資料結構與程式設計

(Data Structure and Programming)

101 學年上學期複選必修課程 901 31900

Homework #2 (Due: 11:00pm, Oct. 23, 2012)

Department:_____ Grade:_____

Id:_____ Name:_____

0. 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.

1. Problem Description

In this homework, we are going to design a command reader "**cmdReader**". It can accept 2 kinds of inputs: standard input (keyboard) and file input (dofile). The syntax for the executable "**cmdReader**" is:

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

where the **bold words** indicate the command name or required entries, square brackets "[ ]" indicate optional arguments, and angle brackets "< >" indicate required arguments. An example of command line entry is:

cmdReader -**File** myDofile.1

Note that the option "-File" can be typed as "-file", "-f", "-F", "-fi"..., etc. Do not type the square or angle brackets.

When you specify the arguments "-File <*doFile*>" for the **cmdReader**, it will read in commands from the *doFile*, and at the end of file, go to the command prompt "`cmd>` " and end the program.

Several functional keys, such as “up/down/right/left arrows”, “back space”, “delete”, “Tab”, “Ctrl-a”, “Ctrl-e”, “Home”, “End”, “PgUp (Page Down)”, “PgDn (Page Up)”, etc should be supported. There should also be a command history array to record the previous command entries.

2. Keyboard Input Specification

1. Printable character (e.g. alphanumeric characters, arithmetic and logic symbols)

   When this kind of character is entered, insert it at the current cursor location. Move the cursor to the next position on the right afterwards. For example, inserting “k33” at the following line (is the cursor) ---

   ```
   cmd> read design abcd fyyh
   ```

   you will get ---

   ```
   cmd> read design abk33cd fyyh
   ```

2. Ctrl-a (LINE_BEGIN_KEY) or Home (HOME_KEY)

   When you press “Ctrl-a” or “Home”, the cursor will move to the beginning of the line. For example, in the following line ---

   ```
   cmd> read design abk33cd fyyh
   ```

   pressing “ctrl-a” you will get ---

   ```
   cmd> read design abk33cd fyyh
   ```

3. Ctrl-e (LINE_END_KEY) or End (END_KEY)

   When you press “Ctrl-e” or “End”, the cursor will move to the end of the line. For example, in the following line ---

   ```
   cmd> read design abk33cd fyyh
   ```

   pressing “End” you will get ---

   ```
   cmd> read design abk33cd fyyh
   ```

4. Backspace (←) (BACK_SPACE_KEY)

   When you press the backspace key (←), it will delete the character before the current cursor. The characters at and to the right of the cursor will be shifted left for one position. For example, when the line is ---

   ```
   cmd> read design abk33cd fyyh
   ```

   and if you press the backspace key (←) once, it will become ---

   ```
   cmd> read design abk3cd fyyh
   ```

   Please note that if the cursor is at the beginning of the line, pressing backspace will not delete anything. Instead, it will produce a beep sound as an error (calling static function “mybeep()” defined in “cmdParser.cpp”).

5. Delete key (DELETE_KEY)

   Delete the character at the current cursor position. Afterwards, the characters to the right of the cursor will be shifted left for one position. For example, when the line is ---

   ```
   cmd> read design abk33cd fyyh
   ```

   and if you press the delete key once, it will become ---

   ```
   cmd> read design abk33d fyyh
   ```

   Please note that if the cursor is at the end of the line, pressing delete key will not delete anything. Instead, it will produce a beep sound as an error.

6. Tab key (TAB_KEY)

   Move the cursor and the substring after the cursor right to the next tab position. Insert space character(s) in between. The tab position is defined by TAB_POSITION in "*cmdParser.h*". For example, let TAB_POSITION = 8, when the line is ---

   ```
   cmd> 1234567890123456
   ```

   and if you press the tab key once, it will become ---

   ```
   cmd> 123456  7890123456
   ```

   On the other hand, if the line is ---

   ```
   cmd> 12345
   ```

   after you press the tab key, it will becom ---

   ```
   cmd> 12345    
   ```

   That is, 3 space characters will be inserted at the end.

7. Enter ↵(NEWLINE_KEY)

   When the "enter" key is pressed, current line will be stored in the command history array (i.e. "vector<string> _history" in the class "CmdParser") and the cursor will move to a new line

   Please note that the leading and ending "spaces" in the line will be removed before the line is stored in the command history array. For example, after pressing "enter" on the line ---

   ```
   cmd>           read design abk33cd fyyh            
   ```

   you should store "`read design abk33cd fyyh`" in the command history array.

