Introdução à Computação II (Noturno)
BCC – Unesp Rio Claro/SP 2015

Lista Obrigatória 02 - Prof. Rafael Oliveira

(Deve ser entregue **em PDF via Moodle: Escolha apenas 5 exercícios para entrega**)

Exercício 01 (Pilhas)

Faça um programa que contenha um menu que apresente as opções abaixo:

1 – Cadastar número
2 – Mostrar todos os números pares cadastrados
3 – Mostrar todos os números ímpares cadastrados
4 – Mostrar todos os números cadastrados
5 – Excluir número
6 – Sair

Observações:

a) Implemente usando uma estrutura tipo pilha.

b) Quando a opção 1 do menu for solicitada, só deve parar de cadastrar quando o usuário pressionar um número negativo.

c) Mostrar mensagem para opção invalidade do menu

Exercício 02 (Pilhas)

Faça um programa que apresente o menu de opções abaixo:

1 – Cadastrar aluno
2 – Cadastrar nota
3 – Calcular a média de um aluno
4 – Listar os nomes dos alunos sem notas
5 – Excluir aluno
6 – Excluir nota
7 – Sair

Observações:

a) Cadastre um aluno (número e nome) de cada vez em uma pilha. Os números dos alunos devem ser gerados automaticamente, partindo do nº 1.

b) Cadastre uma nota (número do aluno e nota) de cada vez em uma fila. Uma nota só pode ser cadastrada se pertencer a um aluno cadastrado na pilha de alunos. Mostrar mensagem caso o aluno não esteja cadastrado. Valide as notas para que sejam digitadas no intervalo entre 0 e 10.

c) O usuário deve digitar o número de aluno para exibir seu nome e sua respectiva média. Caso não existir o aluno ou não houver notas cadastradas para ele emitir mensagem de aviso ao usuário.

d) Um aluno só pode ser excluído se não possuir notas. O usuário não deve escolher o aluno a ser excluído, pois deve obedecer as regras de funcionamento da estrutura de dados tipo pilha.

e) As notas devem ser excluídas respeitando as regras de funcionamento de uma estrutura

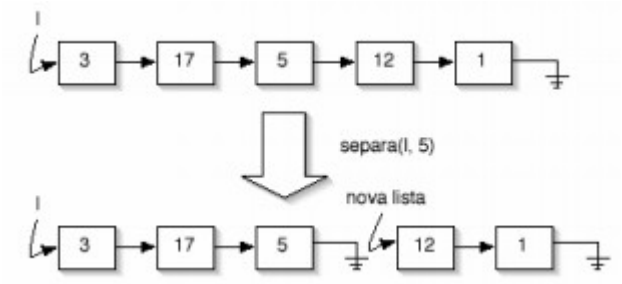
Exercício 03 (Filas)

Implemente um programa em C que utiliza a estrutura de fila dinâmica de elementos do tipo RG (num, nome, dataNascimento). O programa deve mostrar ao usuario um menu permanente com quatro opções. Se o usuário escolher 1, a fila deve ser impressa por completa; se escolher 2, ele deve entrar com o valor do conteudo do novo elemento da fila. Se escolher 3, um elemento da fila deve ser removido. Se escolher 4, o programa encerra.

Adicionalmente, implemente também a opção 5 do menu: que aciona uma outra rotina adicional chamada "inverter" que reposicione os elementos em uma fila F de forma que o início se torne o fim e vice-versa

Exercício 04 (Listas Encadeadas)

A função "Lista* separa (Lista* l, int n);" tem a funcionalidade de receber como parâmetro uma lista encadeada e um valor inteiro n e dividir a lista em duas, de tal forma que a segunda lista comece no primeiro nó logo após a primeira ocorrência de "n" na lista original. A figura a seguir ilustra essa separação:



A função retorna um ponteiro para a segunda sub-divisão da lista original, enquanto "l" deve continuar apontando para o primeiro elemento da primeira subdivisão da lista.
Abaixo, é possível averiguar a implementação de um possível código para implementar a função:

```
/* função separa */
Lista* separa (Lista* l, int n)
{
Lista* p; /* variável auxiliar para percorrer a lista */
Lista* q; /* variável auxiliar para nova lista */
for (p = l; p != NULL ; p = p->prox) {
        if (p->info == n)
        {       q = p->prox;
                p->prox = NULL;
                return q;
        }}
return NULL;
}
```

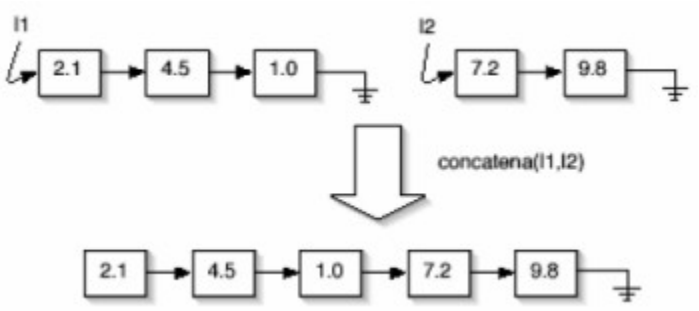
Implemente um programa de uma lista encadeada completa (com todas as operações) na qual seja possível testar e avaliar o funcionamento da função separa.

