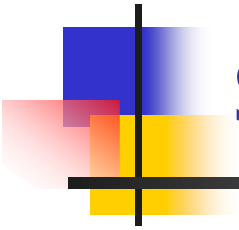


SIN211 Algoritmos e Estruturas de Dados

Prof. João Batista Ribeiro

joao42lbatista@gmail.com

UFV

Universidade Federal de Viçosa

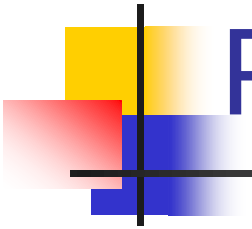
Slides baseados no material da Prof.ª Rachel Reis

Revisão – Lista Linear Estática

- Definição:

```
#define TAML 15
typedef struct sLISTA{
    <tipo> vetor[TAML];
     int n;
}LISTA;
```

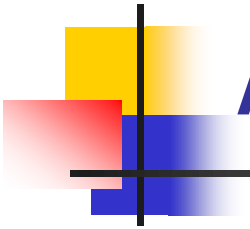
- Operações
  - → void lista_inicializar(Lista *L){ ... }
  - → tipo lista_acessar(Lista *L, int indice) { ... }
  - → int lista_buscar(Lista *L, Tipo valor){...}
  - → int lista_cheia(Lista *L){...}
  - → int lista_inserirfim(Lista *L, Tipo valor){ ... }
  - → int lista_inserir(Lista *L, int indice, Tipo valor){ ... }
  - → int lista_modificar(Lista *L, int indice, Tipo novo_valor){ ... }
  - → int lista_vazia(Lista *L){ ... }
  - → int lista_remover(Lista *L, int indice){ ... }
  - → int lista_tamanho(Lista *L){ ... }



Revisão – Sugestão Arquivos.c

1) ListaLinearEstatica.c:
   - Definição
   - Operações

2) AplicacaoListaLinearEstatica.c
   - # include “ListaLinearEstatica.c”
   - Implementação da função int main(){ ... }

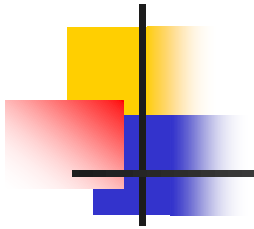


Aula de Hoje

- Lista Linear Dinâmica

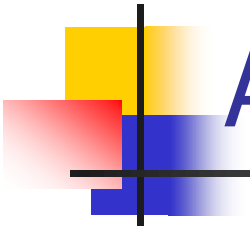
  => Lista encadeada

Melhor solução: Lista Estática ou Dinâmica?



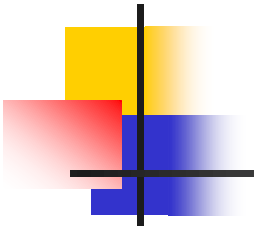
- Jorge que acabou de se formar na UFV-CRP conseguiu emprego em uma empresa de telefonia de grande porte na cidade de São Paulo. Como era do seu interesse ele foi contratado para trabalhar na área de desenvolvimento para celular. O módulo que Jorge terá que implementar corresponde ao módulo de "contatos".

Por que???



Alocação: Sequencial x Encadeada

- Alocação de Memória Sequencial
  - itens agrupados em células consecutivas de memória
- Alocação de Memória Encadeada
  - itens ocupam células espalhadas por toda memória
  - itens são armazenados em blocos de memória denominados nós

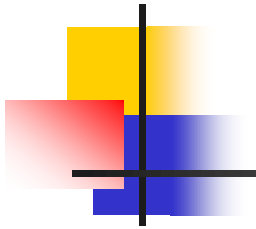


Lista Encadeada

- Algumas vezes não é possível resolver o problema de forma sequencial
- Com *arrays* temos o problema de custo de inserção e/ou remoção no meio da lista

Solução???

