ENGI E1102 Departmental Project Report: Computer Science/Computer Engineering

David Figueroa and Justin Zhao

May 10, 2012

Abstract

Using the powerful programming language of C, the language of the code used in Hewlett-Packard's HP-20b Business Calculator, we, under the provision of Professor Stephen Edwards and Yoonji Shin, redesigned the previous functionalities of the calculator to provide the user with a completely different finished product, a fully-functional Reverse Polish Notation (RPN) calculator. This calculator "hacking" project consisted of four smaller projects that each built upon the previous one. Even though the end goals at each of the four steps were clear, the openness of the programming platform allowed for much freedom in terms of how the given task could be accomplished. As a class, we went over different groups' codes, critically examining and discussing what could have been better or more efficient. Nevertheless, seeing the engineering perspective behind calculators served as a new experience for both of us, successfully immersing ourselves in the language of C.

1 Introduction

The HP 20b Business Calculator shown in Figure 1 is a standard business calculator with special features for financial and business-related applications. In this course, we explore the possibilities of reprogramming the HP 20b calculator to perform RPN computations. In the labs leading up to our final product, we review different groups' solution to the common objectives presented in each of the four labs. The calculator code is built and modified in the programming

language C, which also serves as a valuable memory conservation because the HP 20b only has 128 kb of flash memory. Under the valuable guidance of Professor Edwards and Yoonji Shin, we were able to successfully produce a final RPN calculator product.

In this report we have included a user guide for our RPN Calculator (Section 2), the social implications of the calculator (Section 3), the platform on which our calculator was built (Section 4), the software architecture for our calculator program (Section 5), the details of each lab and the code for each (Section 6), the lessons we learned throughout the class and our experience with the calculator (Section 7), and finally our criticisms of the course (Section 8).

Figure 1: HP 20b Calculator[2]

2 User Guide

2.1 Reverse Polish Notation

In the simplest of situations, Reverse Polish Notation (RPN) calculators require a number to be input followed by another and then an operation is then used to manipulate the two numbers previously input. When inputting expressions to be solved the operators are input after the numbers. To add two numbers such as 7 and 5 you would input [7], [5], [+] instead of what one would normally be used to with the operator in between the two numbers.

2.2 The Stack

Because RPN calculators need for numbers being input to be saved before being manipulated, the numbers are stored into a stack of numbers with a maximum of 10 input numbers. This means that up to 10 numbers may be saved into the calculator for manipulation. By entering a number and pressing INPUT, the number is saved into the next available spot on the stack and will be the number that is manipulated after inputting another number if followed by an operation key.

2.3 The Keyboard

The keys available for our use on the HP 20b calculator are:

- The 10 number keys labeled 0-9, used to input numbers onto the display
- The 4 operation keys labeled +, −, and × used to indicate operations to be performed on the numbers previously input
- The key labeled +/− used to change the number being displayed to positive or negative
- The INPUT key used to save the number being displayed
- The ON/CE key used to turn on the calculator
- The ⇐ key used to erase numbers incorrectly input

2.4 The Display

The Display on the HP 20b calculator is an LCD display with 12 functioning spaces. It displays the number last input, the result after an operation, the next number on the stack to be manipulated or some type of error message.

2.5 Using the HP 20b Calculator as an RPN Calculator

To turn on the calculator simply press the ON/CE key located at the lower left corner of the keyboard area. To input a number simply press the desired number keys on the keyboard and the number will come up on the display.

To start manipulating numbers with the calculator remember that the operations must be input after the numbers. Here are some examples to demonstrate how the calculator works.

Example 1: Adding 3 and 5

1 - Press the [3] key followed by the INPUT key.
2 - Press the [5] key and then press the [+] key. The sum of the two numbers, 8, should now be displayed on the display

Example 2: Subtracting 12 from 8

1 - Press the [8] key followed by the INPUT key
2 - Press the [1] and [2] keys followed by the [−] to get the difference, -4

Example 3: Adding 2 to the product of 3 and 4

1 - Press the [3] key followed by INPUT key
2 - Press the [4] key followed by the [×] key to get 12.
3 - Press the [2] key followed by the [+] key to get 14.

