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

GABRIEL KÁICON BATISTA HILÁRIO

TRABALHO PRÁTICO III

SÃO JOÃO EVANGELISTA
NOVEMBRO - 2022

GABRIEL KÁICON BATISTA HILÁRIO

SISTEMA PARA UMA UNIDADE DE PRONTO ATENDIMENTO(UPA)

SÃO JOÃO EVANGELISTA

NOVEMBRO - 2022

SUMÁRIO

1. INTRODUÇÃO

Este trabalho prático foi documentado para que seja avaliado em conjunto com os códigos na linguagem C/C++, exigido pelo docente Eduardo Augusto da Costa Trindade, dentro da disciplina de Algoritmos e Estruturas de Dados I, ministrada pelo mesmo. Porém a documentação tem cunho expositivo, onde é descrito as funcionalidades do programa, com testes, e desenvolvimento de novas linhas de raciocínio lógico 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 de prioridade com ponteiro, Tipos Abstratos de Dados (TADs), manipulação de tempo, e o uso de bibliotecas, utilizando a linguagem de C++ para escrita dos códigos.

1.2. Objetivos Específicos

Esse trabalho tem como objetivos específicos:

- Apresentar conhecimentos de Fila;
- Apresentar os conhecimentos da biblioteca chrono;
- Apresentar e aprimorar os conhecimentos da biblioteca time.h;
- Apresentar os conhecimentos da biblioteca vector;
- Desenvolver novas linhas de raciocínio para outros tipos de lógicas de aplicativos.

1.3. Justificativa

Ao iniciar os estudos de Algoritmos e Estruturas de Dados I, vemos os conteúdos de Ponteiros, TADs, e estruturas de dados, porém a que nos interessa nesse momento é a fila de prioridade com ponteiro.

Vemos em ponteiros, a manipulação de valores da variável por meio do endereço de memória, utilizando ponteiros. Por último vimos o conteúdo de fila, que consiste na inserção de itens em uma lista, só que ao invés de removermos de várias posições, iremos remover apenas do início, pois no conceito de fila, aquele que chega primeiro, é o primeiro a sair. Temos o conceito de fila, fila de prioridade, e fila circular, dentre outras, e todas essas citadas podem ser com alocação estática(possui um número limitado de itens) ou com alocação dinâmica (possui um número ilimitado de itens). Estamos interessados na fila de prioridade com ponteiro, a que não possui um

limite pré-definido podendo ter um número infinito de itens. O conceito de fila, é simples, de acordo com o primeiro que chega, você vai inserindo itens, de forma posterior a esse inserido, de forma que o primeiro que chega, é sempre o primeiro que sai. Na fila de prioridade é o mesmo, porém há coisas com um nível de prioridade maior, que são colocadas à frente de algo com a prioridade menor, então nem sempre o primeiro que chega é o primeiro que sai, isso vai depender da prioridade.

No trabalho, é exigido que utilizemos a fila de prioridade com ponteiro, porém utilizei de uns conceitos novos apresentados em sala de aula recentemente, como as bibliotecas para manipulação de tempo (time.h e chrono) e uma biblioteca para manipulação de listas (vector), que juntos resultaram na criação do minissistema de gerenciamento da UPA.

Tendo isso tudo em vista, o trabalho foi exigido para que seja possível desenvolver o raciocínio lógico quanto a aplicação dessa estrutura, e com o bônus de novamente aplicar elas em conjunto.

2. DESENVOLVIMENTO

Nesta seção do documento é apresentado, os conceitos aprendidos e o desenvolvimento do trabalho em si, na linguagem C++.

2.1. Conceitos Aplicados

Explicação sucinta dos conceitos de Fila de Prioridade com ponteiro, bibliotecas chrono, time.h e vector, e Tipos Abstratos de Dados (TADs).

2.1.1. Tipos Abstratos de Dados

As estruturas utilizadas como registro para criação de objetos, e as funções de manipulação dessas estruturas, ambas compõem uma TAD. Declaramos esses modelos como *structs*, elas são representações de qualquer coisa no mundo real, sendo ela lógica, abstrata ou física, como por exemplo uma pessoa, que é algo físico, ou um filme digital, que é algo lógico/abstrato, veja o exemplo na figura 1. Cada um tem suas características específicas, uma pessoa nome, sexo, idade, CPF, altura, dentre outras, e um filme título, linguagem, elenco, personagens, duração, categoria, ano de lançamento, dentre outros, e tudo isso pode ser definida dentro de uma struct para cada um deles. Resumindo uma Struct é uma espécie de variável modelo para cadastrar diferentes itens, dentro de um software escrito em C/C++. Acompanhado das structs temos as funções para manipulação dos dados dessa lista, e desses itens, que será visto no próximo tópico.

