Listas não inteiras e não homogêneas

Listas não inteiras e não homogêneas

Evidentemente, o campo ***inf*** de um nó numa lista não precisa necessariamente armazenar, apenas, um valor inteiro.

Pode-se, por exemplo, representar uma lista de strings, por uma lista encadeada, tornando assim, necessários nós contendo vetores de caracteres em seus campos ***inf***.

Listas não inteiras e não homogêneas

Tais nós poderiam ser declarados como:

```
typedef struct node
{
        char inf [100];
        struct node *prox;
} NODE;
```

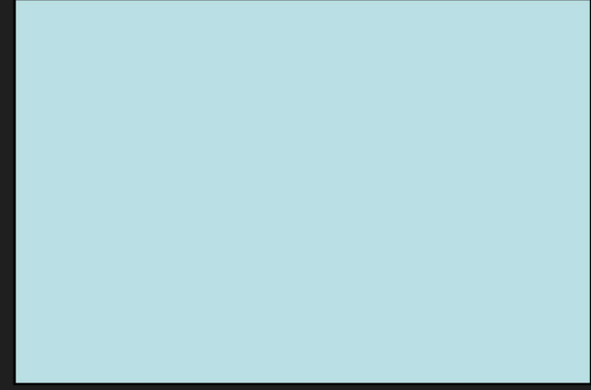
Listas não inteiras e não homogêneas

É provável que determinada aplicação exija nós com registros no campo *inf*.

Por exemplo, podemos necessitar de uma lista de registros de estudantes. Onde, cada registro, contém as seguintes informações: nome do estudante, número de identificação na faculdade, endereço, coeficiente de rendimento e área de especialização.

Os nós para tal implementação podem ser declarados como segue:

```c
typedef struct
{
    char nome [30];
    char id [9];
    char end [100];
    float cr;
    char earea [20];
} INF;
typedef struct node
{
    INF inf;
    struct node *prox;
} NODE;
```

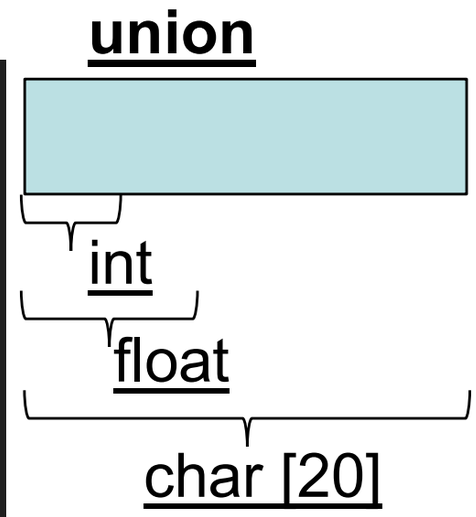


Listas não inteiras e não homogêneas

Para representar listas não-homogêneas (as que contém nós de diversos tipos), podemos utilizar uma união.

Exemplo:

```
1   #define INTGR  1
2   #define FLT 2
3   #define STRING 3
4   typedef struct node   {
5       int etype;
6       union inf {
7           int ival;
8           float fval;
9           char sval[20];
10      } element;
11      struct node *prox;
12  } NODE;
```



Listas não inteiras e não homogêneas

O exemplo anterior define um nó cujos itens podem ser inteiros, números de ponto flutuante ou strings, dependendo do valor de *etype* correspondente.

Como uma união é suficientemente grande para armazenar seu maior componente, as funções ***sizeof*** e ***malloc*** podem ser usadas para alocar armazenamento para o nó. Evidentemente, fica sob a responsabilidade do programador usar os componentes de um nó, conforme for apropriado.

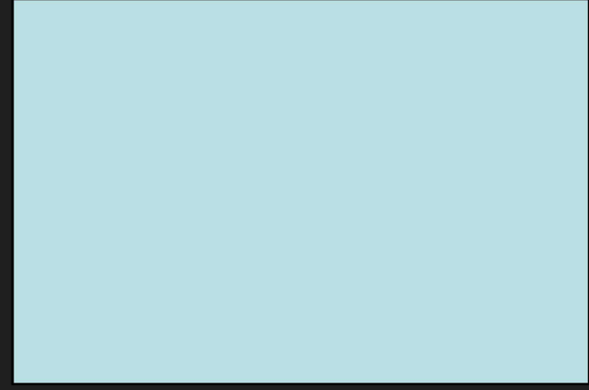
Listas não inteiras e não homogêneas

Com base no que foi visto, proponha a estrutura de dados e implemente a operação de inserção de um nó em uma lista circular duplamente encadeada não homogênea. Onde, os nós podem ter o campo com informação do tipo inteiro, ponto flutuante ou string.

