```
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);
  }
  else
    if(pertence(*pl,v,etype))
      printf("\nO valor já se encontra na lista!\n");
    else
    {
      TAD_LISTA novo;
      novo = (NODE *) malloc (sizeof(NODE));
      if (!novo)
      {
        printf ("\nERRO! Memoria insuficiente!\n");
        exit (2);
      }
```

```
        novo->etype=etype;
        novo->element = v;
        if (*pl==NULL)
          *pl=novo->ant=novo->prox=novo;
        else
        {
          TAD_LISTA aux;
          int p = k;
          for(aux=*pl;k>1;k--,aux=aux->prox);
          novo->ant=aux;
          novo->prox=aux->prox;
          aux->prox=novo;
          novo->prox->ant=novo;
          if (p==t+1)
            *pl = novo;
        }
    }
}
```

```
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 && l->element.ival==v.ival) ||
      (FLT==etype && l->element.fval==v.fval) ||
      (STRING==etype && !strcmp(l->element.sval,v.sval))))
        return 1;
      aux=aux->prox;
    }while(aux!=l);
    return 0;
  }
}
```

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, como, por exemplo, inserções ou retiradas. 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.

UNIVASF

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 lista.

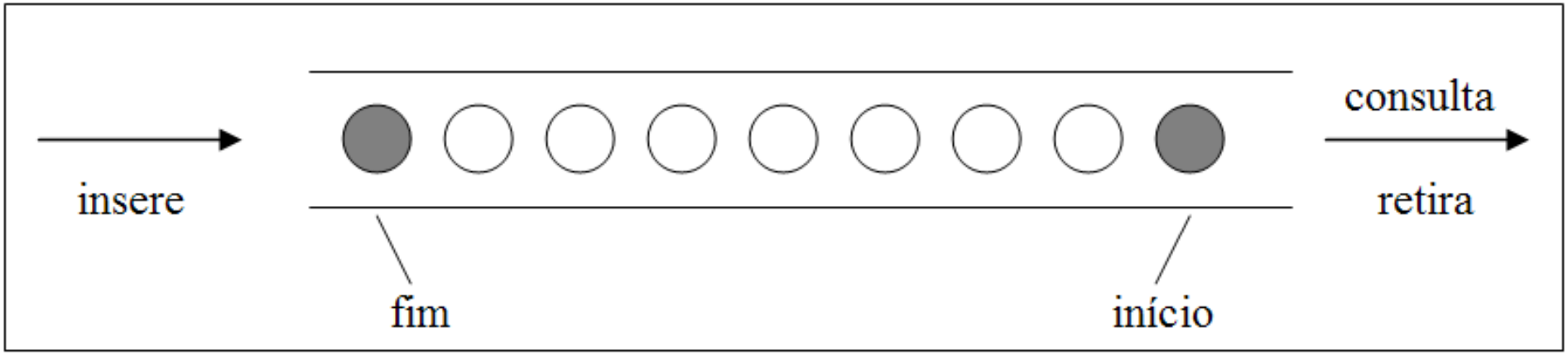
Isto, leva ao critério FIFO (first in, first out) que indica que o primeiro item que entra é o primeiro a sair da estrutura. O modelo intuitivo para isto é o de uma fila para atendimento em um guichê, na qual são atendidas as pessoas pela ordem de chegada.

UNIVASF

Caracterização

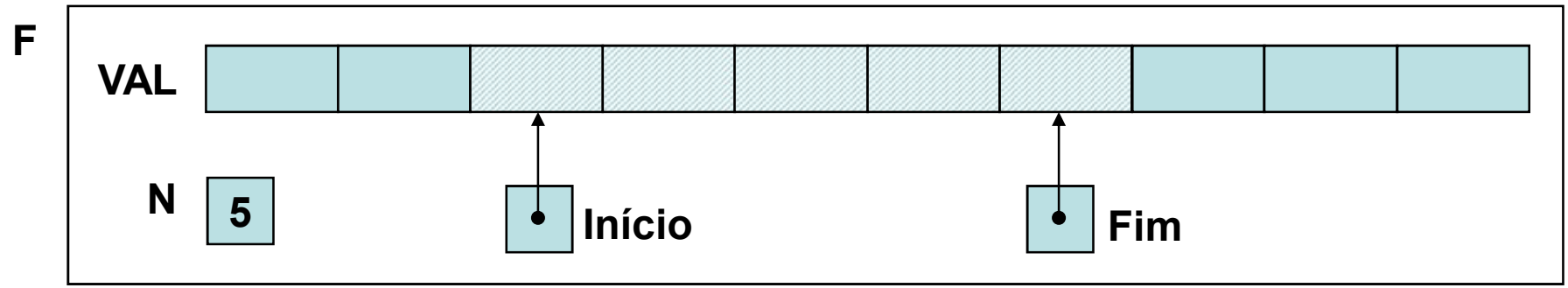
O atendente só tem contato com (só pode consultar) o primeiro (ou o mais antigo) da fila. Novos pretendentes ao serviço entram no fim da fila.

