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:

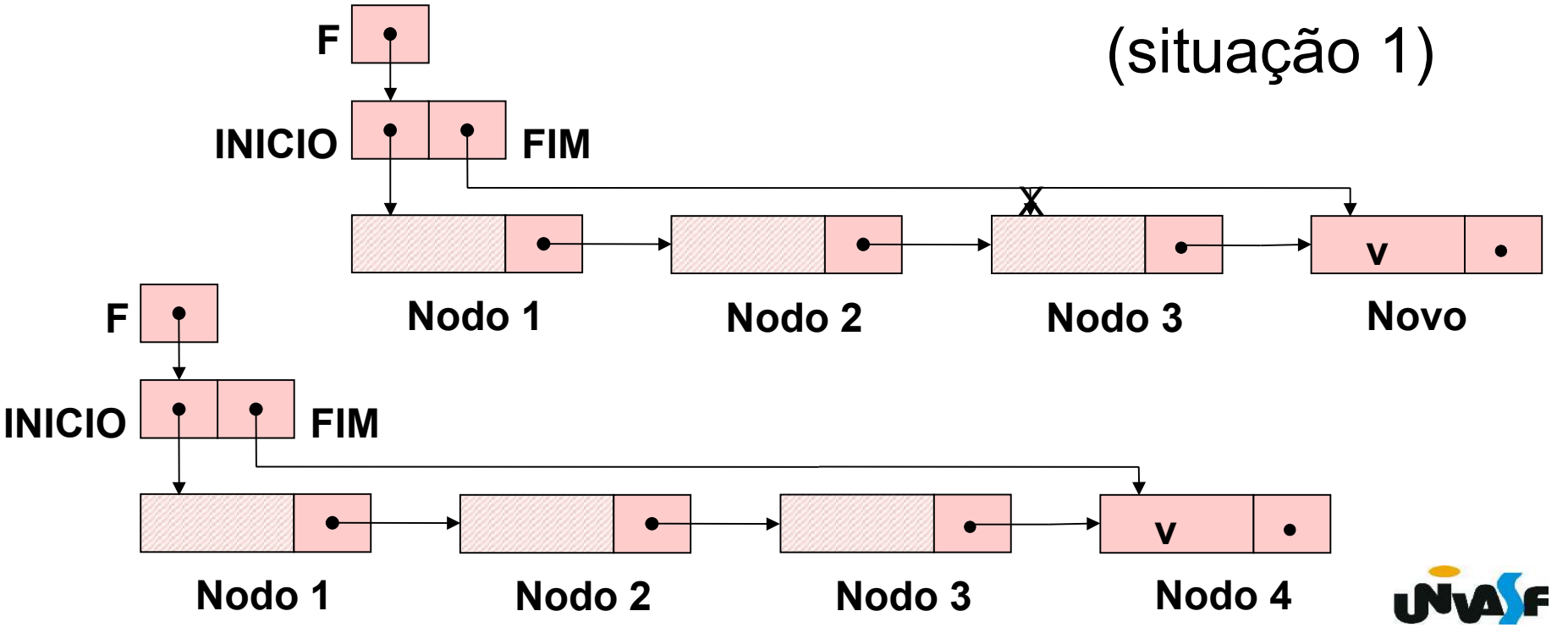
Tem espaço na memória para armazenar mais um elemento?

Todas as situações de inserção são tratadas da mesma forma?

UNIVASF

Fila - Alocação Encadeada

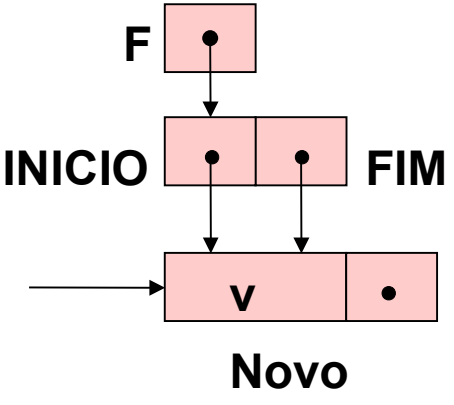
Esquema do processo de inserção de um elemento na fila encadeada.

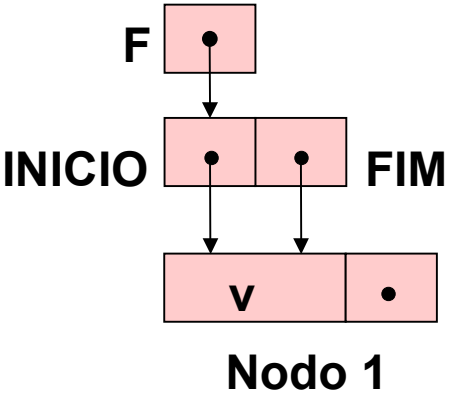


Fila - Alocação Encadeada

Esquema do processo de inserção de um elemento na fila encadeada.

