Projeto 2: Concordo

Clone do Discord, para humanos mais civilizados.

Backend vs Frontend

Em aplicações conectadas é comum usarmos o termo backend e frontend. Usualmente aplicações na internet usam esses dois termos para separar o que “executa do lado do usuário/cliente”, chamado de FRONTend do que “executa do lado do servidor” da aplicação, ou no(s) computador(es) que processam informações vitais para a aplicação, informações que ficam “por trás" da aplicação daí o nome BACKend.

Enquanto as aplicações de *frontend* focam na experiência do usuário, com interfaces mais rebuscadas e infinitas formas de deixar o visual cada vez mais bonito, aplicações de backend são mais parecidas com o que desenvolvemos na disciplina, usualmente tem interface textual e são mais voltadas para desempenho e processamento de dados.

Create/Read/Update/Delete - Crud

Quando tratamos de Backend, embora também esteja presente no frontend em muitos casos, é comum o uso do termo CRUD, que refere as principais manipulações que fazemos nos dados que modelam a aplicação.

Entender como modelar os dados e como realizar atividades de crud é um conhecimento muito importante em diversas áreas, mais especificamente na área de serviços web e distribuídos.

Normalmente o problema já tem um conjunto de dados em sua própria especificação e o trabalho de programação está em modelar classes que representam os dados e realizar as ações sobre os dados como especificado pela aplicação.

Concordo

Nessa atividade você irá criar um sistema chamado "Concordo" com recursos similares ao Discord, mas que vai funcionar somente em modo texto e sem recursos de rede. A ideia principal é simular o “backend” de um serviço com o discord, que, embora de forma simplificada, serve para dar uma boa ideia de como as coisas são feitas nesse nicho de aplicação.

Para a realização desta atividade os seguintes recursos do C++ serão necessários:

- Classes e Objetos
- Standard Template Library (STL)
  - vector
  - map
  - pair

Alguns ponteiros para os recursos citados acima serão enviados no canal da atividade 2 no Discord.

Resumidamente, as seguintes entidades deverão existir no sistema a ser desenvolvido:

- Usuários: informações de uma conta no sistema.
- Servidores: Cada um com um vetor contendo todos os canais de texto deste servidor
    - CanalTexto: Com várias mensagens
        - Mensagens: Escrita por um usuário, em uma data/hora
- Sistema: Inicia o controle dos comandos passando as informações para as entidades responsáveis ou manipulando os parâmetros do comando em sí.

O sistema deverá ser operado através de comandos de uma única linha, compostos por um nome, seguido de parâmetros. Ele deve interpretar, processar e gerar uma saída para cada comando, de acordo com o resultado de seu processamento. Cada comando será especificado em detalhes ao longo desse documento. Exemplo de um comando:

```
create-user <email> <senha_sem_espaços> <nome com espaços>
```

As funcionalidades do sistema serão separadas em duas partes, na primeira parte deverá ser implementado o processamento dos comandos relacionados a Usuário (todos), Servidores (todos) e Canais (somente a criação de canais). Não é necessário nessa primeira parte implementar o recurso de entrar em um canal nem enviar e ler mensagens dentro dele.

Na segunda parte deverá ser implementado os demais comandos do sistema, complementando as operações em Usuário, Servidor, Canais e Mensagens.

Estrutura das classes

- **Usuario**
    - Atributos

Atributo / Tipo	Descrição
id / int	um identificador único para o usuário
nome / string	O nome cadastrado pelo usuário, conforme o comando create-user
email / string	O email cadastrado pelo usuário, conforme o comando create-user
senha / string	A senha cadastrada pelo usuário, conforme o comando create-user

- **Mensagem**
    - Atributos

Atributo / Tipo	Descrição

dataHora / string	Um texto representando um timestamp com a data e hora em que a mensagem foi enviada no formato <DD/MM/AAAA - HH:MM>, exemplo: <08/03/2021 - 11:53>
enviadaPor / int	Id do usuário que enviou a mensagem
conteúdo / string	Conteúdo da mensagem

