INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE MINAS GERAIS – CAMPUS SÃO JOÃO EVANGELISTA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

RHANNA RAMOS DE QUADROS

TRABALHO PRÁTICO III

SÃO JOÃO EVANGELISTA
2022

RHANNA RAMOS DE QUADROS

Sistema de gerenciamento de uma UPA

SÃO JOÃO EVANGELISTA

2022

SUMÁRIO

1. INTRODUÇÃO

Esta documentação tem cunho avaliativo, exigido como forma de avaliação para a disciplina de Algoritmos e Estruturas de Dados I, porém também tem cunho expositivo, onde é descrito todo o caminho percorrido para realização do trabalho prático.

1.1. Objetivo Geral

Este trabalho tem como objetivo geral apresentar na prática os conhecimentos adquiridos nas aulas de Algoritmos e Estruturas de Dados I, a respeito de fila com ponteiro e prioridade, manipulação de tempo, utilizando a linguagem de C/C++.

1.2. Objetivos Específicos

Esse trabalho tem como objetivos específicos:

- Aplicar conhecimentos sobre fila com ponteiro, integrado a fila de prioridade;
- Aplicar conhecimentos a respeito das bibliotecas chrono e vector.
- Aplicar e aprimorar os conhecimentos com a biblioteca time.
- Aplicar conhecimentos adquiridos, em um minissistema de delivery.

1.3. Justificativa

O código inteiro foi escrito na linguagem C/C++, com maior foco na linguagem C++. Quando se aprende sobre Estruturas de Dados, vemos o conteúdo a respeito de Ponteiros como base para todas as estruturas que vamos utilizar ao longo do curso, aprendemos sobre o uso de bibliotecas, como a time.h, chrono e vector, sendo essas 2 primeiras para manipulação de tempo dentro do C++, e última para manipulação de vetores. Posteriormente vimos conteúdos de Estruturas de Dados mesmo, e por último foi visto o conteúdo a respeito de fila, que seria semelhante a uma pilha pela simplicidade, a fila, porém, é horizontal, e ao invés de ter o conceito de “primeiro que entra último que sai”, temos que “o primeiro que entra é o primeiro que sai”. Inicialmente nos é apresentado 3 tipos de fila, a com arranjo que é com alocação estática, ou seja, possui um limite pré-definido, podendo ter um número limitado de itens, a com ponteiro que seria a com alocação dinâmica, ou seja, não possui um limite pré-definido podendo ter um número infinito de itens, e a fila com prioridade, bem semelhante a que existe no mundo real, e pode ser aplicada tanto na fila com arranjo

quanto na com ponteiro, onde inserimos itens na fila de acordo com a prioridade do item.

No trabalho irei apresentar os conceitos de fila de prioridade com ponteiro, junto às novas bibliotecas apresentadas a chrono e time.h(ambas para manipulação de tempo) e a vector(manipular vetores). Todos esses conceitos foram aplicados para realização do trabalho, e assim montar um sistema para gerir uma UPA.

2. DESENVOLVIMENTO

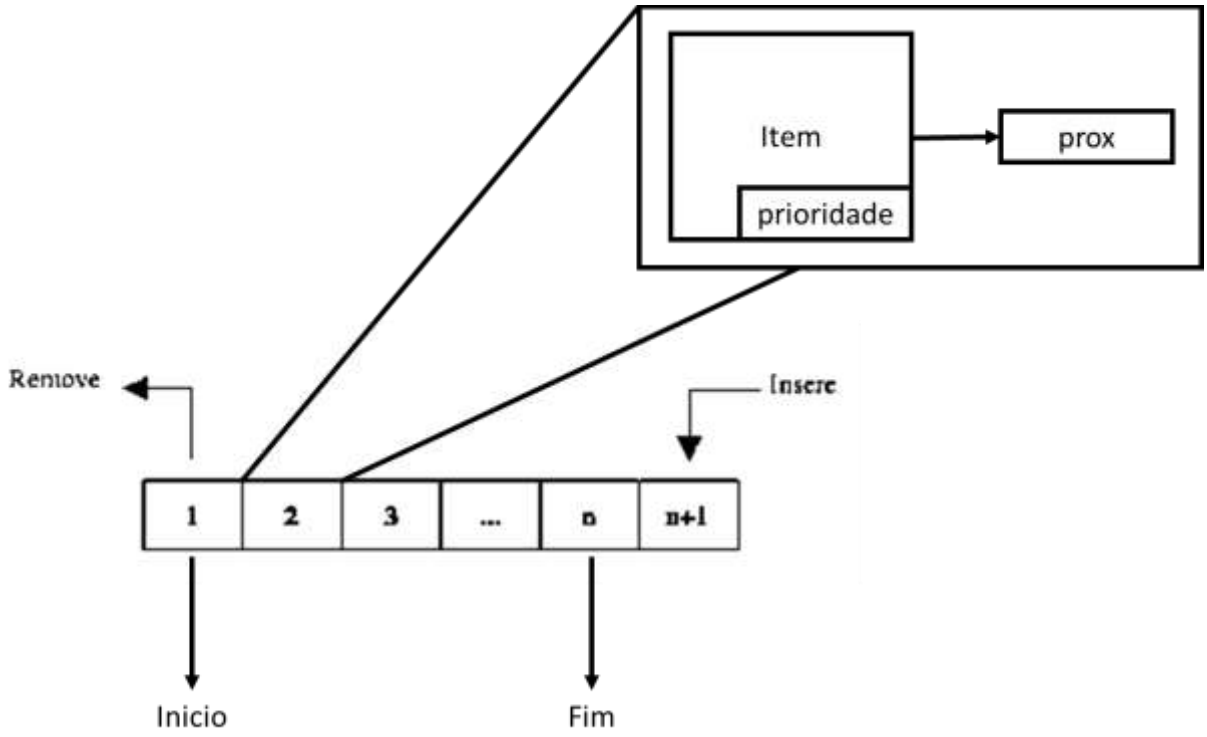
Nesta seção do documento é apresentado o desenvolvimento do trabalho.