- Uso de lista encadeada, também conhecida como lista ligada
- Uma lista encadeada é uma estrutura de dados linear e dinâmica



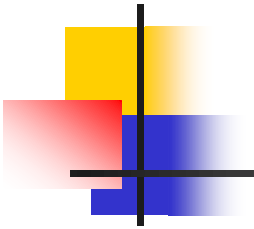
Lista Encadeada

- Vantagens:

  - Facilidade para inserção e remoção de itens em posições arbitrárias;

  - Pouca movimentação dos dados em memória;

  - Útil em aplicações em que o tamanho máximo da lista não precisa ser definido *a priori.*



Lista Encadeada

- Desvantagens:

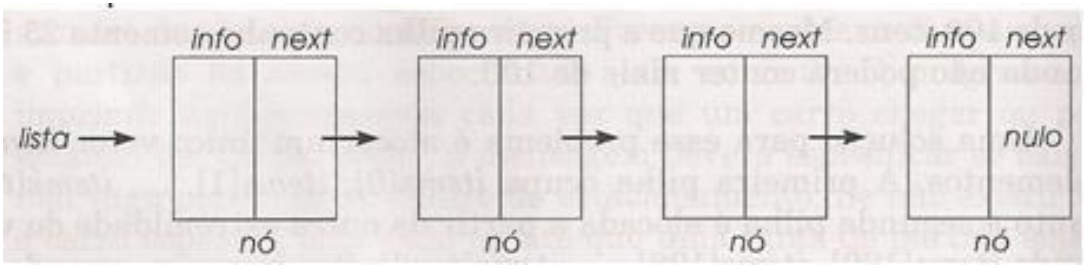
  - Pode consumir mais tempo para preparação do sistema para alocar e liberar armazenamento;

  - Em média, o acesso a um item é mais oneroso que o acesso direto oferecido pelos *arrays*.

Lista Encadeada

- Definição: estrutura de dados que mantém uma coleção de itens em ordem linear sem exigir que eles ocupem posições consecutivas de memória.

- Lista Simplesmente Encadeada
  - itens são armazenados em blocos de memória denominados nós
  - cada nó possui dois campos:
    1) campo de informação
    2) campo do endereço do nó seguinte da lista



Como vocês sugerem a implementação dessa estrutura de dados?

Lista Simplesmente Encadeada (definição)

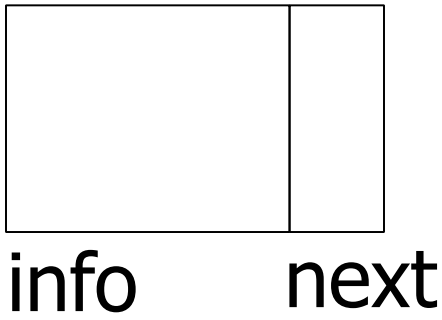
- Definir um nó para uma lista simplesmente encadeada linear cujos nós são alocados dinamicamente

```
typedef struct sCELULA{
   <tipo> info;
   struct sCELULA *next;
}CELULA;
```

Lista Simplesmente Encadeada (definição)

- Considere que o campo informação seja um valor do tipo primitivo **int**

```
typedef struct sCELULA{
    int info;
    struct sCELULA *next;
}CELULA;
```

