LPG0001 – Linguagem de Programação

Introdução a Ponteiros e Passagem de Parâmetros por Referência

Prof. Rui Jorge Tramontin Junior
Departamento de Ciência da Computação
UDESC / Joinville

Introdução

- Em diversas situações, é necessário ter acesso direto a endereços de memória de variáveis e funções;

Introdução

- Em diversas situações, é necessário ter acesso direto a endereços de memória de variáveis e funções;

- Um **ponteiro** (também chamado de *apontador*) é uma variável cujo conteúdo é um **endereço de memória**;

Introdução

- Em diversas situações, é necessário ter acesso direto a endereços de memória de variáveis e funções;

- Um **ponteiro** (também chamado de *apontador*) é uma variável cujo conteúdo é um **endereço de memória**;

- Quando um ponteiro contém o endereço de outra variável, normalmente diz-se que o mesmo **aponta** para a variável.

Introdução

- Um ponteiro pode apontar para:

Introdução

- Um ponteiro pode apontar para:
  - Uma variável;

Introdução

- Um ponteiro pode apontar para:
  - Uma variável;
  - Outro ponteiro;

Introdução

- Um ponteiro pode apontar para:
  - Uma variável;
  - Outro ponteiro;
  - Uma área de memória alocada dinamicamente;

Introdução

- Um ponteiro pode apontar para:
  - Uma variável;
  - Outro ponteiro;
  - Uma área de memória alocada dinamicamente;
  - Uma função.

Situações Típicas de Uso de Ponteiros

- Passagem de parâmetros por referência;

Situações Típicas de Uso de Ponteiros

- Passagem de parâmetros por referência;

- Alocação Dinâmica de Memória:

Situações Típicas de Uso de Ponteiros

- Passagem de parâmetros por referência;

- Alocação Dinâmica de Memória:
  - Vetores (*strings*);

Situações Típicas de Uso de Ponteiros

- Passagem de parâmetros por referência;

- Alocação Dinâmica de Memória:
  - Vetores (*strings*);
  - Matrizes (vetores de *strings*);

Situações Típicas de Uso de Ponteiros

- Passagem de parâmetros por referência;

- Alocação Dinâmica de Memória:
  - Vetores (*strings*);
  - Matrizes (vetores de *strings*);

- *Polimorfismo*, com ponteiros para funções.

Declaração de Ponteiros

- Em C, ponteiros são tipados:
  - Tem o endereço e o tipo de dados;

Declaração de Ponteiros

- Em C, ponteiros são tipados:
  - Tem o endereço e o tipo de dados;

- Na declaração, utiliza-se um * para indicar que a variável é um ponteiro;

Declaração de Ponteiros

- Em C, ponteiros são tipados:
  - Tem o endereço e o tipo de dados;

- Na declaração, utiliza-se um * para indicar que a variável é um ponteiro;

- Exemplos:

```
int *p1;   // ponteiro para int
float *p2; // ponteiro para float
```

Operadores sobre Ponteiros

- Há dois operadores unários:

Operadores sobre Ponteiros

- Há dois operadores unários:

  & : aplicado sobre qualquer variável, recupera o seu endereço;

Operadores sobre Ponteiros

- Há dois operadores unários:

  & : aplicado sobre qualquer variável, recupera o seu endereço;

  * : aplicado sobre um ponteiro, acessa o valor apontado por ele (operador de *indireção*).

EXEMPLO 1: ENDEREÇO DE UMA VARIÁVEL

Exemplo 1: endereço de uma variável

```
int a = 5;
```

Exemplo 1: endereço de uma variável

```
int a = 5;

// Mostra valor da variável a.
printf("A = %d\n", a );
```

Exemplo 1: endereço de uma variável

```
int a = 5;

// Mostra valor da variável a.
printf("A = %d\n", a );

// Mostra endereço da variável a.
printf("Endereço de A = %d\n", &a );
```

Exemplo 1: endereço de uma variável

```
int a = 5;

// Mostra valor da variável a.
printf("A = %d\n", a );

// Mostra endereço da variável a.
printf("Endereço de A = %d\n", &a );

// Mostra endereço da variável a (em hexa)
printf("Endereço de A = %p\n", &a );
```

Mais portável! Nem todo compilador aceita "%d".

