UFRJ
UNIVERSIDADE FEDERAL
DO RIO DE JANEIRO

**UNIVERSIDADE FEDERAL DO RIO DE JANEIRO**

# TRABALHO FINAL - LINGUAGENS DE PROGRAMAÇÃO

**LÍGIA CALINA BUENO BONIFÁCIO DRE: 122046065**

Rio de Janeiro-RJ
2024

**SUMÁRIO**

# 1 INTRODUÇÃO

O objetivo principal deste trabalho é desenvolver um sistema organizacional pessoal baseado em listas de tarefas (*to-do lists*), um método de organização amplamente utilizado. Através de um sistema de gerenciamento de tarefas, é possível aumentar a produtividade, organizar atividades diárias e acompanhar o progresso de tarefas pendentes e concluídas.

Este projeto implementa um sistema de gerenciamento de tarefas utilizando a linguagem C++ para a estrutura principal e funcionalidades básicas, e *Python* para análises avançadas e manipulação de dados. A integração entre C++ e *Python* é realizada com a biblioteca *Pybind11*, que permite a chamada de funções *Python* diretamente a partir de um código C++.

As principais funcionalidades do sistema incluem:

- **Adição de Tarefas:** Criação de novas tarefas com informações como descrição, prioridade e recorrência.
- **Remoção de Tarefas:** Exclusão de tarefas que não são mais necessárias.
- **Edição de Tarefas:** Modificação de informações de tarefas existentes.
- **Conclusão de Tarefas:** Marcar tarefas como concluídas, movendo-as para uma lista de tarefas concluídas.
- **Análise das Tarefas:** Utilização de funções em *Python* para analisar e manipular tarefas.

O foco deste trabalho é estudar a interface entre duas linguagens de programação, especificamente a interação entre C++ e *Python*. Através da biblioteca *Pybind11*, o projeto utiliza scripts *Python* para realizar a análise de conteúdo das tarefas, como contagem de palavras e identificação de palavras-chave nas descrições das tarefas.

Este trabalho explora a integração dessas funções de análise em *Python* com o código principal em C++.

# 2 IMPLEMENTAÇÃO

## 2.1 ESTRUTURA DO PROJETO

A estrutura do projeto é mais complexa do que trabalhos anteriores, envolvendo múltiplas pastas e arquivos. Foram implementadas estruturas de classes para facilitar o entendimento do código. O sistema de tarefas exibe um menu para o usuário, tornando a manipulação do programa mais intuitiva. Por fim, a integração com *Python* foi realizada utilizando a biblioteca `pybind11`, que permite a execução de um interpretador *Python* durante o processo do código, permitindo a chamada de funções de *scripts Python*. Esses detalhes estão descritos nas próximas seções.

### 2.1.1 Arquivos do Projeto

- `tarefa.hpp` e `tarefa.cpp`: Implementação da classe `Tarefa`, que define as propriedades e comportamentos de uma tarefa.
- `sistemaTarefas.hpp` e `sistemaTarefas.cpp`: Implementação da classe principal `SistemaTarefas`, responsável pelo gerenciamento das tarefas.
- `main.cpp`: Arquivo principal do projeto, contendo a função `main()` que inicializa o sistema.
- `analisarPython.py` e `analisarTarefa.py`: Scripts Python para análise de tarefas, realizando funções como contagem de palavras e identificação de palavras-chave.
- `pybind11`: Biblioteca utilizada para integrar o interpretador de Python com o código C++, permitindo a chamada de funções Python a partir do C++.

O fluxo de informações e a arquitetura do programa estão descritos pelo diagrama abaixo.

