Algoritmos e Estruturas de Dados I
Aula 9 - Ponteiro

Prof. Felipe Lara

Endereço de Memória

- Podemos obter o endereço de memória de uma variável usando o operador &:

```c
int myAge = 43;

printf("%d", myAge);  // Imprime o valor de myAge (43)
```

Endereço de Memória

- Podemos obter o endereço de memória de uma variável usando o operador &:

```c
int myAge = 43;

printf("%d", myAge);  // Imprime o valor de myAge (43)

printf("%p", &myAge); // Imprime o endereço de memória de myAge (0x7ffe5367e044)
```

- Você já usou isso no scanf!!!

Ponteiro

- Ponteiro é uma variável que armazena o endereço de memória de outra variável.

Ponteiro

- Ponteiro é uma variável que armazena o endereço de memória de outra variável.
- Uma variável do tipo ponteiro define o tipo de dado para o qual aponta (ex. int).

Ponteiro

- Ponteiro é uma variável que armazena o endereço de memória de outra variável.
- Uma variável do tipo ponteiro define o tipo de dado para o qual aponta (ex. int).
- O ponteiro é criado com o operador *.

Ponteiro

- Ponteiro é uma variável que armazena o endereço de memória de outra variável.
- Uma variável do tipo ponteiro define o tipo de dado para o qual aponta (ex. int).
- O ponteiro é criado com o operador *.
- O endereço da variável apontada é atribuído ao ponteiro.

Ponteiro

```
int myAge = 43;

int *ptr = &myAge;  // ptr é um ponteiro para int, que guarda o endereço de myAge
```

Ponteiro

```c
int myAge = 43;

int *ptr = &myAge;  // ptr é um ponteiro para int, que guarda o endereço de myAge

// Imprime o valor de myAge (43)
printf("%d\n", myAge);

// Imprime o endereço de memória de myAge (0x7ffe5367e044)
printf("%p\n", &myAge);

// Imprime o endereço de memória apontado por ptr, que corresponde ao endereço de
myAge (0x7ffe5367e044)
printf("%p\n", ptr);
```

Ponteiro

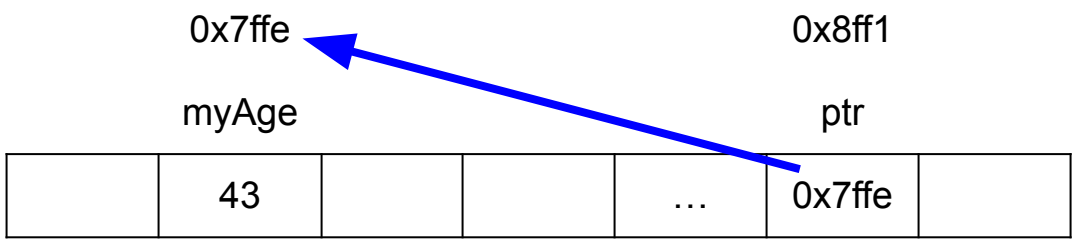
```c
int myAge = 43;

int *ptr = &myAge;  // ptr é um ponteiro para int, que guarda o endereço de myAge

// Imprime o valor de myAge (43)
printf("%d\n", myAge);

// Imprime o endereço de memória de myAge (0x7ffe5367e044)
printf("%p\n", &myAge);

// Imprime o endereço de memória apontado por ptr, que corresponde ao endereço de
myAge (0x7ffe5367e044)
printf("%p\n", ptr);
```



Ponteiro

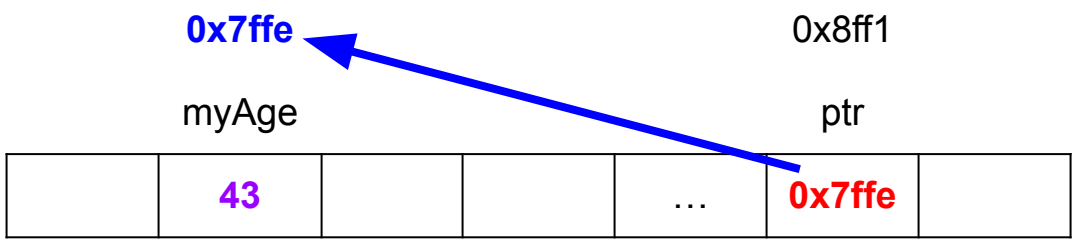
```c
int myAge = 43;

int *ptr = &myAge;  // ptr é um ponteiro para int, que guarda o endereço de myAge

// Imprime o valor de myAge (43)
printf("%d\n", myAge);

// Imprime o endereço de memória de myAge (0x7ffe5367e044)
printf("%p\n", &myAge);

// Imprime o endereço de memória apontado por ptr, que corresponde ao endereço de
myAge (0x7ffe5367e044)
printf("%p\n", ptr);
```



