Lab 4

- CAI

Name: Rasmus Muhrbeck

Date: 2020-12-18

A lab report in the course DT555A Programming in C

Table of Contents

1. Introduction

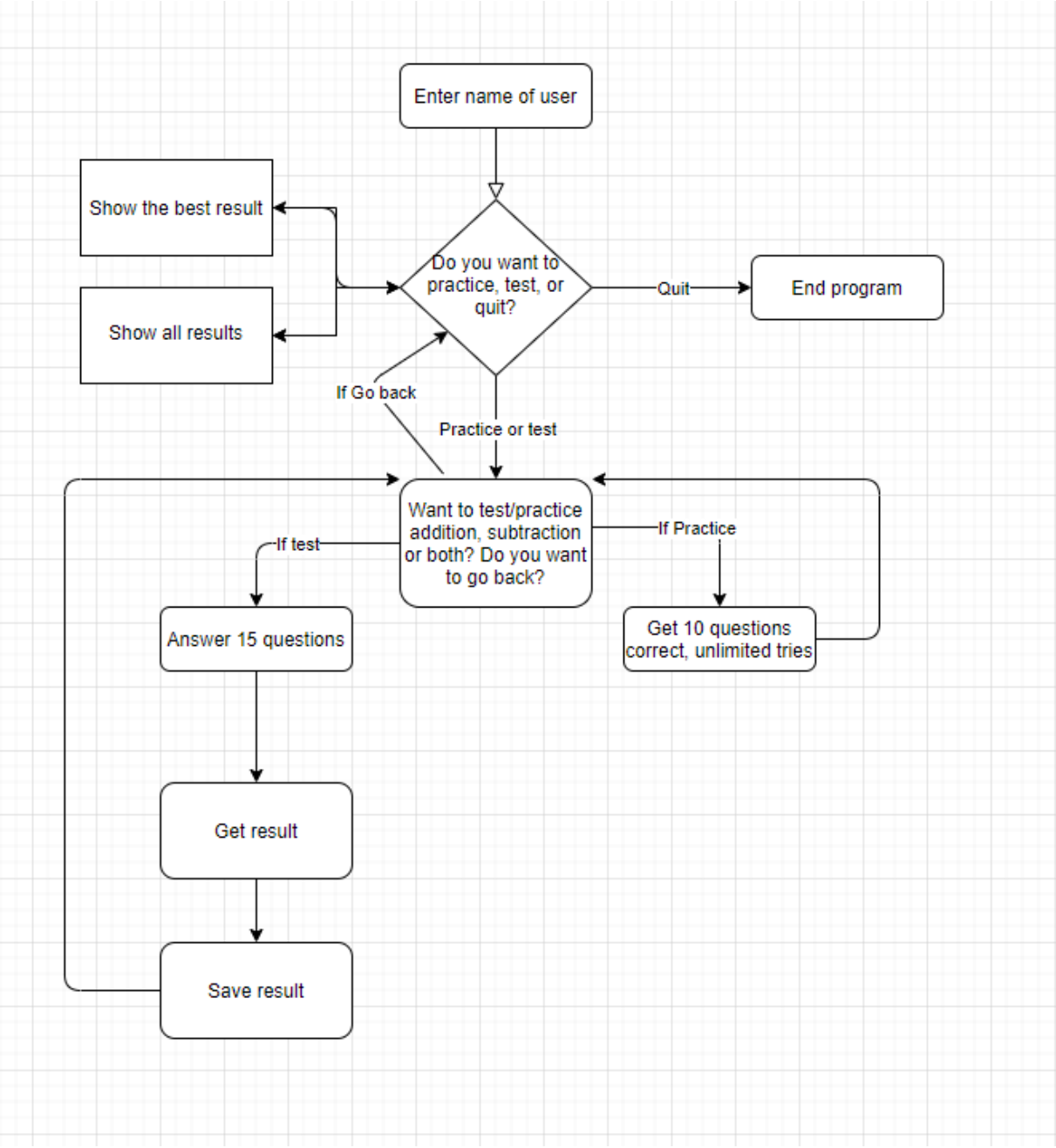
This lab was made to practice further algorithm development. This is done by implementing the things we have learnt to harder and more complicated problems. This lab is not as trivial due to that the problems needs to be divided into more sub-problems. Because this is an extension of lab 3, the problem didn’t need to be solved from scratch.