EXEMPLO 2: USANDO UM PONTEIRO

Exemplo 2: usando um ponteiro

```
int a = 5;
```

Exemplo 2: usando um ponteiro

```
int a = 5;
```

Modelo da Memória

Exemplo 2: usando um ponteiro

```
int a = 5;

int *p = &a; // ?
```

Modelo da Memória

Exemplo 2: usando um ponteiro

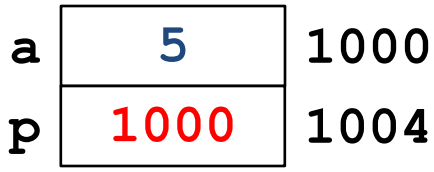
```
int a = 5;

int *p = &a; // Endereço de a.
```

Modelo da Memória

Exemplo 2: usando um ponteiro

```
int a = 5;

int *p = &a; // Endereço de a.

// Mostra valor da variável a.
printf("A = %d\n", *p );
```

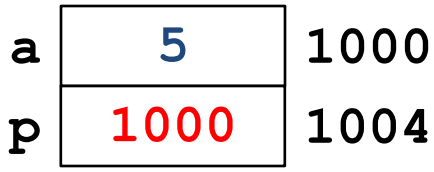
Modelo da Memória



Exemplo 2: usando um ponteiro

```
int a = 5;

int *p = &a; // Endereço de a.

// Mostra valor da variável a.
printf("A = %d\n", *p );

// Mostra endereço da variável a.
printf("Endereço de A = %d\n", p );
```

Modelo da Memória



Exemplo 2: usando um ponteiro

```
int a = 5;

int *p = &a; // Endereço de a.

// Mostra valor da variável a.
printf("A = %d\n", *p );

// Mostra endereço da variável a.
printf("Endereço de A = %d\n", p );

// Mostra endereço da variável a (em hexa)
printf("Endereço de A = %p\n", p );
```

Modelo da Memória

EXEMPLO 3: MAIS PONTEIROS

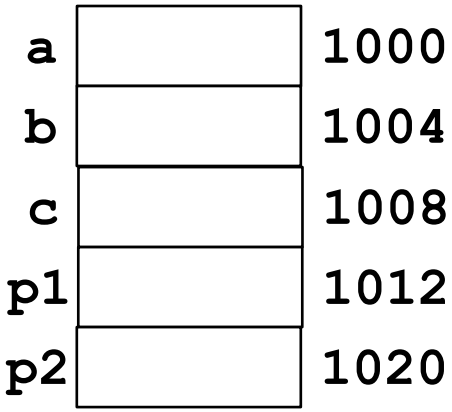
Exemplo 3: mais ponteiros

```
int a, b, c, *p1, *p2;
```

Exemplo 3: mais ponteiros

```
int a, b, c, *p1, *p2;
```

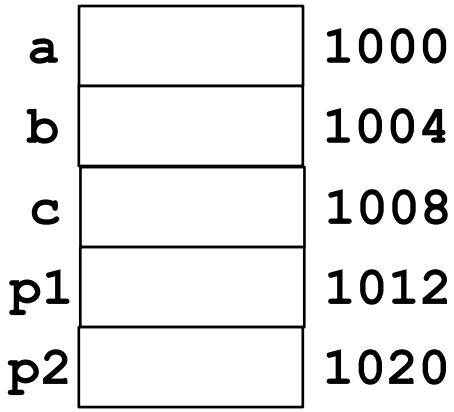
Modelo da Memória



Exemplo 3: mais ponteiros

```
int a, b, c, *p1, *p2;

p1 = &a; // ?
```

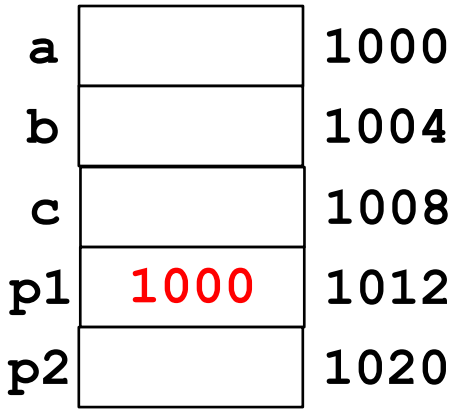
Modelo da Memória



Exemplo 3: mais ponteiros

```
int a, b, c, *p1, *p2;

p1 = &a; // endereço de a.
```

