DT555B Programmering i C

Lab 1 – Algorithm Development

Zandra Hedlund

230926

Table of Contents

1. Introduction

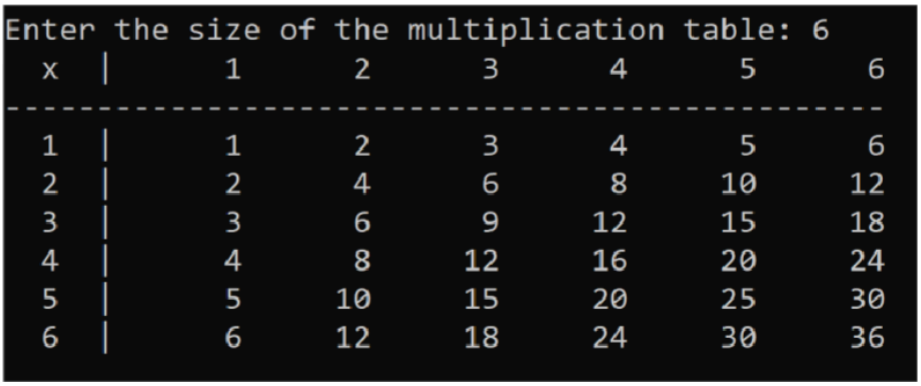
Algorithm development is the process of designing and creating a set of step-by-step instructions or procedures to solve a specific problem or perform a particular task in programming. It involves defining the problem, understanding the requirements, and come up with a sequence of actions that can be followed to achieve the desired outcome. Algorithm development is a fundamental aspect of software engineering and data analysis, and it plays a crucial role in solving complex problems. Effective algorithm development requires a combination of creativity, problem-solving skills, and an understanding of data structures.

Task 1 - Computer Assisted Instruction (CAI)

This task was to write an algorithm for a program that will help primary school students practice addition. The program will generate an addition question, with two random numbers in the range 0-100 and the student will provide the answer. The question will be repeated until the student gives the right answer.

Task 2 Multiplication Table

This task was to write an algorithm for printing a n x n multiplication table of size n. The user will enter a number between 1-11 to generate the table. The table should be formatted with the numbers in straight right aligned columns and a border, as shown below:



2. Design

I started by breaking down the task into smaller tasks, to figure out which steps were needed to get the desired result. I then used pen and paper to make a step-by-step instruction, which I then translated into pseudocode and a flowchart. I used a top-down design, since I think that makes it easy to follow the logic. I tried to install AlgoBuild, but it didn't work on my Macbook, so I used an online tool, lucid.app to make the flowcharts.

Task 1 - Computer Assisted Instruction (CAI)

