Listas não inteiras e não homogêneas

Evidentemente, um nó numa lista não precisa representar um inteiro. Por exemplo, par se representar uma lista de strings de caracteres, por uma lista encadeada, são necessários nós contendo as strings de caracteres em seus campos ***inf***. Tais nós poderiam ser declarados como:

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

Listas não inteiras e não homogêneas

É possível que determinada aplicação exija nós com mais de um item de informação. Por exemplo, todo nó de estudantes de uma lista de estudantes pode conter as seguintes informações: nome do estudante, número de identificação da faculdade, endereço, índice de pontos de graduação e área de especialização. Os nós para tal implementação podem ser declarados assim:

```
typedef struct node
{
    char name [30];
    char id [9];
    char address [100];
    float gpindex;
    char majr [20];
    struct node *prox;
} NODE;
```

Para representar listas não-homogêneas (as que contém nós de diversos tipos), pode ser usada uma união. Por exemplo:

```
#define INTGR      1
#define FLT        2
#define STRING     3
typedef struct node
{
    int etype;
    union
    {
        int ival;
        float fval;
        char sval[20];
    } element;
    char majr [20];
    struct node *prox;
} NODE;
```

Listas não inteiras e não homogêneas

define um nó cujos itens podem ser inteiros, números de ponto flutuante o 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.

Disciplinas de acesso

Muitas vezes é útil impor-se, 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, como inserção ou retiradas, etc. Isto ajuda na modelagem de certos processos que ocorrem no mundo real.

Com o tempo e a prática, foram identificadas algumas disciplinas de acesso aplicadas a estruturas de dados, úteis em diversas aplicações. Dois casos dos mais importantes são casos particulares de listas com disciplinas de acesso, denominados: filas e pilhas.

Filas

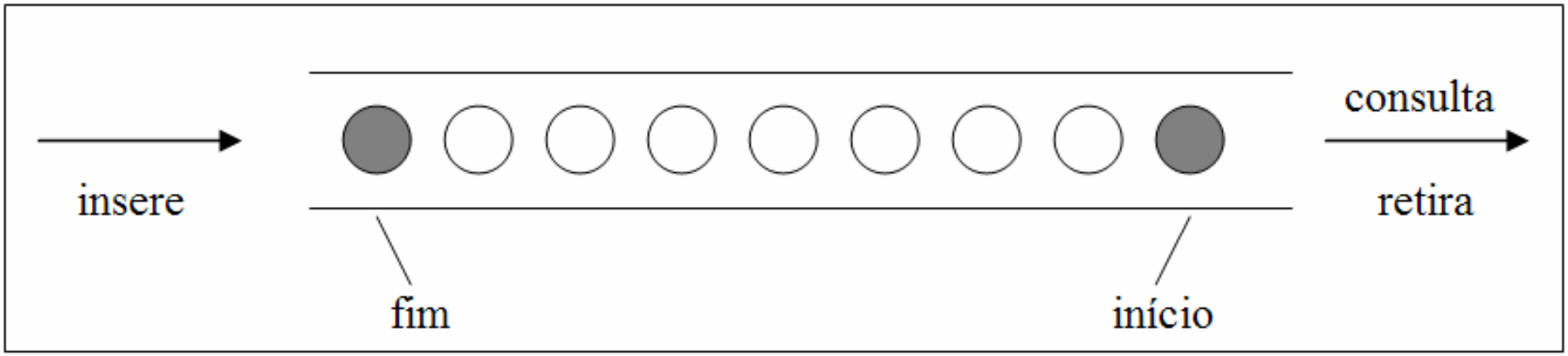
Caracterização

Uma fila é uma lista com restrições de acesso, em que as operações de inserção são realizadas sobre uma das extremidades, o fim da lista, enquanto operações de consulta e retirada são feitas na outra extremidade, o início da fila. Isto leva ao critério FIFO (first in, first out) que indica que o primeiro item que entra é o primeiro a sair da estrutura. O modelo intuitivo para isto é o de uma fila para atendimento em um guichê, na qual são atendidas as pessoas pela ordem de chegada.

