Peter Snipes
Zackery Devers
December 8, 2021
CMPSC 200

Final Report

Team 7

For our project we picked project track 1 which means that we will be doing our project on Low-end Programming Computing Tools. Zach and I were having a discussion on which project we wanted to choose between project 1 and 3. Project 3 caught our attention because it was a Student Designed Project which meant that develop an idea for their own project that focuses on one or more real-world topics in the field of low end computing. Project 1 required that we solve one or more low-end computing problems, programmatically using either C programming language or MIPS or both. So for our project we decided to build a project that consists of a basic calculator with simple functions that allows a C program to be executed simply. In our calculator we chose to include the plus, minus, multiply, divide, exponent, and log function.

Zack and I decided to do our project on the low end program computing tools because this way we can easily show case our prior knowledge from this

school semester and the research that we did while learning about these mathematical operations in C (addition, subtraction, multiplication, division, exponents, and the log function). As I said earlier we were in a little problem in the beginning because we didnt know which project direction we were going to go in. But when we discussed which project would best show our understanding of this coure we decided that project 1 was the best fit. Our motivation for chpooing project 1 is that this program can be very useful when doing some simple math. Other than just math it can help with different type of things in our binary table or while doing some computing like we done in class over this semester that involved math operations. This is very useful because you can efficentlty do different mathematical operations from your computer through this program.

For this project we are going to show our understanding of these operations by solving one or more low-end computing problems. We were given the option between either C programming language or MIPS or both but we decided that C would be the best way for us to dom this project. We will be implementing the addition, subtraction, multiplication, division, exponent, and log functions. We plan on doing our best to focus on low-end computing tasks and simulate the execution of those tasks through one or more programs implemented in C.

An overview of the project that Zack and I am going to do is that it will consists of a basic calculator with simple functions that allows a C program to be executed more simplier. In our calculator we chose to include the addition, subtraction, multiplication, division, exponents, and the log functions. In order to challenge ourselves when making a program that runs using the plus, minus, multiply, divide, exponent, and log functions. Using the math.h library function was not an option. This is because if we utilized other functions it would better showcase our understanding and knowledge that we acquired in class while also showcasing what was learned during our further research of C. Although we ran into some problems while doing the project that we choose we were able to successfully complete this challenge and it was ultimately the better route considering we wanted to showcase our newfound knowledge from this semester.

The challenge that we ran into wile completing this project was that we had to overcome making the calculations work correctly without the use of the math.h library function. The math.h function defines various mathematical functions and one macro. All the functions available in this library take double as an argument and return double as the result. We didnt use the math.h function because of the fact that it has all these built in functions. In order to challenge ourselves while

completing this project we decided that the best way to do this was to not use the math.h function.

The problem that we ran into while implementing the source code without the math.h function was we didn't fully understand the project sheet and built our code incorrectly to your standards of grading. We originally wrote our code with incode operators that were add, subtract, multiply, and divide. What we thought the prompt originally asked for was either a binary calculator with add, subtract, multiply, and divide or a calculator that could do the same functions and log and exponent without the math.h library. What we had to do is go back and fix our problems after our first submission. Originally we had the switch cases looking like this.

```
  switch (op) {
    case '+':
      printf("%.1lf + %.1lf = %.1lf", first, second, first + second);
      break;
    case '-':
      printf("%.1lf - %.1lf = %.1lf", first, second, first - second);
      break;
    case '*':
      printf("%.1lf * %.1lf = %.1lf", first, second, first * second);
      break;
    case '/':
      printf("%.1lf / %.1lf = %.1lf", first, second, first / second);
      break;
    case '^':
      while (second != 0) {
        result *= first;
        --second;
      }
      printf("Answer = %.0Lf", result);
      break;
    case '&':
      printf("%f", Logn(n, r));
      break;

    // operator doesn't match any case constant
    default:
      printf("Error! operator is not correct");
  }
  return 0;
}
```

What we did in order to fulfill your request was fully rewrite the cases and add new functions to the beginning to properly conduct these equations.

```
switch (op) {
  case '+':
    binAdd = binAddition(first, second);
    printf("Binary Addition: %d\n", binAdd);
    break;
  case '-':
    binSub = binSubtracton(first, second);
    printf("Binary Subtraction: %d\n", binSub);
    break;
  case '*':
    while(y != 0){
      product += x;
      y--;
    }
    printf("\nProuduct = %d\n", product);
    break;
  case '/':
    printf("The quotient is %d\n", divide(dividend, divisor));
    break;
  case '^':
    while (second != 0) {
      result *= first;
      --second;
    }
    printf("Answer = %.0Lf", result);
    break;
  case '&':
    printf("%f", Logn(n, r));
    break;

  // operator doesn't match any case constant
  default:
    printf("Error! operator is not correct");
}
return 0;
```

This is the new updated switch cases. To explain these changes lets go down the list. Firstly, we changed the addition to now have a binary add function. This function does all the work of addition without the use of the + operator by doing shifts in the binary value.

```
int binAddition(int a, int b)
{
    int c;
    while (b != 0) {
        c = (a & b) << 1;
        a = a ^ b;
        b = c;
    }
    return a;
}
```

What we did next was change the subtraction to look like this, it does the same thing as the addition with the one tweak in the beginning to convert the addition to subtraction.

```
int binSubtracton(int a, int b)
{
    int carry;
    b = binAddition(~b, 1);

    while (b != 0) {
        carry = (a & b) << 1;
        a = a ^ b;
        b = carry;
    }
    return a;
```

What was left after fixing the subtraction and addition was the multiplication and division. What we did to fix the multiplication was create a while loop function that has repetitive addition in order to simulate multiplication.

```
case '*':
  while(y != 0){
    product += x;
    y--;
  }
  printf("\nProuduct = %d\n", product);
  break;
```

This was very simple to write but does not involve the multiplication operand at all, so I believe it does the correct job that the prompt asks for.

```c
int divide(int x, int y){
    if (y == 0){
        printf("Error!! Divisible by 0");
        exit(-1);
    }
    int sign = 1;
    if (x * y < 0) {
        sign = -1;
    }
    x = abs(x), y = abs(y);
    int quotient = 0;
    while (x >= y){
        x = x - y;
        quotient++;
    }
    printf("The remainder is %d\n", x);
    return sign * quotient;
}
```

This is our fixed divide function that we wrote in order to correctly have a divide function without the use of the divide operand. It takes in the two inputs and relabels them as x and y. There is also a written error function if they enter a 0 because we cannot divide by 0. It is finalized by a while loop that takes in the x and y. If the x is greater than the y we create a new x value with x being subtracted from y multiple times until the x is less than the y, or no longer can intake the y. Every time it can be subtracted from the operand the quotient is added. After there are no longer and more subtractions allowed the quotient value is fulfilled and the remainder is printed. I believe that our exponent and log functions were okay because they were created from scratch to do the equation instead of using prewritten functions.

In conclusion Zack and I were able to complete the Project 1 anf meet the requirements after giving it a second look from what we were told we did wrong. With that knowledge we fixed the project to meet the requirements to satisfy project 1's requirements. This project was a great challenge and also a great learning experience. While completing the project I think we learned different stuff with these functions like when the operation is already imbedded in the program and how to run the program when it isnt already embedded. We also faced a challenge that we completed the project but it was slightly wrong so we faced some adversity and instead iof giving up on the project we decided that we would showcase our knioweldge from this school year and complete the project to meet the full requirements. With that being done I feel like we both feel accomplished and thats the biggest reward anyone can fell.