   In addition, you can press the "enter" key anywhere in the line. For example, you will get the same string stored in the command history array for the following two cases ---

   ```
   cmd> read design abk33cd fyyh
   cmd> read design abk33cd fyyh
   ```

Besides, when the line is empty, or containing only space characters, pressing "enter" key will not store anything into history array but move to a new line.

8. Up arrow ↑ (ARROW_UP_KEY) or PgUp (PG_UP_KEY)

   When the "up arrow" key is pressed, replace the current line with the previous string stored in the command history array. When more "up arrow" keys are pressed, you can go back to earlier stored strings. On the other hand, pressing the "page up (PgUp)" key you will go back for PG_OFFSET lines at once (default PG_OFFSET = 10). If there are less than PG_OFFSET strings before the current line in the command history array, go back to the first string in the command history array instead. In all of the cases, the cursor will be at the end of the line afterwards.

   When you get to the top of the command history array, pressing "up arrow" or "PgUp" key will cause a beep sound as an error and do nothing.

   Note that if the current line is NOT from the command history, pressing "up arrow" or "PgUp" key will temporarily store the current line to the end of the command history array, without removing the leading and ending spaces. You can always go back to this line later with "Down arrow" or "PgDn', even if you type in any printable key (except "enter") to modify the history command string. However, if you press "enter" key when browsing the history, the temporarily stored command line will be replaced and the history command string will be inserted to the end of the history array.

9. Down arrow ↓ (ARROW_DOWN_KEY) or PgDn (PG_DN_KEY)

   Pressing "down arrow" or "PgDn" key allows you to traverse forward to the newly stored strings in the command history array. The rules are similar to those of "up arrow" and "PgUp".

   After reaching the line of currently entered string (note: may also be an empty string), pressing "down arrow" or "PgDn" will cause a beep sound as an error.

10. Please note that what have been stored in the "`CmdParser::_history`" array won't be changed. However, if you retrieve a history string by the Up or Down arrow, and then press the "enter" key, the history string will be stored as a new history entry in the "`_history`" array. See the following examples:

```
[Example 10-1]
1                cmd> 1       _history: {}
<enter>          cmd>         _history: { "1" }          // add to history
2                cmd> 2       _history: { "1" }
<enter>          cmd>         _history: { "1", "2" }     // add to history
<enter>          cmd>         _history: { "1", "2" }   // no change, just a new line
<up arrow>       cmd> 2       _history: { "1", "2", "" }  // retrieving history
0                cmd> 20      _history: { "1", "2" }
<enter>          cmd>         _history: { "1", "2", "20" }   // replace tmp
```

```
[Example 10-2]
1                 cmd> 1       _history: {}
<enter>           cmd>         _history: { “1” }          // add to history
2                 cmd> 2       _history: { “1” }
<enter>           cmd>         _history: { “1”, “2” }     // add to history
<up arrow>        cmd> 2       _history: { “1”, “2”, “” } // retrieving history
0                 cmd> 20      _history: { “1”, “2”, “” }
<up arrow>        cmd> 1       _history: { “1”, “2”, “” } // “20” is NOT saved
<down arrow>      cmd> 2       _history: { “1”, “2”, “” } // retrieving history
<down arrow>      cmd>         _history: { “1”, “2” }     // retrieving tmp history
<down arrow>      cmd>         _history: { “1”, “2” }     // beep sound

[Example 10-3]
1                 cmd> 1       _history: {}
<enter>           cmd>         _history: { “1” }
2                 cmd> 2       _history: { “1” }
<up arrow>        cmd> 1       _history: { “1”, “2” }     // retrieving history
<down arrow>      cmd> 2       _history: { “1” }          // retrieving tmp history
0                 cmd> 20      _history: { “1” }
<up arrow>        cmd> 1       _history: { “1”, “20” }    // retrieving history
<down arrow>      cmd> 20      _history: { “1” }          // retrieving tmp history
<down arrow>      cmd> 20      _history: { “1” }          // beep sound
```

11. Right arrow → (ARROW_RIGHT_KEY)

    Move the cursor right for one character. If end of the line is reached, make a beep sound as an error.

12. Left arrow ← (ARROW_LEFT_KEY)

    Move the cursor left for one character. If beginning of the line is reached, make a beep sound as an error.

13. Any key that is not defined as above (UNDEFINED_KEY)

    Making a beep sound as an error.

14. In this homework, you don’t need to handle “line wrapping”. That is, we assume your entered string will NOT change line. We will NOT test the condition when the entered string is longer than the remaining spaces in the current line.

3. Source Codes

In the “hw2” directory, there are several files:

1. **main.cpp** : entry of the **cmdReader** program. In this homework, you don’t need to modify this file.

