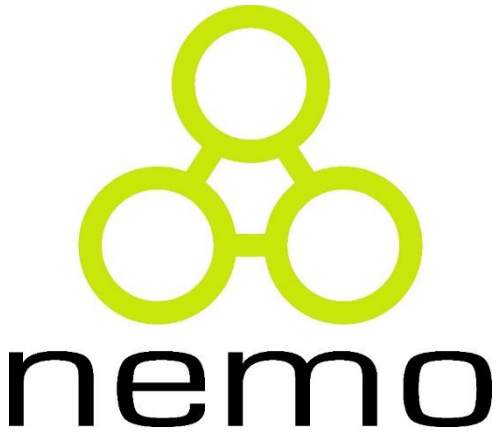


Programação 2

Jordana S. Salamon

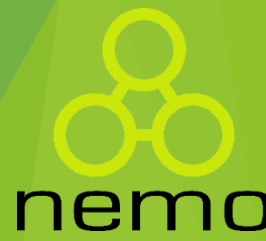
jssalamon@inf.ufes.br

DEPARTAMENTO DE INFORMÁTICA

CENTRO TECNOLÓGICO

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Alocação Dinâmica de Memória



Introdução

- Às vezes precisamos armazenar informações em vetores mas não sabemos a quantidade de informações a armazenar;
- Um ponteiro pode apontar para uma única variável ou para um conjunto de variáveis (vetor, matriz);
- Podemos então declarar um vetor de tamanho indefinido apenas declarando um ponteiro. Ex:
  - **int *v;**
  - **float *vetor;**
- Para efetivamente dizer que esse ponteiro será um vetor, precisamos alocar um espaço na memoria para que ele possa armazenar os dados.

Alocação dinâmica de memória

- Apontadores (ponteiros) permitem alocação de memória em tempo de execução (alocação dinâmica).

- Um ponteiro aponta para uma área de memória livre definida durante a execução do programa.

Função Malloc

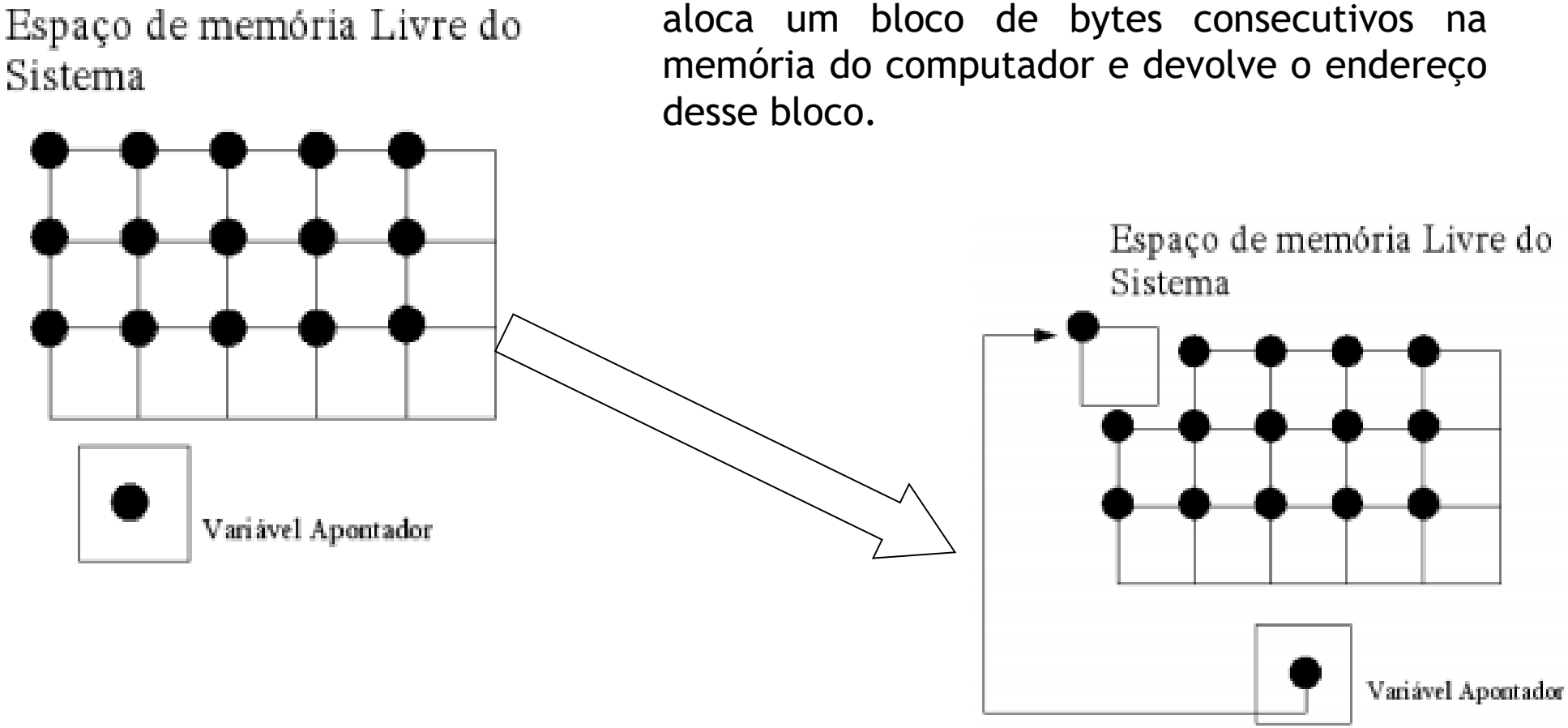
- A biblioteca <stdlib.h> oferece uma função para alocação dinâmica de memória. Veja no exemplo abaixo:

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int i, qtd, *numeros;

    printf("Quantos numeros deseja informar?\n");
    scanf("%d", &qtd);
    numeros = (int *)malloc(qtd * sizeof(int));

    for (i = 0; i < qtd; i++){
        printf("Informe o valor da posicao %d: ", i);
        scanf("%d", &numeros[i]);
    }
}
```

nemo

Função Malloc

A função malloc (abreviatura de memory allocation), da biblioteca padrão stdlib.h, aloca um bloco de bytes consecutivos na memória do computador e devolve o endereço desse bloco.



Função Malloc

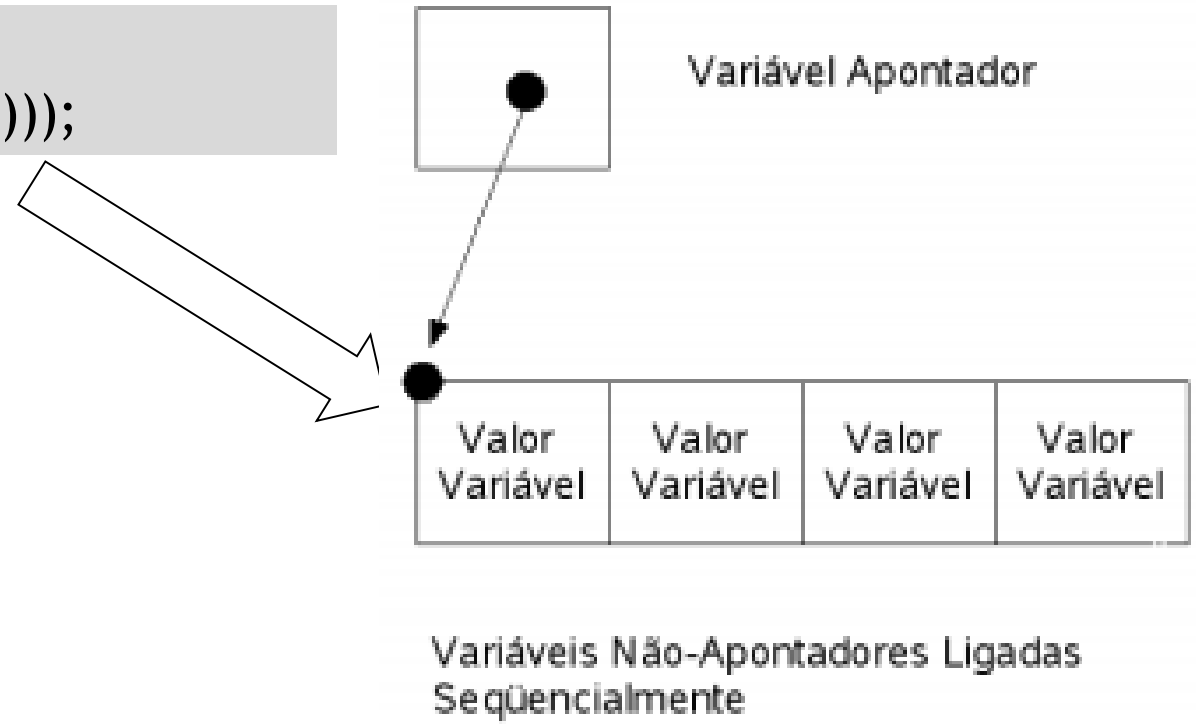
```
numeros = (int *)malloc(qtd * sizeof(int));
```

- A função malloc() recebe como parâmetro o número de bytes a alocar;
- Como não temos certeza de quantos bytes um inteiro ocupa, usamos sizeof(int) para obter esta informação;
- Multiplicamos isso pela quantidade de inteiros que queremos no vetor (qtd);
- A função malloc() retorna um ponteiro genérico (pode ser usada para alocar dinamicamente vetores de qualquer tipo). Por isso precisamos converter o resultado para (int *);

Exemplo de uso da função malloc

- Um espaço de memória para 4 números inteiros é reservado na memória e o endereço inicial é recebido pelo apontador p.