Example 4: Multiplying 2,3 and 4

1 - Press the [2] key followed by INPUT
2 - Press the [3] key followed by INPUT
3 - Press the [4] key followed by [×] to multiply 4 and 3
4 - Press the [×] key again to multiply that product, 12 by 2 to get 24.

Example 5: (10×3) − (5×2)

1 - Press the [1] and [0] keys followed by INPUT
2 - Press the [3] key and [×] to multiply the two numbers to get 30
3 - Press the [5] key followed by INPUT
4 - Press the [2] and the [×] key to multiply 5 and 2 and get 10
5 - Press the [+] key to then add 10 to 30 and get 30

3 Social Implications

Calculators such as the HP 20b are used for all kinds of calculations such as those of a student performing simple calculations to those of scientists calculating a rocket's projections through space. Calculators are needed to perform calculations quickly in many different situations, though our reprogrammed calculator would not do well in projecting the path of a rocket, it is very useful for students learning something simple such as the order of operations. Through this project we see the possibilities of what we can make a simple calculator do by reprogramming it. The program we created for the calculator is just one small example of the many things we can do with calculators.

4 The Platform

The public availability of the HP 20-b calculator's software development kit and schematics made it the ideal calculator to reprogram. At the core of this calculator system is its processor, its LCD display, and its keyboard.[3] In order to communicate with the computer, Professor Edwards installed a JTAG header (seen in Figure 2) onto the back of the calculator. For power, the calculator used up to two C2302 3V batteries. Using Professor Edwards software foundation for the calculator, we were able to reprogram the calculator.

4.1 The Processor

The HP 20-b calculator uses an Atmel AT91SAM7L128 processor, a member of the 7L series of microcontrollers designed for low power (L). The processor

boasts 128 kilobytes of high speed flash memory with 6kb of static random access memory (SRAM).[1] The processor operates at a frequency of 36 MHz, yet consuming just 0.5 mA/MHz when active and 100 nA when powered off. [1] Part of this impressive power saving also comes from the processor's feature of leaving unused peripheral components of the calculator unpowered.[1] The processor controls the power supply to every peripheral and manages every peripheral's power consumption. More details on this processor can be read on the Atmel website. Figure 3 shows a block diagram of the processor.

4.2 Liquid Crystal Display (LCD)

The calculator's LCD display served an important role in the reprogramming of the HP 20-b calculator. When running our new program, the calculator's LCD display was our only means of assessing if our program was truly working correctly. The LCD features fifteen digit display spaces, each space consisting of seven segments. Each of the 12 larger displays are separated by periods and commas while the three smaller ones on the right side are reserved for exponents. There is also one LCD segment on the very left, reserved for printing the negative sign for a negative number, but these LCD displays are unused. For our RPN calculator program, we instead reserve the left-most display space for the negative sign while 11 digits spaces are used for displaying numbers. For modifying the LCD, Professor Edwards provided the class with the functions *lcd_init*, *lcd_put_char7*, and *lcd_print7*. *lcd_init* initializes the LCD display, *lcd_put_char7* puts a character into one of the 15 displays, and *lcd_print7* which prints a sequence of characters. From lab 1, we add the LCD function *lcd_print_int* to put an integer onto the LCD. Figure 4 shows the schematics for the LCD while Figure 5 shows an example of "HELLO" on the LCD.

Figure 2: The Calculator's JTAG connector (Courtesy of Professor Edwards)

