

**UNIVERSIDADE FEDERAL DO ESTADO DE MINAS GERAIS  
ENGENHARIA DE SISTEMAS**

**Programação e Desenvolvimento de Software II  
Trabalho Prático Final**

**Sistema de Gerenciamento de Resíduos**

**Lucas Silva Chaves  
Lucas Gonçalves Bispo  
Pedro Herculano Flores  
Rayane Hellen Dias Soares**

## 1. INTRODUÇÃO

Objetivando a criação de um sistema que auxilie moradores, empresas, condomínios e escolas, na implantação de um projeto de coleta seletiva foi criado um sistema de gerenciamento de resíduos.

O sistema é composto de quatro classes básicas, uma para o usuário, para o resíduo, para o ponto de coleta e por fim uma para o Agendamento da coleta.

Além disso, propôs-se a criação de um menu interativo para o cadastro, alteração e remoção de cada uma dessas classes e a criação de um banco de dados.

## 2. IMPLEMENTAÇÃO

### Usuário:

O Usuário foi feito como uma **classe abstrata**, suas funções foram declaradas como virtuais e suas classes derivadas fazem o **override** delas. Suas filhas são as classes: Pessoa Física e Pessoa Jurídica, decidiu-se utilizar de **polimorfismo** nessas duas classes porque elas compartilham muitos elementos em comum.

Para ambas as classes além da criação de **construtores default e com argumentos** optou-se por fazer a **sobrecarga dos operadores** de entrada e saída (std::cin e std::cout) para que a interação com o menu criado seja mais fácil.

As principais funções das classes são os gets e sets criados para cada atributo que as compõe, uma vez que eles são **privados**.

Usuário	Pessoa Física: Público Usuário
<div>-ID:int - nome:string -nome_usuario:String -endereço:string -senha:string</div> <div>+ set_nome(string) + set_nome_usuario(string) + set_endereco(string) + set_id(int)</div> <div>+ get_nome() const + get_nome_usuario() const + get_endereco() const + get_id() const</div> <div>+ valida_dados() + login(string,string)</div>	<div>- cpf: string</div> <div>+ set_cpf(string) + get_cpf() const + ostream&amp; operator &lt;&lt;(ostream&amp;, const PessoaFisica&amp;) + isteam&amp; operator &gt;&gt;(istream&amp;,PessoaFisica&amp;)</div>
	Pessoa Jurídica: Público Usuário
	<div>- cnpj:string</div> <div>+ set_cnpj(string) + get_cnpj() const + ostream&amp; operator &lt;&lt;(ostream&amp;, const PessoaJuridica&amp;) + isteam&amp; operator &gt;&gt;(istream&amp;,PessoaJuridica&amp;)</div>

### Resíduos:

Assim como a classe usuário, para os resíduos foram implementados gets e sets para todos os seus atributos além da **sobrecarga do operador de entrada e saída** e os **construtores default e por parâmetros**. Além disso foi criada uma enumeração para os tipos de resíduo que podem ser cadastrados, e com ele mais duas funções, uma para converter um inteiro, digitado pelo usuário para o tipo da enumeração e outra função para converter esse tipo criado por nós para uma string que será printada para o usuário.

Resíduos
<ul style="list-style-type: none"><li>- ID: int</li><li>- nome_residuo: string</li><li>- forma_armazenamento: string</li><li>- tipo_residuo: TipoResiduo</li><li>- Quantidade: double</li><li>- Unidade: string</li></ul>
<ul style="list-style-type: none"><li>+ set_ID(int)</li><li>+ set_nome_residuo(string)</li><li>+ set_forma_armazenamento(string)</li><li>+ set_Unidade(string)</li><li>+ set_quantidade(double)</li><li>+ set_tipo_residuo(TipoResiduo)</li><li>+ get_ID()</li><li>+ get_nome_residuo()</li><li>+ get_forma_armazenamento()</li><li>+ get_Unidade()</li><li>+ get_quantidade()</li><li>+ get_tipo_residuo()</li><li>+ ostream&amp; operator &lt;&lt;(ostream&amp;, const Residuo&amp;)</li><li>+ istream&amp; operator &gt;&gt;(istream&amp;,Residuo&amp;)</li></ul>

## Ponto Coleta

O ponto de coleta tem também gets e sets para todos seus atributos, que são privados, além disso utiliza da relação de **composição** para fazer um link entre o local cadastrado e quem é o usuário dono dele, caso tenha. Além disso, há a **sobrecarga dos operadores** de entrada e saída e a criação dos **construtores default e por parâmetros**.

