

# 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 *);
```

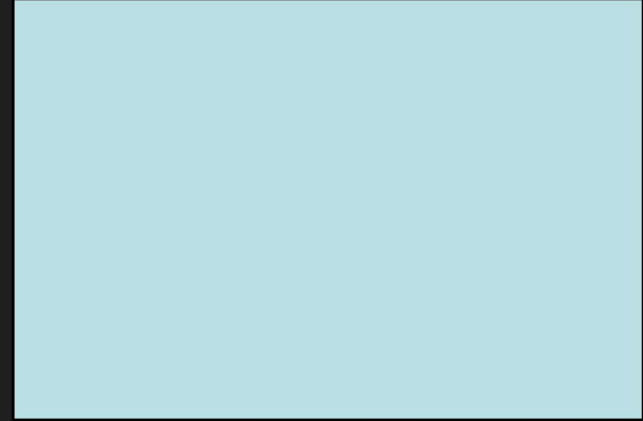
```
1 void cria_filha (FILHA_SEQ *f)
2 {
3     f->N = f->INICIO = 0;
4     f->FIM = -1;
5 }
```

# 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  }
```

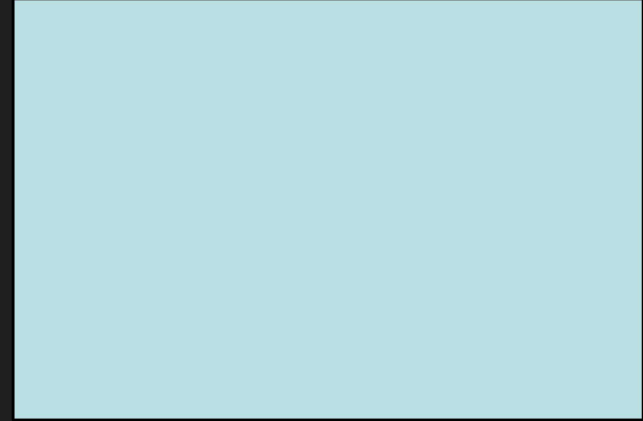


# Fila - Alocação Sequencial

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 *);
```

```
1  int cons (FILASEQ *f)
2  {
3      if (eh_vazia(f))
4      {
5          printf ("\nERRO! Consulta na fila vazia.\n");
6          exit (2);
7      }
8      else
9          return (f->val[f->INICIO]);
10 }
```

# 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 *);
```

```
1 void ret (FILASEQ *f)
2 {
3     if (eh_vazia(f))
4     {
5         printf ("\nERRO! Retirada na fila vazia.\n");
6         exit (3);
7     }
8     else
9     {
10        f->INICIO= (f->INICIO+1) % MAX;
11        f->N--;
12    }
13 }
```

# 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 *);
```

```
1  int cons_ret (FILASEQ *f)
2  {
3      if (eh_vazia(f))
4      {
5          printf ("\nERRO! Consulta e retirada na fila vazia.\n");
6          exit (4);
7      }
8      else
9      {
10         int v=f->val[f->INICIO];
11         f->INICIO= (f->INICIO+1) % MAX;
12         f->N--;
13         return (v);
14     }
15 }
```

## Alocação Sequencial - Exercício

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

```
void gera_fila (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]`.

```
1 void gera_filha (FILHA_SEQ *f, int m, int n)
2 {
3     if (m>n)
4     {
5         printf ("\nERRO! Intervalo invalido.\n");
6         exit (5);
7     } else
8         if (m==n){
9             cria_filha (f);
10            ins (f, m);
11        } else {
12            gera_filha (f, m, n-1);
13            ins (f, n);
14        }
15 }
```



