**Hayden Martz Lab Manager Intern Projects Guide**

The overall goal of this project is to design a terminal-line, interactive program set which will facilitate the management and tracking of the computers running Apple OS X, for potential expansion to cover unix environments (e.g. Linux distributions).

For you, this means experience in coding in C++, the powerful terminal control library called nCurses, and bash scripting—which lets you write code that runs terminal commands. All very useful (and very cool).

The software will run locally on a machine, programmed in C++ with the g++ compiler. g++ version 5.4.0 at the time of writing, on the Linux-02 machine for initial development. This may be moved to Linux-Test-1 machine created under direction from Adam Benabbou, or to the OS X operating system as mandated by the target machines.

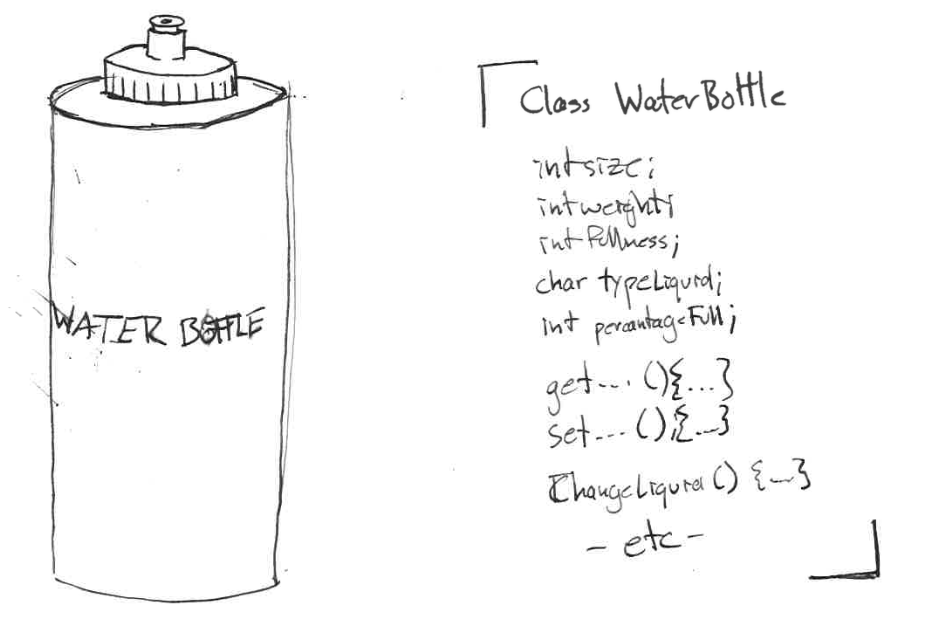
The program will have a dynamic and interactive user interface controlled using the ncurses terminal manipulation library. Menus and submenus will call functions writtenin bash, which is a common method for writing scripts that enter terminal commands. These scripts will call functions from the brew application management suite, call and interpret machine data using GREP (regular expressions), and more. It will accomplish this on remote machines by connecting via secure shell (SSH) into a target computer on the network, and running scripts from the same software package which should also be installed on the target computer.

This software should be a component of the OS X installer image, and will be automatically installed.

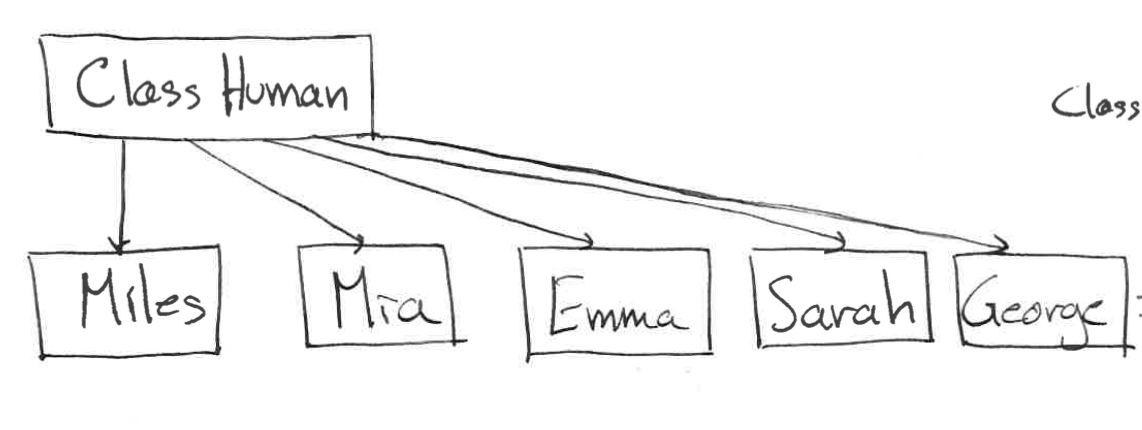
**Background**

C++ is an object oriented programming language which hovers about halfway between the machine code/assembly instructions and the very abstract higher interpreted languages. It has many convenient features common to high level languages, but also allows for the flexibility and control of the program data that is extremely useful for projects.