2.1. Conceitos: Fila, TADs e bibliotecas

Aqui falo de forma bem superficial o que contém no meu programa.

Sobre as TADs, Tipos Abstratos de Dados, em C++ às declaramos como *structs*, ou estruturas de Dados acompanhados com funções para manipular a mesma. *Structs* são representações de qualquer coisa no mundo real, sendo ela lógica ou física, como por exemplo uma pessoa, que é algo físico, ou um filme digital, cada um com suas características específicas. Uma pessoa tem sexo, idade, nome, altura, dentre outras, e um filme tem o elenco, personagens, duração, categoria, ano de lançamento, dentre outros, e tudo isso poderia estar dentro de uma *struct* definida para cada um deles. Resumindo uma *Struct* é uma espécie de modelo para cadastrar diferentes itens, que tem características comuns, em uma lista, dentro de um software escrito em C/C++. Acompanhado das *structs* temos as funções para manipulação dos dados dessa lista, e desses itens. Juntos eles compõem o que chamamos de TAD.

Figura 1



Na figura 1, vemos um desenho esquemático de como seria uma fila com ponteiro. Ela é bem semelhante a um vetor, porém a tipagem de dados a ser inserida podem

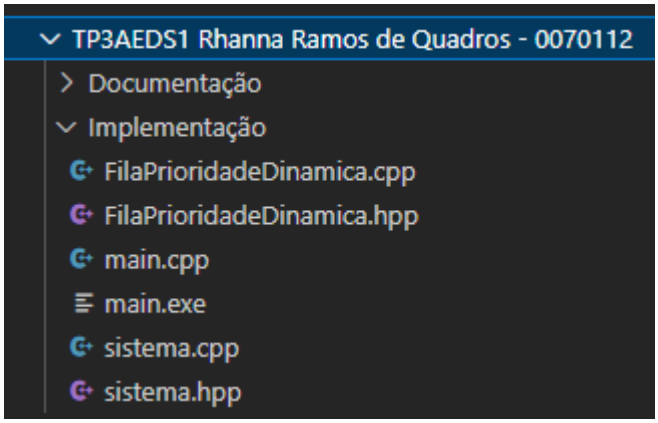
ser as structs que são usadas para representar coisas do mundo real, podendo armazenar mais de um tipo de variável dentro de um único “quadradinho”.

Os elementos são inseridos dentro de um espaço do vetor, ao se inserir novos elementos, eles são alocados no Ultimo, e assim sucessivamente, porém quando se trata de sair, utilizamos o conceito de FIFO - *First in, First out*, ou seja, o primeiro que entra é o primeiro que sai. Quando se trata de uma lista de prioridade, alguns serão inseridos na frente de outros, e isso muda, mas aí podemos definir que a primeira posição sempre é a primeira a sair.

Agora temos nossas bibliotecas, a chrono e time.h, para manipulação de tempo e a vector, para manipular vetores. Chrono foi usada para pegar os horários do sistema de acordo com o momento da execução, e definir tipos de variáveis. Time.h foi usada para sorteio e definição de tipos de variáveis. Vector foi usada para guardar structs, dentro de um laço de repetição, e dentro de outras structs.

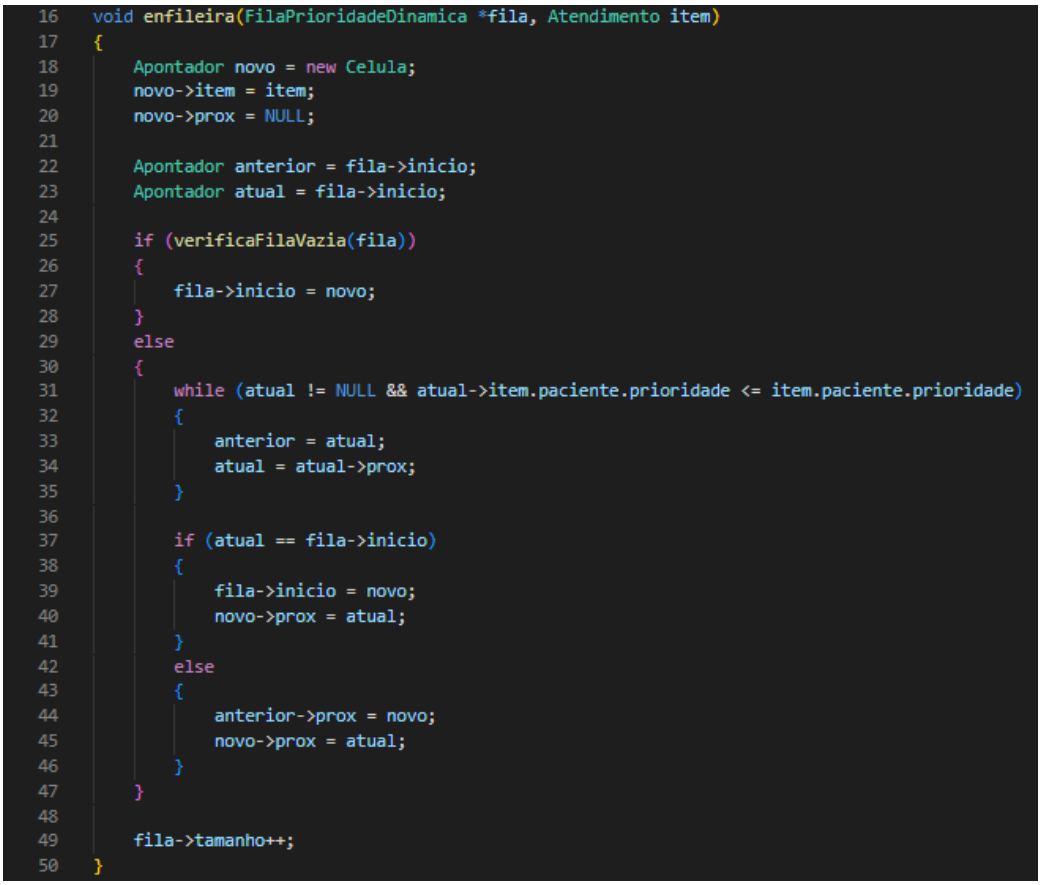
2.2. Implementação

Figura 2



Na figura 2, vemos como foi feita a divisão de arquivos por modularização. Dois arquivos *.cpp e seus respectivos arquivos cabeça(*.hpp), e a main(*.cpp).