# Filas Encadeadas

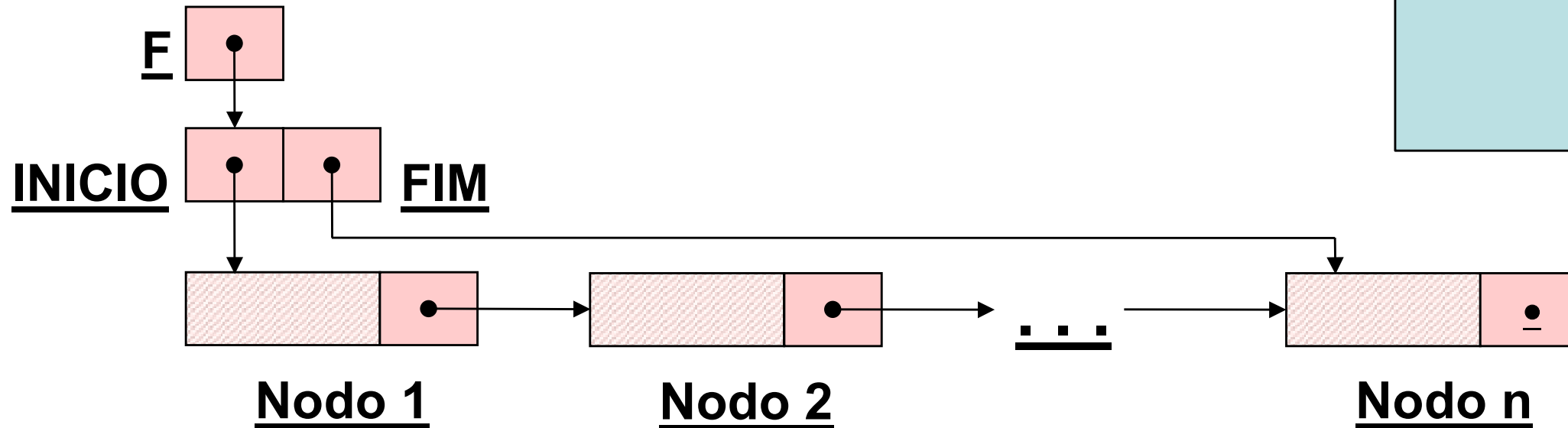
## Fila - Alocação Encadeada

Como debatemos, a alocação sequencial apresenta algumas desvantagens. Em virtude disso, podemos nos utilizar de uma lista encadeada dinâmica para armazenarmos uma fila.

Como operaremos em ambas as extremidades da lista, devemos facilitar o acesso ao último nó.

Uma estratégia, muito aplicada, é a utilização de uma representação baseada em um descritor contendo duas referências, ao primeiro e ao último nó.

# Fila - Alocação Encadeada



Desta forma, definiremos e implementaremos, agora, o TAD `FILA_ENC` (de valores inteiros).

**Obs.:** Se para a aplicação se fizer relevante o descritor pode armazenar, também, o número de elementos na fila.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef struct
7  {
8      NODO *INICIO;
9      NODO *FIM;
10 }DESCRITOR;
11 typedef DESCRITOR * FILA_ENC;
12 void cria_fila (FILA_ENC *);
13 int eh_vazia (FILA_ENC);
14 void ins (FILA_ENC, int);
15 int cons (FILA_ENC);
16 void ret (FILA_ENC);
17 int cons_ret (FILA_ENC);
18 void destruir (FILA_ENC);
```

# Fila - Alocação Encadeada

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

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef struct
7  {
8      NODO *INICIO;
9      NODO *FIM;
10 }DESCRITOR;
11 typedef DESCRITOR * FILA_ENC;
12 void cria_fila (FILA_ENC *);
13 int eh_vazia (FILA_ENC);
14 void ins (FILA_ENC, int);
15 int cons (FILA_ENC);
16 void ret (FILA_ENC);
17 int cons_ret (FILA_ENC);
18 void destruir (FILA_ENC);
```

```
1 void cria_filha (FILHA_ENC *pf)
2 {
3     *pf=(DESCRITOR *)malloc(sizeof(DESCRITOR));
4     if (!*pf)
5     {
6         printf ("\nERRO! Memoria insuficiente!\n");
7         exit (1);
8     }
9     (*pf)->INICIO=(*pf)->FIM=NULL;
10 }
```