- **CanalTexto**
  - Atributos

Atributo / Tipo	Descrição
nome / string	O nome do canal, conforme o comando create-channel
mensagens / vector<Mensagem>	Um vector com mensagens que foram enviadas neste canal

- **Servidor**
  - Atributos

Atributo / Tipo	Descrição
usuarioDonoId / int	Id do usuário que criou o servidor
nome / string	Nome do servidor, passado no comando create-server
descrição / string	Descrição do servidor passada no comando set-server-desc
codigoConvite / string	O código de convite do servidor, se houver. Passado no comando set-server-invite-code. Por padrão deve ser uma deve ser uma string vazia.
canaisTexto / vector<CanalTexto>	Um vetor contendo todos os canais de texto deste servidor
participantesIDs / vector<int>	Um vetor contendo os ids de todos os participantes deste servidor. Um usuário vira participante de um servidor após usar o comando enter-server.

- **Sistema**
  - Atributos

Atributo / Tipo	Descrição
usuarios / vector<Usuario>	Um vetor contendo todos os usuários criados usando o comando create-user
servidores / vector<Servidor>	Um vetor com todos os servidores criados usando o comando create-server

usuariosLogados / map <int, pair<string, string> >	Uma tabela que mapeia os usuários logados atualmente no sistema em um par com duas strings representando o servidor sendo visualizado e o canal de texto, respectivamente. Quando um usuário muda/sai de um servidor ou de canal, esta tabela deve ser atualizada para refletir essa mudança. Quando um usuário logado sai do Concordo (com o comando disconnect) ele deve ser removido desta tabela. Esta tabela reflete exatamente qual o Canal e Servidor atual que estão sendo **visualizados** pelo usuário. Quando um usuário não está visualizando um canal o segundo campo deve ser uma string vazia; quando não está visualizando um Servidor os dois campos devem ser strings vazias (pois um usuário só pode visualizar um canal que está dentro de um servidor que ele esteja visualizando).

A classe **Sistema** só é instanciada uma vez durante o ciclo de vida da aplicação e é responsável por manter os dados "raiz" da aplicação.

Observe que a classe Usuário tem um atributo chamado "id" que é inteiro. Esse ID deve ser gerado automaticamente pelo sistema durante o cadastro e deve ser incremental, ou seja, o primeiro usuário criado deve ter id==1, o segundo id==2 e assim por diante.

Todas as referências para um usuário devem ser feitas por esse ID. Por exemplo, na classe Servidor existe um atributo chamado usuarioDonoId que é o ID do usuário que criou o servidor. Assim como existe também um vetor de inteiros chamado participantesIDs, que trata-se de uma lista de IDs dos usuários que estão no servidor.

Por esse motivo, será importante criar no sistema uma forma de encontrar o objeto Usuario, da lista de usuários, buscando pelo seu ID.

Nas Partes 1 e 2, uma série de comandos do sistema deverão ser implementados e todos os comandos são passíveis de serem executados em qualquer ponto do programa. No entanto, alguns comandos só funcionam caso alguns comandos anteriores tenham sido executados, por exemplo, o comando create-server só funciona se o usuário estiver logado, assim como o comando create-channel só funciona se o usuário estiver em algum servidor. Essas informações (se o usuário está logado, se ele está em algum servidor, etc.) são chamadas de *estado* da aplicação, e devem ser controladas pelo programa à medida que o usuário interage com o sistema.

Funcionalidades (parte 1):

Neste trabalho, usaremos comandos de texto ao invés de menus. Por esta razão, o sistema não será dividido em “telas” mas receberá comandos de texto para executar as funcionalidades. Cada comando altera o estado interno do programa e retorna um texto com o resultado.