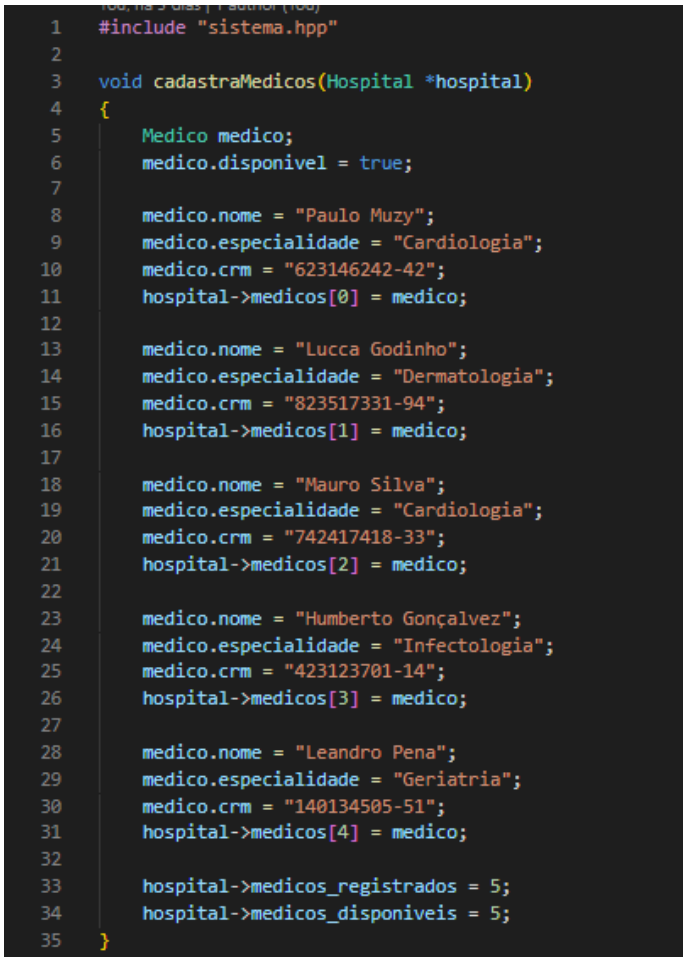
Figura 3



```
16  void enfileira(FilaPrioridadeDinamica *fila, Atendimento item)
17  {
18      Apontador novo = new Celula;
19      novo->item = item;
20      novo->prox = NULL;
21
22      Apontador anterior = fila->inicio;
23      Apontador atual = fila->inicio;
24
25      if (verificaFilaVazia(fila))
26      {
27          fila->inicio = novo;
28      }
29      else
30      {
31          while (atual != NULL && atual->item.paciente.prioridade <= item.paciente.prioridade)
32          {
33              anterior = atual;
34              atual = atual->prox;
35          }
36
37          if (atual == fila->inicio)
38          {
39              fila->inicio = novo;
40              novo->prox = atual;
41          }
42          else
43          {
44              anterior->prox = novo;
45              novo->prox = atual;
46          }
47      }
48
49      fila->tamanho++;
50  }
```

Foram utilizadas as TADs de fila (Apêndice A), com alterações no Enfileira (figura 3), que insere de acordo com a prioridade. O 1° if, serve para a 1ªinserção, o else serve para segunda inserção em diante, usando os apontadores atual(percorrer o laço) e anterior(guardar o valor anterior, para se inserir após ele). Após isso temos o cabeçalho das TADs de fila(Apêndice B). Mais a frente temos o cabeçalho(Apêndice C) do arquivo do sistema. E o arquivo sistema.cpp, será explicado logo abaixo.

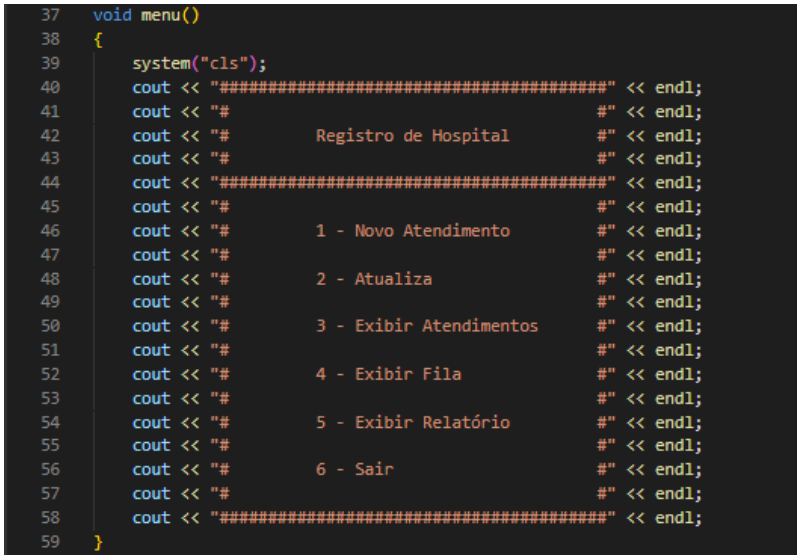
Figura 4



```
#include "sistema.hpp"

void cadastraMedicos(Hospital *hospital)
{
    Medico medico;
    medico.disponivel = true;

    medico.nome = "Paulo Muzy";
    medico.especialidade = "Cardiologia";
    medico.crm = "623146242-42";
    hospital->medicos[0] = medico;

    medico.nome = "Lucca Godinho";
    medico.especialidade = "Dermatologia";
    medico.crm = "823517331-94";
    hospital->medicos[1] = medico;

    medico.nome = "Mauro Silva";
    medico.especialidade = "Cardiologia";
    medico.crm = "742417418-33";
    hospital->medicos[2] = medico;

    medico.nome = "Humberto Gonçalvez";
    medico.especialidade = "Infectologia";
    medico.crm = "423123701-14";
    hospital->medicos[3] = medico;

    medico.nome = "Leandro Pena";
    medico.especialidade = "Geriatria";
    medico.crm = "140134505-51";
    hospital->medicos[4] = medico;

    hospital->medicos_registrados = 5;
    hospital->medicos_disponiveis = 5;
}
```

Na figura 4, temos uma função que insere 5 médicos. Usando apenas uma variável de tipo medico, alterando-a, salvando-o em 1 posição do vetor, e sobrescrevendo a variável, para salvar em outra posição, e assim suscetivelmente. Ele é executado na main.cpp (Apêndice D) após o outro, no inicio da execução do programa, para que não haja perda de tempo ao cadastrar eles.

