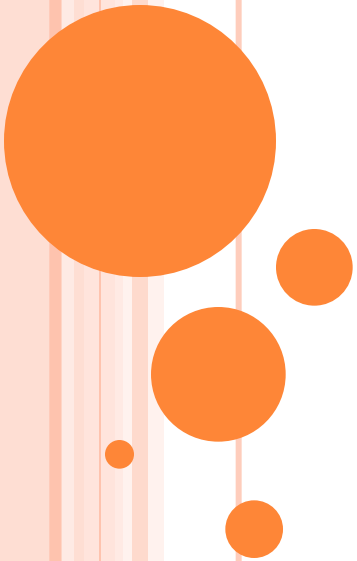


Ponteiros e Listas

Kalinka Regina Lucas Jaquie Castelo Branco

kalinka@icmc.usp.br

Alocação Dinâmica de memória

- Pode-se assumir que as variáveis declaradas na cláusula variável do pseudo-código do algoritmo principal são alocadas uma vez no início da execução e só liberadas ao final da execução. Isso é chamado de alocação estática e gera dois problemas:
  - É preciso estimar corretamente o tamanho máximo que certas variáveis, como vetores e matrizes, vão ocupar.
  - Muito espaço é alocado mas nunca utilizado.
- A área onde variáveis comuns e conjuntos são alocadas é bastante limitada, fazendo com que tipos de dados globais (do algoritmo principal) facilmente preencham o espaço disponível na grande maioria dos sistemas.

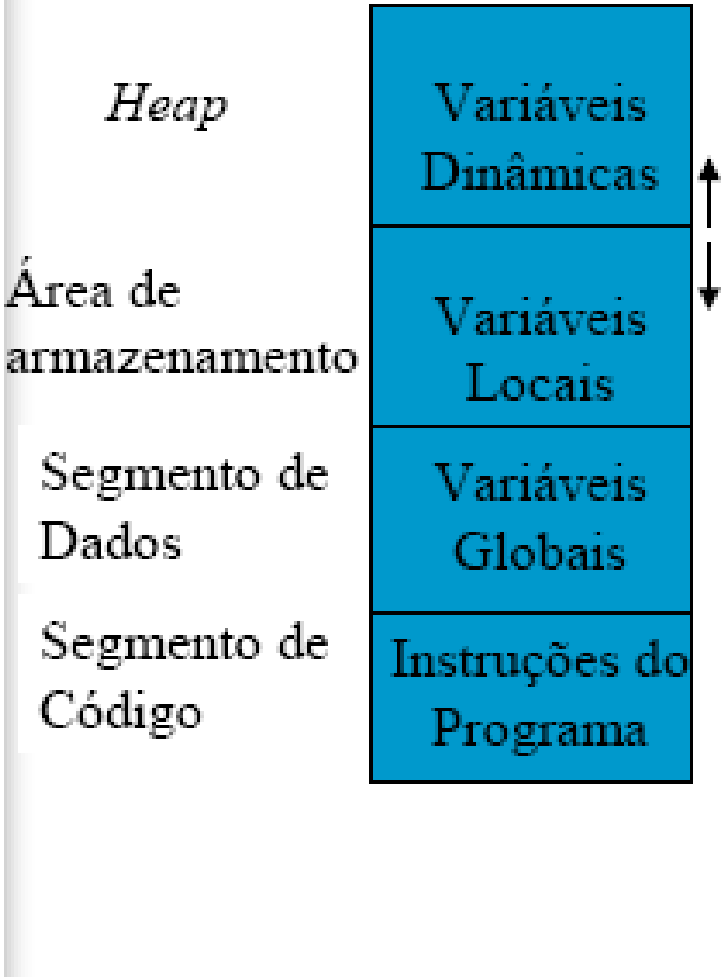
Alocação Dinâmica de Memória

- Uma linguagem de programação costuma gerenciar separadamente regiões de memória interna onde armazena:
  - código;
  - dados;
  - de armazenamento (*stack*) – passagens de parâmetros e variáveis locais.
  - *heap* (uma área de memória, normalmente mais extensa que as demais, para alocação dinâmica).

Alocação Dinâmica de Memória

- Alocar memória dinamicamente significa gerenciar memória (isto é, reservar, utilizar e liberar espaço) durante o tempo de execução. Isso significa que o programador é responsável por controlar a reserva, ocupação e liberação de memória, de forma a ajustar a memória alocada às necessidades de um programa em cada etapa da sua execução.

Mapa de Alocação de Memória para programas no Turbo Pascal



- Todas as variáveis GLOBAIS declaradas até agora (var. simples, vetores, registros, etc.) são ESTÁTICAS, isto é, a memória é alocada pra elas do começo ao fim do programa. As variáveis LOCAIS, também ESTÁTICAS, têm seu espaço alocado quando da chamada à função ou ao procedimento e são liberadas no final deles – tamanho pré-fixado.
- A diferença das variáveis DINÂMICAS é que você pode alocar e dispor de espaço **durante a execução do programa.**

Alocação de Memória Dinâmica

- Vantagens:
  - Expande o espaço de dados total disponível para o programa.
  - Não há necessidade de alocar espaço por estimativa, isto é, não há alocação reservada mas não utilizada (como no caso da maioria dos vetores e matrizes).
  - Uma vez que um dado não é mais utilizado, ele pode ser 'desalocado', liberando memória.

Ponteiros – Introdução

- Ponteiros são variáveis utilizadas para apoiar a alocação dinâmica. Elas armazenam um endereço de memória. Este endereço geralmente é ocupado por um dado (variável) de um determinado tipo;
- Ponteiro tem um tipo associado que determina o tipo do dado que ele é capaz de apontar, isto é, o tipo da variável que seria encontrada no endereço apontado;
- Quando utilizado para variáveis um ponteiro pode ser um ponteiro para um tipo pré-definido da linguagem ou um tipo definido pelo programador.
- Assim, podemos ter ponteiros para *integer*, ponteiro para real, ponteiro para TipoProduto, etc.

