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

Lista Lineares – Lista Seqüencial com Alocação Estática

1. Alocação de Memória:

Ao criar um programa em "C" usualmente temos que especificar, antes de começar a executar o programa, as variáveis que vamos usar, reservando assim um espaço na memória. As variáveis que são alocadas em posições fixas da memória são chamadas de *variáveis estáticas*, e as variáveis que não possuem uma posição fixa, e que são criadas e destruídas durante a execução do programa, são chamadas de *variáveis dinâmicas*.

A alocação de memória no computador pode ser dividida em dois grupos principais:

- Alocação Estática: os dados tem um tamanho fixo e estão organizados seqüencialmente na memória do computador. Um exemplo típico de alocação estática são as variáveis globais, e os vetores (estáticos).
- Alocação Dinâmica: os dados não precisam ter um tamanho fixo, pois podemos definir para cada dado quanto de memória que desejamos usar. Sendo assim vamos alocar espaços de memória (blocos) que não precisam estar necessariamente organizados de maneira seqüencial, podendo estar distribuídos de forma esparsa na memória do computador. Na alocação dinâmica, vamos pedir para alocar/desalocar blocos de memória, de acordo com a nossa necessidade, reservando ou liberando blocos de memória durante a execução de um programa. Para poder "achar" os blocos esparsos na memória usamos as variáveis do tipo *Ponteiro* (indicadores de endereços de memória). As variáveis locais e os parâmetros passados por valor são alocados dinamicamente.

Os comandos em "C" para alocar explicitamente blocos de memória de forma dinâmica, são: calloc e malloc. Exemplos:

```
int  *ptr_int;
double *ptr_pf;
```

```
ptr_int =  (int * ) calloc ( 10 , sizeof (int) );     /* Aloca 10 inteiros em seqüência */
ptr_pf  = (double * ) calloc (10, sizeof (double) );  /* Aloca 10 nros. tipo double */
free (ptr_int);                                       /* Libera a área de memória alocada */
```

Vamos estudar aqui a construção de estruturas de dados em "C", usando estes dois tipos de métodos de alocação de memória: estática e dinâmica. O primeiro tipo de estrutura de dados a ser abordado, que é muito usado no desenvolvimento de programas e aplicações, são as listas. As listas servem para armazenar um conjunto de dados, sejam eles agrupados (vetores = *arrays*) ou não (listas encadeadas).

1.1. Conjunto de dados com alocação estática:

A) *Listas lineares seqüenciais – Vetores Simples*
B) *Filas*
C) *Pilhas*
D) *Deques*

1.2. Conjuntos de dados com alocação dinâmica:

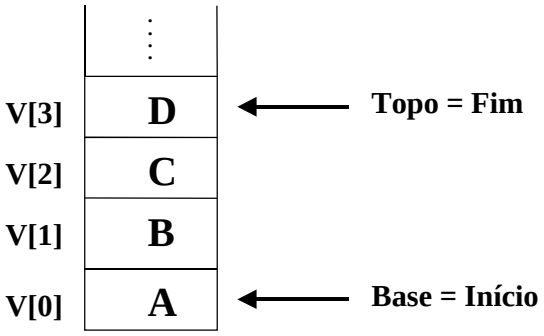
A) *Listas encadeadas simples*
B) *Filas*
C) *Pilhas*
D) *Listas duplamente encadeadas*
E) *Árvores*

---

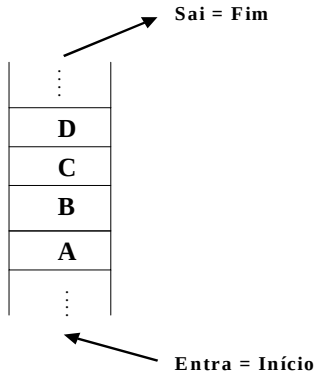
2. Alocação Estática de Memória:

As estruturas de dados para armazenar um conjunto de dados (exemplo: um cadastro com vários registros), podem ser organizadas de formas diferentes, de acordo com a maneira que os dados são inseridos e retirados da memória do computador. As formas mais usuais são: as listas lineares seqüenciais (vetores simples), as filas, as pilhas e os deques.