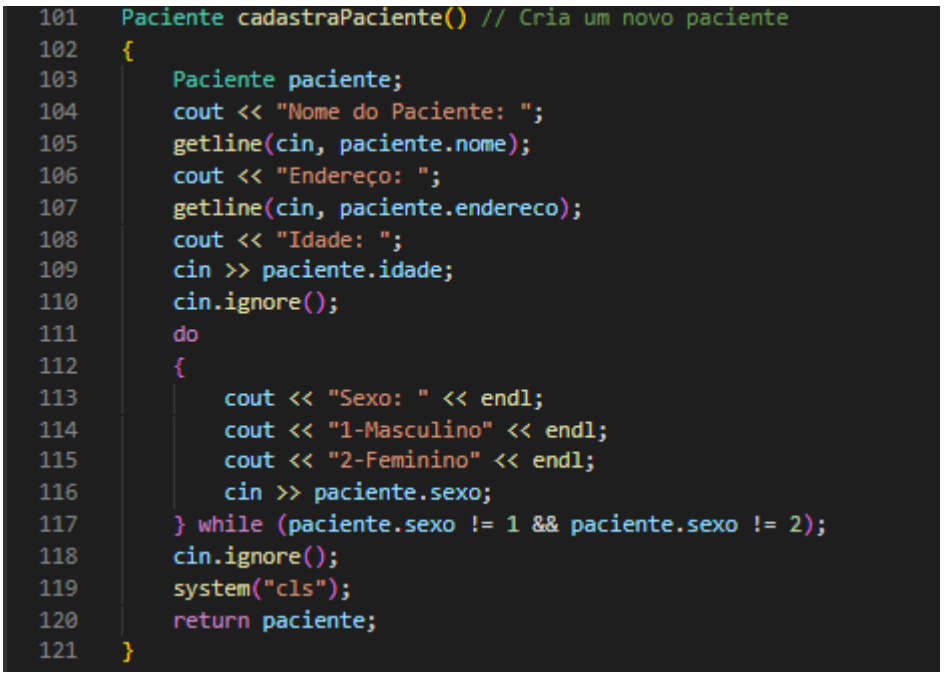
Figura 5



```
void menu()
{
    system("cls");
    cout << "#####################################" << endl;
    cout << "#                                   #" << endl;
    cout << "#         Registro de Hospital      #" << endl;
    cout << "#                                   #" << endl;
    cout << "#####################################" << endl;
    cout << "#                                   #" << endl;
    cout << "#         1 - Novo Atendimento      #" << endl;
    cout << "#                                   #" << endl;
    cout << "#         2 - Atualiza              #" << endl;
    cout << "#                                   #" << endl;
    cout << "#         3 - Exibir Atendimentos   #" << endl;
    cout << "#                                   #" << endl;
    cout << "#         4 - Exibir Fila           #" << endl;
    cout << "#                                   #" << endl;
    cout << "#         5 - Exibir Relatório      #" << endl;
    cout << "#                                   #" << endl;
    cout << "#         6 - Sair                  #" << endl;
    cout << "#                                   #" << endl;
    cout << "#####################################" << endl;
}
```

Na figura 5, temos o menu, que será utilizado para escolher o que será feito, de acordo com as opções.

