UNISINOS - **UNIVERSIDADE DO VALE DO RIO DOS SINOS**
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS (C6/6)
PIP/CA - Programa Interdisciplinar de Pós-Graduação em Computação Aplicada

ALGORITMOS
&
ESTRUTURAS DE DADOS

Professor responsável: *Fernando Santos Osório* **E-mail:** ***osorio@exatas.unisinos.br***
Semestre: 2000/1 **Web:** ***http://www.inf.unisinos.br/~osorio/***
Horário: Noite - de 21/02 à 25/02/2000

Estruturas de Dados com Alocação Dinâmica

1. Alocação de Dinâmica de Memória:

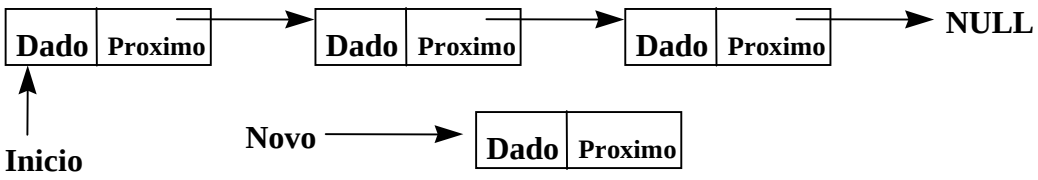
Vamos ver inicialmente um exemplo de uso da alocação dinâmica de memória em "C":

```
typedef    struct  {
                        char   nome [30];
                        int      idade;
                        double  salario;
                    }   registro_func;

main ( )
{
  int       *ptr_int;
  double *ptr_pf;
  registro_func  *ptr_reg;

  ptr_int =  (int * ) calloc ( 10 , sizeof (int) );     /* Aloca 10 inteiros em seqüência */
  ptr_pf  = (double * ) calloc (10, sizeof (double) );  /* Aloca 10 nros. tipo double */
  ptr_reg = (registro_func *) calloc(1, sizeof (registro_func) );  /* Aloca 1 registro*/

  *ptr_int = 1;          /* Coloca o valor 1 no primeiro inteiro da seqüência alocada */
  *(ptr_int+5) = 2;    /* Coloca o valor 2 no sexto  valor inteiro do bloco alocado */
  ptr_int[7] = 3;        /* Coloca o valor 3 no oitavo valor inteiro do bloco alocado */

  strcpy((*ptr_reg).nome, "Fulano");
  (*ptr_reg).idade = 18;
  (*ptr_reg).salario = 100.00;

  strcpy(ptr_reg -> nome, "Ciclano");
  ptr_reg -> idade = 60;
  ptr_reg -> salario = 9999.99;

  free (ptr_int);  free (ptr_pf);  free (ptr_reg);  /* Libera a área de memória alocada */
}
```

As principais características da *alocação dinâmica* de memória são:

- Os dados se encontram espalhados na memória do computador em vários blocos;
- Geralmente as estruturas de dados são compostas por registros que incluem um campo do tipo ponteiro (*elo/link*), o que nos permite encadear os dados e indicar onde está o dado seguinte na memória (visto que os dados estão espalhados na memória do computador);
- Podemos ir solicitando mais e mais memória a medida em que precisarmos de mais espaço para armazenar as informações. Isso nos permite criar programas que usam apenas a memória necessária... e por conseqüência, podemos rodar outros programas, em uma mesma máquina, sem que um único programa monopolize toda a memória disponível desta.
- Podemos liberar espaços de memória quando estes não forem mais necessários ao programa.

Estas estruturas criadas com registros que são encadeados através do uso de ponteiros, que estes blocos espalhados pela memória, vão resultar em estruturas denominadas de **"listas encadeadas"** de alocação dinâmica. A figura abaixo mostra um exemplo de lista encadeada, baseada na estrutura de dados descrita logo mais abaixo:



Usualmente teremos o seguinte tipo de estrutura de dados para descrever uma lista encadeada:

```
#include <stdio.h>

typedef    int    Tipo_Dado;

typedef    struct  bloco {
                              Tipo_Dado      Dado;
                              struct bloco     *Proximo;
                            }   Nodo;

main ( )
{
    Nodo  *Inicio;     /* Uma vez criada a lista encadeado não devemos
                          perder o ponteiro que aponta para o início da lista !   */
    Nodo  *Novo;     /* Usado para apontar para os novos registro alocados  */
    Nodo  *Atual;     /* Usado para percorrer a lista, sem perder o inicio       */
```

```
    /* Criando uma lista de 2 nodos apenas */

    /* Alocação dinâmica: cria o primeiro registro (nodo) */
    Novo = (Nodo *) calloc ( 1, sizeof (Nodo) );
    /* Inicio aponta para o primeiro registro da lista */
    Inicio = Novo;
    Novo->Dado = 1;
    Novo->Proximo = NULL;   /*  NULL = Fim   */

    /* Alocação dinâmica: cria o segundo registro */
    Novo = (Nodo *) calloc ( 1, sizeof (Nodo) );
    /* O proximo depois do inicio é o segundo registro */
    Inicio->Proximo = Novo;
    Novo->Dado = 2;
    Novo->Proximo = NULL;
}
```

