CISC 3110

Project 2: Snake (part 1)

In this project, you'll get a bit of experience working with non-standard libraries, and with the difference between object-oriented design and "pre-object-oriented" design.

The Game

You are probably familiar with the old-school game called snake. If not, you can play it online at playsnake.org (the control keys are the arrow keys). You can also play it on your Linux/vagrant machine. To do that, run the command

```
$ sudo apt install bsdgames
```

(If you're asked for a password, use `vagrant`.)

Then run the command

```
$ worm
```

You can always say

```
$ man worm
```

to learn more about the game.

But for this project, we're going to adapt someone's implementation of the game.

Getting Started

Like most applications that use the Linux terminal screen in interesting ways, the code we'll be using is based on the `ncurses` library. (Don't worry; I'm not going to ask you to write code that uses this directly, but you need to understand essentially what's going on.) To be able to use `ncurses` in our code, we have to install the library:

```
$ sudo apt-get install ncurses-dev
```

(Again, the password is `vagrant`.)

Now, about the code. There's a (mostly) functioning implementation of the game on stackoverflow. Look at the code and the comments. I've made a *very* slightly modified version of the code available at https://bc-cisc3110-f16.github.io/snake_c.cpp. (Essentially, my modifications take C code and make it enough like C++ that g++ will compile it.) Download this code and compile and run it in vagrant. To compile, you need to tell the linker to use the `ncurses` library:

```
$ g++ snake_c.cpp -lncurses
```

Now run it—it works, more or less. See if you can relate the comments about bugs on stackoverflow with the behavior of the running code.

What's Next

Next week, I will give you a slightly revised version of this code—mostly, improving on some of the ncurses code. Then I will ask you to "translate" this code into object-oriented C++ code. Instead of structs and functions, we'll have full classes (`Snake`, `Position`, `Food`, maybe others). And I'll ask you to fix some of the other problems and add a little bit of behavior.

Project 2 (Part 2)

As promised, I've posted some slightly modified snake code at https://bc-cisc3110-f16.github.io/project2.zip.

The main change I've made is to create a `Console` class which contains most of the ncurses-related code. (I've also made it use ncurses just a little bit better.)

So, challenges remaining for you: create `Snake`, `Position`, and `Food` classes. Instead of `init()` methods, write constructors. (Do any of them need destructors?) Will any of these classes have instance variables that are objects of other classes?

Your resulting code should have no global variables and no globally-declared constant values. Also, fix as many bugs and other problems as you can (again, look through the comments on stackoverflow). In particular, make sure a `Position` object actually represents a single location on the 'screen'.

Additionally, your Snake should use dynamic memory allocation. It starts off with a small length; as it eats more food, it becomes longer. You will need to store the locations of each 'segment' of the Snake's body. An array is a good tool, but how long should the array be? One technique: as the size you need increases, allocate a new and bigger array, copy the old contents over, and delete the old array. It's probably too much work to do this every time the Snake grows, so implement a policy: maybe add 10 each time you need to grow, or perhaps double the size each time.

For extra credit, add some color to the game, or make other enhancements to the user interface. For this, you may find the ncurses manual helpful. (You do *not* need the ncurses manual to do the main work of the project—this is only if you're curious and want to learn/do more.)

Grading will be out of 20 points, as follows:

4 Correctness/Efficiency
6 Object-oriented design
5 Style
5 Documentation/comments

Turn in hardcopies of your makefile and complete code, and email me the executable file.

The project is due at the BEGINNING of class on Class 26. (If you're late, you'll miss the RAT.)