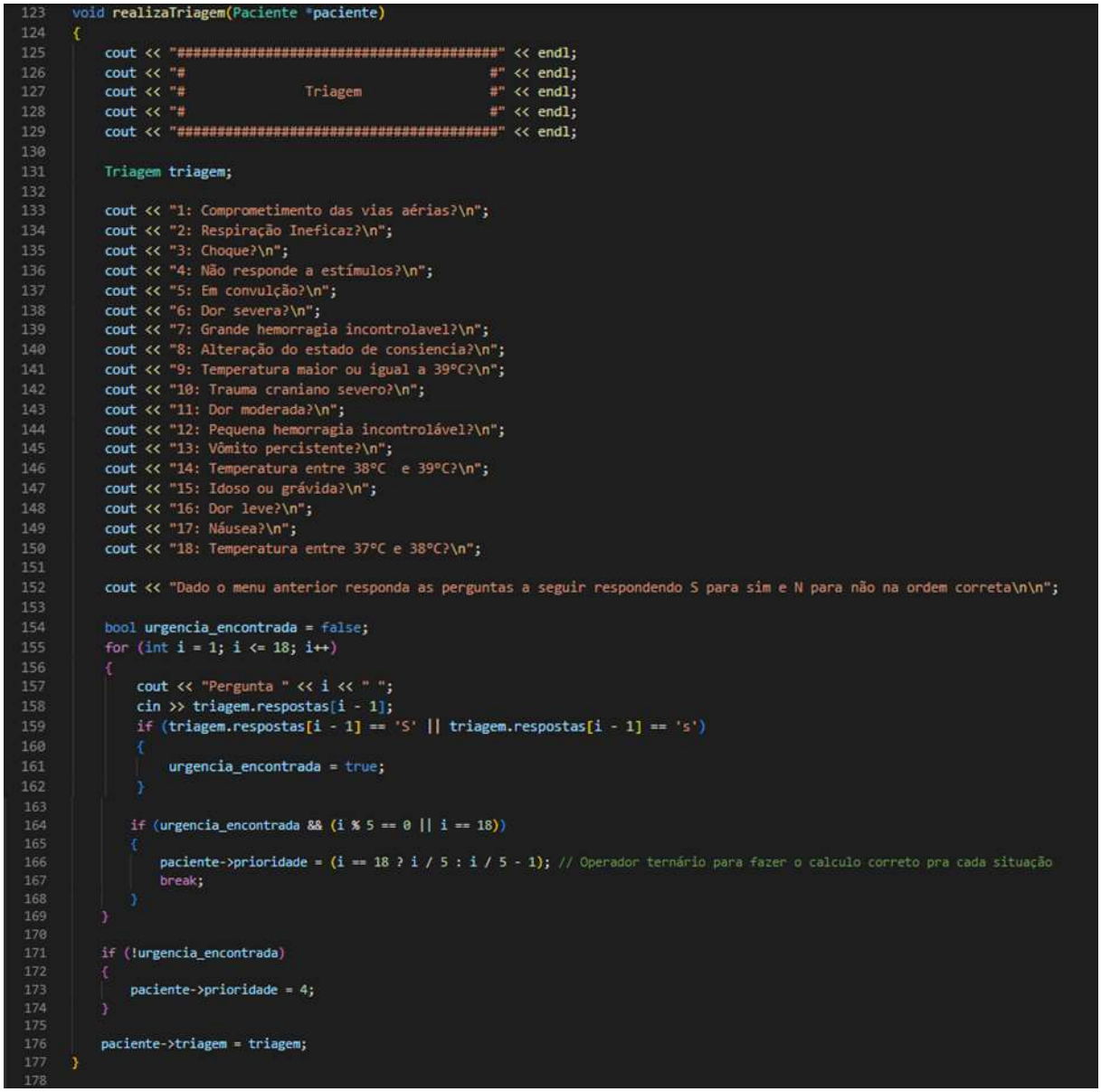
Figura 6



```
Paciente cadastraPaciente() // Cria um novo paciente
{
    Paciente paciente;
    cout << "Nome do Paciente: ";
    getline(cin, paciente.nome);
    cout << "Endereço: ";
    getline(cin, paciente.endereco);
    cout << "Idade: ";
    cin >> paciente.idade;
    cin.ignore();
    do
    {
        cout << "Sexo: " << endl;
        cout << "1-Masculino" << endl;
        cout << "2-Feminino" << endl;
        cin >> paciente.sexo;
    } while (paciente.sexo != 1 && paciente.sexo != 2);
    cin.ignore();
    system("cls");
    return paciente;
}
```

Temos a função de cadastro de paciente, que cria o paciente de forma local, insere os dados dele, e retorna ele, para ser usado como paramêtro em outras funções.

Figura 7



```
void realizaTriagem(Paciente *paciente)
{
    cout << "#####################################" << endl;
    cout << "#                                   #" << endl;
    cout << "#              Triagem              #" << endl;
    cout << "#                                   #" << endl;
    cout << "#####################################" << endl;

    Triagem triagem;

    cout << "1: Comprometimento das vias aérias?\n";
    cout << "2: Respiração Ineficaz?\n";
    cout << "3: Choque?\n";
    cout << "4: Não responde a estímulos?\n";
    cout << "5: Em convulção?\n";
    cout << "6: Dor severa?\n";
    cout << "7: Grande hemorragia incontrolavel?\n";
    cout << "8: Alteração do estado de consiencia?\n";
    cout << "9: Temperatura maior ou igual a 39°C?\n";
    cout << "10: Trauma craniano severo?\n";
    cout << "11: Dor moderada?\n";
    cout << "12: Pequena hemorragia incontrolável?\n";
    cout << "13: Vômito percistente?\n";
    cout << "14: Temperatura entre 38°C  e 39°C?\n";
    cout << "15: Idoso ou grávida?\n";
    cout << "16: Dor leve?\n";
    cout << "17: Náusea?\n";
    cout << "18: Temperatura entre 37°C e 38°C?\n";

    cout << "Dado o menu anterior responda as perguntas a seguir respondendo S para sim e N para não na ordem correta\n\n";

    bool urgencia_encontrada = false;
    for (int i = 1; i <= 18; i++)
    {
        cout << "Pergunta " << i << " ";
        cin >> triagem.respostas[i - 1];
        if (triagem.respostas[i - 1] == 'S' || triagem.respostas[i - 1] == 's')
        {
            urgencia_encontrada = true;
        }

       if (urgencia_encontrada && (i % 5 == 0 || i == 18))
       {
           paciente->prioridade = (i == 18 ? i / 5 : i / 5 - 1); // Operador ternário para fazer o calculo correto pra cada situação
           break;
       }
    }

    if (!urgencia_encontrada)
    {
        paciente->prioridade = 4;
    }

    paciente->triagem = triagem;
}
```

Temos a triagem, um conjunto de 18 perguntas, com todas as respostas inseridas como N. a variavel urgencia_encontrada, serve para definir a urgencia com que o paciente deve ser atendido. O primeiro if, serve para indicar se a urgencia foi encontrada(lembrando que há os níveis de urgência), no segundo if, serve para caso a urgencia tenha sido encontrada nas primeiras 5 perguntas, ele já pula direto pro atendimento, ou então nas primeiras 10, ou então nas primeiras 15, ou então no fim das 18. Caso não haja urgencia, ele será definido com o grau mais baixo de urgência.

Figura 8

```
179   void mostraPrevisaoAtendimento(Paciente paciente)
180   {
181       string previsao_de_atendimento;
182       switch (paciente.prioridade)
183       {
184       case 0:
185           previsao_de_atendimento = "Atendimento Imediato";
186           break;
187
188       case 1:
189           previsao_de_atendimento = "Atendimento em até 10 minutos";
190           break;
191
192       case 2:
193           previsao_de_atendimento = "Atendimento em até 60 minutos";
194           break;
195
196       case 3:
197           previsao_de_atendimento = "Atendimento em até 120 minutos";
198           break;
199
200       case 4:
201           previsao_de_atendimento = "Atendimento em até 240 minutos";
202           break;
203       }
204       cout << "Previsão de Atendimento: " << previsao_de_atendimento << endl;
205   }
```

De acordo com a urgência encontrada na última função apresentada, essa função só vai retorna a mensagem falando qual o nível de urgência.

