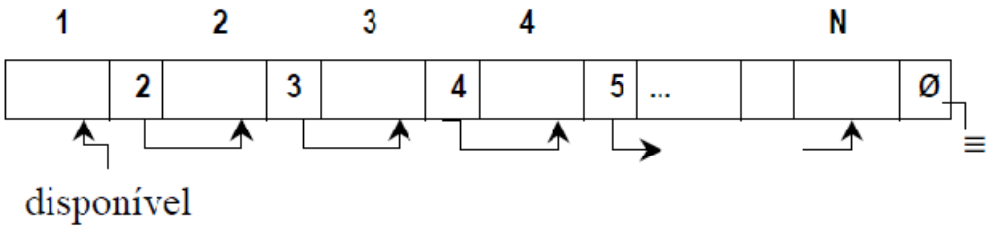
Universidade Federal de Rondônia - UNIR
Estrutura de Dados
Departamento Acadêmico de Ciências da Computação
Aula 11 - Lista Encadeada Estática
Implementação

```
#define TAMVETOR 100
#define NULO -1

/*Definicao da estrutura do no*/

typedef struct{
     int chave; /* exemplo simplificado com chave inteira, mas pode ser
qualquer tipo de dado */
     int proximo;
}no;


/* Definicao da estrutura da lista */

typedef struct {
     int inicio;
     int disponivel;
     no vetor[TAMVETOR];
}lista;
```

Inicialização da Lista Principal Vazia

Todo o vetor pertence à disponível



Lista Principal é vazia:

Operações do TAD Lista

```
/*Cria uma lista vazia. Esta funcao deve ser chamada antes de qq operacao
sobre a lista.*/
void inicializa(lista *l){
     int i;
     l->disponivel = 0;
     l->inicio = NULO;
     for (i = 0; i < (TAMVETOR - 1); i++)
          l->vetor[i].proximo = i + 1;
```

```
          l->vetor[TAMVETOR -1].proximo = NULO;
}

/*retorna 1 se lista vazia, 0 caso contrario.*/
int vazia(const lista *l){
     return (l->inicio == NULO);
}

/*retorna 1 se lista cheia, 0 caso contrario. */
int cheia(const lista *l){
     return (l->disponivel == NULO);
}


/* esvazia a lista */
void esvazia(lista *l){
     int tmp;
     if (!vazia(l))
          while (l->inicio != NULO){
               tmp = l->inicio;
               l->inicio = l->vetor[tmp].proximo;
               l->vetor[tmp].proximo = l->disponivel;
               l->disponivel = tmp;
          }
}


/* retorna o tamanho da lista */
int tamanho(const lista *l){
     int tam = 0, i;
     i = l->inicio;
     while (i != NULO){
          tam++;
          i = l->vetor[i].proximo;
     }
     return tam;
}
/* insere apos o k-esimo elemento – posição relativa */
int insere_apos_kesimo(lista *l, int k, int valor){
     int j = 0, atual, dispo;
     if (!cheia(l)){
          atual = l->inicio;
          while (j < k ){
               atual = l->vetor[atual].proximo;
               j++;
          }
          dispo = l->disponivel;
          l->disponivel = l->vetor[dispo].proximo;
          l->vetor[dispo].chave = valor;
          l->vetor[dispo].proximo = l->vetor[atual].proximo;
          l->vetor[atual].proximo = dispo;
          return 1;
     }
     return 0;
}

/* remove o elemento apos o k-esimo elemento – posição relativa */
int remove_apos_kesimo(lista *l, int k){
```

```
      int j, atual;
      if (tamanho(l) > k){
            atual = l->inicio;
            while (j < k ){
                  atual = l->vetor[atual].proximo;
                  j++;
            }
            j = l->vetor[atual].proximo;
            l->vetor[atual].proximo = l->vetor[j].proximo;
            l->vetor[j].proximo = l->disponivel;
            l->disponivel = j;
            return 1;
      }
      return 0;
}

/* insere apos o elemento de indice k */
int insere_apos(lista *l, int k, int valor){
      int dispo;
      if (!cheia(l)){
            dispo = l->disponivel;
            l->disponivel = l->vetor[dispo].proximo;
            l->vetor[dispo].chave = valor;
            l->vetor[dispo].proximo = l->vetor[k].proximo;
            l->vetor[k].proximo = dispo;
            return 1;
      }
      return 0;
}


/* remove o elemento apos o elemento do indice k */
int remove_apos(lista *l, int k){
      int prox;
      if (!vazia(l)){
            prox = l->vetor[k].proximo;
            l->vetor[k].proximo = l->vetor[prox].proximo;
            l->vetor[prox].proximo = l->disponivel;
            l->disponivel = prox;
            return 1;
      }
      return 0;
}

/* Insere um elemento no inicio da lista */
int insere_inicio(lista *l, int valor){
      int dispo;
      if (!cheia(l)){
            dispo = l->disponivel;
            l->disponivel = l->vetor[dispo].proximo;
            l->vetor[dispo].chave = valor;
            l->vetor[dispo].proximo = l->inicio;
            l->inicio = dispo;
            return 1;
      }
      return 0;
}
```

```
/* Remove o elemento do inicio da lista */
int remove_inicio(lista *l){
      int i;
      if (!vazia(l)){
            i = l->inicio;
            l->inicio = l->vetor[i].proximo;
            l->vetor[i].proximo = l->disponivel;
            l->disponivel = i;
            return 1;
      }
      return 0;
}

/* Inserção ordenada */
int insere_ord(lista *l, int valor){
      int atual, dispo, anterior = NULO;
      if (!cheia(l)){
            atual = l->inicio;
            if (vazia(l))
                  insere_inicio(l, valor);
            else {
                  while ((l->vetor[atual].proximo != NULO)&& (l-
>vetor[atual].chave < valor)){
                        anterior = atual;
                        atual = l->vetor[atual].proximo;
                  } /* fim while */
                  if (anterior == NULO)
                        insere_inicio(l, valor);
                  else {
                        dispo = l->disponivel;
                        l->disponivel =
                        l->vetor[dispo].proximo;
                        l->vetor[dispo].chave = valor;
                        if (l->vetor[atual].chave >= valor){
                              l->vetor[dispo].proximo = atual;
                              l->vetor[anterior].proximo = dispo;
                        } else {
                              l->vetor[dispo].proximo = NULO;
                              l->vetor[atual].proximo = dispo;
                        } /* fim else */
                  } /* fim else */
            } /* fim else */
            return 1;
      } /* fim if cheia */
      return 0;
}


/* Remove um item de valor x da lista. Retorna 1 se sucesso, 0 caso
contrário.*/
int remove_ord(lista *l, int valor){
      int atual, anterior = NULO;
      if (!vazia(l)){
            atual = l->inicio;
            while ((l->vetor[atual].proximo != NULO)&& (l-
>vetor[atual].chave < valor)){
                  anterior = atual;
                  atual = l->vetor[atual].proximo;
```

```
          } /* fim while */
          if (l->vetor[atual].chave == valor){
               if (anterior == NULO)
                    remove_inicio(l);
               else {
                    l->vetor[anterior].proximo =
                    l->vetor[atual].proximo;
                    l->vetor[atual].proximo = l->disponivel;
                    l->disponivel = atual;
               } /* fim else */
               return 1;
          } /* fim if */
     } /* fim if vazia */
     return 0;
}
```

