

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.

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

```
1 void ins(TAD_LISTA *pl, int k, union inf v, int etype) {
2     int t=tam(*pl);
3     if(k<1 || k>t+1){
4         printf("\nERRO!\nIndice invalido para insercao!\n");
5         exit(1);
6     }
7     if(pertence(*pl,v,etype))
8         printf("\n0 valor já se encontra na lista!\n");
9     else{
10        TAD_LISTA novo;
11        novo = (TAD_LISTA) malloc (sizeof(NODE));
12        if (!novo){
13            printf ("\nERRO! Memoria insuficiente!\n");
14            exit (2);
15        }
16        novo->etype=etype;
17        novo->element = v;
```

```
18     if (*p1==NULL)
19         *p1=novo->ant=novo->prox=novo;
20     else{
21         TAD_LISTA aux;
22         int p = k;
23         if (k<=t/2)
24             for(aux=*p1;k>1;aux=aux->prox,k--);
25         else
26             for (aux=*p1;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             *p1 = novo;
33     }
34 }
35 }
```

```
1  int pertence(TAD_LISTA l,union inf v,int etype){
2      if (!l)
3          return 0;
4      else
5      {
6          TAD_LISTA aux=l;
7          do
8          {
9              if (aux->etype==etype &&
10                 ((INTGR==etype && aux->element.ival==v.ival) ||
11                  (FLT==etype && aux->element.fval==v.fval) ||
12                  (STRING==etype && !strcmp(aux->element.sval,v.sval))))
13                  return 1;
14                  aux=aux->prox;
15            }while(aux!=l);
16            return 0;
17        }
18    }
```

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

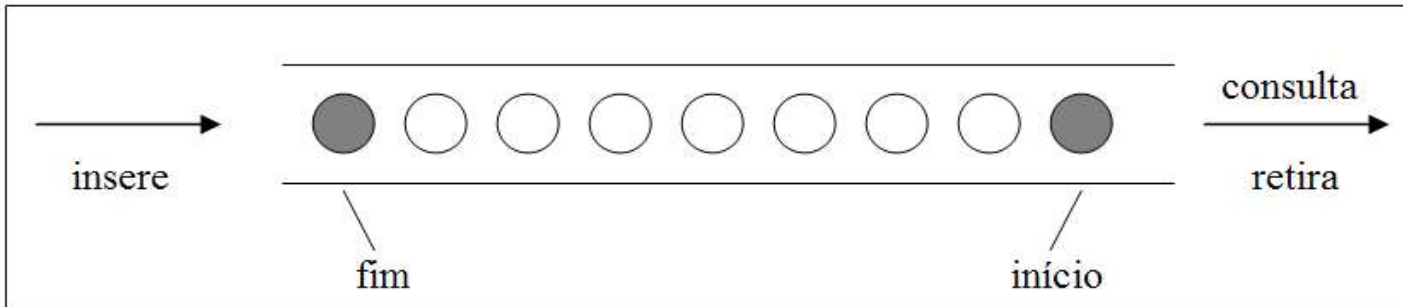
Fila - Caracterização

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).

