HW 2

A Command Line Reader

資料結構與程式設計
Data Structure and Programming

09.23.2015

Objectives

1. Getting familiar with pointers, char arrays, string operations, etc.

2. Analyzing the problem specifications, and defining atomic member functions to support the various requirements.

3. Being able to comprehend existing code and enhance/complete it.

Overlook

- Executable file

  `cmdReader` `[-` *`<doFile>`* `]`

  - Can take "keyboard" and "file" inputs
  - Support several special keys like: UP /DOWN/LEFT/RIGHT arrows, Home, End, PageUp/Down, delete/backspace, tab, etc.

OK OK! How should we start?

Let's start by listing the implementation issues first!!

Issue #1
Instantaneously keyboard response

- How? cin? get()? getc(?
- Uh, long story. Anyway, this part has been well taken care of. You don't need to worry about it.
  - char mygetc(istream& istr);
  - Why (istream& istr)?
    - To take istream and ifstream
  - Usage example:
    char ch = mygetc(cin);

Issue #2
How to detect special key like "UP Arrow"?

[Quiz] Write a program to acquire the ASCII code for "UP Arrow"?

- Uh, how come all the UP/DOWN/LEFT /RIGHT arrows respond the same?
- They are COMBO keys!!
  - Change to a while loop…
  - Are they the same for different machines?
- A "testAsc" program is provided to test your keyboard mapping

Issue #3 How to move the cursor left? How to delete/backspace/insert a char?

- Think: What are shown on the screen have been printed out already…
  - How can they be deleted? How can cursor be moved?
- Try: cout << char(8); // i.e. '\b'
  - Example: progress bar…
- Tip: Need an internal array that records what is being shown on the screen!!
  - And a pointer to record the cursor position, and a pointer to record the end of the string

Issues #4 How about history?

- Fact:
  history can only be created, not removed.
- Solution:
  - Use a "vector<string>" to record the history strings
  - Use an index to point to the place to add history (back), or retrieve history
- Note:
  - When using "UP/DOWN" to move to previous history string, current command line should be "temporarily" recorded

We are almost ready to put up a basic program skeleton!

Issue #5
What are the "atomic operations"?

- It is important to identify the "atomic operations" so that the code can be "modularized" and "optimized".
  - This requires coding experiences and often iterations
- What we do:
  - moveBufPtr(char* const ptr)
  - deleteChar()
  - insertChar(char ch, int repeat)
  - deleteLine()
  - moveToHistory(int index)
  - addHistory()
  - retrieveHistory()

Issue #6 How should the program be structured for different key operations?

- Use “enum” instead of “if-else-if-else-if…”
  - enum ParseChar { … };

  ➔ To define each key operation as an enumerated constant
- Main function body:

```
while (1) {
    ParseChar pch = getChar(istr);
    if (pch == INPUT_END_KEY) break;
    switch (pch) {
      case LINE_BEGIN_KEY :
      case HOME_KEY: moveBufPtr(_readBuf); break;
      case LINE_END_KEY   :
      case END_KEY: moveBufPtr(_readBufEnd); break;
      case BACK_SPACE_KEY : …
```


Issue #7
How do we grade your homework?

- What if our keyboard setting is different from yours?
- Solution:

```
#ifndef TA_KB_SETTING
// You need to modify this part
#else
// This is for TA grading only
// DO NOT change this part!!!!
```

Other notable issues

- What if my keyboard does not have backspace/Home/… key?
  - No worry! You can use any key to represent it!
  - Why?
- How to make a “beep” sound?
  - cout << char(7);
  - What if no sound?
    1. Virtual beep
    2. Never mind!!!!! (why?)
- How do I know my program is “correct”?
  - There is a reference program

Please pay attention to the homework description and rules!

- A spec is a spec
- Please write program on Linux or OS X environment
  - Compilation error will result in deduction of points
- Test, and more tests!
  - We will NOT offer more testcases than what have been included in the hw2.tgz file
  - It’s your responsibility!!!
  - We always have automatic test pattern generator