Ponto Coleta
<ul style="list-style-type: none"><li>- nome(string)</li><li>- endereco(string)</li><li>- user(*Usuario)</li><li>- ID(int)</li></ul>
<ul style="list-style-type: none"><li>+ get_nome: string</li><li>+ get_endereco: string</li><li>+ get_user: *Usuario</li><li>+ get_ID: int</li><li>+ get_user_ID: int</li><li>+ set_nome: bool</li><li>+ set_endereco: bool</li><li>+ set_user: bool</li><li>+ set_ID: bool</li><li>+ ostream&amp; operator &lt;&lt;(ostream&amp;, const PontoColeta&amp;)</li><li>+ istream&amp; operator &gt;&gt;(istream&amp;,PontoColeta&amp;)</li></ul>

## Agendamento

Assim como as outras classes já citadas, o Agendamento tem gets e sets para todos os seus atributos, **construtor default e por parâmetros**, **sobrecarga dos operadores** de entrada e saída além de utilizar da relação de **composição** para fazer a relação entre os usuários doadores e receptores, o local da coleta e os resíduos envolvidos.

Agendamento
<ul style="list-style-type: none"><li>- data_agendado(string)</li><li>- horario_agendado(string)</li><li>- local(PontoColeta*)</li><li>- doador(Usuario*)</li><li>- receptor(Usuario*)</li><li>- ID(int)</li><li>- status(Status)</li><li>- itens_agendamento(list &lt;AgendamentoItens*&gt;*)</li></ul>
<ul style="list-style-type: none"><li>+ get_data_agendada: string</li><li>+ get_horario_agendado: string</li><li>+ get_doador: *Usuario</li><li>+ get_receptor: *Usuario</li><li>+ get_local: *PontoColeta</li><li>+ get_status: Status</li><li>+ get_itens: list &lt;*AgendamentoItens*&gt;</li><li>+ get_ID: int</li><li>+ set_data_agendada: bool</li><li>+ set_horario_agendado: bool</li><li>+ set_ID: bool</li><li>+ set_local: *PontoColeta</li><li>+ set_status: Status</li><li>+ get_itens: list &lt;*AgendamentoItens*&gt;</li><li>+ add_residuo(Residuo*,list &lt;AgendamentoItens*&gt;): bool</li><li>+ remove_residuo(Residuo*):</li><li>+ ostream&amp; operator &lt;&lt;(ostream&amp;, const Agendamento&amp;)</li><li>+ isteam&amp; operator &gt;&gt;(istream&amp;,Agendamento&amp;)</li></ul>

### AgendamentoItens:

Classe que representa os itens em um agendamento de uma coleta a ser feita, armazena um resíduo e sua quantidade.

Contém sets e gets pro seus atributos, o resíduo requerido, sua quantidade;

AgendamentoItens
- resíduo(*Resíduo) - quantidade(double) - id_agendamento(int) - id(int)
+ get_quantidade(): double + get_resíduo(): Resíduo* + set_resíduo(Resíduo*): bool + set_quantidade(double): bool

Além das 4 classes principais, utilizou-se do conceito de **modularização** para a criação de um menu interativo, com ele foram criadas mais 6 novas classes, uma para controlar o menu, um **template** para os controladores das 4 classes principais e o controlador dessas 4 classes.

