ENGI E1112 Departmental Project Report: Computer Science/Computer Engineering

Feifei Kong, Nicole Lewis, Vanshil Shah

December 13, 2011

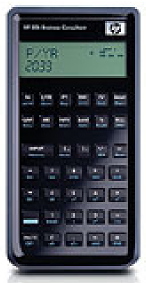
Abstract

As our group programmed a calculator throughout this semester, we learned a lot about the specifics about the HP 20b business calculator, while also discovered the basics of programming in C. The class was designed to expose us, many for the first time, to the world of computer science. To begin with, we looked into how to make the display work by creating a message that scrolls across the screen. This introduced us to writing code while beginning to make us familiar with the calculator. The next lab had us explore keyboards. After actually taking different keyboards apart to explore their design, we programmed how to know if a button on the actual calculator is pushed. Then we made the calculator begin to operate which involved reading the numbers pressed and displaying them on the screen while exploring deeper complexities of writing code. Finally we began to put everything together to make the calculator actually function with basic operations. We set it up as an Reverse Polish Notion (RPN) calculator so the numbers are entered into a stack and then the operations act on that stack. This final challenge combined with the entire course pushed us to combine old and new knowledge and produce a tangible result.

1. Introduction

Coming into Columbia Engineering as freshmen, we enrolled in the ENGI 1102 Gateway Design Course and then signed up for project section 005 which has a computer science and computer engineering focus. Within this course, we were assigned with the general project of programming a calculator. After being initially overwhelmed, the project was broken down and we began to discover the way the calculator operated.

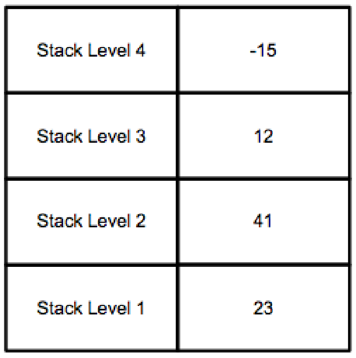
The calculator that we were given to program is a HP 20b Business Consultant Calculator. This calculator is primarily used by people in finance, insurance, real estate, accounting, and statistics.[1] This calculator normally has quite a range of functions, but we only dealt with the basics- the number keypad and simple operations such as addition, subtraction and multiplication. The code that makes these useful in an RPN calculator in lab 4 is displayed in section 6. In lab 3, we also made the backspace and negative signs function appropriately.


[1] The HP-20b

Throughout the semester, we spent a lot of time working to discover the different aspects of the calculator. As explained in detail in section 4, we looked at the platform of the calculator in the actual processor, the LCD display and the way a keyboard functions. As we wrote more code and learned the basics of programming in C, we worked from learning the basics of making the screen print, to reading the keyboard, and finally to pushing numbers and making the calculator function as it is supposed to. The code our group wrote is specifically explained section 6.

2. User Guide

We programmed the calculator to be an RPN calculator. The style of calculator where the operations follow the numbers was developed by Charles Hamblin in the 1950's to prevent the need of parentheses and confusion over the order of operations. It is called the Reverse Polish Notation (RPN) because it works in the opposite way a Polish Notation calculator does, which is where operations are all pushed and then the numbers.[2]

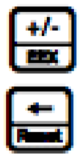


Stack Level 4	-15
Stack Level 3	12
Stack Level 2	41
Stack Level 1	23

[2] A stack

The basis of the calculator operations is a stack. This is where the numbers that have been entered are stored. An example of a stack is displayed in figure 2. The first number entered goes into level one after the user pushes the number and then input. When a second number is inputted then the number in level one moves up to level two and the new number goes into level one. This can continue until all of the levels of the stack are filled, in the figure this limit is four but on the calculators we made the limit ten. When a function key is pushed instead of a input after a number, or after two numbers have had input pressed after them, the bottom two numbers on the stack are taken and the appropriate operation is done to them and the result replaces the bottom number on the stack and everything above it moves down. In the figure, if the addition button is pushed, the 23 and 41 are added to make 64. Stack level one then holds 64, stack level two has the 12 in it and the third level holds the -15. The fourth level still technically holds a -15 as well, but is not going to be used because if the stack ever gets that high again it will be overridden.

