Kristianstad University Sweden

DT555B Programmering i C

- **Lab 3**

Adis Veletanlic

13/11 - 2022

Table of Contents

1. Introduction

The objective of this lab was to practice top-down design to develop a solution to a complicated problem, implement solutions to sub-problems in functions, and to use arrays and strings. Another goal was to modularize the code into smaller modules, so it is easier to both read and work on the code.

This paper focuses on the task “Grade 5 Task Option 2 -- A Computer Aided Instruction Program”. The goal is to create a program which assists students in practicing their skills in basic arithmetic (addition and subtraction).

2. Design

The design patterns I implemented are solely of top-down design, starting with pseudocode and later translating it into a flowchart.

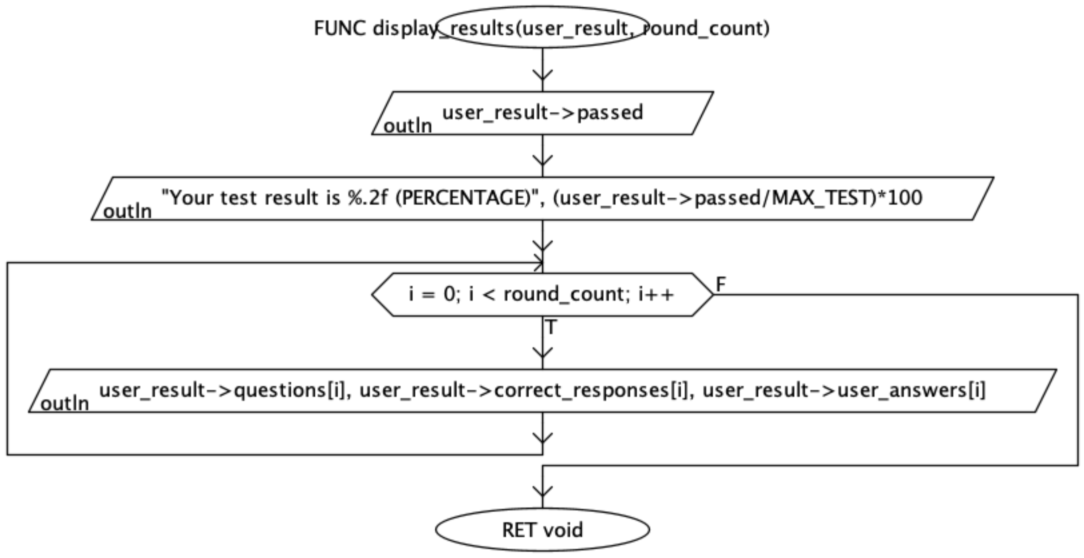
PSEUDOCODE:

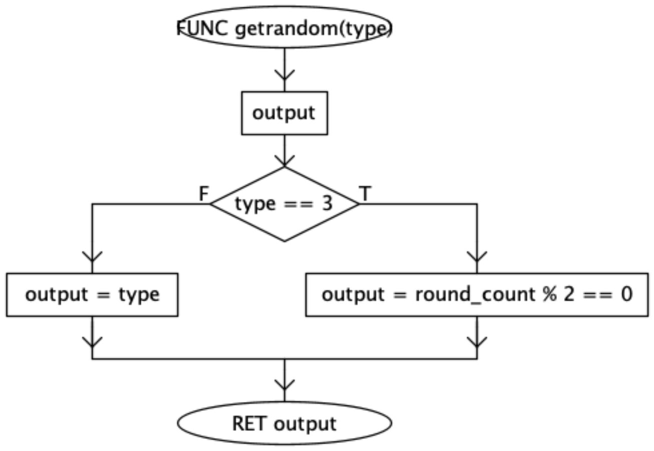
```
FUNCTION main
    Pass In: nothing
    CHAR_MAX = 20
    user_result = UserResultStruct instance
    input_buffer = [CHAR_MAX]
    IN input_buffer
    Call: selectprogram
    Pass Out: 0
ENDFUNCTION

FUNCTION selectprogram
    Pass In: &user_result, input_buffer
    PRACTICE = 1
    TEST = 2
    EXIT = 3
    running = 1
    DO
        OUT menu
        IN input_buffer
        CASE
            PRACTICE: Call: selectexec
            TEST: Call: selectexec
            EXIT: running = 0
        ENDCASE
    WHILE running
    Pass Out: nothing
ENDFUNCTION

FUNCTION selectexec
    Pass in: user_result, program_type, input_buffer
    OUT menu
    IN execution_type
    WHILE execution_type > 3 or exec_type < 0
        IN execution_type
    ENDWHILE
    Call: runprogram
    Pass Out: nothing
ENDFUNCTION

FUNCTION runprogram
    Pass In: user_result, execution_type, program_type, input_buffer
    initialize LIMIT
    IF program_type == PRACTICE
        LIMIT = MAX_PRACTICE
    ENDIF
    ELSE
        LIMIT = MAX_TEST
    ENDIF
    first = 0
    second = 0
```