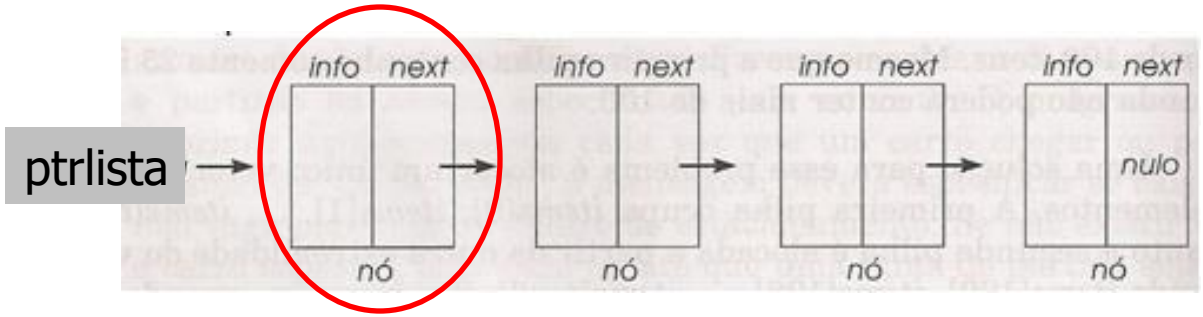


Lista Encadeada

- Uma célula c pode ser declarados como:

```
CELULA c;
```

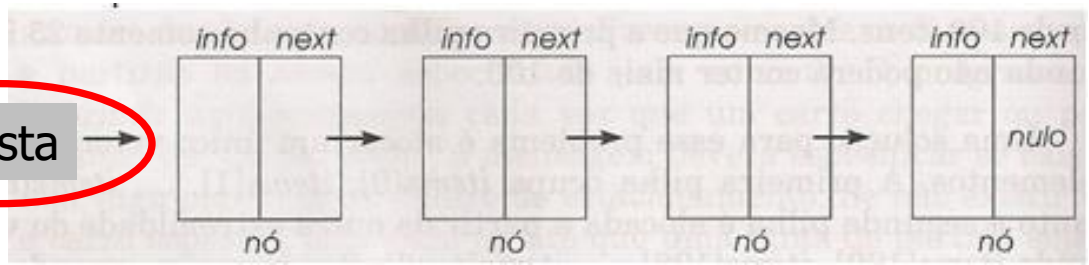
- Se c é uma célula então
  - **`c.info`** é o conteúdo da célula e
  - **`c.next`** é o endereço da próxima célula



Lista Encadeada

```
int main() {
    CELULA * ptrlista;  /* ponteiro externo ptrlista que
     ...                            aponta para o primeiro nó da lista */
}
```

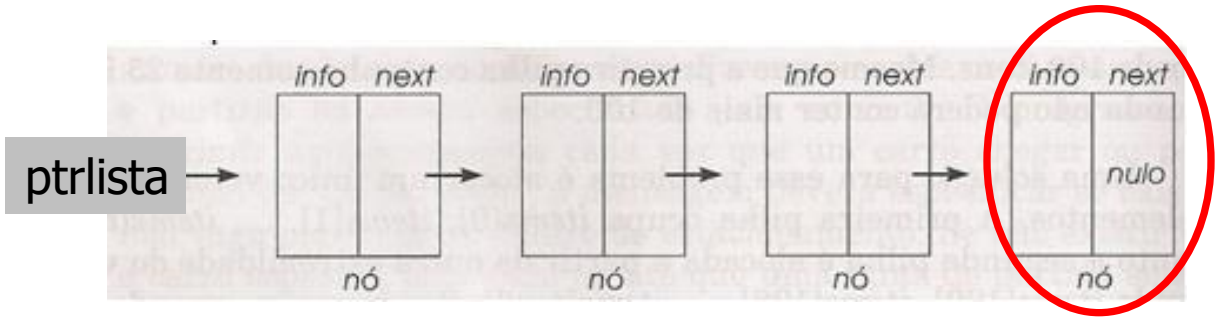
- Sendo ptrlista o endereço da primeira célula então
  - **`ptrlista->info`** é o conteúdo da célula e
  - **`ptrlista->next`** é o endereço da próxima célula



Lista Encadeada

- O campo do próximo endereço do último nó da lista contém um valor especial, conhecido como NULL, que não é um endereço válido.

- Esse ponteiro nulo (ou NULL) é usado para indicar o final de uma lista.