To enter equations into an RPN calculator one must enter at least two numbers before a operation and then always have at least two numbers in the stack to operate on otherwise the calculator will continue to display the last result. When trying to equate 2+1, the user must push [2] [input] [1] [input] to place the 2 and then the 1 on the stack and then the [+] key. During this situation the screen will first display the 2 pushed and will keep that there while input is pressed until the 1 displays when the 1 and input is pressed. Once the plus key is pressed, the calculator will display 3, the result. If any other operations are pushed now, the calculator will remain with the 3 on display until a new number is pushed. To enter a multiple digit number, continue pressing digits before input is pushed. As digits are pressed, the number will grow on the screen and then remain there once input is pressed. For example, to enter the number 12343, press [1] [2] [3] [4] [3] [input].



[3]

Once numbers are already stored in the stack and been operated on, you can then enter more number and operate on them as well and then combine the results. For example, if you pressed [4] [input] [7] [+] [2] [2] [input] [7] [-] [+], to start with the 4 and 7 would be put in the stack and then added with the plus sign to produce a 13. Then the 22 would be put on the bottom, followed by the 7. Then the minus sign would operate on the last two inputs, resulting in a 15. Then the final plus sign would combine the 13 in the earlier spot with the 15 to make a final result of 28.

Other than the basic operations of plus, minus, and multiply, very few of the operation buttons do anything. The divide button does not work due to some technical difficulties with the programming libraries. The negative sign button does work. To make a number negative while inputting it, simply push the negative sign at any point while typing in the number, as long as it is before you push input. The negative sign will appear on the screen right before the digits when the number is negative. Figure 3 displays what the negative sign button looks like, along with the backspace button. The backspace button simply removes the last digit entered for each time it is pressed. For instance, if the numbers 3229 have been pushed, but not yet inputted, and you press the backspace button, the number will become 322, by removing the 9. If you press backspace after entering a negative sign, the last digit will still be removed. To instead get rid of the negative sign, press the negative button again. One can continue to press the backspace button until the entire number on the screen is gone, and a 0 will appear. If one continues to push backspace, nothing will happen. The remaining buttons on the calculator will not have any effect on the calculators functioning.

To begin working on the calculator, one must simply turn it on and a 0 will display on the screen to indicate that it is on and working. Once a calculation has been completed, to reset the calculator the user must unplug the power source to turn it off, plug it back in, and then turn the device on again. The clear button has no features. If an error is displayed on the calculator, one simply has to begin typing a new number to make it start working again. This process will restart the stack so you must begin the whole calculation again. An error message will appear if the calculator if it reaches a stack overflow because the number is too big. When entering numbers, the user cannot something with more than 9 digits. If it gets to this point, the calculator will simply stop responding to numbers after the 9th digit unless the backspace button is pressed. When computing, if the result exceeds 9 digits in length, the calculator will display the error message.

The table below shows some examples of the ways to perform operations that can be combined to do any simple calculations.

Equation	Keys	Display
2+1	[2][input][1][+]	2; 1; 3
15-7	[1][5][input][7][-]	1; 15; 7; 8
4+12+7-3	[4][input][1][2][input][7][input][3][-][+][+]	4; 1; 12; 7; 3; 4; 16; 20
(-1)x10	[1][-][input][1][0][x]	1; -1; 1; 10; -10
(1+2)x3	[1][input][2][+][3][x]	1; 2; 3; 9
(5+3)x(9-3)	[5][input][3][+][9][input][3][-][x]	5; 3; 8; 9; 3; 6; 48

[4] Example RPN

3. Social Implications

The HP-20b calculator is one of a long line of calculators that truly changed the world. Our modern day processors come from microprocessors used in HP and Texas Instruments calculators which were developed in the 1960s and 1970s. The first microprocessor, the Intel 4004 was used first on a Japanese calculator as well.[3] The experience we received in this class showed us how most of the devices that society relies on today came from devices such as the HP-20b that we hacked into. It was more than simply hacking into a calculator. It was hacking into a piece of history that gives an insight into the modern day world. The process of hacking into the calculator changed the way we looked at the world, and seeing every device that we use in a new naked way opened our eyes to the possibilities in this world. Technology will not stop growing and the more it grows, the more benefit it has in society, and this project inspired in us a spirit of development.

4. The Platform

The hardware platform used to implement our project is the HP-20b business consultant calculator. The HP-20b calculator is a finance calculator that is used worldwide and comeswith a high-speed low power processor.