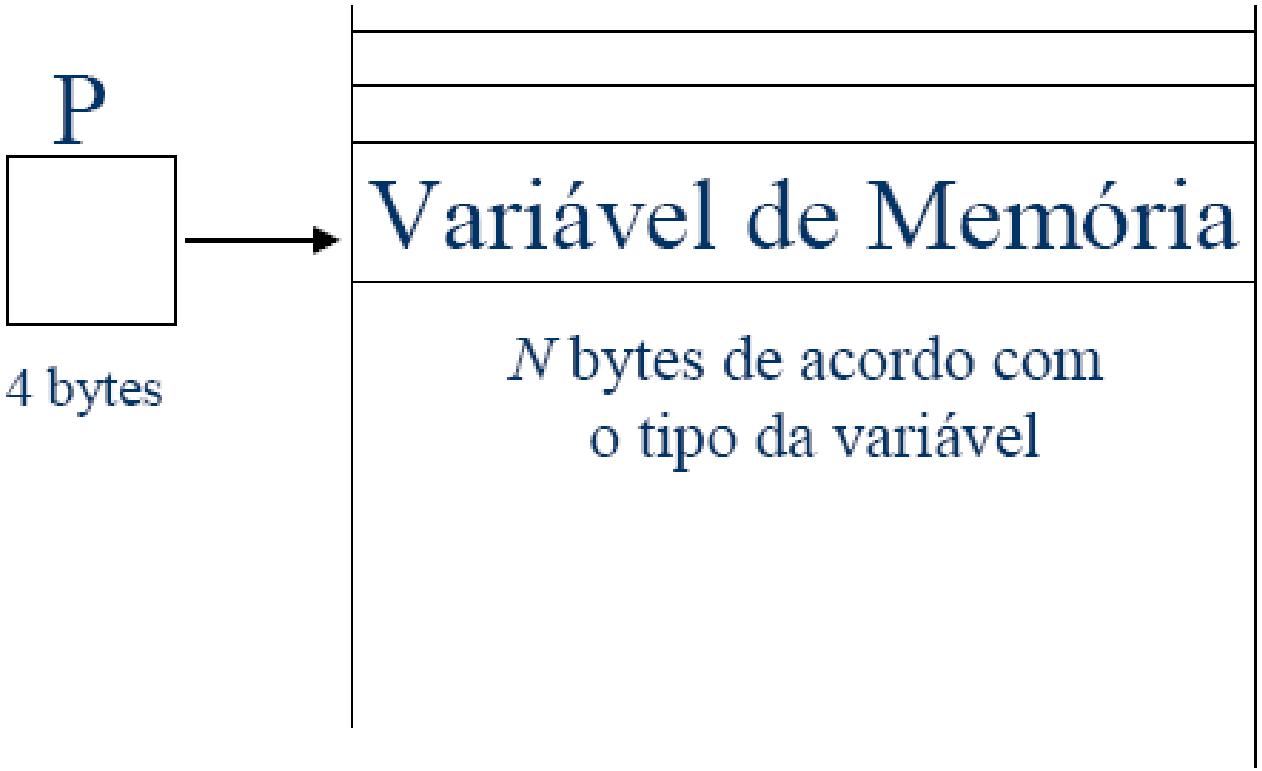
Ponteiros em pseudocódigo: Declaração

- Na linguagem algorítmica há o tipo de dados ponteiro.
- A declaração de um tipo ponteiro se faz com a seguinte sintaxe:
  - tipo nome_do_tipo = >tipo de dado;
  - Ou
    - variável nome_da_variável : >tipo de dado;

Ponteiros

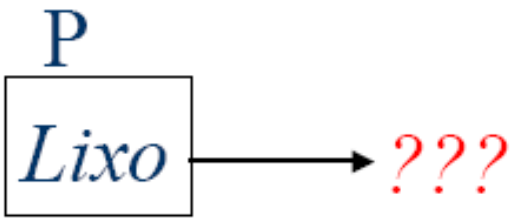
- As variáveis do tipo ponteiro são armazenadas no segmento de dados junto com outras variáveis estáticas do programa.
- Como um ponteiro armazena apenas um endereço de memória, o seu tamanho em bytes é o tamanho necessário para armazenar tal endereço: em geral são usados 4 bytes (o tamanho de um endereço de memória no computador).
- Quando um ponteiro contém o endereço de uma variável, dizemos que o ponteiro está “apontando” para essa variável.

Ponteiros Referenciando Variáveis



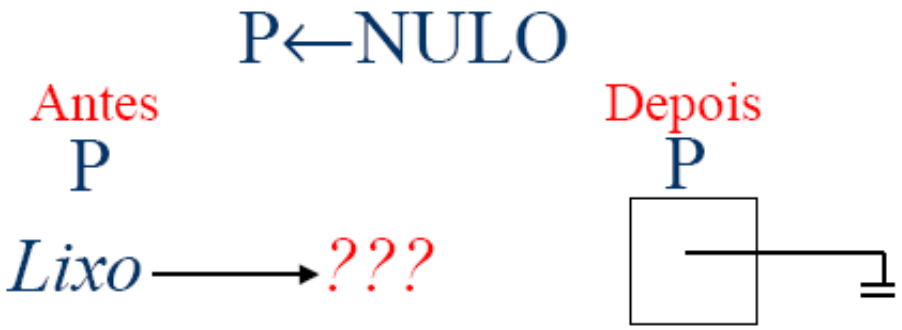
Declaração de Ponteiros

- Ao ser declarado, a reserva de memória para o ponteiro é feita e o espaço alocado pode conter algum "lixo", que aponta para um endereço qualquer de memória.



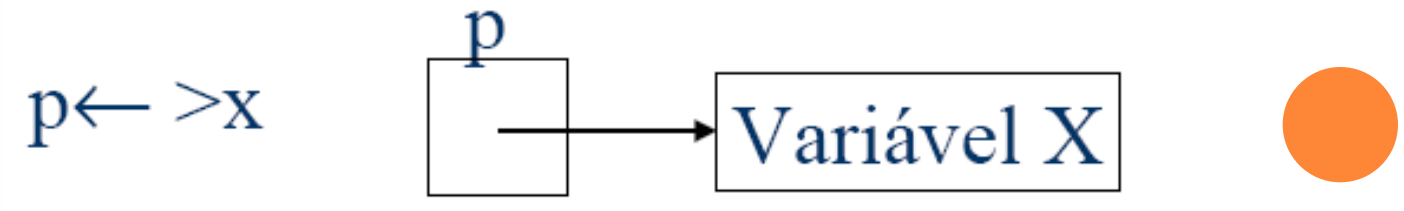
Uso de Ponteiros

- Para evitar tal situação (apontamento para "Lixo"), é necessário fazer com que os ponteiros apontem para o vazio. Essa é a forma de inicializar ponteiros.



Uso de Ponteiros Referenciando Variáveis

- Um ponteiro pode 'apontar' para uma variável existente, isto é, armazenar o seu endereço. Para atribuir um valor para um ponteiro, é necessário indicar que o valor atribuído é um endereço e não o valor mantido pela variável.
- Para isso, utiliza-se o operador '>', que obtém o endereço de uma variável.

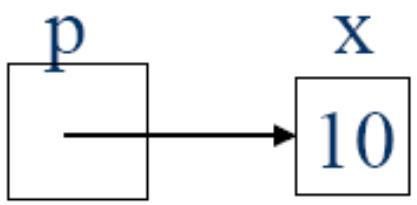


Uso de Ponteiros

- Para armazenar um valor no endereço de memória mantido por um ponteiro, é necessário indicar que o valor é armazenado no endereço apontado e não no próprio ponteiro.
- Para isso, utiliza-se o operador ^, que indica o conteúdo de um endereço.

$p^{\wedge} \leftarrow 10$

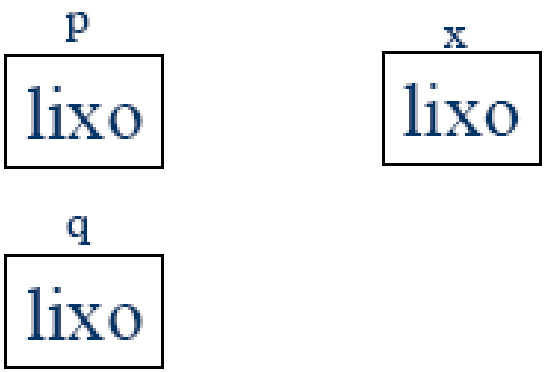
(lê-se conteúdo do endereço apontado por p recebe 10)



Uso em Pseudocódigo

Exemplo

```
variável
x:inteiro
p,q:>inteiro
```

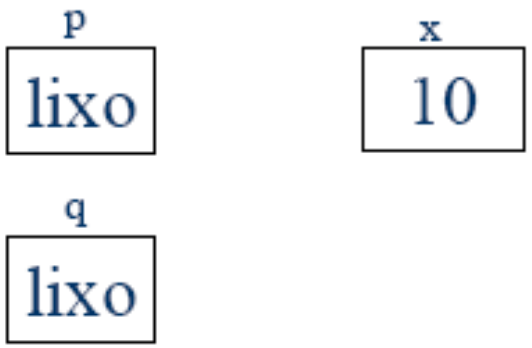



```
x ← 10
p ← >x
q^ ← 30 { ERRO: q aponta p/ lixo }
q ← p
q^ ← p^ + 10
escreva(p^)
```

Uso em Pseudocódigo

Exemplo

```
variável
x:inteiro
p,q:>inteiro
```

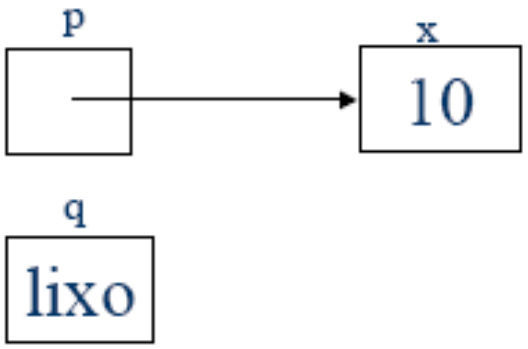



```
x ← 10
p ← >x
q^ ← 30 { ERRO: q aponta p/ lixo }
q ← p
q^ ← p^ + 10
escreva(p^)
```

