Universidade Federal do Rio Grande do Norte
Departamento de Informática e Matemática Aplicada

Estruturas de Dados Básicas I • DIM0119
◁ Projeto de Programação ▷

Programa **ELIS** *(***E***ditor de textos orientado a* **LI***nha***S***)*

November 21, 2018

Apresentação

O objetivo deste projeto de programação é utilizar as estruturas de dados do tipo *lista encadeada* para a resolução de problemas práticos. Um dos pré-requisitos deste exercício é que estas estruturas de dados já estejam implementadas na forma de classes genéricas em C++. A aplicação do TAD *lista encadeada* será na construção de um editor de texto orientado a linhas.

Contents

1 Introdução

Um dos primeiros tipos de programas para editar textos eram orientados a linhas, ou seja, editava-se uma linha por vez (consultar verbete Wikipédia sobre Ed). Tais editores, denominados de *editores orientado a linhas*, possuíam uma interface com usuário bem restrita mas, por outro lado, eram bem versáteis e adequados aos terminais textuais típicos da época do surgimento do computador.

Posteriormente, estes editores evoluíram para os chamados *editores de página cheia*. Mesmo assim, a influência dos primeiros editores de linhas foi tamanha que ainda hoje um dos editores preferidos pela comunidade de programadores, o `vi` ou `vim`, ainda utiliza uma sintaxe de comandos bem similar aos antigos editores de linhas.

2 Tarefa

Sua tarefa consiste em desenvolver um editor de linhas, denominado `ELIS` (Editor de LInha Simples), que armazena o texto em uma lista encadeada, cada linha em um nó separado de uma lista encadeada de cadeia de caracteres (*strings*).

O programa `ELIS` opera em três modos exclusivos e distintos: *normal*, *edição* ou *comando*. No modo de *edição* deve ser exibido o *prompt* '∼ INSERT ∼', para indicar qual é o estado atual. Neste modo é possível entrar com texto para compor uma linha. Pressionando-se `<ENTER>` a linha é finalizada e armazenada na lista encadeada; a seguir uma nova linha é automaticamente disponibilizada para edição.

A Figura 1 demonstra a utilização do programa para a criação de um arquivo fornecido como parâmetro (no caso, "readme.txt"). Neste exemplo o programa está atuando apenas no modo *edição*.

```
$ ./elis readme.txt
1  Esta é a primeira linha↩
2  enquanto que esta é a segunda linha↩
3  mais uma linha↩
4  ↩
5> última linha editada↩
~
~
~
 ~ NORMAL ~
```

Figure 1: Exemplo de uso do `ELIS` no modo *edição*. O símbolo '↩' representa o pressionamento do `<ENTER>`.

Para sair do modo *edição* e entrar no modo *normal* o usuário deve pressionar `<ESC>`, fazendo com que o programa exiba o *prompt* de comando '∼ NORMAL ∼'.

Uma vez no modo *normal* o usuário pode utilizar as teclas `<j>`/`<J>` e `<k>`/`<K>` para

descer ou subir de linha, respectivamente[1]. Outro comando que pode ser invocado no modo normal é o **Undo** com a tecla <u>/<U>. Este comando desfaz os últimos comandos realizados desde a última gravação do arquivo. Confira na Tabela 1 um resumo com a lista de comandos que podem ser executados enquanto no modo *normal*.

Comando	Descrição
K	Navega para a linha de cima, se existir. A linha atual acompanha a movimentação.
J	Navega para a linha de baixo, se existir. A linha atual acompanha a movimentação.
U	Desfaz os últimos comandos realizados desde a última gravação do arquivo.
I	Ativa o modo *edição* (‘∼ INSERT ∼‘).
:	Ativa o modo *comando* (‘∼ COMMAND ∼‘).

Table 1: Lista de teclas associadas aos comandos disponíveis no modo *normal*.

Para sair do modo *normal* e entrar no modo *comando* o usuário deve pressionar <:>, mostrando o *prompt* de comando ‘∼ COMMAND ∼‘. No modo *comando* é possível invocar qualquer um dos comandos descritos na Tabela 2.

No caso do comando **Copy**, a(s) linha(s) são copiadas para um *buffer* temporário e deverão ficar disponíveis para **Paste** até que seu conteúdo seja sobrescrito por outro comando **Copy** ou o arquivo em edição seja fechado.