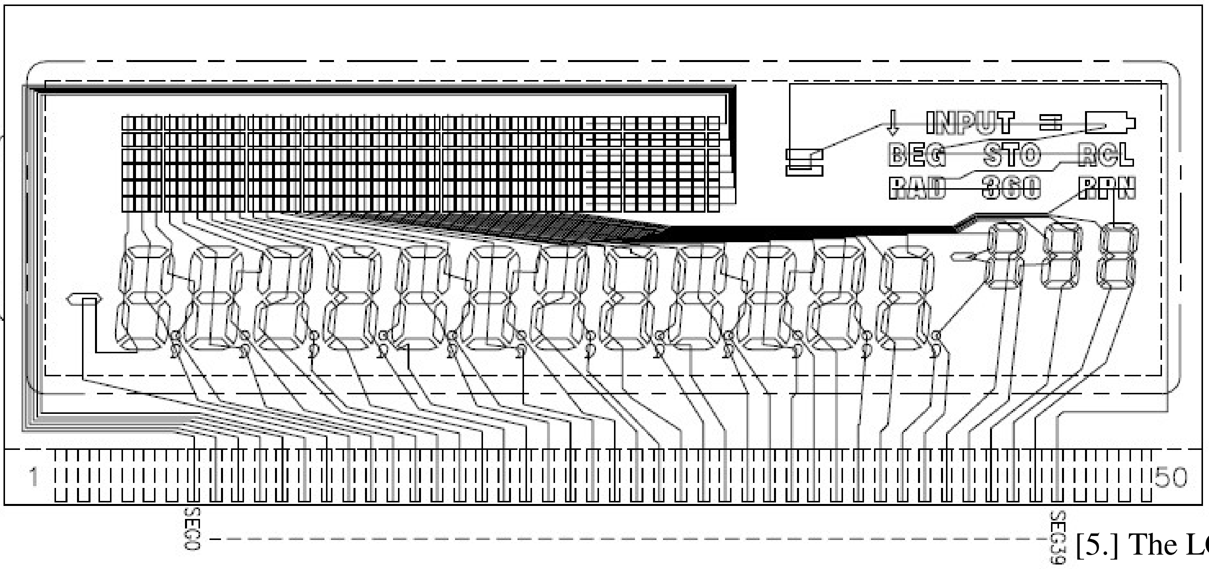
4.1 The Processor

The HP-20b is shipped with an Atmel® SAM7L calculator which is a high-performance processor with an ultra-low power requirement. This processor includes an ARM7TDMI core which is a part of the ARM family of 32-bit microprocessors. The ARM architecture is based on a Reduced Instruction Set Computer (RISC) which includes decode mechanisms that are much simpler than the architectures of Complex Instruction Set Computer (CISC) designs. The ARM core uses a pipeline to increase the speed of the flow of instructions to the processor, which allows the processor and memory to operate continuously. The flow is executed in three stages. The instructions are fetched, decoded so that the machine can use them, and then executed.[4]

The processor also has 128 kilobytes of internal high-speed flash memory, and 6 kilobytes of high-speed Static Random Access Memory (SRAM). Some peripherals that are included in the processor are the memory controller, the power management controller, which puts the calculator into different power saving modes, and the LCD controller, which controls the screen, which has a display capacity of forty segments.[5] The only peripheral that we used for our project was the LCD peripheral, but the code for manipulating the LCD was written by Professor Edwards as it requires a much higher level of programming skill than possessed by three freshman engineering students.

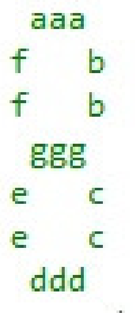
4.2 The LCD Display

The LCD included with the HP-20b calculator is controlled straight from the microprocessor, and has a display capacity of forty segments. The LCD can display 15 numbers, and has a space for two negative signs. The first 12 numbers it can print are large, and the last three are small, as shown in the figure. The LCD is a matrix of pixels,