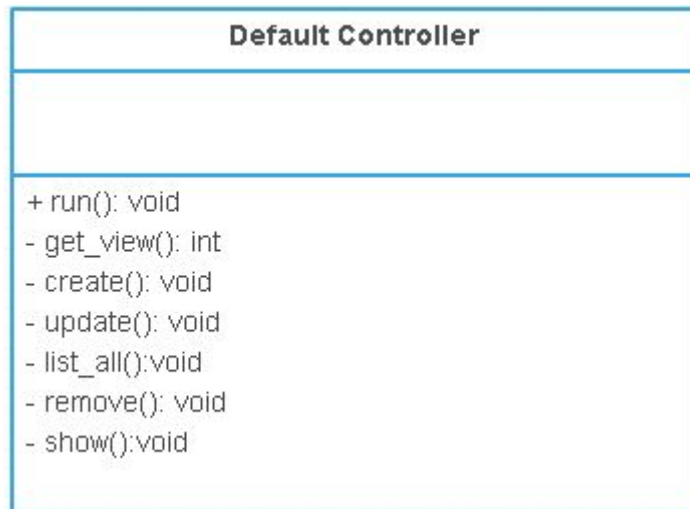
### Main Controller:

Classe que controla o Menu Interativo, ela chama determinado Controlador de classe baseado no que o usuário quer fazer.

Main Controller
- helper : DbHelper* - residuos : ResiduosController* - usuario: UsuarioController* - local: PontoColetaController* - agendamento : AgendamentoController*
- showView: int + run: void

### DefaultController:

DefaultController é um **template** criada para ser adaptada a todo tipo de outro controlador criado, evitando assim a cópia de código, dessa forma todas as suas funções são declaradas **puramente virtuais**.



**run():** Menu interativo que permite o usuário escolher qual ação quer fazer (cadastrar, atualizar, remover, listar, detalhar ou sair do menu)

**get\_view():** Armazena qual ação disponível no menu o usuário quer fazer.

**create():** cadastro.

**update():** alteração.

**show():** detalhar determinado cadastro.

**remove():** remoção.

**list\_all():** lista todos os cadastro

### Usuário Controller:

Classe **derivada** de Default Controller, nela tem todas as funções necessárias para o acesso e controle da classe Usuário, nela é implementando as função de criar um novo usuário, modificar, mostrar, remover, listar e atribuir um ID automaticamente para cada usuário cadastrado.

Além disso, seu atributo (dao), é o que faz a comunicação com o banco de dados do Usuário.

Usuário Controller: public Default Controller
- dao(UsuarioDAO*)
+ run(): void - get_view(): int - create(): void - update(): void - list_all():void - remove(): void - show():void

### Resíduos Controller:

Mesmas funcionalidade de Usuário controller, implementa as funções **herdadas** da classe Default e seu único atributo faz a comunicação com o banco de dados.

Resíduos Controller: public Default Controller
- dao(ResíduosDAO*)
+ run(): void - get_view(): int - create(): void - update(): void - list_all():void - remove(): void - show():void



### PontoColeta Controller:

Mesmas funcionalidade de Usuário controller, implementa as funções **herdadas** da classe Default e seus atributos fazem a comunicação com seu banco de dados e o banco de dados do Usuário que pode estar relacionado ao ponto de coleta.

<b>PontoColeta Controller: public Default Controller</b>
- dao(PontoColetaDAO*) - usuarioDAO (UsuarioDAO*)
+ run(): void - get_view(): int - create(): void - update(): void - list_all():void - remove(): void - show():void

### Agendamento Controller:

Assim como suas funções irmãs, ela implementa as funções herdadas da classe pai e seus atributos extras fazem a comunicação entre o seu banco de dados e o do Usuário, do Ponto de Coleta e dos Resíduos.

<b>Agendamento Controller: public Default Controller</b>
- dao(AgendadomentoDAO*) - localDAO(PontoColetaDAO*) - usuarioDAO (UsuarioDAO*) - residuosDAO(ResiduosDAO*)
+ run(): void - get_view(): int - create(): void - update(): void - list_all():void - remove(): void - show():void

Um outro **módulo** de classe criado foi o que faz as operações com o banco de dados.