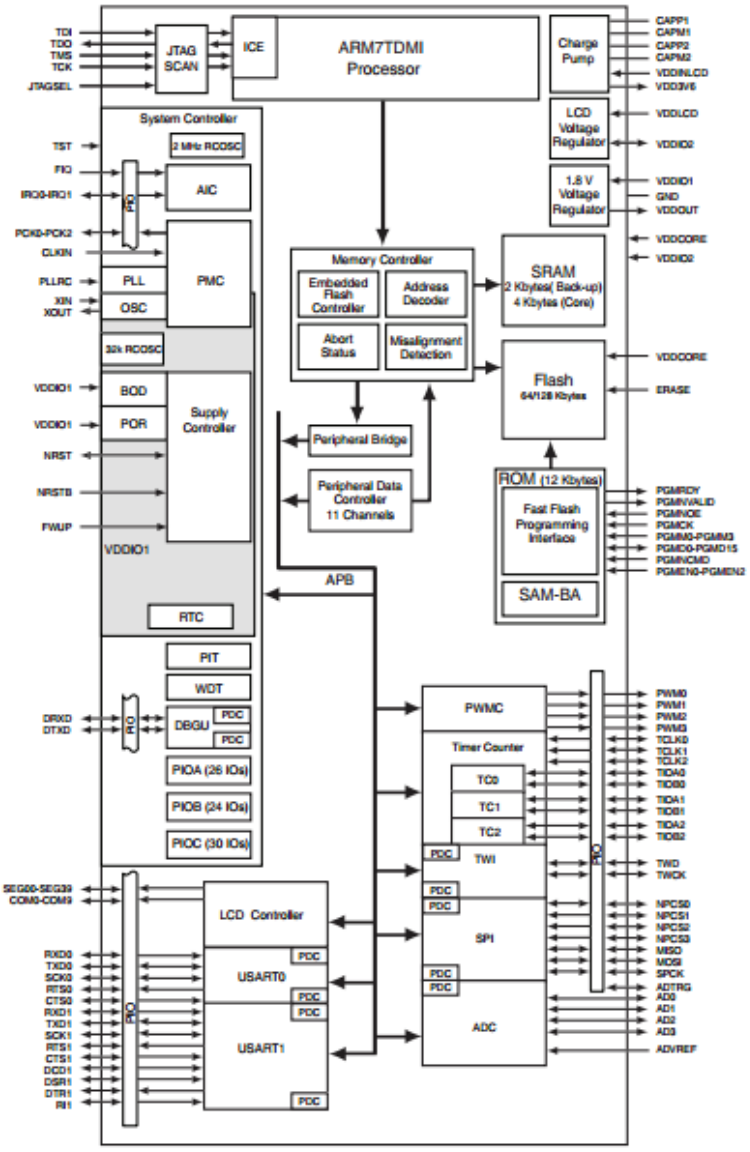

Figure 3: The Calculator's Atmel Microcontroller[1]

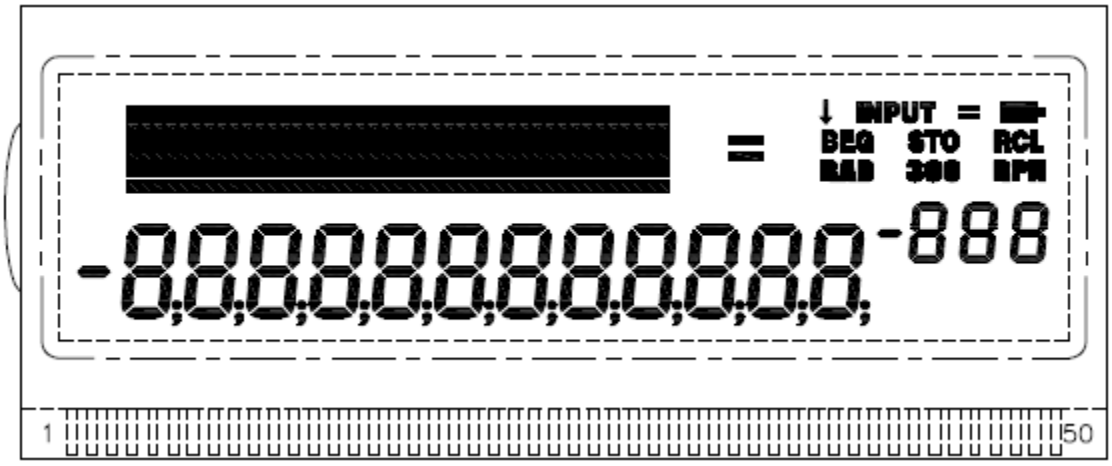

Figure 4: Calculator LCD Layout[2]