Uso em Pseudocódigo

Exemplo

```
variável
x:inteiro
p,q:>inteiro
```

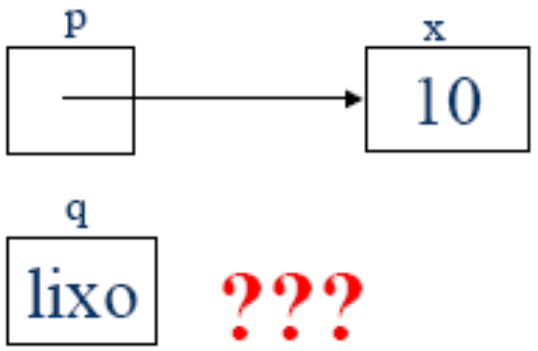



```
x ← 10
p ← >x
q^ ← 30 { ERRO: q aponta p/ lixo }
q ← p
q^ ← p^ + 10
escreva(p^)
```

Uso em Pseudocódigo

Exemplo

```
variável
x:inteiro
p,q:>inteiro
```

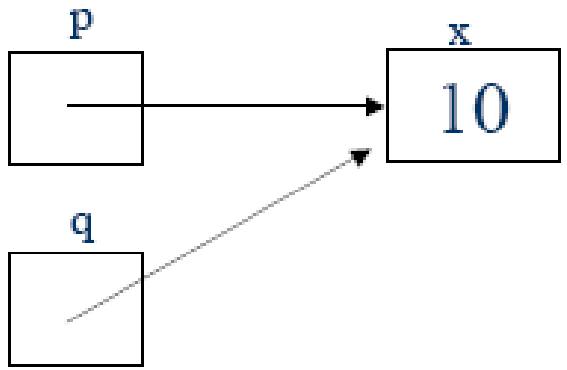



```
x ← 10
p ← >x
q^ ← 30 { ERRO: q aponta p/ lixo }
q ← p
q^ ← p^ + 10
escreva(p^)
```

Uso em Pseudocódigo

Exemplo

```
variável
x:inteiro
p,q:>inteiro
```

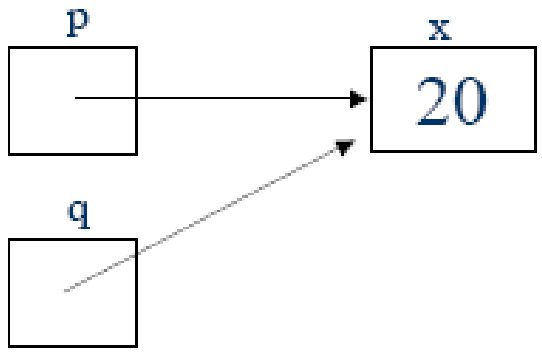



```
x ← 10
p ← >x
q^ ← 30 { ERRO: q aponta p/ lixo }
q ← p
q^ ← p^ + 10
escreva(p^)
```

Uso em Pseudocódigo

Exemplo

```
variável
x:inteiro
p,q:>inteiro
```

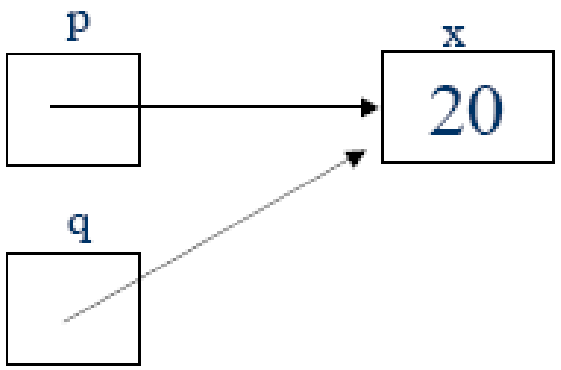



```
x ← 10
p ← >x
q^ ← 30 { ERRO: q aponta p/ lixo }
q ← p
q^ ← p^ + 10
escreva(p^)
```

Uso em Pseudocódigo

Exemplo

```
variável
x:inteiro
p,q:>inteiro
```



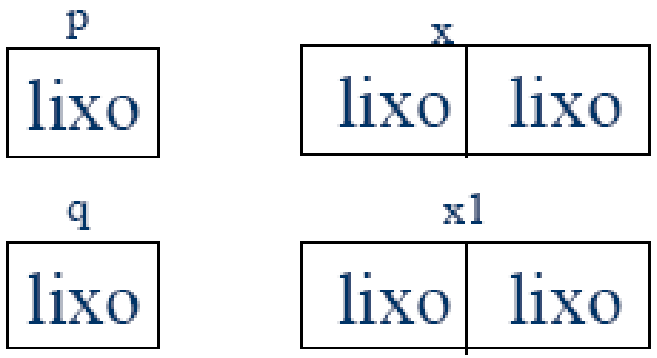

```
x ← 10
p ← >x
q^ ← 30 { ERRO: q aponta p/ lixo }
q ← p
q^ ← p^ + 10
escreva("Resultado = ",p^)
```

Resultado = 20

Uso em Pseudocódigo

Exemplo 2

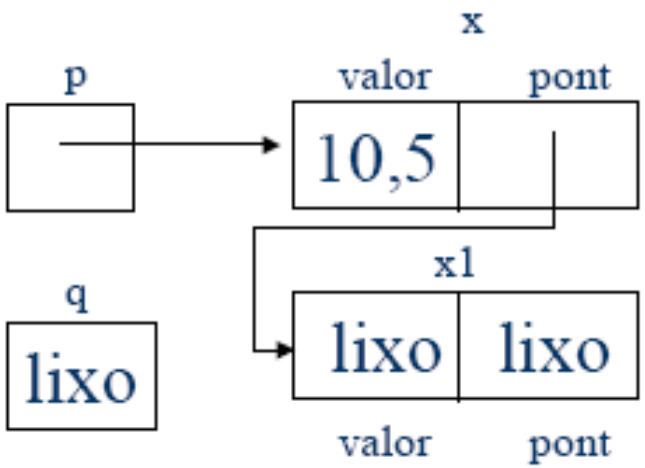
```
tipo
rec = registro
          dado: real
          pont: >rec
      fim registro
variável
x,x1:rec
p,q :>rec
```



Uso em Pseudocódigo

Exemplo 2

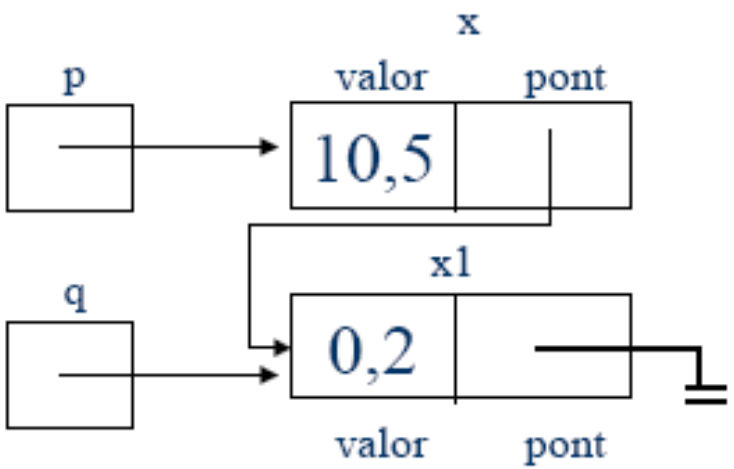
```
variável          x.valor ← 10,5
x,x1:rec          p ← >x
p,q:>rec          x.pont ← >x1
```