Exemplo:

```
$./concordo
create-user isaacfranco@imd.ufrn.br senha12345 Isaac Franco Fernandes
```

```
Criando usuário Isaac Franco Fernandes (isaacfranco@imd.ufrn.br)
Usuário criado
login isaacfranco@imd.ufrn.br senhaerrada
Senha ou usuário inválidos!
login isaacfranco@imd.ufrn.br senha12345
Logado como isaacfranco@imd.ufrn.br
create-server disciplina_lp1
Servidor criado
disconnect
Desconectando usuário isaacfranco@imd.ufrn.br
login joaquim@uol.com.br senhajoaquim
Senha ou usuário inválidos!
quit
Saindo...
```

Para testar o sistema será possível digitar os comandos, um a um, e observar a saída de cada um deles (como no exemplo acima), ou, se desejar, um arquivo de texto com uma sequência de comandos pode ser criado e ser utilizado como entrada do seu sistema utilizando redirecionamento, como pode ser visto abaixo:

```
$./concordo < script.txt
Criando usuário Isaac Franco Fernandes (isaacfranco@imd.ufrn.br)
Usuário criado
Senha ou usuário inválidos!
Logado como isaacfranco@imd.ufrn.br
Servidor criado
Desconectando usuário isaacfranco@imd.ufrn.br
Senha ou usuário inválidos!
Saindo...
```

Observe que nesse caso só será exibido na tela os resultados dos comandos, já que os comandos em si estão no arquivo script.txt (um por linha). Se você implementou seu sistema para receber os comandos do teclado, esse recurso de redirecionamento deve funcionar sem a necessidade de se alterar nada no sistema. Essa é só uma dica para facilitar os testes sem digitar as mesmas coisas várias vezes.

A1 - Se não estiver logado (assim que entra no sistema)

Assim que executar o sistema o usuário não estará logado e somente os seguintes comandos não retornam uma mensagem de erro:

- A1.1 Sair do sistema : comando *quit*
- A1.2 Criar usuário : comando *create-user <email> <senha_sem_espaços> <nome com espaços>*
- A1.3 Entrar no sistema : `login <email> <senha>`

A1.1 - Sair do sistema

Fechar a aplicação, **este comando pode ser executado a qualquer momento pelo usuário**.

- Exemplo de entrada/saída:

```
quit
“Saindo do Concordo”
```

A1.2 - Criar usuário

Deve ser informado e-mail, senha e nome do usuário e tentar cadastrar o usuário no sistema. Caso consiga, deve emitir um texto informando que o usuário foi inserido com sucesso.

Lembre-se que o ID do usuário deve ser criado automaticamente pelo sistema, como dito anteriormente.

O cadastro só deve ser efetivado se o e-mail não existir no cadastro geral de usuários.

- Exemplo de entrada/saída:

```
create-user julio.melo@imd.ufrn.br 12ab34cd Julio Melo
“Usuário criado”
```

```
create-user julio.melo@imd.ufrn.br 12ab34cd Julio Cesar
“Usuário já existe!”
```

Tratamento de erros: deverá ser realizada checagem se o usuário já existe e uma mensagem de retorno apropriada deve ser adicionada.

A1.3 - Entrar no sistema

Esse procedimento verifica se já existe um usuário no cadastro geral com esse e-mail e senha digitados. Se existir, então o usuário efetuou o login com sucesso.

Observação: Efetuar o login significa que você precisa armazenar a informação de que usuário está logado no sistema para que as outras operações possam saber quem está logado no momento. Para tanto use a tabela usuariosLogados, adicionando o id do usuário à tabela, mas com valores nulos de Servidor e Canal.

- Exemplo de entrada/saída:

```
login julio.melo@imd.ufrn.br 12ab34cd
“Logado como julio.melo@imd.ufrn.br”
```

```
login julio.melo@imd.ufrn.br 1326
“Senha ou usuário inválidos!”
```

```
login julio.engcomp@gmail.com 1326
“Senha ou usuário inválidos!”
```

Tratamento de erros: deverá ser realizada checagem se o login é válido e uma mensagem de retorno apropriada deve ser adicionada.

