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

EDUARDA LUIZA MARTINS DE OLIVEIRA

TRABALHO PRÁTICO III

SÃO JOÃO EVANGELISTA
2022

EDUARDA LUIZA MARTINS DE OLIVEIRA

UNIDADE DE PRONTO ATENDIMENTO DE UM HOSPITAL

SÃO JOÃO EVANGELISTA
2022

SUMÁRIO.....

1.INTRODUÇÃO

O trabalho desenvolvido aqui é um sistema para uma Unidade de Pronto Atendimento (UPA) de um Hospital que irá atender pacientes de acordo com a sua triagem. Neste trabalho simulamos os procedimentos de classificação de risco para definir as prioridades nos atendimentos a pacientes (fila de espera). A prioridade é definida pelas cores vermelho, laranja, amarelo, verde e azul. Importante destacar que a linguagem utilizada será C++ e o objetivo é fixar conceitos sobre estruturas de dados, utilizar a Fila Dinâmica, Tipos Abstratos de Dados (TAD) e usar o raciocínio lógico para resoluções de problemas.

Ao decorrer do trabalho nota-se a funcionalidade de cada função e a importância do mínimo domínio dessas matérias citadas acima para resoluções de problemas que muitas das vezes é difícil de não acontecer, porém com devida persistência é possível o desenvolvimento desse Trabalho Prático finalizado com êxito e com suas funcionalidades em perfeito funcionamento.

O problema que foi proposto da Unidade de Pronto Atendimento de um hospital, é um exemplo perfeito para utilização dessas matérias propostas nesse primeiro momento, é muito importante entender a funcionalidade de cada passo realizado nesse código, de extrema importância também entender conceitos básicos de como funciona um pronto atendimento de um hospital para aplicar isso dentro do desenvolvimento do código.

1.1. Objetivo Geral

Este trabalho tem como objetivo geral a implementação de uma Unidade de Pronto Atendimento de um Hospital, com objetivo de trabalhar com inclusão de uma fila de espera, de acordo com as prioridades de urgências ou emergências de cada pacientes, para um controle do hospital.

1.2. Objetivos Específicos

Esse trabalho tem como objetivos específicos:

- Definir prioridades nos atendimentos a pacientes (fila de espera).
- Incluir a prioridade dos pacientes pelas cores vermelho (emergência), laranja (muito urgente), amarelo (urgente), verde (pouco urgente) e azul (não urgente).

- Cada prioridade pelas cores existe um tempo em horas e minutos para atender os pacientes.
- Utilização de TADs para controle da Fila Dinâmica.
- Apresentar a manipulação da disponibilidade dos médicos, tendo em vista que existe 5 médicos cadastrados, onde irá ter um tempo para atender os pacientes devido cada caso de emergência ou urgência. Assim que terminarem de atender um paciente irá atender os outros que estão na fila de espera.
- Apresentar também a funcionalidade de um hospital.
- Exibir um novo atendimento.
- Exibir a fila de espera.
- Exibir relatórios.
- Atualizar os atendimentos.
- Criar uma estrutura de fila.
- Inserir um elemento no fim.
- Retirar o elemento do início.
- Verificar se a fila está vazia.
- Liberar fila.

1.3. Justificativa

Esse trabalho é justificado devido a necessidade de uma Unidade de Pronto Atendimento de um hospital, onde podemos definir as prioridades nos atendimentos dos pacientes de acordo com as cores (vermelho, laranja, amarelo, verde e azul). Então com a implementação desse código facilitaria então esse controle, deixando tudo então da maneira mais simples para verificar o nível de prioridade de cada paciente, gerando também a organização do hospital, contendo também o controle de cada tempo que o médico irá atender os pacientes.

2. DESENVOLVIMENTO

A seguir será apresentado o que foi utilizado para desenvolvimento desse trabalho, como Fila Dinâmica, TADS e também funções para que o trabalho ficasse funcional e usual na Unidade de Pronto Atendimento de um Hospital.

2.1. Fila Dinâmica

Figura 1

A partir daí, todos os pacientes que chegarem à fila serão posicionados ao final dela.

Figura 2

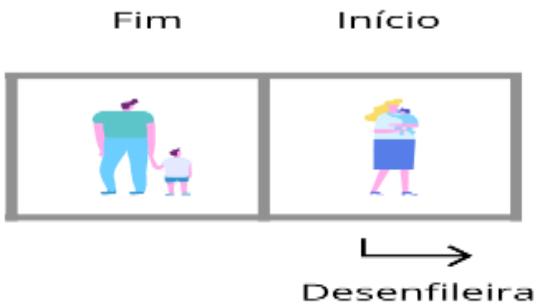
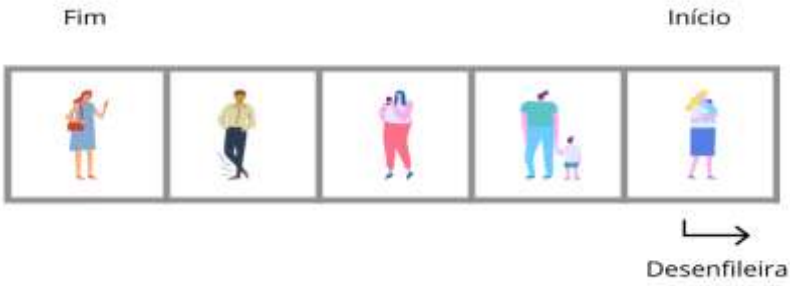


Figura 3

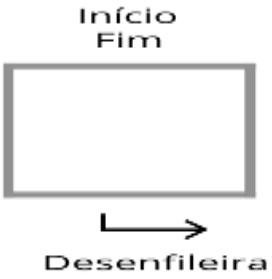


Sempre que o paciente for chamado para ser atendido, este é removido da fila e o paciente que estava na próxima posição passará a ser o início da fila.