Exercício 05 (Listas encadeadas)

Considere estruturas de listas encadeadas que armazenam valores reais. Como, por exemplo, a estrutura mostrada abaixo:

```
struct lista {
   float info;
   struct lista* prox;
};
typedef struct lista Lista;
```

Faça o que é requisitado:
→ Implemente um programa que contenha uma estrutura de lista encadeada completa para gerenciar a struct descrita (inserir, remover, zerar, iniciar, etc).
→ Implemente e teste uma função, que dadas duas listas encadeadas L1 e L2, concatene a lista L2 no final da lista L1, conforme é apresentado na figura abaixo.



A função deve retornar a lista resultante da concatenação, obedecendo a seguinte assinatura:
"Lista* concatena (Lista* l1, Lista l2);"

Dica de implementação: percorra "l1" até encontrar o final da lista. Então, faça o prox do final da "l1" apontar para "l2". Retorne "l1"

Exercício 06 (Ordenação)

Implemente um programa para avaliar/comparar os tempos de execução dos seguintes algoritmos de ordenação: (1) Seleção, (2) Inserção, (3) Shellsort, (4) QuickSort e (5) BublleSort.

Dica 1: Crie uma rotina para criar vetores com 1000 valores gerados randomicamente (valores entre 0 e 1000).
Dica 2: Pesquisa e crie uma rotina para calcular o tempo de execução de funções em milissegundos.
Dica 3: Implementa cada método de ordenação em uma função diferente.

Exercício 07 (Ordenação)

Faça um código para replicar as comparações implementadas no execício anterior para os seguintes casos:
→ vetor totalmente ordenado
→ vetor totalmente desordenado

Exercício 8 (Ordenação)

Considere a execução do algoritmo Quicksort visto em sala (pivô escolhido no meio do vetor) com o vetor [2 10 1 8 20 5 14 13 9].

Implemente um programa que execute o QuickSort para esse vetor e imprima em tela:

→ O número de chamadas do procedimento "Partição" para ordenar esse vetor
→ Qual o pivô utilizado em cada uma das chamadas
→ Quais são as partições (sub-vetores) resultantes de cada uma dessas chamadas

Exercício 9 (Busca)

Escreva um programa em C para ler um vetor A de dimensão definida pelo usuários e realizar uma Busca Sequencial no vetor de um elemento fornecido pelo usuário. O vetor deve conter uma lista de produtos a serem cadastrados com os seguitnes campos: Código(int), Nome (String) e Fornecedor (String). Produtos não podem ser cadastrados com um mesmo código.

As buscas podem ser feitas de modo sequencial, utilizando qualquer um dos campos do produto. Buscas em campos que sejam strings não devem ser case sensitive. O programa deve conter ainda uma função para escrever todos os produtos que estão cadastrados.

Exercício 10 (Busca)

Refaça o Exercício 12 utilizando busca binária. Dessa vez, tais buscas devem ser feitas somente utilizando o campo Código. Obviamente, o vetor de produtos deve estar ordenado. Portanto, antes de fazer a busca, é necessário utilizar um algoritmo de ordenação para garantir sua efetividade.

Exercício 11 (Árvores de busca)

Considere uma árvore binária de busca que armazena valores inteiros. Nesta estrutura, pode ocorrer repetições de um mesmo valor. Assim, os valores associados aos nós das sub-árvores à esquerda são menores que o valor associado à raiz e os valores da sub-árvore direita são maiores ou iguais:
→ implemente a árvore e todas suas possíveis operações que podem ser acessadas pelo usuário por meio de um menu. Incluindo: leitura, remoção, busca e escrita todos os itens;
→ Adicione no menu, uma opção que permita acessar uma função que retorne o número de ocorrências de um dado "X" na árvore. A função deve tirar proveito da estrutura da árvore e deve receber como parâmetro um ponteiro para a árvore e o valor inteiro buscado;
→ Adicione no menu uma opção que permita acessar uma função que imprima todos os nós folhas da árvore.

Exercício 12 (Árvores de busca)

Escreva um programa que crie uma árvore de busca binária a partir de TADs de alunos lidas do teclado. A TAD alunos contém os seguintes campos: RA, Nome e Curso. O programa deve imprimir a árvore nos três modos de percurso. Portanto, o programa deve permitir: cadastro de alunos, busca de alunos pelo RA e impressão de todos os alunos.