#### DbHelper:

Classe para auxiliar no gerenciamento do Banco de Dados, possui métodos para executar as seguintes operações:

- Inicializar a conexão com o Banco de Dados.
- Fechar a conexão com o Banco de Dados.
- Limpar o banco de dados.
- Criar as tabelas necessárias.
- Executar comandos SQL's
- Realizar a leitura de dados, com o auxílio da classe helper Row.
- 

DbHelper
<ul style="list-style-type: none"><li>- dbFiler(sqlite3*)</li><li>- dbName(string)</li><li>- migration(int)</li></ul>
<ul style="list-style-type: none"><li>+ startConnection(): bool</li><li>+ closeConnection(): bool</li><li>+ getDbName(): string</li><li>+ getDatabase(): sqlite3*</li><li>+ up(): void</li><li>+ down(): void</li><li>+ getMigration(): int</li><li>+ preparedStatementSQL(const char*, list&lt;variant*&gt;*,sqlite_stmt**): int</li><li>+ read(const char*,list&lt;variant*&gt;*): list&lt;Row*&gt;*</li><li>+ ruSql(string): bool</li></ul>

### IModelDAO:

**Template** criada para servir de interface para todos os DAO's dos modelos.

Nela temos as operações:

**create:** salva um objeto no banco de dados.

**update:** atualiza um objeto no banco de dados.

**list\_all:** lista todos os objetos num banco de dados.

**find:** busca um objeto no banco de dados a partir do seu id.

**remove:** remove um objeto do banco de dados, nessa função há uma sobrecarga do tipo de parâmetro usado, ou um id ou o próprio objeto.

IModelDAO
- helper(DbHelper*)
+ create(T*): bool + update(T*): bool + list_all(): list<T*>* + find(int): T* + remove(int): bool + remove(T*): bool

### UsuárioDAO:

Classe derivada da template IModelDAO que implementa as funções contidas nela específicas para a classe Usuário.

<b>UsuárioDAO: public IModelDAO</b>
+ create(Usuário*): bool + update(Usuário*): bool + list_all(): list<Usuário*>* + find(int): Usuário* + remove(int): bool + remove(Usuário*): bool - getPessoa(Row* row): Usuário*

### ResíduosDAO:

Classe derivada da template IModelDAO que implementa as funções contidas nela específicas para a classe Resíduos.

<b>ResíduosDAO: public IModelDAO</b>
+ create(Resíduos*): bool + update(Resíduos*): bool + list_all(): list<Resíduos*>* + find(int): Resíduos* + remove(int): bool + remove(Resíduos*): bool - getResíduo(Row* row): Resíduos*

### PontoColetaDAO:

Classe derivada da template IModelDAO que implementa as funções contidas nela específicas para a classe Ponto Coleta. Além disso tem como atributo o banco de dados de usuário para que os dois possam se comunicar.

<b>PontoColetaDAO: public IModelDAO</b>
- usuarioDAO(UsuarioDAO*)
+ create(PontoColeta*): bool + update(PontoColeta*): bool + list_all(): list<PontoColeta*> + find(int): PontoColeta* + remove(int): bool + remove(PontoColeta*): bool - getPontoColeta(Row* row): PontoColeta*

### AgendamentoDAO:

Classe derivada da template IModelDAO que implementa as funções contidas nela específicas para a classe Agendamento. Além disso tem como atributo o banco de dados de usuário, ponto de coleta e resíduos para que possa se ter uma comunicação entre os bancos de dados.

<b>AgendamentoDAO: public IModelDAO</b>
- usuarioDAO(UsuarioDAO*) - residuosDAO(ResiduosDAO*) - localDAO(PontoColetaDAO*)
+ create(AgendamentoDAO*): bool + update(AgendamentoDAO*): bool + list_all(): list<AgendamentoDAO*> + find(int): AgendamentoDAO* + remove(int): bool + remove(AgendamentoDAO*): bool - getAgendamentoDAO(Row* row): AgendamentoDAO*