[5.] The LCD

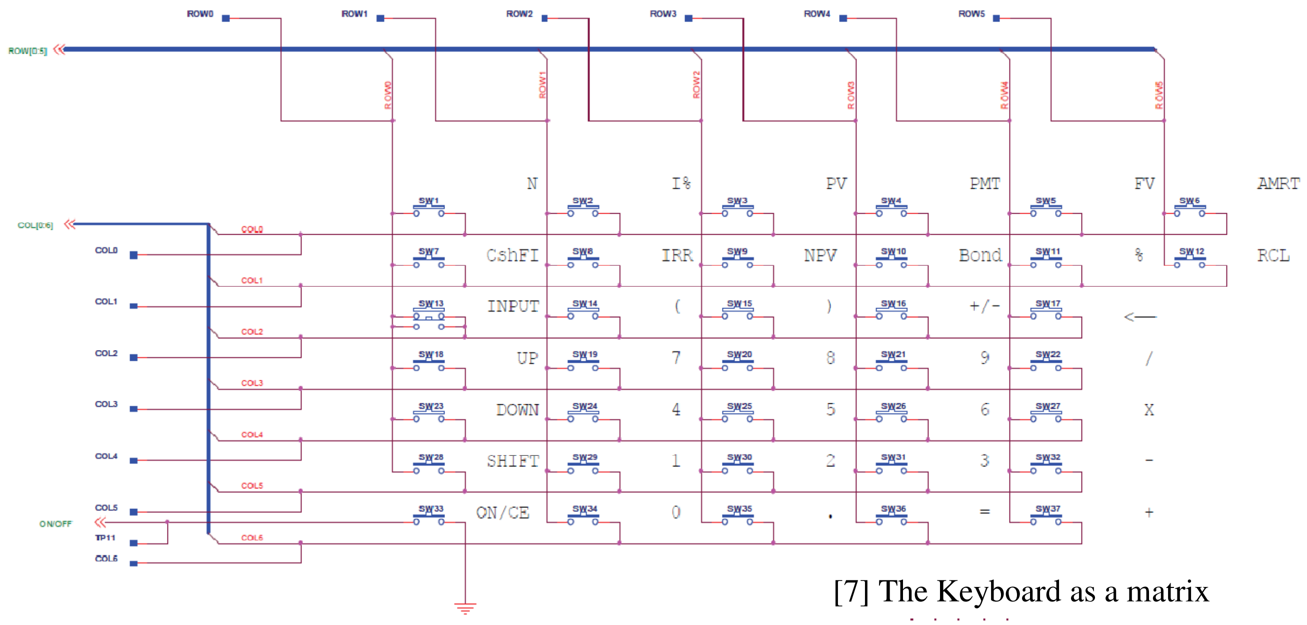
connected via rows and columns of wires. In the C source file, lcd.c, there is a two-dimensional matrix that splits up the display of each number into seven segments as shown in the figure on the right, and uses the locations on the LCD of each of these segments. The lcd_put_char7(char ch, int col) accepts a two parameters, the character to be printed, and the column to be printed in, and uses the matrix that is in the same source file to print out the inputted character in the corresponding 7 segment LCD position.



[6] The representation of a digit on the LCD

4.3 The Keyboard

The keyboard of the HP-20b calculator is a keyboard like any other. We took apart computer keyboards to reveal two sheets of wiring, and one sheet that acts as a buffer in between. Any keyboard is basically made of rows of wires and columns of wires. The keyboard works when we press down on a key and short the top and bottom sheets of wiring together. This shorting is read by the processor, which then knows exactly which key is pressed. The keyboard on the HP-20b is connected to pins on the SAM7L, which is the processor. The figure shows the matrix of keys that is created by the rows and columns of wires in the HP-20b. We wrote code in keyboard.c to control the keyboard. This will be further explored in section 6.



[7] The Keyboard as a matrix

5. Software Architecture

The first behemoth task for us newbie gateway children was to print out "Hello World" on the LCD that we had no idea how to control. Thankfully, the great professor Edwards did much of the dirty work of controlling the LCD, and provided us with lcd_print7(), and lcd_put_char7() functions that allowed us to print what we wanted on the screen. All we needed to do was think of a clever way to wrap the text around and display a scrolling message. The files that we mainly edited were main.c and keyboard.c. Most of the other functions in the package that we received just made the calculator turn on and allowed us to use the LCD.

Main.c and keyboard.c contained the instructions for making the calculator work as we wanted it to, and so we wrote code to send our instructions to the machine. The UNIX environment which no one in our group had experience with previously provided a challenge that was soon overcome, and compiling and sending instructions to the calculator became routine. First, we wrote code to print something on the screen, and then we wrote code to allow for a scrolling display. After this, we got into reading input directly from the user by using code to manipulate the keyboard. The calculator now accepted user inputs in the form of keystrokes on the keypad, and we also implemented code to display what the user inputted on the keyboard. The final challenge was creating a Reverse Polish Notation (RPN) calculator, using all of the code from the previous sections.

