INSTITUTO FEDERAL
Santa Catarina

Relatório Diário de Bordo
Projeto 1 Programação 2

Ramon dos Santos Sobrinho

17 de junho de 2022

Resumo

Esse relatório tem o intuito de explicar de uma forma mais sucinta sobre as estruturas de dados utilizadas no decorrer do código para que ele pudesse funcionar com 100% de eficácia.

Código

Nosso relatório foi separado em 4 arquivos diferentes para um melhor entendimento.

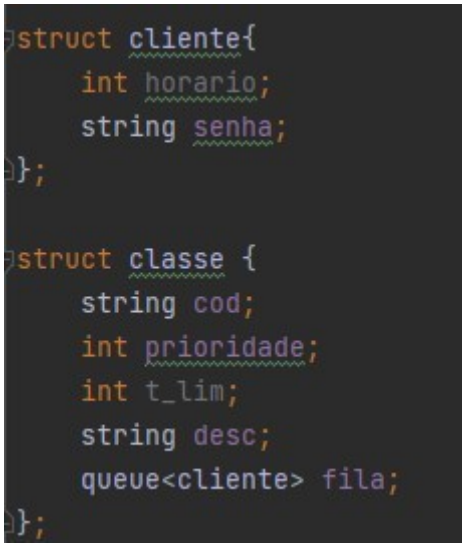
Em relação as estruturas de dados utilizadas no código, foram duas: Listas e Filas.

Em resumo a lista foi utilizada para armazenar as classes que estavam presentes no CSV. Sua importância se da pelo fato de que com ela poderiamos atender os clientes baseados na prioridade da classe escolhida por cada cliente visto que a lista é a única estrutura de dados estudada até então que pode ser ordenada baseada em um critério. A lista teve também sua importância na função de criar as classes em ordem pois ela quem irá armazenar as classes contidas no arquivo. O fato da lista poder ser interada também foi importante pois utilizamos na função de adicionar cliente na fila correta, podendo assim conferir a opção que o usuário digitou e inseri-lo.

A fila, onde o primeiro dado inserido é o primeiro a ser retirado, teve como função garantir que o primeiro cliente a receber uma senha, fosse o primeiro a ser atendido. Ela também foi importante na hora de separar o conteudo das linhas do arquivo em substrings, sendo utilizados posteriormente para definir os paramentros de cada classe.

1 FUNCIONALIDADE.H

Começando pelo arquivo "funcionalidade.h" onde é declarada todas as funções que serão utilizadas dentro do código.
Foram utilizadas duas structs: "cliente" onde é criada uma string onde será guardada a senha do cliente e declarado como inteiro o horário que o cliente acessou, e "classe" onde é criada uma string onde guarda as informações contidas no arquivo .CSV como código, tempo, prioridade e descrição da classe.



Funções declaradas:

- Void separa: responsável por pegar as informações contidas no arquivo e separar em substrings.

```
void separa(const string & algo, char sep, queue<string> & q);
```

- Bool ordena por prioridade: Como o próprio nome já diz, função responsável por organizar a lista de classe por prioridade.

```
bool ordena_por_prioridade (const classe & c1, const classe & c2);
```

- Bool ordena em codigo: Função que irá ordenar a lista de classe por código, esse contido no arquivo .CSV

```
bool ordena_em_codigo(const classe & c1, const classe & c2);
```

- Void cria classes em ordem: Essa função tem como principal objetivo criar as classes que estão contidas no arquivo .CSV e separar cada informação para que o programa funcione como solicitado.

```
void cria_classes_em_ordem (const string & csv_file, list<classe> & filas);
```

- Void adiciona cliente na fila certa: Função que irá adicionar o ciente na sua respectiva fila.

```
void adiciona_cliente(string & cod, list<classe> & filas_de_atendimento);
```

- Void retira cliente: Nessa função é responsável por retirar o cliente da fila quando for chamado para atendimento.

```
void retira_cliente(list<classe> & filas_de_clientes);
```

Partindo para a parte dos menus, temos as seguintes funções:

- Void Menu cliente: Responsável por criar o menu do cliente, onde mostrará na tela as opções em relação a classes para serem escolhidas.

```
void menu_cliente(list<classe> & filas_clientes);
```

- Void Menu atendente: Função responsável por criar a interface do atendente, onde aparecerá as opções para serem escolhidas: Atender cliente ou retornar para menu inicial.

```
void menu_atendente(list<classe> & filas_clientes);
```

- Void Menu inicial: Função essa responsável por criar o menu inicial mostrando ao operador escolher se é atendente, cliente ou se deseja sair do sistema.

```
void menu_inicial(list<classe> & filas_clientes);
```

Para executar as funções declaradas no “funcionalidades.h” separamos em dois arquivos, o primeiro é “funcionalidades.cpp” onde será executada todas as funções referente a criação de classes, senhas e alteração nas filas, e o segundo “menu_usuario.cpp” onde será executada as funções responsáveis pela criação dos menus.