Ponteiro

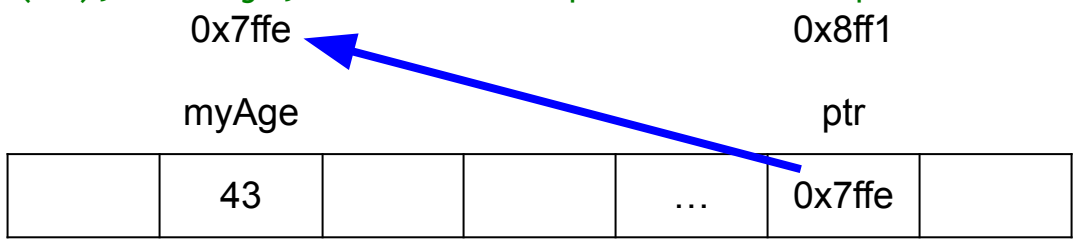
```c
int myAge = 43;

int *ptr = &myAge;  // ptr é um ponteiro para int, que guarda o endereço de myAge

printf("%d\n", myAge);
printf("%p\n", &myAge);
printf("%p\n", ptr);

// Imprime o endereço de memória de ptr (0x7ffe5367f1232)
printf("%p\n", &ptr);

// Imprime o valor de myAge (43), ou seja, o valor do apontamento de ptr
printf("%p\n", *ptr);
```



Ponteiro

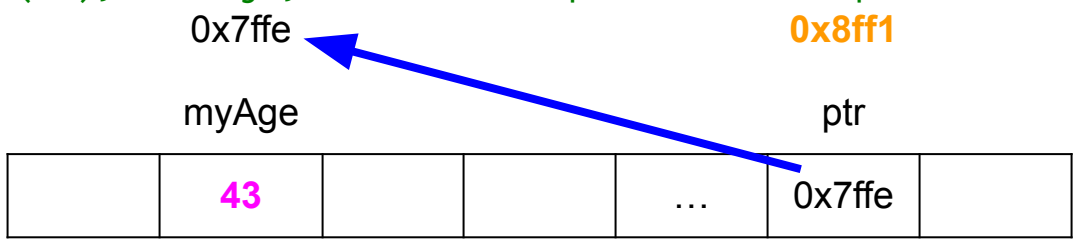
```c
int myAge = 43;

int *ptr = &myAge;  // ptr é um ponteiro para int, que guarda o endereço de myAge

printf("%d\n", myAge);
printf("%p\n", &myAge);
printf("%p\n", ptr);

// Imprime o endereço de memória de ptr (0x7ffe5367f1232)
printf("%p\n", &ptr);

// Imprime o valor de myAge (43), ou seja, o valor do apontamento de ptr
printf("%p\n", *ptr);
```



Ponteiro

- Algumas razões para o uso de ponteiros:
  - Manipular elementos de arranjos;

Ponteiro

- Algumas razões para o uso de ponteiros:
  - Manipular elementos de arranjos;
  - Receber argumentos em funções e procedimentos que necessitem modificar o argumento original;

Ponteiro

- Algumas razões para o uso de ponteiros:
  - Manipular elementos de arranjos;
  - Receber argumentos em funções e procedimentos que necessitem modificar o argumento original;
  - Criar estruturas de dados complexas, como listas encadeadas e árvores binárias, em que um item deve conter referências a outro (AEDs 2);

Ponteiro

- Algumas razões para o uso de ponteiros:
  - Manipular elementos de arranjos;
  - Receber argumentos em funções e procedimentos que necessitem modificar o argumento original;
  - Criar estruturas de dados complexas, como listas encadeadas e árvores binárias, em que um item deve conter referências a outro (AEDs 2);
  - Alocar e desalocar memória do sistema;

Ponteiro

- Algumas razões para o uso de ponteiros:
  - Manipular elementos de arranjos;
  - Receber argumentos em funções e procedimentos que necessitem modificar o argumento original;
  - Criar estruturas de dados complexas, como listas encadeadas e árvores binárias, em que um item deve conter referências a outro (AEDs 2);
  - Alocar e desalocar memória do sistema;
  - Passar para uma função o endereço de outra.