Podemos trabalhar com listas encadeadas de diferentes tipos, resultando nas estruturas de dados descritas a seguir.

- Conjuntos de estruturas de dados com alocação dinâmica a serem estudados:

  A) *Listas encadeadas simples*
  B) *Listas duplamente encadeadas*
  C) *Filas*
  D) *Pilhas*
  E) *Árvores*

---

2. Listas encadeadas simples:

Construção de um conjunto de rotinas genéricas e modulares de manipulação de listas encadeadas simples. Vamos aqui descrever uma possível proposta de um conjunto de rotinas empregadas para este fim.

Rotinas básicas de manipulação de listas encadeadas simples:
- Estruturas de dados com alocação dinâmica
- Inserção no inicio da lista, no final da lista, ou de modo ordenado
- Remoção de qualquer elemento da lista (início, final, ou elemento especificado)

Aplicação típica:
- Lista de elementos de tamanho variável

- Estrutura de dados:

```
typedef    int    Tipo_Dado;

typedef    struct  bloco {
                                Tipo_Dado      Dado;
                                struct bloco      *Proximo;
                             }   Nodo;
```

- Rotinas:

```
void inicializa_lista       (Nodo **N);
int insere_inicio_lista     (Nodo **N, Tipo_Dado Dado);
int insere_fim_lista        (Nodo **N, Tipo_Dado Dado);
int insere_ordenando_lista (Nodo **N, Tipo_Dado Dado);
int remove_inicio_lista (Nodo **N, Tipo_Dado *Dado);
int remove_fim_lista     (Nodo **N, Tipo_Dado *Dado);
int remove_elemento_lista (Nodo **N, Tipo_Dado Dado);
int quantidade_lista      (Nodo *N);
void exibe_lista          (Nodo *N);
int pesquisa_lista        (Nodo **N, Tipo_Dado Dado);
int percorre_lista        (Nodo **N, Tipo_Dado *Dado);
void apaga_lista          (Nodo **N);
```

- Exemplo: inserção de dados no inicio da lista encadeada

```
int insere_inicio_lista (N, Dado)
Nodo **N;
Tipo_Dado Dado;
{
        Nodo   *novo;

        novo = (Nodo *) calloc ( 1, sizeof (Nodo) );
        if  ( novo == NULL )
             return (ERRO);       /* Não conseguiu alocar espaço na memória */
        novo -> Dado = Dado;
        novo -> Proximo = *N;
        *N = novo;
        return (OK);
}
```

- Exemplo: inserção de dados no final da lista encadeada

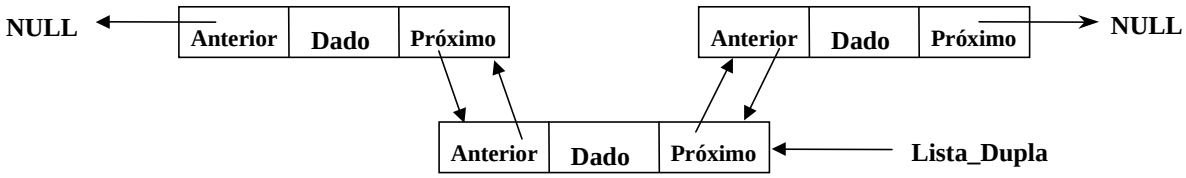
```
int insere_fim_lista (N, Dado)
Nodo **N;
Tipo_Dado Dado;
{
        Nodo  *aux,  *novo;

        novo = (Nodo *) calloc ( 1, sizeof (Nodo) );
        if  ( novo == NULL )
             return(ERRO);
```

```
        novo -> Dado = Dado;
        novo -> Proximo = NULL;

        if ( *N == NULL )
            *N = novo;
        else  {
                aux = *N;
                while ( aux -> Proximo  !=  NULL )
                        aux = aux -> Proximo;
                aux -> Proximo = novo;
              }

        return (OK);
}
```

---

4. Listas duplamente encadeadas

Construção de um conjunto de rotinas genéricas e modulares de manipulação de listas duplamente encadeadas. Vamos aqui descrever uma possível proposta de um conjunto de rotinas empregadas para este fim.