A lista não deve possuir valores replicados.

Dica: Baseie-se na definição do nó vista anteriormente.

```c
typedef struct node
{
    int etype;
    union inf
    {
        int ival;
        float fval;
        char sval[20];
    } element;
    struct node *ant;
    struct node *prox;
} NODE;
typedef NODE * TAD_LISTA;
```



```c
void ins(TAD_LISTA *pl, int k, union inf v, int etype) {
    int t=tam(*pl);
    if(k<1 || k>t+1){
        printf("\nERRO!\nIndice invalido para insercao!\n");
        exit(1);
    }
    if(pertence(*pl,v,etype))
        printf("\nO valor já se encontra na lista!\n");
    else{
        TAD_LISTA novo;
        novo = (TAD_LISTA) malloc (sizeof(NODE));
        if (!novo){
            printf ("\nERRO! Memoria insuficiente!\n");
            exit (2);
        }
        novo->etype=etype;
        novo->element = v;
```

```
18          if (*pl==NULL)
19              *pl=novo->ant=novo->prox=novo;
20          else{
21              TAD_LISTA aux;
22              int p = k;
23              if (k<=t/2)
24                  for(aux=*pl;k>1;aux=aux->prox,k--);
25              else
26                  for (aux=*pl;k<=t;aux=aux->ant,k++);
27              novo->ant=aux;
28              novo->prox=aux->prox;
29              aux->prox=novo;
30              novo->prox->ant=novo;
31              if (p==t+1)
32                  *pl = novo;
33          }
34      }
35  }
```

```c
int pertence(TAD_LISTA l,union inf v,int etype){
    if (!l)
        return 0;
    else
    {
        TAD_LISTA aux=l;
        do
        {
            if (aux->etype==etype &&
            ((INTGR==etype && aux->element.ival==v.ival) ||
            (FLT==etype && aux->element.fval==v.fval) ||
            (STRING==etype && !strcmp(aux->element.sval,v.sval))))
                return 1;
            aux=aux->prox;
        }while(aux!=l);
        return 0;
    }
}
```

Listas não inteiras e não homogêneas

Como exercício de fixação, proponha a estrutura de dados e implemente as operações básicas do TAD lista circular duplamente encadeada não homogênea com nó cabeçalho. Onde, os nós podem ter o campo com informação do tipo inteiro, ponto flutuante ou string.

A lista não deve possuir valores replicados.

Disciplinas de Acesso
- Fila -

Disciplinas de acesso

Muitas vezes é útil impor, para manipulação de uma certa estrutura de dados, restrições quanto à visibilidade de seus componentes ou quanto à ordem que deve ser respeitada para se efetuarem operações.

Dois casos dos mais importantes são casos particulares de listas com disciplinas de acesso, denominados: filas e pilhas.