Figura 9

```
61  void novoAtendimento(Hospital *hospital)
62  {
63      vector<Paciente> lista_pacientes; // Lista de pacientes para realizar atendimento
64      int opcao;
65      do
66      {
67          system("cls");
68          cout << "#######################################" << endl;
69          cout << "#                                     #" << endl;
70          cout << "#           Novo Atendimento          #" << endl;
71          cout << "#                                     #" << endl;
72          cout << "#######################################" << endl;
73
74          Paciente novo_paciente = cadastraPaciente();
75          realizaTriagem(&novo_paciente);
76          lista_pacientes.push_back(novo_paciente); // Adiciona o novo paciente na lista
77          system("cls");
78          mostraUrgencia(novo_paciente);
79          mostraPrevisaoAtendimento(novo_paciente);
80          cout << "\n";
81
82          do
83          {
84              cout << "Deseja realizar um novo atendimento?" << endl;
85              cout << "1-Sim" << endl;
86              cout << "2-Não" << endl;
87              cin >> opcao;
88              cin.ignore();
89          } while (opcao != 1 && opcao != 2);
90      } while (opcao == 1);
91
92      for (Paciente paciente : lista_pacientes) // Percorre a lista de pacientes
93      {
94          Atendimento novo_atendimento;
95          novo_atendimento.paciente = paciente;
96          enfileira(&hospital->fila_de_espera, novo_atendimento);
97      }
98      atualiza(hospital);
99  }
```

Na figura 9, temos a variavel paciente, recebendo o retorno da função na figura 6, logo em seguida esse mesmo retorno é mandado para a triagem, e ele é colocado na lista de vetores, depois é mostrado o nível de urgencia visto no apêndice A, e depois a previsão de atendimento conforme a figura 8. Depois é perguntado se vai querer fazer um novo atendimento, caso sim, ele vai repetir o processo, e quando a resposta for não, ele vai entrar no forrange, percorrendo a lista de pacientes, e inserindo cada um deles na fila, e depois ele atualiza o que já foi feito antes, por meio da função apresentada na figura 11, se foi feito algo.

Figura 10

```
void exibeAtendimentos(Hospital hospital)
{
    if (hospital.atendimentos_em_execucao.size() > 0)
    {
        for (Atendimento atendimento : hospital.atendimentos_em_execucao)
        {
            cout << "Médico: " << atendimento.medico->nome << endl;
            cout << "Paciente: " << atendimento.paciente.nome << endl;
            mostraUrgencia(atendimento.paciente);
            cout << "Início: " << put_time(&atendimento.data_inicio, "%d/%m/%Y %H:%M:%S") << endl;              // Transforma a struct tm em string
            cout << "Previsão de Término: " << put_time(&atendimento.data_termino, "%d/%m/%Y %H:%M:%S") << endl; // Transforma a struct tm em string
            cout << "\n";
        }

        system("pause");
    }
    else
    {
        cout << "Nenhum atendimento em execução" << endl;
        Sleep(1000);
    }
}

void exibeFilaEspera(Hospital hospital)
{
    cout << "#####################################" << endl;
    cout << "#                                   #" << endl;
    cout << "#           Fila de Espera          #" << endl;
    cout << "#                                   #" << endl;
    cout << "#####################################" << endl;

    imprimeFila(&hospital.fila_de_espera);
    system("pause");
}

void exibeRelatorio(Hospital hospital)
{
    cout << "#####################################" << endl;
    cout << "#                                   #" << endl;
    cout << "#             Relatório             #" << endl;
    cout << "#                                   #" << endl;
    cout << "#####################################" << endl;

    cout << "Atendimentos em execução: " << hospital.atendimentos_em_execucao.size() << endl;
    cout << "Médicos Registrados: " << hospital.medicos_registrados << endl;
    cout << "Médicos Disponíveis: " << hospital.medicos_disponiveis << endl;
    cout << "Pacientes Tratados: " << hospital.pacientes_tratados << endl;
    cout << "Pacientes Em Espera: " << hospital.fila_de_espera.tamanho << endl;
    system("pause");
}
```

Na figura 10, vemos 3 funções de exibição, a exibe atendimento, que imprime toda a lista de atendimentos a serem realizados ou que ainda estão sendo realizados, novamente usando o forrange para percorrer a lista de atendimentos. A função para exibir a Fila de espera, que são aqueles pacientes que ainda estão esperando para serem atendidos e a função apara exibir o relatório, que mostra o que já foi feito durante o tempo de execução do programa, um relatório mesmo.

