UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA

Disciplina: [1107186] Estrutura de Dados
Prof. Christian Azambuja Pagot.
Data:
Nome: _____ Matrícula: _____

Prova I

Instruções:

- Esta prova tem duração de 2 horas.
- A prova é individual e sem consulta.
- Antes de iniciar a prova, escreva seu nome e número de matrícula nas folhas da prova e de respostas.
- A interpretação dos enunciados faz parte da verificação.
- A prova pode ser feita a lápis, porém as respostas devem estar a caneta.
- É vedado o uso de equipamentos eletrônicos (celulares, calculadoras, etc.) durante o período da prova.
- As questões que envolvem cálculos devem apresentar todo o desenvolvimento dos mesmos.
- Questões que envolvem a escrita de funções e de programas devem apresentar o código completo (nome de funções, parâmetros, tipos, valores de retorno, declaração de variáveis, etc.).

Questão 1 (2,0 pts.)

Suponha uma lista simplesmente encadeada com *n* nós, onde cada nó é defindo pela seguinte *struct*:

```
struct Node
{
     int value;
     Node* next;
}
```

Assumindo que a memória ocupada por cada nó foi previamente alocada através da função *malloc*(...), escreva uma função na linguagem C que libere toda a memória ocupada pela lista encadeada. O único parêmetro desta função é o ponteiro do primeiro nó da lista. Lembre-se que o ponteiro *next* do último nó da lista aponta para NULL.

O protótipo da função a ser desenvolvida é apresentado abaixo:

```
void DestroiLista(Node* h);
```

Questão 2 (2,5 pts.)

Suponha uma lista simplesmente encadeada, onde cada nó é representado pela mesma *struct* da *Questão 1* e *h* representa o ponteiro para o primeiro nó desta lista.

Escreva um programa que inverta, com auxílio de uma pilha, a direção de uma lista simplesmente encadeada. Após a operação de inversão, o último nó da lista passa a ser o primeiro, e todos os ponteiros *next* devem apontar para o nó anterior da lista. O algoritmo deve **obrigatoriamente** apresentar **custo *O*(*n*)**.

Segue abaixo o protótipo da função a ser desenvolvida (onde ***h*** é o ponteiro para o primeiro nó

da lista a ser invertida):

```
void InverteLista(Node** h);
```

Questão 3 (3,0 pts.)

Você irá construir um dicionário para inserir alguns ítens. A chave *k* de cada item é formada por um par de letras maiúsculas. As únicas letras maiúsculas permitidas na construção das chaves são 'A' (ASCII 65), 'B' (ASCII 66), 'C (ASCII 67) e 'D' (ASCII 68), totalizando 16 possibilidades.

Você decidiu implementar este dicionário através de uma tabela *hash e* resolveu o problema de colisão utilizando encadeamento (*chainning*). Entretanto, devido a problemas de espaço, a tabela *hash* possuirá apenas 8 *buckets*, e como você deseja que as consultas tenham tempo constante, você determina que não deverão haver mais do que duas chaves mapeadas em cada *bucket*.

Considerando o cenário descrito acima, descreva um algoritmo de *hash f*(*k*), onde *k* é a chave, que mapeie as 16 possíveis chaves para a tabela *hash* sem que mais do que 2 chaves sejam mapeadas para o mesmo *bucket*.

Questão 4 (2,5 pts.)

A inserção em tabelas de *hash* pode causar colisões, ou seja, a tentativa de se inserir em uma mesma posição da tabela dois ítems com um mesmo valor de *hash*. Quando isto ocorre, deve-se utilizar alguma técnica de tratamento de colisão. Considerando-se o conjunto ordenado (da esquerda para a direita) de valores {10, 6, 4, 17, 8, 5, 14}, uma tabela de *hash* com 7 posições, e a função de *hash h(x) = x mod 7*, responda:

4.1 - Qual será o estado final da tabela de *hash* após a inserção dos valores na ordem dada, utilizando-se *separate chaining* para o tratamento das colisões. Explique.

4.2 - Qual será o estado final da tabela de hash após a inserção dos valores na ordem dada, utilizando-se *linear probing* para o tratamento das colisões. Explique.