



UFOP

Universidade Federal
de Ouro Preto

Ponteiros

CSI030-PROGRAMAÇÃO DE COMPUTADORES I





UFOP

Universidade Federal
de Ouro Preto

VideoAula

A videoaula do material a seguir pode ser encontrado em:

<https://www.youtube.com/playlist?list=PLKQE10z2LK0FOWL5DQajyQ69vESiYT6BS>





UFOP

Universidade Federal
de Ouro Preto

Introdução

- Vamos começar com um problema prático. Como fazer uma função de troca?

```
? troca(?){\n}\n\nint main(){\n    int x = 1, y =2;\n    troca(?)\n    return 0;\n}
```



UFOP

Universidade Federal
de Ouro Preto

Introdução

- Com o conhecimento que temos até agora não conseguimos construir essa função de maneira simples.
- Temos que encontrar uma maneira simples de permitir que a função troca modifique o conteúdo das variáveis a e b .





UFOP
Universidade Federal
de Ouro Preto

Introdução aos Ponteiros

- A memória em um computador pode ser considerada simplesmente como uma longa fila de “caixas”, com cada caixa tendo um valor e um índice associado a ela.
- Se isso parece com um vetor, é porque é!





Introdução aos Ponteiros

- A memória do computador é apenas um vetor gigante. As "caixas" podem conter diferentes tipos, mas os números associados a cada caixa é apenas um número:

	int	int	char	char	float	float	int				
	↓	↓	↓	↓	↓	↓	↓				
Valor	0	1	't'	'j'	1.0	10.0	2
endereço	0	1	2	3	4	5	6



Introdução aos Ponteiros

- Assim, temos que falar para função troca que ela deve trocar as variáveis com endereço 1 e 6

	int	int	char	char	float	float	int				
	↓	↓	↓	↓	↓	↓	↓				
Valor	0	1	't'	'j'	1.0	10.0	2
endereço	0	1	2	3	4	5	6



Ponteiros

- Toda variável possui um endereço de memória.
- Ponteiro (para um tipo) é um tipo de dado especial que armazena endereços de memória (onde cabem valores do tipo apontado).
- Uma variável que é um ponteiro de um tipo A armazena o endereço de uma outra variável também do tipo A.
- Ponteiros permitem alocação dinâmica de memória, ou seja, alocação de memória enquanto o programa já está sendo executado.



UFOP

Universidade Federal
de Ouro Preto

Declaração de Ponteiros em C

- Variáveis do tipo ponteiro pode ser declaradas assim:

tipo *variável;

tipo *variável1, *variável2;



UFOP

Universidade Federal
de Ouro Preto

Declaração de Ponteiros em C

- Exemplos:

```
char *pc;
```

//pc armazena endereço de variável do tipo char

```
int *pi1, *pi2;
```

//pi1 e pi2 armazenam endereços de variáveis do tipo int.



UFOP

Universidade Federal
de Ouro Preto

Operador &

- Obtém o endereço de memória da variável à qual é aplicado

```
int count;  
int *m;  
count = 5;  
m = &count;
```

variável	count	m
conteúdo	5	0x500
endereço	0x500	0x600



Operador *

- Acessa o conteúdo que está armazenado no endereço indicado pelo ponteiro ao qual é aplicado

```
int count, q;  
int *m;  
count = 5;  
m = &count;  
q = *m;  
*m=10
```

variável	count	q	m
conteúdo	5 10	5	0x500
endereço	0x500	0x600	0x700



Cuidados com Ponteiros (I)

- Não se pode atribuir um valor para o conteúdo de um endereço (utilizando o operador * sobre um ponteiro) sem se ter certeza de que o ponteiro possui um endereço válido!

Errado

```
int a, b;  
int *c;  
b = 10;  
*c = 13;  
//Armazena 13 em qual endereço?
```



Correto

```
int a, b;  
int *c;  
b = 10;  
c = &a;  
*c = 13;
```



UFOP

Universidade Federal
de Ouro Preto

Cuidados com Ponteiros (II)

- Como o operador de conteúdo é igual ao operador de multiplicação, é preciso tomar cuidado para não confundi-los:

Errado

```
int a, b;  
int *c;  
b = 10;  
c = &a;  
*c = 13;  a = b * c;
```

Correto

```
int a, b;  
int *c;  
b = 10;  
c = &a;  
*c = 13;  a = b * (*c);
```



Cuidados com Ponteiros (III)

- Um ponteiro sempre armazena um endereço para um local de memória que pode armazenar um tipo específico.

