ACH2023 – turmas 04 e 94 – EP2 2º Semestre de 2016

Fila de Prioridade – usando Heap

Filas de prioridades são estruturas de dados que gerenciam um conjunto de elementos, cada um com uma prioridade associada.
Dentre as operações previstas numa fila de prioridade estão:

- inserção de um elemento;
- exclusão do elemento de prioridade máxima;
- aumento de prioridade de um elemento;
- redução de prioridade de um elemento;
- consulta da prioridade de um elemento;
- consulta à quantidade de elementos (tamanho) da fila.

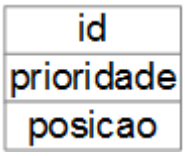
É desejável que todas estas operações sejam realizadas de maneira eficiente (**e neste EP a maior complexidade dessas operações será logarítmica**).
A eficiência dessas operações dependerá da maneira que a fila de prioridade foi implementada (e de sua estrutura subjacente).

Para este EP, vocês deverão implementar um conjunto de funções de gerenciamento de filas de prioridade utilizando principalmente dois conceitos: heap máximo e um arranjo auxiliar de elementos.

A seguir, serão apresentadas as estruturas de dados envolvidas nesta implementação e como elas serão gerenciadas.

A estrutura básica será o *REGISTRO*, que contém três campos: *id* (identificador inteiro do elemento), *prioridade* (número do tipo *float* com a prioridade do elemento), *posicao* (posição do registro no arranjo correspondente ao heap).

```
typedef struct {
  int id;
  float prioridade;
  int posicao;
} REGISTRO, * PONT;
```

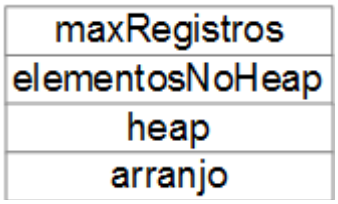


A estrutura *FILADEPRIORIDADE* possui quatro campos: *maxRegistros* é um campo do tipo inteiro que representa a quantidade máxima de registros permitidos na fila de prioridade atual (os *ids* válidos dos registros valerão de 0 [zero] até *maxRegistros*-1); *elementosNoHeap* é um campo do tipo inteiro que representa a quantidade de elementos efetivamente no heap; *heap* corresponde a um ponteiro para um arranjo de ponteiros para elementos do tipo *REGISTRO*, este arranjo representará o heap máximo e terá como tamanho *maxRegistros* e o campo *elementosNoHeap* determinará quantos elementos válidos estão no heap em um dado momento; *arranjo* corresponde a um ponteiro para um arranjo de ponteiros para elementos do tipo *REGISTRO*.

Quando da inicialização de uma fila de prioridades, tanto o arranjo apontado por *heap* quanto o

apontado por *arranjo* são criados com todos seus valores valendo *NULL*. Já que os *ids* válidos variam de 0 a *maxRegistros*-1, então há uma posição específica para guardar o endereço de cada *REGISTRO* (quando ele for criado) no arranjo *arranjo*, permitindo acesso rápido a um registro qualquer a partir de seu respectivo *id*.

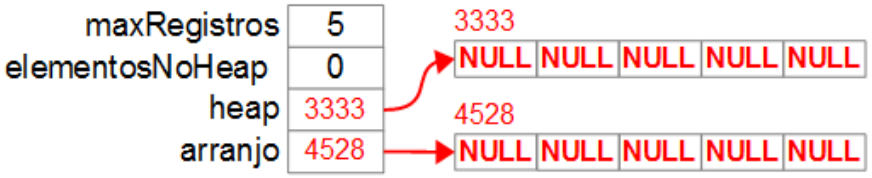
```
typedef struct {
  int maxRegistros;
  int elementosNoHeap;
  PONT* heap;
  PONT* arranjo;
} FILADEPRIORIDADE, * PFILA;
```



A função criarFila é responsável por criar uma nova fila de prioridade que poderá ter até *max r*egistros e deve retornar o endereço dessa fila de prioridades. Observe que os dois arranjos de ponteiros para registros já são criados e têm seus valores inicializados nessa função.