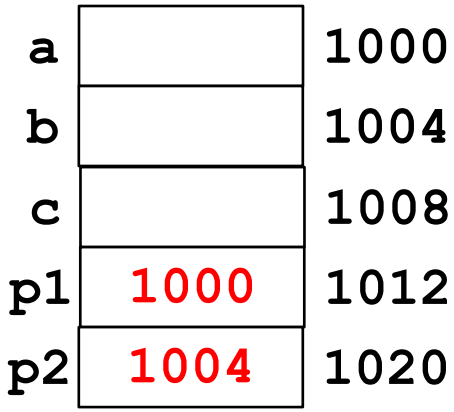
Modelo da Memória



Exemplo 3: mais ponteiros

```
int a, b, c, *p1, *p2;

p1 = &a; // endereço de a.
p2 = &b; // endereço de b.
```

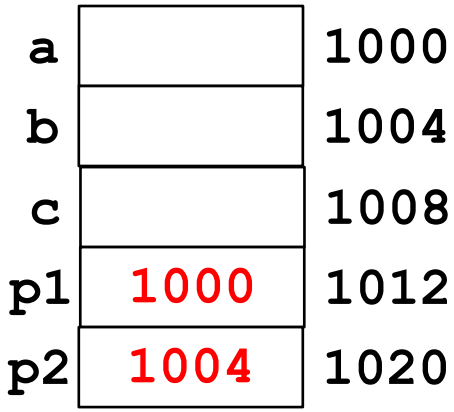
Modelo da Memória



Exemplo 3: mais ponteiros

```
int a, b, c, *p1, *p2;

p1 = &a; // endereço de a.
p2 = &b; // endereço de b.

*p1 = 3; // ?
```

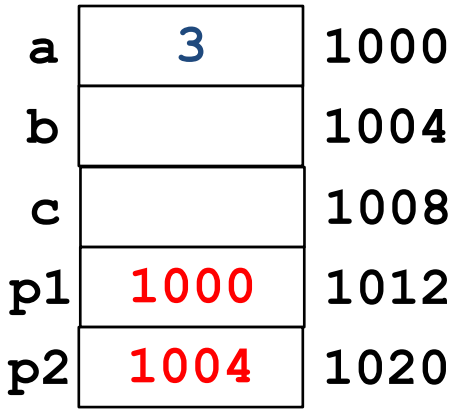
Modelo da Memória



Exemplo 3: mais ponteiros

```
int a, b, c, *p1, *p2;

p1 = &a; // endereço de a.
p2 = &b; // endereço de b.

*p1 = 3; // a = 3;
```

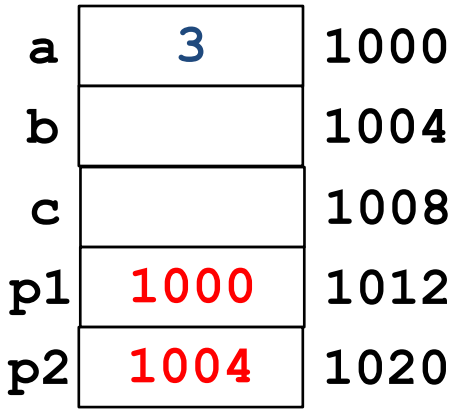
Modelo da Memória



Exemplo 3: mais ponteiros

```
int a, b, c, *p1, *p2;

p1 = &a; // endereço de a.
p2 = &b; // endereço de b.

*p1 = 3; // a = 3;

scanf("%d", p2); // ?
```

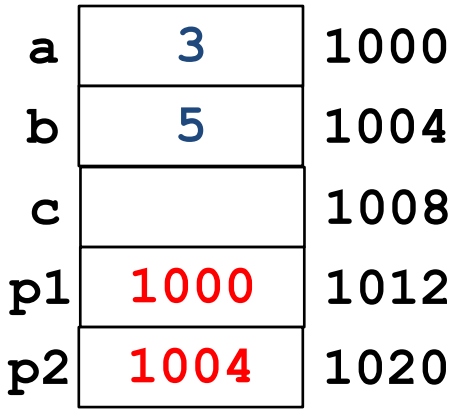
Modelo da Memória



Exemplo 3: mais ponteiros

```
int a, b, c, *p1, *p2;

p1 = &a; // endereço de a.
p2 = &b; // endereço de a.

*p1 = 3; // a = 3;

scanf("%d", p2); // &b (usuário digitou 5)
```

Modelo da Memória



Exemplo 3: mais ponteiros

```
int a, b, c, *p1, *p2;

p1 = &a; // endereço de a.
p2 = &b; // endereço de b.

*p1 = 3; // a = 3;

scanf("%d", p2); // &b (usuário digitou 5)

c = *p1 * *p2;   // ?
```