Filas Sequenciais

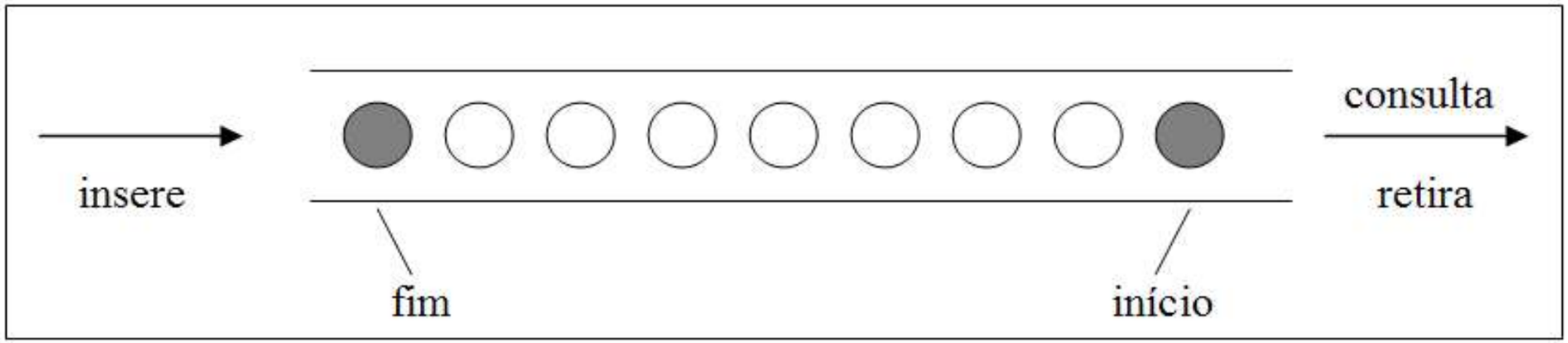
Fila - Caracterização

Uma fila é uma lista com restrições de acesso, sobre as operações de inserção, consulta e retirada.

Isto, leva ao critério FIFO (first in, first out) que indica que o primeiro item que entra é o primeiro a sair da estrutura.

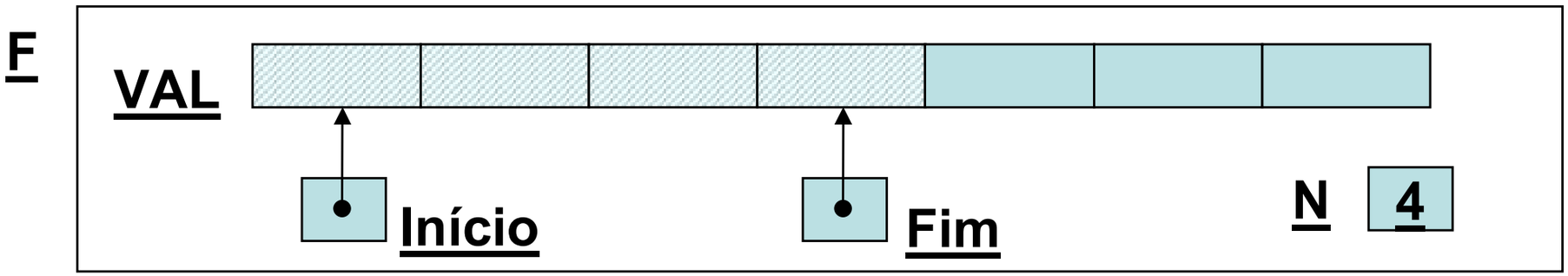
No modelo formal, não há opção de abandono da fila: somente o primeiro pode ser retirado (sair da fila).

Fila - Caracterização



Uma fila, como uma estrutura linear, pode ser armazenada em um vetor, mas necessita de dois cursores, de modo a se ter controle do ***início*** e do ***fim*** da fila. Para facilitar a implementação das operações e torná-las mais eficientes, também é utilizado um inteiro N que contém o número de elementos na fila.

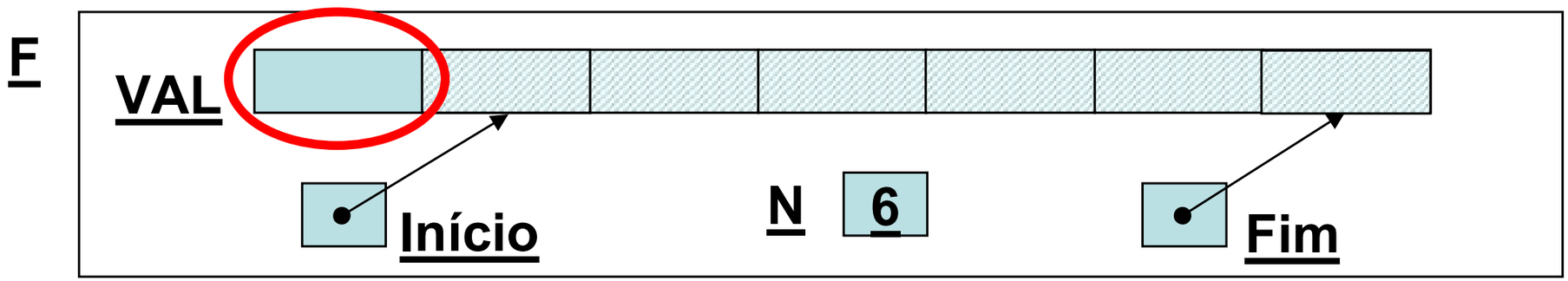
Fica - Alocação sequencial



A implementação das operações pode se dar de modo simples: a fila cresce do começo para o fim do vetor; para se inserir um elemento, incrementa-se o cursor FIM que serve como índice do novo elemento; para consulta, INICIO é o índice usado, o qual, ao ser incrementado, efetua uma retirada.