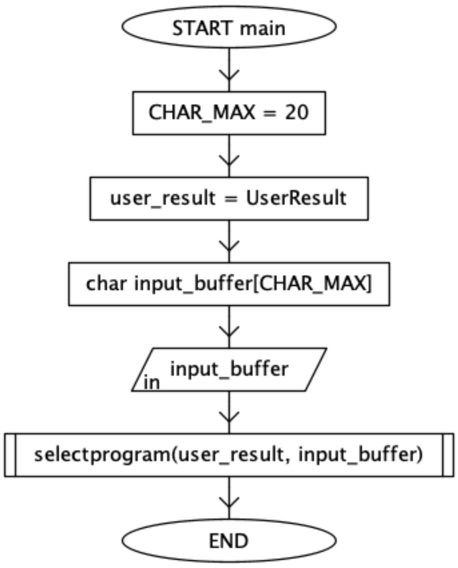
```
operand_count = 0
passed = 0
round_count = 0
initialize correct_index
initialize bad_index
OUT "Now, you will be given {LIMIT} questions to solve: \n"
initialize user_input
user_result.passed = 0
DO
    passed = 0
    first = Call: getrandom
    second = Call: getrandom
    correct_index = Call: getrandom
    OUT "Question {round_count + 1}: "

    CASE
        PRACTICE:
            DO
                bad_index = Call: getrandom
                IN passed
                IF !passed
                    OUT random bad response
                ENDIF
                IF exec_type == 3
                    operand_count = operand_count + 1
                ENDIF
            WHILE !passed
        TEST:
            IN passed
    ENDCASE
    IF program_type == PRACTICE
        OUT random positive response
    ENDIF
    Call: setstruct
```