A2 - Interações básicas com Servidores (se estiver logado)

Caso o usuário já tenha feito seu cadastro e o procedimento de entrar no sistema, as seguintes opções são apresentadas para ele:

- A2.1 - Desconectar do Concordo : comando `disconnect <id-de-usuario-logado>`
- A2.2 - Criar servidores : comando `create-server <id-de-usuario-logado> <nome>`
- A2.4 - Setar código de convite para o servidor : comando `set-server-invite-code <id-de-usuario-logado> <nome> <codigo>`
- A2.5 - Listar servidores : comando `list-servers <id-de-usuario-logado>`
- A2.6 - Remover servidor: comando `remove-server <id-de-usuario-logado> <nome>`
- A2.7 - Entrar em um servidor: comando `enter-server <id-de-usuario-logado> <nome>`
- A2.8 - Sair de um servidor: comando `leave-server <id-de-usuario-logado> <nome>`
- A2.9 - Listar pessoas no servidor: comando `list-participants <id-de-usuario-logado>`

Perceba que todas as operações recebem como argumento um `<id-de-usuario-logado>`. Este id deve ser sempre verificado na tabela usuariosLogados para que seja verificado se o usuário que tem aquele id está logado no sistema (chamou o comando `login`). Você usará esse id para manter a tabela usuariosLogados atualizada ou para realizar as operações desejadas sobre os dados corretos. Assim, quando o comando a ser executado for `list-participants 0`, o sistema deve listar os participantes do servidor em que o usuário com id 0 está visualizando no momento de acordo com a tabela usuariosLogados.

Tratamento de erros: Como todos os comandos recebem um `<id-de-usuario-logado>` é importante que seja sempre checado se o usuário que está sinalizado está logado no momento. Caso o usuário enviado no comando não esteja logado uma mensagem de erro deve ser impressa e o comando deve ser ignorado.

A2.1 - Desconectar do Concordo

Desconecta o usuário com id passado, ou seja, atualiza o atributo que armazena os usuários atualmente logados. A estrutura do comando é a seguinte `disconnect <id-de-usuario-logado>`.

- Exemplo de entrada/saída:

```
disconnect 0
```

```
“Desconectando usuário julio.melo@imd.ufrn.br”
```

```
disconnect 0
“Desconectando usuário isaacfranco@imd.ufrn.br”
disconnect 0
“Não está conectado”
```

Tratamento de erros: deverá ser realizada checagem se o usuário já está conectado e uma mensagem de retorno apropriada deve ser adicionada.

A2.2 - Criar servidores (nome)

Seu sistema deve ter uma funcionalidade de criar um novo servidor passando o nome dele. O comando `create-server <id-de-usuario-logado> <nome-do-servidor>` cria um novo servidor se ele não existir com esse nome.

Observe que o servidor tem um "dono", e na criação de um novo servidor esse dono deve ser o usuário passado no `<id-de-usuario-logado>`.

Você pode mudar a descrição para um servidor já criado (se ele for seu) com o comando `set-server-desc <id-de-usuario-logado> <nome-do-servidor>`.

Observe também que todo servidor é criado sem codigoDeConvite, ou seja, qualquer usuário pode entrar no servidor. Mais à frente está detalhado como adicionar um código de convite no servidor.

- Exemplo de entrada/saída:

```
create-server 0 minha-casa
“Servidor criado”
```

```
create-server 0 minha-casa
“Servidor com esse nome já existe”
```

Tratamento de erros: deverá ser realizada checagem se o servidor já existe e uma mensagem de retorno apropriada deve ser adicionada.

A2.3 - Mudar a descrição do servidor

Deve ser verificado se o servidor que você está tentando mudar a descrição é seu.

