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.

UNIVASF

```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.

UNIVASF

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.

UNIVASF

Filas Sequenciais

UNIVASF

Fila - Caracterização

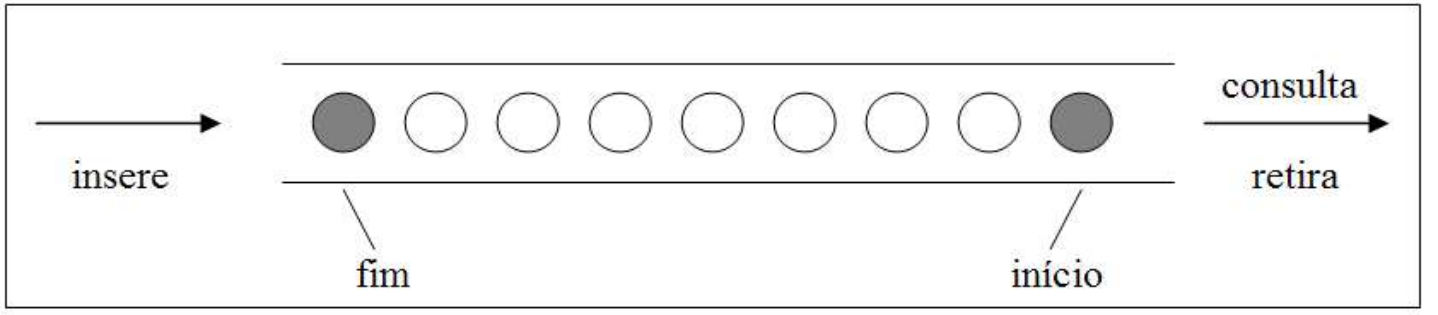
Uma fila é uma lista com restrições de acesso, sobre as operações de inserção, consulta e retirada.

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).

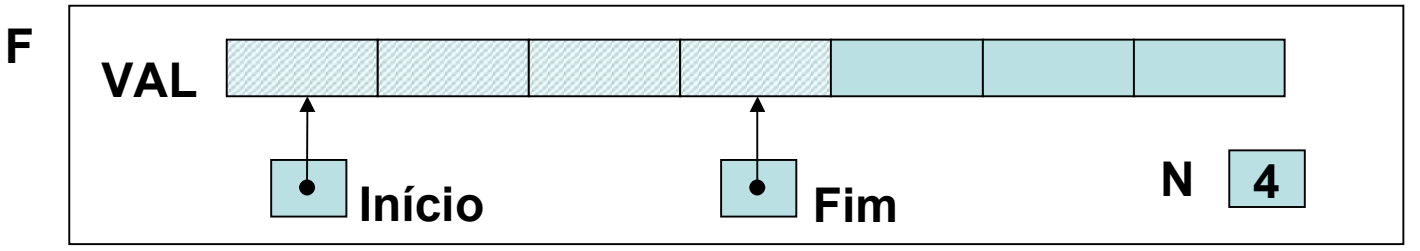
UNIVASF

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.

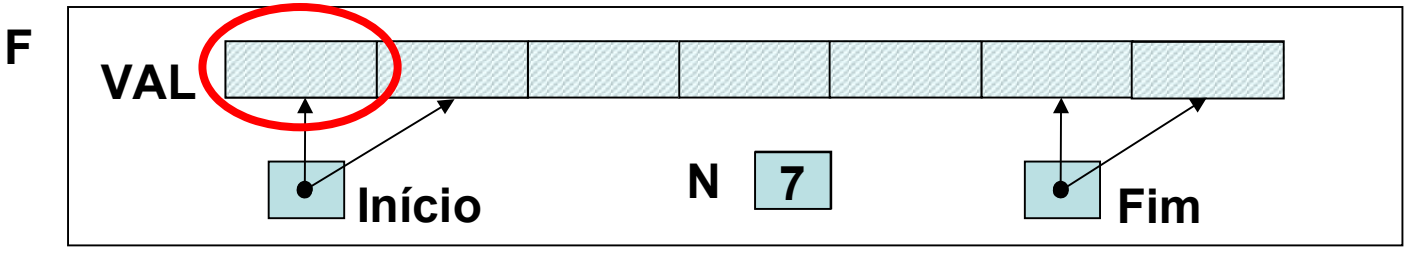
Fica - Alocação sequencial



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.

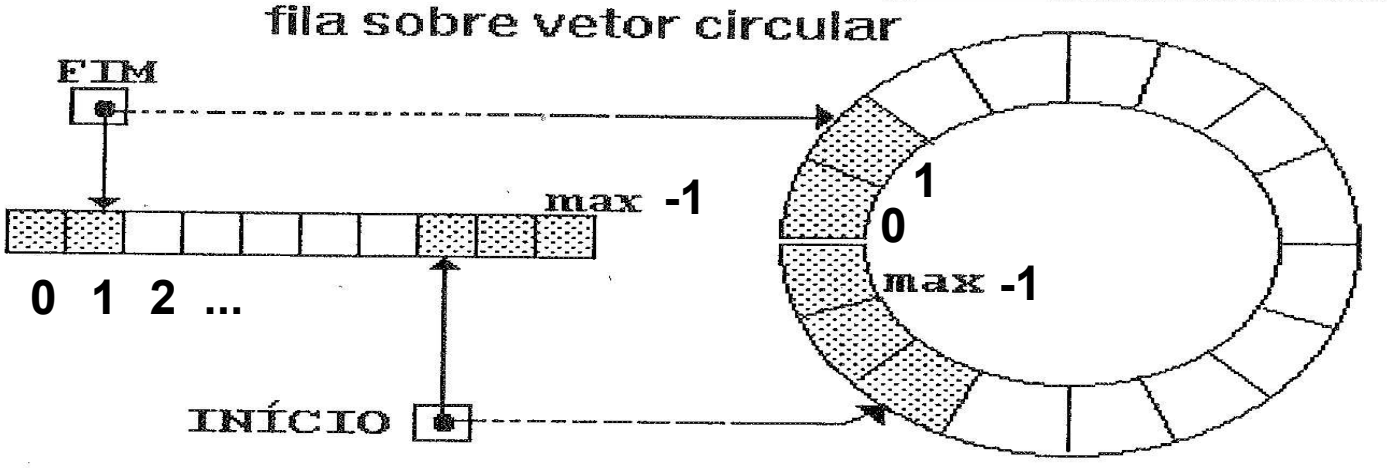
Fila - Alocação sequencial

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).

UNIVASF