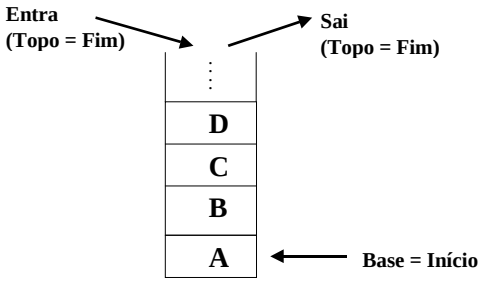
2.1. Listas lineares seqüenciais



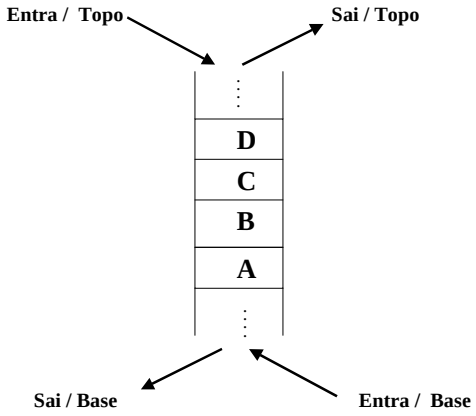
2.2. Filas – **FIFO** = “First In, First Out”



2.3. Pilhas – **LIFO** = “Last In, First Out”



2.4. Deque – “Double Ended Queue”



3. Lista Lineares Seqüenciais

Uma maneira interessante de manipular este tipo de estruturas de dados, é através da construção de um conjunto de rotinas genéricas e modulares de manipulação de listas seqüenciais. Vamos aqui descrever uma possível proposta de um conjunto de rotinas empregadas para este fim.

Rotinas básicas de manipulação de Vetores:
- Estruturas de dados com alocação estática
- Inserção no final do vetor
- Remoção lógica dos elementos

Aplicação típica:
- Pequenos cadastros

- Estrutura de dados:

```
typedef int Tipo_Dado;

typedef struct   {
                    Tipo_Dado   Dado [MAX_VETOR];
                    int         Excluido [MAX_VETOR];
                    int         Inicio, Fim;
                 }  Tipo_Vetor;
```

- Rotinas:

```
void inicializa_vetor   (Tipo_Vetor *V);
int  insere_vetor       (Tipo_Vetor *V; Tipo_Dado Dado);
int  consulta_vetor     (Tipo_Vetor  V; int Indice; Tipo_Dado *Dado);
int  acha_vetor         (Tipo_Vetor  V; Tipo_Dado Dado; int *Indice);
void lista_vetor        (Tipo_Vetor  V);
int  exclui_vetor       (Tipo_Vetor *V; int Indice);
int  atualiza_vetor     (Tipo_Vetor *V; int Indice; Tipo_Dado Novo_Dado);
void compacta_vetor     (Tipo_Vetor *V);
int  vazio_vetor        (Tipo_Vetor  V);
int  quantidade_vetor   (Tipo_Vetor  V);
```

- Exemplo: inserção de dados no vetor

```
int insere_vetor (V, Dado)
Tipo_Vetor *V;
Tipo_Dado Dado;
{
        if (V->Fim < MAX_VETOR)         /* Vetor nao esta cheio ? */
        {
            V->Dado[V->Fim]=Dado;
            V->Excluido[V->Fim]=FALSO;
            (V->Fim)++;
           return(OK);
        }
        else
           return(ERRO);
}
```

4. Filas – FIFO

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

Rotinas básicas de manipulação de FILAS usando vetores:
- Estruturas de dados com alocação estática
- Inserção no final da fila
- Remoção do início da fila
- Fila circular

Aplicação típica:
- Lista de elementos a espera de um tratamento

- Estrutura de dados:

```
typedef int Tipo_Dado;

typedef struct   {
                    Tipo_Dado   Dado [MAX_FILA];
                    int         Inicio, Fim;
                 }  Tipo_Fila;
```

- Rotinas:

```
void inicializa_fila    (Tipo_Fila *F);
int  insere_fila        (Tipo_Fila *F; Tipo_Dado  Dado);
int  retira_fila        (Tipo_Fila *F; Tipo_Dado *Dado);
void lista_fila         (Tipo_Fila  F);
int  consulta_fila      (Tipo_Fila  F; int Indice; Tipo_Dado *Dado);
int  cheia_fila         (Tipo_Fila  F);
int  vazia_fila         (Tipo_Fila  F);
int  quantidade_fila (Tipo_Fila  F);
int  acha_fila          (Tipo_Fila  F; Tipo_Dado Dado; int *Indice);
```