Errado

```
float a, b;  
int *c;  
b = 10.80;  
c = &b; //c é ponteiro para inteiros  
a = *c;  
printf("%f", a);
```

Correto

```
float a, b;  
float *d;  
b = 10.80;  
d = &b;  
a = *d;  
printf("%f", a);
```



UFOP

Universidade Federal
de Ouro Preto

Inicialização de Ponteiros

- Na declaração de um ponteiro, é uma boa prática atribuir a constante **NULL**.
- Isto permite saber se um ponteiro aponta para um endereço válido.

```
float *a = NULL, *b = NULL, c=5;  
a = &c;  
if(a != NULL){  
    b = a;  
    printf("Numero : %f", *b);  
}
```




UFOP

Universidade Federal
de Ouro Preto

Ponteiros e Vetores

- Quando declaramos uma variável do tipo vetor, é armazenada uma quantidade de memória contígua de tamanho igual ao declarado.
- Uma variável vetor armazena o endereço de início da região de memória destinada ao vetor.
- Assim, uma variável vetor também é um ponteiro!
- Quando passamos um vetor para uma função, estamos passando o endereço da memória onde o vetor começa: por isto podemos alterar o vetor dentro da função!



Exemplo: Vetor como parâmetro de função

```
void zeraVet(int vet[], int tam){  
    int i;  
    for(i = 0; i < tam; i++)  
        vet[i] = 0;  
}  
  
int main(){  
    int vetor[] = {1, 2, 3, 4, 5};  
    int i;  
    zeraVet(vetor, 5);  
    for(i = 0; i<5; i++)  
        printf("%d, ", vetor[i]);  
    return 0;  
}
```



UFOP

Universidade Federal
de Ouro Preto

Ponteiros e Vetores: Semelhanças

- Como um vetor armazena um endereço, pode-se atribuir um vetor a um ponteiro para o mesmo tipo dos elementos do vetor:

```
int a[] = {1, 2, 3, 4, 5};
```

```
int *p;
```

```
p = a;
```

- Logo, é possível utilizar um ponteiro como se fosse um vetor:

```
for( i = 0; i < 5; i++)
```

```
    p[ i ] = i * i;
```



Ponteiros e Vetores: Diferenças

- Uma variável vetor armazena um endereço fixo.
- Um ponteiro pode receber por atribuição diferentes endereços.
- Isto significa que não se pode fazer uma atribuição de endereço a uma variável vetor.

```
int a[] = {1, 2, 3, 4, 5};
```

```
int b[5], *p;
```

```
p = a; // Ok. O ponteiro p recebe o endereço do vetor a.
```

```
b = a; // Erro de compilação! O vetor b não pode receber o endereço do vetor a.
```



UFOP

Universidade Federal
de Ouro Preto

Passagem por Cópia e por Referência

- **Passagem por Cópia:** Quando passamos uma variável simples para uma função, o valor da variável é copiado para a variável correspondente no corpo da função e o valor original não sofre as alterações do corpo da função.
- **Passagem por Referência:** Uma variável passada por referência para uma função sofre as alterações do corpo da função, ou seja, as alterações realizadas dentro da função afetam o valor original da variável.



Passagem por Cópia em C

```
void nao_troca(int a, int b){  
    int temp;  
    temp = a;  
    a = b;  
    b = temp;  
}  
  
void main(){  
    int x = 1, y = 2;  
    nao_troca(x, y);  
}
```

variável	x	y	a	b	temp
conteúdo	1	2	1	2	
endereço	500	600	700	800	900

variável	x	y	a	b	temp
conteúdo	1	2	2	1	1
endereço	500	600	700	800	900

variável	x	y
conteúdo	1	2
endereço	500	600



UFOP

Universidade Federal
de Ouro Preto

Simulando Passagem por Referência em C

- A linguagem **C** possui apenas **passagem de parâmetros por cópia**.
- Pode-se simular a **passagem por referência** passando-se como **parâmetro da função o endereço da variável**.
- Este endereço será copiado para um ponteiro dentro da função, permitindo que as alterações dentro da função afetem o valor original da variável.