Figura 4

Este processo se repete até que a fila esteja vazia e nenhum usuário tenha que ser atendido.

Figura 5



As figuras acima apresentam como funciona uma Fila Dinâmica, estrutura foi muito utilizada no desenvolvimento desse trabalho. Como apresentado acima podemos ter uma noção de como funciona a estrutura de dados da fila dinâmica, para iniciar essa explicação temos que ter em mente que toda fila é uma estrutura do tipo FIFO (First In First Out), ou seja, o primeiro elemento inserido será o primeiro a ser removido. Cada elemento da estrutura pode armazenar um ou vários dados e um ponteiro para o próximo elemento, o que permite o encadeamento e mantêm uma estrutura linear. A estrutura do tipo fila possui um ponteiro denominado INICIO, onde todas operações de remoção acontecem e outro denominado FIM, onde acontecem as inserções.

2.2. Tipos Abstratos de Dados (TADs)

As TADs são modelos matemáticos de estruturas de dados que definem o tipo de dados a ser armazenado, as operações possíveis sobre estes dados e o tipo de dados das operações. A TAD tem o papel de definir o que se pode fazer com uma estrutura de dados, que é o tipo de informação ou conjunto das mesmas que pode ser armazenada em uma variável, tem também as funções de acesso dos dados e quais os parâmetros para estas funções e quais os retornos estas funções geram quando se acessam os dados.

2.3. Ponteiro

Ponteiro também conhecido como apontador é uma variável que armazena o endereço de memória, e o interessante que ele pode apontar para um endereço inválido, que contém um valor NULL, nesse Trabalho Prático foi muito utilizado esse conceito para implementação da fila dinâmica, a próxima posição do último sempre apontava para NULL, onde não tinha nem item ali, era realmente nada que ele apontava. Então sabemos que ponteiro é de extrema importância para realização de trabalhos utilizando a fila, pois é utilizado para percorrer a mesma, por isso ele é indispensável.

2.4. Implementação

Para o desenvolvimento desse trabalho foi utilizado diversas funções, que estão referenciadas nos Apêndices A e B no fim do documento, a seguir irei listar e explicar de maneira detalhada como elas funcionam:

- void inicializaFila(TFila *fila) – função padrão de uma TAD para inicializar a estrutura de fila.
- bool verificaFilaVazia(TFila *fila) – função padrão de uma TAD para verificar se a fila só foi inicializada e não foi inserido nada nela.
- void enfileira(TFila *fila, TAtendimento item) – função padrão de uma TAD alterada para o trabalho, que insere de acordo com a prioridade. Nela temos dois apontadores criados localmente, para guardar valores, o atual para percorrer a fila, enquanto o anterior guarda o valor anterior ao que será inserido.
- void desenfileira(TFila *fila, TAtendimento *item) – função padrão de uma TAD, que remove do início da fila.
- void esvaziaFila(TFila *fila) – função usada para executar, a limpeza total da fila, onde o desenfileira é executado até que a fila esteja vazia
- void imprimeFila(TFila *fila) – função padrão de uma TAD, onde é imprimido os itens da fila
- void mostraUrgencia(TPaciente paciente) – função do sistema usada para mostrar a urgência do caso de cada paciente por meio de uma string, de acordo com o valor da prioridade do paciente;

- void cadastraMedicos(Hospital *hospital) – função executada no inicio do código para cadastrar os médicos de forma automática, por meio da próxima função.
- void cadastraMedico(Hospital *hospital, string nome, string especialidade, string crm) – função que insere os médicos no hospital de fato, usando os valores dos parâmetros para atribui-los aos valores da struct médico.
- void novoAtendimento(Hospital *hospital) – nesta função, criamos uma atendimento atribuindo colocando um paciente nele, e mais abaixo adiciona o médico por meio da atualiza. Para isso foi criada uma lista por meio da biblioteca vector, temos a variavel novo_paciente, recebendo o retorno da próxima função que será apresentada, logo em seguida esse mesmo retorno é mandado para a triagem, e ele é colocado na lista vector, depois é mostrado o nível de urgencia visto em uma das funções anteriores, e depois a previsão de atendimento com base em uma das próximas funções. 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 for, percorrendo a lista de pacientes, e inserindo cada um deles na fila, e depois ele atualiza o que já foi feito antes, por meio de uma função apresentada logo a frente, isso se foi feito algo.
- TPaciente cadastraPaciente() – 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.
- void realizaTriagem(TPaciente *paciente) – função para realizar a triagem, um conjunto de 18 perguntas, que inicialmente possui todas as respostas inseridas como N, depois temos a variavel urgencia_encontrada, que 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, ele define o nível de urgência para cada caso, nas primeiras 5 perguntas, se houver pelo menos um S a urgência é vermelha, e ele já pula para o atendimento, se nas primeiras 10 se houver pelo menos um S a urgencia é laranja, ele já pula direto pro atendimento, se nas primeiras 15, se houver pelo menos um S a urgencia é Amarela, se dentro das 18, se houver pelo menos um S a urgencia é verde, e se a urgencia não foi encontrada, então a urgência é Azul.

