Atividade Lista, Pilha e Fila

Estrutura de Dados I

Lista Ligada

Considere a lista ligada dinâmica (inserção ordenada). Desejamos transformar essa estrutura em uma estrutura duplamente ligada (isto é, cada elemento apontará para seu anterior e também para seu sucessor). Em termos de modelagem, haverá apenas uma modificação na estrutura `ELEMENTO`, que receberá um campo adicional chamado de `ant`:

```
typedef int TIPOCHAVE;

typedef struct {
    TIPOCHAVE chave;
    // outros campos...
} REGISTRO;

typedef struct aux {
    REGISTRO reg;
    struct aux *ant, *prox;
} ELEMENTO;

typedef ELEMENTO* PONT;

typedef struct {
    PONT inicio;
} LISTA;
```

Duas funções precisam ser alteradas para que todos os ponteiros da estrutura fiquem consistentes: as funções de inserção e exclusão. Para esta atividade, revise/atualize a função de inserção considerando os seguintes casos:

- Se o elemento a ser inserido for o primeiro da lista, o campo `ant` dele deverá ser `NULL`.

- Caso contrário, o campo `ant` deverá apontar para o elemento anterior.
- O elemento seguinte ao novo deverá ter seu campo `ant` atualizado para apontar para o novo elemento.

A assinatura da função é:

```
bool inserirElemListaOrd(LISTA* l, REGISTRO reg);
```

Observe que, no site da disciplina, já está disponível todo o código da presente estrutura. No entanto, a função de inserção não considera que a lista é duplamente ligada, sendo necessário alterá-la para atualizar o valor do campo `ant` do novo elemento que está sendo inserido. Esta será uma inserção sem repetição, ou seja, se um elemento com a mesma chave já existir, a função deverá retornar `false`.

Além disso, será necessário atualizar o campo `ant` do elemento que, na lista ordenada duplamente ligada, será o sucessor do novo elemento (caso este sucessor exista).

Entregue código completo revisado. A estrutura não será circular e não terá nó-cabeça.

Pilha Estática

Considere a pilha (implementação estática). Em pilhas, muitas vezes não são implementadas funções de busca por elementos com uma determinada chave ou mesmo funções para exibir as chaves dos registros da estrutura. No entanto, o código fornecido apresenta uma função chamada `exibirPilha`, que exibe as chaves dos elementos da pilha, do topo até sua base.

Para esta atividade semanal, implemente a função `exibirPilhaInvertida`, que exibe as chaves dos elementos da pilha, partindo da base até o topo. O esqueleto da função já está disponível no código fornecido para esta semana (no site da disciplina). Basta completá-lo:

```
void exibirPilhaInvertida(PILHA* p) {
    printf("Pilha (da base para o topo): \" ");
    /* COMPLETE A FUNCAO */
    printf("\"\n");
}
```

O código base está disponível no classroom. Complete o código para implementar a funcionalidade.

Deque e Fila

Considere as estruturas apresentadas.

Deque

Em estruturas do tipo DEQUE, muitas vezes não são implementadas funções de busca por elementos com uma determinada chave ou mesmo funções para exibir as chaves dos registros da estrutura. No entanto, é possível implementar esse tipo de funcionalidade.

Para esta atividade semanal, implemente a função `encontrarMax`, que encontrará a maior chave dentro do DEQUE e copiará essa chave para o endereço de memória passado como parâmetro.

O esqueleto da função é:

```
bool encontrarMax(DEQUE* d, int* max) {
    bool resposta = false;
    /* COMPLETAR */
    return resposta;
}
```

Esta função deve:

- Retornar `false` caso o DEQUE não contenha nenhum elemento válido (apenas o nó-cabeça).
- Copiar o valor da maior chave dos elementos válidos do DEQUE para a memória apontada por `max` e retornar `true`.

Fila com Nó-Cabeça

Considere a fila (implementação dinâmica) apresentada na aula. A implementação utilizada anteriormente não incluía um nó-cabeça. Conforme discutido, uma vantagem de utilizar um nó-cabeça é garantir que todos os elementos válidos sempre terão um anterior, e a estrutura nunca ficará vazia (poderá não conter nenhum elemento "válido", mas não será considerada vazia).

Para esta atividade, adapte as funções `inserirNaFila` e `excluirDaFila` para considerar a existência de um nó-cabeça. Esta estrutura não será circular nem duplamente ligada. Apenas será adicionado um nó-cabeça (durante a inicialização), e as funções de inserção e exclusão deverão considerar a existência desse elemento, que nunca deverá ser excluído.

O esqueleto das funções é:

```
bool inserirNaFila(FILA* f, REGISTRO reg) {
    /* COMPLETAR - REVISAR o código desta função */
    PONT novo = (PONT) malloc(sizeof(ELEMENTO));
    novo->reg = reg;
    novo->prox = NULL;
    if (f->inicio == NULL) {
        f->inicio = novo;
    } else {
        f->fim->prox = novo;
    }
    f->fim = novo;
    return true;
}

bool excluirDaFila(FILA* f, REGISTRO* reg) {
    /* COMPLETAR - REVISAR o código desta função */
    if (f->inicio == NULL) {
        return false;
    }
    *reg = f->inicio->reg;
    PONT apagar = f->inicio;
    f->inicio = f->inicio->prox;
    free(apagar);
    if (f->inicio == NULL) {
        f->fim = NULL;
    }
    return true;
}
```

Estas funções respeitam a assinatura e a funcionalidade da inserção e exclusão em filas: os elementos são inseridos no final da fila e excluídos do início.

Instruções de Entrega

A entrega deve ser realizada via GitHub. Crie um repositório público para esta atividade e adicione todos os arquivos necessários, incluindo o código-fonte e outros materiais solicitados.

Após criar o repositório, envie o link público do repositório através do Classroom, na tarefa correspondente. Certifique-se de que o repositório está acessível e bem organizado.