Fila com Prioridade em C

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Estrutura do paciente

struct Paciente {

char nome[50];

int idade;

struct Paciente \*prox;

};

// Função para inserir paciente com prioridade

void enfileirarComPrioridade(struct Paciente \*\*inicio, struct Paciente \*\*fim) {

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

printf("Digite o nome do paciente (sem espaços): ");

scanf("%s", novo->nome);

printf("Digite a idade do paciente: ");

scanf("%d", &novo->idade);

novo->prox = NULL;

if (\*inicio == NULL) {

\*inicio = novo;

\*fim = novo;

return;

}

if (novo->idade >= 60) {

novo->prox = \*inicio;

\*inicio = novo;

printf("Paciente PRIORITÁRIO inserido no início da fila.\n");

} else {

(\*fim)->prox = novo;

\*fim = novo;

printf("Paciente comum inserido no fim da fila.\n");

}

}

// Atender paciente (sempre do início da fila)

void atenderPaciente(struct Paciente \*\*inicio, struct Paciente \*\*fim) {

if (\*inicio == NULL) {

printf("Fila vazia! Nenhum paciente para atender.\n");

return;

}

struct Paciente \*temp = \*inicio;

printf("Atendendo: %s (idade %d)\n", temp->nome, temp->idade);

\*inicio = temp->prox;

free(temp);

if (\*inicio == NULL) {

\*fim = NULL;

}

}

// Mostrar todos os pacientes

void mostrarFila(struct Paciente \*inicio) {

if (inicio == NULL) {

printf("Fila vazia.\n");

return;

}

printf("Fila de pacientes (prioritários primeiro):\n");

while (inicio != NULL) {

printf("- %s (idade %d)\n", inicio->nome, inicio->idade);

inicio = inicio->prox;

}

}

// Função principal com menu

int main() {

struct Paciente \*inicio = NULL;

struct Paciente \*fim = NULL;

int opcao;

do {

printf("\n===== MENU - FILA COM PRIORIDADE =====\n");

printf("1. Inserir paciente na fila\n");

printf("2. Atender próximo paciente\n");

printf("3. Mostrar fila\n");

printf("0. Sair\n");

printf("Escolha uma opcao: ");

scanf("%d", &opcao);

if (opcao == 1) {

enfileirarComPrioridade(&inicio, &fim);

} else if (opcao == 2) {

atenderPaciente(&inicio, &fim);

} else if (opcao == 3) {

mostrarFila(inicio);

} else if (opcao != 0) {

printf("Opcao invalida. Tente novamente...\n");

}

} while (opcao != 0);

// Liberar memória restante

while (inicio != NULL) {

atenderPaciente(&inicio, &fim);

}

return 0;

}

Fila com Prioridade com Duas Filas

1. Inserir paciente (com prioridade)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Estrutura do paciente

struct Paciente {

char nome[50];

int idade;

struct Paciente \*prox;

};

// Inserir paciente na fila correta (prioritária ou comum)

void inserirPaciente(struct Paciente \*\*inicioPrioritaria, struct Paciente \*\*fimPrioritaria,

struct Paciente \*\*inicioComum, struct Paciente \*\*fimComum) {

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

printf("Nome do paciente (sem espaços): ");

scanf("%s", novo->nome);

printf("Idade do paciente: ");

scanf("%d", &novo->idade);

novo->prox = NULL;

if (novo->idade >= 60) {

// Inserir na fila prioritária

if (\*inicioPrioritaria == NULL) {

\*inicioPrioritaria = novo;

\*fimPrioritaria = novo;

} else {

(\*fimPrioritaria)->prox = novo;

\*fimPrioritaria = novo;

}

printf("Paciente PRIORITÁRIO inserido!\n");

} else {

// Inserir na fila comum

if (\*inicioComum == NULL) {

\*inicioComum = novo;

\*fimComum = novo;

} else {

(\*fimComum)->prox = novo;

\*fimComum = novo;

}

printf("Paciente COMUM inserido!\n");

}

}

2. Atender paciente (fila prioritária primeiro)

void atenderPaciente(struct Paciente \*\*inicioPrioritaria, struct Paciente \*\*fimPrioritaria,

struct Paciente \*\*inicioComum, struct Paciente \*\*fimComum) {

struct Paciente \*temp;

if (\*inicioPrioritaria != NULL) {

temp = \*inicioPrioritaria;

printf("Atendendo (PRIORITÁRIO): %s (idade %d)\n", temp->nome, temp->idade);

