Heap

1 Struct

```
struct heap {
  int max; /* tamanho maximo do heap */
  int pos; /* proxima posicao disponivel no vetor */
  int* prioridade; /* vetor das prioridades */
};
```

2 Headers

```
static void heap_monta (Heap* heap, int n, int *prios)
Heap* heap_cria(int max, int qtos, int* iniciais)
void heap_insere(Heap* heap, int prioridade) {
static void corrige_acima(Heap* heap, int pos) {
int heap_remove(Heap* heap) {
static void corrige_abaixo(int* prios, int atual, int tam);
int* meuheapsort (int nums[], int tam) {
static void troca(int a, int b, int* v)
```

3 Implementation

3.1 Initializing

3.1.1 Heap Monta

```
static void heap_monta (Heap* heap, int n, int *prios){
  if (n>heap->max) {
    printf ("valores demais! \n");
    exit(1);
  }
   int pos = n/2;
   int i;
   int j;
   for(i=0,j = pos; j < n ; i++,j++)
      heap->prioridade[j] = prios[i];

   printf("valor de i: %d\n",i);

   for(i, j = pos-1; j >= 0; i++, j--){
      heap->prioridade[j] = prios[i];
      corrige_abaixo(heap->prioridade,j,n);
   }



   heap->pos = n;
}
```

3.1.2 Heap Cria

```
Heap* heap_cria(int max, int qtos, int* iniciais){
  Heap* heap=(Heap*)malloc(sizeof(struct heap));
```

```
  heap->max=max;
  heap->pos=0;
  heap->prioridade=(int *)malloc(max*sizeof(int));
  if (qtos>0) {
    heap_monta (heap, qtos, iniciais);
  }
  return heap;
}
```

3.2 Insertion

3.2.1 Insere

```
void heap_insere(Heap* heap, int prioridade) {
  if ( heap->pos < heap->max ) {
    heap->prioridade[heap->pos]=prioridade;
    corrige_acima(heap,heap->pos);
    heap->pos++;
    }
    else {
      printf("Heap CHEIO!\n");
      exit(1);
    }
}
```

3.2.2 Corrige Acima

```
static void corrige_acima(Heap* heap, int pos) {
  int pai;
  while (pos > 0){
    pai = (pos-1)/2;
    if (heap->prioridade[pai] < heap->prioridade[pos])
      troca(pos,pai,heap->prioridade);
    else
      break;
    pos=pai;
  }
}
```

3.3 Removing

3.3.1 Heap Remove

```
int heap_remove(Heap* heap) {
  if (heap->pos>0) {
    int topo=heap->prioridade[0];
    heap->prioridade[0]=heap->prioridade[heap->pos-1];
    heap->pos--;
    corrige_abaixo(heap->prioridade, 0, heap->pos);
    return topo;
  }
  else {
     printf("Heap VAZIO!");
     return -1;
  }
}

void heap_libera (Heap * h) {
```

```
  free(h->prioridade);
  free(h);
}
```

3.3.2 Corrige Abaixo

```
static void corrige_abaixo(int* prios, int atual, int tam){
  int pai=atual;
  int filho_esq, filho_dir, filho;
  while (2*pai+1 < tam){
    filho_esq=2*pai+1;
    filho_dir=2*pai+2;
    if (filho_dir >= tam) filho_dir=filho_esq;//??
    if (prios[filho_esq]>prios[filho_dir])
        filho=filho_esq;
    else
        filho=filho_dir;
    if (prios[pai]<prios[filho])
           troca(pai,filho,prios);
    else
           break;
    pai=filho;
}
```

3.4 Heap Sort

3.4.1 Cria Lista Ordenada

```
static int *cria_listaordenada (Heap *heap, int tam){

  int * vec = (int*)malloc(sizeof(int) * tam);
  int i = 0;

  while(heap->pos)
  {
    vec[tam-1-i] = heap_remove(heap);
    i++;

  }

  return vec;
}
```

3.4.2 Heap sort

```
int* meuheapsort (int nums[], int tam) {
  Heap *heap = heap_cria (tam, tam, nums);
  int *novosnums = cria_listaordenada (heap, tam); heap_libera(heap);
  return novosnums;
```

3.5 Auxiliary methods

```
void heap_libera (Heap * h) {
  free(h->prioridade);
  free(h);
}

void debug_heap_show (Heap *h, char* title){
```

```
  int i;
  printf("%s={",title);
  for (i=0; i<(h->pos); i++)
    printf("%d,",h->prioridade[i]);
  printf("}\n");

}
static void troca(int a, int b, int* v) {
  int f = v[a];
  v[a] = v[b];
  v[b] = f;
}
```
