Programming a Calculator

-Ashley Kling (ask2203), Joseph Thompson (jot2102), Phillip Godzin (pgg2105)
The HP 20b

- Originally 2 line display
- User is able to toggle RPN on and off
- Calculator is often repurposed and reprogrammed due to open software.
- uses Atmel AT91SAM7L128 30 MHz processor
Reverse Polish Notation

- First commercially available in 1963
- Parenthesis and brackets are unnecessary
- Instead has operations follow the numbers they are operating on
Features of RPN

● Makes use of a stack to store operands

● Each operator only works on two numbers

● Automatic storage of results allows for more complicated operations

● Operations cause calculations to occur immediately
Example

\((1+2)*3+4\) is 1,2,3,4+ . Evaluate to 13.

\(1+2*(3+4)\) turns to 1,2,3,4++ . Evaluate to 15
More on the stacks

Here is a demonstration of the stacks for the operation 4(A+B).

image from: http://www.theteacher.info/websites/ocr/WebPages/F453_Advanced/ConvertPolish/ConvertPolish.html
Lab 1- Display

- Number appears on right side
- Create a counter to keep track of negative
void display(int num) {
    clearScreen(); // gets rid of any current values on the screen
    int temp = num;
    const int ASCII = 48; // the ASCII value of 0 to be able to use numbers as chars
    int i = 0; // used for index
    int remainder = 0; // holds a single digit
    // If the number is 0, print it out and exit
    if (temp == 0){
        lcd_put_char7('0', 11);
        return;
    }
    // Turns a negative number into a positive number
    if (num < 0)
        num = -num;
    while(num!=0)
    {
        remainder = num % 10; // the last digit of num
        lcd_put_char7(remainder + ASCII, 11-i); // places digit in rightmost available index
        num = num/10; // Divides number by 10 for the next iteration
        i++;
    }
    // If the original number is negative, place a minus sign at the index immediately to the left of the first digit
    if (temp < 0)
        lcd_put_char7('-', 11-i);
}
Lab 2

To figure out what is pressed:
● Set all columns high
● Set column you want to test low
● Loop through rows. If a row is low, that is the button being pressed
● The pressed key's row and column numbers are returned

Other implementations:
● 2d matrix for integers
● Defined operations above the 2d matrixes
Lab 2- Finding Pressed Keys

```c
int keyboard_key()
{
    int c = 0;
    int r = 0;

    for(c; c<7; c++)
    {
        r = 0;
        keyboard_column_low(c);
        for(r; r<6; r++)
        {
            if(!keyboard_row_read(r))
            {
                return key[c][r];
            }
        }
        keyboard_column_high(c);
    }
    return NOTHING; // Nothing pressed
}
```
Lab 2- Other

#define X 99 // Nothing important is pressed
// The following buttons are pressed
#define INPUT 16
#define NEGATE 19
#define RETURN 20
#define DIVIDE 15
#define MULTIPLY 14
#define SUBTRACT 13
#define PLUS 12
#define EQUALS 11

// 2D matrix representing the rows and columns of the keyboard
int const key[7][6] = {
    {X,X,X,X,X,X},
    {X,X,X,X,X,X},
    {INPUT, X, X, NEGATE, RETURN, X},
    {X, 7, 8, 9, DIVIDE, X},
    {X, 4, 5, 6, MULTIPLY, X},
    {X, 1, 2, 3, SUBTRACT, X},
    {X, 0, X, EQUALS, PLUS, X}
};
Lab 3 - Storing number and operation

- Used a boolean to differentiate between a number and a function being pressed
- The number and operation pressed are stored in a structure
- If the +/- key is pressed, a variable that is initially 1 is multiplied by -1, then later the number is multiplied by that variable
- If no number is pressed before an operation is pressed, the max integer is returned.
- Numbers that are entered are printed on screen as they are pressed
void keyboard_get_entry(struct entry *result) 
{
    int num_pressed = 0; //boolean to see if an operation was pressed before a number
    int pos = 1; //determines if number is positive or negative: mult by -1 when +/- is pressed
    int tempOp = '';
    result->operation = ''; //Initially no operation
    result->number = 0;
    int keyPressed; //Stores the current key being pressed
    while(((result).operation == ' ')) //While operation + input has not been pressed
    {
        keyPressed = keyboard_key();
        if(keyPressed == NEGATE) //toggle sign of the number
            pos *= -1;
        if(keyPressed >= 0 && keyPressed < 10 && (*result).number < INT_MAX / 10) //number is being pressed
        {
            result->number = (*result).number * 10 + keyPressed;
            num_pressed = 1; // a number has been pressed
        }
    }
}
else if (keyPressed >= PLUS && keyPressed <= DIVIDE) //operation being pressed
{
    tempOp = keyPressed; //store operation
}
if(keyPressed == INPUT){
    result->operation = tempOp; //only set the operation once input has been pressed
    if(num_pressed == 0) //no number has been pressed
        result->number = INT_MAX;
    else
        result->number = (*result).number * pos;
}
if((*result).number != INT_MAX)
    lcd_print_int((*result).number);
else if ((*result).number == INT_MAX){
    lcd_putchar7('M',9);
    lcd_putchar7('A',10);
    lcd_putchar7('X',11);
}
}
Lab 4

- One pointer to the open space in an array with the lowest index
  - used to emulate a stack, in which numbers, both inputted and calculated are stored
  - to stay true to the original implementation of the calculator, the array has a size of 4

- +, -, *, and / functions implemented
  - when a function is pressed, the operation is immediately applied to the two numbers nearest to the stack pointer
else if (keyPressed >= INPUT && keyPressed <= DIVIDE) //operation being pressed
{
    result->operation = keyPressed;
    if(num_pressed == 0) //no number has been pressed
        result->number = INT_MAX;
    else
        result->number = (*result).number * pos;
}
Populating stack and executing operations:

In main.c:

```c
int stack[6];
int stack_size = 0;

while(stack_size < 6){
    keyboard_get_entry(&entry);
    if(entry.number != INT_MAX)
    {
        stack[stack_size] = entry.number;
        stack_size++;
    }
    if(entry.operation != INPUT)
        executeOp(entry.operation, stack, stack_size);
}
```

```c
void executeOp(int op, int stack[], int stack_size)
{
    int num1 = stack[stack_size-2];
    int num2 = stack[stack_size-1];

    int result = 0;
    if (op == PLUS)
        result = num1+num2;
    else if (op == SUBTRACT)
        result = num1-num2;
    else if (op == MULTIPLY)
        result = num1*num2;
    else if (op == DIVIDE)
        result = num1/num2;
    stack_size--;
    stack[stack_size] = result;
    lcd_print_int(result);
}
```
Skills Gained

- Ability to communicate semi-effectively
- Dividing problems into independent chunks
- Integration of hardware and software
- Working with colleagues who possess varying levels of programming skill
- Check your wires!