- void mostraPrevisaoAtendimento(TPaciente paciente) – mostra a previsão de atendimento de acordo com a urgência encontrada na última função apresentada, essa função só vai retorna a mensagem falando qual o nível de urgência.
- void atualiza(Hospital *hospital) – função que chama a finalizaAtendimentos(Hospital *hospital, chrono::system_clock::time_point data_atual) e bool iniciaAtendimentos(Hospital *hospital), para que sejam finalizados os atendimentos, e iniciados os atendimentos que estão na fila de espera.
- bool finalizaAtendimentos(Hospital *hospital, chrono::system_clock::time_point data_atual) – a função cria uma variável booleana, para ser manipulada dentro do for, e dentro dele 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, e logo em seguida, é retornado a mensagem de finalização do atendimento.
- bool iniciaAtendimentos(Hospital *hospital) – na função temos uma variável boolena, para ser manipulada no while, e dentro dele temos um for, para percorrer os médicos e achar 1 que esteja disponível, para atender os pacientes que estão na lista de espera, estes que serão removidos de lá, e inseridos em um atendimento. Depois é pego o horário do sistema, que será 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, e retorna a mensagem de confirmação.

- void exibeAtendimentos(Hospital hospital) – a função exibe atendimento, imprime toda a lista de atendimentos que ainda estão sendo realizados, novamente usando o for para percorrer a lista de atendimentos.
- void exibeRelatorio(Hospital hospital) – a função para exibir o relatório, que mostra o que já foi feito durante o tempo de execução do programa, um relatório mesmo.
- void exibeFilaEspera(Hospital hospital) – a função para exibir a Fila de espera, que são aqueles pacientes que ainda estão esperando para serem atendidos, pois todos os médicos estão ocupados atendendo alguém.

3. CONCLUSÃO

A unidade de pronto atendimento é muito importante para todo hospital, e a implementação desse código também foi de extrema importância para desenvolvimento do controle de um hospital, um controle bem amplo sobre a quantidade médicos para atender os pacientes que estão na fila de espera de acordo com cada prioridade. Com e a utilização de códigos e tecnologia foi algo que realmente se deu muito bem com as soluções desse problema proposto, ou seja, serviu para cronometrar o tempo que cada médico gasta para atender os pacientes de acordo com cada prioridade através das cores.

Ao decorrer do desenvolvimento desse projeto confirmamos que o domínio dessas funcionalidades é realmente muito importante, tendo em vista que ouve muitos erros durante o desenvolvimento do projeto, e como estávamos utilizando diversos conceitos dentro da programação, alguém sem o devido conhecimento muito provavelmente não conseguiria solucionar estes problemas.

Ao fim desse trabalho, podemos concluir que todos os objetivos que foi proposto no trabalho foi alcançado de forma que o código está rodando sem erros e de maneira funcional de acordo com o que foi proposto no início do Trabalho Prático. Objetivos como apresentar a funcionalidade de um hospital, exibir um novo atendimento, atualizar os atendimentos, inserir um elemento no fim, retirar elemento do início, verificar se a fila está vazia, cadastro de médico, triagem, cadastro de paciente, saída de dados e também o registro de hospital (UPA).

Ao longo do trabalho encontrei dificuldades para interpretar o que estava sendo pedido, pedi ajuda a colegas de classe para me auxiliar na interpretação e na escrita do código do trabalho, utilizei os slides disponibilizados pelo professor para sanar dúvidas a respeito das TADs (Tipos Abstratos de Dados). Foi utilizado as informações da biblioteca, sites, e ajuda de colegas de sala para ajudar na conclusão do Trabalho Prático. Acho que consegui de forma sucinta apresentar conhecimentos em uma fila dinâmica. Estou feliz com o resultado, e ele atendeu às minhas expectativas.

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

TRINDADE. Eduardo. Algoritmos e Estrutura de Dados – Fila com Arranjo. 2022. Apresentação PDF. Disponível em: https://ead.ifmg.edu.br/saojoaoevangelista/pluginfile.php/147825/mod_resource/content/1/Aula%209%20-%20Fila.pdf

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

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

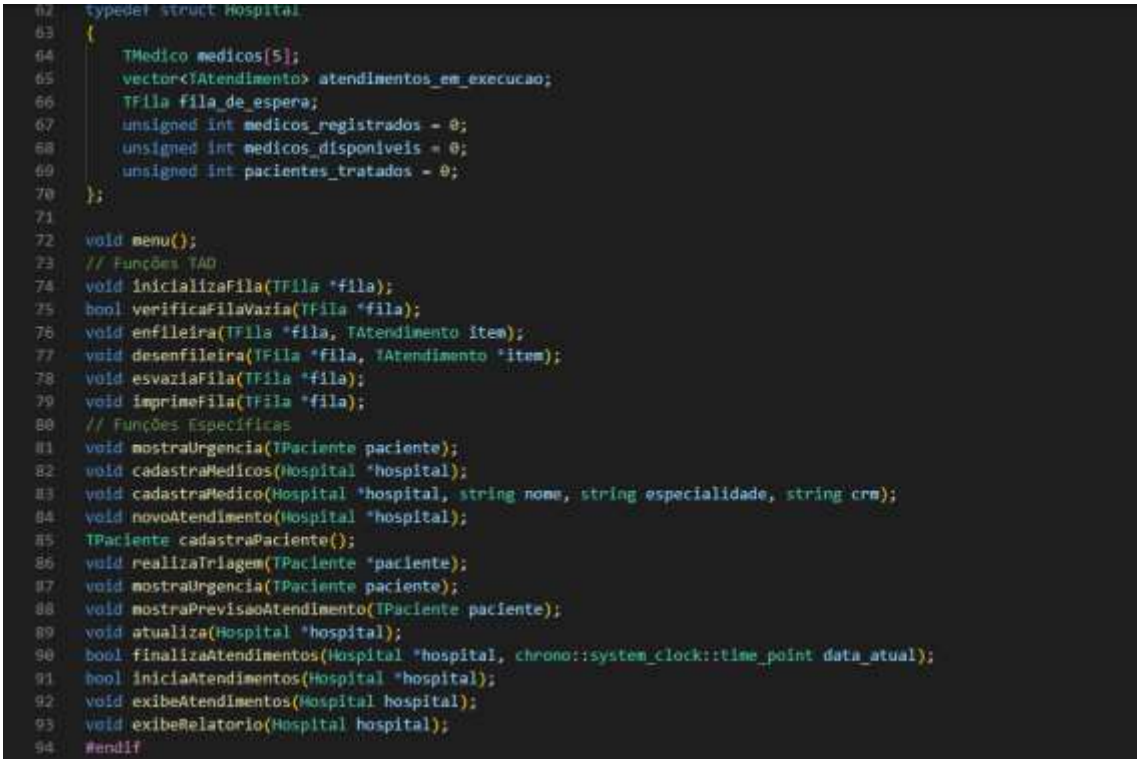
TRINDADE. Eduardo. Algoritmos e Estrutura de Dados – TAD Fila com Ponteiro. 2022. Apresentação PDF. Disponível em: https://github.com/edutrindade/Algoritmos-e-Estruturas-de-Dados-I/tree/master/10.%20Fila%20com%20Ponteiro/10.2%20TAD%20Fila%20com%20Prioridade