Figura 1 – Struct exemplo, sem ligação com o trabalho

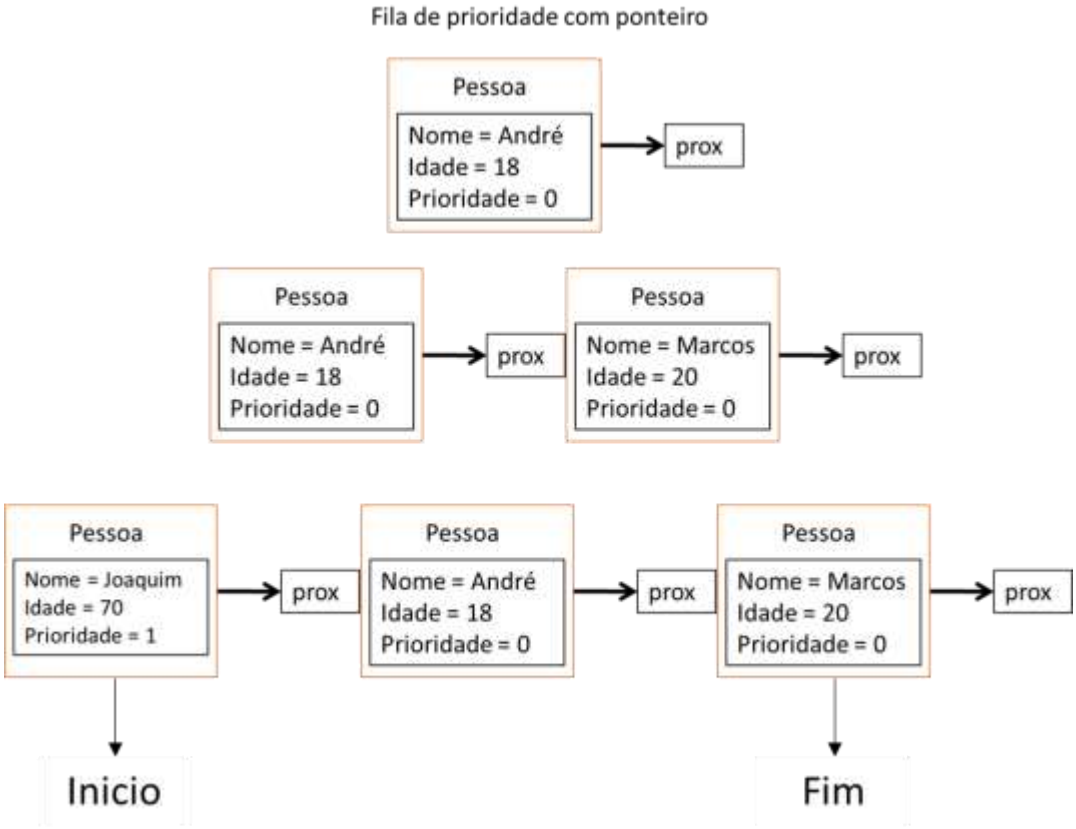
```
6   typedef struct Horario{
7       int hora;
8       int min;
9   };
10
11  typedef struct Data{
12      int dia;
13      int mes;
14      int ano;
15  };
16
17  typedef struct Compromisso{
18      char descricao[100];
19      Data dt;
20      Horario hr;
21  };
```

2.1.2. Fila de prioridade com ponteiro

Figura 2 – Fila

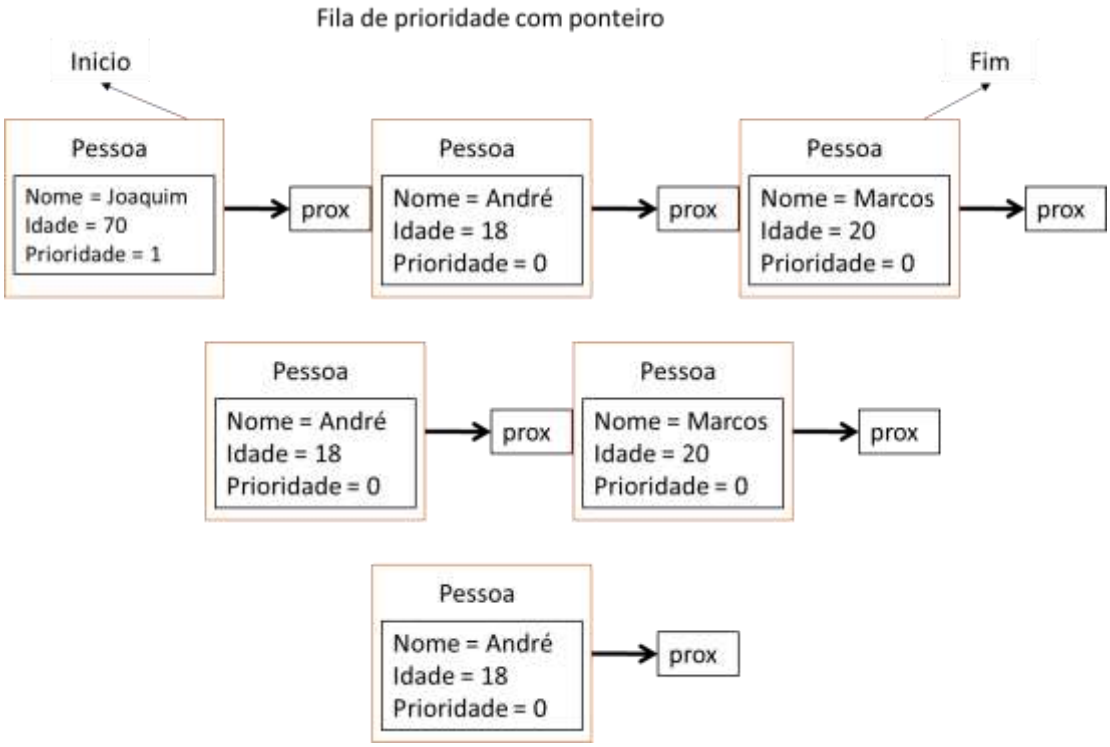
Na figura 2, vemos um desenho esquemático de como é a estrutura de dados da Fila. Na figura 3 vemos o processo de enfileiramento de itens em uma fila de prioridade, e na figura 4 vemos o desenfileiramento.