A Figura 2 apresenta um exemplo de interação com o programa ELIS, no qual algumas operações foram executadas sobre o texto criado na Figura 1. Os comandos podem ser fornecidos através de letras maiúsculas ou minúsculas.

Note que a numeração que aparece na primeira coluna serve apenas para identificar a ordem da linha no texto e, desta forma, não faz parte do texto em si. Portanto, esta numeração **não** deve aparecer no arquivo ascii gravado pelo programa ELIS.

De acordo com a descrição dos comandos apresentados na Tabela 2, um importante conceito é o de **linha atual**. A linha atual consiste em uma linha “selecionada” sobre o qual as operação serão realizadas, caso nenhuma outra linha seja indicada. O programa ELIS sempre possui uma linha atual ativada, normalmente a ultima linha editada. O programa deve sinalizar para o usuário qual a linha atual através da indicação do símbolo ‘>’ logo após o número da linha, como em “7>” no final da Figura 2.

3 Desafios de Implementação

Um dos elementos de desafio deste trabalho é escolher a estrutura de dados mais apropriada. Para melhor identificar as estruturas necessárias, tente antecipar todas as operações que serão necessárias. A partir desta informação, decida qual estrutura de dados utilizar. Leve em

[1]Se desejar, adicione suporte as teclas de setas pra cima e pra baixo.

Comando	Descrição
`W [<name>]`	**Write**. Salva todas as linhas do texto em um arquivo ascii `name`. O comando sem o fornecimento de um nome simplesmente grava o texto no arquivo atual. Se o nome do arquivo atual ainda não foi fornecido o programa deve solicitar um nome ao usuário.
`O <name>`	**Open**. Lê para a memória todas as linhas de texto do arquivo ascii `name`. Se o arquivo indicado não existir um novo arquivo vazio `name` deve ser criado.
`I [n]`	**Insert**. Entra no modo de *edição*, permitindo a inserção de texto *antes* da linha `n`. Se `n` não é fornecido, o texto é inserido *antes* da linha atual.
`A [n]`	**Append**. Entra no modo de *edição*, permitindo a inserção de texto *depois* da linha `n`. Se `n` não é fornecido, o texto é inserido *depois* da linha atual.
`M [n]`	**Modify**. Torna `n` a linha atual. Se `n` não é fornecido então a última linha do texto passa a ser a atual.
`E [n]`	**Edit**. Edita a linha `n`, posicionando o cursor no final da linha. A edição será básica, suportando apenas o uso do `<backspace>` para apagar os caracteres da linha. Isso quer dizer que não é necessário implementar o *scroll* do cursor ao longo da linha (ponto extra). Se `n` não é fornecido então a linha atual será editada.
`D [n [m]]`	**Delete**. Remove linhas `n` até `m`. Se apenas `n` é fornecido, remove-se a linha `n`. Se nenhum número é fornecido, remove-se a linha atual.
`C [n [m]]`	**Copy**. Copia as linhas `n` até `m` para um *buffer*. Se apenas `n` é fornecido, copia-se a linha `n`. Se nenhum número é fornecido, copia-se a linha atual.
`P [n]`	**Paste**. Cola (inserindo) as linhas atualmente armazenada no *buffer depois* da linha `n`, permanecendo no modo de *comando*. Se nenhum número é fornecido, as linhas serão coladas logo após a linha atual.
`H`	**Help**. Exibe um texto de ajuda, explicando de forma resumida quais são os comandos do programa.
`Q`	**Quit**. Encerra o programa. Se o texto atual não tiver sido salvo, o programa deve exibir uma mensagem indicando o fato e confirmar a operação.
`<ESC>`	Volta para o modo *normal* ('∼ NORMAL ∼').

Table 2: Lista de comandos do programa `ELIS`. O símbolo '[ ]' indica os argumentos opcionais.

consideração fatores como a eficiência geral do programa (complexidade temporal) e o consumo de memória (complexidade espacial).