TRINDADE. Eduardo. Algoritmos e Estrutura de Dados – Bibliotecas de Tempo. 2022. Apresentação PDF. Disponível em: https://github.com/edutrindade/Algoritmos-e-Estruturas-de-Dados-I/tree/master/12.%20Bibliotecas%20de%20tempo

5. APÊNDICES

5.1. Apêndice A – TADs (*hpp)

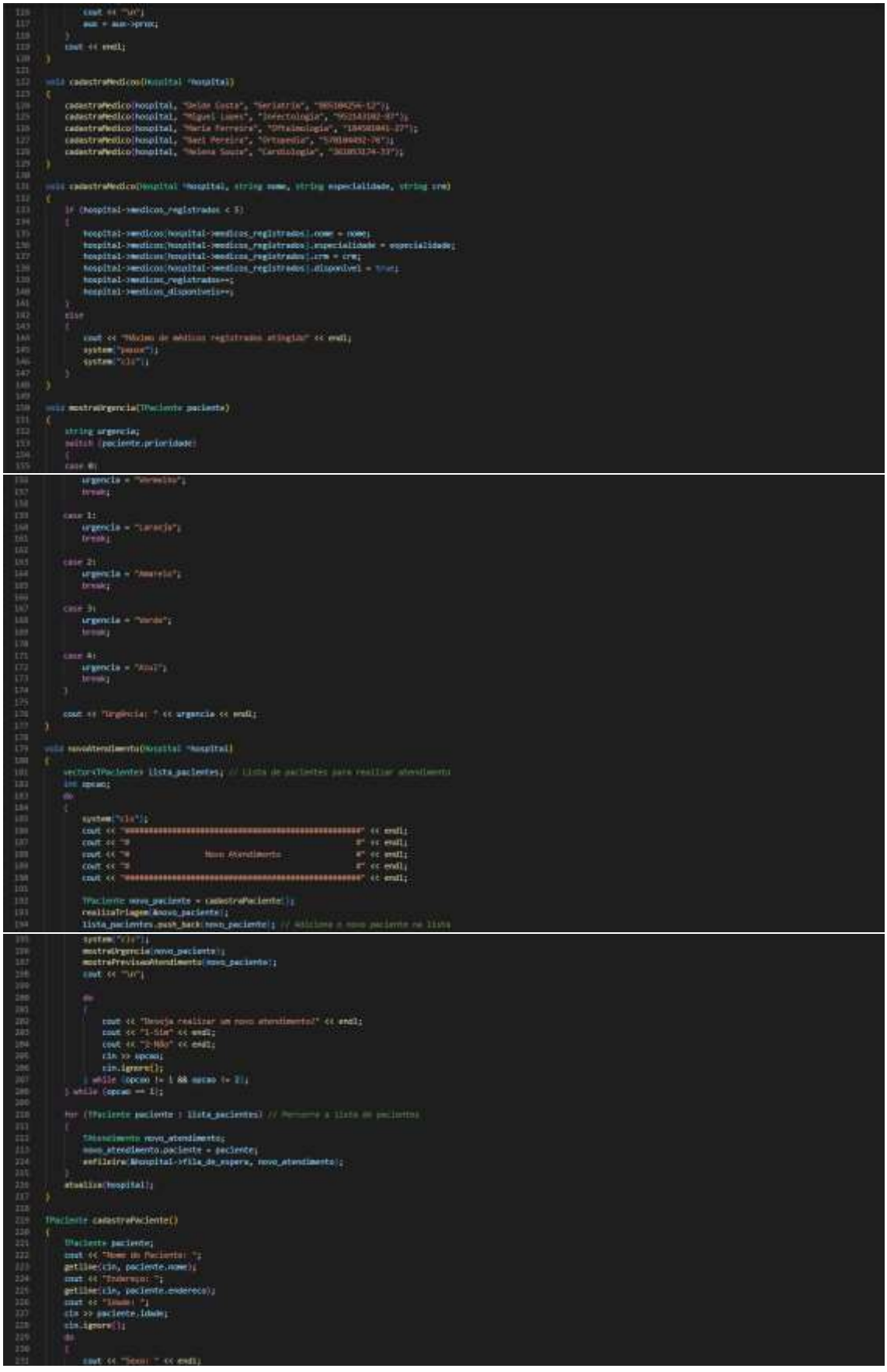
```cpp
#ifndef FUNCOES_H
#define FUNCOES_H

#include <iostream>
#include <chrono>
#include <iomanip>
#include <vector>
#include <string>
#include <windows.h>
#include <time.h>

#define OPCAO_SAIDA 6

using namespace std;

typedef struct
{
    char respostas[18];
} TTriagem;

typedef struct
{
    string nome;
    string endereco;
    unsigned int idade;
    unsigned int sexo;
    TTriagem triagem;
    unsigned int prioridade;
    bool atendido;
} TPaciente;

typedef struct
{
    string nome;
    string especialidade;
    string crm;
    bool disponivel = false;
} TMedico;

typedef struct TAtendimento
{
    TMedico *medico;
    TPaciente paciente;
    tm data_inicio;
    tm data_termino;
} TAtendimento;

typedef struct TCelula_str *TApontador;

typedef struct TCelula_str
{
    TAtendimento item;
    TApontador prox;
} TCelula;

typedef struct TFila
{
    TApontador inicio;
    int tamanho;
} TFila;