Figura 3 – Enfileiramento de itens



Como podemos observar, o enfileiramento ocorre sempre no fim da fila, caso haja alguém com prioridade, como é o caso do Joaquim, ele é inserido na frente de todo mundo, consequentemente ele será o primeiro a ser atendido.

Figura 4 – Desenfileiramento de itens



Como podemos observar, o desenfileiramento ocorre sempre no início da fila, o primeiro que entra é o primeiro que sai.

2.1.3. Biblioteca time.h

Nos cabeçalhos da time.h e ctime(sua evolução) estão localizadas as funções, variáveis, e macros para manipulação de unidades de tempo, como exemplo temos o ano, mês, dia, hora, minutos, e segundos. A partir das funções fornecidas pela mesma é possível obter, por exemplo, a data e o horário retornadas pelo sistema, entre outras funcionalidades relacionadas a operações com data e hora.

2.1.4. Biblioteca vector

Vetores são contêineres de sequência que representam arrays(lista com endereços de memória) que podem mudar de tamanho. Assim como os arrays, os vetores usam locais de armazenamento contíguos para seus elementos, o que significa que seus elementos também podem ser acessados usando deslocamentos em ponteiros regulares para seus elementos, e com a mesma eficiência dos arrays. Internamente, os vetores usam um array alocado dinamicamente para armazenar seus elementos. Diferentemente dos arrays, seu tamanho pode mudar dinamicamente, com seu armazenamento sendo manipulado automaticamente pelo

contêiner. O array pode precisar ser realocado para aumentar de tamanho quando novos elementos são inseridos, o que implica alocar um novo array e mover todos os elementos para ele. Esta é uma tarefa relativamente cara em termos de tempo de processamento, e portanto, os vetores não são realocados cada vez que um elemento é adicionado ao contêiner. Em comparação com os arrays, os vetores consomem mais memória em troca da capacidade de gerenciar o armazenamento e crescer dinamicamente de forma eficiente. Comparado com os outros contêineres de sequências dinâmicas (deques, stacks, queues, lists, etc), os vetores são muito mais eficientes acessando seus elementos (assim como os arrays) e adicionando ou removendo elementos de seu final. Os vetores permitem inserções e exclusões em tempo constante no final da sequência. Inserir ou excluir elementos no meio de um vetor exige tempo linear. A classe vector da biblioteca padrão C++ ´e um modelo de classe para contêineres de sequência. Através dela podemos armazenar elementos de um determinado tipo de maneira linear e realizando acessos aleatórios rápidos a qualquer momento.

2.1.5. Biblioteca chrono

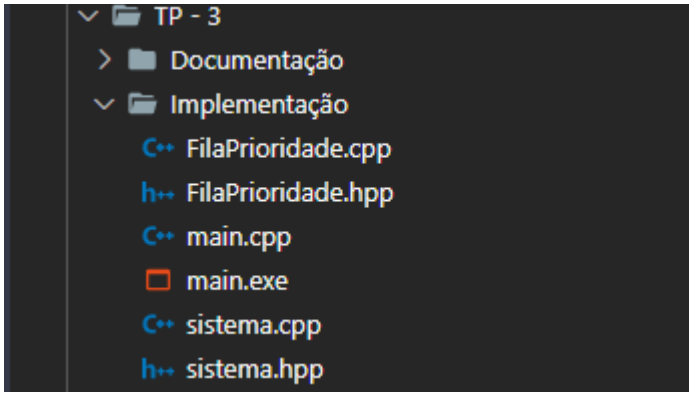
A biblioteca chrono considera o fato de que “relógios” e “temporizadores” (timers e clocks) podem diferenciar uns dos outros em diferentes sistemas, variando sua precisão ao longo do tempo. Assim, o objetivo é prover uma biblioteca independente de precisão, através da separação entre “duração” e “ponto no tempo” (duration e timepoint). A biblioteca padrão C++ provê três relógios:

- system clock: representa pontos associados com o relógio real do sistema.
- steady clock: um relógio que garante que nunca será ajustado.
- high resolution clock: um relógio com o menor período de tique no sistema.

2.2. Implementação

Utilizei uma modularização que está na figura 5, dividindo em 6 arquivos, 3 arquivos *.cpp, 2 arquivo *.hpp, e 1 arquivo *.exe.

Figura 5



No arquivo filaPrioridade.cpp, tenho todas as funções das TADs de fila de prioridade (Apêndice A). No sistema.hpp (Apêndice B), contém o cabeçalho das funções específicas do sistema.cpp, que será explicado logo abaixo.