Lista Encadeada (implementação)

- Inicializar o ponteiro externo à lista encadeada linear com o valor NULL

```
CELULA* init (CELULA *lista){
    lista = NULL;
    return lista;
}
```

→Inicia o ponteiro para uma lista encadeada

→lista é o endereço do ponteiro externo para uma lista encadeada

Lista Encadeada (implementação)

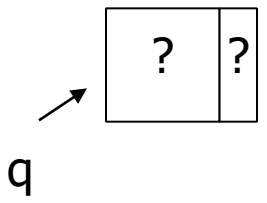
- Criar um nó dinamicamente:

```
CELULA* getnode(){
  return (CELULA *) malloc (sizeof (CELULA));
}
```

→ A função getnode() aloca/cria um nó para uma lista encadeada.

`CELULA *q = getnode();`

- obtém um nó vazio;
- q irá conter o endereço desse nó.

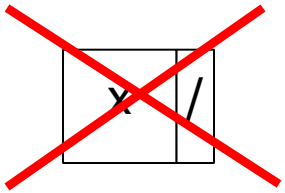


Lista Encadeada
(implementação)

- Liberar o espaço de memória ocupado por um nó.

```
void freenode(CELULA *q){
     free(q);
}
```

→ **q** é o endereço do nó a ser liberado



Lista Encadeada
(implementação)

- Verificar se a lista encadeada está vazia

```
int empty (CELULA *lista){
    if (lista == NULL)
        return 1;
    return 0;
}
```

→ lista é o endereço do primeiro nó da lista encadeada

Lista Encadeada
(Inserindo um nó no final da lista)

```
CELULA* insere_fim (CELULA *lista, int x){
    CELULA *q;
    CELULA *aux;

    q = getnode ();
    if (q != NULL){
        q->info = x;
        q->next = NULL;

        if (empty(lista))
            lista = q;
        ...
```

Lista Encadeada
(Inserindo um nó no final da lista)

```
    else{  // Percorre lista até chegar ao ultimo nó
         aux = lista;
         while (aux->next != NULL)
              aux = aux->next;

         aux->next = q;
    }
    return lista;
  }else{ // Fim do if(q != NULL)
     printf ("\nERRO na alocação do nó.\n");
     return NULL;
  }
}
```

Lista Encadeada
(Inserindo um nó no início da lista)

```
CELULA* insere_inicio (CELULA *lista, int x){
    CELULA *q;
    q = getnode();
    if(q != NULL){
        q->info = x;
        q->next = lista;
        lista = q;
        return lista;
    }else {
        printf("\nERRO na alocacao do nó.\n");
        return NULL;
    }
}
```

→ lista : ponteiro externo para uma lista encadeada

→ x : inteiro que indica o valor a ser inserido

Lista Encadeada (implementação)

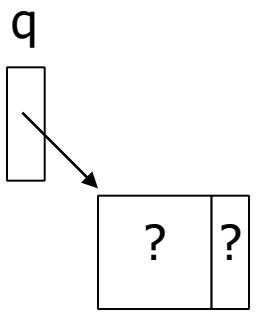
- Imprimir os elementos da lista encadeada

```
void exibe_lista (CELULA *lista){
    CELULA *aux;

    aux = lista;
    while(aux != NULL) {
        printf ("%d\t", aux->info);
        aux = aux->next;
    }
    printf ("\n");
}
```

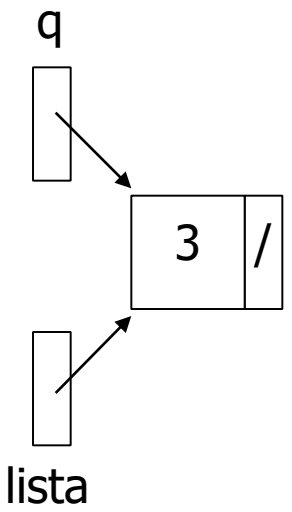
→ lista é o endereço do primeiro nó da lista encadeada

Lista Encadeada
(Criando um primeiro nó)

```
q = getnode();
if (q != NULL){
    q->info = x; /* 3 */
    q->next = NULL;
}
```



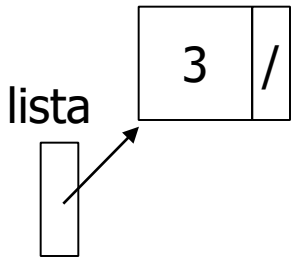

```
lista = q;
```



→Lembrando que **q** armazena o endereço do novo nó e **lista** armazena o endereço do primeiro nó da lista encadeada.