```



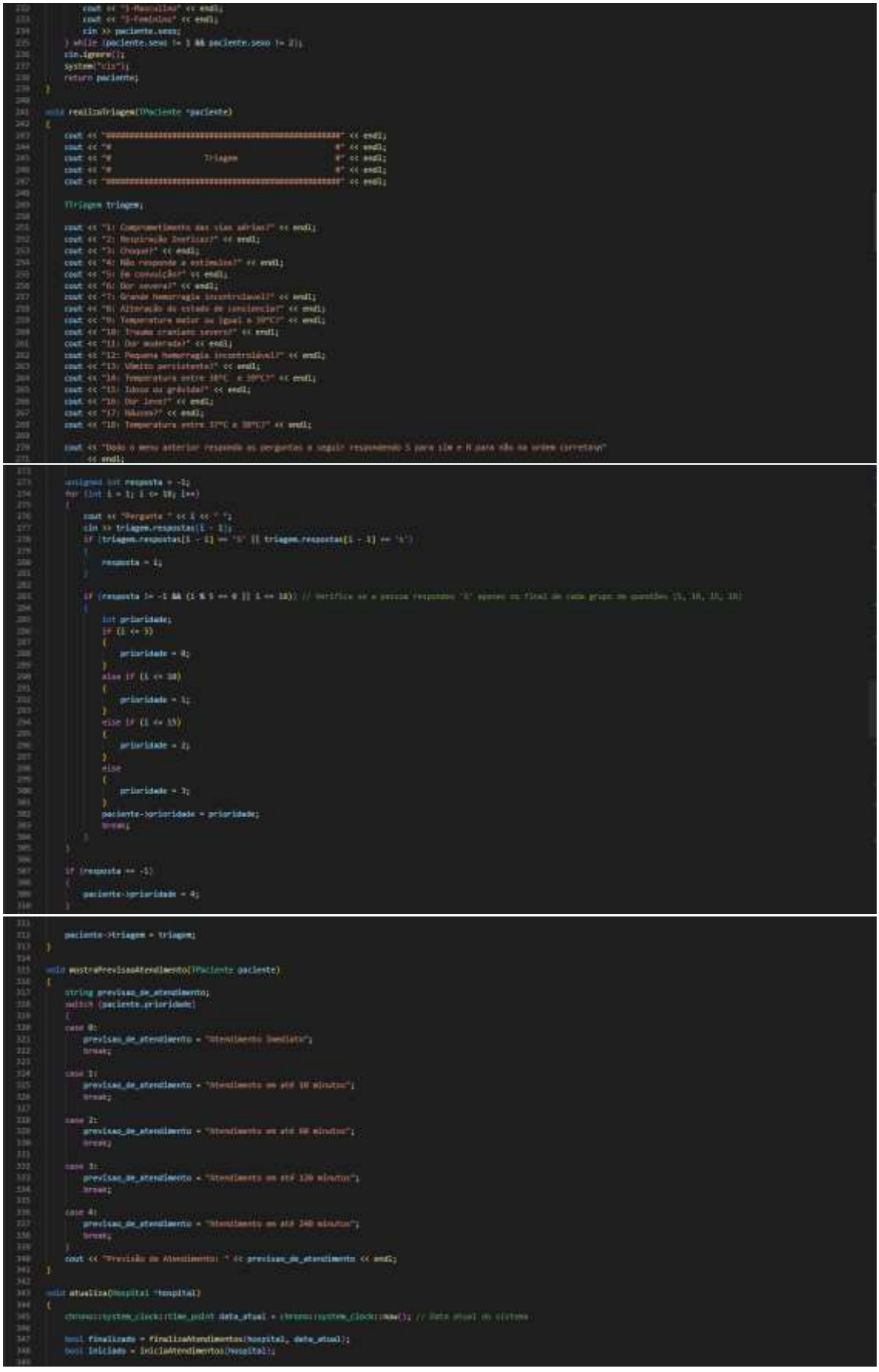
```
typedef struct Hospital
{
    TMedico medicos[5];
    vector<TAtendimento> atendimentos_em_execucao;
    TFila fila_de_espera;
    unsigned int medicos_registrados = 0;
    unsigned int medicos_disponiveis = 0;
    unsigned int pacientes_tratados = 0;
};

void menu();
// Funções TAD
void inicializaFila(TFila *fila);
bool verificaFilaVazia(TFila *fila);
void enfileira(TFila *fila, TAtendimento item);
void desenfileira(TFila *fila, TAtendimento *item);
void esvaziaFila(TFila *fila);
void imprimeFila(TFila *fila);
// Funções Específicas
void mostraUrgencia(TPaciente paciente);
void cadastraMedicos(Hospital *hospital);
void cadastraMedico(Hospital *hospital, string nome, string especialidade, string crm);
void novoAtendimento(Hospital *hospital);
TPaciente cadastraPaciente();
void realizaTriagem(TPaciente *paciente);
void mostraUrgencia(TPaciente paciente);
void mostraPrevisaoAtendimento(TPaciente paciente);
void atualiza(Hospital *hospital);
bool finalizaAtendimentos(Hospital *hospital, chrono::system_clock::time_point data_atual);
bool iniciaAtendimentos(Hospital *hospital);
void exibeAtendimentos(Hospital hospital);
void exibeRelatorio(Hospital hospital);
#endif
```

