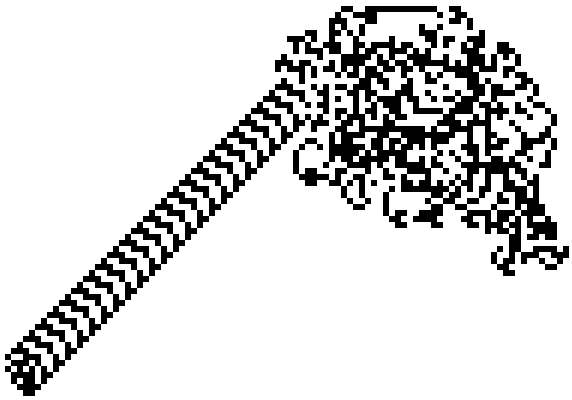
COM S 327, Fall 2016
Programming Project 0



Langton's Ant

Langton's Ant is one of a class of mathematical games known as *cellular automata*. Perhaps the best known of these is Conway's *Game of Life*, in which the cells represent living organisms which breed and die according to specific rules, and which has been studied for decades and has had countless scholarly articles written about it (and which has had a computer implemented within it!).

Cellular automata live in arbitrary mathematical spaces. The standard rules for Langton's Ant (and for Life) have it played in $\mathbb{Z}^2$ (two-dimensional, integer Euclidean space), but simple extensions of the rules can move the game into other spaces, like $\mathbb{Z}^3$. The automata interact with their space, making changes to it, and their behavior is determined by the state of the space.

Langton's Ant is played in a space where every cell has two colors, white and black. The ant has a position (x and y) and a direction (north, south, east, or west). It moves through space according to the following two rules:

1. When making a move from an initially white cell, turn 90° clockwise, toggle the cell color, and move forward one cell.

2. When making a move from an initially black cell, turn 90° counter-clockwise, toggle the cell color, and move forward one cell.

The order of operations in the rules matter! You can try changing them and observe what happens.

See this Wikipedia article for more about Langton's Ant:

https://en.wikipedia.org/wiki/Langton%27s_ant.

The supplied C program contains 4 functions; the only one you should modify is `main()` (you also should not change the global variables, except for `buffer`), but you will need to use them all. The existing main program is for demonstration only; it uses the other functions correctly.

The three functions, `start_encode()`, `next_frame()` and `finish_encode()` are used to generate an MPEG video that you can watch after your program terminates to observe your ant's behavior.

- `start_encode(int x_size, int y_size, int skip)`

  `x_size` and `y_size` are the x and y dimensions, respectively, of your ant's universe. Your ant may make many moves, and as enthralling as this is, it may get boring to watch, so to speed up the video, `skip` will cause the system to encode only every `skip`$^{\text{th}}$ frame.

- `next_frame(char *data)`

  After you've finished rendering each frame, you'll need to call `next_frame()` to signal the encoder to read and encode it (or to correctly skip it). `next_frame()` takes a pointer to an `x_size * y_size` array of `char`. You're not supposed to understand that yet; just call it like it's called in the example, and we'll talk about details of pointers and arrays very soon.

- `finish_encode(void)`

  `finish_encode()` doesn't take any parameters. Simply call it after your simulation has terminated and it will create your video. The video will appear in your working directory with the name `langton.mpg`. `finish_encode()` will also clean up all the temporary files that are created while the system runs.

You are to modify the supplied C program to implement Langton's Ant. Your ant should start in the center-most cell of your field and travel until in leaves the field.
See the syllabus for information about what to turn in and submission format. In particular, you must write, use, and turn in a *Makefile*!

Extra Challenges (nothing below this line is required)

- Modify your world so that your ant lives on a cylinder.
- Modify your world so that your ant lives on a torus; you'll have to change the termination criteria, because even though a torus is finite, it will allow your ant to travel infinitely.
- Modify your world so that edges "reflect" your ant away (or your ant turns around when it sees a wall); again, you'll have to change the termination criteria.
- Modify your world—and it's rules—so that your ant lives in $\mathbb{Z}^3$.
- Add multiple ants to your world. You'll need to create a rule for when an ant moves (or attempts to move) to an occupied cell. Perhaps they have a lilliputian death match?
- Make the field size a command line parameter.
- Modify your ant's initial state, by adding black cells, to make it stay inside your space longer, or to make it leave sooner (or, generally, to change its "standard" behavior of wandering off into infinity after about 10000 steps).
- Add a command line parameter to allow initialization-time state configuration.
- Move the encoder code into it's own file with a header, compile separately, and link it all together.
- Read the NetPBM specs and modify the encoder to allow color, then spruce things up (make your ant red, make the color of "black" cells vary with time, etc.).