- Exemplos de entrada/saída:

```
set-server-desc 0 minha-casa “Este servidor serve como minha
casa, sempre estarei nele”
“Descrição do servidor ‘minha-casa’ modificada!”
```

```
set-server-desc 0 minha-casa2 “Este servidor serve como minha
casa, sempre estarei nele”
“Servidor ‘minha-casa2’ não existe”
```

```
set-server-desc 0 minha-casa55 “Trocando a descrição de servidor
dos outros”
“Você não pode alterar a descrição de um servidor que não foi
criado por você”
```

Tratamento de erros: deverá ser realizada checagem se o usuário é dono do servidor e uma mensagem de retorno apropriada deve ser adicionada.

A2.4 - Setar código de convite para o servidor

Deve ser verificado se o servidor que você está tentando mudar o código de convite é seu.

Se você utilizar o comando sem nenhum código, então o servidor muda seu código para "" e fica liberado para qualquer pessoa entrar.

- Exemplos de entrada/saída:

```
set-server-invite-code 0 minha-casa 4567
"Código de convite do servidor 'minha-casa' modificado!"
```

```
set-server-invite-code 0 minha-casa
"Código de convite do servidor 'minha-casa' removido!"
```

Tratamento de erros: deverá ser realizada checagem se o usuário é dono do servidor e uma mensagem de retorno apropriada deve ser adicionada.

A2.5 - Listar servidores

Exibe os nomes dos servidores no sistema, um por linha.

- Exemplo de entrada/saída:

```
list-servers 0
minha-casa
minha-casa2
RPG-galera
Bate-papo-uol
```

A2.6 - Remover servidor

Deve remover um servidor (informando o seu nome). Só pode ter sucesso na remoção se o dono do servidor for o usuário logado. Veja que se um usuário remove o servidor que ele está visualizando, este comando deve atualizar a tabela usuariosLogados de forma que ele não esteja mais visualizando o servidor removido.

IMPORTANTE: Quando um servidor é removido é necessário verificar a tabela usuariosLogados para atualizar informações de outros usuários que possam estar visualizando aquele servidor. Caso um usuário esteja visualizando um servidor

removido, este comando deve atualizar a tabela para refletir que aquele usuário não está mais visualizando qualquer coisa (campos correspondentes ao Servidor e CanalTexto que estão sendo visualizados recebem strings vazias).

- Exemplo de entrada/saída:

```
remove-server 0 minha-casa
"Servidor 'minha-casa' removido"
```

```
remove-server 0 minha-casa
"Você não é o dono do servidor 'minha-casa'"
```

```
remove-server 0 minha-casa3
"Servidor 'minha-casa3' não encontrado"
```

Tratamento de erros: deverá ser realizada checagem se o servidor já existe e se o usuário é dono do servidor e, então, uma mensagem de retorno apropriada deve ser adicionada.

2.7 - Entrar em um servidor

Na tentativa de entrar em um servidor, se o servidor for aberto (não tiver código de convite), o ID do usuário deve ser inserido na lista de participantes do servidor automaticamente e será necessário salvar a informação que o usuário logado está visualizando o servidor escolhido na tabela usuariosLogados.

Se o servidor não for aberto, o usuário só é adicionado como participante do servidor (bem como a informação de que ele o está visualizando) se o código de entrada fornecido for correto. Todas as vezes que um usuário tentar entrar em um servidor fechado ele precisa informar o código de convite.

Se o usuário logado criou o servidor ele pode entrar nele sem código de convite, mesmo que o mesmo não seja aberto.

- Exemplo de entrada/saída:

```
enter-server 2 dotalovers
"Entrou no servidor com sucesso"
```

```
enter-server 1 concordo-members
"Servidor requer código de convite"
```

```
enter-server 2 concordo-members 123456
"Entrou no servidor com sucesso"
```

Opções para quando se está dentro de um servidor ainda nas funcionalidades da parte 1:

- A2.8 - Sair do servidor : comando `leave-server <id-de-usuario-logado> <nome-servidor>`
- A2.9 - Listar pessoas no servidor : comando `list-participants <id-de-usuario-logado>`

A gestão de canais será implementada nas funcionalidades da parte 2.

