Universidade de São Paulo
Instituto de Física de São Carlos

Trabalho Prático - Parceiros de Trabalho

Disciplina: Estrutura de Dados
Professor: Thiago A. S. Pardo

Marvin Mendes Cabral (11212268) e Matheus da Silva Fonseca (11212247)

São Carlos
2021

1 Descrição do Trabalho

Neste trabalho desenvolvemos um protótipo de 'rede social' de parceiros de trabalho utilizando as técnicas aprendidas na disciplina de Estrura de Dados. Assim, desenvolvemos um TAD que é capaz de armazenar as informações sobre os usuários da rede social de maneira dinâmica e encadeada, sendo seu nome, apelido, parceiros, contatos, mensagens e pedidos de colaboração, bem como realizar as devidas operações.

2 Estruturas utilizadas e justificativas

2.1 User e Users

```
typedef struct usuario{
    char nome[50];
    char apelido[30];
    struct  usuario *prox;

    Lista *parceiros; // Lista de parceiros
    Fila *pedidos; // Fila de pedidos
    Pilha *mensagens; // Pilha de mensagens
} User;

typedef struct Lista_usuarios{
    User *inicio, *fim;
} Users;
```

Primeiramente precisávamos de uma estrutura que guardasse a representação de um usuário da rede social, escolhemos este modo do código acima pela facilidade de se ler e entender para que serve cada pedaço da estrutura, além de permitir a dinamicidade da alocação de memória, e o encadeamento com outros usuários ou "blocos".

A struct "User" é o bloco de informação que guarda uma string com o nome do usuário, outra com o apelido e além disso temos um ponteiro que aponta para o próximo usuário, e a struct "Users" representa o conjunto destes blocos indicando o primeiro e o último deles.

2.2 Lista

```
typedef struct bloco{
    struct bloco *prox;
    elem info[30];
} no;

typedef struct lista{
    no *init, *fim;
} Lista;
```

Utilizamos da estrutura lista para conseguir armazenar de forma dinâmica e encadeada informações sobre quais eram os parceiros de cada usuário, além disso, precisávamos poder acessar qualquer um dos parceiros a qualquer momento, e nisso uma lista também se mostrou mais adequada.

A forma com que foi guardada essa informação foi com a lista contendo strings, sendo estas o apelido de cada um dos parceiros do usuário, com isso conseguíamos chegar ao bloco deste parceiro.

2.3 Fila

```
typedef struct block{
    struct block *prox;
    elem info[30];
} no_F;

typedef struct Fila{
    no_F *init, *fim;
} Fila;
```

A fila foi utilizada pois queremos que os pedidos de parceria que chegam primeiro fossem mostrados primeiro ao usuário que fosse avaliar os pedidos recebidos, as funções de entrada e saída da fila são perfeitas para isso, a de saída além de retirar um elemento da fila, o guardava numa variável, que utilizaríamos depois.

As informações guardadas na fila são strings, cada uma sendo o apelido de alguém que fez um pedido para o usuário, tendo o apelido, conseguimos encontrar o bloco User dele e modificar sua lista de parceiros, caso o pedido seja aceito.

2.4 Pilha

```
typedef struct Piece{
    struct Piece *prox;
    elem info[140];
} no_P;

typedef struct Pilha{
    no_P *init;
} Pilha;
```

A pilha foi escolhida pelo formato que se desejavam ser mostradas as imagens, isto é, queremos que a mensagem mais recente seja mostrada primeiro, e pensando na pilha, a mensagem mais recente ficaria em seu topo e quando formos retirar alguma para exibição, a primeira seria ela.

Como na lista e na fila, as informações na pilha foram guardadas em forma de strings, com um limite de caracteres maior que para os nomes ou apelidos.

