Faculdade de Computação - FACOM

Bacharelado em Sistemas de Informação

FACOM32201 - Algoritmos e Programação II

Prof. Thiago Pirola Ribeiro

FUNÇÕES

Passagem de Parâmetros por Valor

Passagem de Parâmetros

- Na linguagem C, os parâmetros de uma função são sempre passados por **valor**, ou seja, uma cópia do valor do parâmetro é feita e passada para a função .

Passagem de Parâmetros

- Na linguagem C, os parâmetros de uma função são sempre passados por **valor**, ou seja, uma cópia do valor do parâmetro é feita e passada para a função .
- Mesmo que esse valor mude dentro da função, nada acontece com o valor de fora da função.

Passagem por valor

```c
void soma_mais_um(int x) {
    x = x + 1;
    printf("Dentro da funcao: x = %d\n" , x);
}

int main() {
    int x = 5;
    printf("Antes da funcao: x = %d\n",x);

    soma_mais_um(x);

    printf("Depois da funcao:  x = %d\n",x);
    return 0;
}
```

Passagem por valor

```c
void soma_mais_um(int x) {
   x = x + 1;
   printf("Dentro da funcao: x = %d\n" , x);
}

int main() {
   int x = 5;

   printf("Antes da funcao: x = %d\n",x);
   soma_mais_um(x);

   printf("Depois da funcao:  x = %d\n",x);
   return 0;
}
```

```
Antes da funcao: x = 5
Dentro da funcao: x = 6
Depois da funcao:  x = 5
```

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

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
1  void soma_mais_um(int x) {
2     x = x + 1;
3     printf("Dentro da funcao: x = %d\n" , x);
4  }
5
6  int main() {
7     int x = 5;
8     printf("Antes da funcao: x = %d\n",x);
9
10    --> PAROU AQUI <--
11
12    soma_mais_um(x);
13
14    printf("Depois da funcao:  x = %d\n",x);
15    return 0;
16 }
```

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

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	5	x	int
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
1  void soma_mais_um(int x) {
2  --> PAROU AQUI <--
3      x = x + 1;
4      printf("Dentro da funcao: x = %d\n" , x);
5  }
6
7  int main() {
8      int x = 5;
9
10     printf("Antes da funcao: x = %d\n",x);
11     soma_mais_um(x);  --> ENTROU <--
12
13     printf("Depois da funcao:  x = %d\n",x);
14     return 0;
15 }
```

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

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	6	x	int
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
1  void soma_mais_um(int x) {
2     x = x + 1;
3   --> PAROU AQUI <--
4     printf("Dentro da funcao: x = %d\n" , x);
5  }
6
7  int main() {
8     int x = 5;
9
10    printf("Antes da funcao: x = %d\n",x);
11    soma_mais_um(x);        --> ENTROU <--
12
13    printf("Depois da funcao:  x = %d\n",x);
14    return 0;
15 }
```

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

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
1  void soma_mais_um(int x) {
2     x = x + 1;
3     printf("Dentro da funcao: x = %d\n" , x);
4  }
5  --> PAROU AQUI <--
6
7  int main() {
8     int x = 5;
9
10    printf("Antes da funcao: x = %d\n",x);
11    soma_mais_um(x);        --> ENTROU <--
12
13    printf("Depois da funcao:  x = %d\n",x);
14    return 0;
15 }
```

Passagem por valor

```c
void soma_mais_um(int x) {
   x = x + 1;
   printf("Dentro da funcao: x = %d\n" , x);
}

int main() {
   int x = 5;

   printf("Antes da funcao: x = %d\n",x);
   soma_mais_um(x);

   printf("Depois da funcao:  x = %d\n",x);
   return 0;
}
```

```
Antes da funcao: x = 5
Dentro da funcao: x = 6
Depois da funcao:  x = 5
```

