Relatório tarefa 1 - Lívia Lutz dos Santos - 2211055 - 3wa

Objetivo:

Criar um programa para organizar a lista de espera de pacientes de um hospital baseado na ordem de prioridade e dentro da ordem de prioridade, a lista é organizada pela ordem de chegada de cada paciente na mesma prioridade de cor.

A prioridade possui 3 cores: Vermelha, com a prioridade máxima, Amarela, com prioridade média e Verde com prioridade mínima.

A lista mostra os pacientes por ordem de prioridade, portanto, os com prioridade vermelha serão os primeiros da lista e logo, os primeiros a serem atendidos, enquanto os com prioridade verde se encontram no final da lista e serão atendidos por último.

Conforme os pacientes são atendidos , desistem do atendimento ou são transferidos, o programa mostra a lista atualizada e informações sobre quantos pacientes de cada prioridade faltam ser atendidos. Essa informação estará sendo exibida até que todos os atendimentos, desistências ou transferências tenham sido realizados.

Estrutura do programa:

O módulo hospital.h possui todas as funções necessárias para a execução do programa, incluindo a definição do TDA usado para guardar informações sobre cada paciente e as bibliotecas <stdlib.h> e <string.h> que permite a alocação e liberação de memória e facilita a manipulação de strings, respectivamente.

-> TDA de cada paciente:

Inclui um ponteiro de char para armazenar a prioridade que o paciente se encontra, outro ponteiro de char para armazenar seu nome, um int para guardar seu número de chegada ao hospital e um ponteiro para o tipo Paciente definido no módulo hospital.h usando typedef,que aponta para o próximo paciente na fila de espera

```c
/*Estrutura de dados que contem informações sobre a prioridade e o número de chegada de cada paciente no hopital*/

struct paciente {
    /*Prioridade do paciente*/
    char* cor;

    /*Número de chegada do paciente na fila*/
    int numChegada;

    /*Nome do paciente*/
    char* nome;

    /*Ponteiro para o próximo paciente*/
    Paciente* prox;
};
```

Descrevendo as funções implementadas:

-> Função de inicializar a lista (criaLista):

Inicializa a lista como NULL.

```c
Paciente* criaLista(void) {

    /*Retorna o um ponteiro NULL para o primeiro elemento da lista*/

    return NULL;
}
```

-> Função de inserir um paciente na lista (insereNaLista):

Cria um novo Paciente por alocação dinâmica, guardando informações sobre seu número de chegada, prioridade e nome, sendo esses dois últimos guardados por alocação dinâmica de strings.

Após criar o novo paciente por alocação dinâmica, percorre a lista e por ordem de prioridade usando a função numCor com 2 ponteiros (1 para guardar o elemento anterior e o outro para percorrer a lista) até a prioridade no parâmetro da função for encontrada.

Se o elemento for o primeiro da lista, ele terá um ponteiro para o resto da lista, senão ele será inserido no meio da lista.

Retorna a lista atualizada.

```c
/*Função que insere pacientes na lista usando a ordem de prioridades por cor - tem q fazer por numero de chegada tbm*/

Paciente* insereNaLista(char* n, int c,char* nome, Paciente* lista) {

    /*Novo e o ponteiro do novo elemento a ser inserido,
      a guarda a referência para o elemento anterior e p percorre a lista*/

    Paciente* novo, * a = NULL, * p = lista;

    /*Até o final da lista e até acharmos a primeira posicão da prioridade do novo paciente
      guardamos os ponteiros anteriores e andamos com a lista*/

    while (p != NULL && (numCor(p->cor) <= numCor(n))) {
        a = p;
        p = p->prox;
    }

    /*Alocando espaço para o novo paciente e seus dados*/

    novo = (Paciente*)malloc(sizeof(Paciente));

    if (novo == NULL) {
        return NULL;
    }

    novo->cor = (char*)malloc((strlen(n) + 1) * sizeof(char));

    if (novo->cor == NULL) {
        return NULL;
    }

    strcpy(novo->cor, n);

    novo->nome = (char*)malloc((strlen(nome) + 1) * sizeof(char));

    if (novo->nome == NULL) {
        return NULL;
    }
```

```
    strcpy(novo->nome, nome);

    novo->numChegada = c;

    /*Se o novo elemento for inserido no início da lista, continuamos a lista após ele*/

    if (a == NULL) {
        novo->prox = lista;
        lista = novo;
    }

    /*Inserindo o novo elemento no meio da lista*/

    else {
        novo->prox = p;
        a->prox = novo;
    }

    /*Retorna a lista atualizada*/

    return lista;
}
```

