

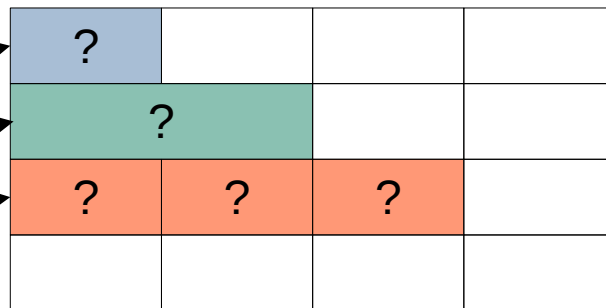
Alocação Dinâmica & Listas

Prof. Denio Duarte
Prof. Geomar A. Schreiner

Alocação Estática

```
int main()
{
    int a;
    float b;
    int v[3];
    a=10;
    b=3.1416;
    v[0]=3;v[1]=4;v[2]=5;
    :
    return 0;
}
```

Memória RAM



Alocação Estática

```
int main()
{
    int a;
    float b;
    int v[3];
    a=10;
    b=3.1416;
    v[0]=3;v[1]=4;v[2]=5;
    :
    return 0;
}
```

Memória RAM

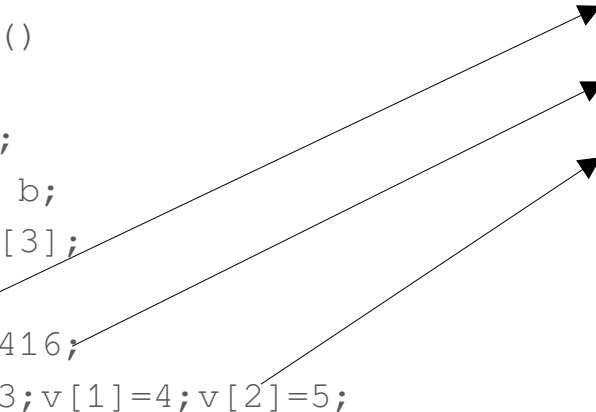
10			
3.1416			
3	4	5	

Os espaços para as variáveis
são alocados durante o carga do
programa para a memória

Alocação Estática

```
int main()
{
    int a;
    float b;
    int v[3];
    a=10;
    b=3.1416;
    v[0]=3;v[1]=4;v[2]=5;
    :
    return 0;
}
```

Memória RAM



10			
3.1416			
3	4	5	

Problema: se tivermos que carregar muitos valores para a memória mas não sabemos o máximo?

Alocação Estática

```
int main()
{
    int a;
    float b;
    int v[3000];
    a=10;
    b=3.1416;
    v[0]=3;v[1]=4;v[2]=5;
    :
    return 0;
}
```

Memória RAM

10					
3.1416					
3	4	5	?	?	...

Problema: se tivermos que carregar muitos valores para a memória mas não sabemos o máximo?

1. Desperdício de memória
2. Falta de memória caso existam mais valores do que o planejado

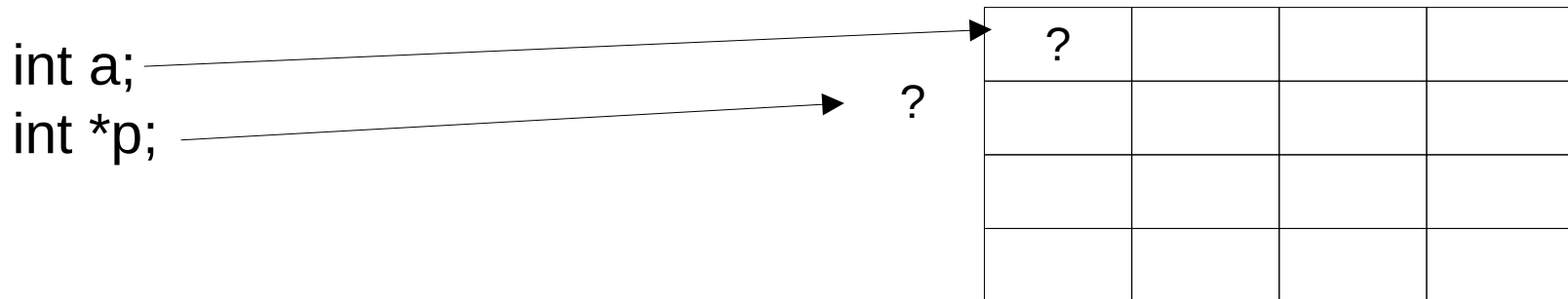
Antes de continuarmos ...

- Recapitulação ponteiros

int a;

int *p;

?

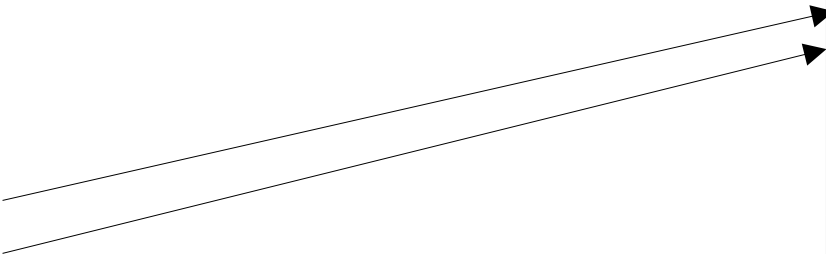


?			

Antes de continuarmos ...