6. Software Details

6.1 Lab 1: A scrolling display

In the Lab 1 assignment, we were asked to create a scrolling message of our choice on the calculator. Our code, as shown below, consists of a nested *for* loop, with the inside loop printing each character of the character array across the screen, and the outside loop determining the placement of each character. As the counter *i* increments, the character placed on the calculator screen (*a[i % 21]*), as well as its placement (*i – j*), increments accordingly. Once the entire message is accounted for, the outside loop increments another counter, *j*, which creates the scrolling effect by essentially decreasing the placement by one unit. As *i* becomes greater than the length of the message, its remainder when divided by 21 is taken as the next character to be placed. The second inside *for* loop simply serves as a way to slow down the scroll speed by making the program count to a large number before proceeding.

Code for the scrolling display

```
#include "AT91SAM7L128.h"
#include "lcd.h"

int main() {
  lcd_init();
  int i, j, n;
  char a[] = ("Empire State of Mind ");
  for (j = 0; j < 1000; j++) {
    for (i = j; i < j + 21; i++) {
      lcd_put_char7(a[i % 21], i - j);
    }
    for(n = 0; n < 50000; n++) {
    }
  }
  return 0;
}
```

6.2 Lab 2: Scanning the keyboard

For the second lab, we scanned the keyboard by first creating a double array of single characters, the symbols and placement both determined according to the calculator's actual buttons. Next, we used a nested *for* loop, with the inside loop counting the row and the outside loop counting the column. The first column is set to low, allowing the program to check each row of that column. If the *keyboard_row_read* function returns false, the row is low, showing that the button is pressed. Two instance variables, *a* and *b* are saved as *i* and *j*, respectively, to indicate the button that is pressed. After the inside *for* loop scans all of the rows, the first column is set back to high, its count is incremented by one, and the process repeats until the entire keyboard is scanned. Once the entire keyboard has been searched, the program finds the correct row and column of the button being pressed from the double character array and prints out the button accordingly.

Code for scanning the keyboard

```
char keyboard_key() {
  char keyb[7][6] = {
    {'N', 'I', 'P', 'M', 'F', 'A'},
    {'C', 'R', 'V', 'B', '%', 'L'},
    {'U', '(', ')', '_', '<'},
    {'^', '7', '8', '9', '/'},
    {'v', '4', '5', '6', '*'},
    {'S', '1', '2', '3', '-'},
    { 0, '0', '.', '=', '+'}
  };
  int j, i, a, b;
  for (j = 0; j < 7; j++) {
    keyboard_column_low(j);
    for (i = 0 ; i < 6 ; i++) {
      if (!keyboard_row_read(i)) {
        a = i;
        b = j;
      }
    }
    keyboard_column_high(j);
  }
  return keyb[b][a];
}
```

6.3 Lab 3: Entering and Displaying Numbers

Lab 3 asked us to restore the original basic functions of a simple calculator, and our code can be seen on the next page. After initializing a set of instance variables, we implemented several *if* conditions inside an infinite loop. The first accounts for all the numerical values that are pressed (the numbers 0-9). It ignores the first digit if it is a zero in order to keep the maximum digits counter (9) correct and to better model the display of an actual calculator. As long as the total number of digits pressed is less than the maximum allowed, the immediate digit pressed (*num*) is changed into an integer. The total value in the calculator (*finalvalue*) is set as the current total value multiplied by 10 and added to the immediate digit pressed. In this way, every time a new digit is pressed, the original number increases by a factor of 10. This new value is stored in a temporary instance variable, *value*, so that *finalvalue* will not change with modifications of *value*. The next *while* loop serves to display the value in its correct position on the screen. While there is still a value in *value*, the display is set as the remainder when *value* is divided by 10, converted back into a character, and it is placed in the rightmost possible spot. *Value* is then modified so that it decreases by a factor of 10 (integers do not round), and the next smallest digit is printed to the left of the first. This is because the counter *count* increments by one every time, essentially decreasing the placement by one as well. The next *while* loop makes sure that the key pressed is released before performing any actions. After the entire loop, the counter *digits*, for the maximum number of digits allowed on the screen, is incremented to keep track of the limit.