5.2. Apêndice B – TADs (*cpp)

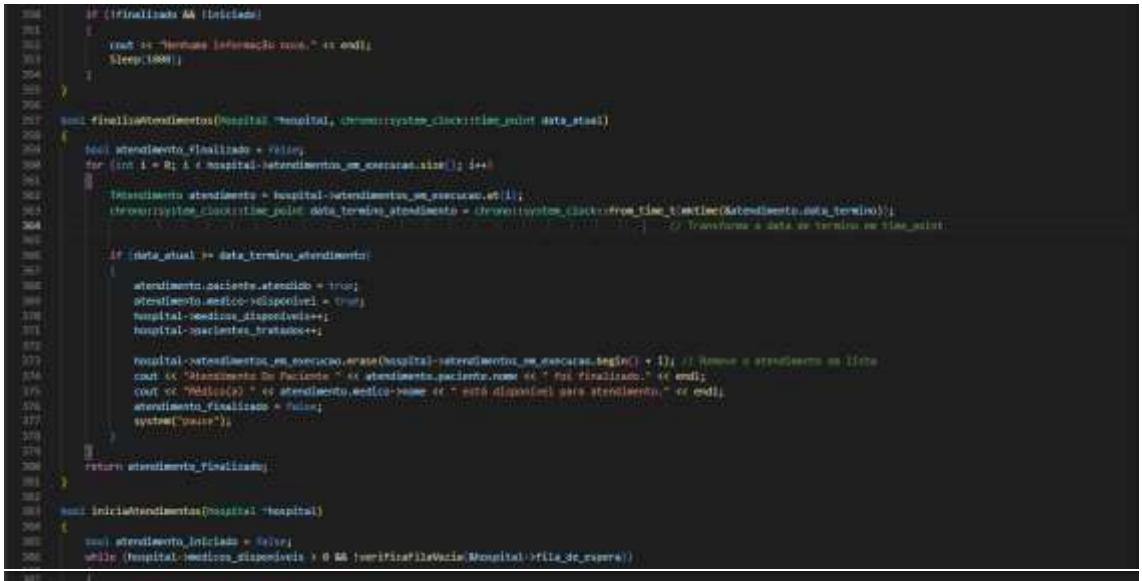
```cpp
#include "funcoes.hpp"

using namespace std;

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 Atendimentos  #" << endl;
    cout << "#                                        #" << endl;
    cout << "#             3 - Exibir Atendimentos    #" << endl;
    cout << "#                                        #" << endl;
    cout << "#             4 - Exibir Fila de Espera  #" << endl;
    cout << "#                                        #" << endl;
    cout << "#             5 - Exibir Relatório       #" << endl;
    cout << "#                                        #" << endl;
    cout << "#             6 - Sair                   #" << endl;
    cout << "#                                        #" << endl;
    cout << "##########################################" << endl;
}

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

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

void enfileira(TFila *fila, TAtendimento item)
{
    TApontador novo = new TCelula;
    novo->item = item;
    novo->prox = NULL;

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

    if (verificaFilaVazia(fila))
    {
        fila->inicio = novo;
    }
    else
    {
        while (atual != NULL && item.paciente.prioridade >= atual->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(TFila *fila, TAtendimento *item)
{
    if (verificaFilaVazia(fila))
    {
        cout << "Fila vazia!" << endl;
        return;
    }
    TApontador aux = fila->inicio;
    *item = aux->item;
    fila->inicio = aux->prox;
    delete aux;
    fila->tamanho--;
}

void esvaziaFila(TFila *fila)
{
    TAtendimento item;
    while (!verificaFilaVazia(fila))
    {
        desenfileira(fila, &item);
    }
}

void imprimeFila(TFila *fila)
{
    if (verificaFilaVazia(fila))
    {
        cout << "Fila vazia!" << endl;
        return;
    }
    TApontador aux = fila->inicio;
    cout << "Pacientes:" << endl;
    while (aux != NULL)
    {
        cout << "\tNome: " << aux->item.paciente.nome << endl;
        cout << "\t";
        mostraUrgencia(aux->item.paciente);
        cout << "\tEndereço: " << aux->item.paciente.endereco << endl;
        cout << "\tIdade: " << aux->item.paciente.idade << endl;
        cout << "\tSexo: " << (aux->item.paciente.sexo == 1 ? "Masculino" : "Feminino") << endl;
```



```
        cout << "\n";
        aux = aux->prox;
    }
    cout << endl;
}

void cadastraMedicos(Hospital *hospital)
{
    cadastraMedico(hospital, "Helen Costa", "Geriatria", "985104256-12");
    cadastraMedico(hospital, "Miguel Lopes", "Infectologia", "962143102-07");
    cadastraMedico(hospital, "Maria Ferreira", "Oftalmologia", "184501041-27");
    cadastraMedico(hospital, "Raul Pereira", "Ortopedia", "5/0104052-26");
    cadastraMedico(hospital, "Helena Sousa", "Cardiologia", "301053174-33");
}

void cadastraMedico(Hospital *hospital, string nome, string especialidade, string crm)
{
    if (hospital->medicos_registrados < 5)
    {
        hospital->medicos[hospital->medicos_registrados].nome = nome;
        hospital->medicos[hospital->medicos_registrados].especialidade = especialidade;
        hospital->medicos[hospital->medicos_registrados].crm = crm;
        hospital->medicos[hospital->medicos_registrados].disponivel = true;
        hospital->medicos_registrados++;
        hospital->medicos_disponiveis++;
    }
    else
    {
        cout << "Número de médicos registrados atingido" << endl;
        system("pause");
        system("cls");
    }
}

void mostraUrgencia(TPaciente paciente)
{
    string urgencia;
    switch (paciente.prioridade)
    {
    case 0:
        urgencia = "Vermelho";
        break;

    case 1:
        urgencia = "Laranja";
        break;

    case 2:
        urgencia = "Amarelo";
        break;

    case 3:
        urgencia = "Verde";
        break;

    case 4:
        urgencia = "Azul";
        break;
    }

    cout << "Urgência: " << urgencia << endl;
}

void novoAtendimento(Hospital *hospital)
{
    vector<TPaciente> lista_pacientes; // Lista de pacientes para realizar atendimento
    int opcao;
    do
    {
        system("cls");
        cout << "=============================================" << endl;
        cout << "#                                           #" << endl;
        cout << "#               Novo Atendimento            #" << endl;
        cout << "#                                           #" << endl;
        cout << "=============================================" << endl;

        TPaciente novo_paciente = cadastraPaciente();
        realizaTriagem(&novo_paciente);
        lista_pacientes.push_back(novo_paciente); // Adiciona o novo paciente na lista
        system("cls");
        mostraUrgencia(novo_paciente);
        mostraPrevisaoAtendimento(novo_paciente);
        cout << "\n";

        do
        {
            cout << "Deseja realizar um novo atendimento?" << endl;
            cout << "1-Sim" << endl;
            cout << "2-Não" << endl;
            cin >> opcao;
            cin.ignore();
        } while (opcao != 1 && opcao != 2);
    } while (opcao == 1);

    for (TPaciente paciente : lista_pacientes) // Percorre a lista de pacientes
    {
        TAtendimento novo_atendimento;
        novo_atendimento.paciente = paciente;
        enfileira(&hospital->fila_de_espera, novo_atendimento);
    }
    atualiza(hospital);
}

TPaciente cadastraPaciente()
{
    TPaciente 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;
```