# Fila - Alocação Encadeada

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

`FILA_ENC`.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef struct
7  {
8      NODO *INICIO;
9      NODO *FIM;
10 }DESCRITOR;
11 typedef DESCRITOR * FILA_ENC;
12 void cria_fila (FILA_ENC *);
13 int eh_vazia (FILA_ENC);
14 void ins (FILA_ENC, int);
15 int cons (FILA_ENC);
16 void ret (FILA_ENC);
17 int cons_ret (FILA_ENC);
18 void destruir (FILA_ENC);
```

```
1  int eh_vazia (FILA_ENC f)
2  {
3      return (f->INICIO == NULL);
4  }
```



# Fila - Alocação Encadeada

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

FILA\_ENC.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef struct
7  {
8      NODO *INICIO;
9      NODO *FIM;
10 }DESCRITOR;
11 typedef DESCRITOR * FILA_ENC;
12 void cria_fila (FILA_ENC *);
13 int eh_vazia (FILA_ENC);
14 void ins (FILA_ENC, int);
15 int cons (FILA_ENC);
16 void ret (FILA_ENC);
17 int cons_ret (FILA_ENC);
18 void destruir (FILA_ENC);
```

# Fila - Alocação Encadeada

## Dicas:

Tem espaço na memória para armazenar mais um elemento?

Todas as situações de inserção são tratadas da mesma forma?



# Fila - Alocação Encadeada

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

FILA\_ENC.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef struct
7  {
8      NODO *INICIO;
9      NODO *FIM;
10 }DESCRITOR;
11 typedef DESCRITOR * FILA_ENC;
12 void cria_fila (FILA_ENC *);
13 int eh_vazia (FILA_ENC);
14 void ins (FILA_ENC, int);
15 int cons (FILA_ENC);
16 void ret (FILA_ENC);
17 int cons_ret (FILA_ENC);
18 void destruir (FILA_ENC);
```

```
1  int cons (FILA_ENC f)
2  {
3      if (eh_vazia(f))
4      {
5          printf ("\nERRO! Consulta em fila vazia!\n");
6          exit (2);
7      }
8      else
9          return (f->INICIO->inf);
10 }
```

# Fila - Alocação Encadeada

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

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef struct
7  {
8      NODO *INICIO;
9      NODO *FIM;
10 }DESCRITOR;
11 typedef DESCRITOR * FILA_ENC;
12 void cria_fila (FILA_ENC *);
13 int eh_vazia (FILA_ENC);
14 void ins (FILA_ENC, int);
15 int cons (FILA_ENC);
16 void ret (FILA_ENC);
17 int cons_ret (FILA_ENC);
18 void destruir (FILA_ENC);
```

# Fila - Alocação Encadeada

## Dicas:

A FILA não é vazia?

Todas as situações de remoção são tratadas da mesma forma?

```
1 void ret (FILA_ENC f)
2 {
3     if (eh_vazia(f))
4     {
5         printf ("\nERRO! Retirada em fila vazia!\n");
6         exit (3);
7     }
8     else {
9         NODO *aux=f->INICIO;
10        f->INICIO=f->INICIO->next;
11        if (!f->INICIO)
12            f->FIM=NULL;
13        free (aux);
14    }
15 }
```



# Fila - Alocação Encadeada

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

`FILA_ENC`.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef struct
7  {
8      NODO *INICIO;
9      NODO *FIM;
10 }DESCRITOR;
11 typedef DESCRITOR * FILA_ENC;
12 void cria_fila (FILA_ENC *);
13 int eh_vazia (FILA_ENC);
14 void ins (FILA_ENC, int);
15 int cons (FILA_ENC);
16 void ret (FILA_ENC);
17 int cons_ret (FILA_ENC);
18 void destruir (FILA_ENC);
```

# Fila - Alocação Encadeada

Com base no que foi visto implemente a operação destruir() que compõem o TAD

FILA\_ENC.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef struct
7  {
8      NODO *INICIO;
9      NODO *FIM;
10 }DESCRITOR;
11 typedef DESCRITOR * FILA_ENC;
12 void cria_fila (FILA_ENC *);
13 int eh_vazia (FILA_ENC);
14 void ins (FILA_ENC, int);
15 int cons (FILA_ENC);
16 void ret (FILA_ENC);
17 int cons_ret (FILA_ENC);
18 void destruir (FILA_ENC);
```