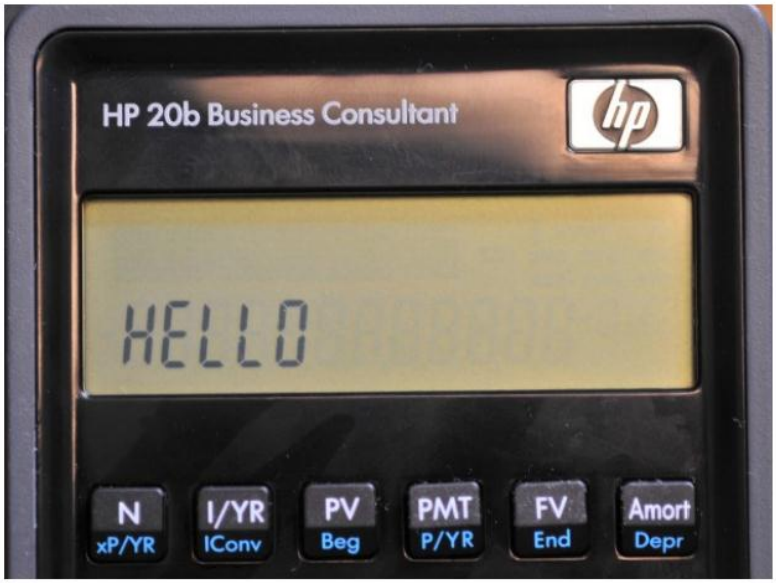

Figure 5: An Example of Characters on the LCD

4.3 Keyboard

The calculator keyboard works by assigning each key to unique combination of a column number and a row number, where the key represents where the column and row intersect. When a key is pressed, the two separated column and row wires are shorted together, indicating the correct key. Using this information, we can find the precise key that is entered by a user. Refer to Figure 6 and 7 for the calculator keyboard layout.

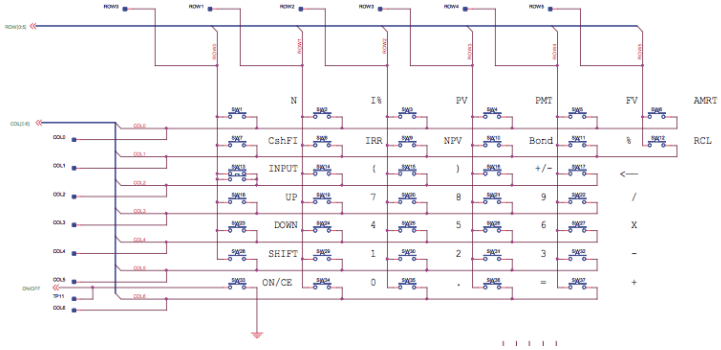

Figure 6: Calculator Keyboard Layout[2]

Figure 7: Calculator Keyboard

5 Software Architecture

The functions we used in our RPN calculator program include *keyboard_key()*, and *keyboard_get_entry(struct)*. The first method, *keyboard_key()*, scans the keyboard to see what key(s) is being pressed. The *keyboard_get_entry(struct)* method then returns the number that has been input and the operation key that was pressed. These are both used in the main method of the RPN calculator lab to receive instructions from the keyboard to make the calculator function.

6 Software Details

6.1 Lab 1: Getting Started: Hello World (Figure 8)

In the first lab of four, we were introduced to the importance of embedded programming for the 10 billion+ prevalent processor chips in our cell phones, computers, and mp3 players. Our task was to create a function in C that takes an integer argument and displays it in decimal form on the calculator LCD screen. For this task, we wrote a helper function in *hello.c*, the calculator's main class, *lcd_display_int*.

