# Documentação

Gustavo Pauzner Mezzovilla

3 de fevereiro de 2023

| | |
|---|---|
| I$s$ | Insere a string $s$ na posição atual do texto |
| A$s$ | Carrega o conteúdo do arquivo de texto de nome $s$ no editor |
| E$s$ | (Sobre)escreve o conteúdo do editor no arquivo de texto de nome $s$ |
| F | Move o cursor um carácter à frente |
| T | Move o cursor um carácter para trás |
| 0 | Move o cursor para o início da linha atual |
| P | Move cursor para início da próxima palavra (dentro da mesma linha) |
| Q | Move cursor para início da palavra atual |
| $ | Move o cursor para o fim da linha atual |
| :$x$ | Move o cursor para o início da linha $x$ |
| :F | Move o cursor para a última linha do arquivo |
| D | Apaga o carácter da posição atual |
| DW | Apaga a palavra em que o cursor se encontra |
| DL | Deleta a linha atual |
| V | Desempilha e insere o conteúdo do topo pilha na posição atual |
| C | Empilha o texto entre a posição marcada e a posição atual (sem modificá-lo) |
| X | Empilha o texto entre a posição marcada e a posição atual e o deleta |
| B$s$ | Busca pela próxima ocorrência do padrão $s$ no texto |
| S$s$/$r$ | Substitui toda ocorrência de $s$ por $r$ no texto a partir da posição atual |
| N | Separa linha atual na posição do cursor |
| U | Unir linha atual e a próxima |
| ! | Encerra o programa |
| J | Ir para próxima linha (manter a mesma coluna, se possível) |
| H | Ir para a linha anterior (manter a mesma coluna, se possível) |

## 1 Estruturas

O presente editor de texto considera as seguintes estruturas de dados:

- `console`: estrutura responsável por capturar a entrada do usuário, que consistirá de funções e strings.

```c
typedef struct CONSOLE
{
    char *letras;
    int tamanho;
} console;
```

- `celula`: estrutura responsável por armazenar um carácter e ponteiro para as células adjacentes.

```c
typedef struct CELULA
{
        char val;
        struct CELULA *next;
        struct CELULA *prev;
} celula;
```

- `linha`: estrutura responsável por armazenar uma linha de texto, que consiste de uma lista duplamente encadeada de células, além de contemplar informações do início e do final da linha, bem como o tamanho da mesma.

```c
typedef struct LINHA
{
        int tamanho;
        struct CELULA *head;
        struct CELULA *tail;
        struct LINHA *up;
        struct LINHA *down;
} linha;
```

- `ponto`: estrutura da variável global cursor, que consiste de um ponteiro para a célula atual, além de os inteiros especificando sua linha e sua coluna.

```c
typedef struct PONTO
{
        int linha;
        int coluna;
        struct CELULA *cel;
} ponto;
```

As funções implementadas se dividem nas seguintes categorias: As que simplesmente realizam uma ação (como mover o cursor, apagar uma linha, etc.) sem nenhum parâmetro a mais, e as que dependem de um *prompt* de texto como entrada. Portanto, é possível compor as funções que não dependem de entradas de texto, e.g., mover o cursor 3 unidades pra trás, deletar um caracter e inserir outro, como ilustrado abaixo (1).

Listing 1: Exemplo de composição de funções

```
>0|vasa
      ^
------------------------
(0,4): T3DIc
------------------------
>0|casa
   ^
```

A utilização de listas duplamente encadeadas foi considerada para a implementação do editor de texto, pois a inserção e remoção de elementos em uma lista duplamente encadeada é feita em tempo constante, enquanto que em uma lista simplesmente encadeada, a inserção e remoção de elementos é feita em tempo linear.

Utilizar estruturas diferentes para as células e para listas também foi importante visando a navegabilidade do texto, em conjunto com a performance. O único problema dessa abordagem é no quesito complexidade de código. Dessa forma, a implementação da ferramenta de busca foi mais simplória, donde optou-se por verificar desde o início do documento até o final linearmente. Para auxiliar essa abordagem, foi considerado o algoritmo de Knuth-Morris e Pratt, que consiste em um algoritmo de busca de padrões em strings, que possui complexidade de tempo linear no número de caracteres no pior caso.