3 Detalhes de compilação e execução

3.1 Compilação

Para fazer a compilação utilizamos o gcc, versões 7.5.0 e 9.3.0. Como já comentado foram feitos 4 TAD's no projeto: Lista, Fila, Pilha e Users. Para cada um deles precisamos gerar um *object file*, bem como para o arquivo principal (main.c), utilizando o comando 'gcc -c' e depois realizar o *link* entre eles para criar o executável (ex). Por fim, rodamos o programa pela maneira usual (./). Utilizamos um script em bash para realizar todas as estapas de maneira mais rápida, este está apresentado no código abaixo:

```
#!/bin/bash

gcc -c lista.c
gcc -c Fila.c
gcc -c pilha.c
gcc -c rede_social.c
gcc -c main.c
gcc -o ex lista.o rede_social.o Fila.o pilha.o main.o
./ex
```

Vale comentar que utilizamos também a biblioteca **string.h** para faciliar a atribuição e comparação entre *strings*.

3.2 Execução do programa

Após ser iniciado o programa apresenta o que chamaremos de painel principal, ou seja, uma tela com a opção de todas as ações que o usuário pode realizar, apresentada na figura 1, juntamente com um teste do Cadastro (opção 1) , utilizado para mostrar já o processo de escolha. As imagens seguintes irão exemplificar uma execução completa do programa, nela são cadastradas 6 pessoas de nomes A, B, C, D, E, F e apelidos a, b, c, d, e, f, respectivamente. As amizades formadas serão:

- $a \rightarrow b, c, d, e$
- $b \rightarrow a, c$
- $c \rightarrow a, b, e$
- $d \rightarrow a$
- $e \rightarrow a, c$
- $f \rightarrow$

As demais explicações serão expostas nas legendas das figuras. Ao final ainda apresentamos figuras exemplificando como lidar com possíveis excessões (ou erros) que surgirem.

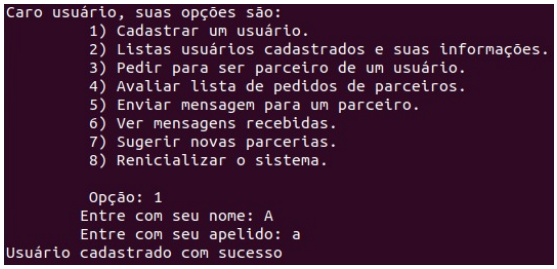
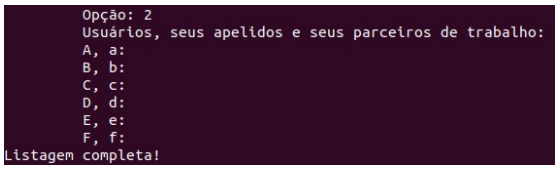
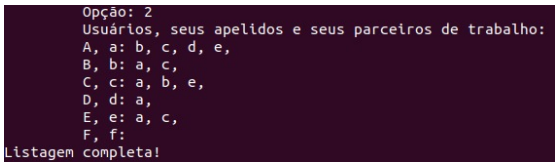


Figure 1: Exemplo do Menu e demonstração da operação de cadastro.



(a)



(b)

Figure 2: Demonstração da operação de mostrar usuários cadastrados e parceiros 2a antes das pareceiras serem realziadas e 2b depois das parcerias serem realziadas.

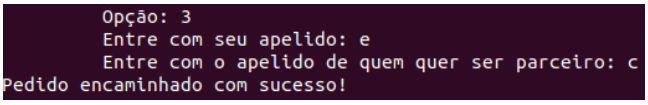


Figure 3: Demonstração da função de enviar pedido de parceria.

