MINISTÉRIO DA EDUCAÇÃO SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA TRIÂNGULO MINEIRO

Curso: Licenciatura em Computação

Disciplina: Estruturas de Dados / fila Professor: Edson Angoti Júnior

Instruções para entrega: 1. Postar os códigos dos exercícios no Github 2. Entregar a tarefa no Google Classroom indicando o link do repositório Github

1. Criar um sistema de cadastro e atendimento de pedidos que tem o seguinte menu:

[1] novo pedido (lê do teclado código numérico e insere na fila)

[2] atender (retirar da fila e mostra na tela e guarda numa outra lista)

[3] listar novos pedidos (não atendidos)

[4] listar pedidos atendidos

[0] sair

2. Suponha que o Beco do Pirão (Praça Tiradentes, Ouro Preto), durante a noite, seja usado como um estacionamento que guarda até 10 carros. Os carros entram pela Praça Tiradentes (PT) e saem pela Rua Barão de Camargos (RBC) (obs: fato fictício gerado a partir de informações extraídas de maps.google.com). Se chegar um cliente para retirar um carro que não esteja estacionado na primeira da RBC, todos os carros entre o carro do cliente e a RBC serão deslocados para fora do estacionamento, o carro do cliente sairá do estacionamento e os outros carros voltarão a entrar pela PT na mesma ordem que saíram pela RBC. Observe que sempre que um carro deixa o estacionamento, todos os carros entre ele e a PT serão deslocados até o começo da RBC de modo que, o tempo inteiro, todos os espaços vazios estão na entrada do estacionamento, ou seja na entrada pela PT. Escreva um programa que leia um grupo de linhas de entrada. Cada linha contém um 'C', de chegada, e um 'P' de partida, além de um número de placa de licenciamento. Presume- se que os carros chegarão e partirão na ordem especificada pela entrada. O programa deve imprimir uma mensagem cada vez que um carro chegar ou partir. Quando um carro chegar e existir vaga, estacione o carro na vaga e imprima a mensagem "Carro estacionado". Se não existir vaga imprima a mensagem "Não há vagas". Quando um carro partir, a mensagem deverá incluir o número de vezes que o carro foi deslocado dentro do estacionamento, incluindo a própria partida, mas não a chegada. Esse número será 0 se o carro for embora a partir da linha de espera.

3. Implemente uma fila usando pilhas. Uma fila pode ser implementada usando duas pilhas. Deixe a fila a ser implementada como q e as pilhas usadas para implementar q sejam stack1 e stack2. q pode ser implementado de duas maneiras:

Método 1 (Tornando a operação enQueue dispendiosa) Esse método garante que o

elemento inserido mais antigo esteja sempre no topo da pilha 1, de modo que a operação deQueue simplesmente seja exibida na pilha1. Para colocar o elemento no topo da pilha1, pilha2 é usada.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <string.h>

#define TAMANHO 3
struct fila {
    int itens[TAMANHO]; //vetor para armazenar os elementos da fila
    int posicao;       //indica o primeiro da fila no vetor
    int qtdeElem;      //indica a quantidade de elementos na fila
};

typedef struct fila Fila;

Fila* criarFila() {
    Fila* fila = (Fila*) malloc(sizeof(Fila));
    if(fila!=NULL){
        fila->posicao=0;
        fila->qtdeElem=0;
    }
    return fila;
}

int tamanhoFila(Fila* fila) {
    return fila->qtdeElem;
}
```

```
void imprimirFila(Fila* fila) {
    printf("\nFila: ");
    int i = fila->posicao;
    int contador;
    for(contador=0;contador<fila->qtdeElem;contador++){
        printf("%i ",fila->itens[i]);
        i = (i+1)%TAMANHO;//lembre-se do "vetor circular"
    }
    printf("\n");
    getch();
}

int inserir(Fila* fila, int item){
    if(fila->qtdeElem>=TAMANHO)
       return 0;
    int posicao=(fila->posicao+fila->qtdeElem)%TAMANHO;
    fila->itens[posicao]=item;
    fila->qtdeElem++;
    return 1;
}

int retirarDaFila(Fila* fila){
    if(fila->qtdeElem==0)
        return 0;
    int elemento = fila->itens[fila->posicao];
    fila->posicao = (fila->posicao+1)%TAMANHO;
    fila->qtdeElem--;
    return elemento;
}
```