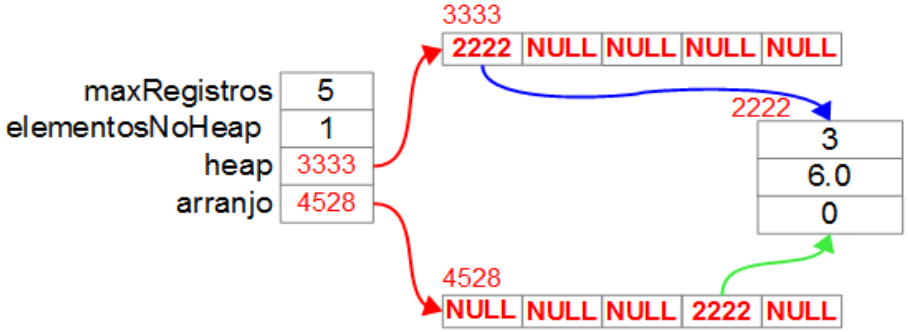
```
PFILA criarFila(int max){
  PFILA res = (PFILA) malloc(sizeof(FILADEPRIORIDADE));
  res->maxRegistros = max;
  res->arranjo = (PONT*) malloc(sizeof(PONT)*max);
  res->heap = (PONT*) malloc(sizeof(PONT)*max);
  int i;
  for (i=0;i<max;i++) {
    res->arranjo[i] = NULL;
    res->heap[i] = NULL;
  }
  res->elementosNoHeap = 0;
  return res;
}
```

Exemplo de fila de prioridade recém criada com *maxRegistros* = 5:

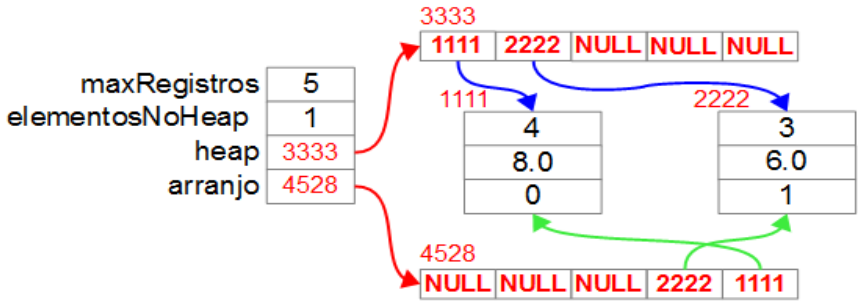


Ao se inserir um novo elemento na estrutura, este deverá ter seu endereço incluído no heap máximo e também deverá ter seu endereço armazenado na respectiva posição do arranjo (apontado pelo campo *arranjo*).

Exemplo de fila de prioridade após a inserção do elemento de *id*=3 e *prioridade*=6,0:



Exemplo de fila de prioridade após a inserção do elemento de *id*=4 e *prioridade*=8,0 (depois da inserção do elemento de *id*=3 e *prioridade*=6,0).



Ao se modificar a prioridade de um elemento da fila de prioridade, este deverá ser reposicionado dentro do heap máximo, caso necessário.

Funções que deverão ser implementadas no EP

int tamanho(PFILA f): função que recebe o endereço de uma fila de prioridade e retorna o número de elementos na fila (ou mais precisamente, o número de elementos válidos no heap máximo).

bool inserirElemento(PFILA f, int id, float prioridade): função que recebe o endereço de uma fila de prioridade, o identificador do novo elemento e o valor de sua prioridade.
Esta função deverá retornar *false* caso:

- o identificador seja inválido (menor que zero ou maior ou igual a *maxRegistros*);
- o identificador seja válido, mas já houver um registro com esse identificador na fila.