(situação 2)





```c
void ins (FILA_ENC f, int v)
{
    NODO *novo;
    novo = (NODO *) malloc (sizeof(NODO));
     if (!novo) {
         printf ("\nERRO! Memoria insuficiente!\n");
         exit (1);
       }
    novo->inf = v;
    novo->next = NULL;
    if (eh_vazia(f))
       f->INICIO=novo;
    else
       f->FIM->next=novo;
    f->FIM=novo;
}
```

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);
```

```c
int cons (FILA_ENC f)
{
    if (eh_vazia(f))
    {
        printf ("\nERRO! Consulta em fila vazia!\n");
        exit (2);
    }
    else
        return (f->INICIO->inf);
}
```

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:

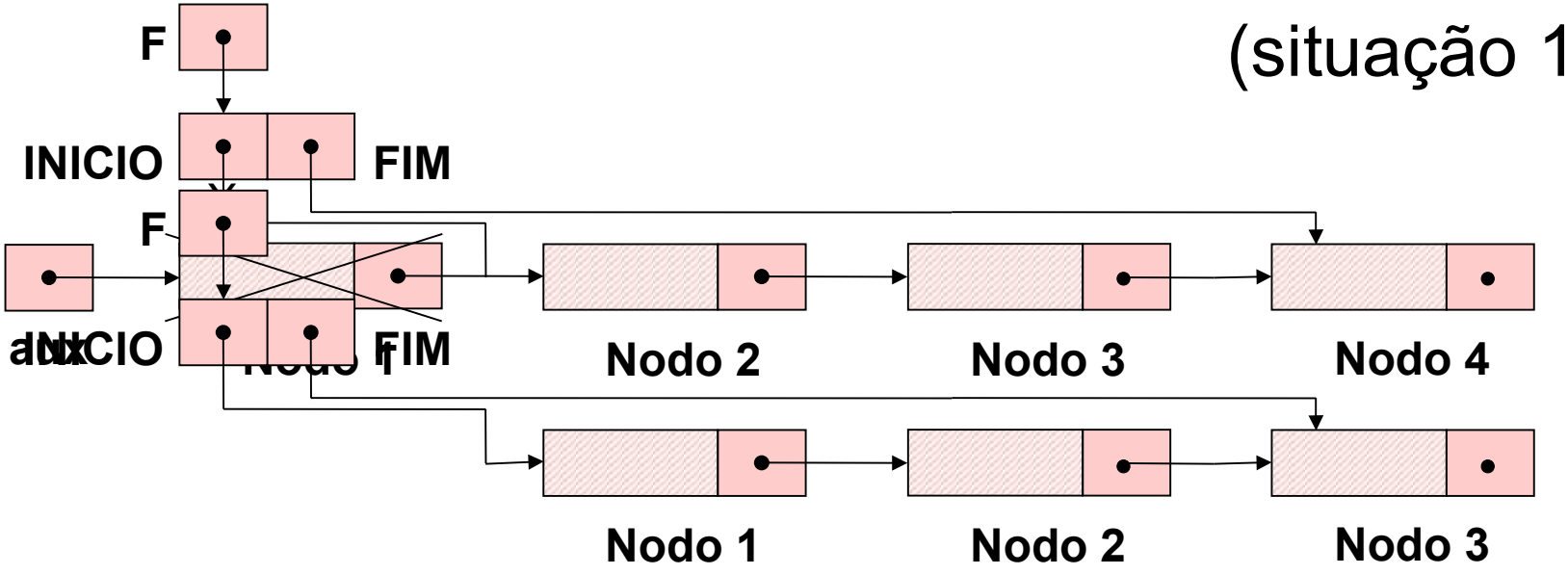
A FILA não é vazia?

Todas as situações de remoção são tratadas da mesma forma?

UNIVASF

Fila - Alocação Encadeada

Esquema do processo de retirada de um elemento na fila encadeada.

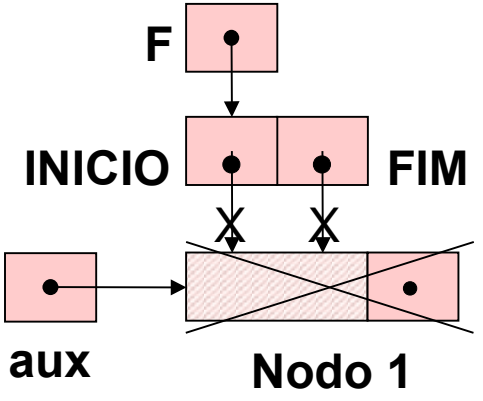


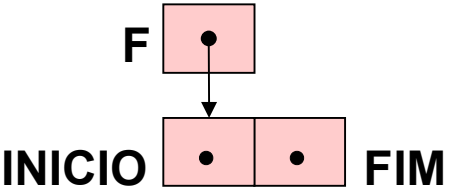
(situação 1)

Fila - Alocação Encadeada

Esquema do processo de retirada de um elemento na fila encadeada.

(situação 2)