```
void entrada(Fila *fila) {
    int placa;
    printf("\nDigite a placa do carro (apenas números): ");
    scanf("%i",&placa);
    int cod = inserir(fila,placa);
    if(cod==1){
        printf("Carro estacionado.");
    } else {
        printf("Não existe vaga.");
    }
}

void saida(Fila *fila) {
    int placaSaindo,placa;
    printf("\nDigite a placa do carro (apenas números): ");
    scanf("%i",&placaSaindo);
    int tamanho = tamanhoFila(fila);
    int contador = 0;
    for(;contador<tamanho;contador++){
        placa = retirarDaFila(fila);
        if(placa==placaSaindo){
                        printf("Carro saindo do estacionamento com %i
deslocamentos",contador);
            break;
        } else {
            inserir(fila,placa);
            imprimirFila(fila);
        }
    }
```

```
}


int main() {
    setlocale(LC_ALL,"");
    Fila* estacionamento = criarFila();
    char op;
    while(1) {
        printf("\n\n==> ");
        scanf(" %c",&op);
        switch(op){
            case 'C':
            case 'c': entrada(estacionamento);break;
            case 'P':
            case 'p': saida(estacionamento);break;
            case '0': exit(0);
        }
    }
}
```

enQueue (q, x)

1) Enquanto a pilha 1 não está vazia, empilhe tudo da pilha 1 para a pilha 2.

2) Empilhe x na pilha 1.

3) Empilhe tudo de volta para a pilha 1.

dnQueue(q)

1) Se a pilha 1 estiver vazia, então imprima mensagem de erro.

2) Desempilhe um item da pilha 1 e o retorne.

Método 2 (Tornando a operação deQueue cara) Nesse método, na operação enqueue, o novo elemento é inserido no topo da pilha 1. Na operação da retirada da fila, se a pilha 2 estiver vazia, todos os elementos serão movidos para a pilha 2 e, finalmente, a parte superior da pilha 2 será retornada.

enQueue (q, x)

1) Empilhe x na pilha 1.

deQueue(q)

1) Se ambas as pilhas estiverem vazias, então imprima mensagem de erro.

2) Se a pilha 2 estiver vazia

Enquanto a pilha 1 não estiver vazia, empilhe tudo da pilha1 para pilha2.

3) Desempilhe o elemento da pilha 2 e o retorne.

```
#include <stdlib.h>
#include <stdio.h>


#define TAMANHO 100
struct pilha {
    int topo;
    char item[TAMANHO];
};


Pilha* criapilha(){
    Pilha* p = (Pilha*) malloc(sizeof(Pilha));
    if(p!=NULL)
        p->topo = 0;
    return p;
}
```

```
void liberar(Pilha* p){
    free(p);
}

void push(Pilha* p, char item){
    p->item[(p->topo)++] = item;
}

char pop(Pilha* p){
    if (empty(p)) {
        printf("pilha vazia");
        exit(1);
    }
    return(p->item[--(p->topo)]);
}

int empty(Pilha* p){
    return (p->topo <= 0);
}
//1) Se a pilha 1 estiver vazia, então imprima mensagem de erro.
//2) Desempilhe um item da pilha 1 e o retorne.
void dequeue(){
    int aux;
    if(empty(pilha1))
        printf("erro");
    else {
        return pop(pilha1);
    }
```

```
}

/*
enQueue (q, x)
 1) Enquanto a pilha 1 não está vazia, empilhe tudo da pilha 1 para a pilha 2.
 2) Empilhe x na pilha 1.
 3) Empilhe tudo de volta para a pilha 1.
*/
void enqueue(int x){
    int aux;
    while(!empty(pilha1)){
        aux = pop(pilha1);
        push(pilha2,aux);
    }
    push(pilha1,x);
    while(!empty(pilha2){
        aux = pop(pilha2);
        push(pilha1,aux);
    }
}

Pilha *pilha1 = criapilha();
Pilha *pilha2 = criapilha();

int main() {

}
```