FUNÇÕES

Passagem de Parâmetros por Referência

Passagem de Referência

- Para passar um parâmetro por referência, coloca-se um asterisco "*" na frente do nome do parâmetro na declaração da função (ou seja, um ponteiro):
  `float sqr (float *num);`

Passagem de Referência

- Para passar um parâmetro por referência, coloca-se um asterisco "*" na frente do nome do parâmetro na declaração da função (ou seja, um ponteiro):
  ```
  float sqr (float *num);
  ```

- Ao se chamar a função, é necessário agora utilizar o operador "&", igual como é feito com a função `scanf()`:
  ```
  y = sqr(&x);
  ```

Passagem de Referência

- No corpo da função, é necessário usar colocar um asterisco "*" sempre que se desejar acessar o conteúdo do parâmetro passado por referência.

- Por Valor:

```
void soma_mais_um(int n){
    n = n + 1;
}
```

- Por Referência:

```
void soma_mais_um(int *n){
    *n = *n + 1;
}
```

Passagem de Referência

```
1  void soma_mais_um(int *x) {
2      *x = *x + 1;
3      printf("Dentro da funcao: x = %d\n" , *x);
4  }
5
6  int main() {
7      int x = 5;
8      printf("Antes da funcao: x = %d\n",x);
9      soma_mais_um(&x);
10     printf("Depois da funcao:
11     x = %d\n",x);
12     return 0;
13 }
```

Passagem de Referência

```c
void soma_mais_um(int *x) {
    *x = *x + 1;
    printf("Dentro da funcao: x = %d\n" , *x);
}

int main() {
    int x = 5;
    printf("Antes da funcao: x = %d\n",x);
    soma_mais_um(&x);
    printf("Depois da funcao: x = %d\n",x);
    return 0;
}
```

```
Antes da funcao: x = 5
Dentro da funcao: x = 6
Depois da funcao:  x = 6
```

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

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
1  void soma_mais_um(int *x) {
2      *x = *x + 1;
3      printf("Dentro da funcao: x = %d\n" , *x);
4  }
5
6  int main() {
7      int x = 5;
8      printf("Antes da funcao: x = %d\n",x);
9      --> Parou Aqui <--
10     soma_mais_um(&x);
11
12     printf("Depois da funcao: x = %d\n",x);
13     return 0;
14 }
```

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

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	1	x	int *
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
1  void soma_mais_um(int *x) {
2  --> Parou Aqui <--
3      *x = *x + 1;
4      printf("Dentro da funcao: x = %d\n" , *x);
5  }
6
7  int main() {
8      int x = 5;
9      printf("Antes da funcao: x = %d\n",x);
10
11     soma_mais_um(&x);  --> ENTROU <--
12
13     printf("Depois da funcao: x = %d\n",x);
14     return 0;
15 }
```

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

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	1	x	int *
49			
50			
51			
52			
53			
54			
55			
56			
57			

Observe que o tipo de x da função é int *

```
1  void soma_mais_um(int *x) {
2  --> Parou Aqui <--
3      *x = *x + 1;
4      printf("Dentro da funcao: x = %d\n" , *x);
5  }
6  int main() {
7      int x = 5;
8      printf("Antes da funcao: x = %d\n",x);
9
10     soma_mais_um(&x);  --> ENTROU <--
11
12     printf("Depois da funcao: x = %d\n",x);
13     return 0;
14 }
```

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

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	1	x	int *
49			
50			
51			
52			
53			
54			
55			
56			
57			