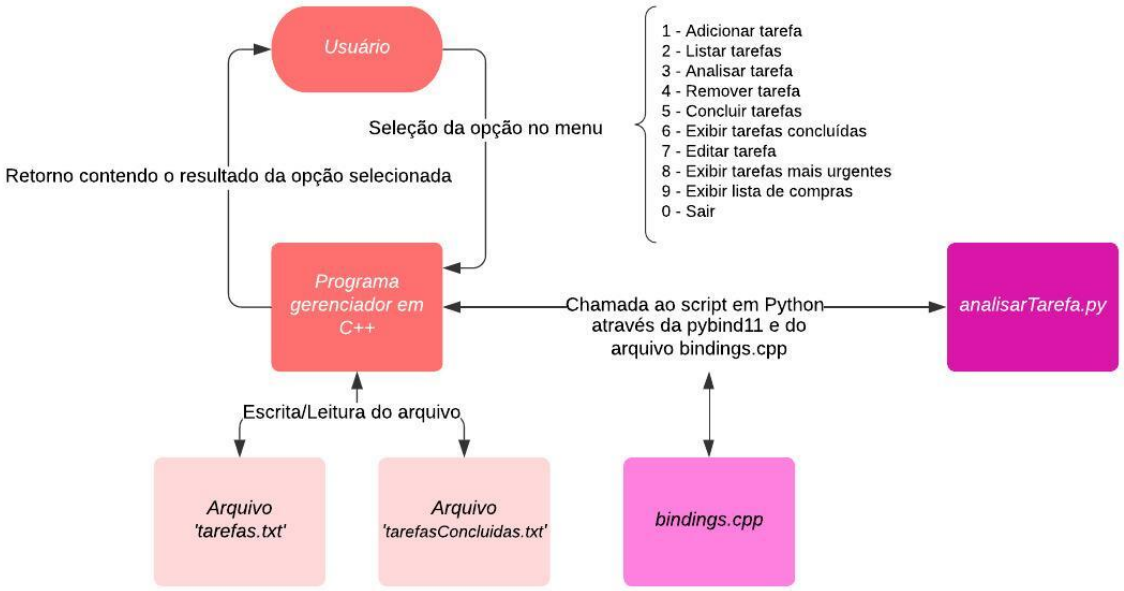


Figura 1 – Desenho da arquitetura do programa.

### 2.1.2 Estrutura das Classes

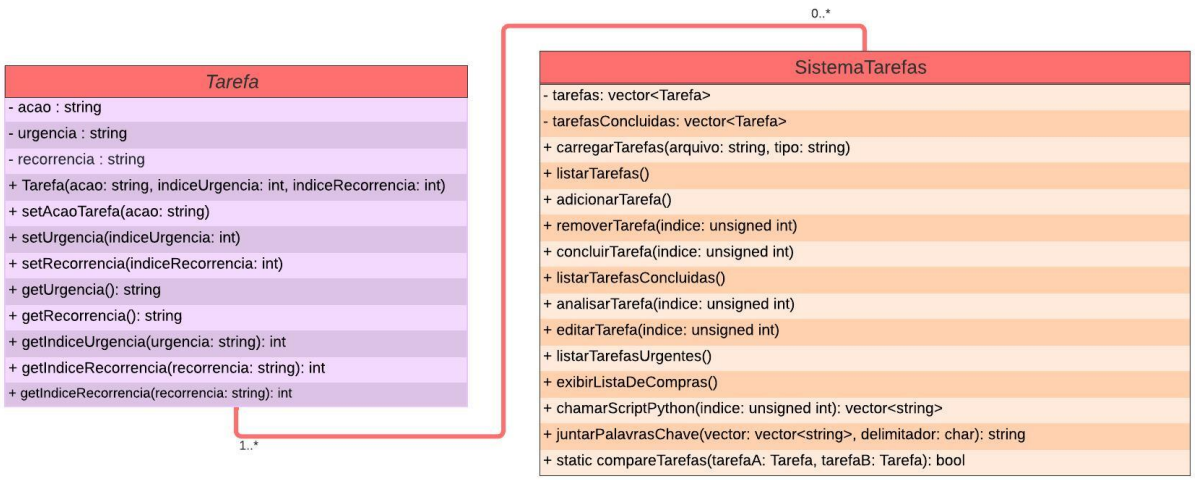


Figura 2 – Diagrama UML de classes do sistema de gerenciamento de tarefas.

A estrutura do programa é baseada em duas classes principais: `Tarefa` e `SistemaTarefas`, conforme descrito no diagrama acima. Estas classes são projetadas para trabalhar juntas e fornecer um sistema robusto de gerenciamento de tarefas.