-> Função de imprimir a lista completa (imprimeLista):

Usa um ponteiro para percorrer a lista e imprimir os dados de número de chegada e prioridade de cada paciente.

```
/*Função que imprime a lista de pacientes a serem atendidos*/

void imprimeLista(Paciente* l) {

    /*Ponteiro que percorre a lista até o final*/
    Paciente* p;

    /*Loop que atualiza a variável p até o final da lista*/
    for (p = l; p != NULL; p = p->prox) {

        /*Imprime os dados de cada paciente*/

        printf("%d - %s - %s\n", p->numChegada, p->nome,p->cor);
    }

}
```

-> Função de remover um paciente já atendido da lista(atendido):

Remove o primeiro elemento da lista, que corresponde ao paciente com prioridade máxima, a cada vez que é chamada e retorna a lista atualizada.

```c
/*Função que retira um paciente que já foi atendido da lista de espera*/

Paciente* atendido(Paciente* lista) {

    /*p guarda referência para o primeiro elemento da lista*/
    Paciente* p = lista;

    /*A lista agora começa do próximo elemento*/
    lista = p->prox;

    /*A memória ocupada pelo primeiro elemento é liberada*/
    free(p);

    /*Retorna a lista atualizada*/
    return lista;
}
```

-> Função de verificar se a lista está vazia (listaVazia):

Retorna 0 se a lista não for NULL, ou seja, vazia. Caso contrário retorna 1.

```c
/*Função que verifica se a lista de pacientes está vazia*/

int listaVazia(Paciente* l) {

    /*Se o primeiro ponteiro for NULL, a lista está vazia e o retorno é 0.
     Caso contrário retorna 1*/

    if (l != NULL) {
        return 0;
    }

    return 1;
}
```

-> Função de liberar a memória ocupada pela lista (liberaLista):

Percorre a lista com 2 ponteiros, 1 para guardar o próximo elemento(t) e o outro que percorre a lista (p), e libera a memória ocupada por p e faz t apontar para t->prox.

```c
/*Função que libera a memória ocupada pela lista de pacientes*/

void liberaLista(Paciente* l) {

    /*Ponteiros para guardar o próximo elemento e percorrer a lista*/
    Paciente* p = l, * t;

    /*Loop percorre a lista até o final*/

    while (p != NULL) {

        /*t guarda a referência para o próximo elemento */
        t = p->prox;

        /*Libera a memória ocupada pelo elemento*/
        free(p);

        /*Ponteiro p recebe a referência de t e continua o loop*/
        p = t;
    }

}
```

->Função de converter a cor da prioridade para um número (numCor):

Função criada com o propósito de facilitar a ordenação por prioridade de cor. Para cada cor passada como parâmetro (Vermelha, Amarela ou Verde) atribui um valor para cada uma delas de acordo com a prioridade (respectivamente , 1, 2 e 3).

```c
/*Função que correlaciona a cor da prioridade do paciente com um número para facilitar a ordenação da fila de espera de atendimento*/

int numCor(char* c) {

    /*Atribuindo um valor numérico a cada cor seguindo a seguinte regra:
      Vermleha : Prioridade máxima , menor número para ser inserido no começo da lista
      Amarela : Prioridade média
      Verde : Priodidade baixa, maior némero para ser inserido no final da lista*/

    if (strcmp(c, "Vermelha") == 0) {
        return 1;
    }

    else if (strcmp(c, "Amarela") == 0) {
        return 2;
    }

    else {
        return 3;
    }
}
```

-> Função para contar quantos pacientes de cada prioridade temos na lista (contaCor):