Figura 11

```
void atualiza(Hospital *hospital)
{
    chrono::system_clock::time_point data_atual = chrono::system_clock::now(); // Pega a data atual do sistema

    for (int i = 0; i < hospital->atendimentos_em_execucao.size(); i++) // Finaliza os Atendimentos
    {
        Atendimento atendimento = hospital->atendimentos_em_execucao.at(i);
        chrono::system_clock::time_point data_termino_atendimento = chrono::system_clock::from_time_t(mktime(&atendimento.data_termino));
                // Transforma a data de termino para time_point para poder comparar as datas

        if (data_atual >= data_termino_atendimento)
        {
            atendimento.paciente.atendido = true;
            atendimento.medico->disponivel = true;
            hospital->medicos_disponiveis++;
            hospital->pacientes_tratados++;

            hospital->atendimentos_em_execucao.erase(hospital->atendimentos_em_execucao.begin() + i);
            cout << "Atendimento Do Paciente " << atendimento.paciente.nome << " foi finalizado." << endl;
            cout << "Médico " << atendimento.medico->nome << " está disponível para atendimento." << endl;
            system("pause");
        }
    }

    while (hospital->medicos_disponiveis > 0 && !verificaFilaVazia(&hospital->fila_de_espera)) // Inicia os Atendimentos
    {
        for (Medico &medico : hospital->medicos)
        {
            if (medico.disponivel)
            {
                Atendimento atendimento;
                desenfileira(&hospital->fila_de_espera, &atendimento);
                medico.disponivel = false;
                atendimento.medico = &medico;
                hospital->medicos_disponiveis--;

                time_t data_atual = chrono::system_clock::to_time_t(chrono::system_clock::now());
                atendimento.data_inicio = *localtime(&data_atual); // Data de inicio recebe a data atual na forma da struct tm
                unsigned int tempo_atendimento;

                switch (atendimento.paciente.prioridade) // Calcula o tempo de atendimento baseano na urgencia
                {
                case 0:
                    tempo_atendimento = 50 + (rand() % 21) - 10;
                    break;

                case 1:
                    tempo_atendimento = 20 + (rand() % 11) - 5;
                    break;

                case 2:
                    tempo_atendimento = 15 + (rand() % 11) - 5;
                    break;

                case 3:
                    tempo_atendimento = 10 + (rand() % 5) - 2;
                    break;

                case 4:
                    tempo_atendimento = 5 + (rand() % 7) - 3;
                    break;
                }

                time_t data_termino = data_atual + (60 * tempo_atendimento);
                atendimento.data_termino = *localtime(&data_termino);

                hospital->atendimentos_em_execucao.push_back(atendimento); // Adiciona o atendimento na lista de atendimentos em execução
                system("cls");
                cout << "Paciente: " << atendimento.paciente.nome << endl;
                cout << "Comparecer ao consultório do Médico: " << medico.nome << endl;
                system("pause");
                break;
            }
        }
    }
}
```

A princípio é criado uma variável para pegar o horário do sistema, por meio da biblioteca chrono, depois ele vai para um for começando em 0, e indo até a quantidade de atendimentos em execução no hospital. Uma variável atendimento será usada para recolher os atendimentos da lista de atendimentos. Na linha abaixo, temos uma variável para pegar o horário de término previsto no momento em que o paciente foi inserido no atendimento, e convertido para o mesmo tipo de variável da variável

usada para pegar o horário do sistema. Se esse horário já tiver sido excedido, significa que o paciente já foi atendido, logo o médico já está disponível para um novo atendimento. Abaixo temos um while, para pegar os pacientes que estão na lista de espera, e inseri-los em um atendimento. Primeiro será verificado qual médico está disponível para atendimento, e logo em seguida, esse médico será direcionado para o atendimento. Depois é pego o horário do sistema de novo, usado como data de início do atendimento, e para calcular o tempo de atendimento junto com a variação, de acordo com a prioridade do paciente, e assim calcular a data de término do atendimento. E assim insere o atendimento de forma definitiva na lista de atendimentos.

3. CONCLUSÃO

Ao decorrer do trabalho, mesmo tendo alguns erros devido ao fato de utilizarmos diversos conceitos aprendidos recentemente, consegui realizar o trabalho, concluindo todos os objetivos que foram propostos no início do trabalho, já que o código está rodando sem erros e de maneira funcional de acordo com o que foi proposto.

Ao longo dos trabalhos que tenho feito tenho aplicado bastante coisas que aprendi no 1° período como a modularização. Apliquei modularização separando o trabalho por arquivos, à título de organização por conselho de colegas de classe, dividindo em 6 arquivos, 1 arquivo *.cpp para a fila e seu arquivo cabeça(*.hpp), 1 arquivo *.cpp para o sistema e seu arquivo cabeça(*.hpp) e as main's (*.cpp e *.exe).

Tenho dificuldade em Algoritmos e Estruturas de Dados, como sempre falo e com ajuda de colegas eu consigo desenvolver o trabalho tranquilamente, e o desenvolvimento é sempre bom, porém gostaria de melhorar isso mais, e não precisar de alguém pra ajudar, creio que até ano que vem eu aprenda bem mais. Estou satisfeita com o resultado do programa, com o funcionamento, minhas expectativas foram atendidas, é sempre bom ver o que é dito em sala de aula aplicada na prática.

4. REFERÊNCIAS

TRINDADE. Eduardo. Algoritmos e Estrutura de Dados – Fila com ponteiro. 2022. Apresentação PDF. Disponível em: https://ead.ifmg.edu.br/saojoaoevangelista/pluginfile.php/149127/mod_resource/content/1/Aula%2011%20-%20Fila%20com%20Ponteiro.pdf. Acesso em: 18 de novembro de 2022.

TRINDADE. Eduardo. Algoritmos e Estrutura de Dados – Data, hora e tempo em C++. 2022. Apresentação PDF. Disponível em: https://ead.ifmg.edu.br/saojoaoevangelista/pluginfile.php/149625/mod_resource/content/1/Aula%2013%20-%20Data%2C%20hora%20e%20tempo%20em%20C%2B%2B.pdf . Acesso em: 18 de novembro de 2022.

TRINDADE. Eduardo. Algoritmos e Estrutura de Dados – Classe Vector. 2022. Apresentação PDF. Disponível em: https://ead.ifmg.edu.br/saojoaoevangelista/pluginfile.php/149748/mod_resource/content/2/Aula%2014%20-%20Classe%20Vector.pdf . Acesso em: 18 de novembro de 2022.

Meu código no GitHub: https://github.com/RhannaQuadros/Algoritmos-e-Estruturas-de-Dados-I-