```
1  int *p;
2  p = (int *)malloc (4*(sizeof(int)));
```



nemo

Retornar Vetores em Funções

- Utilizar ponteiro para vetores nos permite retornar vetores em funções, o que antes não era permitido utilizando alocação estática. Ex:

```c
int* criaVetor(int n){
    int *numeros, i;
    numeros = (int *)malloc(n * sizeof(int));
    for (i = 0; i < n; i++){
        printf("Informe o valor da posicao %d: ", i);
        scanf("%d", &numeros[i]);
    }
    return numeros;
}

int main() {
    int i, qtd, *v;
    printf("Quantos numeros deseja informar?\n");
    scanf("%d", &qtd);
    v = criaVetor(qtd);
}
```

nemo

Manipulando Vetores em Funções

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

int main() {
    int i, qtd, *v;
    printf("Quantos numeros deseja informar?\n");
    scanf("%d", &qtd);
    v = criaVetor(qtd);
    imprimeVetor(v,qtd);
}
```

nemo

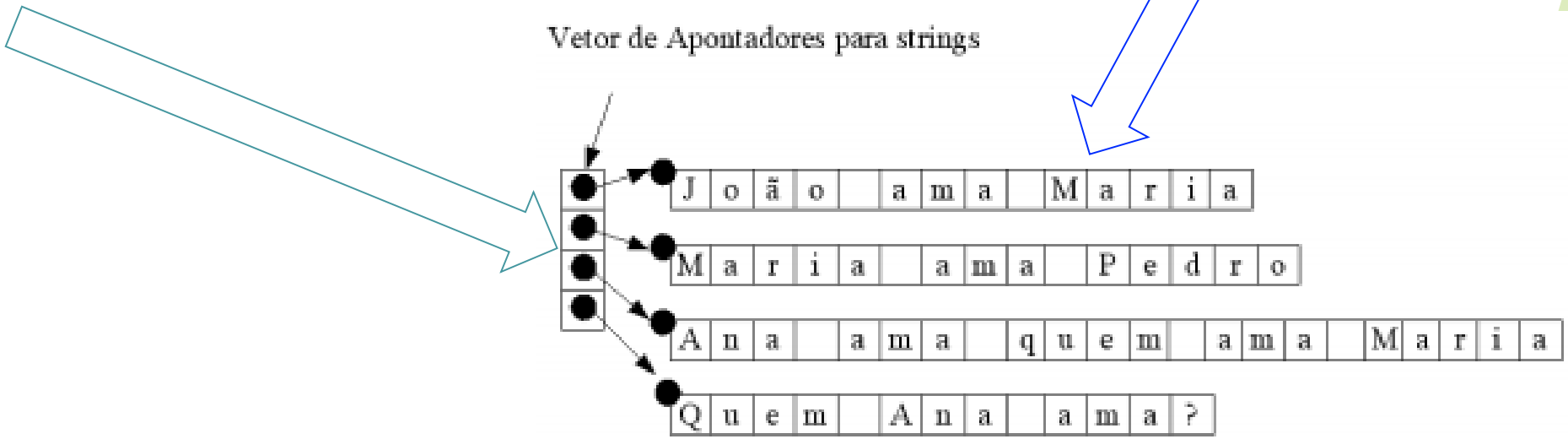
Exemplo interessante - Matrizes

João ama Maria
Maria ama Pedro
Ana ama quem ama Maria
Quem Ana ama?

J	o	ã	o		a	m	a		M	a	r	i	a						
M	a	r	i	a		a	m	a		P	e	d	r	o					
A	n	a		a	m	a		q	u	e	m		a	m	a		M	a	r	i	a
Q	u	e	m		A	n	a		a	m	a	?							

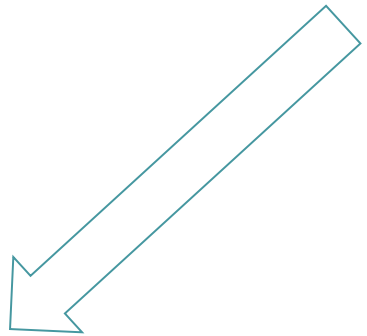
Frases

variáveis apontadoras
endereçando cada frase



nemo

Exemplo interessante - Matrizes



uma outra maneira de manipular strings.
cada elemento do vetor de apontadores para char aponta para uma string.

```
main ( ) {
    char *g[4];

  g[0] = (char *)malloc (strlen ("Joao ama Maria") * sizeof (char));
  strcpy (g[0], "Joao ama Maria");
  g[1] = (char *)malloc (strlen ("Maria ama pedro") * sizeof (char));
  strcpy (g[1], "Maria ama pedro");
  g[2] = (char *)malloc (strlen ("Ana ama quem ama Maria") * sizeof (char));
  strcpy (g[2], "Ana ama quem ama Maria");
  g[3] = (char *)malloc (strlen ("Quem Ana ama?") * sizeof (char));
  strcpy (g[3], "Quem Ana ama?");
}
```

nemo

Outro exemplo interessante - Matrizes

outra maneira de manipular strings: matriz de caracteres alocada dinamicamente.

```
char ** alocaMatriz(int n, int m)
{
      int i;
      mt = (char **) malloc (n*(sizeof(char *)));
      for (i=0; i<n; i++)
         mt[i] = (char *) malloc (m*(sizeof(char)));

      return mt;
}
```