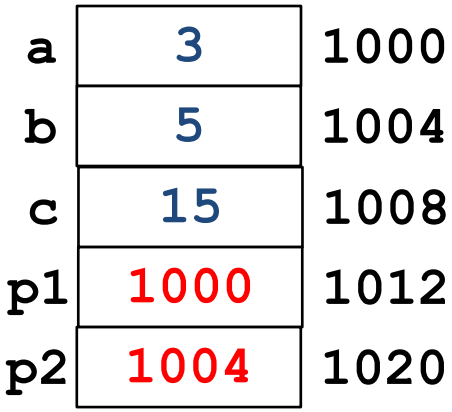
Modelo da Memória

a	3	1000
b	5	1004
c		1008
p1	1000	1012
p2	1004	1020

Exemplo 3: mais ponteiros

```
int a, b, c, *p1, *p2;

p1 = &a; // endereço de a.
p2 = &b; // endereço de b.

*p1 = 3; // a = 3;

scanf("%d", p2); // &b (usuário digitou 5)

c = *p1 * *p2;   // c = a * b;
```

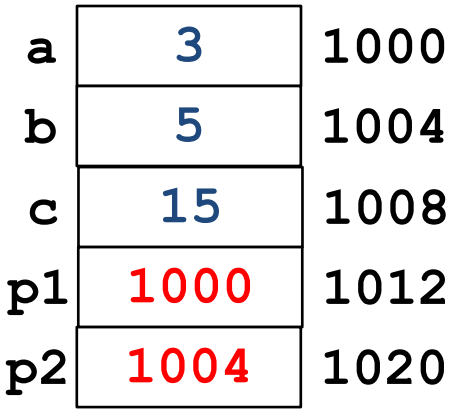
Modelo da Memória



Exemplo 3: mais ponteiros

```
int a, b, c, *p1, *p2;

p1 = &a; // endereço de a.
p2 = &b; // endereço de b.

*p1 = 3; // a = 3;

scanf("%d", p2); // &b (usuário digitou 5)

c = *p1 * *p2;   // c = a * b;

printf("C = %d\n", c); // Mostra c.
```

Modelo da Memória



EXEMPLO 4: CUIDADO COM A INICIALIZAÇÃO

Exemplo 4: cuidado com a inicialização

```
int *p;
```

Exemplo 4: cuidado com a inicialização

```
int *p;


printf("%d\n", *p); // p não foi inicializado!
```

Exemplo 4: cuidado com a inicialização

```
int *p;


printf("%d\n", *p); // p não foi inicializado!
```

- Todo ponteiro não inicalizado aponta para um endereço desconhecido;

Exemplo 4: cuidado com a inicialização

```
int *p;


printf("%d\n", *p); // p não foi inicializado!
```

- Todo ponteiro não inicalizado aponta para um endereço desconhecido;
  - Risco alto do programa travar!

Exemplo 4: cuidado com a inicialização

```
int *p;


printf("%d\n", *p); // p não foi inicializado!
```

- Todo ponteiro não inicalizado aponta para um endereço desconhecido;
  - Risco alto do programa travar!

- É fundamental que todo ponteiro seja inicializado!

Exemplo 4: cuidado com a inicialização

```
int *p;
int n = 5;

printf("%d\n", *p); // p não foi inicializado!
```

- Todo ponteiro não inicalizado aponta para um endereço desconhecido;
  - Risco alto do programa travar!

- É fundamental que todo ponteiro seja inicializado!

Exemplo 4: cuidado com a inicialização

```
int *p;
int n = 5;
p = &n;
printf("%d\n", *p); // agora sim!
```

- Todo ponteiro não inicalizado aponta para um endereço desconhecido;
  - Risco alto do programa travar!

- É fundamental que todo ponteiro seja inicializado!

APLICAÇÃO DE PONTEIROS: PASSAGEM POR REFERÊNCIA

Passagem por Referência

- O uso mais simples de ponteiros consiste no mecanismo chamado **passagem de parâmetro por referência**;

Passagem por Referência

- O uso mais simples de ponteiros consiste no mecanismo chamado **passagem de parâmetro por referência**;

- Em alguns casos, é necessário que uma função modifique uma variável que está fora de seu escopo;

Passagem por Referência

- O uso mais simples de ponteiros consiste no mecanismo chamado **passagem de parâmetro por referência**;

- Em alguns casos, é necessário que uma função modifique uma variável que está fora de seu escopo;

- Tais casos normalmente envolvem situações em que a função precisa retornar mais de um valor.