Percorre a lista e compara cada tipo de prioridade usando a função numCor e para cada prioridade tem uma variável que funciona como um contador que adiciona 1 conforme sua respectiva prioridade é encontrada na busca pela lista.

```c
/*Função que conta a quantidade de pacientes por prioridade de cor*/

void contaCor(int* contaVermelho, int* contaAmarelo, int* contaVerde, Paciente* lista) {

    /*Ponteiro para percorrer a lista*/
    Paciente* p;

    /*Variáveis de contadores para cada prioridade*/

    int contaV = 0, contaA = 0, contaG = 0;

    /*Loop para contar cada prioridade até o final da lista*/

    for (p = lista; p != NULL; p = p->prox) {
        if (numCor(p->cor) == 1) {
            contaV++;
        }
        else if (numCor(p->cor) == 2) {
            contaA++;
        }
        else {
            contaG++;
        }
    }

    /*Guardando o valor de cada contador nas variáveis passadas como parâmetros por referência*/

    *contaVermelho = contaV;
    *contaAmarelo = contaA;
    *contaVerde = contaG;
}
```

-> Função para remover paciente por cor (removePacientePorCor):

A função usa 2 ponteiros para percorrer a lista, a guarda o endereço do elemento anterior e p percorre a lista guardando o endereço do próximo elemento. A busca começa até a lista acabar e enquanto a cor procurada não for encontrada, para isso a função numCor é usada para comparar seus índices inteiros.

Se o paciente for o primeiro elemento, a lista começa do próximo elemento a ele, se não for, fazemos o elemento anterior apontar para o próximo do elemento que achamos e caso o elemento não for encontrado a função retornará NULL.

No final de todos os casos a memória ocupada pelo paciente encontrado é liberada (free(p)) e a função retorna a lista já atualizada.

```c
/*Função que remove o primeiro paciente de uma determinada cor*/

Paciente* removePacientePorCor(Paciente* l, int cor) {

    /*a guarda a referência para o elemento anterior e p percorre a lista*/

    Paciente* p=l,*a=NULL;

    /*Até o final da lista e até acharmos a primeira posição da prioridade do paciente
      guardamos os ponteiros anteriores e andamos com a lista*/

    while (p != NULL && numCor(p->cor) < cor) {
        a = p;
        p = p->prox;
    }

    /*Caso a lista chegue até o final e não encontrarmos a cor, a função retorna NULL*/

    if (p == NULL) {
        return NULL;
    }

    /*Caso o elemento procurado seja o primeiro, a lista começa do próximo elemento*/

    if (a == NULL) {
        l = p->prox;
    }

    /*Se o elemento estiver no meio da lista, o elemento anterior a ele aponta para seu próximo */

    else {
        a->prox = p->prox;
    }

    /*Libera a memória ocupada pelo elemento encontrado*/

    free(p);
    free(p->nome);
    free(p->cor);

    /*Retorna a lista atualizada*/

    return l;
}
```

->Função de remover paciente por nome (retiraPacienteNome):

Esta função funciona de maneira similar a função removePacientePorCor, sendo a única diferença é a condição de comparação na busca, pois a função usa strcmp para comparar os nomes até encontrar o nome do paciente desejado. O tipo de retorno é o mesmo da função removePacientePorNome.

```c
/*Função que remove um paciente de um determinado nome */

Paciente* retiraPacienteNome(Paciente* l, char* n) {

    /*a guarda a referência para o elemento anterior e p percorre a lista*/

    Paciente* p = l, * a = NULL;

    /*Até o final da lista e até acharmos o nome do paciente
      guardamos os ponteiros anteriores e andamos com a lista*/

    while (p != NULL && strcmp(p->nome,n) != 0) {
        a = p;
        p = p->prox;
    }

    /*Caso a lista chegue até o final e não encontrarmos o nome, a função retorna NULL*/

    if (p == NULL) {
        return NULL;
    }

    /*Caso o elemento procurado seja o primeiro, a lista começa do próximo elemento*/

    if (a == NULL) {
        l = p->prox;
    }

    /*Se o elemento estiver no meio da lista, o elemento anterior a ele aponta para seu próximo */

    else {
        a->prox = p->prox;
    }

    /*Libera a memória ocupada pelo elemento encontrado*/

    free(p);
    free(p->nome);
    free(p->cor);

    /*Retorna a lista atualizada*/

    return l;
}
```

Solução:

Usei as bibliotecas <stdio.h>,para usar nas chamadas de printf e scanf, <stdlib.h>,para fazer alocações dinâmicas de memória , <string.h> para facilitar a manipulação de strings e importei a biblioteca "hospital.h" contendo a definição do TAD usado para guardar as informações sobre os pacientes e a definição de todas as funções listadas no item acima que estão presentes no módulo “hospital.c”.