\*inicioPrioritaria = temp->prox;

free(temp);

if (\*inicioPrioritaria == NULL) {

\*fimPrioritaria = NULL;

}

} else if (\*inicioComum != NULL) {

temp = \*inicioComum;

printf("Atendendo (COMUM): %s (idade %d)\n", temp->nome, temp->idade);

\*inicioComum = temp->prox;

free(temp);

if (\*inicioComum == NULL) {

\*fimComum = NULL;

}

} else {

printf("Nenhum paciente na fila.\n");

}

}

---

3. Mostrar fila (com título)

void mostrarFila(const char \*titulo, struct Paciente \*inicio) {

printf("\n%s\n", titulo);

if (inicio == NULL) {

printf("Fila vazia!\n");

return;

}

while (inicio != NULL) {

printf("- %s (idade %d)\n", inicio->nome, inicio->idade);

inicio = inicio->prox;

}

}

4. Função principal (menu)

int main() {

struct Paciente \*inicioPrioritaria = NULL;

struct Paciente \*fimPrioritaria = NULL;

struct Paciente \*inicioComum = NULL;

struct Paciente \*fimComum = NULL;

int opcao;

do {

printf("\n=== MENU - FILA COM PRIORIDADE (DUAS FILAS) ===\n");

printf("1. Inserir paciente na fila\n");

printf("2. Atender próximo paciente\n");

printf("3. Mostrar filas\n");

printf("0. Sair\n");

printf("Escolha uma opcao: ");

scanf("%d", &opcao);

switch (opcao) {

case 1:

inserirPaciente(&inicioPrioritaria, &fimPrioritaria, &inicioComum, &fimComum);

break;

case 2:

atenderPaciente(&inicioPrioritaria, &fimPrioritaria, &inicioComum, &fimComum);

break;

case 3:

mostrarFila("FILA PRIORITÁRIA:", inicioPrioritaria);

mostrarFila("FILA COMUM:", inicioComum);

break;

case 0:

printf("Encerrando...\n");

break;

default:

printf("Opcao inválida!\n");

}

} while (opcao != 0);

// Libera memória restante

while (inicioPrioritaria != NULL || inicioComum != NULL) {

atenderPaciente(&inicioPrioritaria, &fimPrioritaria, &inicioComum, &fimComum);

}

return 0;

}

Lista Duplamente Encadeada em C:

#include <stdio.h>

#include <stdlib.h>

// Estrutura do nó da lista

struct No {

int valor;

struct No \*anterior; // Ponteiro para o nó anterior

struct No \*proximo; // Ponteiro para o próximo nó

};

// Inserir elemento no início da lista

void inserirInicio(struct No \*\*inicio, struct No \*\*fim, int valor) {

struct No \*novo = (struct No \*) malloc(sizeof(struct No));

novo->valor = valor;

novo->anterior = NULL;

novo->proximo = \*inicio;

if (\*inicio != NULL)

(\*inicio)->anterior = novo;

else

\*fim = novo; // Lista estava vazia

\*inicio = novo;

}

// Inserir elemento no fim da lista

void inserirFim(struct No \*\*inicio, struct No \*\*fim, int valor) {

struct No \*novo = (struct No \*) malloc(sizeof(struct No));

novo->valor = valor;

novo->proximo = NULL;

novo->anterior = \*fim;

if (\*fim != NULL)

(\*fim)->proximo = novo;

else

\*inicio = novo; // Lista estava vazia

\*fim = novo;

}

// Remover elemento do início da lista

void removerInicio(struct No \*\*inicio, struct No \*\*fim) {

if (\*inicio == NULL) {

printf("Lista vazia. Nada para remover.\n");

return;

}

struct No \*temp = \*inicio;

printf("Removendo do início: %d\n", temp->valor);

\*inicio = temp->proximo;

if (\*inicio != NULL)

(\*inicio)->anterior = NULL;

else

\*fim = NULL; // Lista ficou vazia

free(temp);

}

// Remover elemento do fim da lista

void removerFim(struct No \*\*inicio, struct No \*\*fim) {

if (\*fim == NULL) {

printf("Lista vazia. Nada para remover.\n");

return;

}

struct No \*temp = \*fim;

printf("Removendo do fim: %d\n", temp->valor);

\*fim = temp->anterior;

if (\*fim != NULL)