2. Design

Task 1 Grade 3: Computer Assisted Instructions

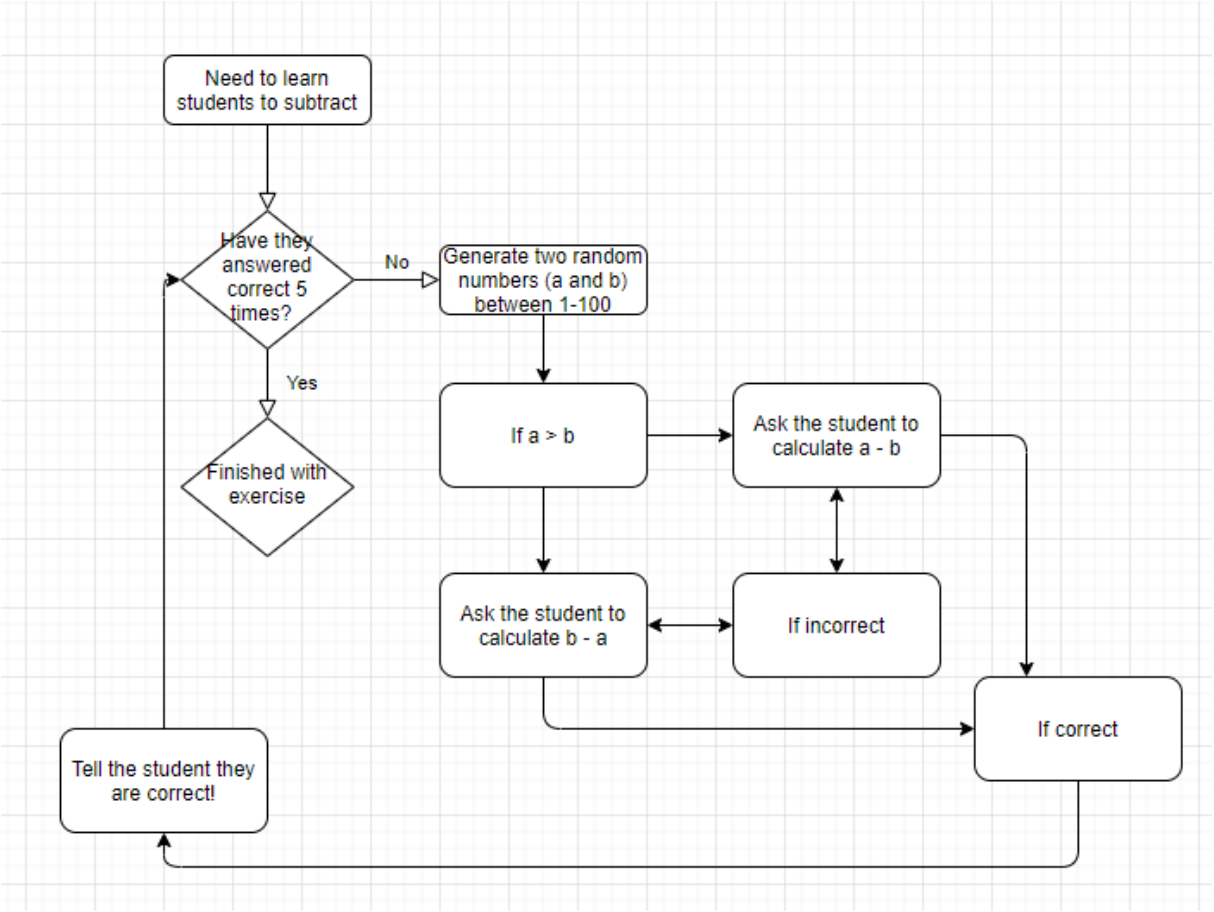
When first looking at the problem I concluded that the previous lab could be used to an extent to complete one of the sub-problems, the practice part. For it to work on the test-part of the solution it needed a small edit.

For the main menu I concluded that I only need to read the input of the user to know where he/she wants to proceed.



