Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
SCC0221 - Introdução à Ciência de Computação I

Trabalho 2: Mini Banco de Dados

Professor: Dr. Marcelo Garcia Manzato (mmanzato@icmc.usp.br)
Monitor PAE: Lucas Padilha Modesto de Araújo (padilha.lucas@usp.br)
Monitor PAE: Yuvisa Quispe Palomino (yuvisa.palomino@gmail.com)
Monitor PAP: Pedro Henrique de Sousa Prestes (pedro.prestes@usp.br)
Monitor PAP: Leonardo Dallagnol (dallagnol_leonardo@usp.br)
Data da Entrega: 30/06/2025

Introdução

Para colocar em prática todos os conceitos básicos de computação vistos até agora, sua tarefa para este trabalho será a implementação de um banco de dados básico que possui apenas as funções de inserção, busca e remoção de dados. As seções abaixo irão esclarecer o seu funcionamento com mais detalhes e definir os comandos que ele precisará ter.

Registro

O primeiro passo para a criação de um banco de dados está em entender o que são registros. Em suma, um registro é um conjunto de dados organizados em campos que compõem uma única unidade de informação. Por exemplo, um registro de uma pessoa pode ser composto pelo nome, gênero e idade. No caso do banco de dados que deverá ser implementado, os campos que cada registro deverá ter está descrito logo abaixo.

```
{
    "id": "int",
    "login": "char[15]",
    "password": "char[30]",
    "gender": "char",
    "salary": "double"
}
```

Para resumir, cada registro será um conjunto de 5 dados: id (um inteiro de 4 bytes), login (uma string de tamanho 15), senha (uma string de tamanho 30), genêro (um único caractere) e o salário (um número de ponto flutuante de 8 bytes). Para este trabalho, considere que os tipos aceitos nos campos serão apenas "int", "char", "double" e strings. Considere também que os campos de id e login serão sempre únicos, isto é, não haverá repetição de valores entre os registros.

Inicialização

Sempre que o banco de dados for inicializado, será informado o nome de um arquivo JSON que contém os registros iniciais. Todos os registros estarão organizados como uma sequencia de dados organizados nos mesmos campos que foram mostrados acima. Um exemplo de um arquivo como este pode ser visto abaixo.

```
[
    {
        "id": 1,
        "login": "joaosilva",
        "password": "senha1234567890",
        "gender": "M",
        "salary": 3500.75
    },
    {
        "id": 2,
        "login": "mariarodr",
        "password": "segura12345678",
        "gender": "F",
        "salary": 4200.00
    },
    {
        "id": 3,
        "login": "lucas_fern",
        "password": "lucaspass123456",
        "gender": "M",
        "salary": 5100.25
    }
]
```

Considere que o JSON sempre será um arquivo válido, ou seja, não haverá campos faltantes, linhas mal-formatados, entre outros problemas. Note que strings e caracteres sempre são denotados por valores entre aspas. Assuma que o arquivo JSON sempre irá conter, **no máximo**, 1000 registros. O banco de dados que deve ser implementado terá que possuir a capacidade de fazer a leitura deste arquivo, armazenar todos os registros em memória dinâmica e informar quantos registros foram lidos com uma mensagem do tipo "**x registro(s) lido(s).**" com **x** indicando a quantidade lida.

Comandos

Inserção

As inserções no banco de dados serão identificadas com o número 1 no início da linha. Para facilitar o processo de inserção, o banco de dados deverá ter um limite máximo de 1000 registros, sendo que, se o usuário tentar adicionar mais dados após este limite, o sistema deverá informar "**Sem espaço para inserção.**".

```
1 33 "login_xyz" "pswrd_xyz" "M" 1020.20
```

Acima está exemplificando como será o comando de inserção para o banco de dados. A ordem dos campos será sempre a mesma: **id, login, password, gender, salary**. Todos os campos estarão separados por espaços e **apenas** strings ou caracteres são inseridos entre aspas. Strings serão sempre case-sensitive. Assuma que **não** haverá inserções problemáticas, como por exemplo, inserções com campos faltantes ou com tipos de dados errados. Sempre que uma inserção for realizada, o sistema deverá informar "**Registro inserido.**".

Observações:

- É necessário que este banco de dados seja implementado com o método *first-fit*. Isso significa que as inserções devem ser alocadas sempre na menor posição livre. Por exemplo, se as posições 0 e 500 estiverem livres para um novo registro, a posição 0 deverá ser prioritária.

