OVERVIEW

• Wrote firmware for an HP 20b calculator, allowing it to display user input and process information

• Worked with and understood the C programming language

• Went from modifying a simple method (lcd_put_char7) to working with control structures, structs, and pointers

• Instead of simply learning the syntax of C in a traditional classroom setting, we learned how to use Computer Science to affect society (i.e. alter what a widely-used device does)

• Took our first step in building applications

• Series of three lab experiments that took us from understanding display, to understanding how a keyboard works, to modifying the behavior of the calculator
HOW RPN CALCULATORS WORK

• The calculator is an RPN calculator, which stands for Reverse Polish Notation.

• You can represent anything mathematically without using parentheses and by saving keystrokes. The style of an RPN calculator, however, is not conventional.

• When entering computations, you must put in the first numerical value and then press INPUT. Then you enter a second value and an operation. The result is you get an evaluated mathematical expression. Thus, putting in a 5, INPUT, 6, + will return a value of 11. This is counter-intuitive to us because we want to do 5, +, 6, =. However, this is actually more efficient.

• For complex situations like evaluating (3 + 5)/(7 + 6), you would divide them into sub-expressions and then divide the two sub-expressions at the very end. So you would enter a sequence of commands identical to 3, INPUT, 5, +, 7, INPUT, 6, +, /. Notice that the division operator is at the end. Running this in your calculator will actually return the correct value of 0.6153846.
A LITTLE BIT OF BACKGROUND

- The software used for the HP 20b calculator is based on entry-system logic, which contains RPN and Algebraic and Chain Algebraic software. There are over 220 built-in functions and menus and prompts are included.
- Atmel AT91SAM7L128 processor
- The `lcd_put_char7` function is the essential tool to display output on the calculator. The function takes in two parameters, the first one a character and the second one a number. The function then returns the character at the given index numerical value.
LAB 1 CODE

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

void clearScreen() {
    int i;
    for(i=0; i<11; i++){
        lcd_put_char7(' ',i);
    }
}

void printOutput(int number) {
    clearScreen();
    int counter, remain; char character;
    int position = 11;
    int absArg = abs(number);
    for(counter = absArg; counter >= 1; counter /= 10) {
        remain = counter % 10;
        character = '0' + remain;
        lcd_put_char7(character, position);
        position--;
    }
    if(number < 0) {
        lcd_put_char7('-','0');
    }
    if(number == 0) {
        lcd_put_char7('0', 11);
    }
}

int main()
{
    lcd_init();
    printOutput(23444);
    printOutput(-0);
    return 0;
}
LAB 1 EXPLANATION

- Goal: display a numerical argument in the calculator
- Clear screen assigns empty spaces in every position in the screen. So it makes the screen ready to accept the number.
- After cleaning up the screen the variable takes the absolute value of input number. In the for loop, the number is kept divided by 10 and the remainder is stored in the right most position until the computer gets the one digit value of the highest digit number. For example, when you enter 205, the first result from for loop is 5 and it is shown in the right most position 11. And the remaining number 20 goes through the same process and 0 in position 10 and 2 in position 9. In case when the input number is negative, the number goes through the same process because the code takes the absolute value of the number prior to the for loop.
- To deal with the negative sign, lcd_put_char7('-', 0) displays '-' in the left most position. Also, if input is zero, the screen shows 0 in the rightmost position by lcd_put_char7('0', 11)
char keyboard_key()
{
    int i;

    char keyboard[7][6] = {
        {'X', 'X', 'X', 'X', 'X', 'X'},
        {'X', 'X', 'X', 'X', 'X', 'X'},
        {'I', '(', ')', 'O', '<'},
        {'^', '7', '8', '9', '%'},
        {'v', '4', '5', '6', 'X'},
        {'s', '1', '2', '3', '-'},
        {' ', '0', '.', '=', '+'}
    };

    for (i = 0; i < 7; i++)
    {
        keyboard_column_high(i);
    }

    int j, k;
    for (j = 0; j < 7; j++)
    {
        keyboard_column_low(j);
        for (k = 0; k < 6; k++)
        {
            if (!keyboard_row_read(k))
            {
                return keyboard[j][k];
            }
        }
        keyboard_column_high(j);
    }

    return ' ';
}
LAB 2 EXPLANATION

- Keyboard_key() works as follows: you first loop through all of the columns and you set them all to high. Initially you assume the worst (i.e. that the value you are trying to access is not in the calculator’s keyboard). Thus, the rowPosition is set to two and the colPosition is set to five. You then loop through all of the rows for each column. If the row is also low, then you have to return both the value of the row and the value of the column.

- We then created a three-dimensional array of all of the possible input values on a calculator and we return the element in that three-dimensional array that has a row position and a column position equal to that of the two low values.

- This function shown is invoked by main.c and is located inside of the keyboard.c file.

- We tested it with a wide range of values. We saw what would happen if you kept pushing down the button (it would keep displaying) and we saw what would happen if you tried to hit two values (only one would be displayed). Thus, we accounted for error.
void keyboard_get_entry(struct entry *result)
{
    int entry=0;
    int negative = 0;

    for (;;){
        //make some loop that calls keyboard_key() to check if it is true so as to see if whether a key is being pressed. While a key is pressed keep looping through and calling keyboard_key() until it is no longer being pressed, so once keyboard_key() is no longer true and then take that value. Every instance of keyboard_key() should be replaced by a variable initialized within the loop

        if (keyboard_key() >= 0 && keyboard_key() <= 9) {
            entry= entry * 10 + (keyboard_key() – '0');
            result->number = negative ? -entry : entry;
        }

        //check the cases that can be pushed
        else if (keyboard_key() == '~') //if the plus minus sign is pushed, make negative true.
        {
            negative = 1;
        }
    }
}
else if (keyboard_key() == '\r' || keyboard_key() == '+' || keyboard_key() == '-' || keyboard_key() == '*' || keyboard_key() == '/'))
    // if one of these are pressed exit and return what is in the pointer
    {
        result->operation = keyboard_key();
        return;
    }

lcd_print_int_neg(negative, entry);
}

The user is going to enter a string of numbers which create one large number by looping through some sort of display modification where the numbers stay on the screen and allow for the addition of other numbers.

When you have finished typing the number that you desire, you enter an operation. The program recognizes this operation and it returns the function, storing the operation and the number in struct.

These are passed to the main function where you can test that this works correctly by displaying the number and the operation.
LESSONS LEARNED

• Make sure that everyone in the team knows their role in the group. If someone has extensive experience with C, have them explain to the other group members what they have learned so they are caught up-to-speed. Leave no group member behind. Have no group member do all of the work.

• The code reviews were really helpful. The feedback was very good and we learned a lot about minimizing code and making code more readable while maintaining overall function.

• Also learned about the balance of commenting a program thanks to feedback
THE TEAM

Ga Young Lee (gl2429)
Noah Stebbins (nes2137)
Nicholas Sun (ns2874)
Bert Ramirez (lar2195)