- Recapitulação ponteiros

```
int a;  
int *p;  
a=10;  
p=&a;
```

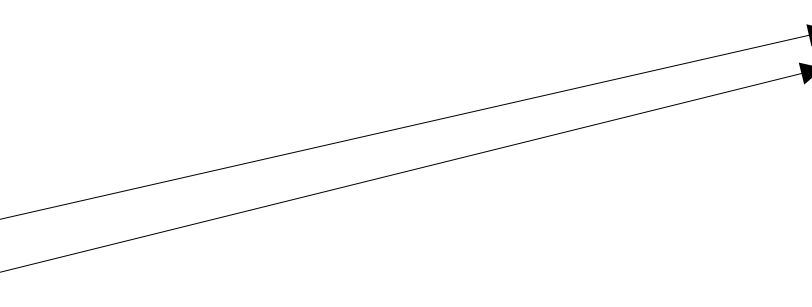


10			

Antes de continuarmos ...

- Recapitulação ponteiros

```
int a;  
int *p;  
a=10;  
p=&a;  
*p=20;
```



20			

Antes de continuarmos ...

- Recapitulação ponteiros

```
int a;
```

```
int *p;
```

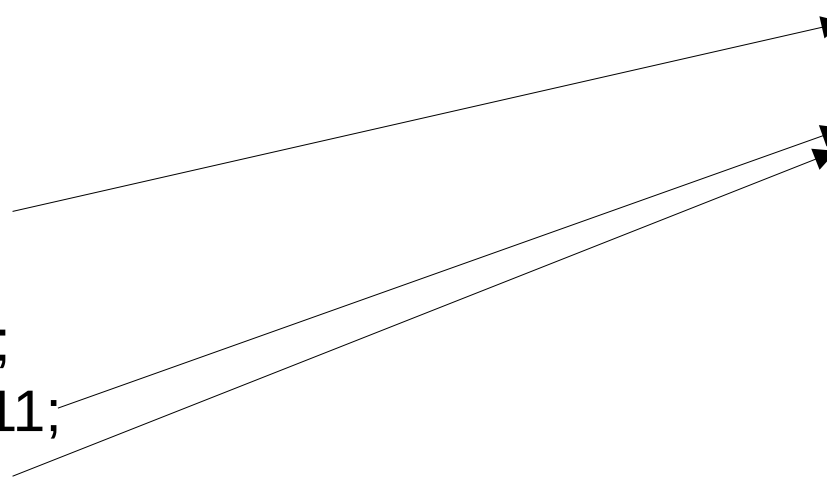
```
a=10;
```

```
p=&a;
```

```
*p=20;
```

```
int b=11;
```

```
p=&b;
```



The diagram illustrates the state of memory after the provided code. It consists of a 4x4 grid of cells. The first row has the value '20' in the first column. The second row has the value '11' in the first column. The remaining cells are empty. Three arrows originate from the code on the left: one from 'a=10;' pointing to the first cell of the first row, one from '*p=20;' pointing to the first cell of the second row, and one from 'p=&b;' pointing to the first cell of the third row. This indicates that the value 10 was stored at the memory address of 'a', the value 20 was stored at the memory address of 'p' (which points to 'a'), and the address of 'b' was stored at the memory address of 'p'.

20			
11			

Antes de continuarmos ...

- Lembrando: vetores são ponteiros disfarçados :)

```
int v[4]={2,4,6,8};  
*v=10;
```

2	4	6	8

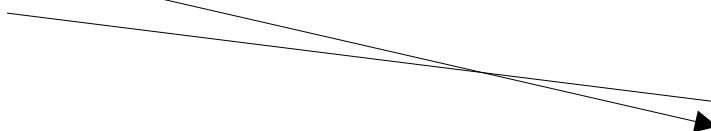
Antes de continuarmos ...

- Lembrando: vetores são ponteiros disfarçados :)

```
int v[4]={2,4,6,8};  
*v=10;  
*(v+1)=12;
```

2	4	6	8

10	12	6	8



Antes de continuarmos ...

- Lembrando: vetores são ponteiros disfarçados :)
 - Por isso não se coloca & no scanf para strings (vetores de caracteres):

```
char st[10];  
scanf("%s", st); // e não scanf("%s", &st)
```

Alocação Dinâmica

- Os espaços para as variáveis são alocados durante a execução do programa.
 - O espaço deve ser alocado e uma variável deve apontar para ele (**variável ponteiro**)
 - A variável que aponta e o espaço alocado devem ser do mesmo tipo
 - A biblioteca **stdlib.h** deve ser incluída para usarmos a função de alocação de memória

Alocação Dinâmica

- Como utilizar?
 - Criar uma variável com o tipo desejado colocando * antes do nome da variável: `int *p;`
 - Para armazenar algo no local que a variável está apontando: `*p=10;`
 - Para alterar o endereço que a variável aponta faz-se: `p=<novo endereço>;` `p=&a;` // & indica endereço de algo
 - Para alocar um espaço de memória e retornar o endereço do local: `malloc(qtdade de bytes)=> p=(int *)malloc(sizeof(int));`

Alocação Dinâmica

- Exemplo:

:

int

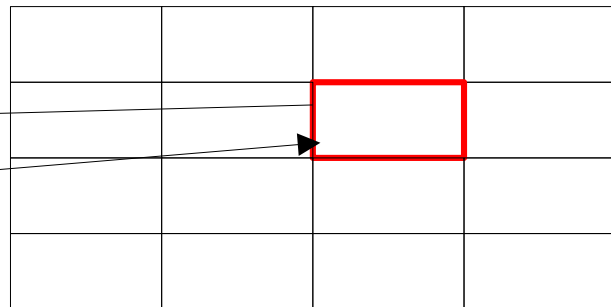
*p;

?