The second *if* statement accounts for any non-numerical value pressed (operations, backspace, or +/- sign). If the symbol is the positive to negative symbol, *finalvalue* is multiplied by -1. In the display, if the value was not negative before, a negative sign is placed on the screen, and the *neg* condition becomes true. However, if the value was negative, the negative sign is taken away, and the *neg* condition becomes false. If the original value was equal to zero, the negative sign still appears, but the value is still zero. If the backspace symbol is pressed, *finalvalue* decreases by a factor of 10, and all of the numbers should move to the right one space. This happens by separating the digits and placing them one by one, starting from the rightmost side, very similar to the placement in the beginning. The digit all the way to the left is deleted by placing a blank space in that spot every time the number is moved to the right. If no number is entered, *finalvalue* becomes equal to the maximum integer value, and once an operation is entered, its symbol is displayed to the right of the numbers, and the program breaks.

Code for entering and displaying numbers

```
define MAX_DIGITS 9
#define LAST_SPOT 11
void keyboard_get_entry(struct entry *result) {
  int c;
  int num = 0;
  int finalvalue = 0;
  char op;
  int count = 0;
  int value;
  char disp;
  int digits = 0;
  int neg = 0;
  for (;;) {
    c = keyboard_key();
    if (c >= '0' && c <= '9') {
      if (!(digits == 0 && c == '0')) {
        if (digits < MAX_DIGITS) {
          num = c - '0';
          finalvalue = finalvalue * 10 + num;
          value = abs(finalvalue);
          count = 0;
          while (value > 0) {
            disp = (value % 10) + '0';
            lcd_put_char7(disp, LAST_SPOT - count);
            value = value / 10;
            count++;
          }
          while (c != 1) {
            c = keyboard_key();
          }
          digits++;
        }
      }
    }
    else if (c != 1) {
      if (c == '~') {
        finalvalue = finalvalue * -1;
```

In the fourth and final lab, we were asked to modify our calculator modeling an RPN calculator. Although some of the functions were similar to the basic calculator created earlier, many parts needed adjustments. Our code has a series of *if* statements inside an infinite *for* loop. The loop first sets *UserInput* as the value that is entered, and *UserOperation* as the operation that is entered. Then, the first *if* statement accounts for the number first entered, making sure that it does not go over the maximum capacity. It adds the number entered into the *stack* array, and increments the counter *StackPointer* by one, moving to the next empty space in the array. Once enter is pushed, that number has been finalized and stored in the stack. The same process happens until an operation is pressed. If there is only one item in the stack, nothing will happen when the operation is pressed. However, if there is more than one number, the operation will perform its task accordingly to the closest two numbers from the bottom of the stack. At the same time, *StackPointer* decreases by one because two of the values are being combined.

The latest value is stored as *value*, and if that number is greater than the maximum number, the calculator will print out an error. However, if *value* is equal to zero, the calculator will print out a zero in the rightmost place. If *value* is anything in between, *value* will be split into individual digits, converted into a character, and printed into the rightmost spot. *Value* is divided by 10 to decrease the number by a factor of 10, and the next digit is placed one spot to the left of the first. If the final value is negative, a negative sign is printed to the left of the display value.

Code for the RPN calculator

```
#include "AT91SAM7L128.h"
#include "lcd.h"
#include "keyboard.h"
#define MAX_STACK 10
#define INT_MAX 2147483647
#define MAX_DIGITS 9
#define MAX_NUMBER 999999999
#define LAST_SPOT 11
int main() {
  int i;
  struct entry entry;
  *AT91C_WDTC_WDMR = AT91C_WDTC_WDDIS;
  lcd_init();
  keyboard_init();
  int stack[MAX_STACK];
  int UserInput;
  char UserOperation;
  int StackPointer = 0;
  int count = 0;
  int value;
  char disp;
  lcd_put_char7('0', LAST_SPOT);
  for(;;) {
    keyboard_get_entry(&entry);
    UserInput = entry.number;
    UserOperation = entry.operation;
    if (UserInput == INT_MAX) {
    }
    else {
      stack[StackPointer] = UserInput;
      StackPointer++;
    }
      if(UserOperation == '\r') {
      }
      else {
        if (StackPointer == 1){
        }
        else {
```

7. Lessons Learned

Our group started off with little to no knowledge of C programming, or even computer programming in general. At first, the course seemed extremely abstract, with much room for individual innovations. However, with our trivial comprehension of the subject matter, it was very difficult to visualize and become familiarized with many of the concepts and syntax nuances. Nevertheless, as the course progressed, we were forced to think outside the box, frequently resorting to trial and error until we figured out many of the bugs in our programs. With both an approachable professor and a friendly TA, we were able to comfortably ask for help, thus expanding our knowledge and increasing our experience with C programming. As a result, we became progressively more absorbed in the new challenges thrown at us, eager to learn even more. Although we are still no more than amateurs, the basic process of planning out on paper, discussing our ideas with one another, and then slowly building the program will be both a fundamental and extremely significant approach to any programming project we are faced with in the future.