Esse algoritmo foi desenvolvido para rodar em qualquer sistema operacional, contudo, para utilizar acentos gráficos e caracteres especiais, é necessário que o sistema operacional seja o Windows, de modo que algumas bibliotecas específicas foram consideradas. Contudo, a adaptação do mesmo não deve ser de difícil implementação para outros sistemas operacionais.

Alguns problemas surgiram durante a implementação desse projeto, como a tentativa de utilizar arquivos de cabeçalho para a implementação das funções, o que não foi possível devido a limitações do compilador utilizado. Priorizando simplicidade, optou-se por implementar todas as funções em um único arquivo, o que não é uma boa prática de programação, mas foi a melhor solução para o problema.

## 2 Descrição das funções

**Inserção/Remoção de texto $s$**

| Chamada | Input | Complexidade |
|---|---|---|
| `I` | $s$: String | $O(\|s\|)$ |
| `D` | | $O(1)$ |
| `DL` | | $O(1)$ |
| `DW` | | $O(\|\text{palavra}\|)$ |

A inserção de texto ocorre através da chamada no console do comando `I`$s$, onde $s$ é uma *string* digitada pelo usuário. A inserção de cada carácter ocorre em tempo constante através da função `insert_char_a_direita`, que recebe um `char`, aloca uma nova célula com seu valor atribuído, e atualiza os ponteiros dos elementos adjacentes.

Listing 2: Exemplo de Inserção de funções

```
>0|hello
       ^
------------------------
(0,5): I world
------------------------
>0|hello world
             ^
```

### Escreve/Sobrescreve/Carrega arquivo

| Chamada | Input | Complexidade |
|---|---|---|
| `E/A` | $s$: Nome de um arquivo | $O(\|\text{letras de } s\|)$ |

Realiza uma das opções acima num arquivo de texto $s$ (`txt, tex, py, c,...`) na pasta em que o arquivo `main.c` se encontra.

Caso seja a função de abrir outro documento (`A`), isso ocasionará no questionamento do sistema se há desejo em salvar o trabalho anterior, caso haja, o arquivo será salvo. Após isso, o programa apagará o conteúdo de todas as linhas e o arquivo será carregado no lugar.

### Movimentação direcional do cursor

Seja $(\texttt{lin}, \texttt{col}) \in \mathbb{N}^2$ a posição atual do cursor em coordenadas, e $n_{\texttt{cima}}, n_{\texttt{baixo}} \in \mathbb{N}$ os tamanhos das linhas de cima e de baixo respectivamente.

| Chamada | Direção | Complexidade |
|---|---|---|
| `T` | Esquerda | $O(1)$ |
| `F` | Direita | $O(1)$ |
| `J` | Baixo | $\leqslant O(\min(\texttt{col}, \lvert n_{\texttt{baixo}} - \texttt{col}\rvert))$ |
| `H` | Cima | $\leqslant O(\min(\texttt{col}, \lvert n_{\texttt{cima}} - \texttt{col}\rvert))$ |
| `0` | Início da linha | $O(1)$ |
| `$` | Final da linha | $O(1)$ |

As funções de movimentação vertical tentam preservar a coluna (se existir) em que o cursor se encontra, pois as funções engatilhadas por `J` e `H` apontam inicialmente o cursor para o início da linha desejada, e assim, navegam na linha de modo a tentar encontrar a coluna mais próxima.

### Movimentação do cursor para uma linha específica

Seja $(\texttt{lin}, \texttt{col}) \in \mathbb{N}^2$ a posição atual do cursor em coordenadas e considere $n$ sendo o número total de linhas.

| Chamada | Input | Complexidade |
|---|---|---|
| `:0` | - | $O(1)$ |
| `:`$x$ | $x \in \mathbb{N}$ | $O(\min(x, \lvert x - \texttt{lin}\rvert, \lvert x - n\rvert))$ |
| `:F` | - | $O(1)$ |

Cada documento possui sempre três referências alocadas: A linha inicial, a linha final e a linha atual. A linha atual é atualizada sempre que o cursor é movido para uma nova linha, e a linha final é atualizada sempre que uma nova linha é inserida no documento. Dessa forma, fez-se a otimização de andar a menor quantidade de passos individuais, identificando qual referência das mencionadas está mais próxima da linha $x$.