UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

ENGENHARIA DE COMPUTAÇÃO

HERIC DA SILVA CRUZ - WILLIAM MARREIRO BRITO

TRABALHO AVALIATIVO DE ESTRUTURA DE DADOS - 02

QUIXADÁ

2022

0.1 Objetivo:

O trabalho teve como objetivo principal explicitar e pôr em prática conhecimentos relacionados a criação e implementação de listas encadeadas.

0.2 Especificação dos tipos abstractos de dados e estrutura de dados utilizados.

Para a implementação do projeto foram usados num total 4 estruturas, entre elas duas classes e duas estruturas de dados (listas encadeadas).

A organização das Estruturas do trabalho ficou da seguinte forma:

CommunityList tem como atributo classes community, as classe communnity tem como atributo uma lista de users e as classes user seguem como base, ou seja, a hierarquia vai de forma decescente de CommunityList a User

User: A classe User foi um dos tipos base do projeto, sua implementação tem um baixo nível de complexidade em comparação com a implementação das estruturas de dados. Um ponto a se destacar é que a classe user é a classe base de todo o projeto, pois a partir dela se deriva a UserList, que é um dos atributos da classe Community.

```cpp
class User
{
private:
    std::string name;  // Nome do usuario
    std::string email; // Email do usuario
    int id;            // Identificador do usuario

public:
    User();                                          // Construtor padrao
    User(std::string name, std::string email, int id); // Construtor
    ~User();                                         // Destrutor
    void print();                                    // Imprime os dados do usuario
    int get_id();                                    // Retorna o id do usuario
    std::string get_name();                          // Retorna o nome do usuario
    std::string get_email();                         // Retorna o email do usuario
};
```

Community: Semelhante à classe User, a classe Community possui um nível de abstração baixo, com operações simples que realizam o básico de uma classe implementada em c++. A partir da classe Community, se deriva a CommunityList.

```cpp
class Community
{
private:
    std::string name;    // Nome da comunidade
    int numberOfMembers; // Numero de membros
    int id;              // Identificador da comunidade
    UserList members;    // Lista de membros da comunidade

public:
    Community();                          // Construtor padrao
    Community(std::string name, int id); // Construtor
    ~Community();                         // Destrutor
    void add_member(User user);           // Adiciona um membro
    void remove_member(User user);        // Remove um membro
    void printMembers();                  // Imprime os membros da comunidade
    void print();                         // Imprime os dados da comunidade
    int get_id();                         // Retorna o id da comunidade
    bool contains_user(User user);        // Verifica se o usuario esta na comunidade
    std::string get_name();               // Retorna o nome da comunidade
    int get_numberOfMembers();            // Retorna o numero de membros da comunidade
    UserList get_members();               // Retorna a lista de membros da comunidade
    User get_member(std::string name);    // Retorna o usuario com o nome passado como parametro
};
```

UserList: A classe UserList é uma lista simplesmente encadeada que realiza diversas operações cujo o principal intuito e suprir as necessidade do sistema AvCOM. Presente nela principalmente operações a serem usadas pela classe Community.

```cpp
struct NodeUser
{
    User data;      // Dados do usuario
    NodeUser *next; // Proximo elemento da lista

    NodeUser(const User data, NodeUser *next = nullptr) // Construtor
    {
        this->data = data;
        this->next = next;
    }
};

class UserList
{
private:
    NodeUser *head; // Primeiro elemento da lista
    NodeUser *tail; // Ultimo elemento da lista
    int Listsize;   // Tamanho da lista

public:
    UserList();                             // Construtor
    ~UserList();                            // Destrutor
    void add(User &data);                   // Adiciona um elemento ordenado por nome
    void removeByName(std::string name);    // Remove um elemento da lista (por nome)
    void removeByEmail(std::string email);  // Remove um elemento da lista (por email)
    void print();                           // Imprime os elementos da lista
    bool containsByName(std::string name);  // Verifica se um elemento esta na lista (por nome)
    bool containsByEmail(std::string email); // Verifica se um elemento esta na lista (por email)
    User getByName(std::string name);       // Retorna um elemento da lista (por nome)
    User getByEmail(std::string email);     // Retorna um elemento da lista (por email)
    int get_size();                         // Retorna o tamanho da listao
};

