Intro to C Programming

Summer 2011

Lab 2: Game of Life Redux

Overview

In this lab, we will extend Lab 1, by implementing Conway's Game of Life, with fewer restrictions. In the first version, the code used static arrays and array indexing. This version will use dynamic memory allocation with `malloc()` and pointer arithmetic `*(ptr+i)` instead of `ptr[i]`.

Part A: Dynamic Allocation

This will use the same code that was created for Lab 1. However, instead of a fixed world size as in lab 1, the state file will now have one new line at the top:

```
50 50
              *
             ***
               *
```

The first line lists the X and Y dimensions of the world to be used. We will still assume that anything outside the world boundary is dead. You should define a `struct` to hold the `max_X` and `max_Y` values and the current and next state pointers. This could look something like:

```
struct world {
  int x;
  int y;
  char *current;
  char *next;
};

...

struct world *world_ptr;

world_ptr->x = /* max_x */;
world_ptr->y = /* max_y */;
...
```

The code will need to be updated to pass the world struct as a parameter to several of the functions so that the bounds can always be tested.

To dynamically allocate the required space, you will need to use malloc. You can either malloc twice as much space in a single piece of memory or malloc two different pieces of memory. This will look something like:

```
char *ptr;

ptr = (char *)malloc(x*y*sizeof(char)*2); /* twice the size */

world_ptr->current = ptr;
world_ptr->next = /* what location ? */
```

Why wouldn't you malloc and free memory every iteration? What advantages does the world struct have?

Part B: Pointer arithmetic

Normally, you can use buffers using array indexing (e.g., `ptr[i]`), for this second second part of the lab, you will need to update the code to use pointer arithmetic. Basically, you must calculate the x and y indices into the 1 dimensional buffer. All computer memory is 1-D and either you or the compiler add any extra structure to memory. One approach for indexing might be:

```
*((world_ptr->next)+x+(y*width)) = next_state(x,y,world_ptr);
```

Most of the pointer arithmetic will be straight forward since we are using `char` and chars are a single byte wide. How would this be different if it was an array of structs, like the world struct?

To finish, write a brief (1 page max.) lab report describing your experience completing this lab, what challenges you faced and how you addressed them, and what you learned from this lab. Turn in a zip file containing all your code, and your lab report, to me via email by the due date.