Lab3: An Editor with List

In Lab3, you will also implement an editor similar to Lab2 but with different features.

Overview

In this lab, we still provide five files:

- ed.cc: this file is the same as that in Lab2.
- Editor.h: this file is like that in Lab2, but it uses ListBuffer instead of Buffer which is used in the previous lab.
- Editor.cc: this file implements class Editor which needs to support 4 kinds of commands.
- ListBuffer.h: this file declares a class named ListBuffer which should be used to store the content from the Editor.
- ListBuffer.cc: this file should give the implementation of ListBuffer. Most importantly, you are required to use the **"list"** data structure for storing the content.

You are free to modify any file except ed.cc.

Details

The class Editor should support 4 kinds of commands.

- The first kind is an input command that contains an integer and a statement (i.e., the input string). The following are some examples. Note that the integer is located at the very beginning of the command and it represents the line number of this statement. On receiving such a command, the Editor should save the corresponding statement together with its line number. Besides, this command allows us to override some existing line.

```
    cmd> 100 let a = b + c
    cmd> 30  if T < 0 then 70
    cmd> 70  printf "hello world\n"
    cmd> 30  printf "Override an existing line\n"
```

- The second kind is a delete command that contains a single character 'd' (at the very beginning) and an integer. The following are some examples. On receiving such a command, the Editor should delete the corresponding line (the input integer is used as the line number). If a user deletes a non-existing line, the Editor just ignores it.

```
cmd> d 100
cmd> d 9999
```

- The third kind is a print command that contains a single word "list". Note that this command is case-insensitive (Hint: The class string in C++ provides helper function for you). The following are some examples. On receiving such a command, the Editor should print all the saved lines in the increasing order.

```
cmd> list
cmd> LiSt

#For the above inputs, the Eidtor will print:
30  printf "Override an existing line\n"
70  printf "hello world\n"
100 let a = b + c
```

- The last command, containing "save" and a filename, is for dumping all the saved lines (in the increasing order) into a file, which is almost the same as the 'w file' command in the previous lab. The following are some examples.

```
cmd> save code.txt
cmd> save hello
```

The class ListBuffer is used by the Editor to save the input lines. And it must utilize a linked list to store the lines.

- We have designed some interfaces for you. You can implement them in ListBuffer.cc and use them in the class Editor.

```
void writeToFile(const string &filename) const;
void showLines() const;
void deleteLine(int line_idx);
void insertLine(int line_idx, const string &text);
```

- **Hint: Recall the linked list you learned. You can design a new class named list*node to represent one input line (i.e., line number and statement), and then link the list*node in order.**

Compile & Run

```
#similar to Lab2
#first, compile
g++ -o ed ed.cc Editor.cc ListBuffer.cc
#second, run
./ed
```

Upload

提交时，请将你完成的源代码压缩成zip压缩包，并重命名为 lab3-XXX.zip ; 并且在lab3-XXX.pdf中简述你的设计。 上传到 ftp://dmkaplony:public@public.sjtu.edu.cn:/upload/c++2019/lab3/ 中。 (其中XXX为学号，如 lab3-518037910001.zip 和 lab3-518037910001.pdf) 如果需要更改，请在文件名后加版本号，最终以最高高版本号为准。如第二次提交可用lab3- 518037910001-2.zip 。