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



Alocação Sequencial

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 começo

Alocação Sequencial

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-1 (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.

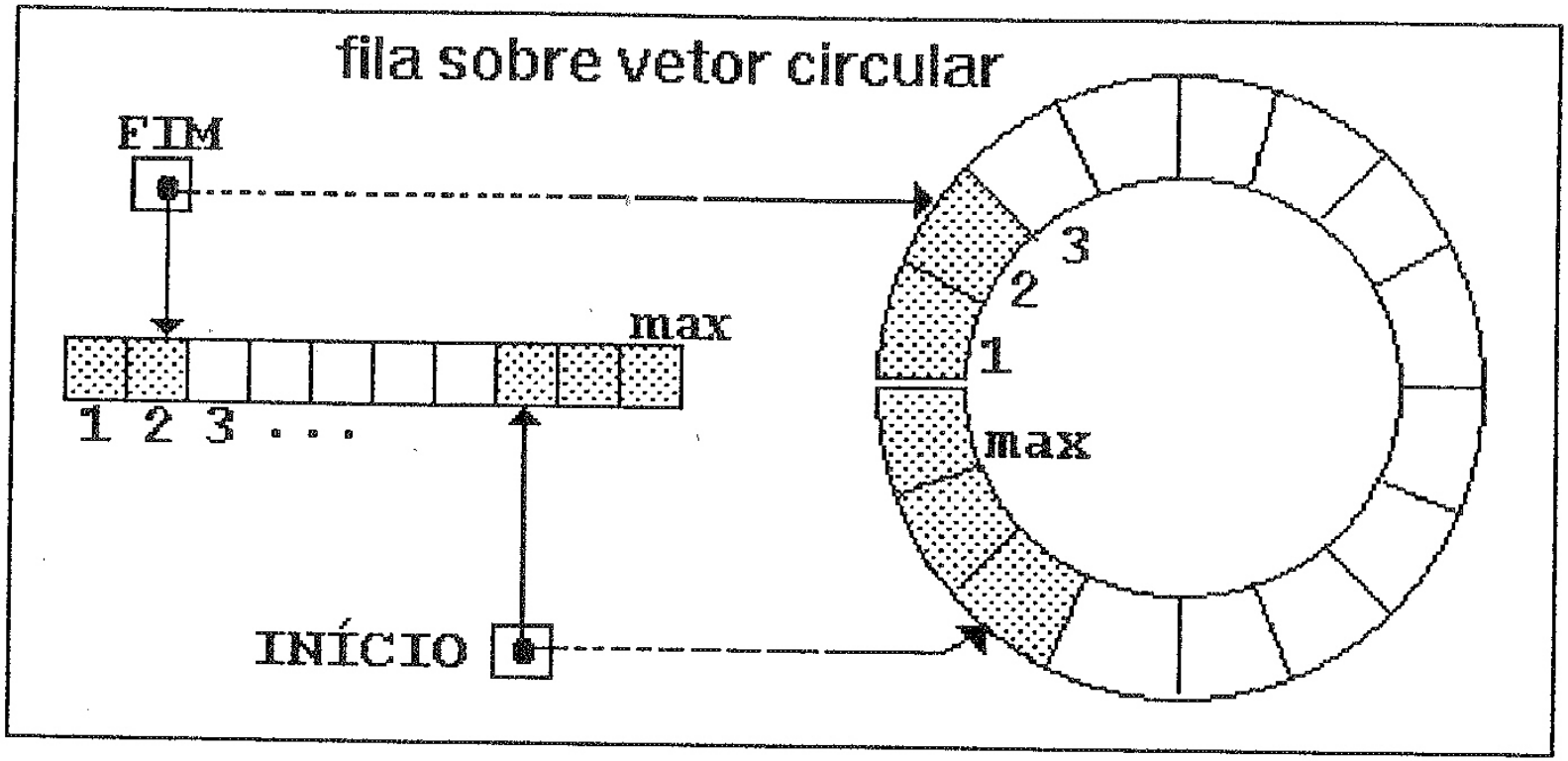
Alocação Sequencial

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.

UNIVASF

Alocação Sequencial



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

Alocação Sequencial

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 e implementarmos o TAD FILA_SEQ (de valores inteiros).

```
/*Definição do TAD FILA_SEQ*/
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_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 *);
```