Alocação Dinâmica

- Exemplo:

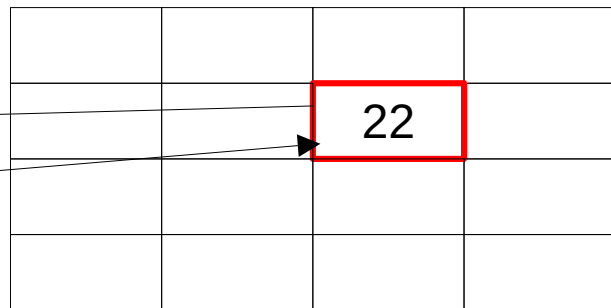
```
:  
int *p;  
p=(int *)malloc(sizeof(int));
```



Alocação Dinâmica

- Exemplo:

```
:  
int *p;  
p=(int *)malloc(sizeof(int));  
*p=22;  
print("Conteúdo do p: %d", *p);  
print("Local de apontamento: %p", p);  
// interessante, veja como fica o scanf  
scanf("%d", p); // e não scanf("%d", &p)
```



		22	

Listas

1. Até agora, vimos uma estrutura de dados: **vetores**
2. Propriedades importantes de um **vetor**:
 - a. São uma área de memória contígua
3. Em aula anterior vimos uma segunda estrutura: **structs**
4. Propriedades importantes de uma **struct**:
 - a. Os elementos (**membros**) de uma **struct** podem ser de tipos diferentes
 - b. Para selecionar um elemento de uma **struct**, especificamos o nome do elemento

Relembrando Struct

1. Para declarar variáveis que são **structs**, podemos escrever

```
struct {  
    int dia;  
    int mes;  
    int ano;  
} data1, data2;
```

```
struct {  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
} funcionario1, funcionario2;
```

2. Representação de **data1** na memória do computador:



3. Os nomes dos membros de uma **struct** não conflitam com outros nomes de fora da struct

Relembrando Struct

1. Definindo um tipo com typedef

```
typedef struct {  
    int dia;  
    int mes;  
    int ano;  
} Data;
```

```
typedef struct {  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
} Funcionario;
```

```
Data data1, data2;  
Funcionario funcionario1, funcionario2;
```

Lista

- Uma lista, de maneira geral, é um conjunto de elementos de mesmo tipo.
- Uma lista pode ser implementada de diversas formas.

- Podemos considerar isso como uma lista?

```
typedef struct {  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
} Funcionario;
```

```
Funcionario funcionarios[25];
```

Lista

- Podemos considerar isso como uma lista?
 - Sim. Vetores criam listas estáticas
 - Já fizemos nos exercícios anteriores.
 - Quais os problemas que ela apresenta?

```
typedef struct {  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
} Funcionario;
```

```
Funcionario funcionarios[25];
```

Lista

- Lista com vetor
 - Quais os problemas que ela apresenta?
 - Tamanho fixo da quantidade máxima de elementos
 - Possível fonte de desperdício de memória

```
Funcionario funcionarios[25];
```

```
typedef struct {  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
} Funcionario;
```

Saudade do Python :)

- Cria-se uma **struct** (classe)
- Uma variável do tipo lista (lista_f)
- Adicionava-se o elemento da classe na lista
- Repetia até o infinito ...

```
class func:  
    id=0  
    nome=""  
    salario=0.0;  
    #  
list_f=[]  
f=func()  
f.id=15  
f.nome='Quinze'  
f.salario=15  
list_f.append(f)  
f=func()  
f.id=16  
f.nome='Dezesseis'  
f.salario=16  
list_f.append(f)
```


Back to reality

1. Implementando lista dinamicamente em C
2. Para declarar variáveis que são ponteiros para **structs**, podemos escrever

```
typedef struct {  
    int dia;  
    int mes;  
    int ano;  
} Data;
```

```
typedef struct {  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
} Funcionario;
```

```
Data *data1;  
Funcionario *funcionario1, *funcionario2;  
  
data1 = (Data *) malloc(sizeof(Data));  
data1->dia=5 // não pode ser data1.dia  
data1->mes=8  
data1->ano=2021
```



```
// Cuidado  
Data dt; // versão estática  
  
dt.dia=5  
dt.mes=8  
dt.ano=2021
```

3. Representação de **data1** na memória do computador:

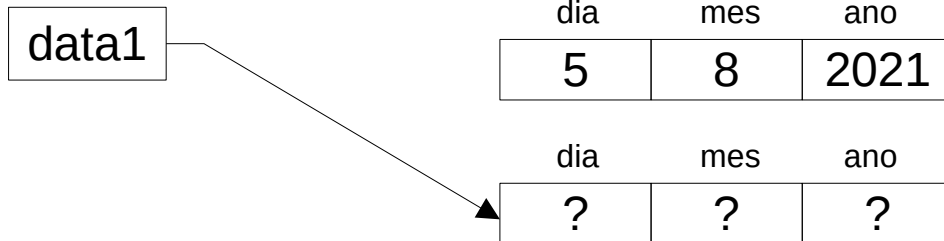


Back to reality

```
typedef struct {  
    int dia;  
    int mes;  
    int ano;  
} Data;
```

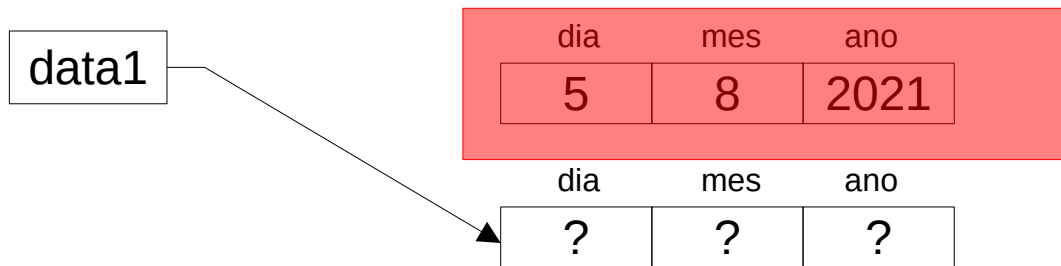
```
typedef struct {  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
} Funcionario;
```

```
Data *data1;  
Funcionario *funcionario1, *funcionario2;  
  
data1 = (Data *) malloc(sizeof(Data));  
data1->dia=5 // não pode ser data1.dia  
data1->mes=8  
data1->ano=2021  
data1 = (Data *) malloc(sizeof(Data));
```