Figura 6

```
void iniciaMedicos(Hospital *hospital){
    Medico medico;
    hospital->medicos_registrados = 5;
    hospital->medicos_disponiveis = 5;
    medico.disponivel = true;

    medico.nome = "Paulo Muzy";
    medico.especialidade = "Cardiologia";
    medico.crm = "783856242-42";
    hospital->medicos[0] = medico;

    medico.nome = "Renato Cariani";
    medico.especialidade = "Nutrição";
    medico.crm = "156717331-94";
    hospital->medicos[1] = medico;

    medico.nome = "Marco Antônio";
    medico.especialidade = "Clínico Geral";
    medico.crm = "853217418-33";
    hospital->medicos[2] = medico;

    medico.nome = "Pedro Braga";
    medico.especialidade = "Infectologia";
    medico.crm = "145723701-14";
    hospital->medicos[3] = medico;

    medico.nome = "Antônio Augusto";
    medico.especialidade = "Neurologia";
    medico.crm = "856234505-51";
    hospital->medicos[4] = medico;
}
```

Na figura 6, temos uma função que insere 5 médicos no início da execução, para que não se perca tempo cadastrando-os. Usa-se uma variável que após escrita com as informações daquele apenas uma variável de tipo medico, alterando-a, salvando-o em 1 posição do vetor, e sobrescrevendo a mesma, para inserir outro médico, e assim suscetivelmente.

Figura 7

```
60  Paciente cadastro_Paciente(){ // perguntas para criação um novo paciente
61      Paciente paciente;
62      cout << "Nome do Paciente: ";
63      getline(cin, paciente.nome);
64      cout << "Endereço: ";
65      getline(cin, paciente.endereco);
66      cout << "Idade: ";
67      cin >> paciente.idade;
68      cin.ignore();
69      do{
70          cout << "Opções: \n1 - Masculino\n2 - Feminino\nSexo: ";
71          cin >> paciente.sexo;
72      }while (paciente.sexo != 1 && paciente.sexo != 2);
73      system("cls");
74      return paciente; //retorna o "objeto" paciente
75  }
```

Na figura 7, temos a função de cadastro de paciente, que cria uma variavel paciente de forma local, e faz as devidas alterações nele, e retorna ele, para que os mesmo possa ser usado em outras funções, como paramêtro.

Figura 8

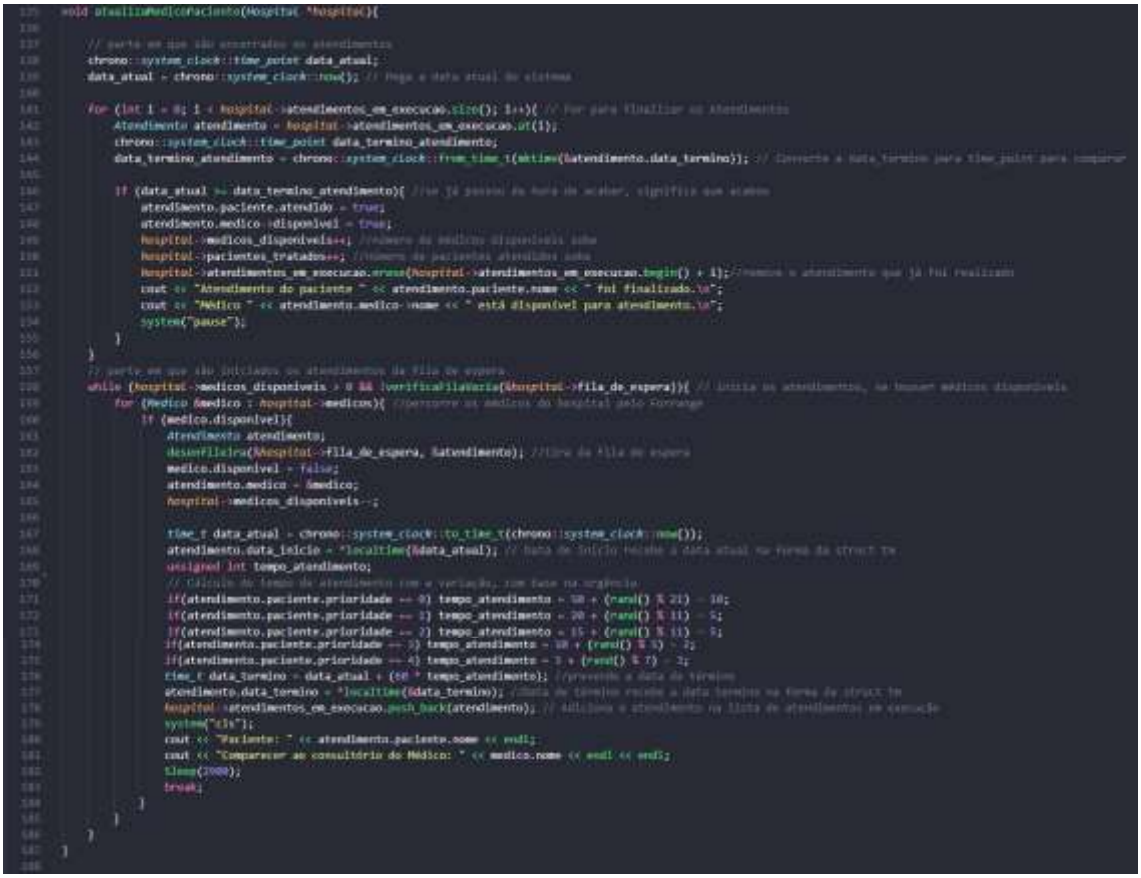
```
void realizarTriagem(Paciente *paciente){
    Triagem triagem;
    cout << "**********************************************************" << endl;
    cout << "**********************************************************" << endl;
    cout << "**                                                      **" << endl;