### Row:

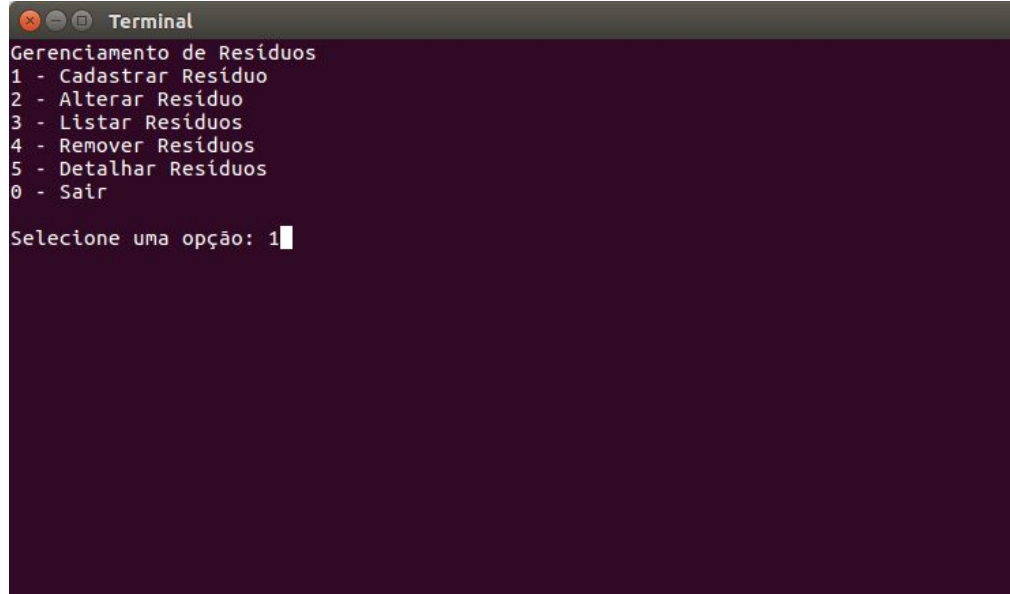
Classe que representa uma linha no banco de dados, criada com o intuito de simplificar o uso do banco por outras classes.

- Feita em cima de uma Map da STL, armazenando o nome da coluna como chave e o valor da coluna/linha.

Row
- *fields: map<string,variant*>
+ field_count(): int + insert(pair<string,variant*>): void + getValuer(string): variant*

### 3. TESTES

Ao compilar o código utilizando o comando **make** e depois **make run** somos levados ao menu Interativo do sistema, comando pela função MainController:

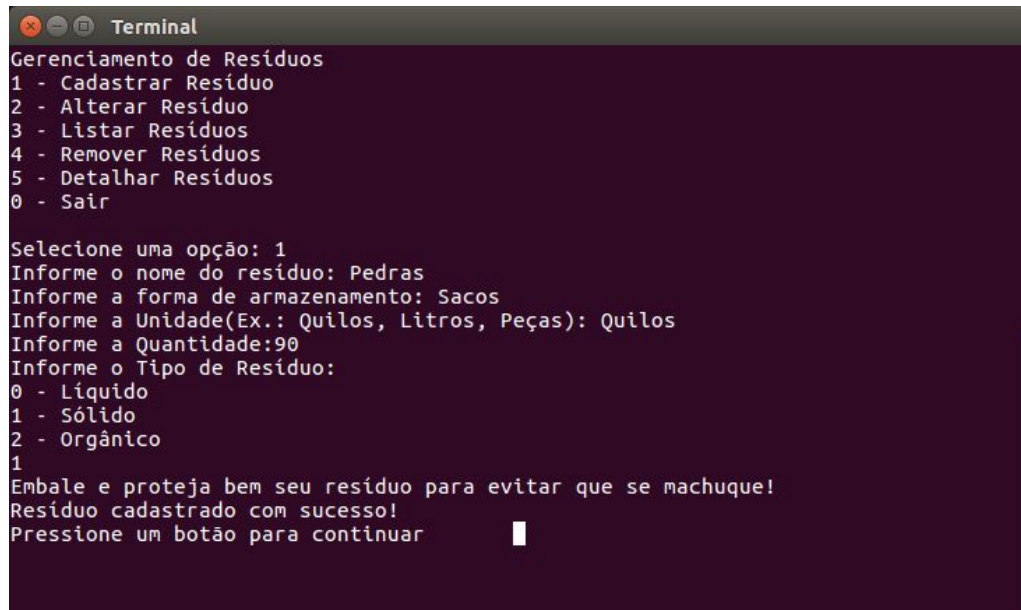


```
Terminal
Gerenciamento de Resíduos
1 - Cadastrar Resíduo
2 - Alterar Resíduo
3 - Listar Resíduos
4 - Remover Resíduos
5 - Detalhar Resíduos
0 - Sair

Selecione uma opção: 1
```

Os testes foram feitos com as classes Resíduos e ResíduosController, mas que apresentam as mesmas funções das outras classes.

