**LISTA ESTÁTICA**

DEFINIÇÃO

- Lista é uma sequência de elementos de mesmo tipo (array), e os elementos se apresentam de forma consecutiva na memória.

- Vantagens:

* acesso rápido e direto aos elementos (através do índice);
* tempo constante para acessar um elemento;
* facilidade em modificar dados.

- Desvantagens:

* definição prévia do tamanho do array;
* dificuldade em inserção e remoção de elementos, por conta da necessidade de deslocamento de outros elementos.

- Casos em que é desejável se utilizar lista estática: listas pequenas, inserção/remoção apenas no final, tamanho bem definido e busca é a operação mais frequente.

- Na definição da lista, é necessário: • variável quantidade – quantidade de elementos na lista; • vetor dos elementos.

- Ex.:

struct lista{

int qtd;

struct aluno dados[MAX];

};

typedef struct lista Lista; // apelido para a lista

Lista \*li;

**LISTA NÃO ORDENADA**

Modelagem

struct usuario{

int chave;

};

// definição lista

struct lista{

int qtd;

struct usuario usuarios[MAX];

};

typedef struct lista Lista;

Inicialização da lista

- Inicializamos um array já criado pelo usuário (na *main*, com um tamanho definido). Só precisamos atualizar a quantidade de elementos da lista com 0.

void inicializa\_lista(Lista\* li){

li->qtd = 0;

}

Inserção

bool insere\_posicao(Lista\* li, struct usuario u, int i){

int k;

// casos em que não é possível inserir: lista não existe,

//a lista está cheia ou a posição i é inválida

if(li == NULL || li->qtd == MAX || i<0 || i>li->qtd)

return false;

// posicionamos na primeira posição livre ao fim da lista e,

// de trás para frente, vamos atribuindo que a posição atual

// recebe a anterior

for(k=li->qtd; k>i; k--)

li->usuarios[k] = li->usuarios[k-1];

// a posição desejada recebe o novo usuário

li->usuarios[i] = u;

// incrementa o tamanho da lista

li->qtd++;

return true;

}

Busca

// busca a posição do elemento desejado

// retorna -1 se a lista não existe ou

// se o elemento não foi encontrado

int busca\_por\_elemento(Lista\* li, int chave){

if(li == NULL)

return -1;

int i = 0;

while(i<li->qtd && li->usuarios[i].chave != chave)

i++;

if(i == li->qtd) //elemento nao encontrado

return 0;

return i;

}

// busca o elemento da posição desejada e

// retorna um usuario

// retorna -1 se a lista não existe ou

// se o elemento não foi encontrado

int busca\_por\_posicao(Lista\* li, int posicao, struct usuario \*u){

if(li == NULL)

return -1;

u = li[posicao].usuarios;

return u;

}

**LISTA ORDENADA**

OPERAÇÕES

Cria lista

Lista \*cria\_lista(){

Lista li \*; // ponteiro para a nova lista

li = (Lista \*)malloc(sizeof(struct lista)); // tipo de dado e tamanho; retorna um ponteiro

if (li != NULL) // se deu certo, o campo qtd da lista é = 0, isto é, a lista é inicializada com 0 elementos

li->qtd = 0;

return li;

}

Inserção

int insere\_lista\_ordenada(Lista \*li, struct aluno al){

if (li == NULL)

return 0;

if (li->qtd == MAX)

return 0;

int k, i;

// i itera sobre a lista para encontrar a posição em que a nova matrícula deve entrar, isto é, até onde al.matricula é menor que a matrícula atual

while (i < li->qtd && li->dados[i].matricula < al.matricula)

i++;

for (k = li->qtd - 1; k >= i; k--) // começando do final da lista, até encontrar a posição i

li->dados[k + 1] = li->dados[k]; // cada posição recebe o elemento anterior, de forma que cada elemento fica deslocado em 1

li->dados[i] = al; // posição i recebe o novo elemento

li->qtd++; // incrementa a quantidade

return 1;

Remoção

int remove\_lista(Lista \*li, int mat){

if (li == NULL)

return 0;

if (li->qtd == 0)

return 0;

// i itera sobre a lista até encontrar o elemento desejado

while (i < li->qtd && li->dados[i].matricula != mat)

i++;

if (i == li->qtd) // se o elemento não existe na lista

return 0;

for (k = i; k < li->qtd - 1; k++) // começando a partir da posição encontrada

li.dados[k] = li.dados[k + 1]; // posição atual recebe a próxima

li->qtd--; // atualiza quantidade da lista

return 1;

}

Consulta pela posição

int consulta\_lista\_posicao(Lista \*li, int pos, struct aluno \*al){

if (li == NULL || pos <= 0 || pos > li->qtd)

return 0;

\*al = li->dados[pos - 1];

return 1;

}

Consulta pelo atributo

int consulta\_lista\_atributo(Lista \*li, int mat, struct aluno \*al) {

if (li == NULL)

return 0;

int k, i = 0;

while (i < li->qtd && li->dados[i].matricula != mat)

i++;

if (i == li->qtd) // caso o elemento não exista na lista

return 0;

\*al = li->dados[i]; // copia o conteúdo do elemento para o ponteiro al

return 1;

}

Busca binária

- Busca binária deve receber, a princípio, início 0 e fim o último elemento da lista, retornado pela função *ultimo\_elemento()*. A função divide a lista ao meio; se a posição meio for o elemento procurado, retorna meio. Se não, compara se este elemento é menor ou maior que o procurado: se o elemento procurado for maior que o meio, o início passa a ser meio+1 e a busca se dá na metade superior;

se o elemento procurado for menor que o meio, o início se mantém e o fim passa a meio-1, buscando na metade inferior.

// marca o fim da lista, para ser utilizado na busca binária

int ultimo\_elemento(Lista\* li){

return li->qtd-1;

}

int busca\_binaria(Lista\* li, struct aluno al, int inicio, int fim){

int meio = (inicio+fim)/2;

if(inicio>fim) return -1; // se não achou o elemento

else if(al.matricula == li->dados[meio].matricula) return meio;

else if(al.matricula > li->dados[meio].matricula) return (busca\_binaria(li,al,meio+1,fim)); // busca na metade superior

else if(al.matricula < li->dados[meio].matricula)return (busca\_binaria(li,al,inicio,meio-1)); // busca na metade inferior

}