**Classe Tarefa**:

- `acao`: Descrição da tarefa.
- `urgencia`: Nível de urgência da tarefa (baixa, média ou alta).
- `recorrencia`: Tipo de recorrência da tarefa (diária, semanal ou mensal).

- **Métodos**:
  - `Tarefa(acao: string, indiceUrgencia: int, indiceRecorrencia: int)`: Construtor que inicializa uma tarefa com os atributos fornecidos.
  - `setAcaoTarefa(acao: string)`: Define a ação da tarefa.
  - `setUrgencia(indiceUrgencia: int)`: Define o nível de urgência da tarefa.
  - `setRecorrencia(indiceRecorrencia: int)`: Define o tipo de recorrência da tarefa.
  - `getAcao() : string`: Retorna a ação da tarefa.
  - `getUrgencia() : string`: Retorna a urgência da tarefa.
  - `getRecorrencia() : string`: Retorna a recorrência da tarefa.
  - `getIndiceUrgencia(urgencia: string) : int`: Retorna o índice de urgência com base em uma string de urgência.
  - `getIndiceRecorrencia(recorrencia: string) : int`: Retorna o índice de recorrência com base em uma string de recorrência.

**Classe SistemaTarefas**:

- `tarefas`: Vetor que armazena as tarefas a serem realizadas.
- `tarefasConcluidas`: Vetor que armazena as tarefas concluídas.
- **Métodos**:
  - `carregarTarefas(arquivo: string, tipo: string)`: Carrega tarefas de um arquivo especificado.
  - `listarTarefas()`: Lista todas as tarefas pendentes.
  - `adicionarTarefa()`: Adiciona uma nova tarefa.
  - `removerTarefa(indice: unsigned int)`: Remove uma tarefa com base em seu índice.
  - `concluirTarefa(indice: unsigned int)`: Marca uma tarefa como concluída.
  - `listarTarefasConcluidas()`: Lista todas as tarefas concluídas.
  - `analisarTarefa(indice: unsigned int)`: Analisa uma tarefa utilizando scripts Python.
  - `editarTarefa(indice: unsigned int)`: Edita uma tarefa existente.
  - `listarTarefasUrgentes()`: Lista todas as tarefas urgentes.
  - `exibirListaDeCompras()`: Exibe uma lista de compras gerada a partir das tarefas.
  - `chamarScriptPython(indice: unsigned int) : vector<string>`: Chama um script Python para realizar análises nas tarefas.

  - `juntarPalavrasChave(vector: vector<string>, delimitador: char) : string`: Junta palavras-chave de um vetor em uma string com um delimitador especificado.
  - `static compareTarefas(tarefaA: Tarefa, tarefaB: Tarefa) : bool`: Compara duas tarefas para ordenação.

O sistema utiliza dois vetores principais para o armazenamento de dados:

- `tarefas`: Armazena as tarefas que ainda precisam ser realizadas.
- `tarefasConcluidas`: Armazena as tarefas que já foram concluídas.

Além disso, o sistema faz uso de dois arquivos auxiliares para persistência de dados:

- `tarefas.txt`: Armazena as tarefas a serem realizadas.
- `tarefasConcluidas.txt`: Armazena as tarefas concluídas.

Estes arquivos são atualizados de forma dinâmica durante a execução do código, garantindo que as informações sobre as tarefas estejam sempre atualizadas nos arquivos em tempo de execução.

### 2.1.3 Integração com Python

Utilizando `pybind11`, o sistema integra funções Python para análise detalhada das tarefas gerenciadas e geração de listas de compras.

A integração com Python permite que o programa em C++ chame dinamicamente funções definidas em scripts Python, utilizando a biblioteca `pybind11` para facilitar essa comunicação. Aqui está o passo-a-passo de como o programa realiza essa integração:

1. **Inicialização do Interpretador Python:** O método `chamarScriptPython` da classe `SistemaTarefas` inicia o interpretador Python utilizando `py::initialize_interpreter()`.

```
vector<string> SistemaTarefas::chamarScriptPython(unsigned int indice) {
    vector<string> resultado;
    py::initialize_interpreter();
```