Ponteiro

- Algumas razões para o uso de ponteiros:
  - Manipular elementos de arranjos;
  - Receber argumentos em funções e procedimentos que necessitem modificar o argumento original;
  - Criar estruturas de dados complexas, como listas encadeadas e árvores binárias, em que um item deve conter referências a outro (AEDs 2);
  - Alocar e desalocar memória do sistema;
  - Passar para uma função o endereço de outra.
  - Manipular arquivos.

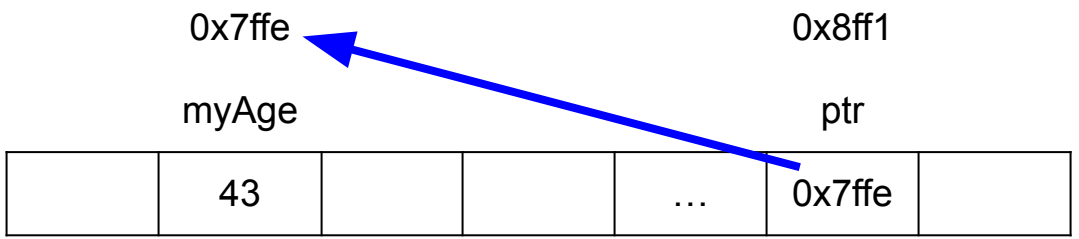
Ponteiro

- Algumas razões para o uso de ponteiros:
  - Manipular elementos de arranjos;
  - Receber argumentos em funções e procedimentos que necessitem modificar o argumento original;
  - Criar estruturas de dados complexas, como listas encadeadas e árvores binárias, em que um item deve conter referências a outro (AEDs 2);
  - Alocar e desalocar memória do sistema;
  - Passar para uma função o endereço de outra.
  - Manipular arquivos.
  - Cuidado! Ponteiros devem ser tratados com atenção, uma vez que é possível corromper dados armazenados em outros endereços de memória.

Ponteiro

Por meio do ponteiro, é possível acessar o conteúdo da variável apontada, usando o operador de indireção.

```
int myAge = 43;
int *ptr = &myAge;
```

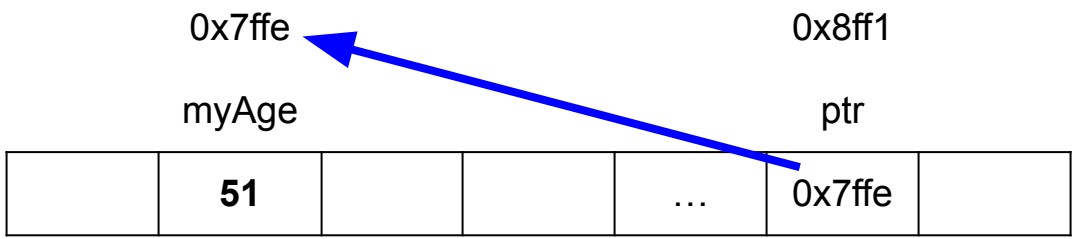


Ponteiro

Por meio do ponteiro, é possível acessar o conteúdo da variável apontada, usando o operador de indireção.

```c
int myAge = 43;
int *ptr = &myAge;

// Altera o valor de myAge usando o ponteiro por derreferência ou indireção
*ptr = 51;
// Imprime o o valor armazenado em myAge (51)
printf("%d\n", myAge);
```



