Trabalho prático 1

Listas encadeadas recursivas e ordenação

1 Problema

Você foi contratado pelo **Facebook** para construir seu novo motor de comentários em postagens. *Mark Zuckerberg* quer que você implemente tal motor de forma que seja possível responder a comentários de forma recursiva, ou seja, um comentário pode ter diferentes níveis de respostas.

As operações que ele deseja são as seguintes:

- Adicionar um novo comentário à postagem;
- Responder a um comentário com outro comentário;
- Remover um comentário, removendo recursivamente as respostas do mesmo;

2 Tipo abstrato de dados

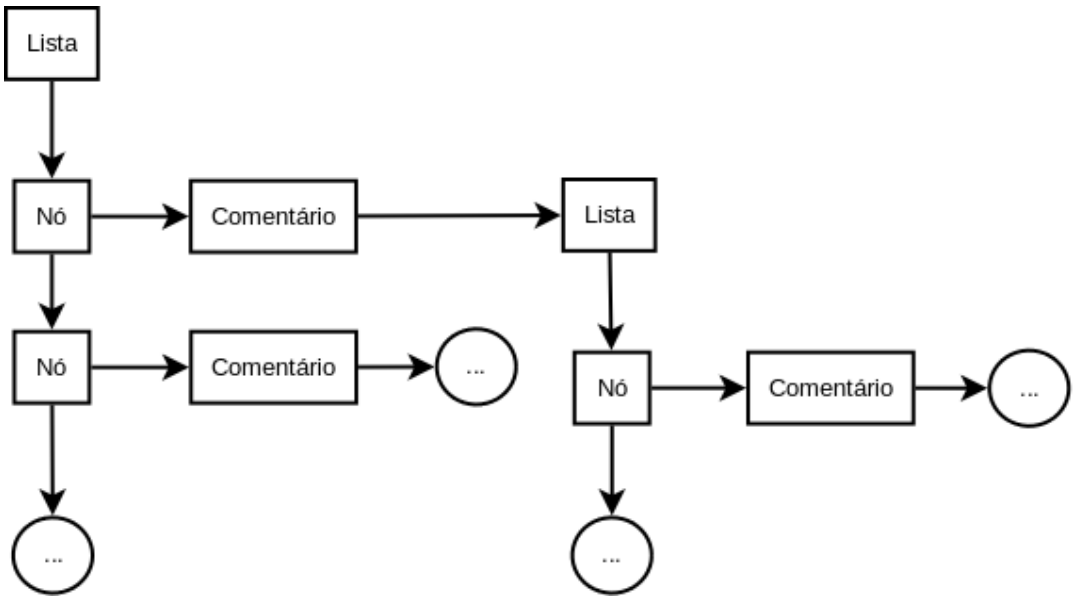

Figura 1: Lista encadeada recursiva

Esse tipo de lista, como apresentado na figura 1, possui nós com apontadores para o próximo nó. Cada nó possui um comentário que pode apontar para um outra lista encadeada, que representa a lista de respostas àquele comentário. A lista-raiz representa os comentários de uma postagem específica. Pode-se usar **opcionalmente** uma célula cabeça (não presente na figura) na lista de forma facilitar as implementações.

Em *C*, pode-se implementar tal estrutura usando as definições do bloco de código 1.

Bloco de código 1: Tipo abstrato de dados. O tipo *TItem* representa o comentário com seu devido identificador e possíveis respostas

```
typedef struct TLista TLista;
typedef struct TCelula TCelula;
typedef struct TItem TItem;

struct TItem {
  unsigned id;
  TLista* respostas;
};

struct TCelula {
  TCelula* proxima;
  TItem* comentario;
};

struct TLista {
  TCelula* primeira;
  TCelula* ultima;
};
```

3 Operações

Nessa seção são reapresentadas as operações a serem implementadas. Um comentário é representado pelo seu **identificador** (*id*). A lista de comandos é apresentada na tabela 1.

Tabela 1: Lista de comandos

Operação	Descrição	Código
Adicionar comentário	Adiciona o comentário com o identificador *id* ao fim da lista de postagens.	A<:id>
Responder a um comentário	Adiciona-se o comentário com identificador *id2* à lista de respostas do comentário com identificador *id1*.	R<:id1>,<:id2>
Remover comentário	Remove-se o comentário com identificador *id* da postagens, removendo recursivamente suas respostas.	D<:id>

Obs.: As operações de *responder* e *remover* devem tratar o caso em que o comentário não existe mais. Quando um comentário não existe mais, não deve haver **nenhum efeito colateral** nos dados. Em um caso real, pode ser que alguém responda a um comentário que já tenha sido apagado pelo autor ou até mesmo tente apagar o mesmo comentário duas vezes.