***Retorna (busca) o endereço de x na Lista. Se x ocorre mais de uma vez, retorna o endereço da primeira ocorrência. Se x não aparece retorna NULO. */**

```
int localizar(const lista *l, int valor){
     int atual, anterior = NULO;
     if (!vazia(l)){
          atual = l->inicio;
          while ((l->vetor[atual].proximo != NULO) && (l-
>vetor[atual].chave != valor)){
               anterior = atual;
               atual = l->vetor[atual].proximo;
          }
          if (l->vetor[atual].chave == valor)
               return atual;
     }
     return NULO;
}
```

/*Retorna (busca) o endereço de x na Lista. Se x ocorre mais de uma vez, retorna o endereço da primeira ocorrência. Se x não aparece retorna NULO. */

```
int localizar_ord(const lista *l, int valor){
     int atual, anterior = NULO;
     if (!vazia(l)){
          atual = l->inicio;
          while ((l->vetor[atual].proximo != NULO) && (l-
>vetor[atual].chave < valor)){
               anterior = atual;
               atual = l->vetor[atual].proximo;
          }
          if (l->vetor[atual].chave == valor)
               return atual;
     }
     return NULO;
}
```

/* recupera o valor do elemento da posição absoluta k */

```
int buscar(const lista *l, int k){
     return l->vetor[k].chave;
```

```
}


/* Imprime a lista */
void imprimir(const lista *l){
     int i;
     if (!vazia(l)){
          printf("\nLista ");
          i = l->inicio;
          while (i != NULO){
               printf("%d ", l->vetor[i].chave);
               i = l->vetor[i].proximo;
          }
     } else printf("\nLista vazia!");
}
```