Uso em Pseudocódigo

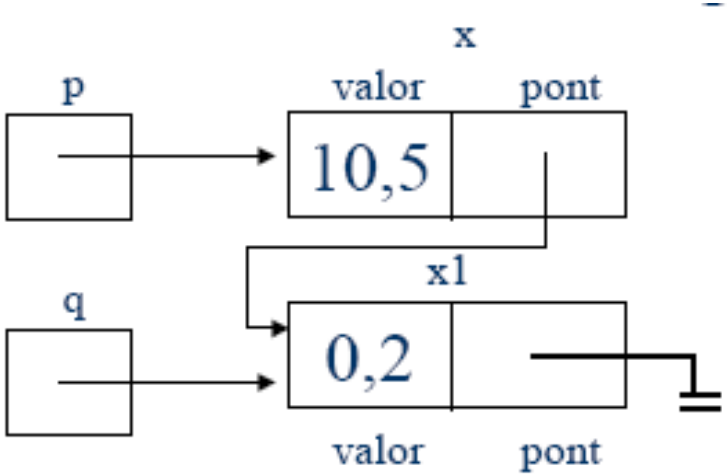
Exemplo 2

```
variável
x,x1:rec
p,q:>rec


x.valor ← 10,5
p ← >x
x.pont ← >x1
x1.valor ← 0,2
q ← x.pont
x1.pont ← NULO
```



Uso em Pseudocódigo

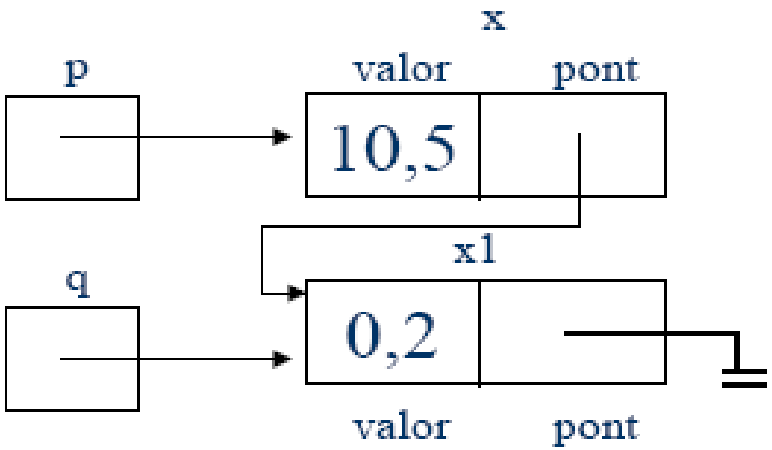
Exemplo 2




```
escreva(p^.valor)
escreva(q^.valor)
escreva(p^.pont^.valor)
escreva(x.valor)
escreva(x.pont^.valor)
escreva(x1.valor)
```

Uso em Pseudocódigo

Exemplo 2




```
escreva(p^.valor)          | 10,5
escreva(q^.valor)          | 0,2
escreva(p^.pont^.valor)    | 0,2
escreva(x.valor)           | 10,5
escreva(x.pont^.valor)     | 0,2
escreva(x1.valor)          | 0,2
```

Uso de Ponteiros para Alocação dinâmica – Pseudocódigo

Alocando espaço na *Heap*

- O uso mais importante de ponteiros é para apoio à alocação dinâmica, isto é, ao invés de apontar variáveis já alocadas do espaço de dados, utilizar o espaço de *heap* para novas variáveis, que podem ser liberadas após o uso, mesmo antes do término do programa.
- Para isso, é necessária uma operação de alocação de memória, e uma para liberação de memória.

Uso de Ponteiros para Alocação dinâmica – Pseudocódigo

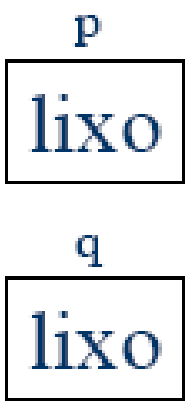
Alocando espaço na *Heap*

- Em pseudocódigo, a operação que aloca memória é implementada por uma função, na forma:
  - aloque(p);
- Esta operação reserva, na *Heap*, espaço suficiente para armazenar um dado do tipo apontado por p. Em p é armazenado o endereço de memória deste dado.
- Em pseudocódigo, a operação que libera memória é implementada por uma função, na forma:
  - libere(p);
    - Esta operação 'retorna' para a *Heap* aquele espaço ocupado pelo dado apontado por p, e anula p.

Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo

```
variável
p,q:>inteiro
```

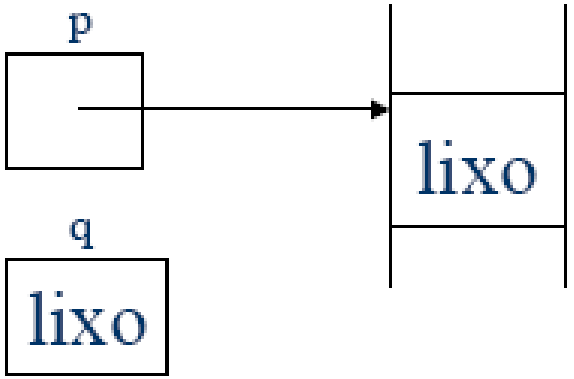



```
aloque(p)
p^ ← 10
q ← p
q^ ← p^ + 10
escreva(p^)
libere(p)
```

Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo

```
variável
p,q:>inteiro
```

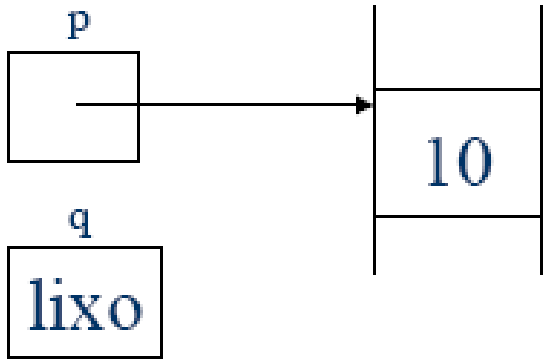



```
aloque(p)
p^ ← 10
q ← p
q^ ← p^ + 10
escreva(p^)
libere(p)
```

Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo

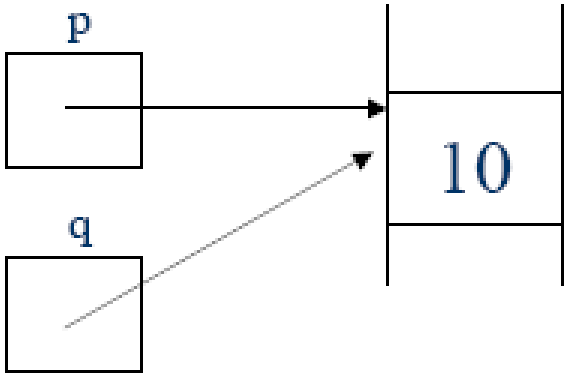
```
variável
p,q:>inteiro


aloque(p)
p^ ← 10
q ← p
q^ ← p^ + 10
escreva(p^)
libere(p)
```



Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo

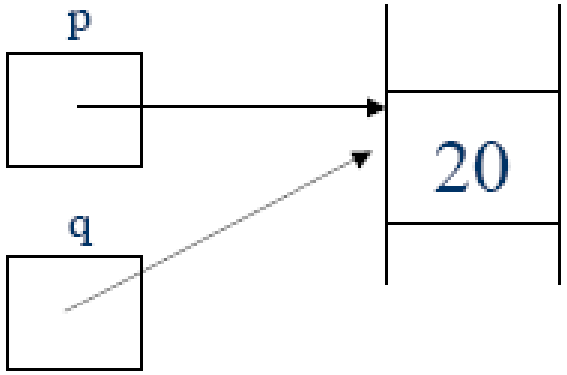
```
variável
p,q:>inteiro



aloque(p)
p^ ← 10
q ← p
q^ ← p^ + 10
escreva(p^)
libere(p)
```



Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo

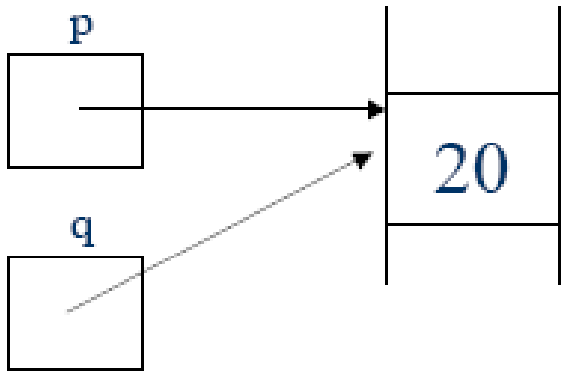
```
variável
p,q:>inteiro



aloque(p)
p^ ← 10
q ← p
q^ ← p^ + 10
escreva(p^)
libere(p)
```



Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo

```
variável
p,q:>inteiro


aloque(p)
p^ ← 10
q ← p
q^ ← p^ + 10
escreva("Resultado = ",p^)
libere(p)
```

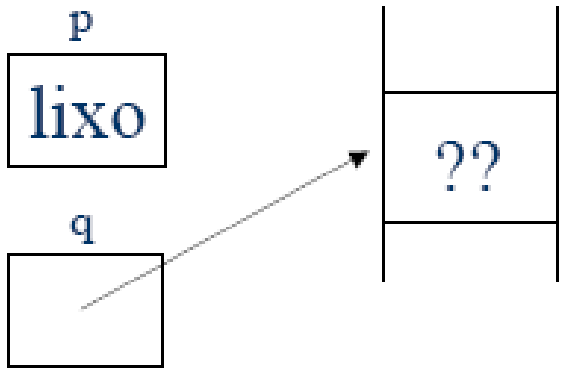


Resultado = 20

Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo

```
variável
p,q:>inteiro
```

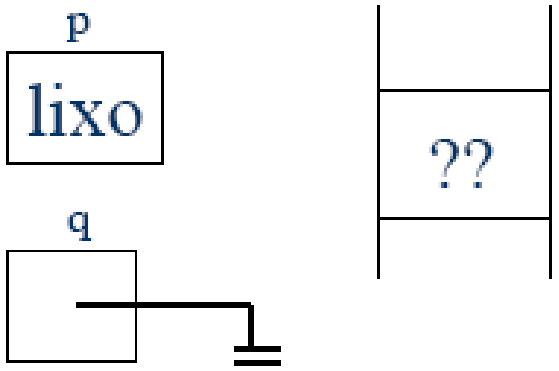



```
aloque(p)
p^ ← 10
q ← p
q^ ← p^ + 10
escreva("Resultado = ",p^)
libere(p)
```

Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo – evitando 'acidentes'

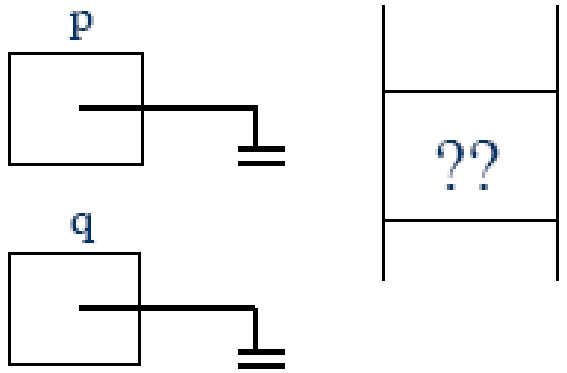
```
variável
p,q:>inteiro



aloque(p)
p^ ← 10
q ← p
q^ ← p^ + 10
escreva("Resultado = ",p^)
libere(p)
q ← nulo
```



Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo – evitando 'acidentes'

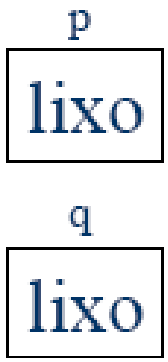
```
variável
p,q:>inteiro



aloque(p)
p^ ← 10
q ← p
q^ ← p^ + 10
escreva("Resultado = ",p^)
libere(p)
q ← nulo
p ← nulo {por garantia, p deve ser anulado}
```



Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo 2

```
tipo
rec = registro
          dado: real
          pont: >rec
      fim registro
variável
p,q :>rec
```

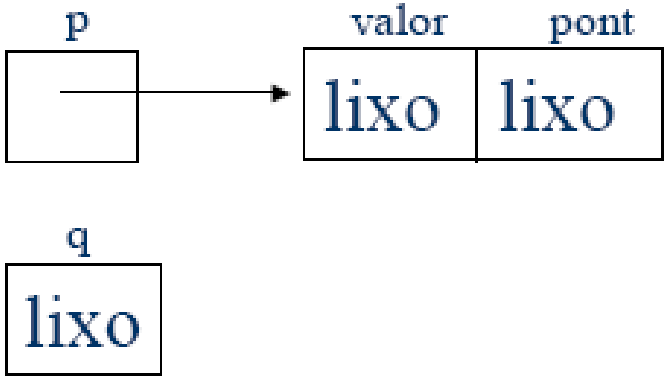


Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo 2

```
variável
p,q:>rec
```

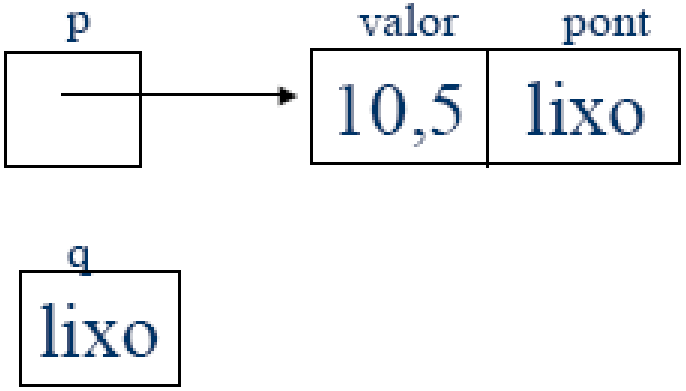
```
aloque(p)
p^.valor ← 10,5
aloque(q)
p^.pont ← q
```



Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo 2

```
variável                aloque(p)
p,q:>rec                p^.valor ← 10,5
                        aloque(q)
                        p^.pont ← q
```

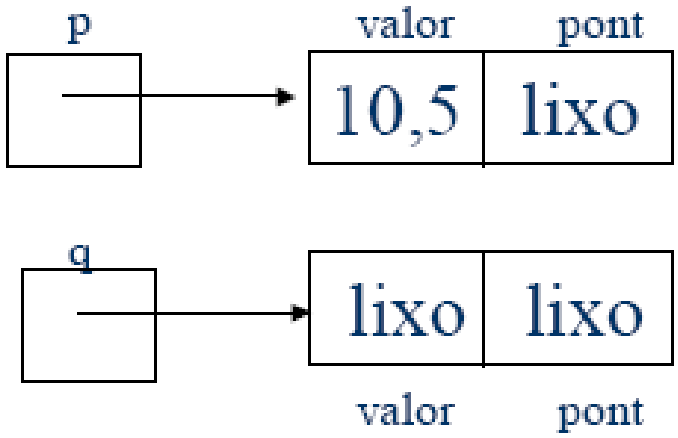


Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo 2

```
variável
p,q:>rec
```

```
aloque(p)
p^.valor ← 10,5
aloque(q)
p^.pont ← q
```

