stdio.h>
2
3  int main() {
4      void *pp = NULL;
5      int *p1 = NULL, p2 = 10;
6
7      p1 = &p2;
8
9      pp = &p2;
10     printf("Endereco em pp: %p \n", pp);
11
12     pp = &p1;
13     printf("Endereco em pp: %p \n", pp);
14
15     pp = p1;
16     printf("Endereco em pp: %p \n", pp);
17 }

```

D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aut

```

Endereco em pp: 000000000061FE0C
Endereco em pp: 000000000061FE10
Endereco em pp: 000000000061FE0C

```

```

Process returned 0 (0x0)   execution
Press any key to continue.

```

`pp = &p2;` Recebe o endereço de um inteiro.
`printf("Endereco em pp: %p \n", pp);`

`pp = &p1;` Recebe o endereço de um ponteiro para inteiro.
`printf("Endereco em pp: %p \n", pp);`

`pp = p1;` Recebe o endereço guardado em p1 (endereço de p2).
`printf("Endereco em pp: %p \n", pp);`

IMPORTANTE!

Será necessário utilizar o operador de *typecast* sobre ele antes de acessar o seu conteúdo.

*Ex_12.c X

```
1  #include <stdio.h>
2
3  void main() {
4      void *pp = NULL;
5      int p2 = 10;
6
7      pp = &p2;
8
9      printf("Conteudo: %d\n", *(int*)pp);
10 }
```

 Ponteiro genérico recebe o endereço de um inteiro..

`printf("Conteudo: %d\n", *(int*)pp);`

Typecast: Converte o ponteiro genérico `pp` para `(int *)` antes de acessar seu conteúdo.

 D:\UFABC\Disciplinas




Conteudo: 10

Process returned 13
Press any key to con

Operações Aritméticas

Ex_13.c X


```

1  #include <stdlib.h>
2
3  void main() {
4      void *p = 0x5DC;
5
6      printf("p = Hexadecimal: %p Decimal: %d \n", p, p);
7
8      p++;  Incrementa o p em uma posição. 1500 + 1byte = 1501
9      printf("p = Hexadecimal: %p Decimal: %d \n", p, p);
10
11     p = p + 15;  Incrementa o p em QUINZE posições. 1501 + 1byte = 1516
12     printf("p = Hexadecimal: %p Decimal: %d \n", p, p);
13
14     p = p - 2;  Decrementa o p em DUAS posições. 1516 - 2bytes = 1514
15     printf("p = Hexadecimal: %p Decimal: %d \n", p, p);
16 }
17

```

IMPORTANTE!!

Se o endereço guardado for, por exemplo, de **UM** inteiro, o **incremento** de uma posição no **PONTEIRO GENÉRICO** (1 byte) **NÃO** levará ao próximo inteiro (4 bytes).

 D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\07\Codigos\Aul

```

p = Hexadecimal: 00000000000005DC Decimal: 1500
p = Hexadecimal: 00000000000005DD Decimal: 1501
p = Hexadecimal: 00000000000005EC Decimal: 1516
p = Hexadecimal: 00000000000005EA Decimal: 1514

```

Referências

- Slides do Prof. Luiz Rozante;
- SALES, André Barros de; AMVAME-NZE, Georges. Linguagem C: roteiro de experimentos para aulas práticas. 2016;
- BACKES, André. Linguagem C Completa e Descomplicada. Editora Campus. 2013;
- SCHILDT, Herbert. C Completo e Total. Makron Books. 1996;
- DAMAS, Luís. Linguagem C. LTC Editora. 1999;
- DEITEL, Paul e DEITEL, Harvey. C Como Programar. Pearson. 2011.