First, we count the number of digits in the number to be displayed. Then, we print every digit in the appropriate LCD space up to the number of digits in the number to be displayed. We reserved the first LCD space for if the number was negative for the "-" character. If the number was 0, then we printed a "0" character in the second LCD space. At the time, it seemed obvious that we would have to count the number of digits before printing the digits, but looking back, we realized that it was not necessary to count the digits first if at all if we decided to print the number aligned to the right side of the calculator. Most calculators that are used today display numbers aligned to the right side, so going back, we would have followed that convention.

```
int lcd_display_int (int toDisplay) {
   int digits = 0;
   int x = abs(toDisplay);
   while (x != 0) {
      x = x/10;
      digits++;
   }
   if (toDisplay < 0) {
      lcd_put_char7('-', 0);
   }
   if (toDisplay == 0) {
      lcd_put_char7(‘0’, 1);
   int i;
   for (i=1; i<=digits; i++) {
      lcd_put_char7((abs(toDisplay)%10)+48, digits-i+1);
      toDisplay = toDisplay/10;
   }
   return 0;
}
```

Figure 8: Our Solution for Lab 1 in *hello.c*

6.2 Lab 2: Listening to the Keyboard (Figure 9 and 10)

In this lab our task was to scan the keyboard to check if any key was being pressed. Initially all of the columns and rows are set to 1. The way to know if a key was pressed is if a row reads 0 when we have set a column to 0 which means that the contacts have made contact and have shorted. In our program, within the *keyboard_key* method, we created a for loop in which a row is set to low and then another for loop goes through each row to check if any reads 0. While no key is pressed the display shows the number 99 in the last two spaces of the display. When a key is pressed the column number of the key is displayed in place of the first 9 and the row number is displayed in place of the second 9 on the display.

```
int keyboard_key () {
   int column;
   int row;
   for (column=0; column<7; column++) {
      keyboard_column_low(column);
      for (row=0; row<6; row++) {
         if (!keyboard_row_read(row)) {
            keyboard_column_high(column);
            return (column*10)+row;
         }
      }
      keyboard_column_high(column);
   }
   return -1;
}
```

Figure 9: Our Solution for Lab 2, *keyboard.c*

```
int main()
{
   // Disable the watchdog timer
   *AT91C_WDTC_WDMR = AT91C_WDTC_WDDIS;
   lcd_init();
   keyboard_init();
   for (;;) {
      int x = keyboard_key();
      if (x == -1) {
         lcd_put_char7('0' + 9, 10);
         lcd_put_char7('0' + 9, 11);
      } else {
         int columnNumber = x/10;
         int rowNumber = x%10;
         lcd_put_char7('0' + columnNumber, 10);
         lcd_put_char7('0' + rowNumber, 11);
      }
   }
   return 0;
}
```

Figure 10: Our Solution for Lab 2, *main.c*

6.3 Lab 3: Entering and Displaying Numbers (Figures 11, 12, and 13)

In lab 3, we enabled the calculator to display entered numbers. The infinite loop in the code keeps the calculator running to find the button that the user pushes. Different buttons correspond to different meanings, translated by *keyboard_key*. Main (see Figure 13) first initializes a *struct entry* that consists of a number and an operation. Our function *keyboard_get_entry* is then called to fill the *struct*. Figure 12 shows the code for our *keyboard_get_entry* function. The way our solution works is to store the entered digits as an array of integers until the user enters an operation or the INPUT button. When this happens, we call on our helper function *get_number* (see Figure 11) that converts the array of integers into a full number, which is returned to fill the result *struct*. The operation button that the user enters is the operation that fills the result structure. The '+/-' button used to negate or de-negate numbers is also used. Each time the button is pushed, the integer "negative" is switched from 0 to 1 or from 1 to 0. Each time the user pushes the backspace button, the last digit that they entered is removed from the number. If the user enters an operation or INPUT before entering a number, the *struct* is filled with *INT_MAX*.

```
int get_number(int storedNumber[], int lcdPosition, int negative) {
   int number = 0;
   int i;
   for (i = 0; i < lcdPosition; i ++) {
      number = number*10 + storedNumber[i];
   }
   if (negative) {
      number = number * -1;
   }
   return number;
}
```