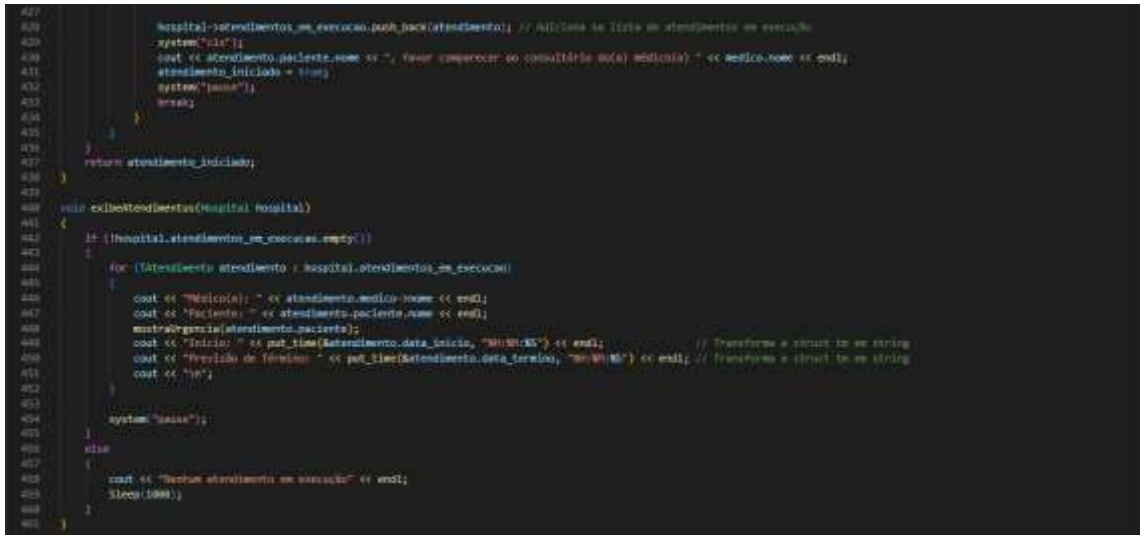


```
        cout << "1-Masculino" << endl;
        cout << "2-Feminino" << endl;
        cin >> paciente.sexo;
    } while (paciente.sexo != 1 && paciente.sexo != 2);
    cin.ignore();
    system("cls");
    return paciente;
}

void realizaTriagem(TPaciente *paciente)
{
    cout << "=============================================" << endl;
    cout << "#                                           #" << endl;
    cout << "#                  Triagem                  #" << endl;
    cout << "#                                           #" << endl;
    cout << "=============================================" << endl;

    TTriagem triagem;

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

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

    unsigned int resposta = -1;
    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')
        {
            resposta = i;
        }

        if (resposta != -1 && (i % 5 == 0 || i == 18)) // Verifica se a pessoa respondeu 'S' apenas no final de cada grupo de questões (5, 10, 15, 18)
        {
            int prioridade;
            if (i <= 5)
            {
                prioridade = 0;
            }
            else if (i <= 10)
            {
                prioridade = 1;
            }
            else if (i <= 15)
            {
                prioridade = 2;
            }
            else
            {
                prioridade = 3;
            }
            paciente->prioridade = prioridade;
            break;
        }
    }

    if (resposta == -1)
    {
        paciente->prioridade = 4;
    }

    paciente->triagem = triagem;
}

void mostraPrevisaoAtendimento(TPaciente paciente)
{
    string previsao_de_atendimento;
    switch (paciente.prioridade)
    {
    case 0:
        previsao_de_atendimento = "Atendimento Imediato";
        break;

    case 1:
        previsao_de_atendimento = "Atendimento em até 10 minutos";
        break;

    case 2:
        previsao_de_atendimento = "Atendimento em até 60 minutos";
        break;

    case 3:
        previsao_de_atendimento = "Atendimento em até 120 minutos";
        break;

    case 4:
        previsao_de_atendimento = "Atendimento em até 240 minutos";
        break;
    }
    cout << "Previsão de Atendimento: " << previsao_de_atendimento << endl;
}

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

    bool finalizado = finalizaAtendimentos(hospital, data_atual);
    bool iniciado = inicioAtendimentos(hospital);
```