Na função main temos 4 variáveis int que funcionam como contadores. qtdPaciente conta quantos pacientes temos na fila de espera do hospital, enquanto as outras 3 variáveis guardam a quantidade de pacientes em cada prioridade por meio da função contaCor. Temos também outra variável int para controlar o input de dados chamada pedido, caso a variável seja = 0, o loop while para obter os dados de cada paciente para e a lista de pacientes é finalizada, enquanto o valor de pedido = 1 o input de dados continua.

Duas variáveis do tipo char* (strings) com alocação estática de memória são usadas para guardar a prioridade e o nome,respectivamente, de cada paciente dentro do loop while de input de dados do usuário.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "hospital.h"


int main(void) {

    /*Variáveis inteiras que guardam contadores e a condição de parada do loop*/

    int qtdPaciente = 0, pedido = 1, contaVermelho = 0, contaAmarelo = 0, contaVerde = 0;

    /*Variáveis de char para guardar a prioridade e nome de cada paciente*/

    char cor[51],nome[81];

    Paciente* lista;

    /*Inicializando a lista de pacientes */

    lista = criaLista();

    /*Começando o cadastro de pacientes*/

    printf("Deseja ser atendido? sim (1) ou nao (0): \n");

    scanf("%d", &pedido);

    /*Enquanto tiver pacientes requisitando atendimento, adicionamos todos na lista de espera por ordem de prioridade de cor
    Neste mesmo loop, contamos a quantidade de pacientes a serem atendidos*/
```

Um ponteiro para paciente é inicializado usando a função criaLista e uma mensagem é mostrada na tela para indicar que o usuário inicie o imput de dados informando se deseja ser atendido (digita 1 e o valor de pedido = 1) ou não (digita 0 e o valor de pedido = 0).

Dentro do loop, o usuário irá digitar sua prioridade e nome e esses dados ,juntamente com o contador de pacientes, serão inseridos na lista. No final de cada inserção adicionamos +1 ao contador de pacientes e exibimos a mensagem de atendimento para continuar ou parar o loop.

Para continuarmos o programa, o teste de lista vazia é feito.Se a lista não estiver vazia, continuamos com a execução, caso contrário a execução para.

```c
while (pedido != 0) {

    /*Guardando a prioridade do paciente*/

    printf("Digite sua prioridade (Vermelha,Amarela ou Verde):\n");

    scanf("%s", cor);

    /*Guardando o nome de cada paciente*/

    printf("Digite seu nome:\n");

    scanf("%s", nome);

    /*Inserindo o paciente na lista.
     Sua posição na fila é o valor do contador da quantidade de pacientes, que inicialmente é 0,
     acrescido de 1*/

    lista = insereNaLista(cor, qtdPaciente + 1, nome, lista);

    /*Acrescentando 1 ao contador de pacientes*/

    qtdPaciente++;

    /*Mensagem para continuar o input do usuário*/

    printf("Deseja ser atendido? sim (1) ou nao (0): \n");

    scanf("%d", &pedido);
}
```

Para o item a, a lista de pacientes é exibida em ordem de prioridade e chegada por prioridade usando a função imprimeLista e também usamos a função contaCor que retorna quantos pacientes temos em cada prioridade, no final exibimos a lista completa e quantos pacientes temos em cada prioridade.