O próximo desafio consiste em desenvolver sua própria rotina de leitura de caracteres para formar uma cadeia. Isto deve ser feito de forma a permitir a captura de teclas especiais, como `<ENTER>` e `<ESC>`. Para facilitar a captura e exibição dos texto em um terminal, será oferecido uma API construída em cima da biblioteca `ncurses` (para saber mais sobre a biblioteca ncurses) do `C++`.

Para facilitar o desenvolvimento do projeto, é muito importante compreender o *diagrama de estados* que o `ELIS` pode assumir em execução. Veja, por exemplo, este diagrama de estados do `vim`. Portanto, recomenda-se a elaboração de um diagrama de estados para o `ELIS` antes de iniciar a implementação do projeto.

```
1  Esta é a primeira linha
2  enquanto que esta é a segunda linha
3  mais uma linha
4
5>  última linha editada↵↦
~
~
~
~
 ~ COMMAND ~ : I 3
```

(a) Acionando o modo *comando* para realizar uma inserção antes da linha 3.

```
1  Esta é a primeira linha
2  enquanto que esta é a segunda linha
3  nova linha inserida↩
4> mais uma linha inserida com o comando 'I'!↵↦
5  mais uma linha
6
7  última linha editada↵↦
~
~
 ~ COMMAND ~ : d 6
```

(b) Apagando a linha 6 apenas.

```
1  Esta é a primeira linha
2  enquanto que esta é a segunda linha
3  nova linha inserida
4>  mais uma linha inserida com o comando 'I'!↵↦
5  mais uma linha
6  última linha editada
~
~
~
 ~ COMMAND ~ : A 6
```

(c) Adicionando uma nova linha após a linha 6.

```
1  Esta é a primeira linha
2  enquanto que esta é a segunda linha
3  nova linha inserida
4  mais uma linha inserida com o comando 'I'!
5  mais uma linha
6  última linha editada
7  inserindo linha depois↩
8> ↵↦
~
 ~ COMMAND ~ : W
```

(d) Gravando o arquivo editado.

Figure 2: Exemplo de uso do `ELIS` no modo *comando*. O símbolo '↩' representa o pressionamento do `<ENTER>` e o símbolo '↵↦' representa o pressionamento do `<ESC>`.

Outro desafio consiste em prever e tratar de forma apropriada o maior número possível de erros de interação usuário-programa. Por exemplo, o que acontece se o usuário pressionar "`:A 10`" em um arquivo que contém apenas 2 linhas? Ou então se o usuário fornecer o comando "`:D 10 5`", devemos indicar um erro ou deduzir que o programa vai apagar da linha 5 até a 10? Pensar em uma boa interface e um tratamento de erros robusto é fundamental para o desenvolvimento de *software* de qualidade.

Pontos extras estão disponíveis apenas para os trabalhos **completos**. Isso quer dizer que os projetos que implementaram todas as funcionalidades descritas neste documento podem ganhar mais pontos se ampliarem a funcionalidade do `ELIS`. Isto pode ser feito de várias maneiras, como por exemplo acrescentando comandos para procurar (*find* ou `F`) palavras ou fragmentos de palavras, procurar e substituir, realizar *scroll* em uma linha para permitir sua edição, etc.

4 Avaliação do Programa

Para a implementação deste projeto **é recomendado** a utilização das classes correspondente a estruturas de dados que foram apresentadas em sala de aula. Desta forma será possível verificar o desempenho do seu trabalho prévio em um projeto real.

O programa completo deverá ser entregue sem erros de compilação, testado e totalmente documentado. O programa `ELIS` será avaliado sob os seguintes critérios:-

- Comando C/P (copy/paste) funciona corretamente (15 pts)
- Comando W (write) funciona corretamente (10 pts)
- Comando E (edit) funciona corretamente (5 pts)
- Comando O (open) funciona corretamente (10 pts)
- Comando I (insert) funciona corretamente (15 pts)
- Comando A (append) funciona corretamente (10 pts)
- Comando M (modify) funciona corretamente (5 pts)
- Comando D (delete) funciona corretamente (15 pts)
- Comando J/K (navegação) funciona corretamente (10 pts)
- Comando U (undo) funciona corretamente (20 pts)
- Troca entre os estados *normal*, *edição* e *comando* correta (5 pts)
- Funcionamento geral correto, isto é, é possível criar, editar e salvar textos (10 pts)

A nota dez é alcançada ao se conquistar $\mathbf{130}$ **pontos**. A pontuação acima não é definitiva e imutável. Ela serve apenas como um guia de como o trabalho será avaliado em linhas gerais. É possível a realização de ajustes nas pontuações indicadas visando adequar a pontuação ao nível de dificuldade dos itens solicitados.