5. APÊNDICES

5.1. Apêndice A – TADs de Fila(*cpp)

```cpp
#include "FilaPrioridadeDinamica.hpp"

using namespace std;

void inicializaFila(FilaPrioridadeDinamica *fila)
{
    fila->inicio = NULL;
    fila->tamanho = 0;
}

bool verificaFilaVazia(FilaPrioridadeDinamica *fila)
{
    return (fila->inicio == NULL);
}

void enfileira(FilaPrioridadeDinamica *fila, Atendimento item)
{
    Apontador novo = new Celula;
    novo->item = item;
    novo->prox = NULL;

    Apontador anterior = fila->inicio;
    Apontador atual = fila->inicio;

    if (verificaFilaVazia(fila))
    {
        fila->inicio = novo;
    }
    else
    {
        while (atual != NULL && atual->item.paciente.prioridade <= item.paciente.prioridade)
        {
            anterior = atual;
            atual = atual->prox;
        }

        if (atual == fila->inicio)
        {
            fila->inicio = novo;
            novo->prox = atual;
        }
        else
        {
            anterior->prox = novo;
            novo->prox = atual;
        }
    }

    fila->tamanho++;
}

void desenfileira(FilaPrioridadeDinamica *fila, Atendimento *item)
{
    if (verificaFilaVazia(fila))
    {
        cout << "Fila vazia!" << endl;
        return;
    }
    Apontador aux = fila->inicio;
    *item = aux->item;
    fila->inicio = aux->prox;
    delete aux;
    fila->tamanho--;
}
```

```
65
66  void esvaziaFila(FilaPrioridadeDinamica *fila)
67  {
68      Atendimento item;
69      while (!verificaFilaVazia(fila))
70      {
71          desenfileira(fila, &item);
72      }
73  }
74
75  void imprimeFila(FilaPrioridadeDinamica *fila)
76  {
77      if (verificaFilaVazia(fila))
78      {
79          cout << "Fila vazia!" << endl;
80          return;
81      }
82      Apontador aux = fila->inicio;
83      while (aux != NULL)
84      {
85          cout << "Paciente: " << aux->item.paciente.nome << endl;
86          mostraUrgencia(aux->item.paciente);
87          cout << "Endereço: " << aux->item.paciente.endereco << endl;
88          cout << "Idade: " << aux->item.paciente.idade << endl;
89          cout << "Sexo: " << (aux->item.paciente.sexo == 1 ? "Masculino" : "Feminino") << endl;
90          cout << "\n";
91          aux = aux->prox;
92      }
93      cout << endl;
94  }
95
96  void mostraUrgencia(Paciente paciente)
97  {
98      string urgencia;
99      switch (paciente.prioridade)
100     {
101     case 0:
102         urgencia = "Vermelho";
103         break;
104
105     case 1:
106         urgencia = "Laranja";
107         break;
108
109     case 2:
110         urgencia = "Amarelo";
111         break;
112
113     case 3:
114         urgencia = "Verde";
115         break;
116
117     case 4:
118         urgencia = "Azul";
119         break;
120     }
121
122     cout << "Urgência: " << urgencia << endl;
123 }
```

5.2. Apêndice B –TADs de Fila(*hpp)

```cpp
#ifndef FILA_PRIORIDADE_DINAMICA
#define FILA_PRIORIDADE_DINAMICA

#include <iostream>
#include <chrono>

using namespace std;

You, há 3 dias | 1 author (You)
typedef struct Triagem
{
    char respostas[18] = {'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N'};
};

You, há 3 dias | 1 author (You)
typedef struct Paciente
{
    string nome;
    string endereco;
    unsigned int idade;
    unsigned int sexo;
    Triagem triagem;
    unsigned int prioridade;
    bool atendido;
};

You, há 3 dias | 1 author (You)
typedef struct Medico
{
    string nome;
    string especialidade;
    string crm;
    bool disponivel = false;
};

You, há 3 dias | 1 author (You)
typedef struct Atendimento
{
    Medico *medico;
    Paciente paciente;
    tm data_inicio;
    tm data_termino;
};

typedef struct Celula *Apontador;

You, há 3 dias | 1 author (You)
typedef struct Celula
{
    Atendimento item;
    Apontador prox;
};

You, há 3 dias | 1 author (You)
typedef struct FilaPrioridadeDinamica
{
    Apontador inicio;
    int tamanho;
};

// Função do Paciente
void mostraUrgencia(Paciente paciente);

// Funções da fila
void inicializaFila(FilaPrioridadeDinamica *fila);
bool verificaFilaVazia(FilaPrioridadeDinamica *fila);
void enfileira(FilaPrioridadeDinamica *fila, Atendimento item);
void desenfileira(FilaPrioridadeDinamica *fila, Atendimento *item);
void esvaziaFila(FilaPrioridadeDinamica *fila);
void imprimeFila(FilaPrioridadeDinamica *fila);

#endif
```

5.3. Apêndice C – Sistema (*.hpp)

```hpp
#ifndef SISTEMA_H
#define SISTEMA_H

#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
#include <windows.h>
#include <time.h>
#include "FilaPrioridadeDinamica.cpp"

using namespace std;

#define OPCAO_SAIDA 6

You, há 3 dias | 1 author (You)
typedef struct Hospital
{
    Medico medicos[5];
    vector<Atendimento> atendimentos_em_execucao;
    FilaPrioridadeDinamica fila_de_espera;
    unsigned int medicos_registrados = 0;
    unsigned int medicos_disponiveis = 0;
    unsigned int pacientes_tratados = 0;
};

void cadastraMedicos(Hospital *hospital);
void menu();
void novoAtendimento(Hospital *hospital);
Paciente cadastraPaciente();
void realizaTriagem(Paciente *paciente);
void mostraUrgencia(Paciente paciente);
void mostraPrevisaoAtendimento(Paciente paciente);
void atualiza(Hospital *hospital);
void exibeAtendimentos(Hospital hospital);
void exibeRelatorio(Hospital hospital);
#endif
```

5.4. Apêndice D – Main.cpp

```cpp
#include "sistema.cpp"

int main(int argc, char const *argv[])
{
    UINT UTF8 = 65001;
    SetConsoleOutputCP(UTF8);

    srand(time(NULL));

    Hospital hospital;
    cadastraMedicos(&hospital);
    inicializaFila(&hospital.fila_de_espera);

    int opcao;
    do
    {
        menu();
        cin >> opcao;
        cin.ignore();

        system("cls");

        switch (opcao)
        {
        case 1:
            novoAtendimento(&hospital);
            break;

        case 2:
            atualiza(&hospital);
            break;

        case 3:
            exibeAtendimentos(hospital);
            break;

        case 4:
            exibeFilaEspera(hospital);
            break;

        case 5:
            exibeRelatorio(hospital);
            break;
        }
    } while (opcao != OPCAO_SAIDA);

    return 0;
}
```