    cout << "**                      TRIAGEM                         **" << endl;
    cout << "**                                                      **" << endl;
    cout << "**********************************************************" << endl;
    cout << "**********************************************************" << endl;
    cout << "**                                                      **" << endl;
    cout << "**   01 - Comprometimento das vias aéreas?              **" << endl;
    cout << "**   02 - Respiração ineficaz?                          **" << endl;
    cout << "**   03 - Choque?                                       **" << endl;
    cout << "**   04 - Não responde a estímulos?                     **" << endl;
    cout << "**   05 - Em convulção?                                 **" << endl;
    cout << "**   06 - Dor severa?                                   **" << endl;
    cout << "**   07 - Grande hemorragia incontrolável?              **" << endl;
    cout << "**   08 - Alteração do estado de consciência?           **" << endl;
    cout << "**   09 - 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 percistente?                           **" << endl;
    cout << "**   14 - Temperatura entre 38°C  e 39°C?               **" << endl;
    cout << "**   15 - É idoso ou grávida?                           **" << endl;
    cout << "**   16 - Dor leve?                                     **" << endl;
    cout << "**   17 - Náusea?                                       **" << endl;
    cout << "**   18 - Temperatura entre 37°C e 38°C?                **" << endl;
    cout << "**                                                      **" << endl;
    cout << "**********************************************************" << endl;
    cout << "**********************************************************" << endl;
    cout << "Dado o menu acima responda as perguntas apenas com S(sim) e N(não) de acordo com o número da pergunta.\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'){ //se alguma das perguntas for sim, a urgencia é encontrada
            urgencia_encontrada = true;
        }
        if (urgencia_encontrada && (i % 5 == 0 || i == 18)){ //if que vai definir o nível da urgência
            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; //ser não foi encontrada nenhuma urgencia, ele espera mais tempo
    }
    paciente->triagem = triagem; //respostas inseridas nos dados do usuario
}
```