Passagem por Referência

- Nesses casos, a função recebe como parâmetro o endereço (*referência*) da variável que precisa modificar;

Passagem por Referência

- Nesses casos, a função recebe como parâmetro o endereço (*referência*) da variável que precisa modificar;

- O parâmetro dessa função será, portanto, um ponteiro;

Passagem por Referência

- Nesses casos, a função recebe como parâmetro o endereço (*referência*) da variável que precisa modificar;

- O parâmetro dessa função será, portanto, um ponteiro;

- O exemplo típico de passagem por referência é a função *scanf()*, que recebe o endereço da variável que vai receber o valor lido do teclado:

Passagem por Referência

- Nesses casos, a função recebe como parâmetro o endereço (*referência*) da variável que precisa modificar;

- O parâmetro dessa função será, portanto, um ponteiro;

- O exemplo típico de passagem por referência é a função *scanf()*, que recebe o endereço da variável que vai receber o valor lido do teclado:

```
scanf("%d", &n);
```

EXEMPLO: INCREMENTO DE VARIÁVEL

Incremento de variável na *main()*

```
int main(){




  return 0;
}
```

Incremento de variável na *main()*

```
int main(){
  int n = 1;



  return 0;
}
```

Incremento de variável na *main()*

```
int main(){
  int n = 1;
  printf("%d\n", n); // 1


  return 0;
}
```

Incremento de variável na *main()*

```
int main(){
  int n = 1;
  printf("%d\n", n); // 1
  n++;               // Incrementa n

  return 0;
}
```

Incremento de variável na *main()*

```
int main(){
  int n = 1;
  printf("%d\n", n); // 1
  n++;               // Incrementa n
  printf("%d\n", n); // 2
  return 0;
}
```

INCREMENTO DE VARIÁVEL NUMA FUNÇÃO

Incremento de variável numa função

```
int main(){
  int n = 1;
  printf("%d\n", n);
  inc( n );
  printf("%d\n", n);
  return 0;
}

void inc( int x ){
  x++;
}
```

Incremento de variável numa função

```
int main(){
  int n = 1;
  printf("%d\n", n);
  inc( n );
  printf("%d\n", n);
  return 0;
}

void inc( int x ){
  x++;
}
```

Modelo da Memória

Incremento de variável numa função

```
int main(){
   int n = 1;
   printf("%d\n", n); // 1
   inc( n );
   printf("%d\n", n);
   return 0;
}

void inc( int x ){
   x++;
}
```

Modelo da Memória

Incremento de variável numa função

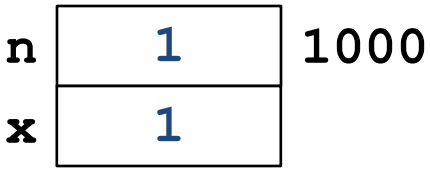
```
int main(){
   int n = 1;
   printf("%d\n", n);
   inc( n );          // chama a função
   printf("%d\n", n);
   return 0;
}

void inc( int x ){
   x++;
}
```

Modelo da Memória

Incremento de variável numa função

```
int main(){
  int n = 1;
  printf("%d\n", n);
  inc( n );
  printf("%d\n", n);
  return 0;
}

void inc( int x ){
  x++;
}
```

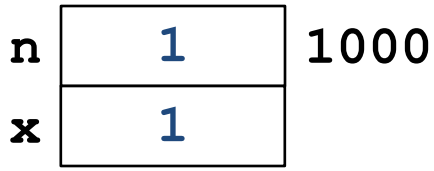
Modelo da Memória



Incremento de variável numa função

```
int main(){
   int n = 1;
   printf("%d\n", n);
   inc( n );
   printf("%d\n", n);
   return 0;
}

void inc( int x ){
   x++;
}
```

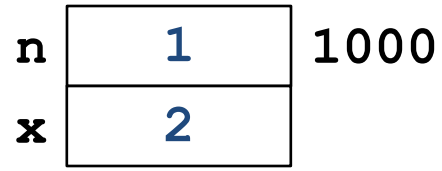
Modelo da Memória



Incremento de variável numa função

```
int main(){
  int n = 1;
  printf("%d\n", n);
  inc( n );
  printf("%d\n", n);
  return 0;
}

void inc( int x ){
  x++;
} // retorna para main()
```