```

CommunityList: A class CommunityList trata-se de uma lista encadeada simples que usa uma struct node como seu elemento principal. Na sua implementação pode se observar um maior nível de abstração das funções, na maioria dela apenas são chamadas funções de classes mais baixas da hierarquia apresentada no início do trabalho.

```cpp
struct NodeCommunity
{
    Community data;      // Dados da comunidade
    NodeCommunity *next; // Proximo elemento da lista

    NodeCommunity(const Community data, NodeCommunity *next = nullptr) // Construtor
    {
        this->data = data;
        this->next = next;
    }
};

class CommunityList
{
private:
    NodeCommunity *head; // Primeiro elemento da lista
    NodeCommunity *tail; // Ultimo elemento da lista
    int size;            // Tamanho da lista

public:
    CommunityList();                                        // Construtor
    ~CommunityList();                                       // Destrutor
    void add(Community &data);                              // Adiciona um elemento no final da lista ordenado por nome
    void remove(std::string name);                          // Remove um elemento da lista e todos os seus membros
    void printMembers(std::string name);                    // Imprime os membros de uma comunidade
    void printCommunitiesByUser(User &user);                // Imprime as comunidades que um usuario participa
    void removeMember(User &user);                          // Remove um usuario de todas as comunidades que ele participa
    void print();                                           // Imprime os elementos da lista
    void includeInCommunity(User &user, Community &community);  // Adiciona um usuario em uma comunidade
    void removeFromCommunity(User &user, Community &community); // Remove um usuario de uma comunidade
    bool contains(std::string name);                        // Verifica se um elemento esta na lista
    Community get(std::string name);                        // Retorna um elemento da lista
    User getUserByName(std::string name);                   // Retorna um usuario da lista de membros de uma comunidade
    int get_size();                                         // Retorna o tamanho da lista
    void PrintCommunitiesAndMembers();                      // Imprime as comunidades e seus membros
};
```

0.3 Tutorial de compilação:

OBS: O tutorial de compilação funciona somente em sistemas linux baseados em debian. Para a compilação será necessario a instalação de alguns pacotes:

G++ (Compilador do C++): Certifique-se de ter o GCC instalado Embora você use o VS Code para editar o código-fonte, você usará o compilador g++ para compilar o código-fonte no Linux. Você também usará o GDB para depuração. O Ubuntu não instala essas ferramentas por padrão, você mesmo precisa instalá-las. Felizmente, é fácil. Primeiro, verifique se o GCC está instalado. Para verificar se está correto, abra uma janela de terminal e digite o seguinte comando:

gcc -v

Caso não esteja instalado execute o seguinte comando para certificar que a biblioteca de pacotes está atualizada:

sudo apt-get update

Depois execute os seguites comandos:

sudo apt-get install build-essential gdb

sudo apt-get install g++

sudo apt-get install make

Com as ferramentas instaladas basta abrir o diretório do projeto e executar o comando:

make

Esse comando irá compilar e em seguida já executará o código. Caso queira apenas executar o código já compilado, basta executar:

make run

0.4 Estrutura do makefile:

Na estrutura do makefile foram criadas variáveis com o objetivo de facilitar a mudança de diretório caso necessário. Fora isso a forma que o código foi discposto permite que a compilação das bibliotecas pode tanto ser feito de forma individual quanto todas juntas.

```
APPS = ./apps
BIN = ./bin
INCLUDE = ./include
OBJ = ./obj
SRC = ./src

all: libs myapp clear run

mainEx: myapp clear run


myapp:
    g++ $(APPS)/program.cpp $(OBJ)/*.o -I $(INCLUDE) -o $(BIN)/program

libUser:
    g++ -c $(SRC)/User.cpp -I $(INCLUDE) -o $(OBJ)/User.o

libUserList:
    g++ -c $(SRC)/UserList.cpp -I $(INCLUDE) -o $(OBJ)/UserList.o

libCommunity:
    g++ -c $(SRC)/Community.cpp -I $(INCLUDE) -o $(OBJ)/Community.o

libCommunityList:
    g++ -c $(SRC)/CommunityList.cpp -I $(INCLUDE) -o $(OBJ)/CommunityList.o

libs: libCommunity libUser libUserList libCommunityList

run:
    $(BIN)/program

clear:
    clear

clean:
    rm $(BIN)/*
    rm $(OBJ)/*
```

0.5 Observações gerais:

Por decisão da equipe não usamos o nome das funções explicitadas no pdf de instruções para a implementação dos dois tipos de lista, no entando todas as funções estão presentes no código, mas com nome diferente. A função main presente no projeto mostra as funções explicitadas no pdf sendo usadas.

0.6 Como o trabalho foi dividido:

Ambos os membros tiveram participação ativa na implementação do código, não existiu uma separação explicita de responsabilidades mas em geral a cada um implementou uma classe e uma lista dessa classe, com ambos trabalhando para resolver os problemas de implementação e pensando em soluções que pudessem facilitar a implementação do projeto.