2. **testAsc.cpp**: a program to test the ASCII code of your keyboard mapping. See Section 4 for more details. In this homework, you don’t need to modify this file either.

3. **cmdCharDef.h/cmdCharDef.cpp**: the header file to define “`enum ParseChar`” and the source code to process the supported key presses, respectively. Please note that different shell terminals may have different keyboard mappings, so we define two sets of keyboard mapping codes in order to work with your and TAs’ terminals. They are separated by “`#ifndef TA_KB_SETTING`” and “`#else`” in both files. The former part is the one you need to fix in order to work with your terminal, and the latter part will be used by TAs to grade your program. So please do not modify the latter part.

4. **cmdParser.h** : header file to define class “CmdParser”.

5. **cmdReader.cpp**: define member functions of class “CmdParser”. In this homework, ALL the TODO’s (except for the keyboard mapping in “cmdCharDef.*”) are in this file.

6. **Makefile**: multiple objectives Makefile. Useful targets in the Makefile include:

   make or **make hw2**: to create the main program “cmdReader”.

   make test: to create the “testAsc” executable to test your keyboard setting.

   make ref: using provided “cmdReader.o.ref” to create the reference program.

   make 32 or **make 64**: to create the symbolic link of “cmdReader.o.ref” for 32 or 64-bit platform, respectively.

   make clean: to clean up all the object files and the executables “cmdReader” and “testAsc”.

   make ctags: to creates the ctags for source code tracing. To trace into a symbol, place your cursor on top of it and type “Ctrl-]”. To get back, type “Ctrl-t”.

4. What you should do?

You are encouraged to follow the steps below for this homework assignment:

1. Read the specification (in Section 2 above) carefully and make sure you understand the requirements.

2. THINK, before you look at the reference code and write!!! Imagine this: if you are asked to implement this **cmdReader** from scratch, how will do you? Try to design your own data structure and program flow. Although this is NOT part of the homework requirements, it will help you appreciate the design of the reference code, and hopefully, you will have the capability to design a software system in the future.

3. "**make test**" and you will get an executable "**testAsc**". Please note that different shell terminals may have different keyboard mappings, so you WILL need to revise the codes in "**cmdCharDef.h"** and "**cmdCharDef.cpp**" in order to see the correct keyboard operations of your terminal. Please refer to the mapping below and the code in the "#else" part of "#ifndef TA_KB_SETTING". They are the keyboard mappings of TAs' terminal and TAs will use them to grade your program, so please DO NOT modify them. Fix the codes in the "#ifndef TA_KB_SETTING" part according to the ASCII codes responded by your **testAsc** program.
   - Ctrl-a : 1
   - Ctrl-e : 5
   - Esc: 27
   - Back space : 127
   - Enter : 10
   - Tab: 9
   - Delete : 27 91 51 126
   - Home: 27 91 49 126
   - End: 27 91 52 126
   - PgUp: 27 91 53 126
   - PgDn: 27 91 54 126
   - Up arrow : 27 91 65
   - Down arrow : 27 91 66
   - Right arrow : 27 91 67
   - Left arrow : 27 91 68

After fixing the codes in "**cmdCharDef.h"** and "**cmdCharDef.cpp**", you should be able to see your keyboard mapping on the screen by "**testAsc**" in the following format:

```
Press "Ctrl-d" to quit
27    91    65    Arrow up
27    91    68    Arrow left
27    91    49    126   Home key
1     Line begin
121   y
4     Input end
```

Please note that if some of the combo keys (e.g. PgUp) do not respond, or you really have problems making "**cmdCharDef.cpp**" work, use some other key (e.g. F3) as a substitution. That should not affect your grading as TAs will use the "#else" part for compilation.

4. Type "**make ref**" to compile the reference program based on your "**cmdCharDef.h**", "**cmdCharDef.cpp**" and the provided "**cmdReader.o.ref**". You may need to type "**make 32**" or "**make 64**" in order to choose the correct platform (type command "**uname -a**". Play with the reference program and understand the details of the spec in this homework.

5. Study the provided source code and the brief summary in Section 3. You then only need to finish the places marked with "**TODO**" in the file "**cmdReader.cpp**". You don't need to add any new data member or functions to it. Please try to comprehend why we define the member functions this way. (Hint: sharing of codes)

6. Complete your coding and compile by "**make**" (or "**make hw2**"). Test your program thoroughly. There is one test script under "hw2" called "**hw2.test1**". You can test your command reader to see whether it matches the correct output response.

5. Grading

We will test your submitted program with various combinations of commands and functional keys. Therefore, please test your program carefully before submitting it.