```c
/*Se a lista de espera não for vazia, o atendimento começa*/

if (listaVazia(lista) == 0) {

    /*a) mostrar a lista de pacientes gerada computacionalmente,nao ordena por num so por cor*/

    printf("Lista de atendimento:\n");

    /*Contamos a quantidade de pacientes em cada prioridade na lista de espera*/

    contaCor(&contaVermelho, &contaAmarelo, &contaVerde, lista);

    printf("\n");

    /*Imprime a lista e a quantidade de pacientes em cada prioridade*/

    imprimeLista(lista);

    printf("\n");

    printf("%d pacientes com prioridade vermelha %d pacientes com prioridade amarela e %d pacientes com prioridade verde\n", contaVermelho, contaAmarelo, contaVerde);

    printf("\n");
```

No item b, o paciente com maior prioridade foi atendido. Para isso, chamamos a função “atendido” que remove o primeiro elemento da lista. Após a chamada da função imprimimos a lista atualizada usando a função imprimeLista() e contamos quantos pacientes restam em cada prioridade com a função contaCor(), exibindo seu resultado após a chamada.

```c
/*Parte b) Paciente com maior prioridade foi atendido */
printf("Paciente com maior prioridade atendido\n");
/*Remove o primeiro paciente da lista*/
lista = atendido(lista);
contaCor(&contaVermelho, &contaAmarelo, &contaVerde, lista);
printf("\n");
/*Imprime a lista e a quantidade de pacientes em cada prioridade*/
imprimeLista(lista);
printf("\n");
printf("%d pacientes com prioridade vermelha %d pacientes com prioridade amarela e %d pacientes com prioridade verde\n", contaVermelho, contaAmarelo, contaVerde);
printf("\n");
```

Já no item c, o primeiro paciente amarelo foi transferido para outro consultório, para isso a função removePacientePorCor foi chamada para atualizar a lista de pacientes e para passar o inteiro correspondente a cor amarela como seu parâmetro, chamamos a função numCor e a string “Amarela” como parâmetro. Logo após, temos a chamada da função contaCor novamente e exibimos a lista atualizada e as informações de cada paciente por prioridade geradas pela contaCor.

```c
/*Parte c) primeiro paciente amarelo foi enviado para outro consultório*/
printf("Paciente amarelo foi para outro consultorio\n");
/*Remove o primeiro paciente amarelo*/
lista = removePacientePorCor(lista, numCor("Amarela"));
contaCor(&contaVermelho, &contaAmarelo, &contaVerde, lista);
printf("\n");
/*Imprime a lista e a quantidade de pacientes em cada prioridade*/
imprimeLista(lista);
printf("\n");
printf("%d pacientes com prioridade vermelha %d pacientes com prioridade amarela e %d pacientes com prioridade verde\n", contaVermelho, contaAmarelo, contaVerde);
printf("\n");
```

O item d) é similar ao começo do programa, onde há um loop while para que os usuários digitem sua prioridade e seu nome para serem inseridos na lista de pacientes do hospital.Nele, novos pacientes chegam ao hospital e há um novo loop while seguindo as mesmas condições e passos do loop inicial para adicioná-los à lista de espera.

```c
/*Parte d)Chegada de novos pacientes*/

/*Mensagem de início do input de dados pelo Usuário*/

printf("Deseja ser atendido? sim (1) ou nao (0): \n");

scanf("%d", &pedido);

while (pedido != 0) {

    /*Guardando a prioridade do paciente*/

    printf("Digite sua prioridade (Vermelha,Amarela ou Verde):\n");

    scanf("%s", cor);

    /*Guardando o nome de cada paciente*/

    printf("Digite seu nome:\n");

    scanf("%s", nome);

    /*Inserindo o paciente na lista.
    Sua posição na fila é o valor do contador da quantidade de pacientes acrescido de 1*/

    lista = insereNaLista(cor, qtdPaciente + 1, nome, lista);

    /*Acrescentando 1 ao contador de pacientes*/

    qtdPaciente++;

    /*Mensagem para continuar o input do usuário*/

    printf("Deseja ser atendido? sim (1) ou nao (0): \n");

    scanf("%d", &pedido);
}
```

```c
/*Contamos a quantidade de pacientes em cada prioridade na lista de espera*/

contaCor(&contaVermelho, &contaAmarelo, &contaVerde, lista);

printf("Lista com novos pacientes\n");

printf("\n");

/*Imprime a lista e a quantidade de pacientes em cada prioridade*/

imprimeLista(lista);

printf("\n");

printf("%d pacientes com prioridade vermelha %d pacientes com prioridade amarela e %d pacientes com prioridade verde\n", contaVermelho, contaAmarelo, contaVerde);

printf("\n");
```

No item e) dois pacientes na prioridade são atendidos, para isso, um loop for é iniciado para que funcione duas vezes e dentro desse loop chamamos a função “atendido” que atualiza a lista quando o primeiro paciente é removido. Para cada remoção, a lista é exibida juntamente com a quantidade de pacientes em cada prioridade usando as funções imprimeLista e contaCor.