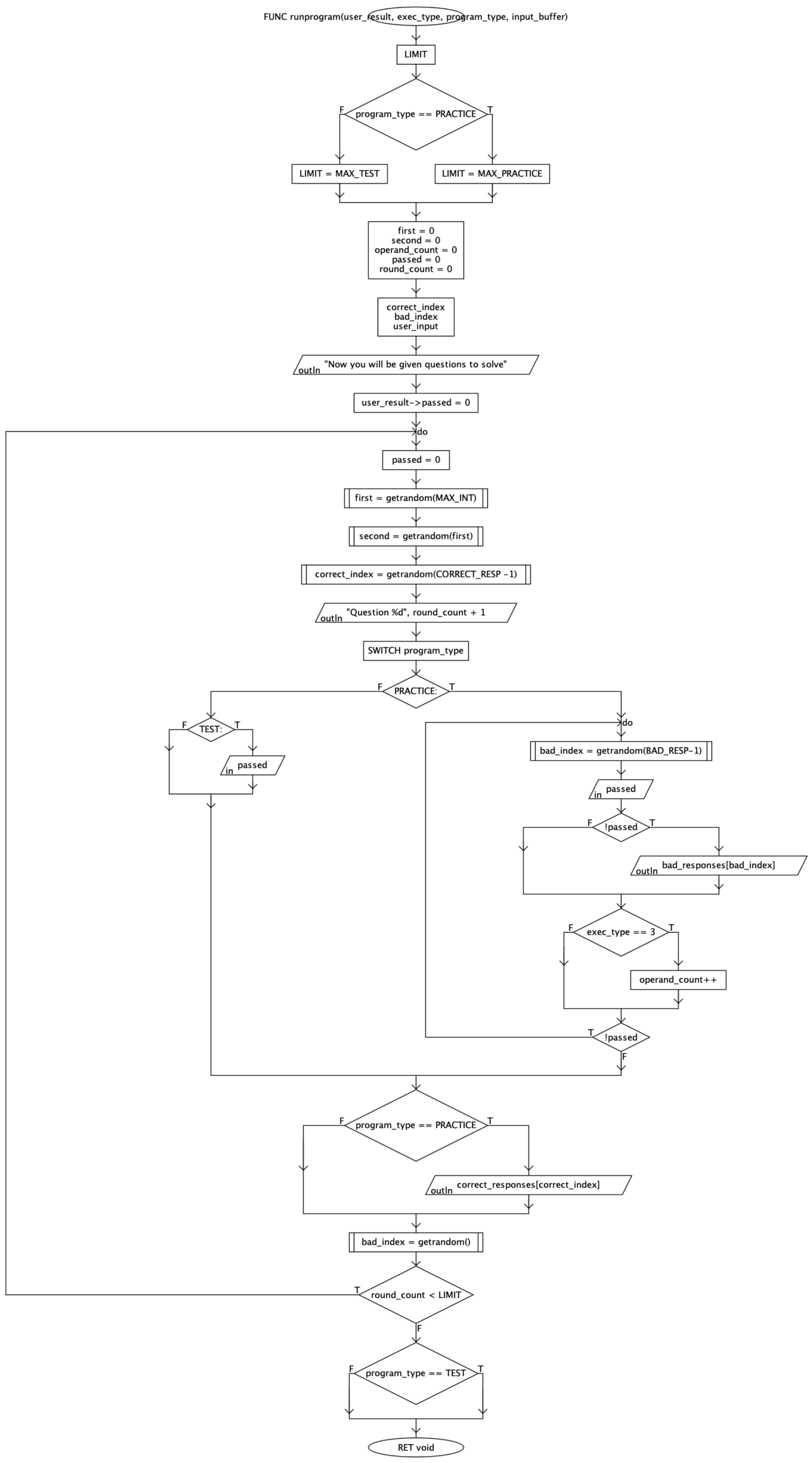
```
        round_count = round_count + 1
    WHILE round_count < LIMIT
    IF program_type == TEST
        Call: display_results
    ENDIF
    Pass Out: nothing
ENDFUNCTION

FUNCTION setstruct
    Pass In: user_result, round_count, first, second, passed, user_input, exec_type
    user_result.questions[round_count] = current question as string
    user_result.correct_responses[round_count] = correct response
    user_result.user_answers[round_count] = user_input
    IF passed
        user_result.passed = user_result.passed + 1
    ENDIF
    Pass Out: nothing
ENDFUNCTION

FUNCTION display_results
    Pass In: user_result, round_count, max
    OUT user_result.passed
    OUT user_result->passed/MAX_TEST*100
    FOR i = 0 To round_count
        OUT user_result->questions[i], user_result->correct_responses[i], user_result-
>user_answers[i]
    ENDFOR
    Pass Out: nothing
ENDFUNCTION

FUNCTION getrandom
    Pass In: max
    Pass Out: random number from 0 to max
ENDFUNCTION
```

FLOWCHARTS:

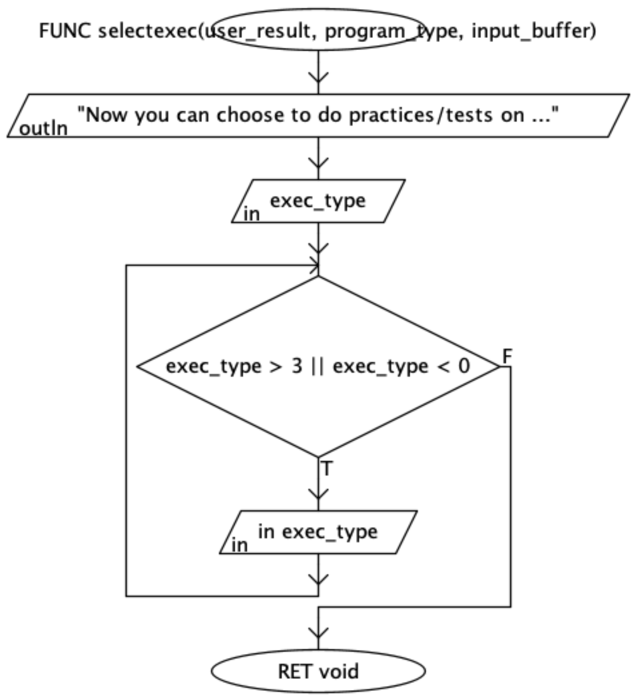

FUNC display_results(user_result, round_count)
outln user_result->passed
outln "Your test result is %.2f (PERCENTAGE)", (user_result->passed/MAX_TEST)*100
i = 0; i < round_count; i++
F
T
outln user_result->questions[i], user_result->correct_responses[i], user_result->user_answers[i]
RET void