Cadastro:



```
Terminal
Gerenciamento de Resíduos
1 - Cadastrar Resíduo
2 - Alterar Resíduo
3 - Listar Resíduos
4 - Remover Resíduos
5 - Detalhar Resíduos
0 - Sair

Selecione uma opção: 1
Informe o nome do resíduo: Pedras
Informe a forma de armazenamento: Sacos
Informe a Unidade(Ex.: Quilos, Litros, Peças): Quilos
Informe a Quantidade:90
Informe o Tipo de Resíduo:
0 - Líquido
1 - Sólido
2 - Orgânico
1
Embale e proteja bem seu resíduo para evitar que se machuque!
Resíduo cadastrado com sucesso!
Pressione um botão para continuar
```

### Alteração:

```
Terminal
Seleção de uma opção: 2
Informe o Id do Resíduo a ser alterado: 4
      Dados do Resíduo
ID: 4
Resíduo: Ferro
Forma de Armazenamento: Caixas
Tipo de Resíduo: Sólido
Quantidade: 160 Quilos

      Atualização
Informe o nome do resíduo: Ferro
Informe a forma de armazenamento: Caixotes
Informe a Unidade(Ex.: Quilos, Litros, Peças): Quilos
Informe a Quantidade:130
Informe o Tipo de Resíduo:
0 - Líquido
1 - Sólido
2 - Orgânico
0

Resíduo alterado com sucesso.
Pressione um botão para continuar
```

### Listar:

```
Terminal
Resíduos cadastrados
ID: 3
Resíduo: Pedra
Forma de Armazenamento: Saco
Tipo de Resíduo: Sólido
Quantidade: 100 Quilos

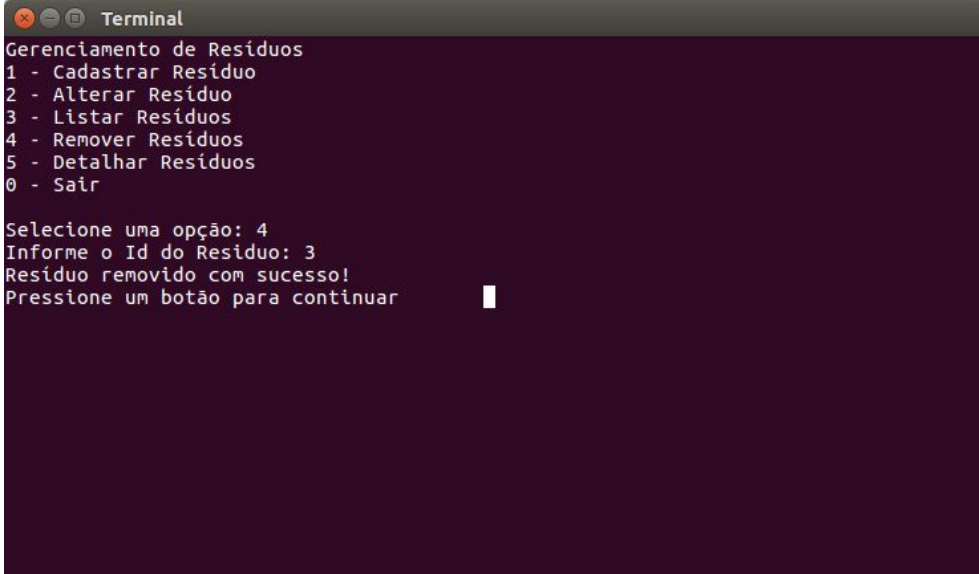
ID: 4
Resíduo: Ferro
Forma de Armazenamento: Caixas
Tipo de Resíduo: Sólido
Quantidade: 160 Quilos

Pressione um botão para continuar
```