Figure 11: Our Solution for Lab 3, Helper Function in *keyboard.c*

```
void keyboard_get_entry(struct entry *result) {
   int a;
   int didOnce = 0;
   int lcdPosition = 0;
   int storedNumber[10];
   int number;
   int negative = 0;
   lcd_put_char7('0', 1);
   for (;;) {
      while (didOnce != 2) {
         a = keyboard_key();
         if ((didOnce==0) && (a=='0' || a=='1' || a=='2' || a=='3' || a=='4' || a=='5' ||
a=='6' || a=='7' || a=='8' || a=='9')) {
            if (a=='0' && storedNumber[0] == 0) {
            didOnce = 1;
            } else {
               lcd_put_char7(a, lcdPosition +1);
               storedNumber[lcdPosition] = a - '0';
               lcdPosition++;
               didOnce = 1;
            }
         }
         if (a==-1 && didOnce==1) {
            didOnce = 0;
         }
         if (a=='\b' && didOnce == 0) {
            if (lcdPosition > 0) {
               lcd_put_char7(' ', lcdPosition);
```

Figure 12: Our Solution for Lab 3, *keyboard.c*

```
        lcdPosition = lcdPosition-1;
        storedNumber[lcdPosition] = 0;
        didOnce = 1;
    }
}
if (storedNumber[0] == 0) {
    lcd_put_char7('0', 1);
}
if ((didOnce==0) && (a=='+' || a=='-' || a=='*' || a=='/')) {
    if (lcdPosition == 0) {
        number = INT_MAX;
        lcd_print_int(number);
        didOnce = 2;
    }
    else {
        lcd_put_char7(a, 11);
        result->operation = a;
        didOnce = 1;
    }
}
if (a=='\r' && didOnce==0) {
    didOnce == 2;
    number = get_number(storedNumber, lcdPosition, negative);
    result->number = number;
    lcd_put_char7('E', 11);
}
if (a=='~' && didOnce == 0 && (storedNumber[0] !=0)) {
```

Figure 12: Our Solution for Lab 3, *keyboard.c*

```
            if (!negative) {
               lcd_put_char7('-', 0);
               negative = 1;
               didOnce = 1;
            } else {
               lcd_put_char7(' ', 0);
               negative = 0;
               didOnce = 1;
            }
         }
         if (lcdPosition > 9) {
            lcd_init();
            lcd_put_char7('E', 0);
            lcd_put_char7('E', 1);
            didOnce = 2;
         }
      }
   }
}
```

Figure 12: Our Solution for Lab 3, *keyboard.c*

```
#include "AT91SAM7L128.h"
#include "lcd.h"
#include "keyboard.h"
int main()
{
  struct entry entry;
  // Disable the watchdog timer
  *AT91C_WDTC_WDMR = AT91C_WDTC_WDDIS;

  lcd_init();
  keyboard_init();
  keyboard_get_entry(&entry);
  return 0;
}

int keyboard_key(){
    int row, col;
    for (col = 0 ; col < NUM_COLUMNS ; col++) {
        keyboard_column_low(col);
    for (row = 0 ; row < NUM_ROWS ; row++)
        if (!keyboard_row_read(row)) {
        keyboard_column_high(col);
        return keyboard_keys[col][row];
        }
        keyboard_column_high(col);
    }
    return -1;
}
```

Figure 13: Our Solution for Lab 3, *main* and *keyboard_key*

6.4 Lab 4: An RPN Calculator (Figure 14)

Our goal for lab 4 was to make our calculator work as an RPN calculator. The calculator functions are all written right into the *main* method shown in Figure 14 below. The first thing we have in our code is the max stack size being defined as 10. In our infinite for loop, the first check, in the form of a while loop, is that the index on the stack is still within the stack. If the index goes past the stack size it returns an error message "Overflow" and moves the index back so you can input the last number again without saving it into the stack or just editing the two previous numbers in the stack.

Once in the while loop, *keyboard_get_entry(&entry)* is run. The program then checks for a number having been pressed before INPUT and makes sure that *INT_MAX* is not accidentally saved into the stack by returning an "ERROR" message and letting you input a new number. In the else portion the program goes into another if statement that once again makes sure that *INT_MAX* isn't saved into the stack and moves the index back one so that after inputting more than two numbers, the user can just press multiple operation keys and still have it work correctly with the previous numbers. If the number returned was not *INT_MAX* then the number is saved into the stack.