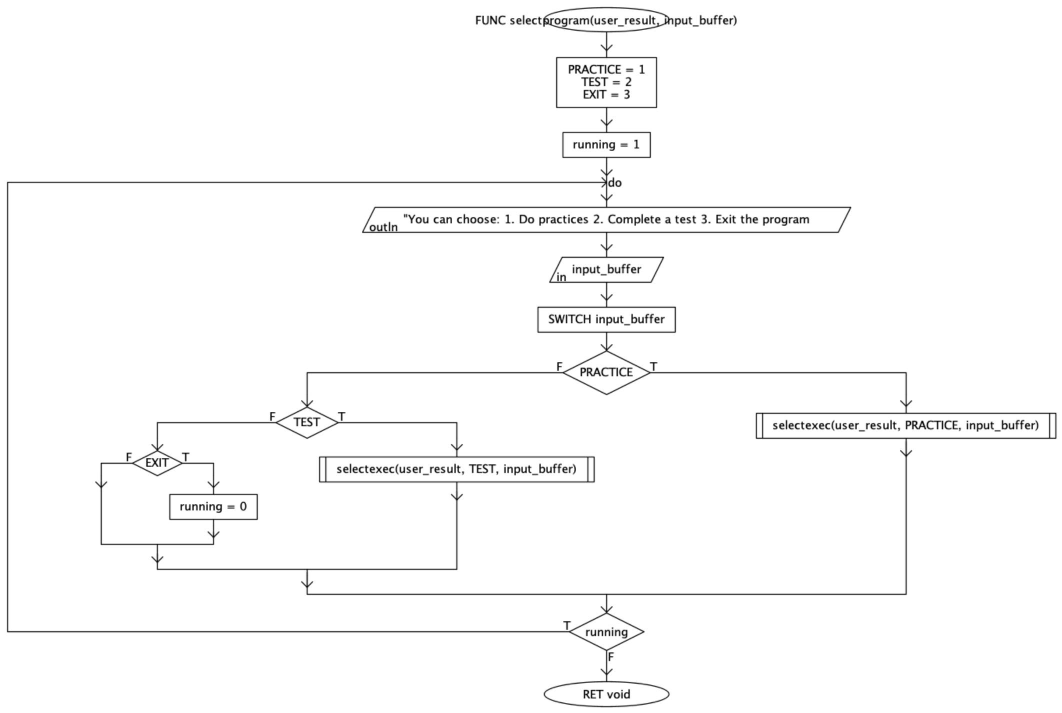

FUNC getrandom(type)
output
type == 3
F
T
output = type
output = round_count % 2 == 0
RET output