Remover:

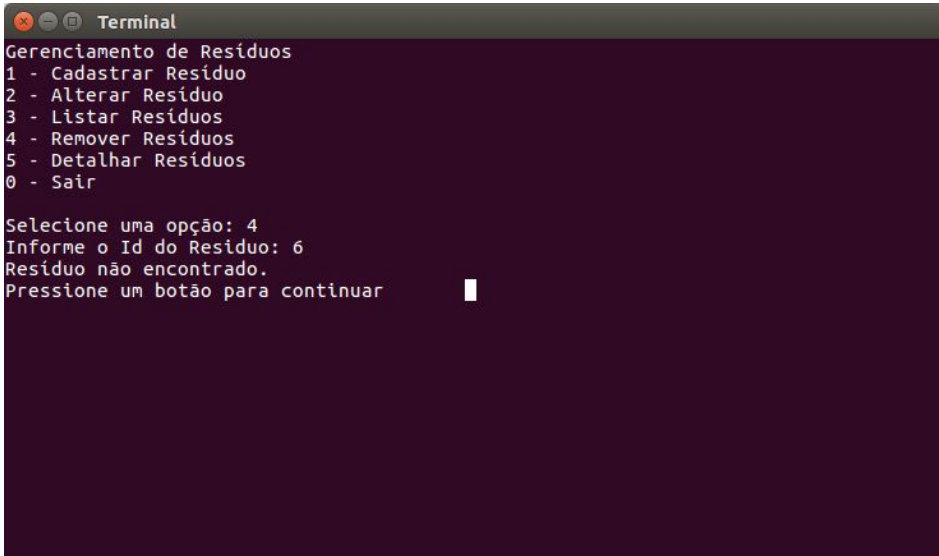
1. ID Existente



```
Terminal
Gerenciamento de Resíduos
1 - Cadastrar Resíduo
2 - Alterar Resíduo
3 - Listar Resíduos
4 - Remover Resíduos
5 - Detalhar Resíduos
0 - Sair

Selecione uma opção: 4
Informe o Id do Resíduo: 3
Resíduo removido com sucesso!
Pressione um botão para continuar
```

2. ID Inexistente



```
Terminal
Gerenciamento de Resíduos
1 - Cadastrar Resíduo
2 - Alterar Resíduo
3 - Listar Resíduos
4 - Remover Resíduos
5 - Detalhar Resíduos
0 - Sair

Selecione uma opção: 4
Informe o Id do Resíduo: 6
Resíduo não encontrado.
Pressione um botão para continuar
```

Detalhar:

```
Terminal
ID: 1
Resíduo: Pedras
Forma de Armazenamento: Sacos
Tipo de Resíduo: Sólido
Quantidade: 100 Kg

Pressione um botão para continuar
```

Casos específicos:

Cadastro de um ponto de coleta associado a um usuário inexistente

```
Terminal
Cadastro de Ponto de Coleta

Informe o nome do local: BH Shopping
Informe o endereço: Belo Horizonte

Deseja vincular o ponto de coleta com um usuário?
0 - NAO
1 - SIM
1
Informe o Id do Usuário: 0
Id não encontrado, o usuário não será vinculado.
Ponto de Coleta cadastrado com sucesso!
Pressione um botão para continuar
```

Resultado,é feito o cadastro da mesma forma mas sem associar a nenhum usuário.

Além disso, com a ajuda do **Framework doctest** foram feitos alguns arquivos de testes para testar o funcionamento das 4 classes principais do nosso código (Usuário, Resíduos, PontoColeta, Agendamento) bem como agilizar a verificação das mesmas após mudanças. Para compilar os arquivos de testes, deve se utilizar do comando **make tests**.