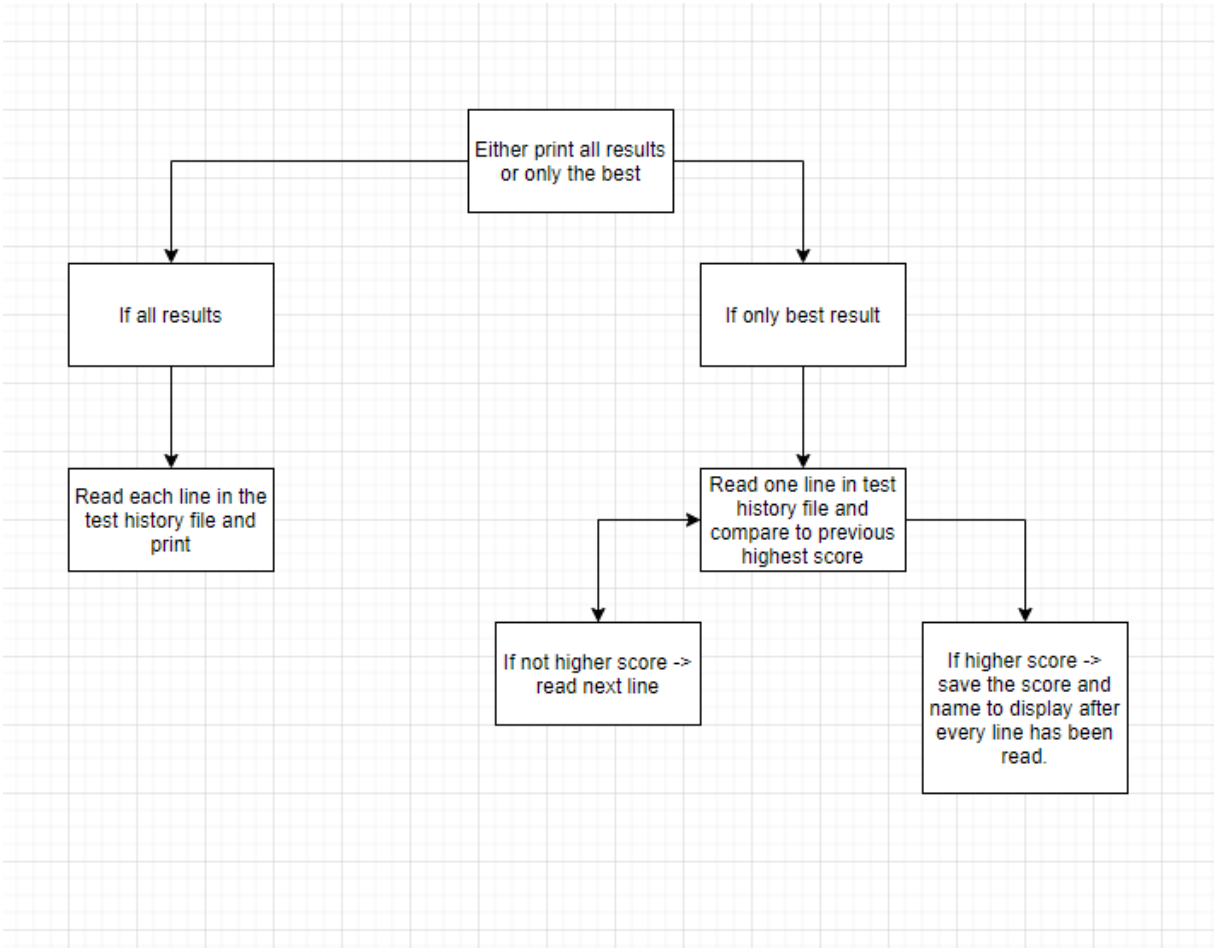
For the practice part thought was to have a counter that counts the amount of correct answers given by the student, and when it reached a certain number (5 in flowchart) it would be

finished. While the student hasn't answered enough correct guesses there would be 2 random numbers generated, lets call them a and b. To implement all three tests in one function I also included the users' choice as a parameter. This meant that I could alter whether the user was given addition, subtraction or both. In the case of negative numbers and the fact that my primary school students don't know negative numbers, I have to check if random number a is bigger than random number b. If that is the case, the program will ask the student to calculate a – b, and in the other case what b – a is. If the calculation is incorrect, the program will state that it was wrong and go back to asking the student to calculate the same a – b, or b – a, depending on the case. If the calculation is correct, the program will congratulate the student and after that check if the student has answered correct a certain amount of times. The program will then ask the student to calculate another two random numbers a and b. This will continue until the correct amount of answers required is achieved, which then will end the practice. The same logic was used to divide the addition-part of the practice.



