Instituto Federal do Espírito Santo — IFES (Campus Serra)
Coordenadoria de Informática — Cin

Trabalho de Programação

Simulação de Realização de Exames de Raio-X (parte 1)

Valor: 10 pontos
Deadline: 07 de julho de 2024

Prof. Thiago M. Paixão
thiago.paixao@ifes.edu.br

1 Objetivo

O trabalho prático de programação consiste em simular o processo de realização de exames de raio-X de tórax em um hospital, com ênfase na organização da fila para laudo médico. A cada momento, pacientes chegam ao hospital e exames são realizados mediante disponibilidade de aparelhos. A IA[1] sugere diagnósticos preliminares e os exames são encaminhados para laudo de acordo com a disponibilidade da equipe de radiologistas.

Neste etapa do trabalho, o objetivo é implementar dois tipos abstratos de dados (TADs): `Patient` e `Exam` Esses TADs serão utilizados para modelar os pacientes que chegam ao hospital e os exames de raio-X.

2 Tipo abstrato de dados

2.1 Patient

O TAD `Patient` representa um paciente no sistema. Este TAD deve conter as seguintes informações:

- Identificação única do paciente (`int id`);
- Nome do paciente (`char* name`);
- Data de nascimento (`struct tm* birthdate`)[2].

Além da estrutura com os campos mencionados, o TAD deve implementar as seguintes funções:

- `Patient* create_patient(int id, const char *name, struct tm *birthdate, struct tm *arrival)`: Cria um novo paciente, alocando memória para a estrutura, preenchendo os campos internos com os dados passados como parâmetros e retornando um ponteiro para a estrutura criada.
- `void destroy_patient(Patient *patient)`: Libera a memória alocada para a estrutura do paciente.
- `int get_patient_id(Patient* patient)`: Retorna a identificação única do paciente.
- `const char get_patient_name(Patient* patient)`: Retorna o nome do paciente.
- `struct tm* get_patient_birthdate(Patient patient)`: Retorna a data de nascimento do paciente.

[1] Não implementaremos a IA propriamente dita, apenas simularemos seu resultado.
[2] Para saber mais sobre o tipo `struct tm`, acesse `https://en.cppreference.com/w/c/chrono/tm`.

2.2 Exam

O TAD `Exam` representa, no sistema, um exame realizado por um paciente. Este TAD deve conter as seguintes informações:

- Identificação única do exame (`int id`);
- Identificação única do paciente (`int patient_id`);
- Identificação única do aparelho de raio-X utilizado para realização do exame (`int rx_id`);
- Horário de realização do exame (`struct tm* time`).

Além da estrutura com os campos mencionados, o TAD deve implementar as seguintes funções:

- `Exam* create_exam(int id, int patient_id, int rx_id, struct tm *time)`: Cria um novo exame, alocando memória para a estrutura, preenchendo os campos com os dados passados como parâmetros e retornando um ponteiro para a estrutura criada.
- `void destroy_exam(Exam *exam)`: Libera a memória alocada para a estrutura de exame.
- `int get_exam_id(Exam *exam)`: Retorna a identificação única do exame.
- `int get_exam_patient_id(Exam *exam)`: Retorna a identificação única do paciente associado ao exame.
- `int get_exam_rx_id(Exam exam)`: Retorna a identificação única do aparelho de raio-X utilizado no exame.
- `struct tm* get_exam_time(Exam *exam)`: Retorna o horário de realização do exame.

3 Testando a implementação

Para testar sua implementação, utilize a implementação da função `main` disponibilizada no Código 1. Os módulos adicionais dever ser implementados separadamente e **importados** no módulo principal.

Código 1: Teste.

```c
// Função principal de teste
int main()
{
    // Definindo uma data de nascimento fictícia para o paciente
    struct tm birthdate = {0}; // Inicializar a estrutura com zeros
    birthdate.tm_year = 90; // Ano 1990
    birthdate.tm_mon = 5; // Junho (0-indexed)
    birthdate.tm_mday = 15; // Dia 15

    // Criando um paciente
    Patient *patient = create_patient(1, "João Silva", &birthdate);

    // Imprimindo informações do paciente criado
    printf("Paciente criado:\n");
    printf("ID: %d\n", get_patient_id(patient));
    printf("Nome: %s\n", get_patient_name(patient));
    printf("Data de Nascimento: %s\n", asctime(get_patient_birthdate(patient)));

    // Definindo uma data e hora fictícias para o exame
    time_t current_time;
```

```c
    struct tm *exam_time;

    // Tempo do exame éo tempo atual
    current_time = time(NULL); // Obter o tempo atual em segundos desde 01/01/1970
    exam_time = localtime(&current_time); // Converter o tempo para a hora local

    // Criando um exame associado ao paciente criado
    Exam *exam = create_exam(101, get_patient_id(patient), 1, exam_time);

    // Imprimindo informações do exame criado
    printf("\nExame criado:\n");
    printf("ID: %d\n", get_exam_id(exam));
    printf("ID do Paciente: %d\n", get_exam_pacient_id(exam));
    printf("ID do Aparelho de Raio-X: %d\n", get_exam_rx_id(exam));
    printf("Data e Hora do Exame: %s\n", asctime(get_exam_time(exam)));

    // Liberando a memória alocada
    destroy_exam(exam);
    destroy_patient(patient);

    return 0;
}
```

4 Critérios de avaliação

A avaliação deste trabalho levará em consideração os seguintes critérios:

1. Implementação e uso adequado dos TADs: Até **8 pontos** serão atribuídos à implementação adequada dos TADs, o que também inclui a estruturação adequada do código (modularização) e a utilização de boas práticas de programação para evitar o acesso direto aos campos internos dos TADs em módulos clientes.
2. Documentação do código: Até **0,5 pontos** serão atribuídos à qualidade da documentação incorporada ao código. Certifique-se de incluir comentários significativos que expliquem a lógica por trás das implementações.
3. README descritivo: Até **1,5 pontos** será concedido pela criação de um arquivo README.md descritivo e informativo no seu repositório. O README deve fornecer informações claras sobre como executar e utilizar o seu projeto. Além disso, deve exibir a estrutura do projeto e apresentar os principais TADs.
4. Apresentação (a ser agendada): Um fator real α será aplicado após a apresentação do projeto. α será um número entre 0 e 1, refletindo a qualidade da apresentação e a capacidade de explicar e defender o projeto.
5. Correção: Será atribuído $\beta = 1$ se não houver falhas críticas no projeto; caso contrário, $\beta = 0$. Isso avaliará se o projeto funciona conforme o esperado e se atende aos requisitos especificados.
6. Dias de atraso: Será descontado 20% do valor total do trabalho a cada dia de atraso.

A nota será calculada utilizando a equação $nota = (1 - d/5) \times \alpha \times \beta \times P$, onde P é a soma dos pontos dos critérios 1 a 4. É importante notar que a nota será zerada após 5 dias de atraso.

Importante: O programa será testado num ambiente Linux Ubuntu 22.04 com GCC 11. Recomendo FORTEMENTE desenvolver e testar nesse ambiente.

5 O que entregar?

1. Um link para um repositório .git com código fonte do projeto: `Makefile` e arquivos `.c` e `.h`.
2. A documentação/relatório será feita no arquivo README.md do repositório.

Bom trabalho!