Universidade Federal de Pernambuco
Cin – Centro de Informática

Prof: Adriano Sarmento
Data: 08/11/2011
Data de Entrega: 22/11/2011

Considerações:

- **É Proibido usar a biblioteca conio.h;**
- **Leia a lista toda o quanto antes, para evitar más interpretações e muitas dúvidas em cima da hora;**
- **Envie uma prévia da lista, pelo menos um dia antes da data final de entrega, para o caso de acontecer algum imprevisto;**
- **A lista é para ser feita individualmente. Qualquer tentativa de cópia acarretará o zeramento da lista de todos os envolvidos;**
- **Em caso de dúvida, envie email para listaip@googlegroups.com;**
- **Atenção para a liberação de memória no final dos programas. Será cobrado que o espaço alocado no decorrer do programa seja totalmente liberado no final do mesmo.**

Sexta Lista – IP/Eng. Da Computação – 2011.2

`Questão 1)` Joãozinho é viciado em Batalha Naval, e precisa de material para estudar esse jogo. Então, ele pediu aos alunos de IP para montar um simulador de partidas, mostrando a ordem dos eventos que ocorreram.

Mas como Joãozinho também é programador, ele, para facilitar, já implementou o gerador de jogadas, então só restou aos alunos receber a saida do gerador, e imprimir num arquivo o que ocorreu na partida. Todo jogo tem um vencedor.
A matriz do jogo deve ser implementada usando uma lista encadeada simples de listas encadeadas simples, onde cada nó armazena um valor booleano que indica se há um navio naquela posição.

A entrada(**L6Q1.in**) consistirá de:

```
T       //Um inteiro dizendo o tamanho TxT do tabuleiro.
P       //Um inteiro dizendo quantos navios cada jogador terá.
N D1 X1 Y1 D2 X2 Y2    //P linhas cada uma contendo as descrições dos navios
.       //dos dois jogadores: Um inteiro N dizendo quantas casas os dois navios
        //terão, a direção D
.       //que os navios estarão (string: "vertical" ou "horizontal") e as
        //coordenadas X e Y da posição
.       //da sua posição mais à esquerda ( horizontal) ou à cima (vertical) dos
        //navios, primeiro o do jogador 1, depois do jogador 2.
N D1 X1 Y1 D2 X2 Y2
X Y     //coordenadas, no formato X Y, de onde o jogador da vez jogou uma bomba,
        //começando pelo jogador 1.
.       // Até o jogo acabar.
.
.
X Y
//FIM DA ENTRADA
```

Saída(**L6Q1.out**):

```
Jogador J jogou uma bomba em X Y e acertou S
.       //Qual jogador J, onde X e Y ele jogou uma bomba, e o que ele
.       //acertou(string S: "agua" ou "um navio")
.       // Para cada jogada deve ser impressa uma linha.
.        //OBS: uma casa previamente afundada é considerada AGUA.
.
Jogador J jogou uma bomba em X Y e acertou S
Jogador J ganhou  //após todos os navios de um jogador serem abatidos, o outro
                  //jogador ganha.
//FIM DA SAIDA

Saída:
Jogador J jogou uma bomba em X Y e acertou S
.       // Qual jogador J, onde X e Y ele jogou uma bomba, e o que ele
.       //acertou(string: "agua" ou "um navio")
.       // Para cada jogada deve ser impressa uma linha.
.
.
Jogador J jogou uma bomba em X Y e acertou S
Jogador J ganhou       //após todos os navios de um jogador serem abatidos, o
                       //outro jogador ganha.
//FIM DA SAIDA
```

Questão 2) A Torre de Hanói é um "quebra-cabeça" que consiste em uma base contendo três pinos, em um dos quais são dispostos alguns discos uns sobre os outros, em ordem crescente de diâmetro, de cima para baixo. O problema consiste em passar todos os discos de um pino para outro qualquer, usando um dos pinos como auxiliar, de maneira que um disco maior nunca fique em cima de outro menor em nenhuma situação. O número de discos pode variar sendo que o mais simples contém apenas três.
Sabendo disso, crie um software que simule uma torre de Hanói com um número variável de discos. Cada pino da torre será representada por uma pilha de discos, onde cada disco é um elemento da pilha, armazenando um inteiro, recebido pelo usuário e a referência para o disco abaixo dele.

A cada movimento, será impressa a configuração atual das 3 torres.

OBS: As pilhas devem ser implementadas utilizando o conceito de listas encadeadas.

A entrada(**L6Q2.in**) consistirá de:

```
N                 //Número de discos
X1                //Tamanho do menor disco
X2                //Tamanho do segundo menor disco
.
.
.
XN                //Tamanho do maior disco
                  //Fim da entrada
```

Saída (**L6Q2.out**):

```
Estado inicial:
Torre A: XN XN-1 ... X1 //Primeira torre
Torre B:                //Vazia no estado inicial
Torre C:                //Vazia no estado inicial

Movimento 1:
```

```
Torre A: XN XN-1 ... X2 //Configuração das torres
Torre B: X1             //Após o primeiro movimento
Torre C:

.
.
.

Movimento Z:

Torre A:                //Vazia no estado final
Torre B:                //Vazia no estado final
Torre C: XN XN-1 ... X1 //Completamente movida
```

Questão 3) Um programador fã de Formula 1, após muitas manhãs de domingo sem poder checar o resultado das corridas no seu OS de linha de comando preferido, resolveu criar um programa para poder ficar a par das corridas.
Porém, no meio do desenvolvimento do programa, ele percebeu que, por ser muito atarefado, não conseguiria tempo para completar o programa.
Então, o programador pediu aos alunos de IP de EC-CIn que completassem seu programa.