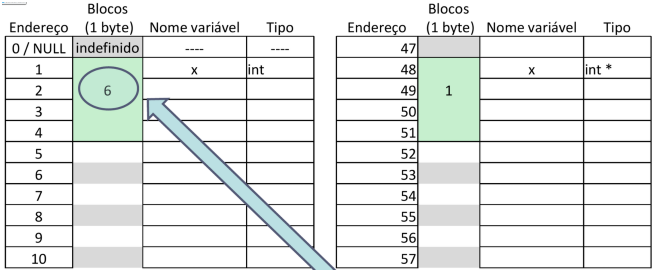
Observe que o valor copiado para x da função é 1, e não 5. Endereço de x, e não valor de x

```
1  void soma_mais_um(int *x) {
2  --> Parou Aqui <--
3      *x = *x + 1;
4      printf("Dentro da funcao: x = %d\n" , *x);
5  }
6  int main() {
7      int x = 5;
8      printf("Antes da funcao: x = %d\n",x);
9
10     soma_mais_um(&x);  --> ENTROU <--
11
12     printf("Depois da funcao: x = %d\n",x);
13     return 0;
14 }
```

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

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	1	x	int *
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
1  void soma_mais_um(int *x) {
2  --> Parou Aqui <--
3      *x = *x + 1;
4      printf("Dentro da funcao: x = %d\n" , *x);
5  }
6
7  int main() {
8      int x = 5;
9      printf("Antes da funcao: x = %d\n",x);
10
11     soma_mais_um(&x);  --> ENTROU <--
12
13     printf("Depois da funcao: x = %d\n",x);
14     return 0;
15 }
```



Observe que o valor de x (`main`) foi alterado pela função, por meio de um ponteiro

```
1  void soma_mais_um(int *x) {
2     *x = *x + 1;
3  --> Parou Aqui <--
4     printf("Dentro da funcao: x = %d\n" , *x);
5  }
6
7  int main() {
8     int x = 5;
9     printf("Antes da funcao: x = %d\n",x);
10
11    soma_mais_um(&x);  --> ENTROU <--
12
13    printf("Depois da funcao:
14    x = %d\n",x);
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1		x	int
2	6		
3			
4			
5			
6			
7			
8			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			

```
1  void soma_mais_um(int *x) {
2      *x = *x + 1;
3      printf("Dentro da funcao: x = %d\n" , *x);
4  }
5  --> Parou Aqui <--
6
7  int main() {
8      int x = 5;
9      printf("Antes da funcao: x = %d\n",x);
10
11     soma_mais_um(&x);  --> ENTROU <--
12
13     printf("Depois da funcao:
14     x = %d\n",x);
15     return 0;
16 }
```

Passagem de Referência

```c
void soma_mais_um(int *x) {
    *x = *x + 1;
    printf("Dentro da funcao: x = %d\n" , *x);
}

int main() {
    int x = 5;
    printf("Antes da funcao: x = %d\n",x);

    soma_mais_um(&x);

    printf("Depois da funcao:
    x = %d\n",x);
    return 0;
}
```

```
Antes da funcao: x = 5
Dentro da funcao: x = 6
Depois da funcao:  x = 6
```

FUNÇÕES

Arrays como parâmetros

Arrays como parâmetros

- Arrays são sempre passados por referência para uma função;

Arrays como parâmetros

- Arrays são sempre passados por referência para uma função;
  - A passagem de arrays por referência evita a cópia desnecessária de grandes quantidades de dados para outras áreas de memória durante a chamada da função, o que afetaria o desempenho do programa.

Arrays como parâmetros

- Arrays são sempre passados por referência para uma função;
  - A passagem de arrays por referência evita a cópia desnecessária de grandes quantidades de dados para outras áreas de memória durante a chamada da função, o que afetaria o desempenho do programa.
- É necessário declarar um segundo parâmetro (em geral uma variável inteira) para passar para a função o tamanho do array separadamente.

Arrays como parâmetros

- Arrays são sempre passados por referência para uma função;
  - A passagem de arrays por referência evita a cópia desnecessária de grandes quantidades de dados para outras áreas de memória durante a chamada da função, o que afetaria o desempenho do programa.
- É necessário declarar um segundo parâmetro (em geral uma variável inteira) para passar para a função o tamanho do array separadamente.
  - Quando passamos um array por parâmetro, independente do seu tipo, o que é de fato passado é o endereço do primeiro elemento do array.