Resultados:

```
[doctest] doctest version is "2.2.0"
[doctest] run with "--help" for options
=====
[doctest] test cases:      7 |      7 passed |      0 failed |      0 skipped
[doctest] assertions:    36 |     36 passed |      0 failed |
[doctest] Status: SUCCESS!
```

**Os testes feitos foram os seguintes:**

Para a classe usuário:

- Teste 1 - Upcasting
- Teste 2 - Construtor Pessoa Física
- Teste 3 - Construtor Pessoa Jurídica

Para a classe resíduos:

- Teste 4 - Construtor Resíduos

Para a classe PontoColeta:

- Teste 5 - Construtor PontoColeta

Para a classe Agendamento:

- Teste 6 - Construtor Agendamento

Para a classe Agendamento Itens:

- Teste 7 - Construtor Agendamento Itens

#### 4. FUNCIONALIDADE EXTRA

A funcionalidade extra proposta é a persistência dos dados, num banco de dados relacional, no caso o SQLite. A escolha do SQLite se dá pela sua portabilidade e facilidade de utilização, visto que é bastante leve e portátil, funcionando em diversas plataformas.

Para interfacear o C++ com o SQLite, foi necessário utilizar a biblioteca libsqlite3-dev, que fornece o ambiente para compilação da interface do C++ com o SQLite, permitindo assim a comunicação com o banco. Além disso, foi necessário a utilização do header “sqlite3.h”.

No desenvolvimento do trabalho foram feitas classes e módulos auxiliares, de modo a realizar uma interface com a biblioteca que comunica com o banco de dados, para facilitar o trabalho e tornar o código legível e mais fácil de ser entendido e alterado, se assim necessário.

Para isso, foi necessário a utilização de containers da STL e criação de estruturas de dados específicas.

Uma dessas estruturas, foi a “variant”, um struct que possui:

- Union contendo string, double e int
- Marcador para definir qual o tipo armazenado na Union.

Desse modo, podemos armazenar strings, double e int num tipo só, economizando o máximo de espaço possível, em conjunto com a criação da struct variant, foram criados métodos auxiliares, para trabalhar com a variant, no caso:

- getInt(): Retorna um int dado uma variant.
- getDouble(): Retorna um double dado uma variant.
- getString(): Retorna uma string dado uma variant.
- getVariant(): Retorna um ponteiro para variant, a partir de um parâmetro, possui uma sobrecarga de três parâmetros, int, double e string

Segue a definição da struct variant e de seus métodos auxiliares:

```
enum dataTypes {
    ... t_string,
    ... t_int,
    ... t_double ...
};

typedef struct struct_variant {
    ... union union_data {
        ... int int_value;
        ... double double_value;
        ... std::string string_value;
    } data;
    ... dataTypes variantType;
} variant;

double getDouble(variant* data);
int getInt(variant* data);
std::string getString(variant* data);
variant* getVariant(int value);
variant* getVariant(double value);
variant* getVariant(std::string value);
```

## **5. CONCLUSÃO**

Durante a elaboração desse trabalho foi possível usar diversos conceitos aprendidos durante o curso de programação e desenvolvimento de software II, ficou claro pro grupo como a aplicação dos conceitos de POO, como modularização, polimorfismo, composição, classes e objetos, encapsulamento, herança facilita e muito a manutenção e leitura do código à medida que ele aumenta sua complexibilidade.

Além disso, o desenvolvimento da funcionalidade extra, proporcionou um grande aprendizado relacionado a utilização de banco de dados relacionais.

A principal dificuldade encontrada foi de começar o projeto, uma vez que pensar em todas as classes e as relações entre elas, bem como os atributos e métodos de cada uma é um processo demorado.

## 6. BIBLIOGRAFIA

1. GEEK for Geeks. Disponível em: <<https://www.geeksforgeeks.org/>>. Acesso em: 09 dez. 2018.
2. CPLUSPLUS. Disponível em: <<http://www.cplusplus.com/>>. Acesso em: 09 dez. 2018.
3. Slides dados em Sala de Aula. Disponíveis no Moodle.
4. DUDLER, Roger. **Git - the simple guide**. Disponível em: <<https://rogerdudler.github.io/git-guide/>>. Acesso em: 09 dez. 2018.