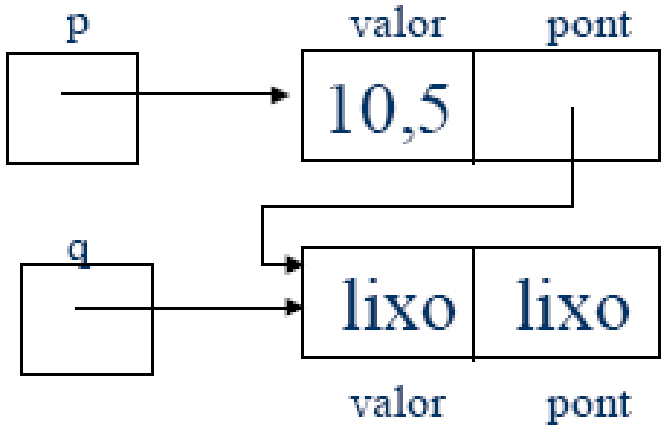


Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo 2

```
variável
p,q:>rec
```

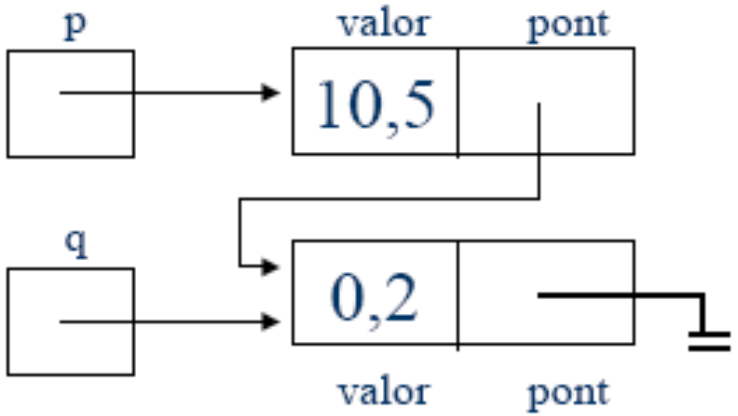
```
aloque(p)
p^.valor ← 10,5
aloque(q)
p^.pont ← q
```



Alocação dinâmica com ponteiros em Pseudocódigo

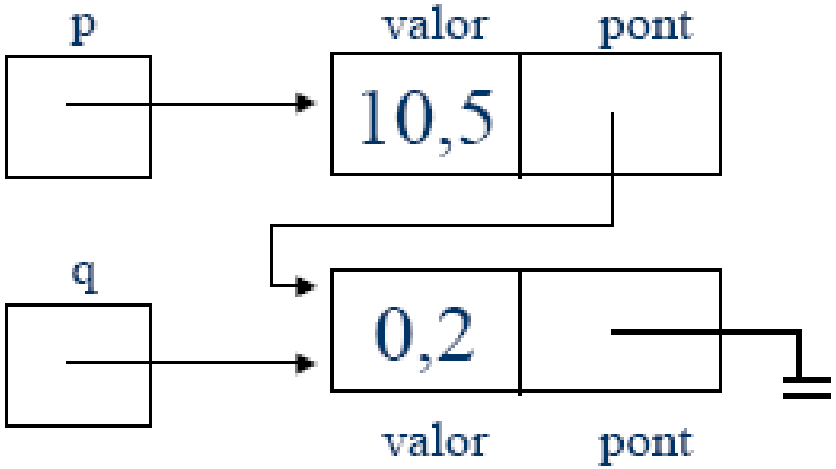
Exemplo 2

```
variável
p,q:>rec

aloque(p)
p^.valor ← 10,5
aloque(q)
p^.pont ← q
q^.valor ← 0,2
q^.pont ← NULO
```



Alocação dinâmica com ponteiros em Pseudocódigo

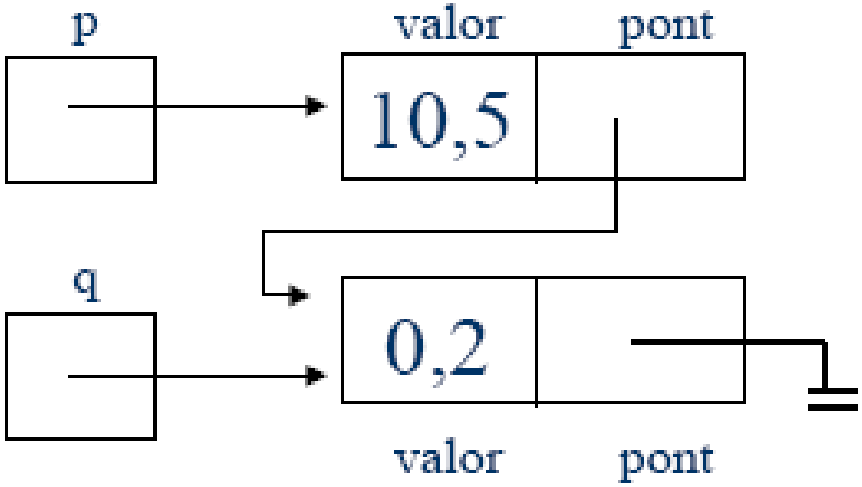
Exemplo 2




```
escreva(p^.valor)
escreva(q^.valor)
escreva(p^.pont^.valor)
```

Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo 2

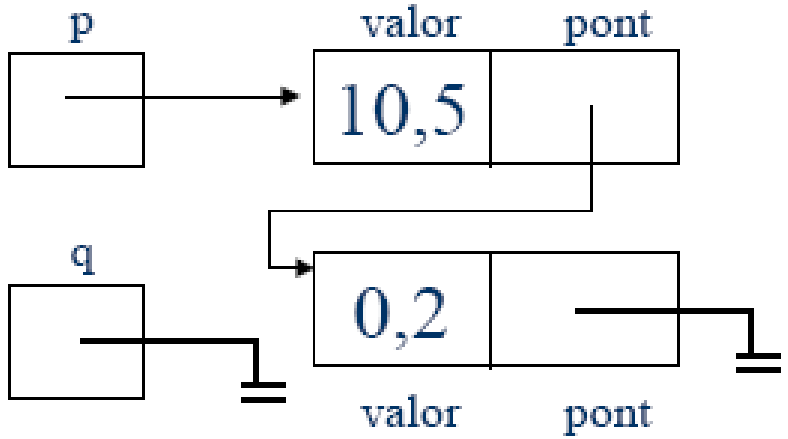



```
escreva(p^.valor)          10,5
escreva(q^.valor)          0,2
escreva(p^.pont^.valor)    0,2
```

Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo 2 – anulando q ainda se consegue acesso ao segundo dado armazenado

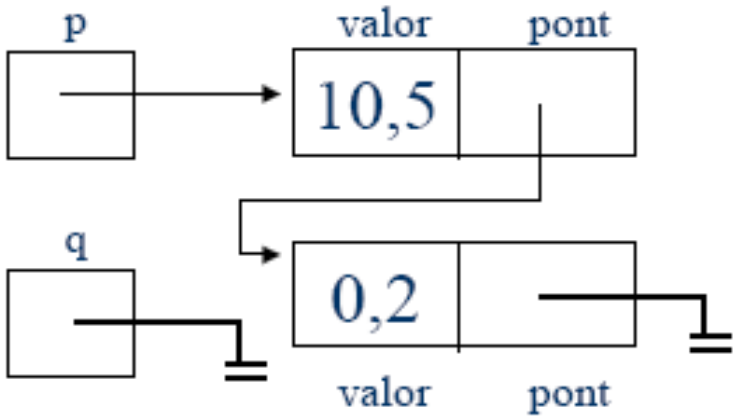
```
variável
p,q:>rec
aloque(p)
p^.valor ← 10,5
aloque(q)
p^.pont ← q
q^.valor ← 0,2
q^.pont ← NULO
q ← NULO
```



Alocação dinâmica com ponteiros em Pseudocódigo

Exemplo 2 – anulando q ainda se consegue acesso ao segundo dado armazenado

```
variável
p,q:>rec
 aloque(p)
 p^.valor ← 10,5
 aloque(q)
 p^.pont ← q
 q^.valor ← 0,2
 q^.pont ← NULO
 q ← NULO
 escreva(p^.pont^.valor)
```



Alocação de Memória

- **“Alocação de memória” diz respeito a como a** memória (necessária para o armazenamento dos dados) é reservada em um programa
- Existem 2 formas de um programa em C alocar memória:
  - Estática
  - Dinâmica

Alocação Estática

- Ocorre na declaração de variáveis globais e variáveis locais;
- No caso de variáveis globais e variáveis locais estáticas, o armazenamento é fixo durante todo o tempo de execução do programa;
- (Variáveis globais são alocadas em tempo de compilação);
- No caso de variáveis locais, o armazenamento dura o tempo de vida da variável.