Na figura 8, temos a triagem, que são 18 perguntas, com todas as respostas inseridas como N inicialmente. Temos uma variavel booleana chamada urgencia_encontrada, que serve para definir a urgência 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, 10 ou 15 perguntas, ele já pula direto pro atendimento, setando a prioridade de acordo com a parada de pergunta, se parou no 5, ele vai dividir por 5 e subtrair por 1(5/5 - 1 = 0), e temos a urgência de primeiro nível, a vermelha, se parou no 10, ele irá dividir por 5 e subtrair por 1(10/5 - 1 = 1), e temos o segundo nível de urgência, se parou no 15, vamos dividir por 5 e subtrair por 1(15/5 - 1 = 2), e temos o nível de urgência amarelo, se parou no 18, vamos dividir por 5 e subtrair por 1(18/5 - 1 = 2,6, porém pelo fato da prioridade ser tipo inteiro, ele vai arredondar para 3), e temos o nível de urgência verde. Caso não haja urgencia, ele será definido com o grau mais baixo de urgência, que seria o 4, o nível de urgência azul.

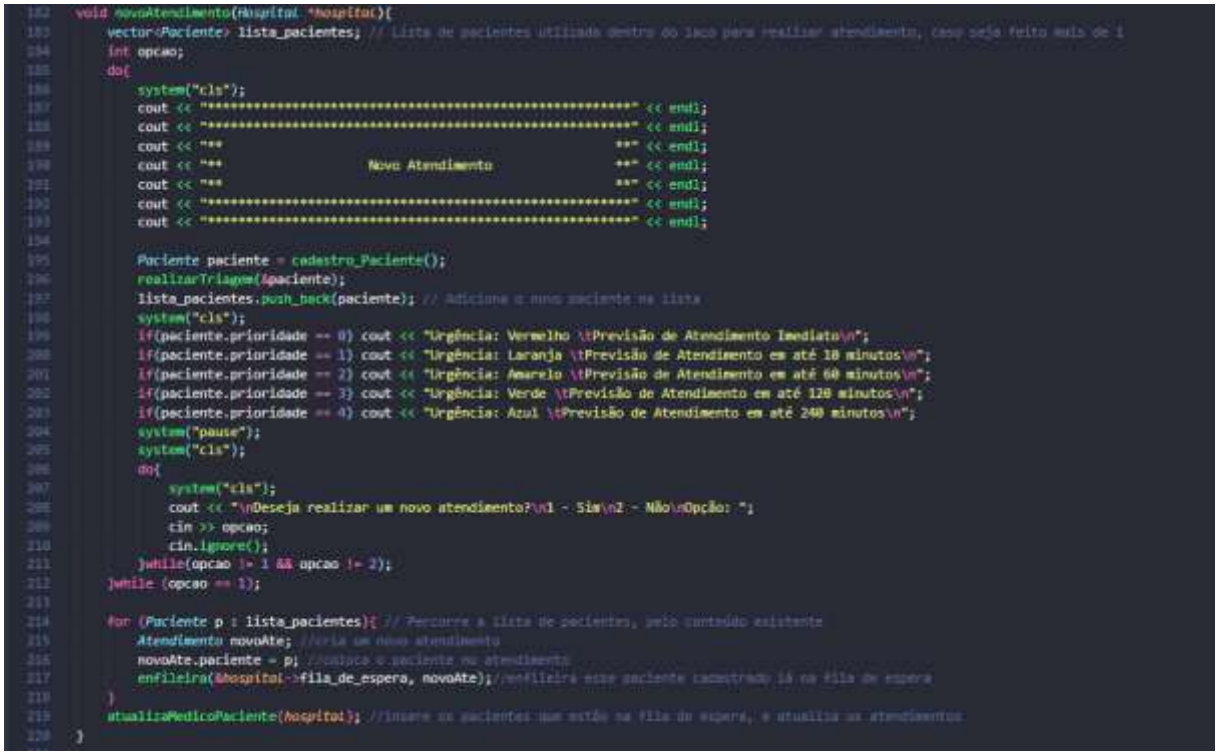
Figura 9



Na figura 9, temos uma função que atualiza o estado de um paciente que está sendo atendido, o estado do médico que está atendendo, e o estado do paciente que está na fila de espera. A função cria uma variável para pegar o horário do sistema, por meio da função system_clock::now(), da biblioteca chrono. Depois percorre-se toda a lista de atendimentos por meio de um for começando em 0, e indo até a quantidade de atendimentos em execução no hospital que estão do array, criado por meio da vector. Uma variável do tipo Atendimento que será usada para recolher os atendimentos doo array 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, então 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, por meio doo forrange, e logo em seguida, esse médico será direcionado para o atendimento. Depois é pego o horário do sistema de novo, este será usado como data de início do atendimento, e para calcular o tempo de atendimento junto com a variação, de acordo com a urgência dele, e assim calcular a data de término do atendimento. E assim insere o atendimento de forma definitiva na lista de atendimentos.

Figura 10



```
void novoAtendimento(Hospital *hospital){
    vector<Paciente> lista_pacientes; // Lista de pacientes utilizada dentro do laço para realizar atendimento, caso seja feito mais de 1
    int opcao;
    do{
        system("cls");
        cout << "*****************************************************" << endl;
        cout << "*****************************************************" << endl;
        cout << "**                                                 **" << endl;
        cout << "**                  Novo Atendimento               **" << endl;
        cout << "**                                                 **" << endl;
        cout << "*****************************************************" << endl;
        cout << "*****************************************************" << endl;

        Paciente paciente = cadastro_Paciente();
        realizarTriagem(&paciente);
        lista_pacientes.push_back(paciente); // Adiciona o novo paciente na lista
        system("cls");
        if(paciente.prioridade == 0) cout << "Urgência: Vermelho \tPrevisão de Atendimento Imediato\n";
        if(paciente.prioridade == 1) cout << "Urgência: Laranja \tPrevisão de Atendimento em até 10 minutos\n";
        if(paciente.prioridade == 2) cout << "Urgência: Amarelo \tPrevisão de Atendimento em até 60 minutos\n";
        if(paciente.prioridade == 3) cout << "Urgência: Verde \tPrevisão de Atendimento em até 120 minutos\n";
        if(paciente.prioridade == 4) cout << "Urgência: Azul \tPrevisão de Atendimento em até 240 minutos\n";
        system("pause");
        system("cls");
        do{
            system("cls");
            cout << "\nDeseja realizar um novo atendimento?\n1 - Sim\n2 - Não\nOpção: ";
            cin >> opcao;
            cin.ignore();
        }while(opcao != 1 && opcao != 2);
    }while (opcao == 1);

    for (Paciente p : lista_pacientes){ // Percorre a lista de pacientes, pelo conteúdo existente
        Atendimento novoAte; //cria um novo atendimento
        novoAte.paciente = p; //coloca o paciente no atendimento
        enfileira(&hospital->fila_de_espera, novoAte);//enfileira esse paciente cadastrado lá na fila de espera
    }
    atualizaMedicoPaciente(hospital); //insere os pacientes que estão na fila de espera, e atualiza os atendimentos
}
```

Na figura 10, temos a variavel paciente, recebendo o retorno da função na figura 7, logo em seguida esse mesmo retorno é mandado para a triagem, e ele é colocado no array de lista de paciente, depois é mostrado o nível de urgencia e depois a previsão de atendimento conforme a prioridade. Depois é perguntado se irá realizar 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 de espera, e depois ele atualiza, inserindo esses pacientes em um atendimento, por meio da função apresentada na figura 9.