Back to reality

1. Quando alocamos um novo espaço na memória perdemos o endereço do anterior :(



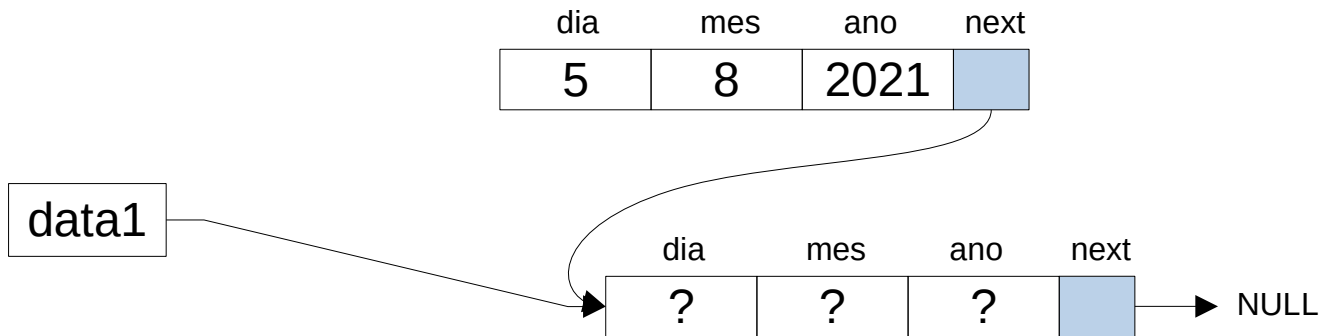
2. Em Python a lista era acessada com o índice: `lista[0]`, `lista[1]`, ...
3. Em C só daria certo se alocarmos tudo de uma vez só, mas cairíamos no problema de “o que é tudo?”
 - `data1 = (Data *) malloc(sizeof(Data)*100);` // igual a `Data data1[100];`
 - Agora poderíamos fazer: `data1[0]`, ... `data1[99]`

Lista encadeada

- Uma lista encadeada representa uma sequência de objetos, de mesmo tipo, na memória.
 - Os espaços alocados não estão organizados contiguamente (de forma contínua)
- Cada elemento da sequência armazena seu valor e o endereço do próximo elemento
 - Ou seja, junto a cada um dos elementos da lista, explicitamente armazenamos o endereço para o próximo elemento da lista

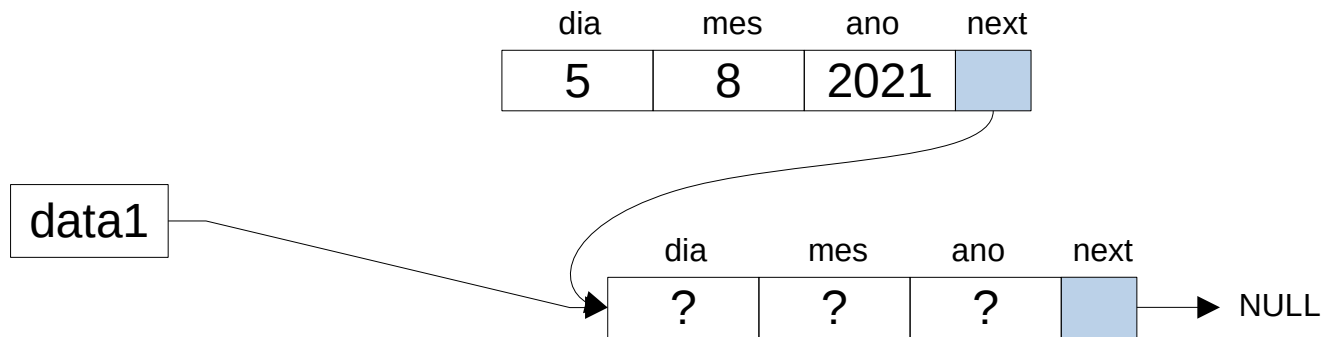
Lista encadeada

- A lista é chamada de encadeada pois dentro do espaço alocado tem um ponteiro que aponta para o próximo
 - Isso se faz necessário para os espaços não ficarem perdidos



Lista encadeada

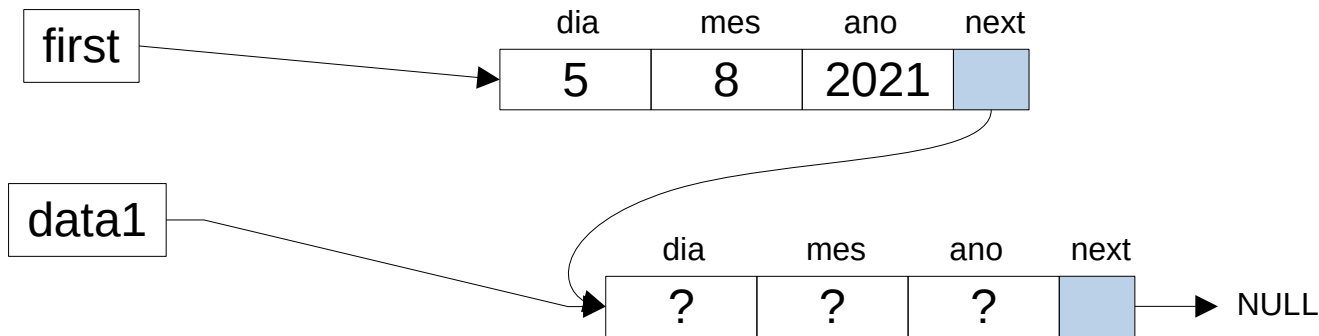
- A lista é chamada de encadeada pois dentro do espaço alocado tem um ponteiro que aponta para o próximo
 - Isso se faz necessário para os espaços não ficarem perdidos



Ok. Agora conseguimos “caminhar”
pelas regiões alocadas, mas como
começar a caminhada?