Arrays como parâmetros

- Na passagem de um array como parâmetro de uma função podemos declarar a função de diferentes maneiras, todas equivalentes:

```
void imprime(int *m, int n);

void imprime(int m[], int n);

void imprime(int m[5], int n);
```

```
1    void imprime(int *m,int n){
2        int i;
3        for (i=0; i< n;i++)
4            printf("%d \n", m[i]);
5    }
6
7    int main(){
8        int n[5] = {1,2,3,4,5};
9        imprime(n,5);
10       return 0;
11   }
```

```c
void imprime(int *m,int n){
    int i;
    for (i=0; i< n;i++)
        printf("%d \n", m[i])
    ;
}

int main(){
    int n[5] = {1,2,3,4,5};
--> PAROU AQUI <--
    imprime(n,5);
    return 0;
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			
31			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

```c
void imprime(int *m,int n){
--> PAROU AQUI <--
    int i;
    for (i=0; i< n;i++)
      printf("%d \n", m[i]);
}

int main(){
    int n[5] = {1,2,3,4,5};
    imprime(n,5); // ENTROU
    return 0;
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			
31			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	?	m	int*
49			
50			
51			
52	?	n	int
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

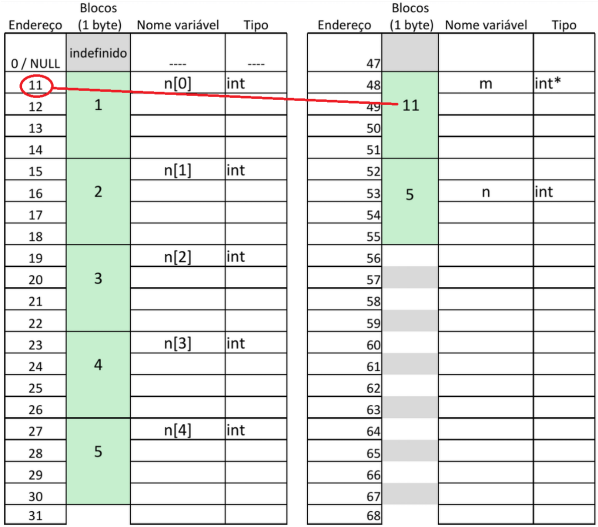
```c
void imprime(int *m,int n){
--> PAROU AQUI <--
    int i;
    for (i=0; i< n;i++)
      printf("%d \n", m[i]);
}

int main(){
    int n[5] = {1,2,3,4,5};
    imprime(n,5); // ENTROU
    return 0;
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			
31			

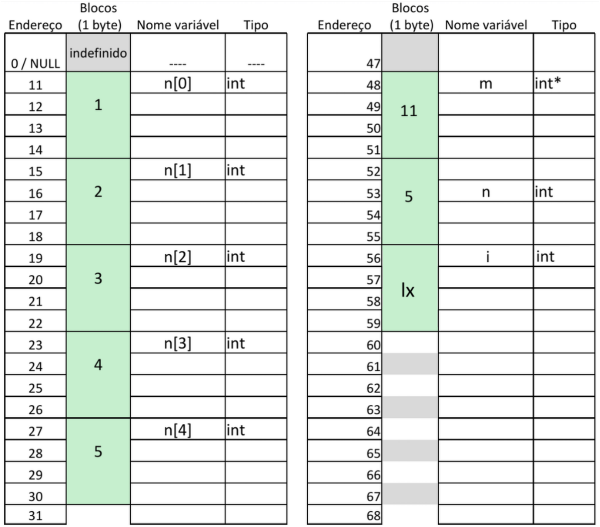
Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	11	m	int*
49			
50			
51			
52	5	n	int
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