Os itens abaixo correspondem à descontos, ou seja, pontos que podem ser retirados da pontuação total obtida com os itens anteriores:-

- Presença de erros de compilação e/ou execução (até -20%)

- Falta de documentação do programa com Doxygen (até -10%)
- Vazamento de memória identificado com o valgrind (até -10%)
- Falta ou incompletude do arquivo `README.md` (até -10%)

Você deve escrever um arquivo `README.md` (formato Markdown) com, pelo menos, informações sobre como o programa foi implementado, i.e. quais estruturas de dados foram utilizadas, explicações sobre o funcionamento de cada um de seus comandos (com exemplos), indicação dos componentes da equipe desenvolvedora (com email), instruções para compilação e instalação. Fique à vontade para incluir no arquivo `README.md` qualquer outra informação que a equipe julgar relevante.

Boas práticas de programação

Recomenda-se fortemente o uso das seguintes ferramentas:-

- Doxygen: para a documentação de código e das classes;
- Git: para o controle de versões e desenvolvimento colaborativo;
- Valgrind: para verificação de vazamento de memória;
- gdb: para depuração do código; e
- Makefile: para gerenciar o processo de compilação do projeto.

Recomenda-se também que sejam realizados testes de utilização do programa em várias situações. Procure organizar seu código em várias pastas, conforme vários exemplos apresentados em sala de aula, com pastas como `src` (arquivos `.cpp`), `include` (arquivos `.h`), `bin` (arquivos `.o` e executável) e `data` (arquivos de entrada e saída de dados).

Uma forma de validar o seu programa é inserir diretivas de compilação condicional para compilar o seu projeto ora usando suas classes (por exemplo, `Lista`), ora usando classes equivalentes do STL (`vector`, `list`, etc.). Esta estratégia permite isolar erros no programa `ELIS` de erros na implementação das classes básicas.

5 Autoria e Política de Colaboração

O trabalho pode ser realizado **individualmente** ou em **duplas**, sendo que no último caso é importante, dentro do possível, dividir as tarefas igualmente entre os componentes.

Qualquer equipe pode ser convocada para uma entrevista. O objetivo da entrevista é duplo: confirmar a autoria do trabalho e determinar a contribuição real de cada componente em relação ao trabalho. Durante a entrevista os membros da equipe devem ser capazes de explicar, com desenvoltura, qualquer trecho do trabalho, mesmo que o código tenha sido desenvolvido pelo outro membro da equipe. Portanto, é possível que, após a entrevista, ocorra redução da nota geral do trabalho ou ajustes nas notas individuais, de maneira a refletir a verdadeira contribuição de cada membro, conforme determinado na entrevista.

O trabalho em cooperação entre alunos da turma é estimulado. É aceitável a discussão de ideias e estratégias. Note, contudo, que esta interação **não** deve ser entendida como permissão para utilização de código ou parte de código de outras equipes, o que pode caracterizar a situação de plágio. Em resumo, tenha o cuidado de escrever seus próprios programas.

Trabalhos plagiados receberão nota **zero** automaticamente, independente de quem seja o verdadeiro autor dos trabalhos infratores. Fazer uso de qualquer assistência sem reconhecer os créditos apropriados é considerado **plagiarismo**. Quando submeter seu trabalho, forneça a citação e reconhecimentos necessários. Isso pode ser feito pontualmente nos comentários no início do código, ou, de maneira mais abrangente, no arquivo texto `README.md`. Além disso, no caso de receber assistência, certifique-se de que ela lhe é dada de maneira genérica, ou seja, de forma que não envolva alguém tendo que escrever código por você.

6 Entrega

Você deve submeter um único arquivo com a compactação da pasta do seu projeto. Se for o caso, forneça também o link Git para o seu projeto. O arquivo compactado deve ser enviado **apenas** através da opção Tarefas da turma Virtual do Sigaa, em data divulgada no sistema.

◄ FIM ►