Lista encadeada

- A lista é chamada de encadeada pois dentro do espaço alocado tem um ponteiro que aponta para o próximo
 - Isso se faz necessário para os espaços não ficarem perdidos

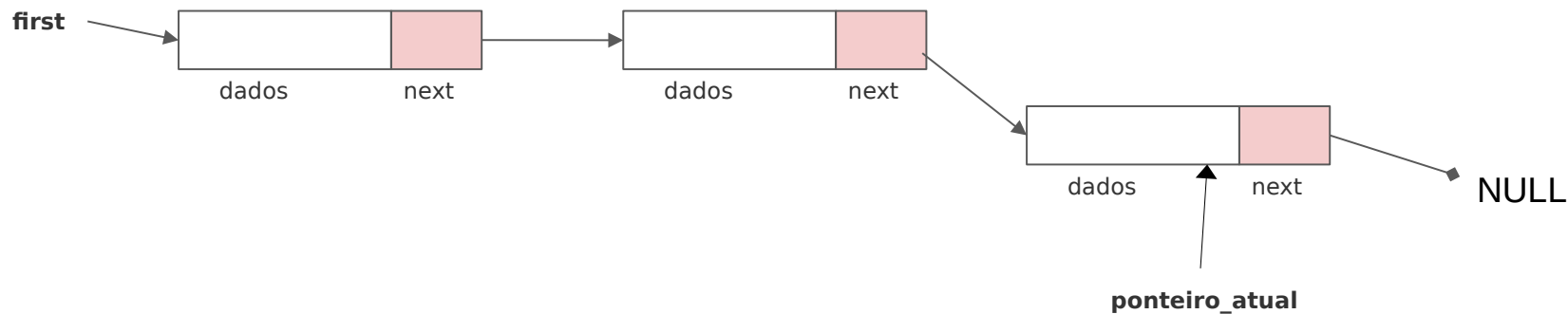


Ok. Agora conseguimos “caminhar”
pelas regiões alocadas, mas como
começar a caminhada?

Basta criar um ponteiro auxiliar que
vai nos dizer quem é o primeiro :)

Lista encadeada

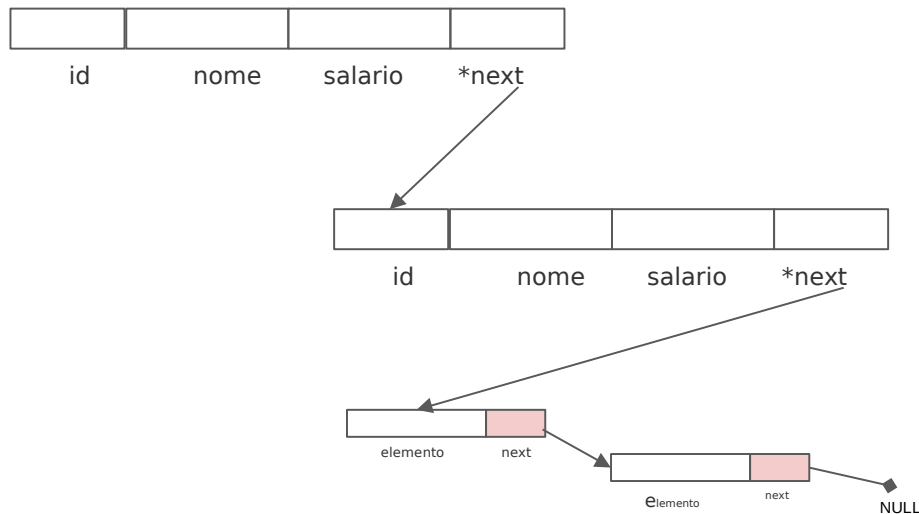
- A lista é chamada de encadeada pois dentro do espaço alocado tem um ponteiro que aponta para o próximo
 - Esquema



Lista encadeada

- Uma lista encadeada representa uma sequência de objetos, de mesmo tipo, na memória. Cada elemento da sequência armazena seu valor e o endereço do próximo elemento
 - Ou seja, junto a cada um dos elementos da lista, explicitamente armazenamos o endereço para o próximo elemento da lista

```
struct funcionario{  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
    struct funcionario *next;  
};  
typedef struct funcionario Funcionario;
```



Lista encadeada

- Uma lista encadeada representa uma sequência de objetos, de mesmo tipo, na memória. Cada elemento da sequência armazena seu valor e o endereço do próximo elemento
- Os elementos de uma lista não ocupam uma área contígua de memória (como os vetores), o que não permite acesso direto aos elementos.
- Para acessar um elemento, é necessário que todos os elementos estejam encadeados.

```
struct funcionario{  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
    struct funcionario *next;  
};  
typedef struct funcionario Funcionario;
```

Lista encadeada

- Como criamos a lista

```
struct funcionario{  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
    struct funcionario *next;  
}  
typedef struct funcionario Funcionario;
```

```
Funcionario *first; //head é legal também :)  
  
first = (Funcionario *) malloc(sizeof(Funcionario));  
  
first->id = 1;  
strcpy(first->nome, "Pafuncio");  
first->salario = 3000.0;  
first->next = NULL;
```

Memória

first

?


```
Funcionario *first; //head
```

```
first = (Funcionario *) malloc(sizeof(Funcionario));
```