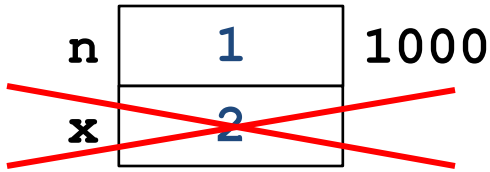
Modelo da Memória



Incremento de variável numa função

```
int main(){
   int n = 1;
   printf("%d\n", n);
   inc( n );
   printf("%d\n", n); // 1
   return 0;
}

void inc( int x ){
   x++;
}
```

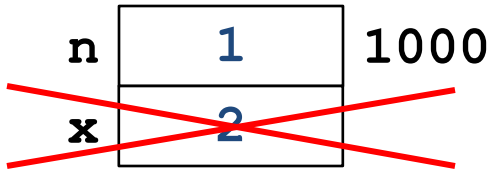
Modelo da Memória



Incremento de variável numa função

```
int main(){
   int n = 1;
   printf("%d\n", n);
   inc( n );
   printf("%d\n", n); // 1
   return 0;
}

void inc( int x ){ // Passagem por valor!
   x++;
}
```

Modelo da Memória



SOLUÇÃO: PASSAGEM POR REFERÊNCIA

Solução: passagem por referência

```
int main(){
   int n = 1;
   printf("%d\n", n);
   inc( &n );           // Passagem por referência
   printf("%d\n", n);
   return 0;
}

void inc( int *x ){ // Parâmetro é ponteiro!
   (*x)++;           // Utiliza * para acessar variável
}
```

Solução: passagem por referência

```
int main(){
   int n = 1;
   printf("%d\n", n);
   inc( &n );
   printf("%d\n", n);
   return 0;
}

void inc( int *x ){
   (*x)++;
}
```

Modelo da Memória

Solução: passagem por referência

```
int main(){
   int n = 1;
   printf("%d\n", n); // 1
   inc( &n );
   printf("%d\n", n);
   return 0;
}

void inc( int *x ){
   (*x)++;
}
```

Modelo da Memória

Solução: passagem por referência

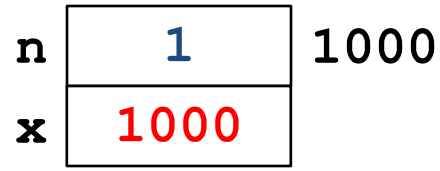
```
int main(){
   int n = 1;
   printf("%d\n", n);
   inc( &n );          // Chama a função
   printf("%d\n", n);
   return 0;
}

void inc( int *x ){
   (*x)++;
}
```

Modelo da Memória

Solução: passagem por referência

```
int main(){
   int n = 1;
   printf("%d\n", n);
   inc( &n );
   printf("%d\n", n);
   return 0;
}

void inc( int *x ){
   (*x)++;
}
```

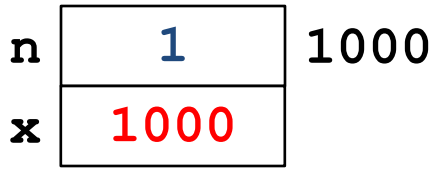
Modelo da Memória



Solução: passagem por referência

```
int main(){
   int n = 1;
   printf("%d\n", n);
   inc( &n );
   printf("%d\n", n);
   return 0;
}

void inc( int *x ){
   (*x)++;    // Incrementa n!
}
```

Modelo da Memória



Solução: passagem por referência

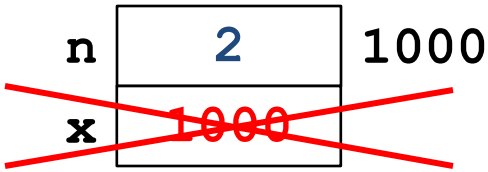
```
int main(){
   int n = 1;
   printf("%d\n", n);
   inc( &n );
   printf("%d\n", n);
   return 0;
}

void inc( int *x ){
   (*x)++;
} // retorna para main()
```

Modelo da Memória

Solução: passagem por referência

```
int main(){
   int n = 1;
   printf("%d\n", n);
   inc( &n );
   printf("%d\n", n); // 2 :-)
   return 0;
}

void inc( int *x ){
   (*x)++;
}
```

Modelo da Memória



MAIS EXEMPLOS

Mais exemplos

- Função que recebe tempo em segundos, e converte para horas, minutos e segundos;

- Função para calcular as raízes de uma equação de 2º grau;

Serão apresentados no Ambiente de Desenvolvimento.