Rotinas básicas de manipulação de listas duplamente encadeadas:
- Estruturas de dados com alocação dinâmica
- Inserção no inicio da lista, no final da lista, ou de modo ordenado
- Remoção de qualquer elemento da lista (início, final, ou elemento especificado)

Aplicação típica:
- Lista de elementos de tamanho variável, permite percorrer a lista nos dois sentidos

- Estrutura de dados:

```
        typedef   int   Tipo_Dado;

        typedef   struct  bloco_ld  {
                                        Tipo_Dado          Dado;
                                        struct bloco_ld    *Proximo, *Anterior;
                                  }  Nodo_LD;
```

- Rotinas:

```
void inicializa_ld          (Nodo_LD **LD);
void posiciona_inicio_ld    (Nodo_LD **LD);
void posiciona_fim_ld       (Nodo_LD **LD);
int insere_antes_ld         (Nodo_LD **LD, Tipo_Dado Dado);
int insere_depois_ld        (Nodo_LD **LD, Tipo_Dado Dado);
int insere_inicio_ld        (Nodo_LD **LD, Tipo_Dado Dado);
int insere_fim_ld           (Nodo_LD **LD, Tipo_Dado Dado);
int insere_ordenando_ld     (Nodo_LD **LD, Tipo_Dado Dado);
int pesquisa_ld             (Nodo_LD **LD, Tipo_Dado Dado);
int remove_nodo_ld          (Nodo_LD **LD, Tipo_Dado *Dado);
int remove_dado_ld          (Nodo_LD **LD, Tipo_Dado *Dado);
int quantidade_ld           (Nodo_LD *LD);
void exibe_ld               (Nodo_LD *LD);
int percorre_lista          (Nodo_LD **LD, Tipo_Dado *Dado);
void apaga_ld               (Nodo_LD **LD);
```

- Exemplo: posiciona o ponteiro no início da lista duplamente encadeada

```
void posiciona_inicio_ld (LD)
Nodo_LD   **LD;
{
        if  ( *LD != NULL )
        {
            while (  (*LD) -> Anterior != NULL )
                    *LD = (*LD) -> Anterior;
        }
}
```

- Exemplo: inserção de dados antes da posição apontada na lista duplamente encadeada

```
int insere_antes_ld (LD, Dado)
Nodo_LD   **LD;
Tipo_Dado Dado;
{
        Nodo_LD   *novo, *aux;

        novo = (Nodo_LD *) calloc ( 1, sizeof (Nodo_LD) );
        if  ( novo == NULL )
             return (ERRO);
        novo -> Dado = Dado;

        if ( *LD == NULL )
        {
            *LD = novo;
             novo -> Anterior = NULL;
             novo -> Proximo = NULL;
        }
```

```
        else
        {
            aux = (*LD) -> Anterior;
            (*LD) -> Anterior = novo;
            novo -> Proximo = (*LD)
            novo -> Anterior = aux;
            if  ( aux != NULL )
                aux -> Proximo = novo;
        }
        return (OK);
}
```

- Exemplo: inserção de dados no início da lista duplamente encadeada

```
int insere_inicio_ld (LD, Dado)
Nodo_LD   **LD;
Tipo_Dado Dado;
{
        posiciona_inicio_ld(LD);
        return ( insere_antes_ld (LD, Dado) );
}
```

---

4. Filas, Pilhas, Deques e Ordenação usando alocação dinâmica:

As filas, pilhas e deques podem ser facilmente implementadas usando rotinas genéricas de manipulação de listas, sejam estas simples ou duplamente encadeadas. Uma vez que podemos implementar facilmente rotinas de inserção no início e/ou no fim de uma lista encadeada, e também podemos implementar facilmente rotinas de remoção do início e/ou do fim de uma lista encadeada com alocação dinâmica, por conseqüência, podemos criar rotinas de manipulação destas outras estruturas de dados baseadas nas rotinas descritas anteriormente. É importante salientar que em alguns casos talvez seja interessante criar rotinas mais otimizadas, como por exemplo, na inserção de nodos no final de uma lista. Neste caso, seria interessante que fosse evitada a necessidade de percorrer toda a lista até encontrar o nodo final, cada vez que um novo nodo fosse adicionado no final desta lista.

- Filas:
  - Insere no início
  - Retira do fim

- Pilhas:
  - Insere no fim
  - Retira do fim

- Deques:
  - Insere no início ou no fim
  - Retira do início ou do fim

- Ordenação: A inserção pode ser feita em qualquer posição, o que facilita a ordenação