(\*fim)->proximo = NULL;

else

\*inicio = NULL; // Lista ficou vazia

free(temp);

}

// Mostrar lista do fim ao início

void mostrarInverso(struct No \*fim) {

if (fim == NULL) {

printf("Lista vazia.\n");

return;

}

printf("Lista do fim ao início: ");

while (fim != NULL) {

printf("%d ", fim->valor);

fim = fim->anterior;

}

printf("\n");

}

int main() {

struct No \*inicio = NULL;

struct No \*fim = NULL;

int opcao, valor;

do {

printf("\n====== MENU - LISTA DUPLAMENTE ENCADEADA ======\n");

printf("1. Inserir no início\n");

printf("2. Inserir no fim\n");

printf("3. Remover do início\n");

printf("4. Remover do fim\n");

printf("5. Mostrar do fim ao início\n");

printf("6. Sair\n");

printf("Digite a opção: ");

scanf("%d", &opcao);

switch (opcao) {

case 1:

printf("Digite o valor: ");

scanf("%d", &valor);

inserirInicio(&inicio, &fim, valor);

break;

case 2:

printf("Digite o valor: ");

scanf("%d", &valor);

inserirFim(&inicio, &fim, valor);

break;

case 3:

removerInicio(&inicio, &fim);

break;

case 4:

removerFim(&inicio, &fim);

break;

case 5:

mostrarInverso(fim);

break;

case 6:

printf("Encerrando...\n");

break;

default:

printf("Opção inválida.\n");

}

} while (opcao != 6);

// Liberar todos os nós restantes

while (inicio != NULL) {

struct No \*temp = inicio;

inicio = inicio->proximo;

free(temp);

}

return 0;

}

Função: Remover elemento por valor na lista duplamente encadeada

// Remove o primeiro nó que contém o valor especificado

void removerPorValor(struct No \*\*inicio, struct No \*\*fim, int valor) {

if (\*inicio == NULL) {

printf("Lista vazia. Nada para remover.\n");

return;

}

struct No \*atual = \*inicio; // Ponteiro auxiliar para percorrer a lista

// Percorre a lista até encontrar o valor ou chegar ao fim

while (atual != NULL && atual->valor != valor) {

atual = atual->proximo;

}

// Se percorreu tudo e não encontrou o valor

if (atual == NULL) {

printf("Valor %d não encontrado na lista.\n", valor);

return;

}

// Se o nó a ser removido é o único da lista

if (atual == \*inicio && atual == \*fim) {

\*inicio = NULL;

\*fim = NULL;

}

// Se o nó está no início

else if (atual == \*inicio) {

\*inicio = atual->proximo;

(\*inicio)->anterior = NULL;

}

// Se o nó está no fim

else if (atual == \*fim) {

\*fim = atual->anterior;

(\*fim)->proximo = NULL;

}

// Nó está no meio

else {

atual->anterior->proximo = atual->proximo;

atual->proximo->anterior = atual->anterior;

}

printf("Valor %d removido da lista.\n", atual->valor);

free(atual); // Libera a memória do nó removido

}

Atualização no menu e no switch

Adicionar no menu:

printf("7. Remover por valor\n");

Adicionar no switch:

case 7:

printf("Digite o valor a ser removido: ");

scanf("%d", &valor);

removerPorValor(&inicio, &fim, valor);

break;

Implementação de Fila com Vetor

#include <stdio.h>

#include <stdlib.h>

#define TAM 5 // Define o tamanho máximo da fila

// Estrutura da fila

struct fila {

int dados[TAM]; // Vetor fixo para armazenar os elementos da fila

int inicio; // Índice do primeiro elemento

int fim; // Índice onde será inserido o próximo elemento

};

// Inicializa a fila (coloca início e fim em 0)

void inicializarFila(struct fila \*f) {

f->inicio = 0;

f->fim = 0;

}

// Verifica se a fila está vazia

int filaVazia(struct fila \*f) {

return f->inicio == f->fim;

}

// Verifica se a fila está cheia

int filaCheia(struct fila \*f) {

return f->fim == TAM;

}

// Insere elemento na fila

void inserirNaFila(struct fila \*f, int valor) {

if (filaCheia(f)) {

printf("Fila cheia! Não é possível inserir.\n");

return;

}

f->dados[f->fim] = valor;

f->fim++;

printf("Inserido %d na fila.\n", valor);