nemo

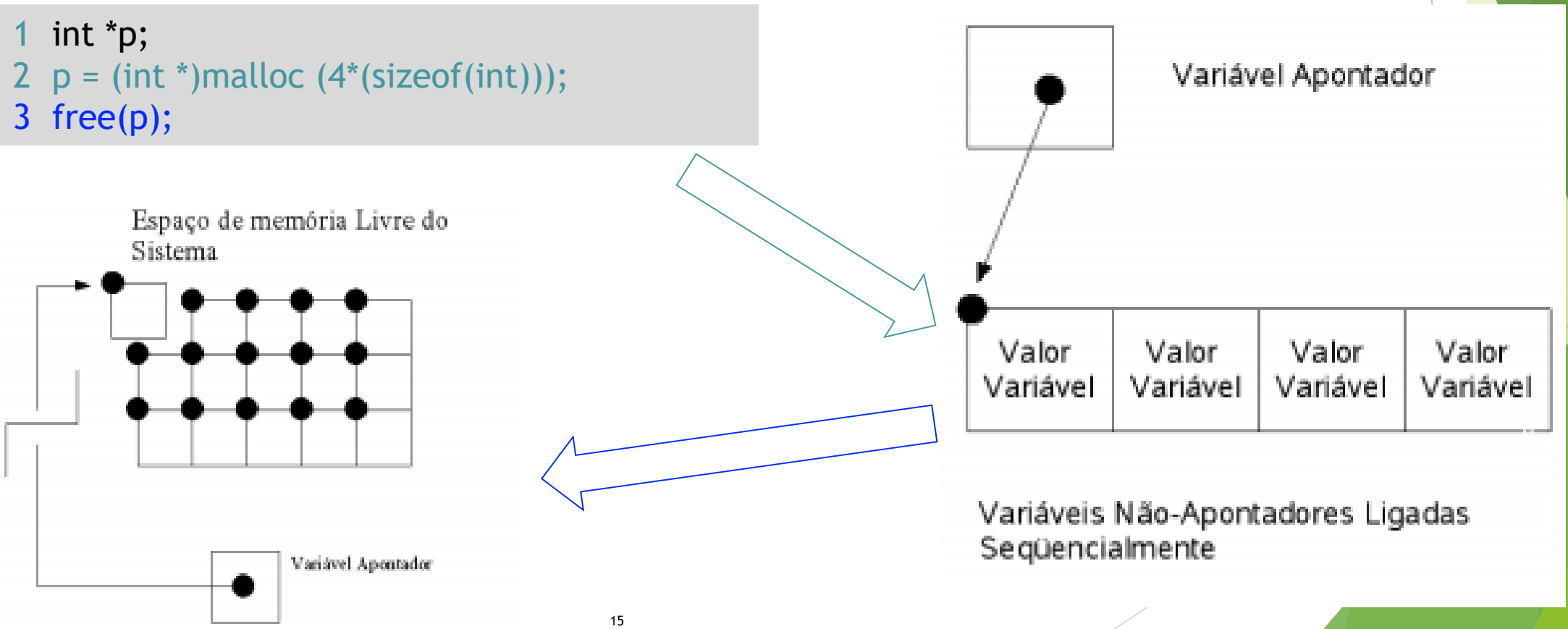
Função Free

- As variáveis alocadas estaticamente dentro de uma função desaparecem assim que a execução da função termina.
- Já as variáveis alocadas dinamicamente continuam a existir mesmo depois que a execução da função termina. Se for necessário liberar a memória ocupada por essas variáveis, é preciso recorrer à função free.
- A função free desaloca a porção de memória alocada por malloc.
- A instrução free (ptr) avisa ao sistema que o bloco de bytes apontado por ptr está disponível para reciclagem. A próxima invocação de malloc poderá tomar posse desses bytes.
- **Não aplique a função free a uma parte de um bloco de bytes alocado por malloc (ou realloc). Aplique free apenas ao bloco todo.**

Função Free

- Um espaço de memória para 4 números inteiros é reservado na memória e o endereço inicial é recebido pelo apontador p.

```
1 int *p;
2 p = (int *)malloc (4*(sizeof(int)));
3 free(p);
```



Função Realloc

- Às vezes é necessário alterar, durante a execução do programa, o tamanho de um bloco de bytes que foi alocado por malloc.

- Isso acontece, por exemplo, durante a leitura de um arquivo que se revela maior que o esperado. Nesse caso, podemos recorrer à função realloc para redimensionar o bloco de bytes.

- A função realloc recebe o endereço de um bloco previamente alocado por malloc (ou por realloc) e o número de bytes que o bloco redimensionado deve ter. A função aloca o novo bloco, copia para ele o conteúdo do bloco original, e devolve o endereço do novo bloco.

- Se o novo bloco for uma extensão do bloco original, seu endereço é o mesmo do original (e o conteúdo do original não precisa ser copiado para o novo). Caso contrário, realloc copia o conteúdo do bloco original para o novo e libera o bloco original (invocando free).

