Lab 07
Jonathan Gusler
3/28/2019

Overview
This lab's primary purpose was to introduce the use of the Universal Synchronous/Asynchronous Receiver Transmitter (USART) hardware included in the readyAVR board. This was used to transmit the contents of arrays to a computer monitor. Secondarily, it introduced the use of the strcpy function from the string.h library used in the C programming language.

Background
This lab requires knowledge of C programming language, cellular automata, cellular automata Rule 26, how to create and manipulate arrays in C, and how to setup and use PuTTy.

Discussion
The program simulates the life 122 generations of cellular automata which follow Rule 26. The cells are represented by space and * characters stored in arrays. The primary array is called current, holds the current generation, and is displayed 122 times. The next generation is built in another array, called next, which is populated based on the current generation and the rule set for Rule 26. Once it is populated, it replaces the contents of array current, which is then displayed again. Both arrays are initialized and next is copied into current using the strcpy function, which does require a cast for both arrays because strcpy takes char types as arguments but both arrays are unsigned char types. The code is shown in Appendix A.

Analysis and Results
Half resulting pattern is seen in **Figure 1**, the rest is a duplicate.

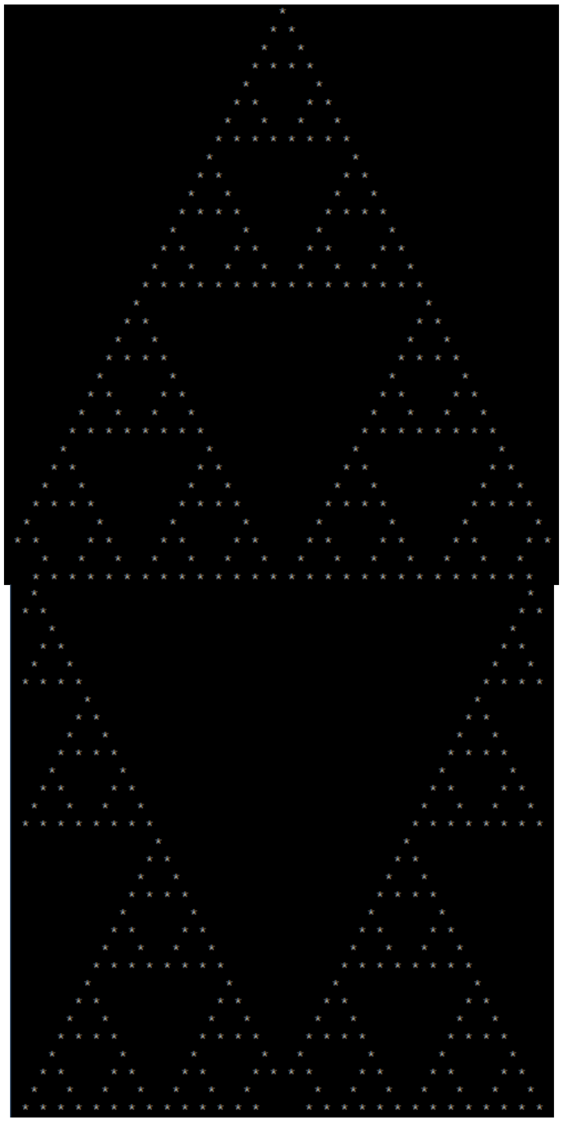

Figure 1: First half of resulting pattern produced by Rule 26, the other half is a duplicate.

Conclusion

The lab was successful in implementing cellular automata Rule 26. It was also successful in introducing the strcpy function, use of USART, type casting, and arrays.

Appendix A

```
/***************************************************************************
 * Lab07.c
 *
 *   Created: 3/16/2019
 *    Author: Jonathan Gusler
 * Processor: ATmega128A (on a ReadyAVR)
 *
 * This program is a cellular automata simulator. Given a starting
generation,
 * the colony of cells go through 122 generations, each defined by the
previous.
 * Each new cell is determined by what it was previously, as well as what its
 * neighbors were in the previous generation. This program used Rule 26
*create
 * the next generations.
 * YES I KNOW THESE LINES ARE SCREWED UP BUT THAT IS WORDS FAULT, I CANT FIX
IT
 * kinda want to watch Star Trek: The Next Generation TM now...
 *
 **************************************************************************/

/* #include file declarations go here */
#include "UARTLibrary.h"
#include <string.h>

/* #define declarations go here */
#define BaudRate 9600
#define DataBits 8
#define StopBits 1
#define NoParity 0

// prototype subroutine definitions
void initializeArray(unsigned char *cp);
void displayArray(unsigned char *cp);
void updateArray(unsigned char *cp);

// Array declaration for array "current"
unsigned char current[61];


int main(void)
{

      // Init the UART
      uart_init(BaudRate, DataBits, StopBits, NoParity);

      // Start by initializing the first generation, called current.
      // Next calculate the next gen, copy the next gen to the current gen
      // Display current
      initializeArray(current);

      for(unsigned char i = 0; i < 122; i++){
            displayArray(current);
            updateArray(current);
      }
}

void initializeArray(unsigned char *cp)
{
      // Initial generation is 30 spaces, a '*', followed by 30 more spaces
      strcpy((char*)current, "                              *
");
      return;
}
void displayArray(unsigned char *cp)
{
      // Send the entire array to the comm port
      for(unsigned char i = 0; i < 61; i++)
      {
            uart_tx(current[i]);
      }
      // New line character
      uart_tx(0x0A);
      // Carriage return character
      uart_tx(0x0D);
      return;
}

void updateArray(unsigned char *cp)
{
      // Init array the will that acts as a temporary next gen
      unsigned char next[61];
      strcpy((char*)next, "
");

      /* " " is white, * is black
      If a cell is on an edge, the "missing" cell is considered white
      Rule 26 rule set with the middle cell being updated
      Rule 1: '   ' => ' '
      Rule 2: '  *' => '*'
      Rule 3: ' * ' => ' '
      Rule 4: ' **' => '*'
      Rule 5: '*  ' => '*'
      Rule 6: '* *' => ' '
      Rule 7: '** ' => ' '
      Rule 8: '***' => ' '
      */

      for(unsigned char i = 0; i <= 60; i++)
      {
      if(current[i - 1] == ' ' && current[i] == ' ' && current[i + 1] == ' ')
            next[i] = ' ';

      if(current[i - 1] == ' ' && current[i] == ' ' && current[i + 1] == '*')
            next[i] = '*';

      if(current[i - 1] == ' ' && current[i] == '*' && current[i + 1] == ' ')
            next[i] = ' ';

      if(current[i - 1] == ' ' && current[i] == '*' && current[i + 1] == '*')
            next[i] = '*';
```

```
	if(current[i - 1] == '*' && current[i] == ' ' && current[i + 1] == ' ')
		next[i] = '*';

	if(current[i - 1] == '*' && current[i] == ' ' && current[i + 1] == '*')
		next[i] = ' ';

	if(current[i - 1] == '*' && current[i] == '*' && current[i + 1] == ' ')
		next[i] = ' ';

	if(current[i - 1] == '*' && current[i] == '*' && current[i + 1] == '*')
		next[i] = ' ';
	}

	// Copy the next gen array into the current one after current was 
displayed
	strcpy((char*)current, (char*)next);
	return;
}
```