Gabriel Káicon Batista Hilário SI-221

A princípio dormi na aula, por cansaço acumulado, em outro dia tive uma emergência pessoal, então pessoalmente fiquei atrasado na matéria e demorei a entender, procurei vídeo-aulas a respeito, busquei a ajuda do professor fora do horário de aula e só assim consegui entender de fato a estrutura, muito simples, me faltava um guia, para entender a ideia geral, e consertar o código, na remoção eu estava "destruindo" a informação, e não desconsiderando ela. A implementação foi bem mais fácil e tranquila, consegui entender a linha de raciocínio tranquilamente, comentei o código todo para quando for estudar ou até mesmo ensinar alguma pessoa, estou satisfeito com meu aprendizado nessa parte da matéria.
Abaixo terá um print do código, comentado o máximo possível, com as funções da estrutura Heap:

```
#include "FilaPrioridadeHeap.hpp"

// Cria uma nova fila de prioridade, aloca memória para ela e inicializa o contador de elementos.
fila_prioridade* cria_FilaPrio() {
    fila_prioridade* fp = new fila_prioridade;
    fp->qtd = 0; // Inicializa a quantidade de elementos como zero.
    return fp;   // Retorna o ponteiro para a nova fila de prioridade.
}

// Libera a memória alocada para a fila de prioridade.
void libera_FilaPrio(fila_prioridade* fp) {
    delete fp; // Libera a memória alocada para a fila.
}

// Consulta o paciente de maior prioridade na fila.
int consulta_FilaPrio(fila_prioridade* fp, char* nome) {
    if (fp == nullptr || fp->qtd == 0)
        return 0; // Retorna 0 se a fila está vazia ou é nula.

    strcpy(nome, fp->dados[0].nome); // Copia o nome do paciente de maior prioridade.
    return 1; // Retorna 1 indicando que a consulta foi bem-sucedida.
}

// Promove um elemento recém-inserido para sua posição correta no heap.
void promoverElemento(fila_prioridade* fp, int filho) {
    int pai;
    struct paciente temp;
    pai = (filho - 1) / 2; // Calcula o índice do pai do elemento.
    while ((filho > 0) && (fp->dados[pai].prio <= fp->dados[filho].prio)) {
        temp = fp->dados[filho];
        fp->dados[filho] = fp->dados[pai]; // Troca o elemento com o pai.
        fp->dados[pai] = temp; //fim da troca
        filho = pai;
        pai = (pai - 1) / 2; // Move para o pai do pai.
    }
}

// Insere um novo paciente na fila de prioridade.
int insere_FilaPrio(fila_prioridade* fp, char* nome, int prioridade) {
    if (fp == nullptr)
        return 0; // Retorna 0 se a fila é nula.

    if (fp->qtd == MAX)
        return 0; // Retorna 0 se a fila está cheia.

    // Insere o novo elemento e sua prioridade.
    strcpy(fp->dados[fp->qtd].nome, nome);
    fp->dados[fp->qtd].prio = prioridade;

    // Reorganiza o heap para manter as propriedades da fila de prioridade.
    promoverElemento(fp, fp->qtd);

    fp->qtd++; // Incrementa o contador de elementos.
    return 1;   // Retorna 1 para indicar a inserção bem-sucedida.
}
```

```cpp
// Rebaixa um elemento para sua posição correta após a remoção do elemento de maior prioridade.
void rebaixarElemento(fila_prioridade* fp, int pai) {
    struct paciente temp;
    int filho = 2 * pai + 1; // Calcula o índice do filho esquerdo.
    while (filho < fp->qtd) {

        // Verifica se o filho direito tem prioridade maior.
        if (filho < fp->qtd - 1 && fp->dados[filho].prio < fp->dados[filho + 1].prio)
            filho++; // Avança para o filho seguinte.

        // Compara a prioridade do pai com a do filho.
        if (fp->dados[pai].prio >= fp->dados[filho].prio)
            break; // Encontrou a posição correta.

        temp = fp->dados[pai];
        fp->dados[pai] = fp->dados[filho]; // Troca o elemento com o filho.
        fp->dados[filho] = temp;

        pai = filho;
        filho = 2 * pai + 1; // Move para o próximo nível.
    }
}

// Remove o paciente de maior prioridade da fila.
int remove_FilaPrio(fila_prioridade* fp) {
    if (fp == nullptr)
        return 0; // Retorna 0 se a fila é nula.

    if (fp->qtd == 0)
        return 0; // Retorna 0 se a fila está vazia.

    fp->qtd--; // Decrementa o contador de elementos.
    fp->dados[0] = fp->dados[fp->qtd]; // Move o último elemento para a raiz.

    // Reorganiza o heap após a remoção.
    rebaixarElemento(fp, 0);

    return 1; // Retorna 1 para indicar a remoção bem-sucedida.
}

// Retorna a quantidade de elementos na fila.
int tamanho_FilaPrio(fila_prioridade* fp) {
    if (fp == nullptr)
        return -1; // Retorna -1 se a fila é nula.
    else
        return fp->qtd; // Retorna a quantidade de elementos.
}

// Verifica se a fila de prioridade está cheia.
int estaCheia_FilaPrio(fila_prioridade* fp) {
    if (fp == nullptr)
        return -1; // Retorna -1 se a fila é nula.
    return (fp->qtd == MAX); // Retorna 1 se a fila está cheia, 0 caso contrário.
}

// Verifica se a fila de prioridade está vazia.
int estaVazia_FilaPrio(fila_prioridade* fp) {
    if (fp == nullptr)
        return -1; // Retorna -1 se a fila é nula.
    return (fp->qtd == 0); // Retorna 1 se a fila está vazia, 0 caso contrário.
}

// Imprime os elementos da fila de prioridade junto com suas prioridades e índices.
void imprime_FilaPrio(fila_prioridade* fp) {
    if (fp == nullptr)
        return; // Retorna sem fazer nada se a fila é nula.

    for (int i = 0; i < fp->qtd; i++)
        cout << i << ") Prio: " << fp->dados[i].prio << "\tNome: " << fp->dados[i].nome << endl;
    // Imprime o índice, prioridade e nome de cada paciente na fila.
}
```