}

// Remove elemento da fila

void desenfileirar(struct fila \*f) {

if (filaVazia(f)) {

printf("Fila vazia! Nada para remover.\n");

return;

}

printf("Removido: %d\n", f->dados[f->inicio]);

f->inicio++;

}

Função principal com menu da fila

int main() {

struct fila f; // Declara uma variável do tipo fila

int opcao, valor;

inicializarFila(&f); // Inicializa a fila

do {

printf("\n====== MENU - FILA COM VETOR ======\n");

printf("1. Inserir na fila\n");

printf("2. Remover da fila\n");

printf("3. Mostrar fila\n");

printf("4. Sair\n");

printf("Digite a opção: ");

scanf("%d", &opcao);

switch (opcao) {

case 1:

printf("Digite o valor a ser inserido: ");

scanf("%d", &valor);

inserirNaFila(&f, valor);

break;

case 2:

desenfileirar(&f);

break;

case 3:

mostrarFila(&f);

break;

case 4:

printf("Encerrando o programa...\n");

break;

default:

printf("Opção inválida!\n");

}

} while (opcao != 4);

return 0;

}

Definições e inicialização da Fila Circular

#include <stdio.h>

#include <stdlib.h>

#define TAM 5 // Define o tamanho máximo da fila

// Estrutura da fila circular

struct fila {

int dados[TAM];

int inicio;

int fim;

};

// Inicializa a fila (coloca início e fim em 0)

void inicializarFila(struct fila \*f) {

f->inicio = 0;

f->fim = 0;

}

Verificações da fila

// Verifica se a fila está vazia

int filaVazia(struct fila \*f) {

return f->inicio == f->fim;

}

// Verifica se a fila está cheia

int filaCheia(struct fila \*f) {

return ((f->fim + 1) % TAM) == f->inicio;

}

Inserção na fila circular

void enfileirar(struct fila \*f, int valor) {

if (filaCheia(f)) {

printf("Fila cheia! Não é possível inserir.\n");

return;

}

f->dados[f->fim] = valor;

f->fim = (f->fim + 1) % TAM;

printf("Elemento %d inserido com sucesso!\n", valor);

}

Remoção da fila circular

void desenfileirar(struct fila \*f) {

if (filaVazia(f)) {

printf("Fila vazia! Nada para remover.\n");

return;

}

int removido = f->dados[f->inicio];

f->inicio = (f->inicio + 1) % TAM;

printf("Elemento %d removido com sucesso!\n", removido);

}

Mostrar fila circular

void mostrarFila(struct fila \*f) {

if (filaVazia(f)) {

printf("Fila vazia!\n");

return;

}

printf("Fila: ");

int i = f->inicio;

while (i != f->fim) {

printf("%d ", f->dados[i]);

i = (i + 1) % TAM;

}

printf("\n");

}

Função principal com menu

// Função principal com menu

int main() {

struct fila fila; // Declara a fila

inicializarFila(&fila); // Inicializa a fila

int opcao, valor;

do {

// Menu de opções

printf("\n===== MENU - FILA CIRCULAR COM VETOR =====\n");

printf("1. Inserir elemento\n");

printf("2. Remover elemento\n");

printf("3. Mostrar fila\n");

printf("4. Sair\n");

printf("Escolha uma opção: ");

scanf("%d", &opcao);

switch (opcao) {

case 1:

printf("Digite o valor a ser inserido: ");

scanf("%d", &valor);

enfileirar(&fila, valor);

break;

case 2:

desenfileirar(&fila);

break;

case 3:

mostrarFila(&fila);

break;

case 4:

printf("Encerrando o programa...\n");

break;

default:

printf("Opção inválida. Tente novamente.\n");

}

} while (opcao != 4); // Repete até o usuário escolher sair

return 0;

}

Lista Duplamente Encadeada Circular.