2 FUNCIONALIDADE.CPP

Em funcionalidades.cpp, como mencionado anteriormente estão a grande maioria das funções responsáveis pela parte mais “afundo” do código.

- Void separa: responsável por pegar as informações contidas no arquivo e separar em substrings. Nessa função é utilizado o while como forma de percorrer a string e encontrar onde deve ser separado. Nesse caso o inicio de uma posição se dá pelo fim da outra possição, separadas por um "char separador".

```
while (i != string::npos) {
    int pos1 = algo.find_first_not_of( c: sep, pos: i);
    int pos2 = algo.find( c: sep, pos: pos1);
    if (pos2 == string::npos) algo.substr( pos: pos1);
    aux = algo.substr( pos: pos1, n: pos2-pos1);
    q.push( x: aux);
    i = pos2;
}
```

- Bool ordena por prioridade: Como o próprio nome já diz, função responsável por organizar a lista de classe por prioridade.

```
bool ordena_por_prioridade (const classe & c1, const classe & c2);
```

- Bool ordena em codigo: Função que irá ordenar a lista de classe por código, esse contido no arquivo .CSV

```
bool ordena_em_codigo(const classe & c1, const classe & c2);
```

- Void cria classes em ordem: Essa função tem como principal objetivo criar as classes que estão contidas no arquivo .CSV e separar cada informação para que o programa funcione como solicitado. Nessa parte é feita a abertura do arquivo .CSV, utilizado o if pois caso o arquivo for inválido, retornar uma mensagem de arquivo inválido e assim encerra a função. Utilizamos um while para enquanto ler a linha do arquivo, separar as strings da linha e inserir na fila parâmetros separados. Primeiro item como código, segundo prioridade, terceiro como tempo e quarto sendo a descrição da classe mencionada no arquivo.

```
while(getline( &: arq, &: linha_arq)){
    separa( algo: linha_arq,sep, &: parametros_separados);
    while(!parametros_separados.empty()){
        classe.cod = parametros_separados.front();
        parametros_separados.pop();
        classe.prioridade = stoi( str: parametros_separados.front());
        parametros_separados.pop();
        classe.t_lim = stoi( str: parametros_separados.front());
        parametros_separados.pop();
        classe.desc = parametros_separados.front();
        parametros_separados.pop();
        filas.push_back( x: classe);
    }
}
```

Por fim é organizado as classes por prioridade.

```
filas.sort( comp: ordena_por_prioridade); //Organiza a lista baseado na prioridade
```

- Void adiciona cliente na fila certa: Função que irá adicionar o ciente na sua respectiva fila.

Utilizamos o if/else para conferir o tamanho da fila e passar ao cliente a senha correspondente. Caso a fila passe de 9 pessoas, atribuir mais uma casa decimal "00", se passar de 99, atribuir mais uma casa decimal "000" e assim por diante.

Após mostrado a senha ao cliente, o sistema retorna ao menu inicial.

```
for(auto & x : classe & : filas_de_atendimento){
    if(cod == x.cod){
        if(x.fila.size()<9){
            novo_cliente.senha = x.cod + "00" + to_string( val: x.fila.size()+1);
        }else if (x.fila.size() >= 9 && x.fila.size() < 99){
            novo_cliente.senha = x.cod + "0" + to_string( val: x.fila.size()+1);
        }else if (x.fila.size()>=99){
            novo_cliente.senha = to_string( val: x.fila.size()+1);
        }
        // Insere o cliente na fila referente a opção escolhida e mostra sua senha
        x.fila.push( x: novo_cliente);
        sleep(2);
        cout << novo_cliente.senha << endl << endl;
        return menu_inicial( &: filas_de_atendimento);
    }
}
```

- Void retira cliente: Nessa função é responsável por retirar o cliente da fila quando for chamado para atendimento.

Criamos dois interadores, um para ser utilizado no início da lista de classe e outra no fim da lista de classe.

```
auto atual : iterator<classe>  = filas_de_clientes.begin();
auto fim : iterator<classe>  = filas_de_clientes.end();
fim--;
```

Na sequência é utilizado um while seguindo de um if para conferir se o interador está na última classe da lista, caso positivo, retorna uma mensagem de que todas as filas estão vazias.

```
while (atual != filas_de_clientes.end()) {
    if (atual == fim && fim->fila.empty()) {
        cout << "No momento nao tem cliente em espera, voce sera redirecionado ao menu inicial." << endl << endl;
        menu_inicial( &: filas_de_clientes);
        return;
```

Posteriormente utilizado um else, se caso contrário a situação acima, quer dizer que possui clientes a serem atendidos, retornando assim uma mensagem de atendendo, na sequência retornando ao menu inicial.

```
    } else {
        if (!atual->fila.empty()) {
            cout << "Voce esta atendendo: " << atual->fila.front().senha << endl << endl;
            atual->fila.pop();
            menu_atendente( &: filas_de_clientes);
            return;
        } else {
            atual++;
        }
    }
}
```

3 MENU_USUARIO.CPP

Partindo para o outro arquivo “Menu_usuario.cpp” onde executa todas as funções responsáveis por criar os menus apresentáveis no sistema.