```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 *);
```

Fila - Alocação Sequencial

Com base no que foi visto implemente a operação cria_fila() que compõem o TAD FILA_SEQ.

```
1   typedef struct
2   {
3       int N;
4       int INICIO;
5       int FIM;
6       int val[MAX];
7   }FILA_SEQ;
8   void cria_fila (FILA_SEQ *);
9   int eh_vazia (FILA_SEQ *);
10  int tam (FILA_SEQ *);
11  void ins (FILA_SEQ *, int);
12  int cons (FILA_SEQ *);
13  void ret (FILA_SEQ *);
14  int cons_ret (FILA_SEQ *);
```

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

Fila - Alocação Sequencial

Com base no que foi visto implemente a operação eh_vazia() que compõem o TAD FILA_SEQ.

```
1   typedef struct
2   {
3       int N;
4       int INICIO;
5       int FIM;
6       int val[MAX];
7   }FILA_SEQ;
8   void cria_fila (FILA_SEQ *);
9   int eh_vazia (FILA_SEQ *);
10  int tam (FILA_SEQ *);
11  void ins (FILA_SEQ *, int);
12  int cons (FILA_SEQ *);
13  void ret (FILA_SEQ *);
14  int cons_ret (FILA_SEQ *);
```

```c
int eh_vazia (FILA_SEQ *f)
{
    return (!f->N);
}
```

Fila - Alocação Sequencial

Com base no que foi visto implemente a operação tam() que compõem o TAD FILA_SEQ.

```
1   typedef struct
2   {
3       int N;
4       int INICIO;
5       int FIM;
6       int val[MAX];
7   }FILA_SEQ;
8   void cria_fila (FILA_SEQ *);
9   int eh_vazia (FILA_SEQ *);
10  int tam (FILA_SEQ *);
11  void ins (FILA_SEQ *, int);
12  int cons (FILA_SEQ *);
13  void ret (FILA_SEQ *);
14  int cons_ret (FILA_SEQ *);
```

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

Fila - Alocação Sequencial

Com base no que foi visto implemente a operação ins() que compõem o TAD FILA_SEQ.

```
 1  typedef struct
 2  {
 3      int N;
 4      int INICIO;
 5      int FIM;
 6      int val[MAX];
 7  }FILA_SEQ;
 8  void cria_fila (FILA_SEQ *);
 9  int eh_vazia (FILA_SEQ *);
10  int tam (FILA_SEQ *);
11  void ins (FILA_SEQ *, int);
12  int cons (FILA_SEQ *);
13  void ret (FILA_SEQ *);
14  int cons_ret (FILA_SEQ *);
```

Fila - Alocação Sequencial

Com base no que foi visto implemente a operação cons() que compõem o TAD FILA_SEQ.

```
1   typedef struct
2   {
3       int N;
4       int INICIO;
5       int FIM;
6       int val[MAX];
7   }FILA_SEQ;
8   void cria_fila (FILA_SEQ *);
9   int eh_vazia (FILA_SEQ *);
10  int tam (FILA_SEQ *);
11  void ins (FILA_SEQ *, int);
12  int cons (FILA_SEQ *);
13  void ret (FILA_SEQ *);
14  int cons_ret (FILA_SEQ *);
```

Fila - Alocação Sequencial

Com base no que foi visto implemente a operação ret() que compõem o TAD FILA_SEQ.

```
 1  typedef struct
 2  {
 3      int N;
 4      int INICIO;
 5      int FIM;
 6      int val[MAX];
 7  }FILA_SEQ;
 8  void cria_fila (FILA_SEQ *);
 9  int eh_vazia (FILA_SEQ *);
10  int tam (FILA_SEQ *);
11  void ins (FILA_SEQ *, int);
12  int cons (FILA_SEQ *);
13  void ret (FILA_SEQ *);
14  int cons_ret (FILA_SEQ *);
```

Fila - Alocação Sequencial

Com base no que foi visto implemente a operação cons_ret() que compõem o TAD FILA_SEQ.

```
1   typedef struct
2   {
3       int N;
4       int INICIO;
5       int FIM;
6       int val[MAX];
7   }FILA_SEQ;
8   void cria_fila (FILA_SEQ *);
9   int eh_vazia (FILA_SEQ *);
10  int tam (FILA_SEQ *);
11  void ins (FILA_SEQ *, int);
12  int cons (FILA_SEQ *);
13  void ret (FILA_SEQ *);
14  int cons_ret (FILA_SEQ *);
```