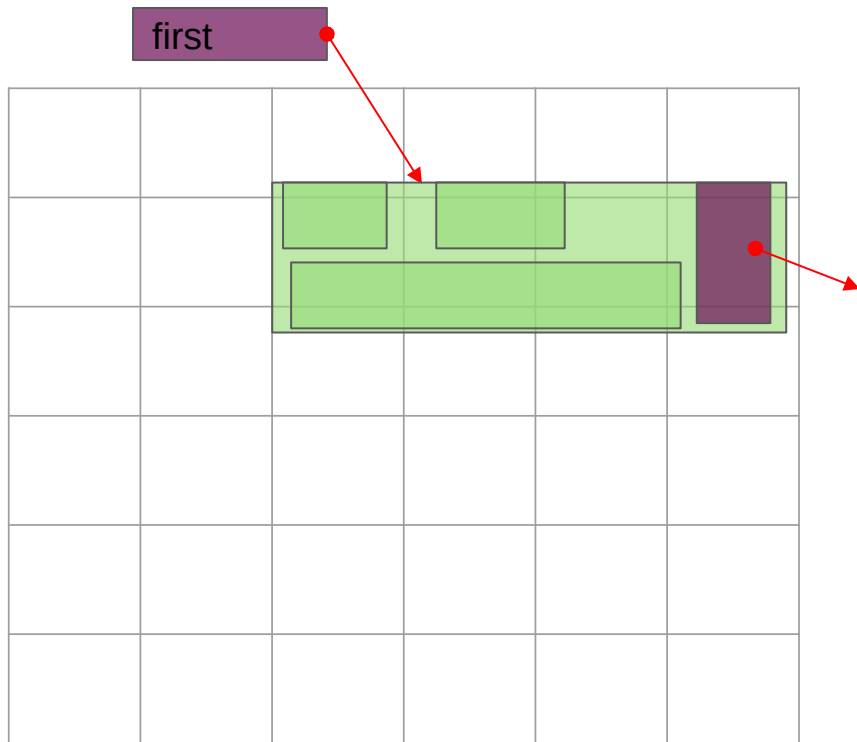
```
first->id = 1;
```

```
strcpy(first->nome, "Pafuncio");
```

```
first->salario = 3000.0;
```

```
first->next = NULL;
```

Memória



```
Funcionario *first; //head
```

```
first = (Funcionario *) malloc(sizeof(Funcionario));
```

```
first->id = 1;
```

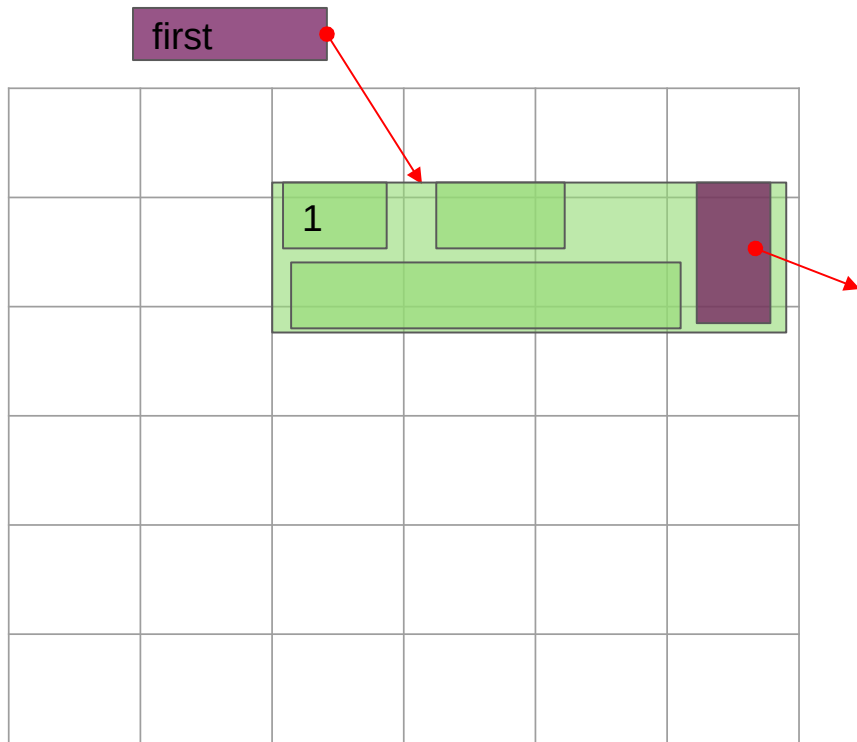
```
strcpy(first->nome, "Pafuncio");
```

```
first->salario = 3000.0;
```

```
first->next = NULL;
```



Memória



```
Funcionario *first; //head
```

```
first = (Funcionario *) malloc(sizeof(Funcionario));
```

```
first->id = 1;
```