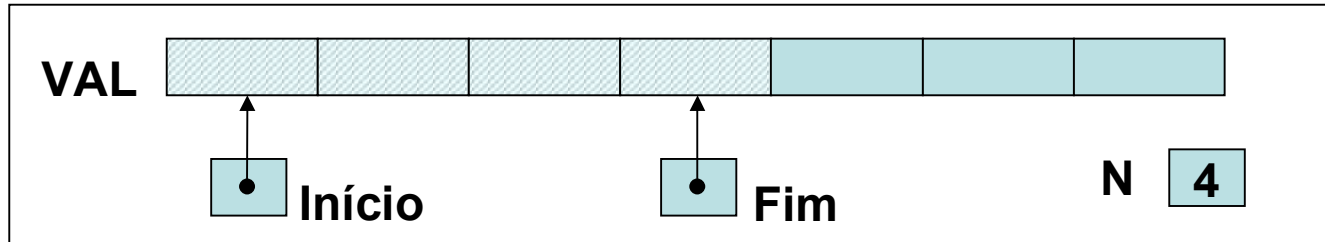
Fila - Caracterização



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

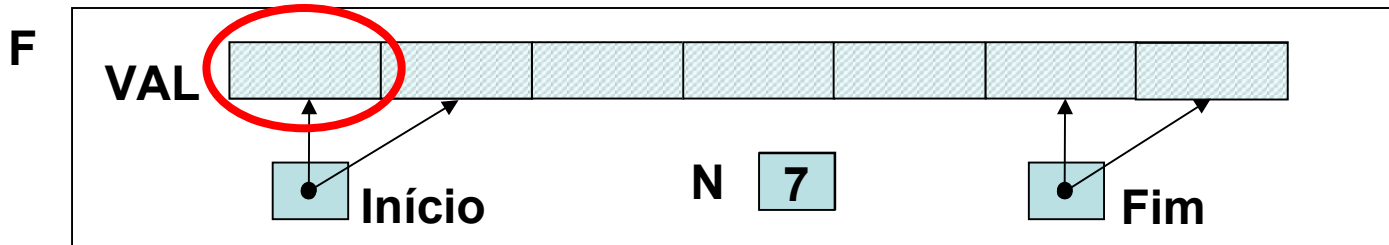
F



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.

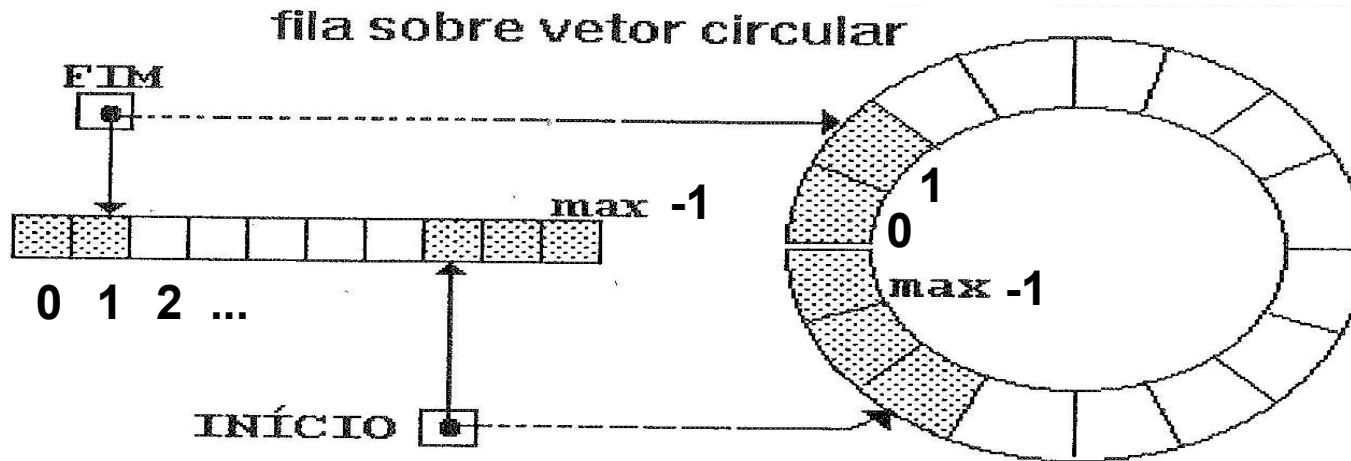
Fila - Alocação sequencial

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).

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

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 *);
```

```
1 void cria_filha (FILHA_SEQ *f)
2 {
3     f->N = f->INICIO = 0;
4     f->FIM = -1;
5 }
```

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 *);
```

```
1  int eh_vazia (FILASEQ *f)
2  {
3      return (!f->N);
4  }
```

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 *);
```

```
1  int tam (FILASEQ *f)
2  {
3      return (f->N);
4  }
```

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 *);
```

```
1 void ins (FILASEQ *f, int v)
2 {
3     if (f->N == MAX)
4     {
5         printf ("\nERRO! Estouro na fila.\n");
6         exit (1);
7     }
8     f->FIM= ((f->FIM)+1) % MAX;
9     f->val[f->FIM]=v;
10    f->N++;
11 }
```

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 *);
```