UFOP

Universidade Federal
de Ouro Preto

Simulando Passagem por Referência em C

```
void troca(int *a, int *b){  
    int temp;  
    temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
int main(){  
    int x = 1, y = 2;  
    troca(&x, &y);  
    return 0;  
}
```

variável	x	y	a	b	temp
conteúdo	1	2	500	600	
endereço	500	600	700	800	900

variável	x	y	a	b	temp
conteúdo	2	1	500	600	1
endereço	500	600	700	800	900

variável	x	y
conteúdo	2	1
endereço	500	600



Usos de passagem por referência

- Pode-se retornar mais de um valor em uma função através de passagem por referência.
- Cria-se um procedimento e “retorna-se” os diferentes resultados em parâmetros recebidos por referência no procedimento.
- Como as alterações nestes parâmetros são visíveis fora do procedimento, pode-se retornar mais de um valor como resultado da execução do procedimento!



UFOP

Universidade Federal
de Ouro Preto

Exercícios com Ponteiros

CSI030-PROGRAMAÇÃO DE COMPUTADORES I



Exemplo 1

- Como percorrer e preencher um vetor sem usar o V sem usar o []

```
int main(){  
    int V[5];  
  
    ???  
    return 0;  
}
```



Exemplo 1

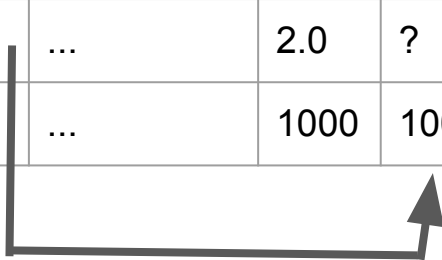
- Uma variável vetor armazena o endereço de início da região de memória destinada ao vetor.

Variável	x	y	V	...							
Valor	0	1	1001	...	2.0	?	?	?	?	?	..
endereço	0	1	2	...	1000	1001	1002	1003	1004	1005	...

Exemplo 1

- Uma variável vetor armazena o endereço de início da região de memória destinada ao vetor.

Variável	x	y	V								
Valor	0	1	1001	...	2.0	?	?	?	?	?	..
endereço	0	1	2	...	1000	1001	1002	1003	1004	1005	...





Exemplo 1

- Vamos criar um ponteiro para ajudar.

```
int *p;
```

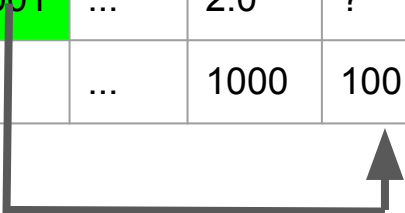
Variável	x	y	V	p	...							
Valor	0	1	1001	???	...	2.0	?	?	?	?	?	..
endereço	0	1	2	3	...	1000	1001	1002	1003	1004	1005	...

Exemplo 1

- Vamos fazer esse ponteiro apontar para V

$p = V;$

Variável	x	y	V	p	...							
Valor	0	1	1001	1001	...	2.0	?	?	?	?	?	..
endereço	0	1	2	3	...	1000	1001	1002	1003	1004	1005	...

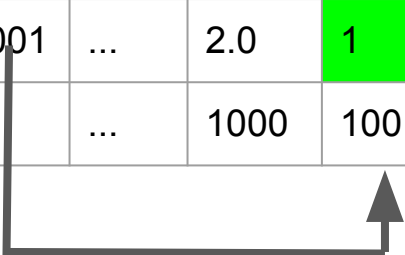


Exemplo 1

- Vamos atribuir algo para onde p aponta:

`*p = 1;`

Variável	x	y	V	p	...							
Valor	0	1	1001	1001	...	2.0	1	?	?	?	?	..
endereço	0	1	2	3	...	1000	1001	1002	1003	1004	1005	...