```
1  void imprime(int *m,int n){
2  --> PAROU AQUI <--
3      int i;
4      for (i=0; i< n;i++)
5        printf("%d \n", m[i]);
6  }
7
8  int main(){
9      int n[5] = {1,2,3,4,5};
10     imprime(n,5); // ENTROU
11     return 0;
12 }
```



```c
void imprime(int *m,int n){
    int i;
    for (i=0; i< n;i++)
--> PAROU AQUI <--
        printf("%d \n", m[i]);
}

int main(){
    int n[5] = {1,2,3,4,5};
    imprime(n,5); // ENTROU
    return 0;
}
```



```c
void imprime(int *m,int n){
    int i;
    for (i=0; i< n;i++)
--> PAROU AQUI <--
        printf("%d \n", m[i]);
}

int main(){
    int n[5] = {1,2,3,4,5};
    imprime(n,5); // ENTROU
    return 0;
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			
31			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	11	m	int*
49			
50			
51			
52	5	n	int
53			
54			
55			
56	0	i	int
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

```c
void imprime(int *m,int n){
    int i;
    for (i=0; i< n;i++)
// m[0] => end.  11
      printf("%d \n", m[i]);
// m[0] => valor:  1
}

int main(){
    int n[5] = {1,2,3,4,5};
    imprime(n,5);  // ENTROU
    return 0;
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			
31			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	11	m	int*
49			
50			
51			
52	5	n	int
53			
54			
55			
56	0	i	int
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

```
1  void imprime(int *m,int n){
2      int i;
3      for (i=0; i< n;i++)
4  // m[1] => end.  15
5          printf("%d \n", m[i]);
6  // m[1] => valor:  2
7  }
8
9  int main(){
10     int n[5] = {1,2,3,4,5};
11     imprime(n,5);  // ENTROU
12     return 0;
13 }
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			
31			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	11	m	int*
49			
50			
51			
52	5	n	int
53			
54			
55			
56	1	i	int
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

```c
void imprime(int *m,int n){
    int i;
    for (i=0; i< n;i++)
--> PAROU AQUI <--
        printf("%d \n", m[i]);
}

int main(){
    int n[5] = {1,2,3,4,5};
    imprime(n,5); // ENTROU
    return 0;
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			
31			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	11	m	int*
49			
50			
51			
52	5	n	int
53			
54			
55			
56	1	i	int
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

```
1  void imprime(int *m,int n){
2      int i;
3      for (i=0; i< n;i++)
4  // m[2] => end.  19
5          printf("%d \n", m[i]);
6  // m[2] => valor:  3
7  }
8
9  int main(){
10     int n[5] = {1,2,3,4,5};
11     imprime(n,5);  // ENTROU
12     return 0;
13 }
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			
31			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	11	m	int*
49			
50			
51			
52	5	n	int
53			
54			
55			
56	2	i	int
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

```
1  void imprime(int *m,int n){
2      int i;
3      for (i=0; i< n;i++)
4  // m[3] => end.  23
5          printf("%d \n", m[i]);
6  // m[3] => valor:  4
7  }
8
9  int main(){
10     int n[5] = {1,2,3,4,5};
11     imprime(n,5);  // ENTROU
12     return 0;
13 }
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			
31			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	11	m	int*
49			
50			
51			
52	5	n	int
53			
54			
55			
56	3	i	int
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

Arrays como parâmetros

- Por que precisamos passar o tamanho do vetor?

Arrays como parâmetros

- Por que precisamos passar o tamanho do vetor?
- Poderíamos usar `sizeof` para descobrir o tamanho?

Arrays como parâmetros

- Por que precisamos passar o tamanho do vetor?
- Poderíamos usar `sizeof` para descobrir o tamanho?

```
1 void imprime(int *m,int n){
2  int i;
3  for (i=0; i< n;i++)
4  printf ("%d \n", m[i]);
5 }
6
7 int main(){
8    int n[5] = {1,2,3,4,5};
9    imprime(n,5);
10   return 0;
11 }
```

Arrays como parâmetros

- Por que precisamos passar o tamanho do vetor?
- Poderíamos usar `sizeof` para descobrir o tamanho?