Figura 11

```
void exibeAtendimentos(Hospital hospital){
    if (hospital.atendimentos_em_execucao.size() > 0){
        for (Atendimento atendimento : hospital.atendimentos_em_execucao){
            cout << "\nMédico: " << atendimento.medico->nome;
            cout << "\nPaciente: " << atendimento.paciente.nome << endl;
            if(atendimento.paciente.prioridade == 0) cout << "Urgência: Vermelho\n";
            if(atendimento.paciente.prioridade == 1) cout << "Urgência: Laranja\n";
            if(atendimento.paciente.prioridade == 2) cout << "Urgência: Amarelo\n";
            if(atendimento.paciente.prioridade == 3) cout << "Urgência: Verde\n";
            if(atendimento.paciente.prioridade == 4) cout << "Urgência: Azul\n";
            cout << "\nInício: " << put_time(&atendimento.data_inicio, "%d/%m/%Y %H:%M:%S");          // converte a struct tm em string
            cout << "\nPrevisão de Término: " << put_time(&atendimento.data_termino, "%d/%m/%Y %H:%M:%S") << endl << endl; // converte a struct tm em string
        }
        system("pause");
    }
    else cout << "Nenhum atendimento em execução." << endl;
    Sleep(1500);
}
```

Na figura 11, vemos a função que exibe o 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.

Figura 12

```
240  void exibeFilaEspera(Hospital hospital){
241      cout << "**********************************************************" << endl;
242      cout << "**********************************************************" << endl;
243      cout << "**                                                      **" << endl;
244      cout << "**                    Fila de Espera                    **" << endl;
245      cout << "**                                                      **" << endl;
246      cout << "**********************************************************" << endl;
247      cout << "**********************************************************" << endl;
248      imprimeFila(&hospital.fila_de_espera);
249      system("pause");
250  }
```

Na figura 12, temos a função para exibir a Fila de espera, que são aqueles pacientes que ainda estão esperando para serem atendidos

Figura 13

```
252  void exibeRelatorio(Hospital hospital){
253      cout << "**********************************************************" << endl;
254      cout << "**********************************************************" << endl;
255      cout << "**                                                      **" << endl;
256      cout << "**                     Relatório                        **" << endl;
257      cout << "**                                                      **" << endl;
258      cout << "**********************************************************" << endl;
259      cout << "**********************************************************" << endl;
260      cout << "\nQuantidade de atendimentos em andamento: " << hospital.atendimentos_em_execucao.size();
261      cout << "\nQuantidade de médicos registrados: " << hospital.medicos_registrados;
262      cout << "\nQuantidade de médicos disponíveis: " << hospital.medicos_disponiveis;
263      cout << "\nQuantidade de pacientes atendidos e tratados: " << hospital.pacientes_tratados;
264      cout << "\nQuantidade de pacientes em espera: " << hospital.fila_de_espera.tamanho;
265      system("pause");
266  }
```

Na figura 13 temos a função apara exibir o relatório, que mostra o que já foi feito durante o tempo de execução do programa, o número de médicos registrados e quantos estão disponíveis, quantos pacientes já foram atendidos, e quantos ainda estão esperando.

3. CONCLUSÃO

Ao longo do trabalho tive várias ideias de como realizar o problema, a parte de fila de prioridade pode ser feita de várias formas, como separando 3 filas, 1 com prioridade, 1 sem prioridade, e no fim, concatenar as duas em uma outra fila, ou então fazemos ela com ponteiro e inserimos após o id, ou nesse caso após o item de maior prioridade, e por aí vai. Reutilizei os códigos padrões das TADs e fiz as devidas alterações, reutilizei um pouco do código do último trabalho que fiz, também fiz uso dos slides disponibilizados pelo professor para sanar dúvidas a respeito das bibliotecas, e usuários externos para retirar dúvidas sobre uso das bibliotecas.

Creio eu que meu desenvolvimento foi bom nesse trabalho, desenvolvi novas formas de raciocínio e tive que abstrair e diminuir muitas coisas como de costume, para que assim o trabalho ficasse leve, sem ser muito extenso. Tive que buscar ajuda externa e conhecimento externo para realização do trabalho, mas mesmo assim consegui atingir meus objetivos estipulados no início do trabalho, explicar bem o que eu estava fazendo dentro de cada função, apresentar conhecimentos em Fila, e com confiança aprimorar meus conhecimentos na biblioteca time.h, e ter uma motivação para estudar mais sobre as aplicações da biblioteca chorno e vector. Estou feliz e satisfeito com o resultado, e ele atendeu às minhas expectativas além do que eu esperava.

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: 16 de novembro de 2022.

Link do código no *Github*: https://github.com/gKaicon/Arquivos_C_and_C-withClasses/tree/main/AEDs%20I/TP/TP%20-%203