A2.8 - Sair do servidor

Deve desconectar de um servidor dado como parâmetro, desta forma o usuário é removido da lista de participantes daquele servidor. Se o usuário saiu do servidor que ele estava visualizando no momento, este comando deve atualizar a tabela usuariosConectados para refletir que nenhum servidor está sendo visualizado (campos Servidor e CanalTexto recebem strings vazias).

- Exemplo de entrada/saída (caso ele esteja visualizando algum servidor):

```
leave-server 0 minha-casa
“Saindo do servidor ‘minha-casa’”
```

- Exemplo de entrada/saída (caso ele não seja membro de nenhum servidor, isto é o usuário já saiu de todos os servidores que participava ou não fez enter-server em qualquer servidor):

```
leave-server 0 minha-casa
“Você não está em qualquer servidor”
```

Tratamento de erros: deverá ser realizada checagem se o usuário está conectado no servidor em que este comando é executado. Caso o usuário não esteja conectado naquele servidor uma mensagem de erro apropriada deve ser mostrada.

A2.9 - Listar pessoas no servidor

Exibe todos os usuários que estão no servidor que o usuário está visualizando atualmente (somente o nome de cada).

Dica: A classe Servidor deve conter uma lista dos ids dos usuários que estão no servidor. Essa lista é de inteiros, mas você pode utilizar esses inteiros para obter o usuário realizando uma busca de usuários por ID na lista de usuários geral do sistema.

- Exemplo de entrada/saída:

```
list-participants 0
tomé
bebé
eu
eu-mesmo
irene
```

Funcionalidades (parte 2)

Com as funcionalidades da parte 1 prontas, seu sistema deve ter implementado as operações em usuários e servidores. Nesta parte, iremos implementar os recursos de canais e mensagens.

B1 - Gestão de canais (se tiver entrado no servidor)

- B1.1 - Listar canais do servidor : comando `list-channels <id-de-usuario-logado>`
- B1.2 - Criar um canal no servidor : comando `create-channel <id-de-usuario-logado> <nome>`
- B1.3 - Entrar em um canal do servidor: comando `enter-channel <id-de-usuario-logado> <nome>`
- B1.4 - Sair do canal : comando `leave-channel <id-de-usuario-logado>`

De forma similar à parte 1 todas as funções desta parte recebem um `<id-de-usuario-logado>` que deve ser levado em consideração na hora da execução dos comandos. Assim, quando o comando enviado é `list-channels 0`, o sistema deve exibir os canais que estão disponíveis no servidor que o usuário de id 0 está visualizando de acordo com a tabela usuariosLogados.

> **Tratamento de erros:** De forma similar aos comandos da parte A, os comandos dessa parte também devem verificar se o usuário está logado.

B1.1 - Listar canais do servidor

Exibe todos os canais do servidor que está sendo visualizado pelo cliente com id fornecido, mostrando os nomes de todos os canais.

- Exemplo de entrada/saída:

```
list-channels 0
#canais de texto
casa-de-mae-joana
aqui-nós-faz-o-trabalho
```

B1.2 - Criar um canal do servidor

Permite criar um canal no servidor que está sendo visualizado pelo usuário informando seu nome .

Observe que canais dentro do servidor não podem ter o mesmo nome.

- Exemplo de entrada/saída:

```
create-channel 0 casa-de-mae-joana
“Canal de texto ‘casa-de-mae-joana’ criado”
```

```
create-channel 0 casa-de-mae-joana
“Canal de texto ‘casa-de-mae-joana’ já existe!”
```

> **Tratamento de erros:** deverá ser realizada checagem se o canal já existe e uma mensagem de retorno apropriada deve ser adicionada.

B1.3 - Entrar em um canal

Entra em um canal presente na lista de canais do servidor que o usuário está visualizando. Quando um usuário entra em um canal o sistema deve atualizar no atributo usuariosLogados a informação de que o usuário está visualizando o canal em que acabou de entrar.