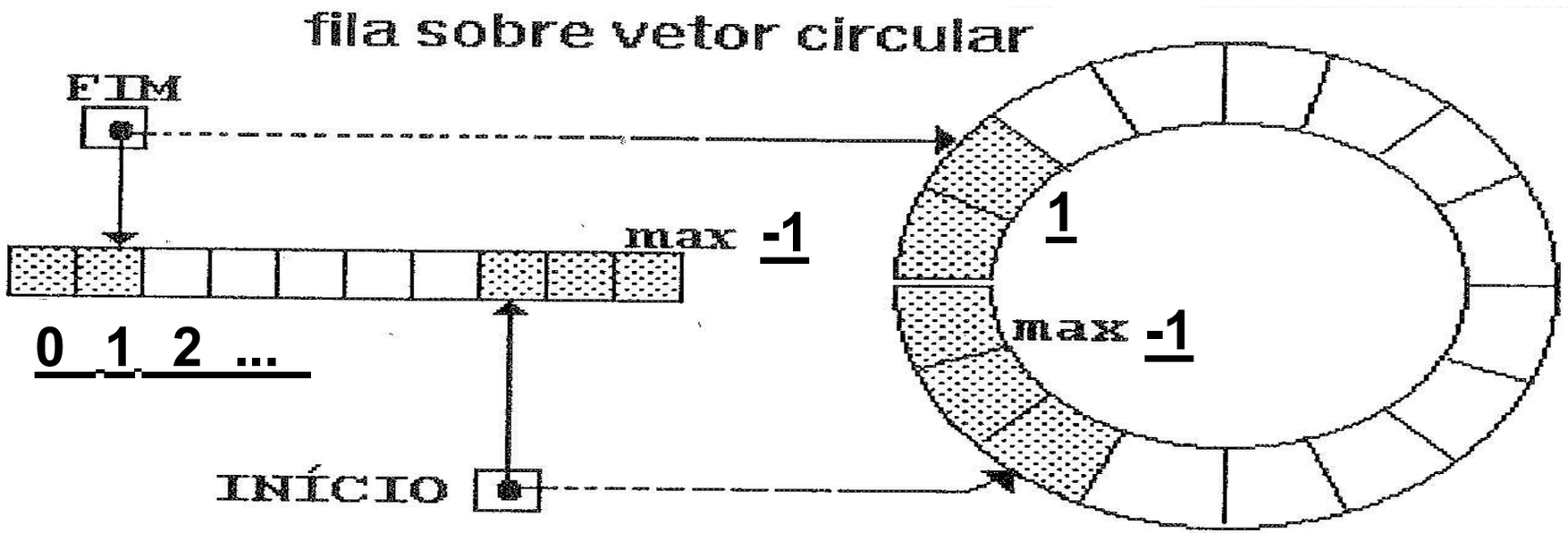
Fila - Alocação sequencial

Qual o problema com esta proposta?



Fila - Alocação sequencial

Como resolver este problema?



Fila - Alocação sequencial

O truque de implementação se resume a fazer o cursor de inserção, sempre que chegar a MAX-1, assumir 0 no próximo incremento.

Um operador que ajuda nisso é o %, pois, para todo k< MAX, k % MAX = k, mas para k = MAX, k % MAX = 0.

Agora já temos os conhecimentos necessários para definirmos e implementarmos o TAD FILA_SEQ (de valores inteiros).

```c
typedef struct
{
    int N;
    int INICIO;
    int FIM;
    int val[MAX];
}FILA_SEQ;
void cria_fila (FILA_SEQ *);
int eh_vazia (FILA_SEQ *);
int tam (FILA_SEQ *);
void ins (FILA_SEQ *, int);
int cons (FILA_SEQ *);
void ret (FILA_SEQ *);
int cons_ret (FILA_SEQ *);
```