Fila - Alocação Encadeada

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

```
typedef struct nodo
{
   int inf;
   struct nodo * next;
}NODO;
typedef struct
{
   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 cria_fila (FILA_ENC *pf)
{
   *pf=(DESCRITOR *)malloc(sizeof(DESCRITOR));
   if (!*pf)
   {
      printf ("\nERRO! Memoria insuficiente!\n");
      exit (1);
   }
   (*pf)->INICIO=(*pf)->FIM=NULL;
}
```

Fila - Alocação Encadeada

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

```
typedef struct nodo
{
   int inf;
   struct nodo * next;
}NODO;
typedef struct
{
   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);
```

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

Fila - Alocação Encadeada

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

```
typedef struct nodo
{
   int inf;
   struct nodo * next;
}NODO;
typedef struct
{
   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 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.

```
typedef struct nodo
{
   int inf;
   struct nodo * next;
}NODO;
typedef struct
{
   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);
```

```
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.

```
typedef struct nodo
{
   int inf;
   struct nodo * next;
}NODO;
typedef struct
{
   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 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.

```
typedef struct nodo
{
   int inf;
   struct nodo * next;
}NODO;
typedef struct
{
   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);
```

```
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);
   }
}
```

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.

UNIVASF

Filas – Exemplo de Aplicação

Uma aplicação interessante para filas é a ordenação por distribuição, descrita a seguir.

Seja uma lista l composta de n chaves, cada qual representada por um inteiro numa base $b > 1$. O problema consiste em ordenar essa lista.

O algoritmo utiliza b filas, denotadas por f_i, $0 \leq i \leq b-1$. Seja d o comprimento máximo da representação das chaves na base b. O algoritmo efetua d iterações, em cada uma das quais a tabela é percorrida.

Filas – Exemplo de Aplicação

A primeira iteração destaca, em cada nó, o dígito menos significativo da representação b-ária de cada chave. Se este for igual a k, a chave correspondente será inserida na fila f_k.

Ao terminar o percurso da tabela, esta se encontra distribuída pelas filas, que devem então ser concatenadas em sequência, isto é, f_0, depois f_1, f_2, etc.

Para essa tabela, já disposta numa ordem diferente da original, o processo deve ser repetido levando-se em consideração o segundo dígito da representação, ...