```c
void ret (FILA_ENC f)
{
    if (eh_vazia(f))
    {
        printf ("\nERRO! Retirada em fila vazia!\n");
        exit (3);
    }
    else {
        NODO *aux=f->INICIO;
        f->INICIO=f->INICIO->next;
        if (!f->INICIO)
            f->FIM=NULL;
        free (aux);
      }
}
```

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);
```

```c
int cons_ret (FILA_ENC f)
{
    if (eh_vazia(f))
    {
        printf ("\nERRO! Consulta e retirada em fila vazia!\n");
        exit (4);
    }
    else {
       int v=f->INICIO->inf;
       NODO *aux=f->INICIO;
       f->INICIO=f->INICIO->next;
       if (!f->INICIO)
          f->FIM=NULL;
       free (aux);
       return (v);
    }
}
```

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);
```

```c
void destruir (FILA_ENC f)
{
    NODO *aux;
    while (f->INICIO)
    {
        aux=f->INICIO;
        f->INICIO=f->INICIO->next;
        free(aux);
    }
    free(f);
}
```

Fila - Alocação Encadeada

Como exercício, baseando-se no TAD anterior, defina e implemente o TAD FILA_ENC (de valores inteiros). Onde o descritor deve armazenar o número de elementos na fila e teremos a operação que determinará o tamanho da fila.

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

```
1  void cria_fila (FILA_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;

11 }
```

Fila - Alocação Encadeada

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

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

```c
void ins (FILA_ENC f, int v)
{
   NODO *novo;
   novo = (NODO *) malloc (sizeof(NODO));
    if (!novo)  {
        printf ("\nERRO! Memoria insuficiente!\n");
        exit (1);
      }
   novo->inf = v;
   novo->next = NULL;
   if (eh_vazia(f))
      f->INICIO=novo;
   else
      f->FIM->next=novo;
   f->FIM=novo;
   f->ne++;
}
```

Fila - Alocação Encadeada

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

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

```c
void ret (FILA_ENC f)
{
    if (!f->INICIO)
    {
        printf ("\nERRO! Retirada em fila vazia!\n");
        exit (3);
    }
    else {
        NODO *aux=f->INICIO;
        f->INICIO=f->INICIO->next;
        if (!f->INICIO)
            f->FIM=NULL;
        free (aux);
        f->ne--;
      }
}
```

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

```c
int cons_ret (FILA_ENC f)
{
   if (!f->INICIO)
   {
       printf ("\nERRO! Consulta e retirada em fila vazia!\n");
       exit (4);
    }
   else {
      int v=f->INICIO->inf;
      NODO *aux=f->INICIO;
      f->INICIO=f->INICIO->next;
      if (!f->INICIO)
         f->FIM=NULL;
      free (aux);
      f->ne--;
      return (v);
    }
}
```

Fila - Alocação Encadeada

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

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