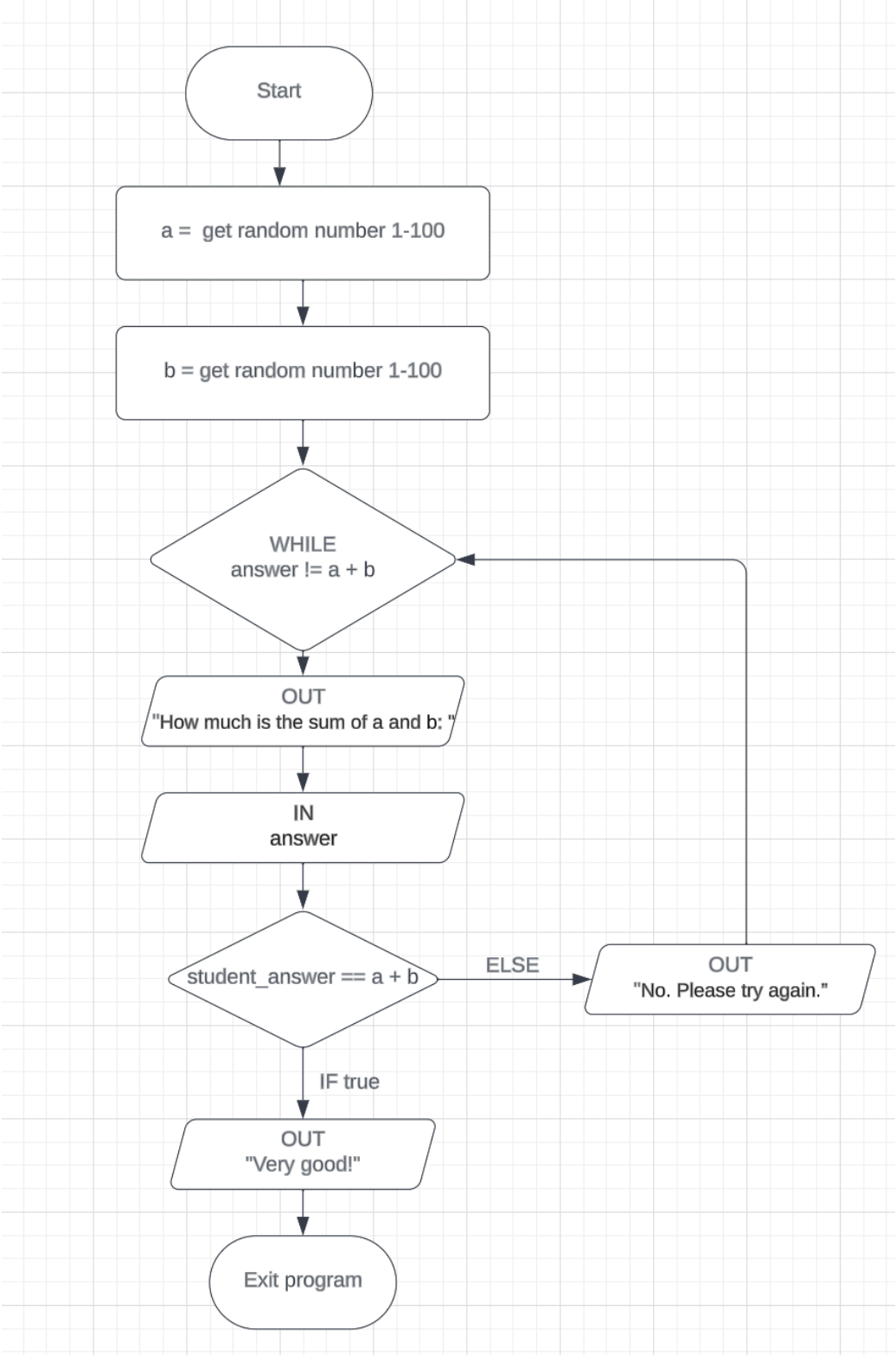
My algorithm starts with assigning two integer variables, a and b, which will be assigned two randomized numbers, using a randomizing function, like rand(). The program will then enter a while loop, that will run as long as the students answer is not the same as the sum of the two numbers. The student will be asked to add the two numbers together and input an answer. The program will compare the students answer to the sum of the two generated numbers. If the students answer and the sum of the two numbers are identical, the program prints "Very good!" and the exit. If the answer is wrong, the loop starts over and the student gets the same question again, until the answer is correct and the while condition evaluates to false.

Pseudocode

```
a = get a random number in the range 0—100
b = get a random number in the range 0—100

WHILE answer != a+b:
        OUT "How much is the sum of a and b: "
        IN answer
        IF answer == a+b
                OUT "Very good!"
                Exit program
        ELSE
                OUT "No. Please try again"
        END WHILE
END PROGRAM
```

Flowchart



Test cases

Input both right and wrong answers and check that the program makes the correct calculations and behave as expected.