Ponteiro

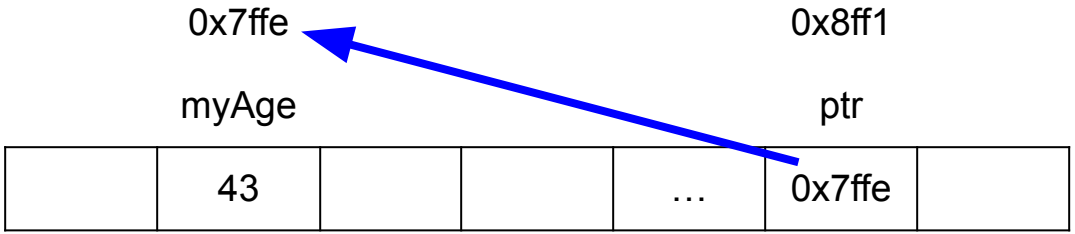
Diferentes formas de fazer o apontamento:

Forma 1:

```
int myAge = 43;
int *ptr = &myAge;
```

Forma 2:

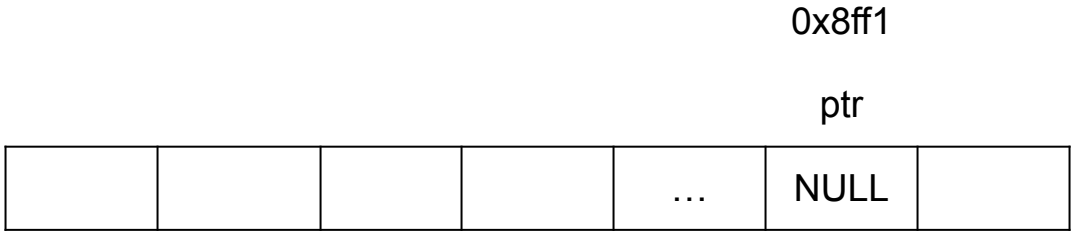
```
int myAge = 43;
int *ptr;
ptr = &myAge;
```



Ponteiro

- Ponteiros devem ser inicializados quando são definidos ou então em uma instrução de atribuição.
- Ponteiros podem ser inicializados com NULL, zero, ou um endereço.
  - NULL: não aponta para nada, é uma constante simbólica.
  - Inicializar um ponteiro com zero é o mesmo que inicializar com NULL.
  - As expressões abaixo são equivalentes.

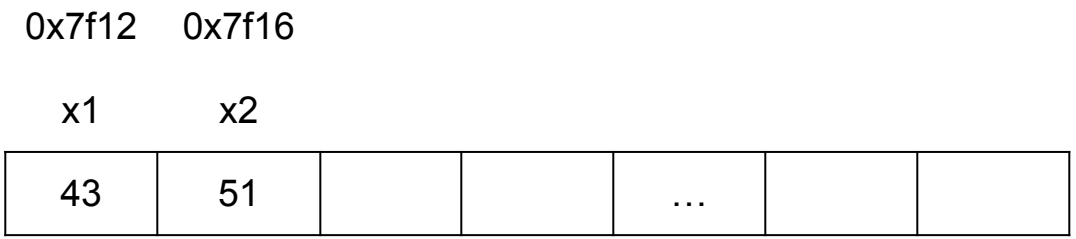
```
int *ptr = NULL;
int *ptr = 0;
```



Operações com Ponteiros

- Por meio do ponteiro, é possível acessar o conteúdo da variável apontada, usando o operador de indireção *.

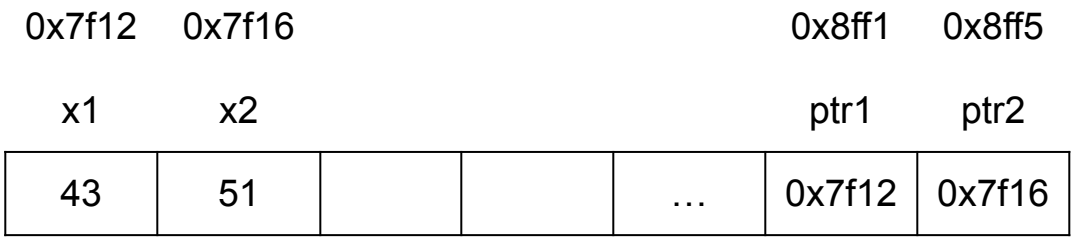
```
int x1 = 43, x2 = 51;
```



Operações com Ponteiros

- Por meio do ponteiro, é possível acessar o conteúdo da variável apontada, usando o operador de indireção *.

```
int x1 = 43, x2 = 51;
int *ptr1 = &x1, *ptr2 = &x2;
```