Object Oriented Code is a style of programming which represents a new evolution in computer software. As you probably guessed, it centers around an object. Put generally, an object is something that has data and methods. For example, take a water bottle. It has it’s size, shape, weight, how full it is, what is in it, etc.. You can act upon it by picking it up, moving it, drinking from it, or filling it up.



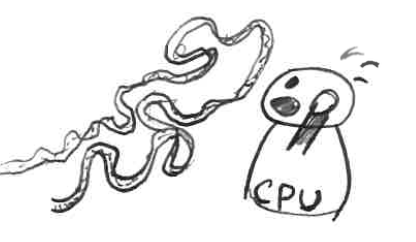
In object oriented programming, everything that you manipulate is represented as an instance of an object. You do this by writing a generic class—which stores the data as local variables of that object, and represents actions done to the object through methods that manipulate the data.

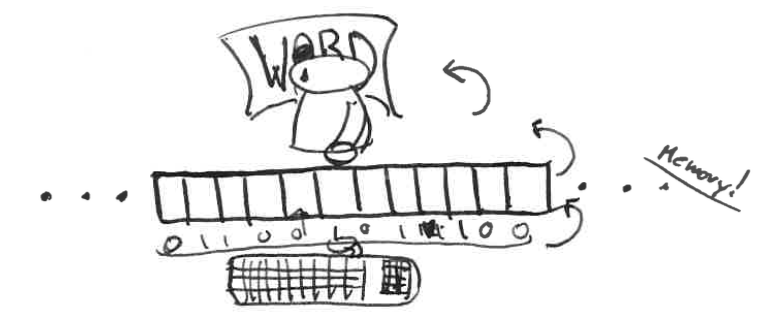


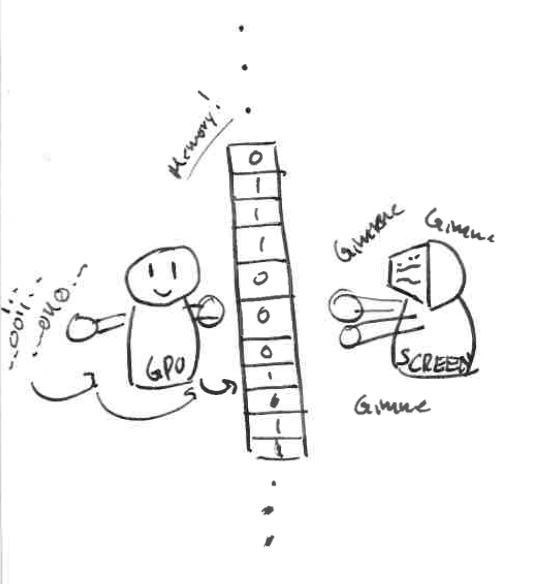
Instance

In a program, you then initialize an instance of that class as a regular variable, much like an int or char (e.g. Human Miles = Miles(<data>); and int velocity = 0; Here, Miles is declared as an instance of a Human—with the appropriated data assigned by the constructor). A program calls these methods—akin to a function—which manipulates the variables. Both the variables and methods are local to function instance, and have unique memory locations distinct from the main program memory.

nCurses is a software library for C++ which dynamically controls the user input and sets the terminal output. Some background is useful for understanding what makes this different from the manipulation you are used to.

Modern computers are essentially very fast data shuffling machines. Based on a set of bits (which represent the program instruction), they will perform operations on memory locations. To handle I/O, a portion of memory is reserved by the operating system. When a key is pressed on a keyboard or the mouse is moved, the signal is converted into a bit pattern and stores itself into that memory location. The operating system can then go look at that value (at its leisure), read the values into CPU cache, and then assign it into whatever software you are using (which currently, is probably Microsoft Word).



Displays work similarly, but in reverse. Here, the OS sets aside a bunch of memory locations whose value is set by whatever software you are running. There are n-bits set aside for each pixel on the screen (what a coincidence), which is called the color depth. Most monitors you use are probably 32 bits. There is a regularly occurring process which hijacks the CPU (or GPU), reads those bits into the cache, and then sets the appropriate pixel LEDs/LCD/whatever to those bit values (coincidence? I think NOT).

Now, the terminal manipulation you are used to is very complex. The terminal has processes running to automatically clear and refresh the screen for you. It gets each key press from the keyboard memory location and moves it into a special buffer for you, for each key press. It detects an enter key and shovels the data into a nice package (variable) for you. This is nice for most purposes, but it also hides a bit of power.

nCurses strips this away. Instead of having a buffer that automatically populates with the keyboard data, you have to request each key press (and put it in a memory location) yourself. Otherwise they are ignored (very useful for sanitizing input. Instead of reading it in and disposing of what you don’t want, you only hear what you want to hear).

Output is where it really shines. Instead of having to fiddle with arrays to refresh the screen all at once (which you had to do in FORTRAN), nCurses does that for you.

Essentially, you can imagine there are two screens. One that the user sees—which is static—and one that you have a cursor on and you can manipulate.

In one command (mvprintw(row, column, char\*);) the cursor on the hidden screen is moved to the specified location and the character is placed on it. The refresh command takes all those characters from the hidden screen, and puts them in the same place on the user screen. There is also the clear function, which wipes the visible screen.