- Exemplo de entrada/saída:

```
enter-channel 0 casa-de-mae-joana
"Entrou no canal 'casa-de-mae-joana'"
```

```
enter-channel 0 introspecção
"Canal 'introspecção' não existe"
```

Tratamento de erros: deverá ser realizada checagem se o canal existe, caso contrário uma mensagem de retorno apropriada deve ser adicionada.

B1.4 - Sair de um canal

Sai do canal que o usuário está visualizando no momento. Este comando deve atualizar o atributo usuariosLogados para refletir que ele não está mais visualizando canal algum.

- Exemplo de entrada/saída:

```
leave-channel 0
"Saindo do canal"
```

B2 - Gestão de mensagens (se tiver entrado no servidor e em algum canal)

Ao entrar em um canal o usuário pode enviar mensagens e visualizar as mensagens anteriores que tenham sido enviadas naquele canal. Assim os seguinte comandos são possíveis:

- B2.1 - Enviar mensagem para o canal : comando `send-message <id-de-usuario-logado> <mensagem>`
- B2.2 - Visualizar mensagens do canal : comando `list-messages <id-de-usuario-logado>`

B2.1 - Enviar mensagem para o canal

Cria uma mensagem e adiciona na lista de mensagens do canal visualizado pelo usuário passado. A mensagem deve ser criada com o conteúdo digitado, a data/hora atual e com o atributo enviadaPor com o ID do usuário recebido no comando.

- Exemplo de entrada/saída:

```
send-message 0 Oi pessoal querem TC?
```

B2.2 - Visualizar mensagens do canal:

Ao visualizar as mensagens em um canal o usuário deve ser capaz de visualizar os seguintes campos da mensagem: (1) O nome do usuário que criou a mensagem; (2) Data/hora da criação da mensagem; e (3) O conteúdo da mensagem.

- Exemplo de entrada/saída:

```
list-messages 0
Julio<08/03/2021 - 11:53>: Assim não tem condições, como que a
galera vai conseguir terminar isso tudo em 4 semanas?
Isaac<08/03/2021 - 12:00>: Eles conseguem confio na galera
Renan<08/03/2021 - 12:00>: Semestre passado fizemos assim e
ninguém entregou :/
```

```
list-messages 0
“Sem mensagens para exibir”
```

Autoria e política de colaboração

A cooperação entre estudantes da turma é estimulada, sendo aceitável a discussão de ideias e estratégias. Contudo, tal interação não deve ser entendida como permissão para utilização de (parte de) código fonte de outros colegas, o que pode caracterizar situação de plágio. Trabalhos copiados em todo ou em parte de outros colegas ou da Internet serão rejeitados.

Entrega

O código deverá vir acompanhado de arquivos de teste que servirão para validar o seu programa. Utilize a dica dada no início do trabalho para criar arquivos com comandos que são usados para testar tanto os casos de erro quanto os casos de execução bem sucedida dos comandos.

A entrega do trabalho deverá incluir um arquivo **README** no formato Markdown, contendo as seguintes informações:

- Como compilar o projeto;
- Como executar o projeto;
- Como executar o conjunto dos testes planejados por você (ou grupo).
- Limitações ou funcionalidades não implementadas no programa.

Boas Práticas de Programação

No desenvolvimento do projeto, recomendamos preferencialmente o uso das

seguintes ferramentas:

- **Doxygen**: ferramenta de geração automática de documentação de código;
- **Valgrind e/ou *address sanitizer***: ferramentas de identificação de acesso inválido à memória ou *memory leaks*;
- **GDB:** ferramenta de *debugging*;
- **Makefile ou Cmake**: ferramentas de automação do processo de compilação.

Tente organizar seu código em diferentes diretórios, por exemplo </*src*> para arquivos .cpp, </*include*> para arquivos .hpp e .h, </*bin*> *ou* </*build*> para os .o ou executáveis e </*data*> para arquivos de entrada do programa.