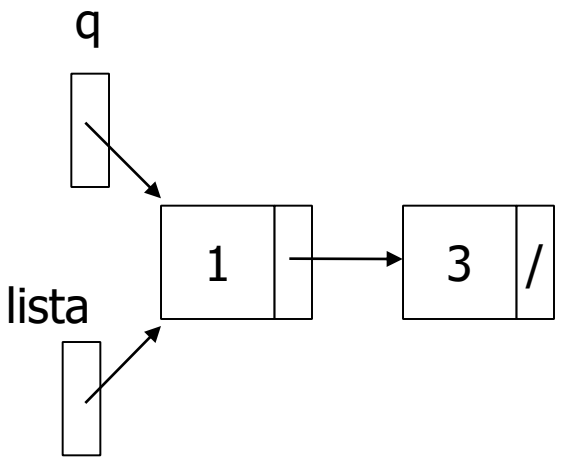
Lista Encadeada
(Inserindo um nó no início da lista)

Antes



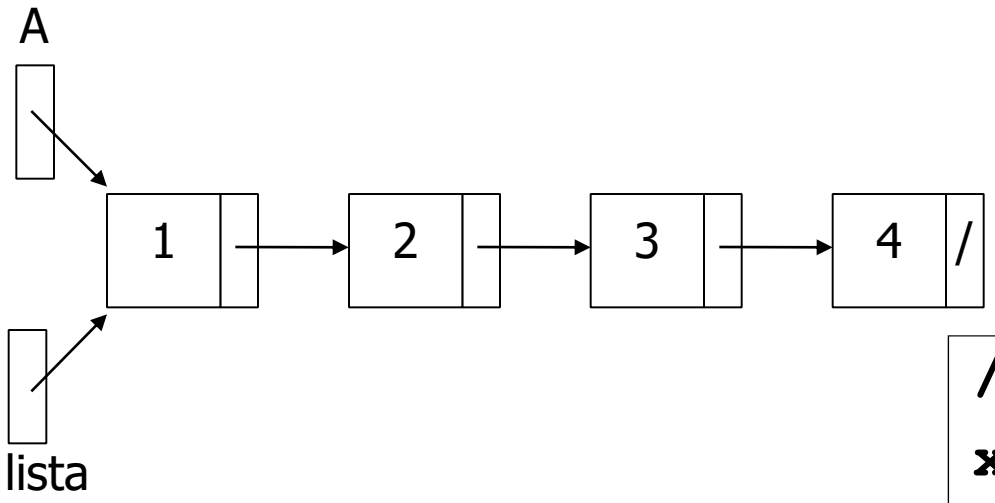

```
q = getnode();
if (q != NULL){
   q->info = x; /* 1*/
   q->next = lista;
   lista = q;
}
```

Depois



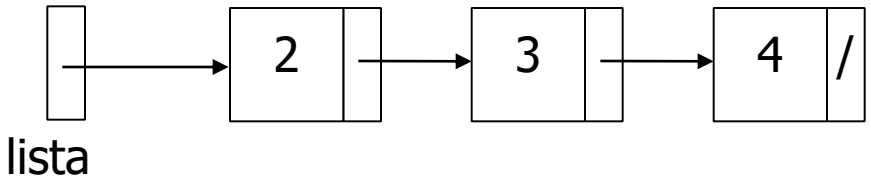
Como remover um nó no início da lista?

