Informações gerais

Neste laboratório, você será introduzido ao conceito e prática de listas encadeadas simples. A lista encadeada é uma estrutura de dados linear que utiliza ponteiros para ligar sequencialmente seus elementos, ou nós. Cada nó contém dois campos: um para armazenar o dado e outro para armazenar o endereço do próximo nó na lista.

O objetivo desta atividade é refazer a atividade de gerenciamento da ficha bibliográfica do Laboratório 2, porém dessa vez utilizando a lista encadeada simples ao invés de vetores. Assim, será fornecido um novo TAD, com os mesmos operadores, porém adaptado para ser uma lista ligada.

Observações importantes:

1. Neste laboratório será permitido o uso apenas das bibliotecas `stdio.h`, `stdlib.h` e `string.h`.
2. O arquivo `ficha.h` contém os mesmos operadores e funções de manipulação de lista de referências desenvolvidos anteriormente. No entanto, a `struct` é de uma lista encadeada simples. Este arquivo **não deve ser modificado** em hipótese alguma. Alterações devem ser feitas apenas nos arquivos de implementação e cliente (`ficha.c` e `cliente.c`).
3. Para compilar sua solução, utilize o `Makefile`, chamando no terminal o comando `make`.

TAD para manipulação de fichas de referências bibliográficas utilizando lista ligada

Há algum tempo, você organizou suas referências bibliográficas codificando um programa em C que organizava as fichas em um vetor de registros. Todavia, você não ficou satisfeito(a) com o resultado e resolveu refatorar o seu código, substituindo o vetor por uma lista ligada.

Assim como fizemos anteriormente, coletaremos informações sobre diferentes artigos da entrada padrão, armazenando os dados de cada artigo em um registro. A estrutura do registro deverá conter os seguintes campos:

- Número DOI (Digital Object Identifier) do artigo;
- Número de autores;
- Últimos sobrenomes dos autores;
- Ano de publicação;
- Volume;
- Registro que aponta para o próximo elemento.

Para manipular esses registros, implementaremos os seguintes operadores:

- **busca_ficha**: devolve o índice da ficha para um determinado DOI;
- **cria_fichario**: inicia uma nova lista de fichas;
- **imprime_ficha**: imprime na saída padrão uma determinada ficha;
- **insere_ficha**: insere na lista de fichas uma determinada ficha;
- **le_ficha**: lê os dados de uma ficha da entrada padrão;
- **remove_ficha**: remove da lista de fichas a ficha com um determinado DOI;
- **destroi_ficha**: libera uma lista de fichas da memória.

Os operadores acima estão prototipados como funções no arquivo `ficha.h`, acompanhados de documentação com descrições sobre os tipos dos parâmetros que recebem, o que fazem e o que devolvem. Esse arquivo também contém a definição do tipo `ficha`, implementado como `struct`, conforme mencionado acima.

Nas questões a seguir, iremos implementar gradualmente, no arquivo `ficha.c`, cada um desses

operadores.

Questão 1 (5 pontos) - Cliente e inserção de fichas

Nesta questão, implementaremos no arquivo `cliente.c` um loop para ler da entrada padrão uma sequência de comandos e de fichas de referências bibliográficas, um comando ou uma ficha por linha.

Os comandos são os seguintes:

- **N**: Instancia um fichário;
- **I k**: insere `k` fichas no fichário. Essas fichas são fornecidas nas `k` linhas seguintes ao comando, uma ficha por linha;
- **P**: imprime na saída padrão todas as fichas cadastradas, na sequência DOI, sobrenomes dos 3 primeiros autores e ano de publicação (se o artigo tiver mais do que 3 autores, inclua o `"et al."` após o sobrenome do terceiro autor);
- **F**: finaliza a execução do programa.

Por exemplo, se para executar o seu programa for fornecida a seguinte sequência de comandos:

```
N
I 2
10.48550/arXiv.1706.03762 7 Vaswani Shazeer Parmar Uszkoreit Jones Gomez
Kaiser 2017 30
10.1038/s41586-021-03819-2 4 Jumper Evans Pritzel Green 2021 596
P
I 2
10.1145/130385.130401 3 Boser Guyon Vapnik 1992 5
10.1109/5.726791 4 Lecun Bottou Bengio Haffner 1998 86
P
F
```

O seu programa terá que resultar na seguinte saída:

```
10.1038/s41586-021-03819-2 Jumper, Evans, Pritzel, et al. (2021) 596
10.48550/arXiv.1706.03762 Vaswani, Shazeer, Parmar, et al. (2017) 30
10.1109/5.726791 Lecun, Bottou, Bengio, et al. (1998) 86
10.1145/130385.130401 Boser, Guyon, Vapnik (1992) 5
10.1038/s41586-021-03819-2 Jumper, Evans, Pritzel, et al. (2021) 596
10.48550/arXiv.1706.03762 Vaswani, Shazeer, Parmar, et al. (2017) 30
```

Após codificar o cliente e também as operações necessárias na implementação, compile tudo utilizando o `Makefile` e teste executando o seguinte comando:

```
./cliente.bin < teste_Q1.in
```

Onde `teste_Q1.in` é um arquivo contendo uma sequência de comandos como a exemplificada acima.

Questão 2 (5 pontos) - Busca e remoção de fichas

Agora vamos expandir o cliente da questão anterior, codificando mais duas operações na

implementação e acrescentando seus respectivos comandos no cliente:

- **B DOI**: busca pelo registro que contém `DOI` na lista de fichas e o imprime sem o `DOI`. Caso o mesmo não seja encontrado, deverá ser impresso **"DOI inexistente"**;
- **R DOI**: busca pelo registro que contém `DOI` na lista de fichas e o remove (setando como string vazia). Se o registro for removido com sucesso, deverá ser impresso **"DOI DOI removido"**; caso contrário, deverá ser impresso **"DOI DOI inexistente"**.

Por exemplo, se para executar o seu programa for fornecida a seguinte sequência de comandos:

```
N
I 2
10.48550/arXiv.1706.03762 7 Vaswani Shazeer Parmar Uszkoreit Jones Gomez
Kaiser 2017 30
10.1038/s41586-021-03819-2 4 Jumper Evans Pritzel Green 2021 596
P
I 2
10.1145/130385.130401 3 Boser Guyon Vapnik 1992 5
10.1109/5.726791 4 Lecun Bottou Bengio Haffner 1998 86
P
B 10.1038/s41586-021-03819-2
B 10.0000/11111
R 10.1038/s41586-021-03819-2
B 10.1038/s41586-021-03819-2
R 10.0000/22222
I 1
10.48550/arXiv.2304.00985 2 Bühler Stoffer 2006 289
P
F
```

O seu programa terá que resultar na seguinte saída:

```
DOI 10.1038/s41586-021-03819-2 removido
DOI 10.1038/s41586-021-03819-2 inexistente
DOI 10.0000/22222 inexistente
```