For those down the road, do not be intimidated by the first project. Because we do not spend a lot of time formally learning how to program, the course may seem extremely difficult for those who have not done any programming beforehand. The class is, for the most part, learning while doing. It is important to always ask questions and never be afraid of knowing too little. However, at the same time, never think of your code as the best possible solution, because there is always room for improvement. Treat the class with a lighthearted yet passionate manner, and it will turn out to be very rewarding.

8. Criticisms of the Course

This course, overall, was one of the most enjoyable for us this semester. One of the most important reasons was because the class was relatively small, making it much easier to communicate one on one with the professor. Although the style of the class was very laid-back, the objectives and requirements were clear. This allowed students to have fun with the assignments, instead of treating them as burdens. With so much room for self-improvement, each lab became a personal puzzle for each group to solve. The code reviews were extremely helpful because they helped our group realize the imperfections within our codes that we would not have noticed if we did not need to present it out loud. Sometimes when we reviewed our code as a group, we gleamed over those details, but with the professor and the TA's constructive criticism, we saw why a line of code was extraneous or confusing.

The course may have been a little less intimidating if we spent more time being introduced to C programming. Although we did spend some sessions going over a few of the basic concepts, many of the students did not have a programming background, especially with C, before coming to the class. Therefore, figuring out the syntax sometimes became troublesome. This made many of the labs difficult, even though our thought processes and general ideas were correct. Nevertheless, with time came experience, and therefore, we were eventually able to focus on the implementation of the assignment.

References

1. "HP 20b Business Consultant Financial Calculator Overview." HP: Hewlett-Packard. Web. 09 Dec. 2011. <http://h10010.www1.hp.com/wwpc/us/en/sm/WF05a/215348-215348-64232-20036-215349-3732534.html>.
2. "RPN - Reverse Polish Notation." The Calculator Home Page. Web. 09 Dec. 2011. <http://www.calculator.org/rpn.aspx>.
3. "Intel Museum – The Intel 4004." *Intel.com*. Web. 9 Dec. 2011. <http://www.intel.com/about/companyinfo/museum/exhibits/4004/index.htm>.
4. "ARM7TDMI Technical Reference Manual." Atmel, Apr. 2001. Web. 10 Dec. 2011. <http://www.atmel.com/dyn/resources/prod_documents/DDI0029G_7TDMI_R3_trm.pdf>.
5. "AT91 ARM Thumb-Based Microcontroller." *Atmel*. Web. 10 Dec. 2011. <http://www.atmel.com/dyn/resources/prod_documents/doc6257.pdf>.

Picture References

[1] "HP 20b Business Consultant Financial Calculator Overview." HP: Hewlett-Packard. Web. 09 Dec. 2011. <http://h10010.www1.hp.com/wwpc/us/en/sm/WF05a/215348-215348- 64232-20036-215349-3732534.html>.
[2] Hewlett-Packard Company. The RPN Stack. HP 20b Business Consultant Financial Calculator Manual. 18. Web. 9 Dec. 2011. <http://h10010.www1.hp.com/wwpc/pscmisc/vac/us/product_pdfs/HP_20b_Online_Manual.pdf>.
[3] Hewlett-Packard Company. The RPN Stack. HP 20b Business Consultant Financial Calculator Manual. 16. Web. 12 Dec. 2011. <http://h10010.www1.hp.com/wwpc/pscmisc/vac/us/product_pdfs/HP_20b_Online_Manual.pdf>.
[5] *LCD Artwork*. Digital image. Web. 9 Dec. 2011. <http://h20000.www2.hp.com/bizsupport/TechSupport/SoftwareIndex.jsp?lang=en&cc=us&prodNameId=3732535&prodTypeId=215348&prodSeriesId=3732534&swLang=13&taskId=135&swEnvOID=54>.
[7] Edwards, Stephen. "Repurposing an HP Calculator." Web. 10 Dec. 2011. <http://www.cs.columbia.edu/~sedwards/classes/2011/gateway-fall/keyboard.pdf>.