CS 241 Lab 09 (RAM Error Injection)

Benjamin Stream
Solomon Himelbloom

March 29, 2021

1 Answers to Questions

- **Assignment 0:**
- Part 2 — Does this compile?
- No, it does not compile.
- Sketch uses 4154 bytes (12%) of program storage space. Maximum is 32256 bytes. Global variables use 2377 bytes (116%) of dynamic memory, leaving -329 bytes for local variables. Maximum is 2048 bytes.
- *Use Serial.begin(); and Serial.print(bigstring); to try to print the string.*
- Part 3 — Does this compile?
- No, it does not compile.
- Sketch uses 4426 bytes (13%) of program storage space. Maximum is 32256 bytes. Global variables use 2546 bytes (124%) of dynamic memory, leaving -498 bytes for local variables. Maximum is 2048 bytes.

- *Use the F() macro to store this string in program (flash) memory instead of RAM. You'll need to do this as an argument to Serial.print, not as a separate variable declaration.*
- Part 4 — Does this compile? Does it actually print?
- Yes, it compiles and prints.
- Sketch uses 3838 bytes (11%) of program storage space. Maximum is 32256 bytes. Global variables use 188 bytes (9%) of dynamic memory, leaving 1860 bytes for local variables. Maximum is 2048 bytes.
- *What does this mean about how and where Arduino strings are stored?*
- Strings are a very large data type, and it becomes inefficient to store large strings in RAM, thus they should be committed to storage or flash.
- **Assignment 1:**
- *What's the base-10 value of each bit? List the values. How many bits are in an Arduino "int"?*
- The decimal value of each bit (in base-10) is: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, +/- value; otherwise $2^0 \rightarrow 2^{14}$, and a +/- bit.
- There are 16 bits (2 bytes) in an Arduino Uno.
- Other Observations — Numbers (sometimes) switch from positive to negative. PC ints are 32-bit integers, whereas Arduino int 16 bits.

2 Arduino Commands

- **KEY:** Region — Start Pointer — Number of Bytes — Impact of Bit Errors

- Local string (inside) — Lstr — 7 — Medium impact: completely devastated the string. Also dependent on the number of times we loop the corrupt memory command (1000 times in our case).

- Global string (outside) — Gstr — 7 — Medium impact: completely devastated the string. Also dependent on the number of times we loop the corrupt memory command (1000 times in our case).

- A local array of ints (inside) — Larr — sizeof(Larr) — Small impact: if there are errors every iteration gets redefined since it is a local variable.

- A global array of ints (outside) — Garr — sizeof(Garr)— Large impact: errors can occur since this is a global variable that is not getting redefined. After a few thousand loops the 0 starts to become unrecognizable.

- All global variables — 256 — 300 — Ultra-large impact: Scrambling global does scramble all the global variables and any other variable outside the scope. Our help menu is almost unreadable now. Also, you may need to restart Arduino.

- All stack variables — 2000 — 300 — Large impact: Scrambling the stack may lead to the Arduino resetting.

3 Conclusion

- *Sketch uses 5526 bytes (17%) of program storage space. Maximum is 32256 bytes. Global variables use 710 bytes (34%) of dynamic memory, leaving 1338 bytes for local variables. Maximum is 2048 bytes.*

4 Appendix

4.1 Source Code

```
// Benjamin Stream & Solomon Himelbloom
// Assignment 2

void setup() {
  Serial.begin(9600);
  Serial.print("Ready for commands (v2.0)\n");
  Serial.print("*: Clears buffer.\n");
  Serial.print("help: Opens the help menu.\n");
  randomSeed(512);
}

String buffer;

// Global variables
int Garr[10] = {0};
char *Gstr = "gello\n";
String statement = "Scrambled ";

// Autoclears Buffer
void clearBuffer() {
  buffer = "";
}

void corruptMemory(void *startPointer, int nBytes, long bitErrorRate) {
  const unsigned long megaMask = 0xFFFFF; // == 20 set bits, approx 1 million
  unsigned char *start = (unsigned char *)startPointer;
  for (int i = 0; i < nBytes; i++)
    for (unsigned int bit = 0; bit < 8; bit++)
      if ((random()&megaMask) < bitErrorRate)
        start[i] ^= (1 << bit); // flip this bit
}

void loop() {

  while (Serial.available()) {
    char c = Serial.read();
    buffer += c;
    switch (c) {
      // Manual Buffer Clear if it gets cluttered
      case '*':
        Serial.print("Clearing Buffer...\n");
        Serial.print("Current Buffer:" + buffer + "\n");
        clearBuffer();
        Serial.print("Buffer Cleared!\n");
      default:

      case 'p':
        if (buffer == "help") {
          // "help" should print a brief summary
          // of the currently supported commands.
          Serial.print("\nHelp Menu (Lab 09.2):\n");
          Serial.print("*: Clears buffer.\n");
          Serial.print("localString: Tests corrupting local strings\n");
          Serial.print("globalString: Tests corrupting global strings\n");
          Serial.print("localArray: Tests corrupting local Array\n");;
          Serial.print("globalArray: Tests corrupting global Array\n");
        Serial.print("globalVar: Tests corrupting global variables\n");
        Serial.print("stackVar: Tests corrupting the stack \n");
        clearBuffer();
      }

    case 'g':
      if (buffer == "localString") {
        for (int i = 0; i < 1000; i++) {
          char *Lstr = "hello\n";
          corruptMemory(*Lstr, 7, random(10000, 20000));
          Serial.println(*Lstr);
          clearBuffer();
        }
      }

      if (buffer == "globalString") {
        for (int i = 0; i < 1000; i++) {
          corruptMemory(*Gstr, 7, random(10000, 20000));
          Serial.println(*Gstr);
          clearBuffer();
        }
      }

    case 'y':
      if (buffer == "localArray") {
        for (int i = 0; i < 1000; i++) {
          int Larr[10] = {0};
          corruptMemory(*Larr, sizeof(Larr), random(10000, 20000));
          Serial.println(*Larr);
          clearBuffer();
        }
      }

      if (buffer == "globalArray") {
        for (int i = 0; i < 1000; i++) {
          corruptMemory(*Garr, sizeof(Garr), random(10000, 20000));
          Serial.println(*Garr);
          clearBuffer();
        }
      }

    case 'r':
      if (buffer == "globalVar") {
        corruptMemory(256, 300, random(10000, 20000));
        Serial.println(statement + buffer);
        clearBuffer();
      }

    case 'k':
      if (buffer == "stackVar") {
        corruptMemory(2000, 300, random(10000, 20000));
        Serial.println(statement + buffer);
        clearBuffer();
      }
    }
  }
  delay(250);
}
```