Alocação Dinâmica

- Para a alocação estática, é necessário que se saiba de antemão (antes do início do programa) a quantidade de memória que será necessária;
- Estimativas podem ser feitas, porém há o risco de sub ou superestimar;
- Assim, muitas vezes é necessário que um programa possa ir ajustando a memória a ser usada durante sua execução;
- "Ajustar" = alocar ou desalocar.

Alocação Dinâmica

- Alocação dinâmica é o método pelo qual um programa ajusta dinamicamente a quantidade de memória a ser usada durante sua execução;
- Permite otimizar o uso da memória
- É implementado mediante funções de alocação/liberação da memória, as quais o programador precisa usar de maneira coerente

Alocação Dinâmica

- Principais funções C para alocação dinâmica de memória:
  - malloc e free;
- Implementadas na biblioteca stdlib.h
- Há diversas outras funções mais específicas, as quais normalmente estão implementadas em stdlib.h (iremos nos fixar em malloc e free).

Usando Malloc

- Serve para alocar memória;
- Devolve um ponteiro para o início da quantidade de memória alocada;
- Exemplo:
  - char *p;
  - p=malloc(1000);
- O trecho acima aloca 1000 bytes de memória para o armazenamento de caracteres.

Usando Malloc

- Porém:
  - **A memória ocupada por um determinado tipo pode variar de máquina para máquina!**
  - **As implementações devem ser independentes da máquina!**
- **Solução: usar o operador sizeof**

Usando Malloc

- **sizeof retorna o tamanho de uma variável ou de um tipo**
  - sizeof funciona em tempo de compilação
- Exemplo de uso de sizeof:
  - float x;
  - printf(“um float ocupa %d bytes nesta maquina”,sizeof(x));
  - printf(“um int ocupa %d bytes nesta maquina”, sizeof(int));

Usando Malloc

- int *p;
- p=malloc(50*sizeof(int));

- **Aloca memória para armazenar 50 inteiros (de maneira contígua na memória)**

Usando Malloc

- Porém, pode ser que o programa tenha alocado muita memória, a ponto de não restar espaço na área de heap (efetivamente, isto pode ocorrer);
- Se malloc não conseguir alocar memória, ele retornará um ponteiro nulo;
- Logo, é preciso testar o resultado de malloc:

```
char *p;
if((p=malloc(50*sizeof(int))) == NULL )
{
    printf("Não foi possivel alocar memoria\n");
    exit(1);
}
```

Usando Free

- Libera memória previamente alocada de maneira dinâmica, por meio de uma das funções de alocação dinâmica (devolve memória ao “heap”)
- Recebe como argumento um ponteiro para a porção de memória que se deseja liberar
- Jamais use free com um argumento inválido, pois isto destrói a lista de memória livre!!
  - free(p);

Alocação Dinâmica

- Exemplo 1
- Usando malloc para alocar estaticamente

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 1000
main()
{
   int x[MAX];... }
```

- **Se usuário do programa informar a quantidade exata de valores inteiros que ele irá precisar armazenar, o trecho acima poderá ser modificado, usando malloc.**

Alocação Dinâmica

Exemplo 1

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int n, i, *p;
    printf("Entre com o numero de elementos: \n");
    scanf("%d", &n );
    if( (p=malloc(n*sizeof(int))) == NULL )
    {
        printf("Não foi possível alocar memória\n");
        exit(1);
}
...
```

Alocação Dinâmica

Exemplo 1

```
if((p=malloc(n*sizeof(int))) == NULL)
{
    printf("Não foi possível alocar
    memoria\n");
    exit(1);
}
    for(i=0; i<n; i++, p++);
    {
    printf("Entre com o %d valor: \n", i+1);
    scanf("%d", p);
     }
```

Alocação Dinâmica

- Exemplo 1:
- Neste primeiro exemplo, a memória foi alocada estaticamente;
- Porém, malloc permite que se aloque memória em qualquer ponto do programa.

Alocação Dinâmica

- Exemplo 2:
- Suponhamos que um programa precise armazenar uma quantidade de números inteiros que varia ao longo da sua execução;
- Um vetor estaticamente alocado não nos servirá, pois queremos evitar que durante a execução falte memória ou que ocorra desperdício de memória;
- Para resolver este problema, iremos criar uma lista de números inteiros, dinamicamente alocada.

Alocação Dinâmica

- **Exemplo 2:**
- Primeiro passo será criar uma estrutura básica (struct) que irá armazenar um número inteiro;
- Como a memória será alocada de maneira não contígua, iremos precisar de um campo para estabelecer a ligação entre duas estruturas;

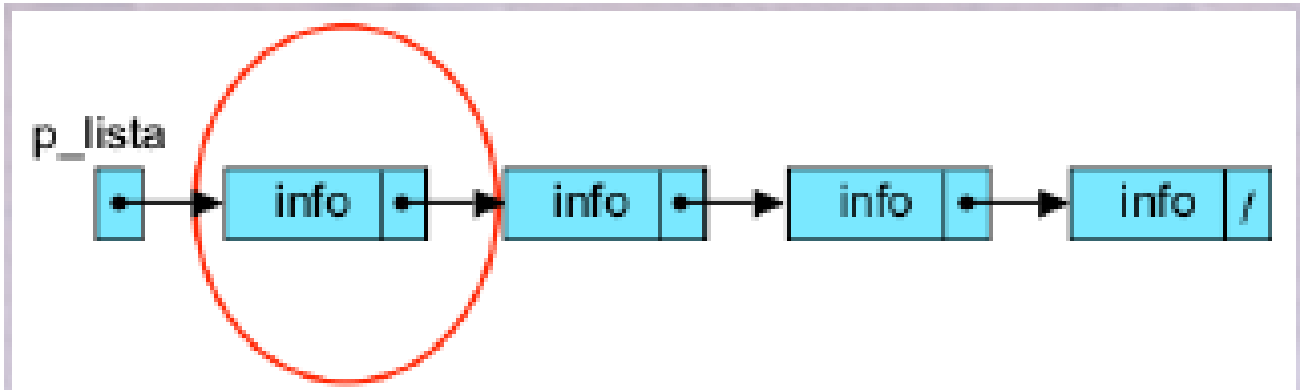
```
typedef struct nodo
{
    int info;
    struct nodo *prox;
} NODO;
```

Alocação Dinâmica

- Exemplo 2: uma lista (encadeamento)

```
typedef struct nodo
{
    int info;
    struct nodo *prox;
} NODO;
```

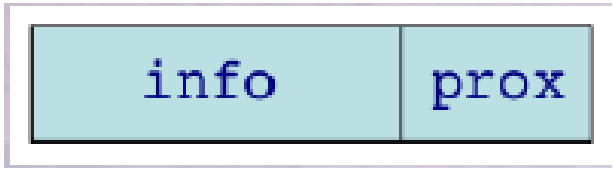
- Esta estrutura será usada para compor uma lista dinâmica (também chamada, encadeada).