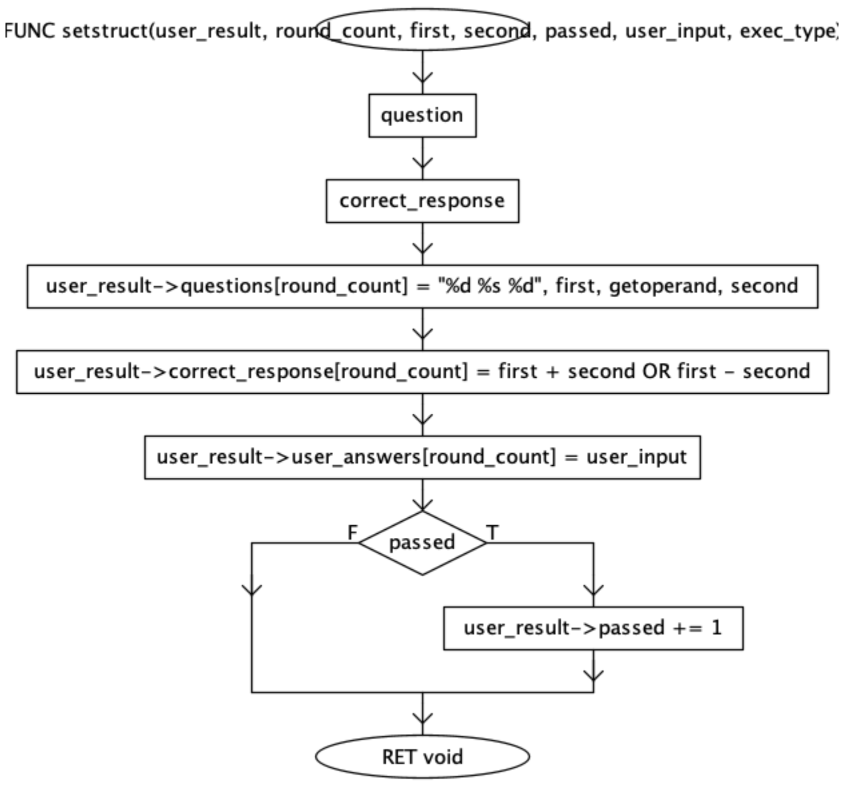

START main
CHAR_MAX = 20
user_result = UserResult
char input_buffer[CHAR_MAX]
in
input_buffer
selectprogram(user_result, input_buffer)
END



FUNC runprogram(user_result, exec_type, program_type, input_buffer)
LIMIT
program_type == PRACTICE
F
T
LIMIT = MAX_TEST
LIMIT = MAX_PRACTICE
first = 0
second = 0
operand_count = 0
passed = 0
round_count = 0
correct_index
bad_index
user_input
outln "Now you will be given questions to solve"
user_result->passed = 0
do
passed = 0
first = getrandom(MAX_INT)
second = getrandom(first)
correct_index = getrandom(CORRECT_RESP -1)
outln "Question %d", round_count + 1
SWITCH program_type
PRACTICE:
F
T
TEST:
F
T
in passed
do
bad_index = getrandom(BAD_RESP-1)
in passed
!passed
F
T
outln bad_responses[bad_index]
exec_type == 3
F
T
operand_count++
!passed
T
F
program_type == PRACTICE
F
T
outln correct_responses[correct_index]
bad_index = getrandom()
round_count < LIMIT
T
F
program_type == TEST
F
T
RET void


FUNC selectexec(user_result, program_type, input_buffer)
outln "Now you can choose to do practices/tests on ..."
in exec_type
exec_type > 3 || exec_type < 0
F
T
in in exec_type
RET void


FUNC selectprogram(user_result, input_buffer)
PRACTICE = 1
TEST = 2
EXIT = 3
running = 1
do
outln "You can choose: 1. Do practices 2. Complete a test 3. Exit the program
in input_buffer
SWITCH input_buffer
PRACTICE
F
T
selectexec(user_result, PRACTICE, input_buffer)
TEST
F
T
selectexec(user_result, TEST, input_buffer)
EXIT
F
T
running = 0
running
T
F
RET void


FUNC setstruct(user_result, round_count, first, second, passed, user_input, exec_type)
question
correct_response
user_result->questions[round_count] = "%d %s %d", first, getoperand, second
user_result->correct_response[round_count] = first + second OR first - second
user_result->user_answers[round_count] = user_input
passed
F
T
user_result->passed += 1
RET void

3. Implementation and Test

Task "Grade 5-- Computer Assisted Instruction (CAI)"

This task was an extension of the CAI program from the first lab in the course. The goal is to create an interface that allows students to practice their knowledge in basic arithmetic. For this extended version, a more advanced menu is implemented along with the ability to *only* test and to *only* practice. Both the practice and test options present the student with a menu where they can choose between which questions will appear, which kind of operator will be used between the numbers. The questions can also be mixed, mixing the operands in-between questions.

For this version of the CAI program, I decided to make use of header files and a nicer folder structure, allowing me to modularize the code into blocks and files, making it easier to read and debug with GDB if I needed to.

To test the program, simply build and execute the binary file compiled. A menu will be presented that tells you which options you have.

4. Results and discussion

There are four major takeaways for me regarding this lab assignment. Firstly, I learned how to implement header files in my code and continue to improve how I modularize my code.
Secondly, I gained new insight on how arrays are used in the C language and how they are passed between functions, and how they are to be declared.
Thirdly, making use of different methods I made myself to ensure correct user input with fgets, strtol and pointers.
Lastly, I realized how to properly use structs. It was not necessary for this task, although I decided that it could be a good opportunity to use structs in this case to make the program easier for myself to develop.

The development time was quite large, since I made sure that my program is solid, somewhat easy to read for the instructor of the course, and structured in a coherent manner, with regards to folders, files, and naming conventions.

5. References

- -