```c
int tam (FILA_ENC f)
{
    return (f->ne);
}
```

Filas

Como vimos, uma fila nada mais é do que uma lista com uma disciplina de acesso.

Logo, podemos nos utilizar de todos os conceitos vistos em listas para implementarmos filas.

Por exemplo, podemos utilizar uma lista circular para armazenar uma fila.

Como exercício de fixação, implemente um TAD FILA, armazenado a mesma em uma lista circular.

UNIVASF

Pilhas
Alocação Sequencial

UNIVASF

Pilha - Caracterização

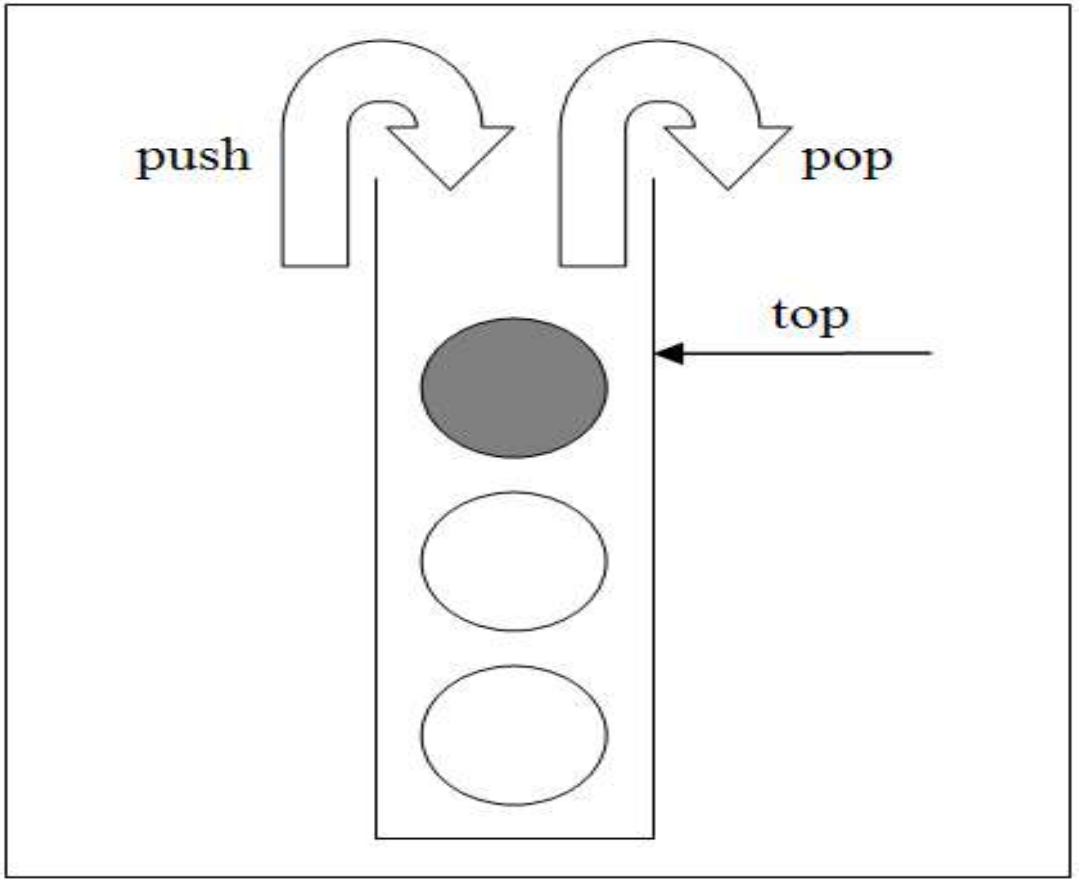
Uma pilha é uma lista com restrições de acesso, onde todas as operações só podem ser aplicadas sobre uma das extremidades da lista, denominada topo da pilha.

Com isso estabelece-se o critério LIFO (Last In, First Out), que indica que o último item que entra é o primeiro a sair.

O modelo intuitivo para isto é, por exemplo, uma pilha de livros onde só se pode visualizar o último empilhado e este é o único que pode ser retirado; Qualquer novo empilhamento se fará sobre o último da pilha.

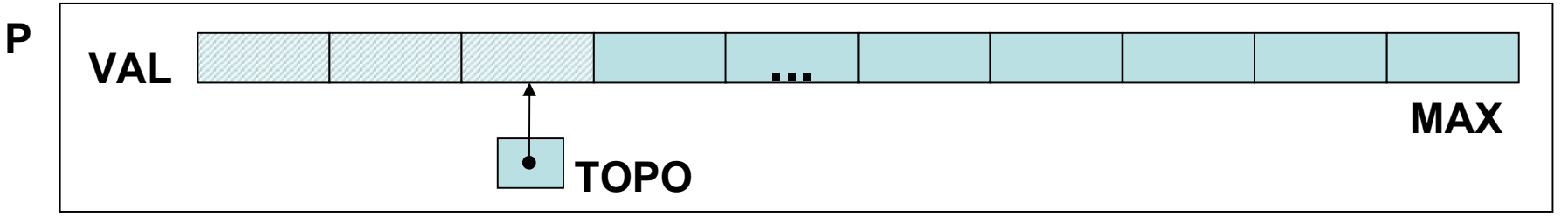
UNIVASF

Pilha - Caracterização



Pilha – Alocação Sequencial

Uma forma de se implementar uma pilha é armazená-la num vetor VAL de MAX elementos associado com um cursor inteiro TOPO que indica onde está o topo da pilha, como se vê abaixo:



De posse destas informações definiremos e implementaremos agora o TAD PILHA_SEQ de valores inteiros.