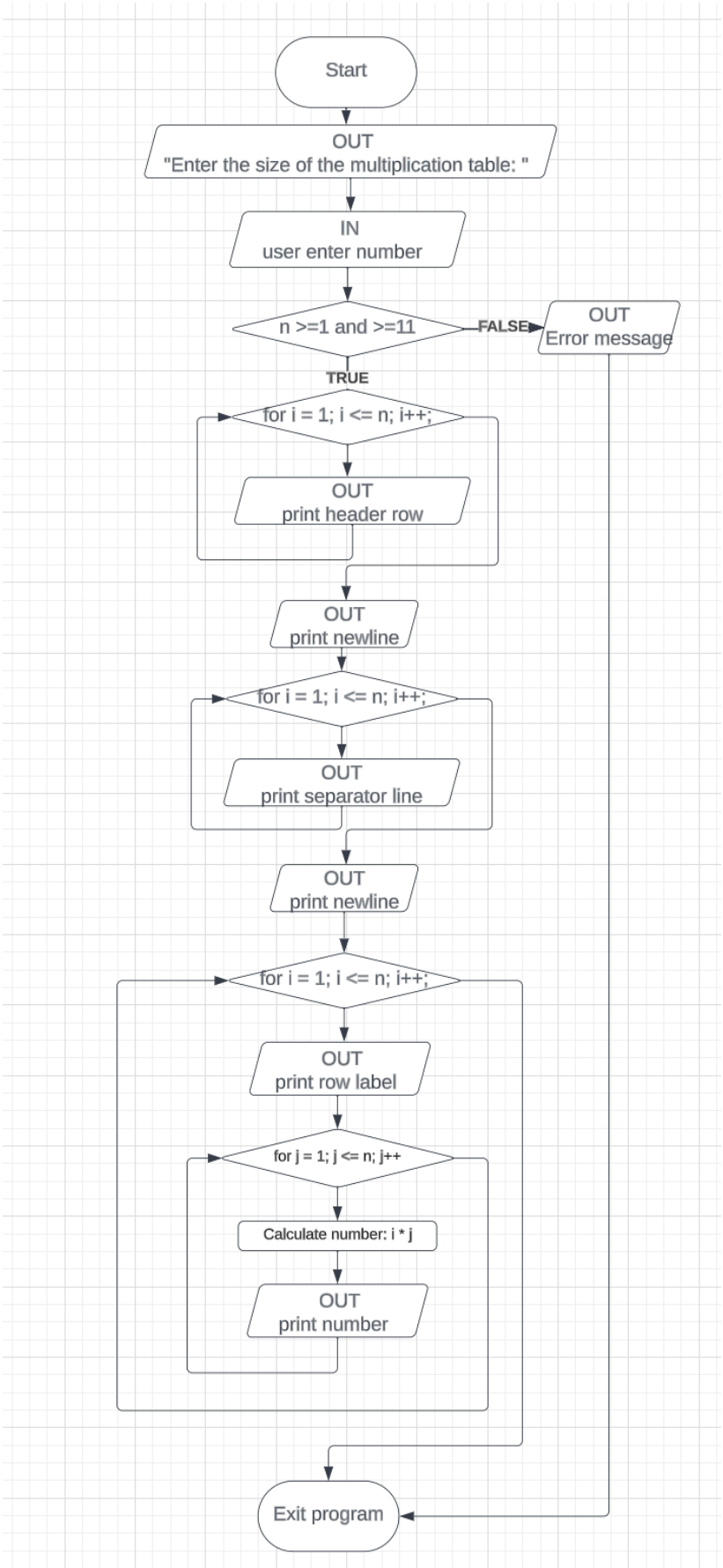
Task 2 - Multiplication Table

My algorithm starts by printing a statement asking the user to input a number. If the number is less than one or larger than 11, the program will print an error message and exit. If the number is in the correct range, it will start with printing the header row. This is done using a for loop, that will run n times to print the numbers in the header. To get the formatting correct, it should start with printing " x |" before the loop, but since the subject said that we didn't have to consider how to produce the output format, I left that out of the flowchart. I also chose not to include how to print the columns with the right spacing and alignment. After the header is printed, it will print a newline and another for loop is used to print the separator line. To print the table row by row, I'm using a nested for loop. The outer loop prints the row label on the left side of the table and the inner loop calculates and prints each number. After each iteration of the inner loop it should also print a newline to continue on the next line, but I accidentally left that out of the flow chart. When all numbers are printed, the program exit.

Pseudocode

```
OUT "Enter the size of the multiplication table: "
IN number
IF number <1 or >=11
        OUT "Error message"
        Exit program
ELSE
        FOR i = 1; i <= n; i ++;
                OUT print header row
        END FOR
        OUT print newline
        FOR i = 1; i <= n; i ++;
                OUT print separator line
        END FOR
        OUT print newline
        FOR  i = 1; i <= n; i ++;
                OUT print row label
                FOR j = 1; j <= n; j++;
                        calculate number: i * j
                        OUT print number
        END FOR
        OUT print newline
END PROGRAM
```

Flowchart


Start
OUT
"Enter the size of the multiplication table: "
IN
user enter number
n >=1 and >=11
FALSE
OUT
Error message
TRUE
for i = 1; i <= n; i++;
OUT
print header row
OUT
print newline
for i = 1; i <= n; i++;
OUT
print separator line
OUT
print newline
for i = 1; i <= n; i++;
OUT
print row label
for j = 1; j <= n; j++
Calculate number: i * j
OUT
print number
Exit program

Test cases

Test with different number inputs and check that the calculations is correct. Make sure the formatting stays the same, even when it prints 3 digit numbers, as will be the case with multiplication table 10 and 11.