- Void Menu cliente: Responsável por criar o menu do cliente, onde mostrará na tela as opções em relação a classes para serem escolhidas.

Ordena a lista de classe em código para mostrar em ordem para o cliente selecionar uma das opções. Como mencionado anteriormente, a lista permite ordenar baseado em um critério, sendo muito importante nessa parte do código.

```
filas_clientes.sort( comp: ordena_em_codigo); //Ordena as opções apresentadas ao cliente

for(auto &x : classe & : filas_clientes) {
    cout << x.cod << ": " << x.desc << endl; // Apresenta na tela as opções com suas respectivas descrições
}
cout << endl;
cout << "Digite SAIR caso queira sair do menu do cliente" << endl;
```

Na sequência utilizamos um while para fazer a leitura do que o cliente digitar. Se digitar SAIR, retorna ao menu principal, se não adiciona o cliente na fila da opção selecionada.

```
while(true){
    cin >> cod;
    if(cod == "SAIR"){
        menu_inicial( &: filas_clientes);
        break;
    }
    adiciona_cliente( &: cod,  &: filas_clientes);
    cout << endl;
}
```

- Void Menu atendente: Função responsável por criar a interface do atendente, onde aparecerá as opções para serem escolhidas: Atender cliente ou retornar para menu inicial.

```
void menu_atendente(list<classe> & filas_clientes) {
    string op;

    cout << "Menu do atendente " << endl << endl;
    cout << "Digite 1 para atender um cliente" << endl;
    cout << "Ou digite 2 para sair do menu do atendente" << endl;

    cin >> op;
    cout << endl;
```

Utilizamos um loop na base do if/else para receber o que for digitado pelo cliente. Caso digite 1 retira o cliente da fila e atende o mesmo, caso digite 2 retorna para o menu inicial.

Se por acaso o atendente digitar alguma opção incorreta, através do else é mostrado que a opção foi incorreta.

```
    if(op == "1") {
        retira_cliente( &: filas_clientes);
        return;
    } else if (op == "2") {
        menu_inicial( &: filas_clientes);
        return;
    } else {
        cout << "Opcao digitada não corresponde a nenhuma das opcoes acima" << endl;
        cout << endl;
        menu_atendente( &: filas_clientes);
        return;
    }
}
```

- Void Menu inicial: Função essa responsável por criar o menu inicial mostrando ao operador escolher se é atendente, cliente ou se deseja sair do sistema.

```
void menu_inicial(list<classe> & filas_clientes){
    string opcoes;
    cout << "Menu do sistema" << endl << endl;
    cout << "Digite 1 para acessar o menu do atendente" << endl;
    cout << "Digite 2 para acessar o menu do cliente" << endl;
    cout << "Ou digite 3 para sair" << endl;

    cin >> opcoes;
```

Através do loop a base de if/else é executado o que for digitado, opção 1 entra no menu do atendente, opção 2 entra no menu do cliente e opção 3 encerra o sistema. Utilizando por último um else caso o cliente digite a opção invalida, mostra novamente o menu.

```
    if(opcoes == "1") {
        menu_atendente( &: filas_clientes);
        return;
    } else if (opcoes == "2") {
        menu_cliente( &: filas_clientes);
        return;
    } else if(opcoes == "3") {
        cout << "Sistema encerrado, até breve!" << endl;
        return;
    } else {
        cout << "Opcao digitada nao corresponde as opcoes acima" << endl;
        menu_inicial( &: filas_clientes);
    }
}
```

4 MAIN.CPP

Nesta função main é onde será feita a chamada das outras funções presentes nos outros arquivos.
Declaramos a lista com as classes, que estarão vazias no início do programa, na sequência é criada as classes e inseridas nas listas vazias. A parte do if fará o seguinte: Se a lista de classes estiver vazia, o programa irá mostrar o menu inicial.

```
int main (int argc, char * argv[]) {
    list<classe> lista_com_classes;
    cria_classes_em_ordem( csv_file: argv[1],  &: lista_com_classes);
    cout << endl;

    if(!lista_com_classes.empty()){
        menu_inicial( &: lista_com_classes);
    }
}
```