The next thing that comes up is the switch which then goes into the specific case according to the operation key pressed and manipulates the numbers. The number is then printed instantaneously and the program should then go back to the beginning of the for loop and continue running smoothly through the while loop and continuing to run once *keyboard_get_entry* return its *struct* from the user input.

```
#include "AT91SAM7L128.h"
#include "lcd.h"
#include "keyboard.h"
#define STACK_SIZE 10

int main() {
   *AT91C_WDTC_WDMR = AT91C_WDTC_WDDIS;
   lcd_init();
   keyboard_init();
   struct entry entry;
   int stack[STACK_SIZE];
   int index = 0;
   lcd_put_char7('0', 11);
   for (;;) {
      while (index < STACK_SIZE) {
         keyboard_get_entry(&entry);
         if (entry.number == INT_MAX && entry.operation == '\r') {
            lcd_print7("ERROR");
         } else {
            if (entry.number == INT_MAX) {
               index--;
               lcd_print_int(stack[index]);
            } else {
               stack[index] = entry.number;
            }
```

Figure 14: Our Solution for Lab 4, *main*

```
            if (index > 1) {
               switch (entry.operation) {
                  case '\r':
                     break;
                  case '+':
                     index--;
                     stack[index] = stack[index] + stack[index+1];
                     break;
                  case '-':
                     index--;
                     stack[index] = stack[index] - stack[index+1];
                     break;
                  case '*':
                     index--;
                     stack[index] = stack[index] * stack[index+1];
                     break;
               }
            } else  {
               lcd_print7("ERROR");
            }
            lcd_print_int(stack[index]);
            index++;
         }
      }
      lcd_print7("Overflow");
      index = 9;
   }
}
```

Figure 14: Our Solution for Lab 4, *main*

7 Lessons Learned

The most important lesson we learned over the course of this semester was the importance of taking advantage of guidance when you have it. Even though we were allowed equal access to the computer lab and calculators as other groups, our group fell a little short of time for completing the final fourth lab. While it was not procrastination or a lack of concentration that caused this, we do believe that it was our lack of fully taking advantage of the guidance the class offered. This behavior included not being able to go to class because of sickness, accidentally sleeping through a class, and not asking for help as often as we should when we could. As a consequence, we had to borrow a calculator from Professor Edwards after the last lab session and finish the last lab outside of class. Although we finished in the end, it took several more hours outside of class than it would have taken in class to finish the fourth and final lab.

We also learned the power of thinking through the design of the code on paper before coding it onto the computer. For some of the labs, because we began writing code before planning anything out, we wouldn't realize very inefficient problems in our code until much later. When we were stuck and had to debug, we would often lose track of where and what we were doing because we had no direction in the beginning.

8 Criticisms of the Course

There aren't too many things that can be criticized about the course. Professor Edwards' website was very helpful in providing the sufficient resources to begin the course. As people enter the lab with mixed computer science experience, it was a little harder for those unfamiliar with C to complete each of the labs. This is not to say a bad thing, however, as we learn the most when we are the most

challenged. The “Shotgun Intro to C” presentation was an especially helpful resource for getting through the labs. More so than anything, however, the active supervision and accessible help of Professor Edwards and Yoonji was what really made the lab valuable and worthwhile. The release of the next lab pushed those who hadn’t finished yet to hurry up while giving those ahead of the game another “game” to play. Each group worked at a different pace, and the handling of this fact was handled quite well, from our group’s perspective. Ultimately, the purpose of the lab was to challenge us in the realm of computer science while giving us a little bit of experience with C. The class accomplished this goal well and we look forward to more awesome computer science projects in the future.

9 References

[1] At91 arm thumb-based microcontroller preliminary. Online http://www.atmel.com/dyn/resources/prod_documents/doc6257.pdf, February 2008.

[2] Hp-20b business consultant financial calculator manual. Online http://h10010.www1.hp.com/wwpc/pscmisc/vac/us/product_pdfs/HP_20b_Online_Manual.pdf.

[3] Hp-20b. Online http://en.wikipedia.org/wiki/HP_20b.