UNIVASF

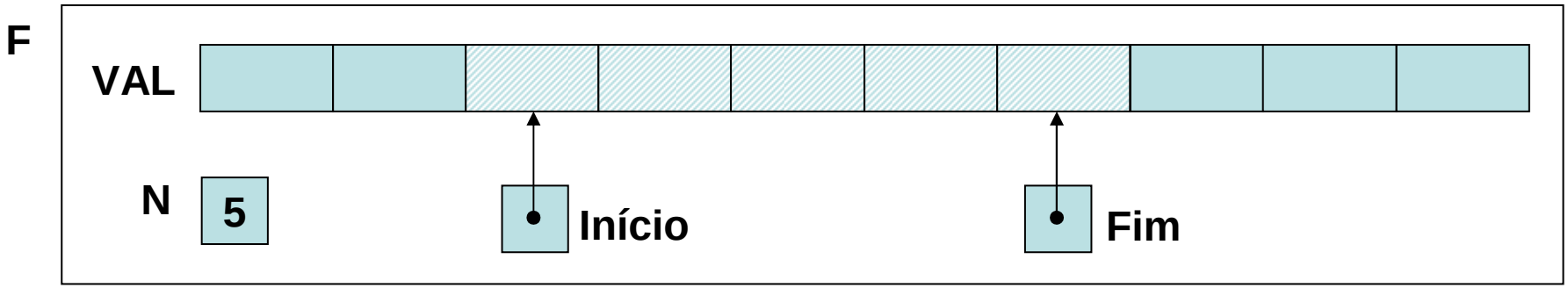
Caracterização

O atendente só tem contato com (só pode consultar) o primeiro (ou o mais antigo) da fila. Novos pretendentes ao serviço entram no fim da fila. No modelo formal, não há opção de abandono da fila: somente o primeiro pode ser retirado (sair da fila).



Alocação Seqüencial

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.



A implementação das operações pode se dar de modo simples: a fila cresce do

Alocação Seqüencial

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.

Qual o problema com esta proposta?

Ao ocorrerem inserções FIM se aproximará até alcançar o valor MAX (índice do último elemento do vetor), ao ocorrerem retiradas (INICIO terá sido incrementado), chegará à situação em que há espaço no vetor mas não se pode mais inserir na fila.

Alocação Seqüencial

Como resolver este problema?

Visando principalmente um melhor uso do espaço de armazenamento, visualizaremos o vetor como uma estrutura circular, em que o primeiro elemento sucede o último, de modo que pode-se então aproveitar, continuando com as inserções na fila, os espaços iniciais do vetor, liberados pelas retiradas.

Isso é conseguido apenas pelo controle incorporado aos algoritmos das operações, mantendo a mesma estrutura física.

UNIVASF

Alocação Seqüencial

O truque de implementação se resume a fazer o cursor de inserção, sempre que chegar a MAX, assumir 0 no próximo incremento. Um operador que ajuda nisso é o % (resto da divisão inteira), pois, para todo k< MAX, k % MAX = k, mas para k = MAX, k % MAX = 0.

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

```
typedef struct
{
    int N;          /*número de elementos*/
    int INICIO;     /*índice do primeiro elemento*/
    int FIM;        /*índice do último elemento*/
    int val[MAX]; /*vetor de elementos*/
}FILA_SEQ;
void cria_lista (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 *);
```

```
void cria_fila (FILA_SEQ *f)
{
   f->N = f->INICIO = 0;
   f->FIM = -1;
}

int eh_vazia (FILA_SEQ *f)
{
   return (f->N == 0);
}

int tam (FILA_SEQ *f)
{
   return (f->N);
}
```

```
void ins (FILA_SEQ *f, int v)
{
   if (tam(f) == MAX)
   {
      printf ("\nERRO! Estouro na fila.\n");
      exit (1);
   }
   f->FIM= ++(f->FIM) % MAX;
   f->val[f->FIM]=v;
   f->N++;
}
```

```
int cons (FILA_SEQ *f)
{
   if (eh_vazia(f))
   {
      printf ("\nERRO! Consulta na fila
   vazia.\n");
      exit (2);
   }
   else
      return (f->val[f->INICIO]);
}
```

```
void ret (FILA_SEQ *f)
{
   if (eh_vazia(f))
   {
      printf ("\nERRO! Retirada na fila
   vazia.\n");
      exit (3);
   }
   else
   {
      f->INICIO= ++f->INICIO % MAX;
      f->N--;
   }
}
```

```
int cons_ret (FILA_SEQ *f)
{
   if (eh_vazia(f))
   {
      printf ("\nERRO! Consulta e retirada na
    fila vazia.\n");
      exit (4);
   }
   else
   {
      int v=f->val[f->INICIO];
      f->INICIO= ++f->INICIO % MAX;
      f->N--;
      return (v);
   }
}
```

Alocação Seqüencial - Exercício

Implemente, no TAD FILA_SEQ, utilizando recursividade, a seguinte operação:

void gera_lista (FILA_SEQ *f, int m, int n);

a qual utilizando-se das operações do TAD FILA_SEQ produz uma fila de inteiros correspondente a [m..n].