- Exemplo: inserção de dados na fila

```
int insere_fila (F, Dado)
Tipo_Fila *F;
Tipo_Dado Dado;
{
   int prox;

   prox=F->Fim+1;
   if (prox == MAX_FILA)
      prox=0;
   if (prox == F->Inicio)
      return(ERRO);
   else
   {
      F->Dado[F->Fim]=Dado;
      F->Fim=prox;
      return(OK);
   }
}
```

5. Pilhas – LIFO:

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

Rotinas básicas de manipulação de PILHAS usando vetores:
- Estruturas de dados com alocação estática
- Inserção no topo da pilha
- Remoção do topo da pilha

Aplicação típica:
- Lista de "tarefas pendentes", passagem de parâmetros nas linguagens de programação

- Estrutura de dados:

```
typedef int Tipo_Dado;

typedef struct   {
                    Tipo_Dado   Dado [MAX_PILHA];
                    int         Base, Topo;
                 }  Tipo_Pilha;
```

- Rotinas:

```
void inicializa_pilha   (Tipo_Pilha *P);
int  insere_pilha       (Tipo_Pilha *P; Tipo_Dado  Dado);
int  retira_pilha       (Tipo_Pilha *P; Tipo_Dado *Dado);
void exibe_pilha        (Tipo_Pilha  P);
int  quantidade_pilha   (Tipo_Pilha P);
int  cheia_pilha        (Tipo_Pilha  P);
int  vazia_pilha        (Tipo_Pilha  P);
void esvazia_pilha      (Tipo_Pilha *P);
```

- Exemplo: inserção de dados na pilha

```
int insere_pilha (P, Dado)
Tipo_Pilha *P;
Tipo_Dado Dado;
{
   ...  /* Vide rotinas de manipulação de vetores (lista linear seqüencial)  */
        /* Insere no topo... */
}
```

6. Deque – Double Ended Queue

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

Rotinas básicas de manipulação de DEQUES usando vetores:
- Estruturas de dados com alocação estática
- Inserção no início **ou** no final do deque
- Remoção do início **ou** do final do deque
- Estrutura de dados "circular"

Aplicação típica:
- Lista de elementos com múltiplas formas de considerar/manipular a ordem destes

- Estrutura de dados:

```
typedef int Tipo_Dado;

typedef struct   {
                   Tipo_Dado   Dado [MAX_DEQUE];
                   int              Inicio, Fim;
                 }  Tipo_Deque;
```

- Rotinas:

```
void inicializa_deque      (Tipo_Deque *D);
int  insere_inicio_deque  (Tipo_Deque *D; Tipo_Dado  Dado);
int  insere_final_deque    (Tipo_Deque *D, Tipo_Dado Dado);
int  retira_inicio_deque   (Tipo_Deque *D; Tipo_Dado *Dado);
int  retira_final_deque    (Tipo_Deque *D; Tipo_Dado *Dado);
void lista_deque           (Tipo_Deque  D);
int  acha_deque            (Tipo_Deque  D; Tipo_Dado Dado; int *Indice);
int  cheio_deque           (Tipo_Deque  D);
int  vazio_deque           (Tipo_Deque  D);
int  quantidade_deque      (Tipo_Deque  D);
void  apaga_deque          (Tipo_Deque *D);
```

- Exemplo: inserção de dados no início do deque

```
int insere_inicio_deque (D, Dado)
Tipo_Deque *D;
Tipo_Dado Dado;
{
  ...
}
```

- **<u>TRABALHO PRÁTICO</u>**: implementar as rotinas descritas acima (inicializa_deque, insere_ inicio_deque, insere_final_deque, retira_inicio_deque, retira_final_deque e lista_deque). Fazer um programa que permita executar estas rotinas através de um menu com as seguintes opções: insere inicio, insere final, retira inicio, retira final, lista e sair do programa. O usuário do programa poderá executar interativamente as rotinas implementadas, inserindo e removendo dados e também vendo como ficaram os dados no deque.