```c
typedef struct
{
    int TOPO;
    int VAL[MAX];
}PILHA_SEQ;
void cria_pilha (PILHA_SEQ *);
int eh_vazia (PILHA_SEQ *);
void push (PILHA_SEQ *, int);
int top (PILHA_SEQ *);
void pop (PILHA_SEQ *);
int top_pop (PILHA_SEQ *);
```

Pilha - Alocação Sequencial

Com base no que foi visto implemente a operação cria_pilha() que compõem o TAD PILHA_SEQ.

```
1  typedef struct
2  {
3      int TOPO;
4      int VAL[MAX];
5  }PILHA_SEQ;
6  void cria_pilha (PILHA_SEQ *);
7  int eh_vazia (PILHA_SEQ *);
8  void push (PILHA_SEQ *, int);
9  int top (PILHA_SEQ *);
10 void pop (PILHA_SEQ *);
11 int top_pop (PILHA_SEQ *);
```

UNIVASF

```c
void cria_pilha (PILHA_SEQ *p)
{
    p->TOPO = -1;
}
```

Pilha - Alocação Sequencial

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

```
1  typedef struct
2  {
3      int TOPO;
4      int VAL[MAX];
5  }PILHA_SEQ;
6  void cria_pilha (PILHA_SEQ *);
7  int eh_vazia (PILHA_SEQ *);
8  void push (PILHA_SEQ *, int);
9  int top (PILHA_SEQ *);
10 void pop (PILHA_SEQ *);
11 int top_pop (PILHA_SEQ *);
```

UNIVASF

```c
int eh_vazia (PILHA_SEQ *p)
{
    return (p->TOPO == -1);
}
```

Pilha - Alocação Sequencial

Com base no que foi visto implemente a operação push() que compõem o TAD PILHA_SEQ.

```
1  typedef struct
2  {
3      int TOPO;
4      int VAL[MAX];
5  }PILHA_SEQ;
6  void cria_pilha (PILHA_SEQ *);
7  int eh_vazia (PILHA_SEQ *);
8  void push (PILHA_SEQ *, int);
9  int top (PILHA_SEQ *);
10 void pop (PILHA_SEQ *);
11 int top_pop (PILHA_SEQ *);
```

```c
void push (PILHA_SEQ *p, int v)
{
    if (p->TOPO == MAX-1)
    {
        printf ("\nERRO! Estouro na pilha.\n");
        exit (1);
    }
    p->VAL[++(p->TOPO)]=v;
}
```

Pilha - Alocação Sequencial

Com base no que foi visto implemente a operação top() que compõem o TAD PILHA_SEQ.

```
1  typedef struct
2  {
3      int TOPO;
4      int VAL[MAX];
5  }PILHA_SEQ;
6  void cria_pilha (PILHA_SEQ *);
7  int eh_vazia (PILHA_SEQ *);
8  void push (PILHA_SEQ *, int);
9  int top (PILHA_SEQ *);
10 void pop (PILHA_SEQ *);
11 int top_pop (PILHA_SEQ *);
```

```c
int top (PILHA_SEQ *p)
{
    if (eh_vazia(p))
    {
        printf ("\nERRO! Consulta na pilha vazia.\n");
        exit (2);
    }
    else
        return (p->VAL[p->TOPO]);
}
```

Pilha - Alocação Sequencial

Com base no que foi visto implemente a operação pop() que compõem o TAD PILHA_SEQ.

```
1  typedef struct
2  {
3      int TOPO;
4      int VAL[MAX];
5  }PILHA_SEQ;
6  void cria_pilha (PILHA_SEQ *);
7  int eh_vazia (PILHA_SEQ *);
8  void push (PILHA_SEQ *, int);
9  int top (PILHA_SEQ *);
10 void pop (PILHA_SEQ *);
11 int top_pop (PILHA_SEQ *);
```

```c
void pop (PILHA_SEQ *p)
{
    if (eh_vazia(p))
    {
        printf ("\nERRO! Retirada na pilha vazia.\n");
        exit (3);
    }
    else
        p->TOPO--;
}
```

Pilha - Alocação Sequencial

Com base no que foi visto implemente a operação top_pop() que compõem o TAD PILHA_SEQ.

```
1  typedef struct
2  {
3      int TOPO;
4      int VAL[MAX];
5  }PILHA_SEQ;
6  void cria_pilha (PILHA_SEQ *);
7  int eh_vazia (PILHA_SEQ *);
8  void push (PILHA_SEQ *, int);
9  int top (PILHA_SEQ *);
10 void pop (PILHA_SEQ *);
11 int top_pop (PILHA_SEQ *);
```

UNIVASF