Caso contrário, a função deverá alocar memória para esse novo registro, colocar o endereço dele no arranjo de registros e colocar o endereço dele na posição correta do heap máximo (de acordo com sua prioridade), acertando todos os ponteiros necessários e o campo *posicao*, e retornar *true*.

bool aumentarPrioridade(PFILA f, int id, float novaPrioridade): função que recebe o endereço de uma fila de prioridade, o identificador do elemento e o novo valor de sua prioridade.
Esta função deverá retornar *false* caso:

- o identificador seja inválido (menor que zero ou maior ou igual a *maxRegistros*);
- o identificador seja válido, mas não haja um registro com esse identificador na fila.
- o identificador seja válido, mas sua prioridade já seja maior ou igual à nova prioridade passada como parâmetro da função.

Caso contrário, a função deverá atualizar a prioridade do registro, reposicioná-lo (se necessário) dentro da fila de prioridade (no heap máximo) e retornar *true*. Observação: esta função não deverá criar um novo registro.

bool reduzirPrioridade(PFILA f, int id, float novaPrioridade): função que recebe o endereço de uma fila de prioridade, o identificador do elemento e o novo valor de sua prioridade.
Esta função deverá retornar *false* caso:

- o identificador seja inválido (menor que zero ou maior ou igual a *maxRegistros*);
- o identificador seja válido, mas não haja um registro com esse identificador na fila.
- o identificador seja válido, mas sua prioridade já seja menor ou igual à nova prioridade passada como parâmetro da função.

Caso contrário, a função deverá atualizar a prioridade do registro, reposicioná-lo (se necessário) dentro da fila de prioridade (no heap máximo) e retornar *true*. Observação: esta função não deverá criar um novo registro.

PONT removerElemento(PFILA f): função que recebe como parâmetro o endereço de uma fila de prioridade e deverá retornar *NULL* caso a fila esteja vazia. Caso contrário, deverá retirar o primeiro elemento do heap máximo, reorganizar o heap (acertando os elementos e ponteiros necessários), colocar o valor *NULL* na posição correspondente desse elemento no arranjo *arranjo* e retornar o endereço do respectivo registro. A memória desse registro não deverá ser apagada, pois o usuário pode querer usar esse registro para alguma outra coisa.

bool consultarPrioridade(PFILA f, int id, float resposta)*: função que recebe o endereço de uma fila de prioridade, o identificador do elemento e um endereço para uma memória do tipo *float*.
Esta função deverá retornar *false* caso:

- o identificador seja inválido (menor que zero ou maior ou igual a *maxRegistros*);
- o identificador seja válido, mas não haja um registro com esse identificador na fila.

Caso contrário, a função deverá colocar na memória apontada pela variável *resposta* o valor da prioridade do respectivo elemento e retornar *true*.

Informações gerais:

Os EPs desta disciplina são trabalhos individuais que devem ser submetidos pelos alunos via sistema TIDIA até o dia 26/11 às 23:00h (com margem de tolerância de 59 minutos).

Vocês receberão três arquivos para este EP:

- FilaDePrioridade.h que contém a definição das estruturas, os *includes* necessários e o cabeçalho/assinatura das funções. Vocês não deverão alterar esse arquivo.
- FilaDePrioridade.c que conterá a implementação das funções solicitadas (e funções adicionais, caso julguem necessário). Este arquivo já contém o esqueleto geral das funções e alguns códigos implementados.
- usaFilaDePrioridade.c

Você deverá submeter **apenas** o arquivo FilaDePrioridade.c, porém renomeie este arquivo para seu número USP.c (por exemplo, 3140792.c) antes de submeter.

Não altere a assinatura de nenhuma das funções e não altere as funções originalmente implementadas (*exibirLog* e *criarFila*) .

Nenhuma das funções que vocês implementarem deverá imprimir algo. Para *debugar* o programa, você pode imprimir coisas, porém, na versão a ser entregue ao professor, suas funções não deverão imprimir nada (exceto pela função *exibirLog* que já imprime algumas informações).

Você poderá criar novas funções (auxiliares), mas não deve alterar o arquivo FilaDePrioridade.h. Seu código será testado com um versão diferente do arquivo usaFilaDePrioridade.c. Suas funções serão testadas individualmente e em grupo.

Todos os trabalhos passarão por um processo de verificação de plágios. Em caso de plágio, todos os alunos envolvidos receberão nota zero.