Aula 2: Listas

Professor(a): Virgínia Fernandes Mota
http://www.dcc.ufmg.br/∼virginiaferm

ALGORITMOS E ESTRUTURAS DE DADOS - SETOR DE INFORMÁTICA

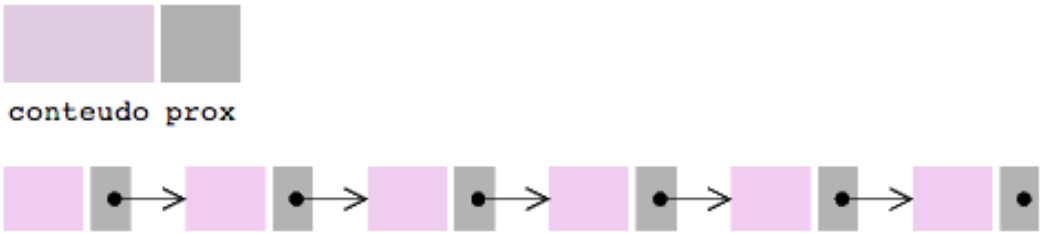
Listas Encadeadas

- Uma lista encadeada é uma representação de uma sequência de objetos na memória do computador.
- Cada elemento da sequência é armazenado em uma célula da lista: o primeiro elemento na primeira célula, o segundo na segunda e assim por diante.
- Listas Encadeadas *x* Vetores: listas são mais flexíveis e mais baratas (computacionalmente).

Listas Encadeadas

Uma **lista encadeada** (*linked list* ou lista ligada) é uma sequência de células; cada célula contém um objeto de algum tipo e o endereço da célula seguinte.

```
1 struct lista {
2     int info;
3     struct lista *prox;
4 };
5 typedef struct lista Lista;
```



O último elemento sempre apontará para ***null****.*

Função de Criação

```c
/*função de criação: retorna uma lista vazia*/

Lista* lst_cria(){
    return NULL;
}
```

- A lista vazia é representada pelo ponteiro **NULL**.

Função de Inserção

```c
/*inserção no início da lista: retorna a lista atualizada*/
Lista* lst_insere(Lista *l, int i){
    Lista *novo = (Lista *) malloc (sizeof(Lista));
    novo->info = i;
    novo->prox = l;
    return novo;
}
```

Função de Inserção

Exemplo: Criar uma lista inicialmente vazia e inserir nela novos elementos

```
1 int main(){
2     Lista *l;
3     l = lst_cria();
4     l = lst_insere(l, 23);
5     l = lst_insere(l, 42);
6     ...
7
8     return 0;
9 }
```

Existe outra maneira de fazer a inserção?

Função de Inserção

Sim!

```
1 void list_insere(Lista **l, int i){
2     Lista *novo = (Lista *) malloc (sizeof(Lista));
3     novo->info = i;
4     novo->prox = *l;
5     *l = novo;
6 }
```

Dessa forma, uma função *main* chamaria essa função do seguinte modo:
Lista *l = lst_cria();
lst_insere(&l, 42);

Percorrer e determinar lista vazia

Função que percorre os elementos da lista

```
1  /*função imprime: imprime valores dos elementos*/
2  void lst_imprime(Lista *l){
3      Lista *p;
4      for (p = l; p != NULL; p = p->prox)
5          printf("info %d ", p->info);
6  }
```

Função que verifica se a lista está vazia

```
1  /*função vazia: retorna 1 se vazia ou 0 se não vazia*/
2  int lst_vazia(Lista *l){
3      if (l == NULL)
4          return 1;
5      else
6          return 0;
7
8  //ou somente return (l == NULL);
9
10 }
```

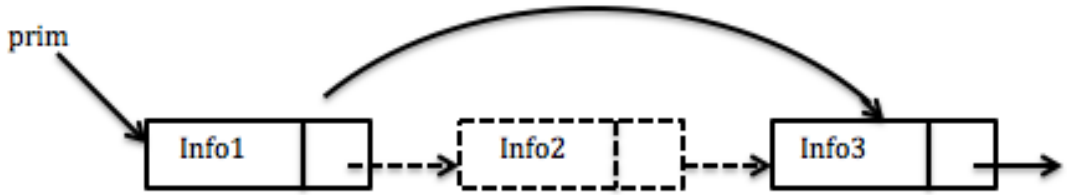
Função de Busca

```
1  /*função busca: busca um elemento na lista*/
2  Lista *lst_busca(Lista *l, int v){
3      Lista *p;
4      for (p = l; p != NULL; p->prox){
5          if (p->info == v)
6              return p;
7      }
8
9      return NULL;
10 }
```