```c
/*Parte e) Atender 2 pacientes na prioridade*/

/*Loop que funciona 2 vezes para atender os 2 pacientes*/

for(int i=0;i<2;i++){

    /*Retira o primeiro paciente da lista*/

    lista = atendido(lista);

    /*Conta quantos pacientes temos em cada prioridade*/

    contaCor(&contaVermelho, &contaAmarelo, &contaVerde, lista);

    /*Exibe a quantidade de pacientes atendidos, que é o contador da quantidade de loops,
    que inicialmente é 0, acrescido de 1*/

    printf("%d pacientes na prioridade foram atendidos\n",i+1);

    printf("\n");

    /*Exibe a lista atualizada com a quantidade de pacientes em cada prioridade a cada loop*/

    imprimeLista(lista);

    printf("\n");

    printf("%d pacientes com prioridade vermelha %d pacientes com prioridade amarela e %d pacientes com prioridade verde\n", contaVermelho, contaAmarelo, contaVerde);

    printf("\n");
}
```

Por fim, no item f), o paciente Felipe na prioridade verde desiste do atendimento, Assim usamos a função retiraPacienteNome para removê-lo da lista. Com a lista atualizada, usamos as funções imprimeLista e contaCor para exibir a lista atualizada retornada pela função retiraPacienteNome e a quantidade de pacientes em cada prioridade.

```c
    /*Parte f) Paciente Felipe (verde) desiste*/

    /*Retira o paciente de nome Felipe da lista*/

    lista = retiraPacienteNome(lista,"Felipe");

    /*Conta quantos pacientes tem em cada prioridade*/

    contaCor(&contaVermelho, &contaAmarelo, &contaVerde, lista);

    /*Imprime a lista atualizada após a remoção do paciente com a quantidade de pacientes por prioridade*/

    printf("Paciente Felipe(Verde) desistiu\n");

    printf("\n");

    imprimeLista(lista);

    printf("\n");

    printf("%d pacientes com prioridade vermelha %d pacientes com prioridade amarela e %d pacientes com prioridade verde\n", contaVermelho, contaAmarelo, contaVerde);

    printf("\n");

    /*No final do atendimento, liberamos a memória ocupada pela lista de espera*/

    liberaLista(lista);
}

return 0;
```

No fim, a função main retorna 0 se funcionar corretamente.

Conclusões:

Uma parte difícil foi implementar a ordenação da lista de acordo com o número de chegada e também o jeito de implementar o input de dados do usuário com um loop foi um pouco complicado, pois fiquei em dúvida em qual condição de parada do loop era mais adequada para usar.

A função de tirar um paciente por nome e cor foi uma dificuldade em relação às condições de comparação.

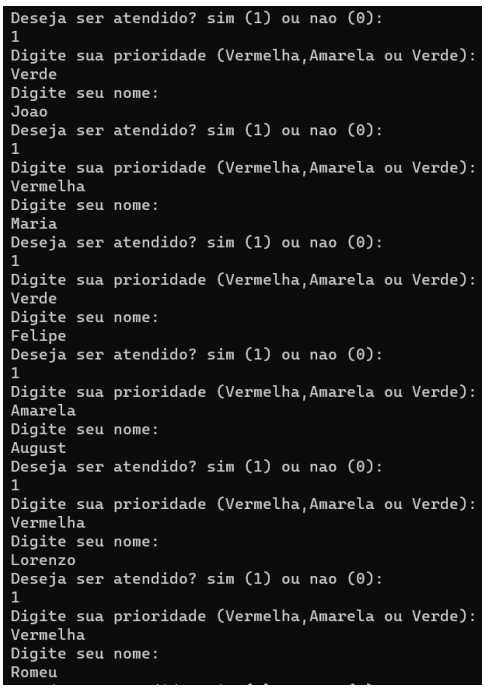
Em relação ao resto do programa não tive maiores dificuldades.

Observações:

A função retiraPacienteNome remove o paciente de acordo com o nome, apenas.

As outras funções funcionam normalmente.

Resultados:



```
Deseja ser atendido? sim (1) ou nao (0):
1
Digite sua prioridade (Vermelha,Amarela ou Verde):
Verde
Digite seu nome:
Carlos
Deseja ser atendido? sim (1) ou nao (0):
1
Digite sua prioridade (Vermelha,Amarela ou Verde):
Vermelha
Digite seu nome:
Clara
Deseja ser atendido? sim (1) ou nao (0):
0
Lista de atendimento:

2 - Maria - Vermelha
5 - Lorenzo - Vermelha
6 - Romeu - Vermelha
8 - Clara - Vermelha
4 - August - Amarela
1 - Joao - Verde
3 - Felipe - Verde
7 - Carlos - Verde

4 pacientes com prioridade vermelha 1 pacientes com prioridade amarela e 3 pacientes com prioridade verde

Paciente com maior prioridade atendido

5 - Lorenzo - Vermelha
6 - Romeu - Vermelha
8 - Clara - Vermelha
4 - August - Amarela
1 - Joao - Verde
3 - Felipe - Verde
7 - Carlos - Verde

3 pacientes com prioridade vermelha 1 pacientes com prioridade amarela e 3 pacientes com prioridade verde
```

```
Paciente amarelo foi para outro consultorio

5 - Lorenzo - Vermelha
6 - Romeu - Vermelha
8 - Clara - Vermelha
1 - Joao - Verde
3 - Felipe - Verde
7 - Carlos - Verde

3 pacientes com prioridade vermelha 0 pacientes com prioridade amarela e 3 pacientes com prioridade verde

Deseja ser atendido? sim (1) ou nao (0):
1
Digite sua prioridade (Vermelha,Amarela ou Verde):
Verde
Digite seu nome:
Homer
Deseja ser atendido? sim (1) ou nao (0):
1
Digite sua prioridade (Vermelha,Amarela ou Verde):
Amarela
Digite seu nome:
Renzo
Deseja ser atendido? sim (1) ou nao (0):
1
Digite sua prioridade (Vermelha,Amarela ou Verde):
Vermelha
Digite seu nome:
Joaquim
Deseja ser atendido? sim (1) ou nao (0):
1
Digite sua prioridade (Vermelha,Amarela ou Verde):
Amarela
Digite seu nome:
Manoela
```

```
Lista com novos pacientes

5 - Lorenzo - Vermelha
6 - Romeu - Vermelha
8 - Clara - Vermelha
11 - Joaquim - Vermelha
10 - Renzo - Amarela
12 - Manoela - Amarela
1 - Joao - Verde
3 - Felipe - Verde
7 - Carlos - Verde
9 - Homer - Verde

4 pacientes com prioridade vermelha 2 pacientes com prioridade amarela e 4 pacientes com prioridade verde

1 pacientes na prioridade foram atendidos

6 - Romeu - Vermelha
8 - Clara - Vermelha
11 - Joaquim - Vermelha
10 - Renzo - Amarela
12 - Manoela - Amarela
1 - Joao - Verde
3 - Felipe - Verde
7 - Carlos - Verde
9 - Homer - Verde

3 pacientes com prioridade vermelha 2 pacientes com prioridade amarela e 4 pacientes com prioridade verde

2 pacientes na prioridade foram atendidos

8 - Clara - Vermelha
11 - Joaquim - Vermelha
10 - Renzo - Amarela
12 - Manoela - Amarela
1 - Joao - Verde
3 - Felipe - Verde
7 - Carlos - Verde
9 - Homer - Verde
```

```
2 pacientes com prioridade vermelha 2 pacientes com prioridade amarela e 4 pacientes com prioridade verde

Paciente Felipe(Verde) desistiu

8 - Clara - Vermelha
11 - Joaquim - Vermelha
10 - Renzo - Amarela
12 - Manoela - Amarela
1 - Joao - Verde
7 - Carlos - Verde
9 - Homer - Verde

2 pacientes com prioridade vermelha 2 pacientes com prioridade amarela e 3 pacientes com prioridade verde
```