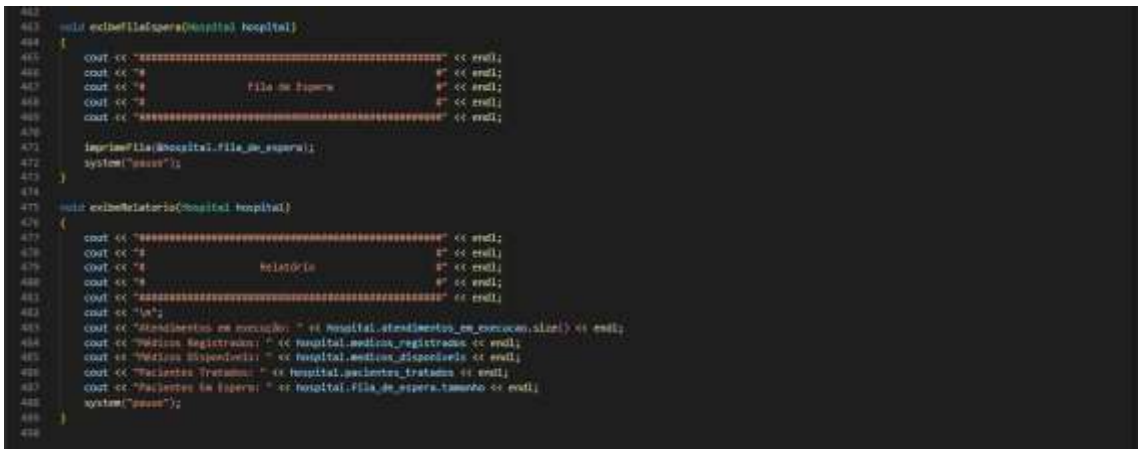


```
    if (!finalizado && !iniciado)
    {
        cout << "Nenhuma informação nova." << endl;
        Sleep(1000);
    }
}

bool finalizaAtendimentos(Hospital *hospital, chrono::system_clock::time_point data_atual)
{
    bool atendimento_finalizado = false;
    for (int i = 0; i < hospital->atendimentos_em_execucao.size(); i++)
    {
        TAtendimento 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 término em time_point

        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); // Remove o atendimento da lista
            cout << "Atendimento Do Paciente " << atendimento.paciente.nome << " foi finalizado." << endl;
            cout << "Médico(a) " << atendimento.medico->nome << " está disponível para atendimento." << endl;
            atendimento_finalizado = false;
            system("pause");
        }
    }
    return atendimento_finalizado;
}

bool iniciaAtendimentos(Hospital *hospital)
{
    bool atendimento_iniciado = false;
    while (hospital->medicos_disponiveis > 0 && !verificaFilaVazia(&hospital->fila_de_espera))
```

```
    {
        for (TMedico &medico : hospital->medicos)
        {
            if (medico.disponivel)
            {
                TAtendimento 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 início recebe a data atual na forma de struct tm
                unsigned int tempo_atendimento;

                switch (atendimento.paciente.prioridade) // Calcula o tempo de atendimento com base na prioridade
                {
                case 0:
                    tempo_atendimento = 30 + (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); // Transformando o tempo de atendimento em minutos e adicionando à hora atual
                atendimento.data_termino = *localtime(&data_termino);
```



```
                hospital->atendimentos_em_execucao.push_back(atendimento); // Adiciona na lista de atendimentos em execução
                system("cls");
                cout << atendimento.paciente.nome << ", favor comparecer ao consultório do(a) médico(a) " << medico.nome << endl;
                atendimento_iniciado = true;
                system("pause");
                break;
            }
        }
    }
    return atendimento_iniciado;
}

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

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



```
463
464 void exibeFilaEspera(Hospital hospital)
465 {
466     cout << "##############################################" << endl;
467     cout << "#                                            #" << endl;
468     cout << "#               Fila de Espera               #" << endl;
469     cout << "#                                            #" << endl;
470     cout << "##############################################" << endl;
471
472     imprimeFila(&hospital.fila_de_espera);
473     system("pause");
474 }
475
476 void exibeRelatorio(Hospital hospital)
477 {
478     cout << "##############################################" << endl;
479     cout << "#                                            #" << endl;
480     cout << "#                 Relatório                  #" << endl;
481     cout << "#                                            #" << endl;
482     cout << "##############################################" << endl;
483     cout << "\n";
484     cout << "Atendimentos em execução: " << hospital.atendimentos_em_execucao.size() << endl;
485     cout << "Médicos Registrados: " << hospital.medicos_registrados << endl;
486     cout << "Médicos Disponíveis: " << hospital.medicos_disponiveis << endl;
487     cout << "Pacientes Tratados: " << hospital.pacientes_tratados << endl;
488     cout << "Pacientes Em Espera: " << hospital.fila_de_espera.tamanho << endl;
489     system("pause");
490 }
491
```