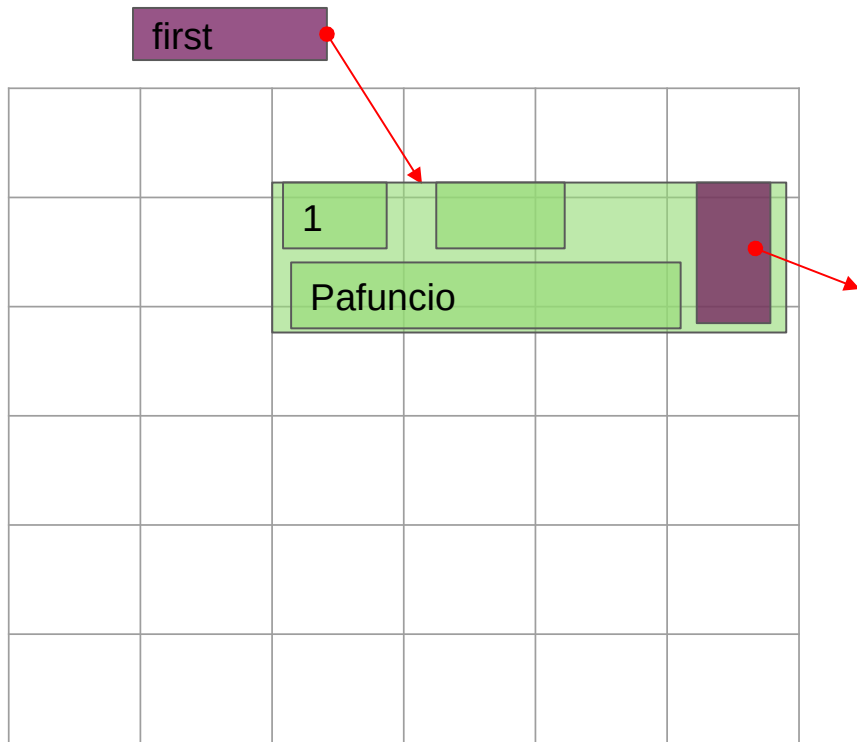
```
strcpy(first->nome, "Pafuncio");
```

```
first->salario = 3000.0;
```

```
first->next = NULL;
```



Memória



```
Funcionario *first; //head
```

```
first = (Funcionario *) malloc(sizeof(Funcionario));
```

```
first->id = 1;
```

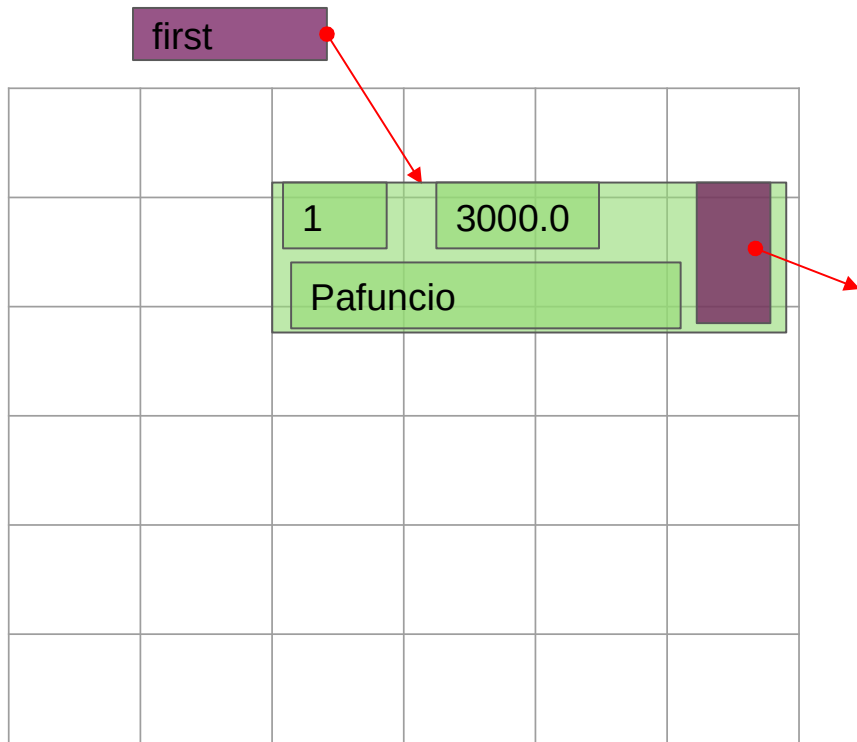
```
strcpy(first->nome, "Pafuncio");
```

```
first->salario = 3000.0;
```

```
first->next = NULL;
```



Memória



```
Funcionario *first; //head
```

```
first = (Funcionario *) malloc(sizeof(Funcionario));
```

```
first->id = 1;
```

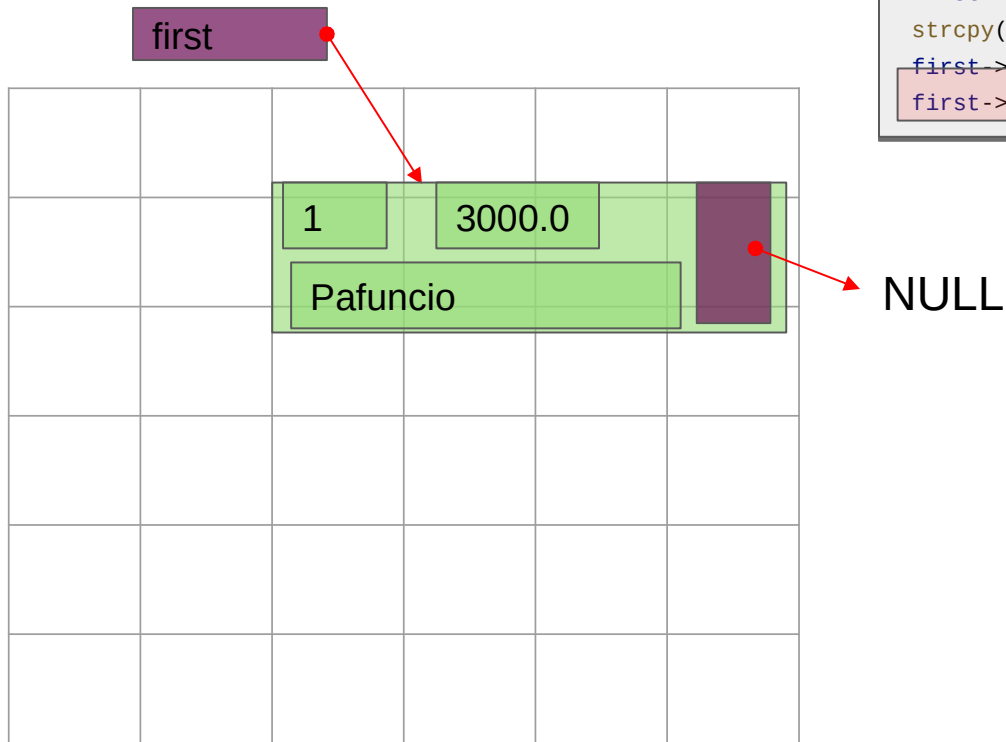
```
strcpy(first->nome, "Pafuncio");
```

```
first->salario = 3000.0;
```

```
first->next = NULL;
```