Todas as funções essas funções foram escritas em c++, armazenadas em um arquivo *.cpp, e possuem um cabeçalho *.hpp, Logo abaixo temos o cabeçalho das funções acima, junto com as structs que farão parte do Heap:

```cpp
#ifndef FILAPRIORIDADEHEAP_H
#define FILAPRIORIDADEHEAP_H

#include <iostream>
#include <cstring>
#include <windows.h>

const int MAX = 100;

using namespace std;
typedef struct paciente {
    char nome[30];
    int prio;
};
typedef struct fila_prioridade {
    int qtd;
    struct paciente dados[MAX];
};

typedef fila_prioridade FilaPrio;

FilaPrio* cria_FilaPrio();
void libera_FilaPrio(FilaPrio* fp);
int consulta_FilaPrio(FilaPrio* fp, char* nome);
int insere_FilaPrio(FilaPrio* fp, char* nome, int prioridade);
int remove_FilaPrio(FilaPrio* fp);
int tamanho_FilaPrio(FilaPrio* fp);
int estaCheia_FilaPrio(FilaPrio* fp);
int estaVazia_FilaPrio(FilaPrio* fp);
void imprime_FilaPrio(FilaPrio* fp);

#endif
```

Logo abaixo temos a Main, onde os testes foram realizados, e logo abaixo temos o print do prompt de comando, onde o arquivo c++, foi executado.

```cpp
#include "FilaPrioridadeHeap.cpp"

int main() {

    UINT CPAGE_UTF8 = 65001;
    UINT CPAGE_DEFAULT = GetConsoleOutputCP();
    SetConsoleOutputCP(CPAGE_UTF8);

    FilaPrio* fp = cria_FilaPrio();
    paciente itens[6] = {{"Andre", 1}, {"Bianca", 2},{"Carlos", 5}, {"Nilza", 8}, {"João Vitor", 6}, {"Kaicon", 4} };

    cout << "\n\nHeap como foi inserido" <<  endl;
    for (int i = 0; i < 6; i++) {
        cout << i << ") Prio: " << itens[i].prio << " nome: " << itens[i].nome << endl;
        insere_FilaPrio(fp, itens[i].nome, itens[i].prio);
    }
    cout << "\nHeap após o primeiro rearranjo" << endl;
    cout << "================================" << endl;
    imprime_FilaPrio(fp);

    // teste
    cout << "Heap com a inserção do Teste, e obviamente executando o rearranjo" << endl;
    cout << "================================" << endl;
    insere_FilaPrio(fp, "Teste", 9);
    imprime_FilaPrio(fp);

    cout << "Heap 'jogando' o maior para o final, e desconsiderando ele como informação, e imprimindo o restante" << endl;
    cout << "================================" << endl;
    remove_FilaPrio(fp);
    imprime_FilaPrio(fp);

    cout << "Impressão final de como a fila está, após o rearranjo" << endl;
    cout << "================================" << endl;
    for (int i = 0; i < 6; i++) {
        char nome[30];
        consulta_FilaPrio(fp, nome);
        cout << i << ") " << nome << endl;
        remove_FilaPrio(fp);
    }

    libera_FilaPrio(fp);
    return 0;
}
```

C:\Windows\system32\cmd.exe

```
Heap como foi inserido
0) Prio: 1 nome: Andre
1) Prio: 2 nome: Bianca
2) Prio: 5 nome: Carlos
3) Prio: 8 nome: Nilza
4) Prio: 6 nome: João Vitor
5) Prio: 4 nome: Kaicon

Heap após o primeiro rearranjo
================================
0) Prio: 8      Nome: Nilza
1) Prio: 6      Nome: João Vitor
2) Prio: 4      Nome: Kaicon
3) Prio: 1      Nome: Andre
4) Prio: 5      Nome: Carlos
5) Prio: 2      Nome: Bianca
Heap com a inserção do Teste, e obviamente executando o rearranjo
================================
0) Prio: 9      Nome: Teste
1) Prio: 6      Nome: João Vitor
2) Prio: 8      Nome: Nilza
3) Prio: 1      Nome: Andre
4) Prio: 5      Nome: Carlos
5) Prio: 2      Nome: Bianca
6) Prio: 4      Nome: Kaicon
Heap 'jogando' o maior para o final, e desconsiderando ele como informação, e imprimindo o restante
================================
0) Prio: 8      Nome: Nilza
1) Prio: 6      Nome: João Vitor
2) Prio: 4      Nome: Kaicon
3) Prio: 1      Nome: Andre
4) Prio: 5      Nome: Carlos
5) Prio: 2      Nome: Bianca
Impressão final de como a fila está, após o rearranjo
================================
0) Nilza
1) João Vitor
2) Carlos
3) Kaicon
4) Bianca
5) Andre

Pressione qualquer tecla para continuar. . .
```