Busca

As buscas no banco de dados serão identificadas com o número 2 no início da linha. Não será necessário nenhuma implementação de algoritmos eficientes de busca para este trabalho, isto é, todas as buscas podem ser feitas de forma linear. Quando uma busca não obtiver nenhum retorno, seja pelo valor ou campo inexistente, o sistema deverá informar "**Nada encontrado.**".

```
2 "id" 2
```

Acima está exemplificando como será o comando de busca para o banco de dados. Cada busca será acompanhada do nome do campo, este que estará **sempre** entre aspas, e do valor que será buscado separados por um espaço. Caso uma busca seja bem-sucedida, o registro deverá ser impresso seguindo a mesma estética do JSON com a seguinte ordem dos campos:

```
{
    "id": 2,
    "login": "mariarodr",
    "password": "segura12345678",
    "gender": "F",
    "salary": 4200.00
}
```

Observações:

- A quantidade de espaços utilizada antes da impressão dos campos é igual a 4.
- Use apenas 2 casas decimais durante a impressão de números com ponto flutuante.
- A impressão dos registros para as buscas que retornarem mais de um resultado deve seguir a ordem das posições.

Remoção

As remoções no banco de dados serão identificadas com o número 3 no início da linha. Em suma, uma remoção consiste em uma busca com a diferença de que o registro deverá ser descartado ao contrário de ser mostrado. Nos casos em que a busca pelo registro a ser removido falhar, o sistema deverá informar "**Remoção inválida.**". A remoção também deve falhar se o campo não existir.

```
3 "id" 2
```

Acima está exemplificando como será o comando de remoção para o banco de dados. Para uma remoção bem sucedida, o sistema deverá informar "**Registro removido.**". Os registros que forem removidos poderão ser substituídos por novas entradas.

Tarefa

Conforme dito anteriormente, você deverá criar um programa contendo a implementação do banco de dados que foi descrito neste documento. A entrada será composta pelo nome do arquivo JSON para inicialização seguido de várias linhas de comando.

Segue abaixo um exemplo de entrada e saída:

Entrada	**Saída**

data_1.json
2 ”gender” ”M”
1 33 ”sol_x” ”axc331” ”M” 1020.20
2 ”salary” 1020.20
3 ”id” 33
3 ”id” 5555
3 ”gender” ”M”
2 ”gender” ”M”
	`3 registro(s) lido(s).`
	{
	`"id": 1,`
	`"login": "joaosilva",`
	`"password": "senha1234567890",`
	`"gender": "M",`
	`"salary": 3500.75`
	}
	{
	`"id": 3,`
	`"login": "lucas_fern",`
	`"password": "lucaspass123456",`
	`"gender": "M",`
	`"salary": 5100.25`
	}
	`Registro inserido.`
	{
	`"id": 33,`
	`"login": "sol_x",`
	`"password": "axc331",`
	`"gender": "M",`
	`"salary": 1020.20`
	}
	`1 registro(s) removido(s).`
	`Remoção inválida.`
	`2 registro(s) removido(s).`
	`Nada encontrado.`

Observação: Neste exemplo, o conteúdo do ”data_1.json” é idêntico ao JSON mostrado na seção de inicialização no início deste documento.

Requisitos

- O trabalho será individual.
- Utilize apenas a linguagem C com as bibliotecas tradicionais.
- Entrega até 30/06/2025 (23:55) no RunCodes (https://runcodes.icmc.usp.br/).
- Faça um código legível, documentado e organizado. Os códigos serão analisados manualmente, portanto, isso será critério de avaliação.
- Todos os registros deverão ser armazenados em **memória dinâmica (heap)**. O restante das variáveis pode ficar em memória estática.
- O programa não deverá possuir vazamentos de memória. A verificação pode ser feita através do uso do "valgrind" (não funciona em WSL) ou com a compilação do código com "-fsanitize=address" (ex: gcc -fsanitize=address -g main.c -o main).

Observações e Dicas

- Plágios resultarão em nota zero para todos os envolvidos.
- O sistema não aceitará submissões após o prazo, mesmo que seja de poucos minutos de atraso. Por isso, é recomendado submeter o trabalho com antecedência.
- Tente deixar o código modularizado. Isso pode ajudar no desenvolvimento do trabalho.
- **Não deixe para última hora. Bugs são imprevisíveis.**