

Homework 1

Reading: Tanenbaum, Chapter 1

Due: Jan. 31, 2007

Problem 1 (0 Points)

Read the solution to Assessment 1 and write a C/C++ program that demonstrates Problems 4, 5 and 6 on assessment 1.

Problem 2 (0 Points)

If you are new to the Unix environment, do reading suggested on the course Web page. You will find the reading by following the **Introduction to Linux** link under **Online Notes and Tutorials**.

Problem 3 (0 Points)

Write a small C++ test program that will accept plain text lines each consisting of two words and either update or show the *value* of a symbol. A word is any sequence of symbols separated by one or more white spaces (space or tab character). The symbol table is defined as a map from one symbol to second symbol which is considered the value of the symbol:

```
struct ltstr {
    bool operator()(const char *s1, const char *s2) const {
        return strcmp(s1, s2) < 0;
    }
};
...
map<const char *, char *, ltstr> value;
```

There are two kinds of lines. If the first word is "show", the second word is considered to be a symbol for which you want to display the symbol and its value. Otherwise, the first word is the name and the second is its new value.

In the following example, the meaning is shown to the right:

```
x      123y      # set the value of "x" to "123y"
show x          # should output "x 123y"
894 xxx        # set the value of "894" to "xxx"
x      abc      # set the value of "x" to "abc"
show 894       # should output "894 xxx"
```

Problem 4 (6 Points)

Write a C/C++ program that will eventually evolve into a special command-line interpreter. For now, the program will just do some simple lexical analysis and outputting. In the description below, a command-line is a sequence of characters separated by one or more consecutive white-space characters. A white-space character is either a space or a tab character. Each non-white-space character sequence is called a *word*. Here are three examples:

```
toggle v
set x 3-12
exec myprog hello 8 57
```

The commands have 2, 3 and 5 words respectively. Note that 3-12 is one word since there are no intervening white space characters.

- The program has the following synopsis:

```
hw1-4 < Commands
```

In this assignment, we are only interested in the syntax of the language and the semantics of the language is left undefined.

- Your program should do the following for each command line:
 - Display on `stdout` the command line exactly as it appears on input.
 - Place the words into a 100-byte buffer. Suppose that there are N words in the command. The buffer contains an *unsigned long* sequence number starting at 0, and then, $N + 1$ pointers immediately followed by the character strings which are packed one string after the other but with the NUL character terminating each string. The first N words are pointers to the words in the buffer. Word N (counting from 0) contains a null pointer indicating the end of the pointer array.
 - Then, display the contents of the buffer. In this format, the sequence number is displayed in human readable form in a 3-character field; pointers are displayed in each character is displayed in hexadecimal (each output field is separated by a space character. For example, the command `'set foo 32'` might be displayed as follows if the buffer begins at address `0xbffff7ec` and the sequence number is 5:

```
5 0xbffff7fc 0xbffff800 0xbffff804 (nil) 73 65 74 0 66 6f 6f 0 33 32 0
```

- Your program should exit when either the first word is `quit` or an *end-of-file* condition is encountered on `stdin`.

Here is a detailed description of the language:

- Each command is separated by a newline character (`'\n'`).
- All commands have a variable number of arguments which are separated from each other by white-space. However, you can assume that a command never has more than five words nor will overflow your 100-byte buffer.

Submit:

- Source listing
- Test output
- A short explanation of why the test output indicates that your program is operating correctly.

The following man pages might be useful: `strcmp(3)`, `memset(3)`, `isdigit(3)`, `atoi(3)`, `strtol(3)`, `strtok(3)`, `printf(3)`, `exit(3)`, `memcpy(3)`, `fgets(3)`, `strlen(3)` and `ascii(7)`. The digit in parentheses indicates the probable section where you can find the man page; i.e., `'man 3 memset'` looks for `memset` in section 3. Note that in Solaris, you will need the `-s` flag; i.e., `'man -s 3 memset'`.