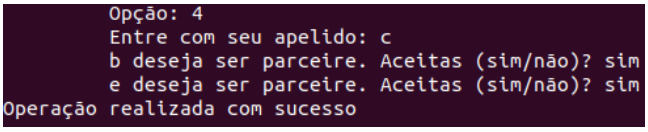
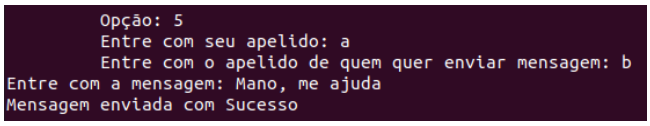
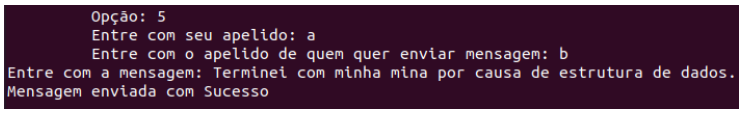


Figure 4: Demonstração da função de avaliar pedido de parceria.



(a)



(b)

Figure 5: Demonstração da operação de enviar mensagem. Foram enviadas duas para teste, sendo a primeira da imagem 5a e a segunda da imagem 5b.

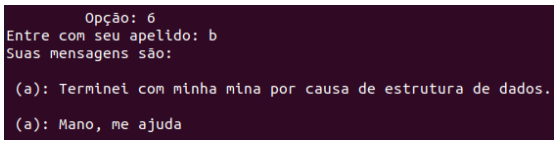


Figure 6: Demonstração da função de ver mensagens.

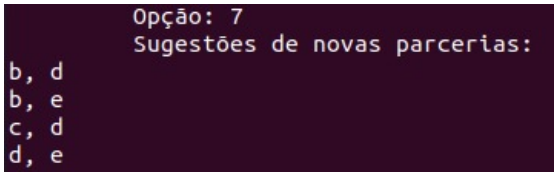


Figure 7: Demonstração da função de sugerir amizades.

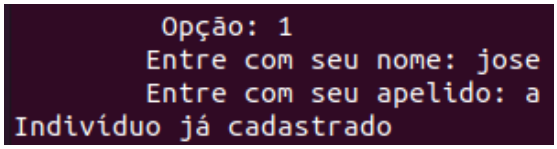


Figure 8: Demonstração de um erro possível ao se cadastrar um usuário no sistema, onde se tentar cadastrar um apelido já utilizado

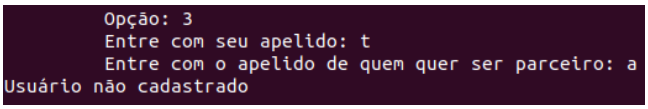


Figure 9: Demonstração de um erro que pode aparecer em todas as opções que pedem um apelido após serem chamadas, aqui demonstrado na opção 3. Neste erro o apelido indicado não foi cadastrado.

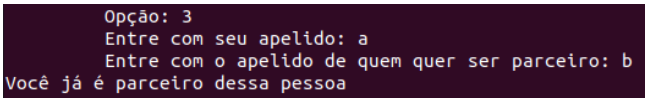


Figure 10: Demonstração de erro na opção 3, ocorre quando um usuário solicita ser parceiro de alguém que já está em sua lista de parceiros.

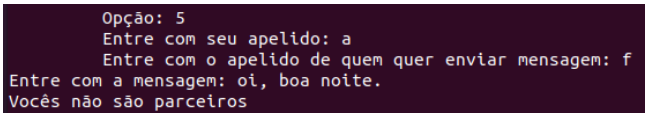


Figure 11: Demonstração de erro na opção 5, quando um usuário tenta enviar uma mensagem a alguém que não está em sua lista de parceiros.

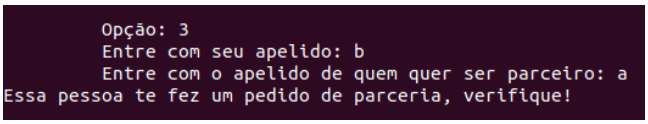


Figure 12: Demonstração de erro na opção 3, nesse caso a havia feito um pedido de parceria para b e b tentou fazer um pedido para a, então foi informado que já havia um pedido de parceria entre os dois.