3. Implementation and Test

Task 1 - Computer Assisted Instruction (CAI)

In my implementation, I include the following header files and standard functions:
stdio.h: printf, scanf
stdlib.h: rand, srand
time.h: time

I declare three integer variables: a, b and answer. To get randomized values, I use srand(time(0)); to ensure that the random number generator is seeded with a different value each time the program is executed. The time function, when called with 0 as argument, returns the current time in seconds since the Unix epoch (January 1, 1970). Here it will pass that value to the srand function as a seeding value. As long as the current time has changed, it will provide a different seeding value to initialize the random number generator. Without seeding, it will produce the same sequence of numbers every time the program is run.

To get two random numbers in the range 0-100, I initialize a and b to rand() % 101. The answer variable is initialized to 0. I then use a while loop that runs until the condition (answer != a + b) is evaluated to false. I use printf() to display the question and scanf() to retrieve the answer, which I assign to the answer variable.

Inside the loop there's a nested if/else statement. If (answer = a + b) is true, printf is used to display "Very good!" the program exits by return(0). If not, printf is used to display "No. Please try again." and the loop starts over.

The program was tested by compiling and running it, while providing different inputs, both wrong and right answers, to check if it performed as expected.

Task 2 - Multiplication Table

In my implementation, I include the following header files and standard functions:
stdio.h: printf, scanf

I declare three integer variables: n, to store user input and i and j to use as loop iterators. i and j are initialized to 0. The program starts by using printf to display "Enter the size of the multiplication table (1 to 11): “. I then use scanf to retrieve the input. The program checks if the value is within the valid range. if (n <= 0 || n > 11), the following message is displayed using printf: "Invalid input. Please enter a value between 1 and 11.” and the program exits with return(1);

If the input is valid, the program will generate and print the multiplication table. All printing is done using printf. The table is printed with the following steps:

1. Printing the header of the table, starting with “ x |“
2. Printing column headers for the numbers from 1 to n, using a for loop. Format specifier for printf is set to “%4d”, an integer with a minimum field width of four characters. If the integer has less than four digits, spaces will be added to the left of the number to make it occupy a total of four characters. If the integer has four or more characters, it will be printed without leading spaces. In this case, no number will have more than three digits, so 4d is enough.
3. Printing “ |” to separate the header from the table data.
4. It enters a for loop that generates the rows of the multiplication table. For each value of i from 1 to n, it prints i as the left column.
5. It then enters a nested for loop that generates the columns for the current row. For each value of j from 1 to n, it calculates and prints the result of i * j. After finishing the

inner loop for a row, it moves to the next line to start a new row. The outer loop continues until all rows are printed.

The program exits by return(0). The output is a multiplication table with rows and columns, that displays the product of numbers from 1 to n.

The program was tested by compiling and running it, with different inputs from -1 to 12, to make sure it works in the correct range, handle inputs out of range and that the table format works with all numbers in the intended range.

4. Results and discussion

The hardest part was making the flowcharts in digital format, every program I tried was either not compatible with my computer or was extremely buggy. It took a very long time making them, which was very annoying since that wasn't really the main task. Besides that, I think it was a good exercise. Algorithms are of course essential in programming and learning to plan ahead before diving into the actual coding, really makes things easier later on.

In the second exercise I realized while testing, that I should probably have used an other loop, so that the program didn't exit if the input value is out of range, but since I wrote the algorithm that way, I chose to stick with that solution in my implementation.

5. References

The system manual for functions rand, srand, time, using command line arguments:
"man rand"
"man srand"
"man time"
"man printf"
"man scanf"

I didn't use any other external references, besides the lectures provided on Canvas. I used

VS Code to write and run the code.