UNIVASF

Filas – Exemplo de Aplicação

e assim sucessivamente até que tenham sido feitas tantas distribuições quantos são os dígitos na chave de ordenação.

Observaremos agora um exemplo dessa ordenação, onde b=10 e d=2.

lista: **19 13 05 27 01 26 31 16 02 09 11 21 60 07**

Interação 1: 1ª distribuição (unidades simples)

lista: 19 13 05 27 01 26 31 16 02 09 11 21 60 07

$fila_0$: 6<u>0</u>

$fila_1$: 0<u>1</u>, 3<u>1</u>, 1<u>1</u>, 2<u>1</u>

$fila_2$: 0<u>2</u>

$fila_3$: 1<u>3</u>

$fila_4$:

$fila_5$: 0<u>5</u>

$fila_6$: 2<u>6</u>, 1<u>6</u>

$fila_7$: 2<u>7</u>, 0<u>7</u>

$fila_8$:

$fila_9$: 1<u>9</u>, 0<u>9</u>

lista: 60 01 31 11 21 02 13 05 26 16 27 07 19 09

Interação 2: 2ª distribuição (dezenas simples)

lista: 60 01 31 11 21 02 13 05 26 16 27 07 19 09

$fila_0$: 01, 02, 05, 07, 09

$fila_1$: 11, 13, 16, 19

$fila_2$: 21, 26, 27

$fila_3$: 31

$fila_4$:

$fila_5$:

$fila_6$: 60

$fila_7$:

$fila_8$:

$fila_9$:

lista: 01 02 05 07 09 11 13 16 19 21 26 27 31 60

Filas – Exemplo de Aplicação – Exercício

Utilizando-se dos TAD´s LISTA_ENC e FILA_ENC, implemente a operação ord_por_dist a qual recebe como entrada uma referência para uma lista encadeada de inteiros e a ordena através do processo de ordenação por distribuição.

OBS: b=10 e d=4.

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

Caracterização

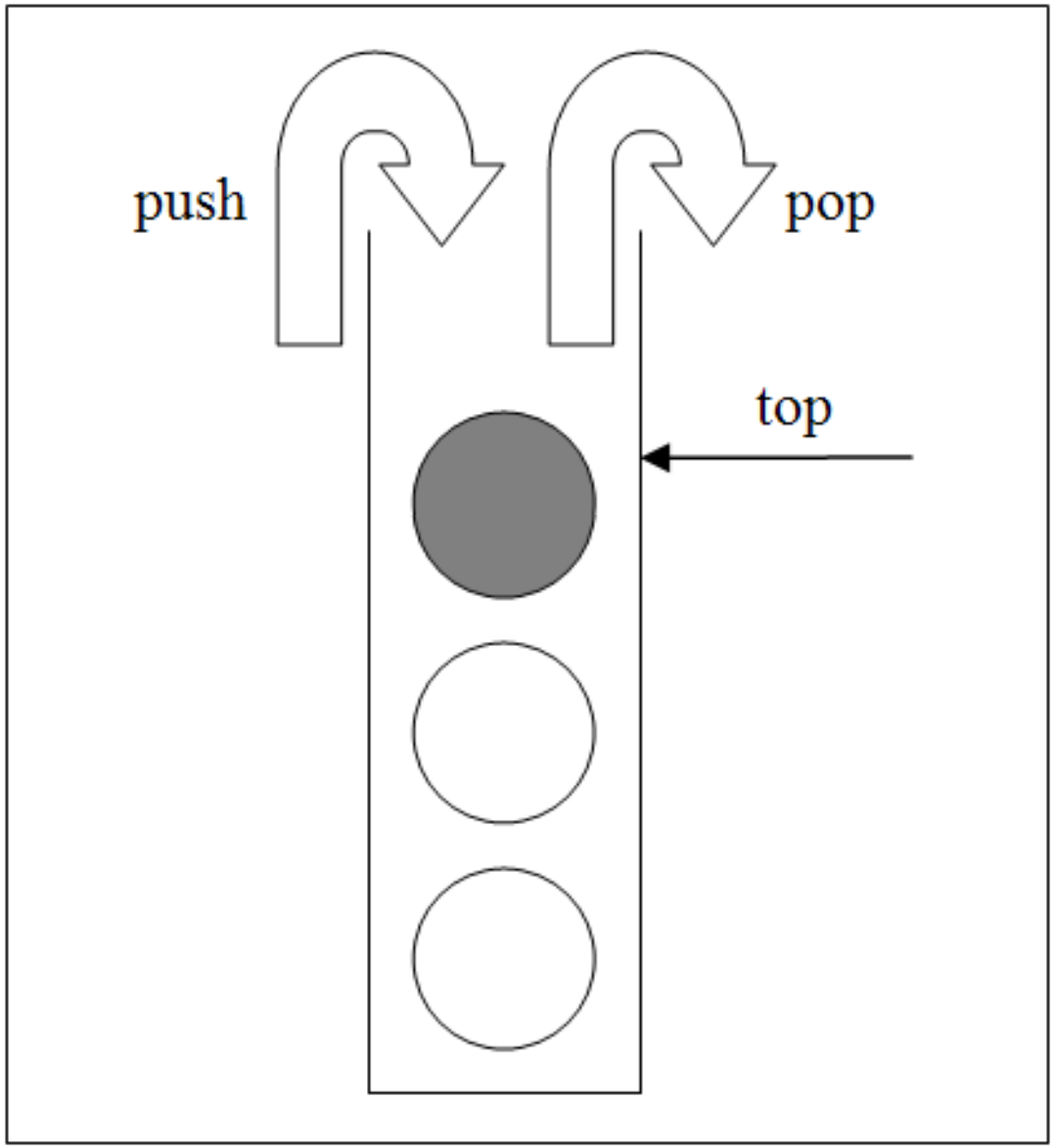
Uma pilha é uma lista com restrições de acesso, onde todas as operações só podem ser aplicadas sobre uma das extremidades da lista, denominada topo da pilha.

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 o de uma pilha de pratos, ou livros, etc, na qual só se pode visualizar (consultar) o último empilhado e este é o único que pode ser retirado; E, também, qualquer novo empilhamento (inserção) se fará sobre o último da pilha.

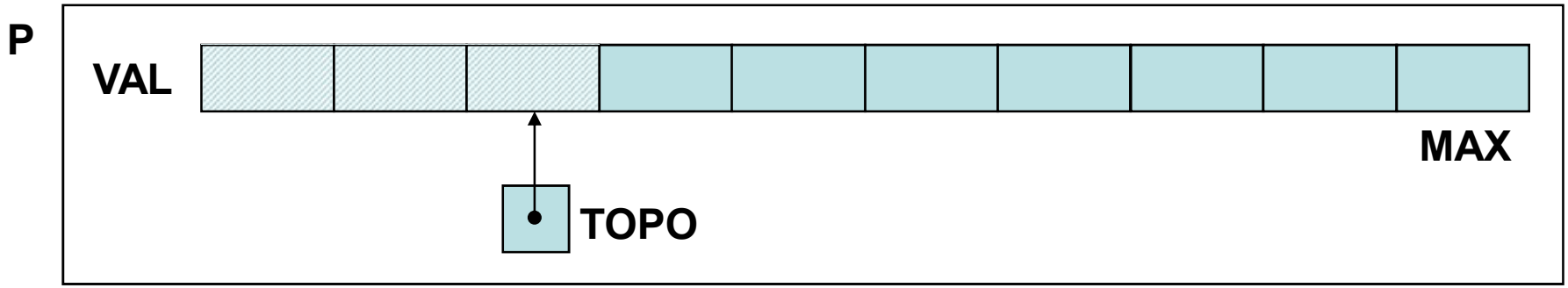
UNIVASF

Caracterização



Alocação Sequencial

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



A Implementação das operações é trivial:

- a pilha cresce do começo para o fim do vetor;

Alocação Sequencial

- uma pilha vazia tem o cursor TOPO igual a -1;

- a cada inserção, o cursor topo é incrementado para apontar para a próxima posição livre do vetor, onde é armazenado o novo elemento;

- uma retirada decrementa o cursor;

- uma consulta devolve o valor do elemento indexado por TOPO.

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

```
/*Definição do TAD PILHA_SEQ*/
typedef struct
{
   int TOPO; /*índice do topo da pilha*/
   int VAL[MAX];    /*vetor de elementos*/
}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 *);
```