Função para Remoção

Remoção do primeiro elemento da lista

Remoção de um elemento no meio da lista



Função para Remoção

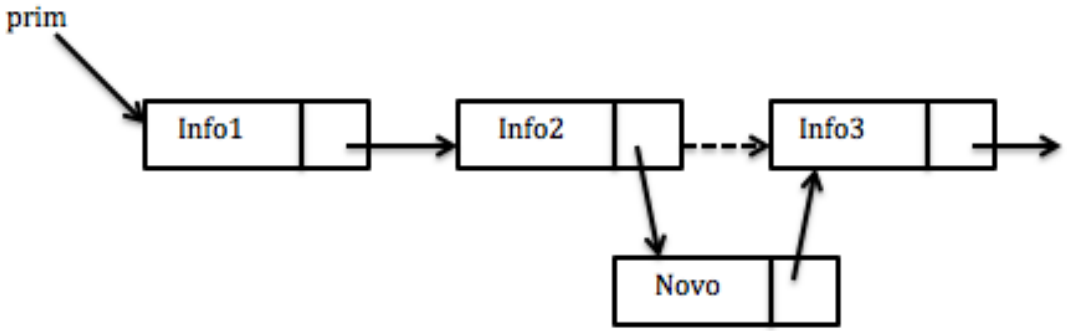
```
1  /*função retira: retira elemento da lista*/
2  Lista *lst_retira(Lista *l, int v){
3      Lista *ant = NULL;
4      Lista *p = l;
5
6      while (p != NULL && p->info != v){
7          ant = p;
8          p = p->prox;
9      }
10
11     if (p == NULL)
12         return l; // elemento não encontrado, retorna lista original
13
14     if (ant == NULL)
15         l = p->prox; // retira elemento do início
16     else
17         ant->prox = p->prox; // retira elemento do meio
18
19     free(p);
20
21     return l;
22 }
```

Função para Liberar

```c
/*função busca: busca um elemento na lista*/
void lst_libera(Lista *l){
    Lista *p = l;
    while (p != NULL){
        Lista *t = p->prox;
        free(p);
        p = t;
    }
}
```

Manutenção lista ordenada

- A função de inserção vista anteriormente armazena os elementos na lista na ordem inversa à ordem de inserção.
- Se quisermos manter os elementos da lista em uma determinada ordem, temos de encontrar a posição correta para inserir o novo elemento.



Manutenção lista ordenada

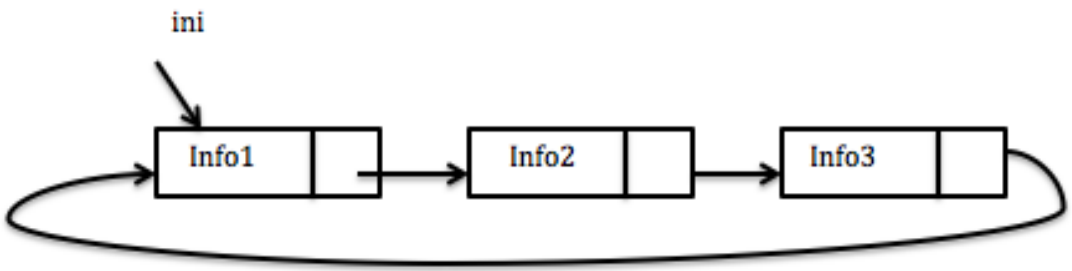
```
1  Lista *lst_insere_ordenado (Lista *l, int v){
2      Lista *novo;
3      Lista *ant = NULL;
4      Lista *p = l;
5
6      while (p != NULL && p->info < v){
7          ant = p;
8          p = p->prox;
9      }
10
11     novo = (Lista*) malloc (sizeof(Lista));
12     novo->info = v;
13
14     if (ant == NULL){
15         novo->prox = l;
16         l = novo;
17     }
18     else{
19         novo->prox = ant->prox;
20         ant->prox = novo;
21     }
22
23     return l;
24 }
```

Listas circulares

- Algumas aplicações necessitam representar conjuntos cíclicos.
- Exemplo: Manipular figuras geométricas, as arestas que delimitam uma face podem ser agrupadas por uma estrutura circular.
- Em uma **lista circular**, o último elemento tem como próximo ponteiro o primeiro elemento da lista.