Memória



```
Funcionario *first; //head
```

```
first = (Funcionario *) malloc(sizeof(Funcionario));
```

```
first->id = 1;
```

```
strcpy(first->nome, "Pafuncio");
```

```
first->salario = 3000.0;
```

```
first->next = NULL;
```

Lista encadeada

- Adicionando outros elementos
 - Preciso pensar no encadeamento da lista

```
struct funcionario{  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
    struct funcionario *next;  
}  
typedef struct funcionario Funcionario;
```

```
Funcionario *f, *aux, first=NULL;  
for (/*QUANTOS EU QUISE*/){  
    f = (Funcionario *)malloc(sizeof(Funcionario));  
    f->id = contador; //contador é uma variável qualquer  
    scanf("%s",&f->nome);  
    f->salario = 3000.0;  
    f->next = NULL;  
    if (first == NULL){  
        first = f; aux=f; // inicialmente, todos apontam pa-  
    } else {                // para a primeira região alocada  
        aux->next=f; // aux deve apontar sempre para a região  
        aux=f;      // anterior à nova alocada  
    }  
}
```

Lista encadeada

- Como imprimimos os elementos
 - Para imprimir devemos iterar sobre todos os elementos partindo do primeiro

```
struct funcionario{  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
    struct funcionario *next;  
}  
typedef struct funcionario Funcionario;
```

```
Funcionario *aux; //vai ser nosso 'contador'  
for (aux = first; aux != NULL; aux = aux->next){  
    //aqui aux vale o elemento atual na lista.  
    printf("Funcionario id: %d, nome: %s, salario: %lf\n", aux->id, aux->nome, aux->salario);  
}
```

Lista encadeada

- Como criamos a lista
 - Existe várias formas
 - Outra possibilidade

```
struct funcionario{  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
    struct funcionario *next;  
};  
typedef struct funcionario Funcionario;
```

```
typedef struct{  
    Funcionario *first;  
} Lista;
```

Lista encadeada

- Como criamos a lista
 - Existe várias formas
 - Outra possibilidade

```
typedef struct {  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
} Funcionario;
```

```
struct lista {  
    Funcionario *elemento;  
    struct lista *next;  
};  
typedef struct lista Lista;
```

Lista encadeada

- Como criamos a lista
 - Existe várias formas
 - Outra possibilidade

```
typedef struct {  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
} Funcionario;
```

```
struct lista {  
    Funcionario *elemento;  
    struct lista *next;  
};  
typedef struct lista Lista;
```

```
Lista *first;
```

Atenção! Achtung! Attento! Watch out!

```
struct funcionario{  
    int id;  
    char nome[41];  
    double salario;  
    Data nascimento;  
    struct funcionario *next;  
};  
typedef struct funcionario Funcionario;
```

```
typedef struct {  
    int dia;  
    int mes;  
    int ano;  
} Data;
```

```
Funcionario func;  
func.id=10;  
strcpy(func.nome, "Your Name");  
func.salario=4500.45;  
func.nascimento.dia=10;  
func.nascimento.mes=8;  
func.nascimento.ano=2000;  
func.next=NULL;
```

```
Funcionario *func;  
func=(Funcionario *)malloc(sizeof(Funcionario))  
func->id=10;  
strcpy(func->nome, "Your Name");  
func->salario=4500.45;  
func->nascimento.dia=10;  
func->nascimento.mes=8;  
func->nascimento.ano=2000;  
func->next=NULL;
```

Importante

- Toda a memória alocada deve ser liberada se não os espaços ficarão perdidos
 - Algumas linguagens possuem um sistema de limpeza de memória: **garbage collector**
 - O C padrão não!
 - `free()`

```
Funcionario *first; //head é legal também :)

first = (Funcionario *) malloc(sizeof(Funcionario));
first->id = 1;
strcpy(first->nome, "Pafuncio");
first->salario = 3000.0;
first->next = NULL;
:
free(first);
```


Exercícios

1. Considerando as definições a seguir, faça o que é pedido nos itens abaixo:

```
typedef struct {  
    int dia;  
    int mes;  
    int ano;  
} Data;
```

```
struct funcionario{  
    int id;  
    char nome[41];  
    double salario;  
    Data nascimento;  
    struct funcionario *next;  
};  
typedef struct funcionario Funcionario;
```

- Crie as estruturas indicadas, e crie o primeiro funcionário da lista encadeada;
- Adicione um segundo funcionário mantendo o encadeamento;
- Crie uma função que receba o ponteiro inicial da lista e imprima todos os elementos (funcionários)

Exercícios

2. Considerando a estrutura proposta no exercício anterior, faça as seguintes adaptações em seu programa:
 - a. O programa deve ler (do teclado) vários registros de funcionários (quando id for igual a 0 a entrada é finalizada).
 - b. Use a mesma função implementada anteriormente e imprima a lista para ver se todos os elementos estão presentes
 - c. **Super desafio:** crie uma nova função que imprimir a lista na ordem inversa
 - Dica: vimos uma possível técnica na aula última aula :)