Até agora, o programa recebe da internet eventos das corridas, e manda através de um arquivo texto comandos para a segunda parte a ser desenvolvida pelos alunos.

Existem 4 comandos do arquivo que foram implementados, cada um em uma linha:

"X Passou Y"
Onde X e Y são corredores, X antes do comando estando logo atrás de Y, e depois do comando X estará a frente de Y.

"Volta"
O corredor mais proximo da largada completa mais uma volta.

"Lider"
Escreve na saída o nome do corredor que lidera a corrida

"X Quebrou"
O carro do corredor X quebrou

Os corredores, segundo especificação do programador, deverão ser estruturas com:

```
Nome( 20 caracteres)
Numero de voltas restantes a ser completadas (inteiro)
```

Postas em uma lista encadeada circular.

Antes de começar os comandos, o arquivo de entrada (**L6Q3.in**) conterá:

```
N       //Um inteiro, que será o numero de corredores.
V       //Um inteiro, que será o numero de voltas da corrida.
NOME1   //E os nomes dos corredores, linha por linha, na ordem em que
NOME2   //estarão dispostos na largada.
.
.
.
```

```
NOMEN
COMANDO1
COMANDO2
COMANDO3        //Comandos
.
.
.
ULTIMOCOMANDO
//FIM DE ARQUIVO
```

Ao quebrar, ou passar pela largada e completar o numero de voltas total, o corredor sai da corrida (e da lista encadeada circular)
O programa encerra quando todos os corredores completarem a corrida ou quebrarem.

No fim do programa, deve ser impresso no arquivo de saída o nome dos corredores, em ordem de chegada, e os nomes dos que tiveram seus carros quebrados, em ordem de quebra.

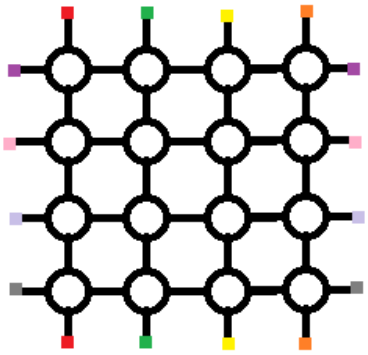
Saida(**L6Q3.out**):

```
NOMEA   //nomes dos corredores liderando a corrida em dados momentos (comando
NOMEB   //"Lider")
NOMEC   //se não houver nenhuma comando Lider, não imprima nenhuma linha nesse
.       //grupo, mas imprima uma linha em branco antes dos vencedores
.
.
NOME@

1 - NOMEDOVENCEDOR1    //X linhas, com os nomes de quem completou a corrida, em
2 - NOMEDOVENCEDOR2    //ordem de chegada.
3 - NOMEDOVENCEDOR3
.
.
.
X - NOMEDOVENCEDORX

Quebrou 1 - NOMEDEQUEMQUEBROU1        //Y linhas, com os nomes de quem
Quebrou 2 - NOMEDEQUEMQUEBROU2        //quebrou, em ordem de quebra.
Quebrou 3 - NOMEDEQUEMQUEBROU3
Quebrou 4 - NOMEDEQUEMQUEBROU4
.
.
.
Quebrou Y - NOMEDEQUEMQUEBROUY
//FIM DE ARQUIVO
```

Questão 4) Uma organização precisa criptografar todos os seus documentos, e você foi contratado para executar essa tarefa. A criptografia funciona de forma parecida com a de um cubo mágico, onde o texto é colocado sobre uma grade e sofre inúmeras rotações, a fim de deixa-lo irreconhecível.
O programa receberá um arquivo de texto (**L6Q4.in**) como entrada, e organizará o mesmo, colocando cada letra do texto em um nó de lista encadeada. Cada linha e coluna do texto será tratada como uma lista duplamente encadeada circular.
Deve haver uma limitação no número de caracteres por linha(80 no máximo), porém o arquivo de entrada pode ter mais, cabendo ao programador tratar o texto de forma adequada. Quando uma linha tiver menos de 80 caracteres, os restantes devem ser preenchidos com o caractere NULL.



As cores representam as ligações nas pontas das listas.

Depois de organizar o texto, o programa deve receber a partir de outro arquivo (**L6Q4.cripto**) a sequência de rotações a ser aplicada no texto. A entrada vai seguir ao seguinte formato:

L[x]C[y]O[z]T[w]

Onde x e y são as coordenadas da posição do nó, z, um inteiro de 0 a 3, informa a orientação da rotação (Cima, Baixo, Esquerda, Direita, respectivamente) e w fornece o número de vezes que a linha ou coluna deve ser rotacionada. Cada linha do arquivo de entrada irá conter apenas **uma** instrução. Ao fim da criptografia, o programa deverá criar um novo arquivo(**L6Q4.out**), contendo o texto criptografado.