Listas circulares

```c
void lcirc_imprime(Lista *l){
     Lista *p = l;
     if (p)
       do{
            printf("%d ", p->info);
            p = p->prox;
       } while (p != l);
}
```



Listas duplamente encadeadas

- Nas listas encadeadas simples não temos como percorrer os elementos na ordem inversa de maneira eficiente.
- O encadeamento simples também dificulta a retirada de um elemento.
- Na lista duplamente encadeada cada célula contém o endereço da célula anterior e o da célula seguinte.

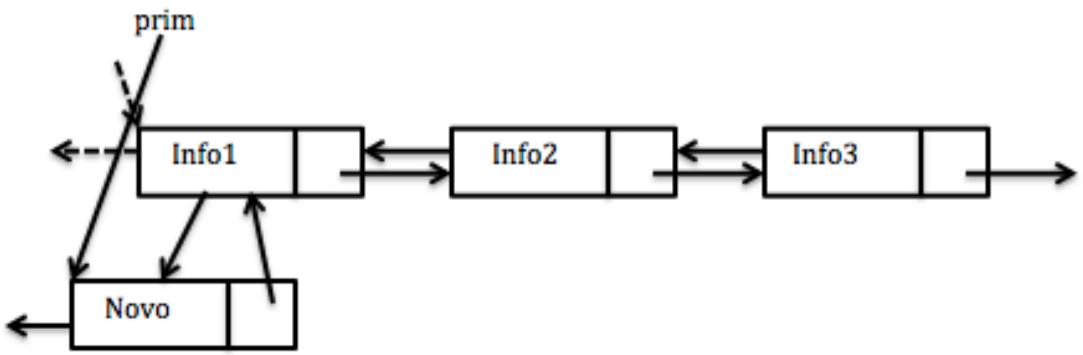
Listas duplamente encadeadas

```c
struct lista2{
    int info;
    struct lista2 *ant;
    struct lista2 *prox;
};
```

Listas duplamente encadeadas - Inserção

```c
/*insere no início da lista*/
Lista2 *lst2_insere(Lista2 *l, int v){
    Lista2 *novo = (Lista2 *) malloc (sizeof(Lista2));
    novo->info = v;
    novo->prox = l;
    novo->ant = NULL; //inserindo no início da lista
    if (l != NULL)
        l->ant = novo;

    return novo;
}
```



Listas duplamente encadeadas - Busca

```c
/*função busca: busca um elemento na lista*/

Lista2 *lst2_busca(Lista2 *l, int v){
    Lista *p;
    for (p = l; p != NULL; p = p->prox)
        if (p->info == v)
            return p;
    return NULL; //não encontrou o elemento
}
```

Listas duplamente encadeadas - Remoção

- A função de remoção se torna mais complicada, porém é possível retirar um elemento tendo apenas o ponteiro para o mesmo.

  p->ant->prox = p->prox;

  p->prox->ant = p->ant;

```
1  /*função retira: retira elemento da lista*/
2  Lista2 *lst2_retira(Lista2 *l, int v)}
3      Lista2 *p = lst2_busca(l, v);
4      if (p == NULL)
5          return l; //elemento não encontrado
6
7      if (l == p)
8          l = p->prox; //se retirar o primeiro elemento da lista
9      else
10         p->ant->prox = p->prox;
11
12     if (p->prox != NULL)  //testa se é o último elemento da lista
13         p->prox->ant = p->ant;
14
15     free(p);
16
17     return l;
18 }
```

Exercícios

1. Implementar as funções/procedimentos apresentados em sala para a manipulação de listas encadeadas, listas circulares e listas duplamente encadeadas. Crie *listas.c* e *listas.h* para a manipulação das listas e *main.c* para testes. Crie um *makefile* para compilar o código.
2. Acrescente uma função para comparar duas listas encadeadas.
3. Problema de Josephus (com 2).

Na próxima aula...

Pilhas