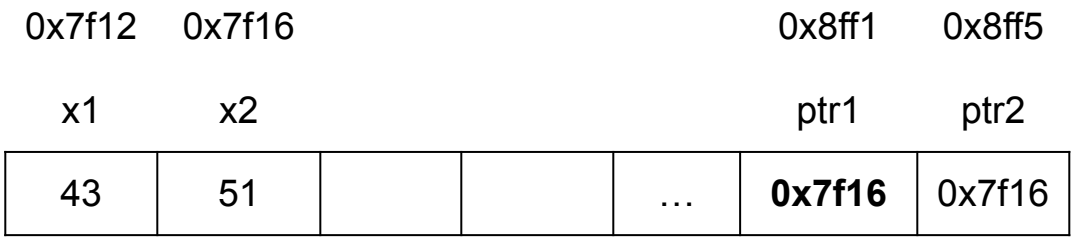
Operações com Ponteiros

- Por meio do ponteiro, é possível acessar o conteúdo da variável apontada, usando o operador de indireção *.

```
int x1 = 43, x2 = 51;
int *ptr1 = &x1, *ptr2 = &x2;

// ptr1 passa a apontar para x2
ptr1 = ptr2;

// Imprime o valor armazenado por x2 (51)
printf("%d\n", *ptr1);
```



Operações com Ponteiros

```c
int myAge = 43;
int *ptr = &myAge;
// Imprime o endereço de memória de myAge (0x7ffe5367e044)
printf("%p\n", ptr);
// Imprime o tamanho em memória de myAge – int - em bytes (4)
printf("%d\n", sizeof(myAge));
```

Operações com Ponteiros

```
int myAge = 43;
int *ptr = &myAge;
// Imprime o endereço de memória de myAge (0x7ffe5367e044)
printf("%p\n", ptr);
// Imprime o tamanho em memória de myAge – int - em bytes (4)
printf("%d\n", sizeof(myAge));
// Incrementa o ponteiro – vai para o próximo endereço de memória com referência a int
– avança 4 bytes
ptr++;
// Imprime o endereço de memória apontado por ptr (0x7ffe5367e048).
printf("%p\n", ptr);
```

Operações com Ponteiros

```
int myAge = 43;
int *ptr = &myAge;
// Imprime o endereço de memória de myAge (0x7ffe5367e044)
printf("%p\n", ptr);
// Imprime o tamanho em memória de myAge – int - em bytes (4)
printf("%d\n", sizeof(myAge));
// Incrementa o ponteiro – vai para o próximo endereço de memória com referência a int
– avança 4 bytes
ptr++;
// Imprime o endereço de memória apontado por ptr (0x7ffe5367e048).
printf("%p\n", ptr);
// Imprime o endereço de memória correspondente ao avanço de 8 bytes (2 * 4) a partir
de ptr (0x7ffe5367e050). ptr não foi alterado
printf("%p\n", ptr+2);
// Imprime o endereço de memória apontado por ptr (0x7ffe5367e048).
printf("%p\n", ptr);
```

Operações com Ponteiros

Por meio do ponteiro, é possível acessar o conteúdo da variável apontada, usando o operador de indireção *.

```
int x1 = 43, x2 = 51;
int *ptr1 = &x1, *ptr2 = &x2;
// Imprime o endereço de memória apontado por ptr1 (0x7ffe5367e044).
printf("%d\n", ptr1);
// Imprime o endereço de memória apontado por ptr2 (0x7ffe5367e048).
printf("%d\n", ptr2);
// Imprime a quantidade de avanços de ptr1 para ptr2 (1).
printf("%d\n", ptr2 - ptr1);
ptr2++;
// Imprime a quantidade de avanços de ptr1 para ptr2 (2).
printf("%d\n", ptr2 - ptr1);
```

Exercício

O que será impresso pelo programa?

```
#include <stdio.h>

int main()
{
    int a=5, b=12, c;
    int *p;
    int *q;
    p = &a;
    q = &b;
    c = *p + *q;
    printf("c = %d", c);
}
```

Exercício

O que será impresso pelo programa?

```
#include <stdio.h>

int main()
{
    int a=5, b=12, c;
    int *p;
    int *q;
    p = &a;
    q = &b;
    c = *p + *q;
    printf("c = %d", c);
}
```