```
1  void imprime(int *m,int n){
2   int i;
3   for (i=0; i< n;i++)
4   printf ("%d \n", m[i]);
5  }
6
7  int main(){
8     int n[5] = {1,2,3,4,5};
9     imprime(n,5);
10    return 0;
11 }
```

na main()

sizeof(n) => 20 (4 bytes x 5)
sizeof(int) => 4
sizeof(n) / sizeof(int) => 20/4 = 5

Arrays como parâmetros

- Por que precisamos passar o tamanho do vetor?
- Poderíamos usar `sizeof` para descobrir o tamanho?

```
1 void imprime(int *m,int n){
2  int i;
3  for (i=0; i< n;i++)
4  printf ("%d \n", m[i]);
5 }
6
7 int main(){
8    int n[5] = {1,2,3,4,5};
9    imprime(n,5);
10   return 0;
11 }
```

na função imprime()

sizeof(m) => 4 ou 8 (32 ou 64 bits)
sizeof(int) => 4
sizeof(m) / sizeof(int) => 4/4 = 1

na main()

sizeof(n) => 20 (4 bytes x 5)
sizeof(int) => 4
sizeof(n) / sizeof(int) => 20/4 = 5

Arrays como parâmetros

- Foi visto que para arrays, não é necessário especificar o número de elementos para a função.

```
void imprime (int *m, int n);
void imprime (int m[], int n);
```

Arrays como parâmetros

- Foi visto que para arrays, não é necessário especificar o número de elementos para a função.

```
void imprime (int *m, int n);
void imprime (int m[], int n);
```

- No entanto, para arrays com mais de uma dimensão, é necessário especificar o tamanho de todas as dimensões, exceto a primeira

```
void imprime (int m[][5], int n);
```

Arrays como parâmetros

- Na passagem de um array para uma função, o compilador precisa saber o tamanho de cada elemento, não o número de elementos.

Arrays como parâmetros

- Na passagem de um array para uma função, o compilador precisa saber o tamanho de cada elemento, não o número de elementos.

- Uma matriz pode ser interpretada como um array de arrays.
  `int m[4][5]`: array de 4 elementos onde cada elemento é um array de 5 posições inteiras.

Arrays como parâmetros

- Com isso, o compilador precisa saber o tamanho de cada elemento do array.

```
int m[4][5];
void imprime (int m[][5], int n);
```

Arrays como parâmetros

- Com isso, o compilador precisa saber o tamanho de cada elemento do array.

```
int m[4][5];
void imprime (int m[][5], int n);
```

- Na notação acima,foi informado ao compilador que estamos passando um array, onde cada elemento dele é outro array de 5 posições inteiras.

Arrays como parâmetros

- Isso é necessário para que o programa saiba que o array possui mais de uma dimensão e mantenha a notação de um conjunto de colchetes por dimensão.

Arrays como parâmetros

- Isso é necessário para que o programa saiba que o array possui mais de uma dimensão e mantenha a notação de um conjunto de colchetes por dimensão.

- As notações abaixo funcionam para arrays com mais de uma dimensão. Mas o array é tratado como se tivesse apenas uma dimensão dentro da função:

```
void imprime (int *m, int n);
void imprime (int m[], int n);
```

FUNÇÕES

Struct como parâmetro

Struct como parâmetro

- Quando se trabalhar com estruturas pode-se passar para a função:
  - um campo:
    - por valor
    - por referência
  - toda a estrutura:
    - por valor
    - por referência

Struct como parâmetro

Passando um **campo** por **valor**