5. APÊNDICES

5.1. Apêndice A – TADs de Fila(*.cpp)

```cpp
#include "FilaPrioridade.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 cel = new Celula;
    cel->item = item; //dentro da célula tem um item
    cel->prox = NULL; //e o apontador da célula acima recebe null
    Apontador anterior = fila->inicio; //guardar a primeira posição que será usada para percorrer a fila
    Apontador atual = fila->inicio; //guardar a primeira posição que será usada para percorrer a fila

    if (verificaFilaVazia(fila)) fila->inicio = cel; //se a fila for vazia, insere na primeira posição
    else{ //se não vai procurar onde inserir
        while (atual != NULL && (atual->item.paciente.prioridade <= item.paciente.prioridade)){
            //condição acima serve para verificar se a prioridade do item inserido é maior, que os itens já inseridos
            anterior = atual; //se não for, ele troca os valores de anterior e atual, guardando sempre o último de maior prioridade
            atual = atual->prox;
        }

        if (atual == fila->inicio){//se ao sair do while, o atual, for o primeiro posição, a inserção é feita
            fila->inicio = cel; //primeira posição recebe a célula criada
            cel->prox = atual; //apontador recebe null
        }
        else{
            anterior->prox = cel; // o valor antes de atual, vai receber como próximo a célula
            cel->prox = atual; //o apontador prox recebe null, para realizar a próxima inserção
        }
    }
    fila->tamanho++;
}
```

```cpp
void desenfileira(FilaPrioridadeDinamica *fila, Atendimento *item){
    if (verificaFilaVazia(fila)){
        cout << "Fila vazia!" << endl;
        return;
    }
    //a remoção é feita no início da fila
    Apontador aux = fila->inicio;
    *item = aux->item;
    fila->inicio = aux->prox;
    delete aux;
    fila->tamanho--;
}

void esvaziaFila(FilaPrioridadeDinamica *fila){
    Atendimento item;
    while (!verificaFilaVazia(fila)){
        desenfileira(fila, &item);
    }
}
```

```cpp
void imprimeFila(FilaPrioridadeDinamica *fila){
    if (verificaFilaVazia(fila)){
        cout << "Fila vazia!" << endl;
        return;
    }
    Apontador aux = fila->inicio;
    while (aux != NULL) {
        cout << "Paciente: " << aux->item.paciente.nome << endl;
        if(aux->item.paciente.prioridade == 0) cout << "Urgência: Vermelho\n";
        if(aux->item.paciente.prioridade == 1) cout << "Urgência: Laranja\n";
        if(aux->item.paciente.prioridade == 2) cout << "Urgência: Amarelo\n";
        if(aux->item.paciente.prioridade == 3) cout << "Urgência: Verde\n";
        if(aux->item.paciente.prioridade == 4) cout << "Urgência: Azul\n";
        cout << "Endereço: " << aux->item.paciente.endereco << endl;
        cout << "Idade: " << aux->item.paciente.idade << endl;
        cout << "Sexo: " << (aux->item.paciente.sexo == 1 ? "Masculino" : "Feminino") << endl;
        cout << "\n";
        aux = aux->prox;
    }
    cout << "\n";
}
```

5.2. Apêndice B – Funções do Sistema

```
#ifndef SISTEMA_H
#define SISTEMA_H
#include <vector>
#include "FilaPrioridade.cpp"
#define OPCAO_SAIDA 5

using namespace std;


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 iniciaMedicos(Hospital *hospital);
void menu();
Paciente cadastro_Paciente();
void realizarTriagem(Paciente *paciente);
void atualizaMedicoPaciente(Hospital *hospital);
void novoAtendimento(Hospital *hospital);
void exibeAtendimentos(Hospital hospital);
void exibeFilaEspera(Hospital hospital);
void exibeRelatorio(Hospital hospital);

#endif
```

5.3. Apêndice C – TADs de Fila(*.hpp)

```cpp
#ifndef FILA_PRIORIDADE_DINAMICA
#define FILA_PRIORIDADE_DINAMICA

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

using namespace std;

typedef struct Triagem{
    char respostas[18] = {'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N'};
};

//unsigned vai ser utilizado para evitar que usuarios digitem valores negativos
typedef struct Paciente{
    string nome;
    string endereco;
    unsigned int idade;
    unsigned int sexo;
    Triagem triagem;
    unsigned int prioridade;
    bool atendido;
};

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

typedef struct Atendimento{
    Medico *medico;
    Paciente paciente;
    tm data_inicio;
    tm data_termino;
};

typedef struct Celula *Apontador;

typedef struct Celula{
    Atendimento item;
    Apontador prox;
};

typedef struct FilaPrioridadeDinamica{
    Apontador inicio;
    int tamanho;
};

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
```

5.4. Apêndice D – Main.cpp

```cpp
#include "sistema.cpp"

int main(){
    UINT CPAGE_UTF8 = 65001;
    UINT CPAGE_DEFAULT = GetConsoleOutputCP();
    SetConsoleOutputCP(CPAGE_UTF8);
    HANDLE colors = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(colors, 2); // Define a cor verde para o texto
    srand(time(NULL));

    int opcao;
    Hospital hospital;
    iniciaMedicos(&hospital);
    inicializaFila(&hospital.fila_de_espera);
    do{
        atualizaMedicoPaciente(&hospital);
        menu();
        cin >> opcao;
        cin.ignore();
        system("cls");
        switch (opcao){
            case 1:
                novoAtendimento(&hospital);
                break;
            case 2:
                exibeAtendimentos(hospital);
                break;
            case 3:
                exibeFilaEspera(hospital);
                break;
            case 4:
                exibeRelatorio(hospital);
                break;
            case 5:
                system("cls");
                cout << "Saindo..." << endl;
                Sleep(1000);
                break;
        }
    }while (opcao != OPCAO_SAIDA);
    return 0;
}
```