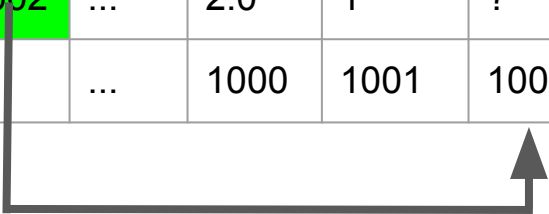


Exemplo 1

- O operador ++ pode ser usado para mover a posição do ponteiro para a próxima posição:

p++;

Variável	x	y	V	p	...							
Valor	0	1	1001	1002	...	2.0	1	?	?	?	?	..
endereço	0	1	2	3	...	1000	1001	1002	1003	1004	1005	...

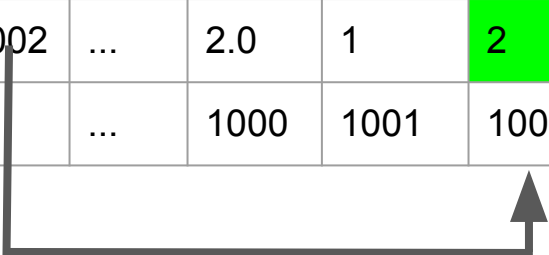


Exemplo 1

- Agora podemos atribuir para a próxima posição:

$*p = 2;$

Variável	x	y	V	p	...							
Valor	0	1	1001	1002	...	2.0	1	2	?	?	?	..
endereço	0	1	2	3	...	1000	1001	1002	1003	1004	1005	...



Exemplo 1

- Repetindo o processo:

```
int main(){  
    int V[5];  
    int *p,i;  
  
    p = V;  
    for(i=0;i < 5;i++){  
        *p = i + 1;  
        p++;  
    }  
    // vamos imprimir V de forma normal para conferir o resultado  
    for(i=0;i < 5;i++){  
        printf("%d ",V[i]);  
    }  
    return 0;  
}
```



UFOP

Universidade Federal
de Ouro Preto

Aritmética de Ponteiros

- Ponteiros podem ser utilizados nas seguintes operações aritméticas:
- Ponteiros podem ser incrementados e decrementados
- Pode-se somar ou subtrair inteiros a ponteiros
- Um ponteiro pode ser subtraído de outro (resultando na quantidade de elementos do tipo do ponteiro existente no intervalo entre os dois ponteiros!)



UFOP

Universidade Federal
de Ouro Preto

Exemplo 2

- Construir uma função que retorna uma referência (ponteiro) para o maior elemento de uma matriz.
- Modificar o elemento da matriz usando esse ponteiro.



Exemplo 2

- A função deve receber a matriz e deve retornar um ponteiro.

```
int main(){  
  
    int *pMaior;  
    int Mat[5][5] = { { 4, 6, 7, 8, 31},  
                      { 0, -30, 4, 200, 0},  
                      { 0, 10, 4, 220, 1},  
                      { 3, 15, 4, 120, 8},  
                      { 7, -32, 4, 100, 0}};  
  
    pMaior = Maior(Mat);  
  
    return 0;  
}
```

Exemplo 2

- A função deve receber a matriz e deve retornar um ponteiro.

```
int* Maior(int Mat[5][5]){  
    int i,j, *pMaior;  
  
    pMaior = &Mat[0][0];  
  
    for(i=0;i<5;i++){  
        for(j=0;j<5;j++){  
            if(Mat[i][j] > (*pMaior))  
                pMaior = &Mat[i][j];  
        }  
    }  
  
    return pMaior;  
}
```

Exemplo 2

- Podemos modificar o elemento da matriz

```
int main(){  
  
    int *pMaior;  
    int Mat[5][5] = { { 4, 6, 7, 8, 31},  
                      { 0, -30, 4, 200, 0},  
                      { 0, 10, 4, 220, 1},  
                      { 3, 15, 4, 120, 8},  
                      { 7, -32, 4, 100, 0}};  
  
    pMaior = Maior(Mat);  
  
    printf("O maior elemento %d\n",(*pMaior));  
    *pMaior = 0;  
    printf("O elemento da posicao (2,3) e %d\n",Mat[2][3]);  
  
    return 0;  
}
```