For the testing part the same logic was applied with implementing the users' choice an as input parameter. But instead of needing to answer correctly 10 times, the test consists of 15 questions and will end after 15 answers. I implemented the test so that it gives a return value

of how many times the user answered correctly. This made it possible to calculate the percentage of the test result.



For the printing of scores it depends on if the user only wants to print the best result or every score. If the user wants to print every score then the program will read a line in the text file and print that so the user can see each result.

If the user only wants to see the best result, then we need to read each line and evaluate if that score is better than the ones before it. If it is, then we can save it as a local variable and later print it to the user.

For the saving of data a function that takes the amount of correct answers as input opens a file for appending, and writes in the user name and the score.

3. Implementation and Test

I start my code with implementing some main variables that I'm going to use. Firstly I ask the user for his/her name and save it to a local variable. Then I have a do-while loop that consists of the GUI for the user. Here the user can choose between 3 alternatives; practice, test or quit. By not choosing to quit, which quits the program, the user will be presented by a menu with 4 alternatives; addition, subtraction, both, and go back. These choices are the same for both practice and test.

For the practice part of the test I implement two variables to determine when the test will stop

```
/* Define how many correct answers necessary */
int amountOfCorrectAnswerRequired = 10;
int correctAnswers = 0;
/* While not enough correct answers given */
while(correctAnswers < amountOfCorrectAnswerRequired)
```

What I first do in the while loop is checking which scenario the user wants to practice on, if it is the pure subtraction or addition, the program will do this:

```
if(testScenario != 3)
    {
       additionOrSubtraction = additionOrSubtraction + 2;
    }
```

And if the user want to practice both:

```
else
    {
       additionOrSubtraction++;
    }
```

This is used later to determine if the user will face addition or subtraction by evaluating the modulus of the variable:

```
if(additionOrSubtraction % 2 != 0)
```

This way the program can adapt depending on what the user wants to practice on.

For the negative part I used this logic where I generate two random numbers and evaluate which one is biggest:

```
if(a > b)
    {
       aBigger = true;
```

```
    }
    else
    {
       aBigger = false;
    }
```

If a is bigger then the question to the student will be to calculate a-b. This will be evaluated and if it's the correct answer then this code segment will execute:

```
if(ans == a - b)
{ /
                * Correct answer, add one to the answer counter */
                printf("Correct\r\n");
                correctAnswers++;
                correctAns = true;
}
```

For the positive part I check if the total of a + b > 100, and if it is I divide both a and b with 2. This way the total will be below 100 and my students can only count to 100 so that's fair. I then implement it the same way as for the negative part if the user answers correctly.

The test part of the program is implemented in the same way, but instead of allowing the user unlimited tries on each question, the user only has 1 guess per question. If the user answer correctly a counter will be incremented and in the end of the program this counter will be returned to the main program to evaluate the result of the test. This is done by:

```
testCorrectAnswer = TestCalculation(calculationChoice);
resultOfTest = 100.0 * testCorrectAnswer/amountOfQuestionsTest;
printf("Result of test: %f percent!\r\n", resultOfTest);
```

The saving and loading of the file are based on the current working directory and therefore it does not matter where you save the project, it will still be able to load and save the file. The implementation of displaying the best result is shown below

```
while(fgets(str, 50, fp) != NULL)
    {
       currentNameHighScore = strtok(str, ":");
       // = tokenOfStr;
       highScore = atoi(strtok(NULL, ":"));
```

```
        if(highScore > highestScore)
        {
            highestScore = highScore;
            strcpy(nameOfHighScore, currentNameHighScore);
        }
    }
```

Where we first read a line until it returns NULL, which means there are no more lines to read. The test result list is structured as NAME:RESULT, which means that the line can be split into two part with the delimiter “:” to extract the name and result of the high score. To get the highest score a comparison between the previous highest score is made and saved if it is higher, and ignored if the highscore is lower than the previous high score.

4. Results and discussion

The preparation from lab 1, 2 and 3 came in handy in this lab due to a bit more complex problem. The way to break down the problem into several sub-problems made it easier to construct a code that works. By visualize the whole problem into several smaller problems I constructed a code with several smaller code segments that makes the program easy to read. The addition of saving to file and printing results were already quite small problems, and therefore it wasn't necessary to divide them into smaller problems.