Antes




```
//Remoção do nó A
x = A->info;
lista = lista->next;
freenode (A);
```

Depois



Lista Encadeada
(Remover um nó no início da lista)

```
CELULA* remove_inicio(CELULA *lista){
    CELULA *q;
    q = lista;
    if (!empty(lista)) { // Há itens na lista
        lista = q->next;
        freenode (q);
        return lista;
    } else {
        printf ("\nERRO: lista vazia.\n");
        return NULL;
    }
}
```

Lista Encadeada
(Pesquisar um valor na lista)

```
CELULA* pesquisa (CELULA *lista, int x){
    CELULA *q;

    if (!empty(lista)){
        q = lista;
        while (q != NULL) {
            if (q->info == x)
                return q; //encontrou o nó
            q = q->next;
        }
    }
    return NULL; //não encontrou o nó
}
```

→ Caso exista item com valor indicado em x, retorna o endereço do primeiro nó que contém tal valor

Lista Encadeada
(Remover um nó da lista)

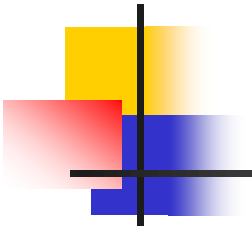
```
CELULA* remove_valor (CELULA *lista, int x){
    CELULA *q;
    CELULA *aux;

    if ((q = pesquisa (lista, x)) != NULL) {
        aux = lista;
        if (aux == q) // Nó que está no inicio da lista
            remove_inicio (lista);
        ...
```

Lista Encadeada
(Remover um nó da lista)

```
        else {
            while (aux->next != q)
                aux = aux->next;
            aux->next = q->next;
            freenode (q);
        }
        return lista; //removeu
    }
    return NULL; //não removeu
}
```

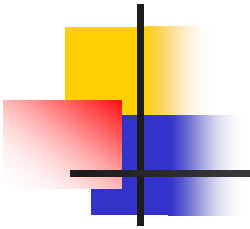
Lista Encadeada
(Função main)

```
int main(){
    CELULA *ptrlista;
    ptrlista = init(ptrlista);
    ptrlista = insere_inicio (ptrlista, 7);
    ptrlista = insere_fim (ptrlista,3);
    ptrlista = insere_fim (ptrlista,9);
    exibe_lista(ptrlista);
    ptrlista = remove_valor (ptrlista,3);
    exibe_lista(ptrlista);
    ...
    getchar();
    return 0;
}
```



Exercício

1) Crie um arquivo ListaLinearDinamica.c e implementa as seguintes informações de uma lista linear dinâmica:
   - ✓ Definição
   - ✓ Operações
     - init
     - getnode
     - freenode
     - empty
     - exibe_lista
     - insere_inicio, insere_fim
     - remove_inicio, remove_valor
     - pesquisa

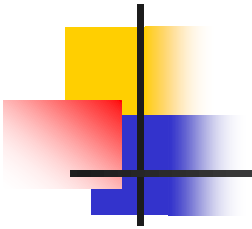


Exercício

2) Crie um arquivo AplicacaoListaLinearDinamica.c que manipule as informações de uma lista linear dinâmica a partir das funções criadas no item 1.

Crie um menu de opções para usuário com as seguintes opções:

1) Exibir elementos da lista
2) Pesquisar elemento na lista
3) Inserir elemento no início da lista
4) Inserir elemento no final da lista
5) Remover elemento do início da lista
6) Remover elemento do final da lista



Leitura Recomendada

- TENENBAUM A., LANGSAM Y. e AUGENSTEIN M. J. Estrutura de Dados usando C. Editora Makron, 1995.

→ Pág. 223, seção 4.2 - até pág. 229
→ Pág. 231 ("Operações getnode e freenode") – até pág. 233
→ Pág. 256 ("Listas ligadas usando variáveis dinâmicas) – até pág. 258
→ Pág. 260 ("Exemplos de operações de listas em C") – até pág. 262