4 Entradas, saídas e exemplos

A entrada será um arquivo de texto onde cada linha representa os comentários de uma postagem usando os códigos definidos na tabela 1.

A saída será um arquivo de texto onde cada linha representa o estado final dos comentários da postagem em dois formatos: **original** e **ordenado por identificador** de forma ascendente. A ordenação deve ser feita de forma que todas as listas sejam ordenadas **individualmente** e de forma ascendente, do menor identificador para o maior identificador. A saída da lista deve seguir o seguinte algoritmo, tanto para a lista original quanto para a lista ordenada:

```
imprime(comentarios):
  para cada comentario em comentarios:
    imprime o identificador do comentario
    se o comentario possui respostas:
      imprime(respostas do comentario) // etapa recursiva
```

A tabela 2 apresenta alguns tipos de entrada, suas representações na lista recursiva, a lista após ordenação e a saída em texto. O símbolo '–' representa a sublista.

Tabela 2: Exemplos

Entrada	Representação	Representação pós-ordenação	Saída
A2A1R2,4R2,3	2 –4 –3 1	1 2 –3 –4	2 4 3 1 - 1 2 3 4
A2A1R2,4R2,3D2	1	1	1 - 1
A2A1R2,4R2,3D4	2 –3 1	1 2 –3	2 3 1 - 1 2 3

5 *Kit* de desenvolvimento

O projeto possui diversos arquivos para auxiliar na implementação. Ele já define uma estrutura para o projeto. Os arquivos são os seguintes:

- *examples.txt*: 2000 exemplos de entrada gerados de forma aleatória.
- *answers.txt*: A resposta correta de cada um dos 2000 exemplos.
- *examples.py*: Um gerador de exemplos aleatórios. Para esses exemplos você não terá a resposta correta. Para gerá-los use o seguinte comando:

  ```
  make examples
  ```

  Obs.: Precisa-se da linguagem **Python** para executá-lo.

- *test.py*: Um programa que confere se a saída do seu programa no arquivo *output.txt* é igual às respostas corretas no arquivo *answers.txt*. Para executar os testes:

  ```
  make test
  ```

  Obs.: Precisa-se da linguagem **Python** para executá-lo.

- *solution/TLista.h*: O cabeçalho do *TLista*. Aqui está definido o tipo abstrato de dados como apresentado anteriormente.
- *solution/TLista.c*: Implementação das operações da lista definidas no cabeçalho.

- *solution/main.c*: O programa principal. Esse programa deverá receber dois argumentos ao ser executado, o arquivo de entrada e o arquivo de saída, de forma que ele seja executado da seguinte maneira, por exemplo:

  ```
  ./main examples.txt output.txt
  ```

  Para compilar o projeto:

  ```
  make build
  ```

  Para testar o programa gerado com os testes de *examples.txt*:

  ```
  make try
  ```

Todos esses comandos só vão funcionar caso se siga a estrutura proposta pelo *kit* de desenvolvimento e use os nomes de arquivos como os descritos acima. Dê uma olhada no arquivo *Makefile* para descobrir como tudo funciona. De forma a fazer bom uso do *kit*, é necessário que o mesmo seja usado em ambiente *Linux*.

6 Entregáveis

O exercício deve ser implementado na linguagem *C* usando bibliotecas de padrões como *stdio*, *stdlib* e possivelmente *string*. Deve-se entregar todo o código desenvolvido para resolver o problema. Esse código deve ter como arquivo principal o arquivo *main.c*. O *Makefile* deverá ser entregue junto ao código, **principalmente** no caso de alterações do mesmo. O código deve ser organizado e bem comentado.

Também deverá ser entregue uma documentação de **2** a **4** páginas do projeto, contendo os seguintes tópicos:

1. **Introdução**: Fale sucintamente sobre o projeto como um todo, seus desafios e seus resultados gerais.
2. **Desenvolvimento**: Fale como você implementou cada uma das operações e qual a ordem de complexidade de cada uma delas.
3. **Análise experimental** (extra / opcional): Usando exemplos de tamanhos diferentes, teste o tempo de execução do seu programa e verifique se os resultados experimentais estão de acordo com a complexidade teórica do programa.
4. **Conclusão**: Escreva suas conclusões sobre o projeto, suas dificuldades e os resultados obtidos. Responda às seguintes perguntas (extra / opcional):

   Essa estrutura de dados é a estrutura ideal para esse problema? Quais são suas desvantagens? Qual estrutura de dados seria melhor para resolver esse problema?