- A propósito, o tamanho do novo bloco pode ser menor que o do bloco original.

Função Realloc

- Suponha, por exemplo, que alocamos um vetor de 1000 inteiros e depois decidimos que precisamos de duas vezes mais espaço:

```
int *v;
v = malloc (1000 * sizeof (int));
for (int i = 0; i < 990; i++)
    scanf ("%d", &v[i]);
v = realloc (v, 2000 * sizeof (int));
for (int i = 990; i < 2000; i++)
    scanf ("%d", &v[i]);
```

Problemas causados por apontadores

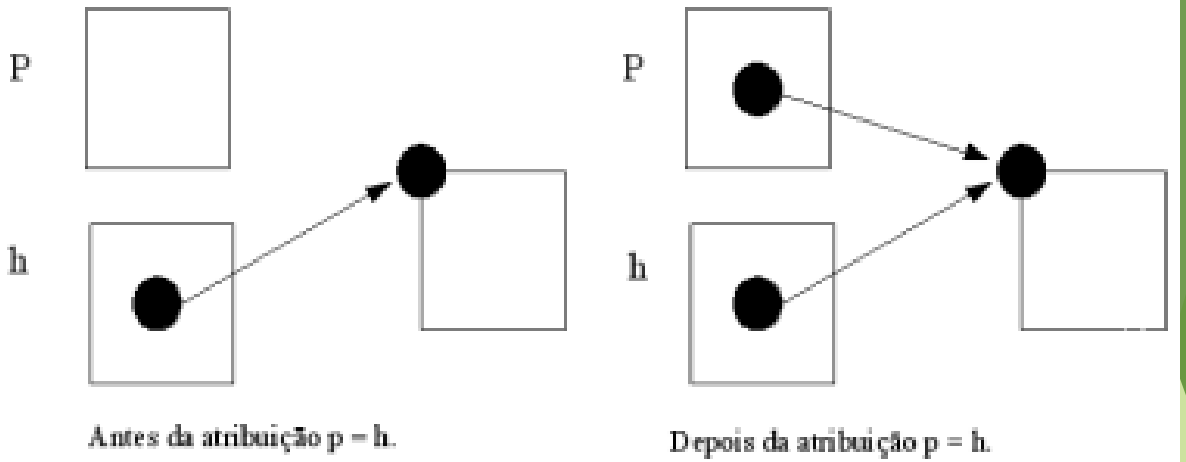
- Apontadores não inicializados
  - um erro muito comum é usar o ponteiro antes de fazê-lo apontar para algum endereço válido, ou seja, sem inicializá-lo.
  - as consequências são imprevisíveis, podendo provocar uma paralisação do sistema (“halt”).

```
1  float *p;
2  float h = 15.0;
3  *p = h;
```

nemo

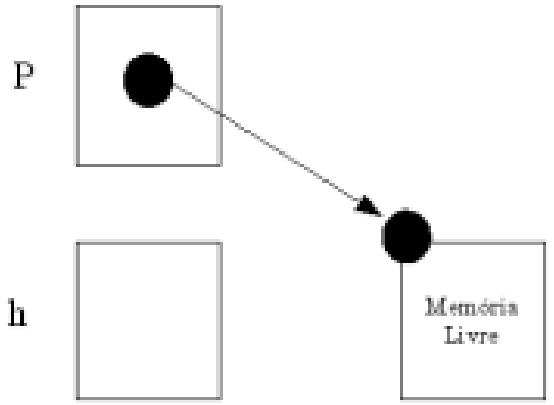
Problemas causados por apontadores

- Referência pendente
  - é liberado um endereço que é apontado por mais de uma variável apontador.



Antes da atribuição p = h.

Depois da atribuição p = h.



Depois do free (h), p aponta para um endereço de memória Livre.

```
1  int *p;
2  int *h;
3  h = (int*)malloc (sizeof (int));
4  p = h;
5  free (h);
```

a área para onde o ponteiro h apontava foi liberada usando a função free(h), mas p continua apontando para lá

nemo

That’s all Folks!