Estrutura e Inserção na Lista

#include <stdio.h>

#include <stdlib.h>

// Estrutura do nó da fila

struct No {

int valor;

struct No \*anterior; // Ponteiro para o nó anterior

struct No \*proximo; // Ponteiro para o próximo

};

// Inserir elemento no fim da fila

void enfileirar(struct No \*\*inicio, struct No \*\*fim, int valor) {

struct No \*novo = (struct No\*)malloc(sizeof(struct No)); // Cria novo nó

novo->valor = valor;

novo->anterior = NULL;

novo->proximo = NULL;

if (\*inicio == NULL) {

// Se a fila estiver vazia, o novo nó é o único

\*inicio = novo;

\*fim = novo;

novo->proximo = novo;

novo->anterior = novo;

} else {

// Ajusta os ponteiros para adicionar no fim da fila

novo->anterior = \*fim;

novo->proximo = \*inicio;

(\*inicio)->anterior = novo;

(\*fim)->proximo = novo;

\*fim = novo; // Atualiza o ponteiro fim

}

printf("Elemento %d inserido na fila.\n", valor);

}

Remover Elemento do Início da Fila

// Remover elemento do início da fila

void desenfileirar(struct No \*\*inicio, struct No \*\*fim) {

if (\*inicio == NULL) {

printf("Fila vazia! Nada para remover.\n");

return;

}

struct No \*removido = \*inicio; // Guarda o nó a ser removido

if (\*inicio == \*fim) {

// Fila com apenas um elemento

\*inicio = NULL;

\*fim = NULL;

} else {

\*inicio = removido->proximo;

(\*inicio)->anterior = \*fim;

(\*fim)->proximo = \*inicio;

}

printf("Elemento %d removido da fila.\n", removido->valor);

free(removido); // Libera memória

}

Mostrar a Fila

// Mostrar a fila do início até o fim (uma volta completa)

void mostrarFila(struct No \*inicio) {

if (inicio == NULL) {

printf("Fila vazia.\n");

return;

}

struct No \*atual = inicio;

printf("Fila atual: ");

do {

printf("%d ", atual->valor); // Mostra o valor do nó atual

atual = atual->proximo;

} while (atual != inicio); // Para quando der uma volta completa

printf("\n");

}

Função Principal com Menu

// Função principal: com menu

int main() {

struct No \*inicio = NULL; // Ponteiro para o início da fila

struct No \*fim = NULL; // Ponteiro para o fim da fila

int opcao, valor;

do {

printf("===== MENU - FILA DUPLAMENTE CIRCULAR =====\n");

printf("1. Inserir elemento\n");

printf("2. Remover elemento\n");

printf("3. Mostrar fila\n");

printf("4. Remover por valor\n");

printf("0. Sair\n");

printf("Escolha uma opcao: ");

scanf("%d", &opcao);

switch (opcao) {

case 1:

printf("Digite o valor a ser inserido: ");

scanf("%d", &valor);

enfileirar(&inicio, &fim, valor); // Insere no fim

break;

case 2:

desenfileirar(&inicio, &fim); // Remove do início

break;

case 3:

mostrarFila(inicio); // Mostra a fila

break;

case 4:

printf("Digite o valor a ser removido: ");

scanf("%d", &valor);

removerPorValor(&inicio, &fim, valor);

break;

case 0:

printf("Encerrando o programa.\n");

break;

default:

printf("Opcao invalida. Tente novamente.\n");

}

} while (opcao != 0); // Continua até a opção ser 0

return 0;

}

Remover um Nó pelo Valor

// Remove o primeiro nó que contém o valor especificado

void removerPorValor(struct No \*\*inicio, struct No \*\*fim, int valor) {

if (\*inicio == NULL) {

printf("Lista vazia. Nada para remover.\n");

return;

}

struct No \*atual = \*inicio; // Começa a busca pelo primeiro nó

int encontrou = 0;

// Percorre a lista circular até encontrar ou completar uma volta

do {

if (atual->valor == valor) {

encontrou = 1;

break;

}

atual = atual->proximo;

} while (atual != \*inicio);

if (!encontrou) {

printf("Valor %d não encontrado na lista.\n", valor);

return;

}

if (\*inicio == \*fim && atual == \*inicio) {

// Só há um nó na lista

\*inicio = NULL;

\*fim = NULL;

} else if (atual == \*inicio) {

// Atual é o primeiro nó da lista

\*inicio = atual->proximo;

(\*inicio)->anterior = \*fim;

(\*fim)->proximo = \*inicio;

} else if (atual == \*fim) {

// Atual é o último nó

\*fim = atual->anterior;

(\*fim)->proximo = \*inicio;

(\*inicio)->anterior = \*fim;

} else {

// Nó está no meio

atual->anterior->proximo = atual->proximo;

atual->proximo->anterior = atual->anterior;

}

printf("Valor %d foi removido da lista.\n", atual->valor);

free(atual); // Libera a memória do nó removido

}