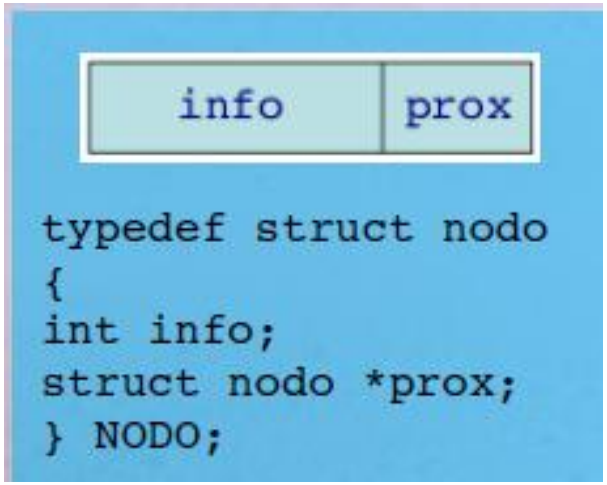
Alocação Dinâmica

- Exemplo 2:
- Alocando memória para um elemento do tipo NODO

```
NODO *p;
if ((p=malloc(sizeof(NODO))) == NULL )
{
    printf("Não foi possível alocar memória para
       NODO\n");
    exit (1);
}
```



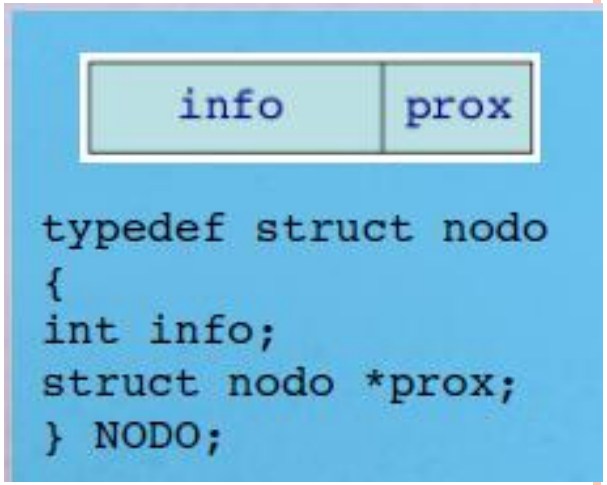
Alocação Dinâmica



- Exemplo 2:
- Alocando memória para um elemento do tipo NODO e preenchendo-o

```
NODO *p;
int x;
if ((p=malloc(sizeof(NODO))) == NULL )
{
    printf("Não foi possível alocar memória para NODO\n");
    exit (1);
}
    printf ("Entre com um valor inteiro:\n");
    scanf("%d", &x);
    p>info = x;
```

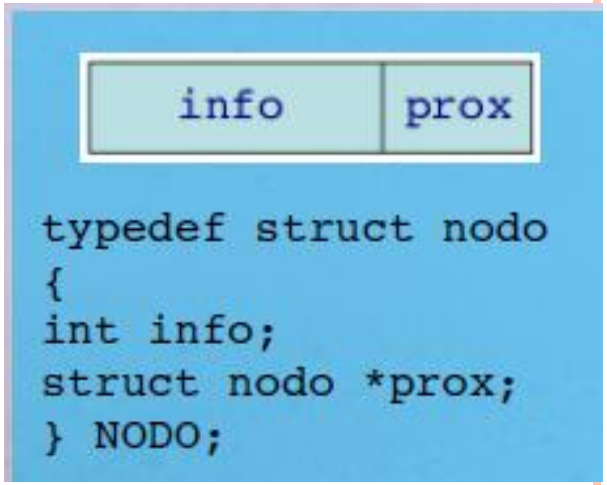
Alocação Dinâmica



- Exemplo 2: Programa Completo

```
#include <stdio.h>
#include <stdlib.h>
NODO *cria_nodo();
void insere( NODO *p, NODO **lista );
void imprime_lista( NODO *lista );
main()
{
    NODO *p_lista, *p_novo;
    p_lista = NULL; //ponteiro para a lista; NULL significa lista vazia
    p_novo = cria_nodo();
    insere(p_novo, &p_lista);
    imprime_lista( p_lista);
}
```

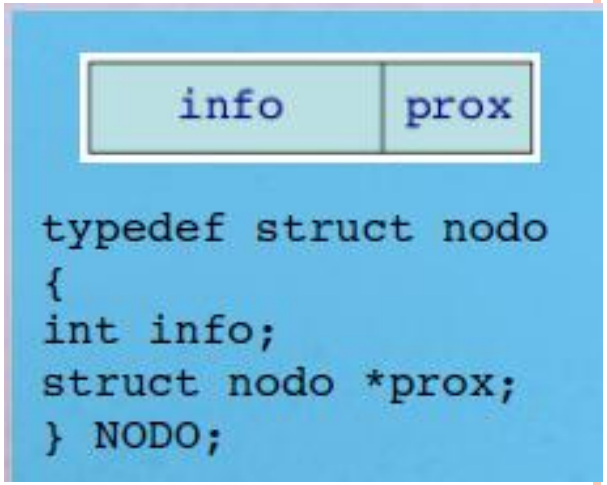
Alocação Dinâmica



- Exemplo 2: Programa Completo

```
NODO *cria_nodo()
{
NODO *p;
int x;
if ((p=malloc(sizeof(NODO))) == NULL)
    {
     printf("Não foi possivel alocar memoria para NODO\n");
     exit(1);
    }
     printf("Entre com o valor inteiro:\n")
     scanf("%d", &x);
     p>info = x; // preenchendo a informacao no campo info
     p>prox = NULL; // convem sempre inicializar todos os campos
     return p;
}
```

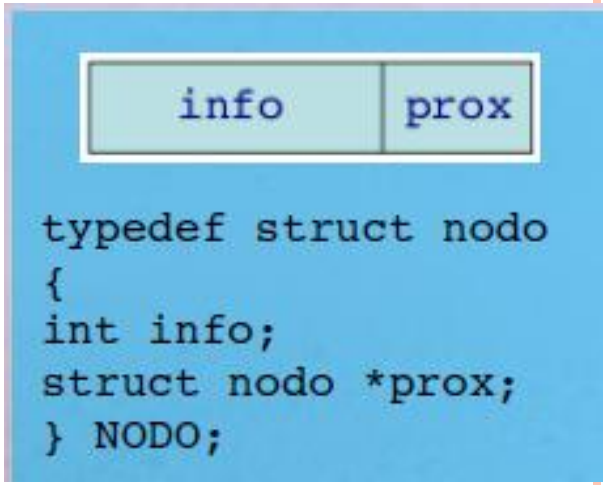
Alocação Dinâmica



- Exemplo 2: Programa Completo

```
// Insere no final da Lista
void insere (NODO *p, NODO **lista)
{
    NODO *aux;
    aux = *lista;
    if (aux==NULL)
    *lista = p;
    else
    {
        while(aux>prox != NULL)
        aux=aux>prox;
        aux>prox = p;
    }
}
```

Alocação Dinâmica

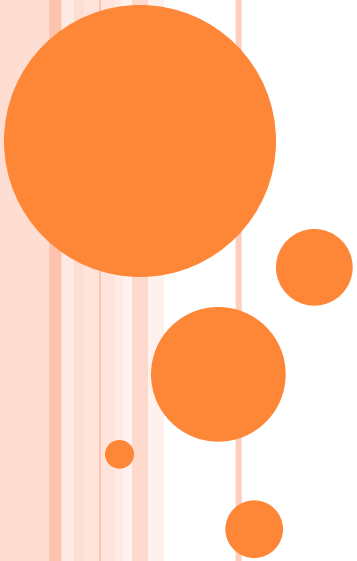


- Exemplo 2: Programa Completo

```
// Imprime o conteudo da lista"
void imprime_lista (NODO *lista)
{
    int i;
    NODO *aux;
    if (lista == NULL)
        printf("Lista Vazia!!\n");
    else
    {
        printf("Conteudo da Lista: \n");
        for (aux = lista, i=1; aux!=NULL; aux = aux>prox, i++)
        printf("%d.o elemento: %d, i, aux>info);
    }
}
```

Exercícios

- Faça uma função que insira um elemento no COMEÇO da lista;
- Faça uma função que insira um determinado elemento n, sendo que n é dado pelo usuário;
- Faça uma função que imprima um determinado elemento n dado pelo usuário;



Ponteiros e Listas

Kalinka Regina Lucas Jaquie Castelo Branco

kalinka@icmc.usp.br