```c
#include <stdio.h>

struct ponto {
   int x, y;
};

void imprime_valor(int n) {
   printf("Valor = %d\n", n);
}

int main(){
   struct ponto p1 = {10, 20};
   imprime_valor(p1.x);
   imprime_valor(p1.y);
   return 0;
}
```

Saída:

Valor = ?

Valor = ?

Struct como parâmetro

Passando um **campo** por **valor**

```c
#include <stdio.h>

struct ponto {
   int x, y;
};

void imprime_valor(int n) {
   printf("Valor = %d\n", n);
}

int main(){
   struct ponto p1 = {10, 20};
   imprime_valor(p1.x);
   imprime_valor(p1.y);
   return 0;
}
```

Saída:

Valor = 10
Valor = 20

Struct como parâmetro

Passando um **campo** por **referência**

```c
#include <stdio.h>
struct ponto {
    int x, y;
};

void imprime_valor(int *n) {
    *n = *n + 1;
    printf("Valor = %d\n", *n);
}

int main(){
    struct ponto p1 = {10, 20};
    imprime_valor(&p1.x);
    imprime_valor(&p1.y);
    return 0;
}
```

Saída:

Valor = ?

Valor = ?

Struct como parâmetro

Passando um **campo** por **referência**

```c
#include <stdio.h>
struct ponto {
    int x, y;
};

void imprime_valor(int *n) {
    *n = *n + 1;
    printf("Valor = %d\n", *n);
}

int main(){
    struct ponto p1 = {10, 20};
    imprime_valor(&p1.x);
    imprime_valor(&p1.y);
    return 0;
}
```

Saída:

Valor = 11

Valor = 21

Struct como parâmetro

Passando um **struct** por **valor**

```c
#include <stdio.h>
struct ponto {
    int x, y;
};

void imprime_valor(struct ponto p) {
    printf("x = %d\n", p.x);
    printf("y = %d\n", p.y);
}

int main(){
    struct ponto p1 = {10, 20};
    imprime_valor(p1);
    return 0;
}
```

Saída:

x = 10

y = 20

Struct como parâmetro

Passando um **struct** por **referência**

```c
#include <stdio.h>
struct ponto {
    int x, y;
};

void atribui(struct ponto *p) {
    (*p).x = 10;
    (*p).y = 20;;
}

int main(){
    struct ponto p1;
    atribui(&p1);
    printf("x = %d\n", p1.x);
    printf("y = %d\n", p1.y);
    return 0;
}
```

Saída:

x = 10

y = 20

Struct como parâmetro

Passando um **struct** por **referência**

```c
#include <stdio.h>
struct ponto {
    int x, y;
};

void atribui(struct ponto *p) {
    (*p).x = 10;
    (*p).y = 20;;
}

int main(){
    struct ponto p1;
    atribui(&p1);
    printf("x = %d\n", p1.x);
    printf("y = %d\n", p1.y);
    return 0;
}
```

Atenção

É preciso colocar parênteses.
Caso contrário, o compilador entenderá `*p.x` como `*(p.x)`

Struct como parâmetro

Passando um **struct** por **referência**

```c
#include <stdio.h>
struct ponto {
    int x, y;
};

void atribui(struct ponto *p) {
    (*p).x = 10;
    *p.x = 10;  // ERRADO
    *(p.x) = 10;  // ERRADO
}
```

Reforçando...

Ao acessar uma estrutura passada por referência, não podemos esquecer de colocar parênteses antes de acessar o seu campo.

Struct como parâmetro

Passando um **struct** por **referência**

```c
#include <stdio.h>
struct ponto {
    int x, y;
};

void atribui(struct ponto *p) {
    (*p).x = 10;
    (*p).y = 20;
}

void atribui_2(struct ponto *p) {
    p->x = 10;
    p->y = 20;
}
```

Lembre-se...

Podemos utilizar o operador -> (“SETA”) para acessar o campo de uma estrutura passada por referência

Faculdade de Computação - FACOM

Bacharelado em Sistemas de Informação

Prof. Thiago Pirola Ribeiro