Resposta:

c = 17

Exercício

O que será impresso pelo programa?

```
#include <stdio.h>

int main()
{
    int x, y, *p;
    y = 0;
    p = &y;
    x = *p;
    x = 4;
    (*p)++;
    x--;
    (*p) += x;
    printf("x=%d  y=%d *p=%d", x, y, *p);
}
```

Exercício

O que será impresso pelo programa?

```
#include <stdio.h>

int main()
{
    int x, y, *p;
    y = 0;
    p = &y;
    x = *p;
    x = 4;
    (*p)++;
    x--;
    (*p) += x;
    printf("x=%d  y=%d *p=%d", x, y, *p);
}
```

Resposta:

x=3 y=4 *p=4

Funções - Passagem de Parâmetro por Ponteiro

- Passagem por ponteiro permite que a função altere o valor dos parâmetros e essa alteração persista no ambiente de chamada

Funções - Passagem de Parâmetro por Ponteiro

- Passagem por ponteiro permite que a função altere o valor dos parâmetros e essa alteração persista no ambiente de chamada

- Para passar um parâmetro por ponteiro na linguagem C, o parâmetro deve ser um ponteiro, ou seja:
  - Na **criação da função** use o operador ***** (é um ponteiro - indicar o tipo de dado e o operador * para as variáveis que serão referência)
  - Na **chamada da função** use o operador **&**

Funções - Passagem de Parâmetro por Ponteiro

```c
#include <stdio.h>
void maisDois(int *num1, int *num2){

  *num1 = *num1 + 2;
  *num2 = *num2 + 2;
}


int main(){
  int a = 5, b = 10;
  printf("\n\nEles valem %d, %d\n", a, b);
  maisDois(&a, &b);
  printf("\n\nEles agora valem %d, %d\n", a, b);
  return 0;
}
```

Funções - Passagem de Parâmetro por Ponteiro

```c
#include <stdio.h>
void maisDois(int *num1, int *num2){

  *num1 = *num1 + 2;
  *num2 = *num2 + 2;
}


int main(){
  int a = 5, b = 10;
  printf("\n\nEles valem %d, %d\n", a, b);
  maisDois(&a, &b);
  printf("\n\nEles agora valem %d, %d\n", a, b);
  return 0;
}
```

a = 5 e b = 10

a = 7 e b = 12

Funções - Passagem de Parâmetro por Ponteiro

```c
#include <stdio.h>
void trocar(int *num1, int *num2){

  int temp;
  temp = *num1;
  *num1 = *num2;
  *num2 = temp;
}


int main(){
  int a = 5, b = 10;
  printf("\n\nEles valem %d, %d\n", a, b);
  trocar(&a, &b);
  printf("\n\nEles agora valem %d, %d\n", a, b);
  return 0;
}
```

Funções - Passagem de Parâmetro por Ponteiro

```c
#include <stdio.h>
void trocar(int *num1, int *num2){

  int temp;
  temp = *num1;
  *num1 = *num2;
  *num2 = temp;
}


int main(){
  int a = 5, b = 10;
  printf("\n\nEles valem %d, %d\n", a, b);
  trocar(&a, &b);
  printf("\n\nEles agora valem %d, %d\n", a, b);
  return 0;
}
```

a = 5 e b = 10

a = 10 e b = 5

Funções - Passagem de Parâmetro por Ponteiro

```
void calculaMedia(int, int, float *);

int main()
{
  int num1, num2;
  float media;
  printf("Digite o primeiro valor:");
  scanf("%d", &num1);
  printf("Digite o segundo valor:");
  scanf("%d", &num2);
  calculaMedia(num1, num2,  &media);
  printf("A media e:  %.1f", media);
  return 0;
}
void calculaMedia(int n1, int n2, float *m)
{
   *m = ( n1+ n2 ) / 2.0;
}
```

Nesse exemplo, a média não é retornada na função calculaMedia(), mas como o parâmetro é por referência, qualquer alteração no valor da média dentro da função é refletido no ambiente da chamada. Observe que no protótipo apenas indicamos com tipo + * a variável que será parâmetro por referência

Na chamada da função usamos o operador &, observe que apenas a variável média foi passada por ponteiro

Na criação da função usamos o operador *

PUC Minas

Dúvidas?