2. **Verificação do Índice da Tarefa:** Verifica se o índice da tarefa é válido dentro do vetor de tarefas. Caso contrário, exibe uma mensagem de erro e retorna uma lista vazia.

```
    if (indice < 1 || indice > tarefas.size()) {
    cout << "Indice de tarefa invalido." << endl;
```

```
return resultado;}
```

3. **Obtenção do Conteúdo da Tarefa:** Obtém o conteúdo da tarefa específica através do método `getAcao()` da classe `Tarefa`.

```
string conteudoTarefa = tarefas[indice - 1].getAcao();
```

4. **Chamada das Funções Python:**
   a) Importa o módulo Python `analisarTarefa` - arquivo analisarTarefa.py utilizando `py::module_::import("analisarTarefa")`.
   b) Atribui as funções Python `contar_palavras` e `identificar_palavras_chave` a variáveis locais no C++ (`contar_palavras_func` e `identificar_palavras_chave_func`).
   c) Chama essas funções Python, passando o conteúdo da tarefa como argumento. Os resultados são convertidos de volta para C++ usando `cast<int>()` para a contagem de palavras e `cast<py::list>()` para a lista de palavras-chave.

```
try {
 py::module_ script = py::module_::import("analisarTarefa");
 auto contar_palavras_func = script.attr("contar_palavras");
 auto identificar_palavras_chave_func
 = script.attr("identificar_palavras_chave");

 int contagemPalavras
 = contar_palavras_func(conteudoTarefa).cast<int>();
 py::list py_palavras_chave
 = identificar_palavras_chave_func(conteudoTarefa).cast<py::list>()

 vector<string> palavrasChave;
 for(unsigned int i=0; i<py_palavras_chave.size();++i){
    palavrasChave.push_back(py_palavras_chave[i].cast<string>());
 }
```

5. **Processamento dos Resultados:** Os resultados retornados pelas funções Python são processados e armazenados em um vetor de strings `resultado`, contendo o conteúdo da tarefa, a contagem de palavras e as palavras-chave identificadas.

```
resultado.push_back(conteudoTarefa);
resultado.push_back(to_string(contagemPalavras));
resultado.push_back(juntarPalavrasChave(palavrasChave, ' '));
```

6. **Finalização do Interpretador Python:** Finaliza o interpretador Python utilizando `py::finalize_interpreter()`.
7. **Retorno dos Resultados:** Retorna o vetor `resultado` para o programa principal em C++.

Essa integração possibilita que o sistema de gerenciamento de tarefas utilize as capacidades avançadas de análise textual do Python, enquanto mantém a estrutura principal do programa em C++. A configuração dos bindings é realizada no arquivo `bindings.cpp`, onde a função `chamarScriptPython` é exposta para ser acessada pelo Python como `chamar_script_python`.

```
#include <pybind11/pybind11.h>
#include "sistemaTarefas.hpp"

namespace py = pybind11;

PYBIND11_MODULE(analisarTarefa_modulo, m) {
    m.def("chamar_script_python", SistemaTarefas::chamarScriptPython);
}
```

## 3 CONCLUSÃO

O trabalho apresentou a implementação de um sistema avançado de gerenciamento de tarefas em C++, com foco na estruturação das classes, persistência de dados e integração dinâmica com Python para análise textual. A estrutura do projeto foi projetada para proporcionar uma organização clara e eficiente, utilizando classes como `Tarefa` e `SistemaTarefas` para gerenciar tarefas com atributos como descrição, urgência e recorrência.

A integração com Python foi realizada utilizando a biblioteca `pybind11`, permitindo que funções Python fossem chamadas diretamente do código C++ para análise detalhada das tarefas, como contagem de palavras e identificação de palavras-chave. Essa abordagem oferece ao sistema capacidades avançadas de processamento textual, mantendo a estrutura principal em C++.

A aplicação prática desses conceitos resultou em um sistema capaz de carregar, manipular, analisar e persistir tarefas de forma eficiente. A estrutura modular do projeto facilita a manutenção e extensão futura, enquanto a integração com Python amplia suas funcionalidades analíticas.

Por fim, o projeto demonstrou não apenas a aplicação de princípios de orientação a objetos e persistência de dados em C++, mas também a integração eficaz entre linguagens de programação para atender a requisitos específicos de análise e processamento de dados textuais em tempo real.