Hopefully this brief stream of consciousness will help you understand a bit. What will really help is writing some code—and we will start with a basic menu.

**Project 1: C++, nCurses, and Interactive Menus.**

**Task 1:**

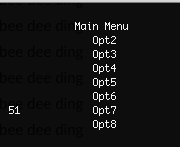
In C++ and nCurses, write a basic menu.

It should have a title (the ultimate goal is remote management). Make 5-7 menu options appear below it. When I hit a number/character associated with each option, have something pop up on screen somewhere indicating that’s what I chose (or, if you are feeling adventurous, have it wipe the screen, display “option \_\_\_\_\_ selected,” wait for input, then go back to the main menu.

As to what to say for each option…You’re smart. Be creative. If it’s good it probably won’t even get changed.

Let me know if you need help. Below is an example I wrote in under an hour. It may be frustrating, nCurses can be fiendishly tricky—but I am certain you can both do it.

Example:

 1 #include<ncurses.h>

2 #include<stdio.h>

3 #include<stdlib.h>

4

5 int main(){

6 initscr();

7 cbreak();

8 noecho();

9 curs\_set(0);

10

11 int input;

12 while(true){

13 //print stuff

14 mvprintw(1, 12, "Main Menu");

15 for(int i=2; i<9; i++){

16 mvprintw(i, 15, "Opt");

17 mvprintw(i, 18, "%d", i);

18 }

19

20 //get input

21 input = getch();

22 mvprintw(7, 1, "%d", input);

23

24 //reset screen

25 refresh();

26 }

27

28 endwin();

29 }

Good Website: <http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/>

**Task 2:**

Read menu options from a text file and display them as a menu.

Create a text file with your menu options. Read the strings from the text file into an array. Use this array to print out the menu options onto the screen.

Next: have an index (eg. Int selected). Use this to simulate which option is selected. Highlight the selected option. When down or up are pressed, increment or decrement the value. (Warning: be aware of boundary conditions).Task 3:

Create a code that will get information about a text field, then create an interactive text field using ncurses. The left/right arrow keys should move the cursor (which is visible). You can let it behave such that when deleting, no text moves, and when typing over something the text is just overwrite, but it would be nice if you accounted for these.

Inputs: x-position, y-position, field length (num characters)

Hint: use a char array model the field, and use an integer to keep track of the cursor position.

Bonus: Get the mouse to work (Don’t do this).

**Task 4:**

Take the code from the previous page, and turn it into an object (class).

Preferably, set them up as distinct files. You main want to write a variation on crun for this.

Get Set for: x-position, y-position, length

Methods to: Clear, enter a character, enter a character array, write to a file (that is passed in as a parameter)

**Task 5:**

Take task 4 and create a text editor with it. Only the size of the screen—so it’s more like a textbox. Don’t worry about making it as fancy as VIM—I just want to be able to put text anywhere on the screen, even if it is a bit tedious.

\*\*There may be limited benefit to continuing. Do so at your own discretion.\*\*

Bonus: implement loading from file and writing to file.

Super Bonus: implement scrolling, so the files can be larger than the terminal size.

C++ Guide

int main(){} //The main program method. Returns int (for error codes).

int <name>; //Declaring an integer of name <name>. I recommend you don’t //initialize at the same time. But you *can* do it anywhere now.

Object <o\_name> = new Object() //Instantiating an object.

<name> = <value>; //variable initialization: assigning a value to your variable.

while(<test>){…} //While loop, repeats until test is false AT BEGINNING, runs //through and then exits at end.

Do{…}while(<test>); //Do-While loop, executes until test is false AT END, then exits.

For(int i=<start>, <end test>, <increment value>){} //For loop. Starting at <start>, loops //until <end test> is false by the <increment value>.  
//e.g. for(int i=1; i<=10; i++){}

#include <fstream> //Allows you to work with file streams (open and read files)

I also recommend you write a command to automatically compile, change the name of the executable, and run the executable. You can try doing this on your own, or use mine. The file is called “crun” and to use it you type “crun programName” for the file “programName.cpp.

1 #!/bin/bash

2 set -e

3 g++ $1.cpp -lncurses

4 mv a.out $1.run

5 ./$1.run

nCurses Guide

#include <ncurses.h> //At very top, before main(). Allows you to use the library and //commands.

RUN THESE 4 AT BEGINNING TO INITIALIZE SCREEN

initscr(); //Initializes the screen.

cbreak(); //Disables terminal line buffering

noecho(); //Don’t let getch echo characters.

curs\_set(0); //Set cursor visibility. 0=Invisible, 1=Terminal normal,   
//2 = Terminal High Vis

IN CODE

clear(); //Empties visible screen

refresh(); //Updates changes to screen.

mvprintw(i, j, “something”); //Moes to row i, column j, and prints something starting //there.

mvchgat(i, j, l, F1 | F2, 0, NULL); //Moves to row i, column j, and formats l characters //starting there with formatting F1, F2, etc..

<int var> = getch(); //Gets the character entered by the user. Only one character. Stores //as the ASCII value.

endwin(); //Ends the screen session. Put at end of program. DO NOT //FORGET.