Tutorial 3: Introduction to C Part II

Faculty of Engineering and Applied Science

SOFE 3950U : Operating Systems | CRN: 74171

Due: February 5th, 2024

Group 8

Daniel Amasowomwan [100787640]
daniel.amasowomwan@ontariotechu.net

Stanley Watemi [100648403]
stanley.watemi@ontariotechu.net

Fayomi Toyin [100765921]
oluwatoyin.fayomi@ontariotechu.net

Conceptual Questions

1. List each of the modes for the fopen function to the perform the following operations: read, write, read and write, append to a file.
    - Read Mode(“r”)- opens the file for reading
    - Write Mode (“w”)- creates an empty file for writing
    - Append mode(“a”)- Appends data to a file
    - Read and Write mode (“r+”)-Opens a file to update reading and writing.
    - Write and Read mode (“w+”)- Creates an empty file for both reading and writing
    - Append and Read mode (“a+”)-Opens a file for reading and appending.

2. Does dynamic memory use the stack or heap? What is the difference between the stack and heap?

a. Dynamic memory uses heap or is allocated from heap using standard library.

b. Difference between the stack and heap
    - Stack provides static memory allocation that is stores temporary variables while heap provides dynamic allocation
    - Local variables can be accessed using a stack while by default, heap is used to access global variables
    - Elements in a stack are stored one at a time (linear) while in heap elements are stored randomly.
    - Access time is faster in a stack while it is slower in heap
    - Stacks have a fixed size while the size of a heap may vary.

3. Explain what a pointer is, and provide examples (in C code) of how to change the address that a pointer points to and how to access the data the pointer points to.
    - The Pointer in C, is a variable that stores the address of another variable. A pointer can also be used to refer to another pointer function.

4. Read the documentation on the malloc and free functions and explain briefly how to use malloc.
    - The function malloc() is used to allocate the requested size of bytes and it returns a pointer to the first byte of allocated memory. It returns a null pointer, if it fails.

      pointer_name = (cast-type*) malloc(size);

5. What is the difference between malloc and calloc?
    - Malloc function creates a block of memory at a fixed time while calloc function assigns a specific number of blocks of memory to a single variable.
    - Calloc function requires two arguments while Malloc requires just one.
    - Calloc indicates contiguous memory allocation and it adds some memory overhead while malloc indicates memory allocation and it does not add any extra memory overhead.

Application Questions

All of your programs for this activity can be completed using the template provided, where you fill in the remaining content. A makefile is not necessary, to compile your programs use the following command in the terminal. **If you do not have clang then replace clang with gcc.**

```
clang -Wall -Wextra -std=c99 <program name>.c -o <program name>
```

Example:

```
clang -Wall -Wextra -std=c99 question1.c -o question1
```

You can then execute and test your program by running it with the following command.

```
./<program name>
```

Example:

```
./question1
```

Template

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{

}
```

1. Create a program that does the following
    - Prompts the user for their **first name**, **age**, and **height** (hint use a character array for strings).
    - Prints back to the console, their first name, age, and height
    - You will need to review the **scanf** documentation to complete this

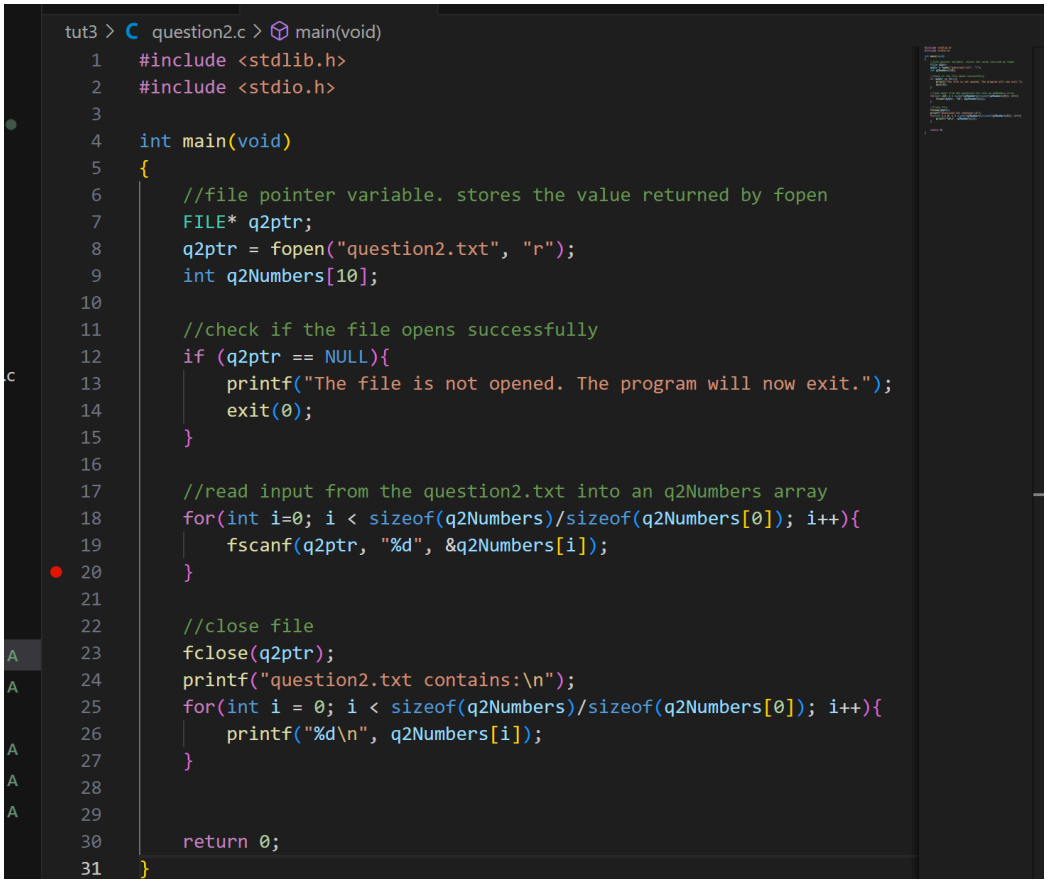
question1.c > ...

```c
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    int age;
    char firstName[32];
    float height;

    printf("Input your first name: ");
    fgets(firstName, sizeof(firstName), stdin);

    printf("Input your age: ");
    scanf("%d", &age);

    printf("Input your height: ");
    scanf("%f", &height);

    printf("Your name is %sYour age is %d\nYour height is %.
    1f\n", firstName, age, height);
}

```

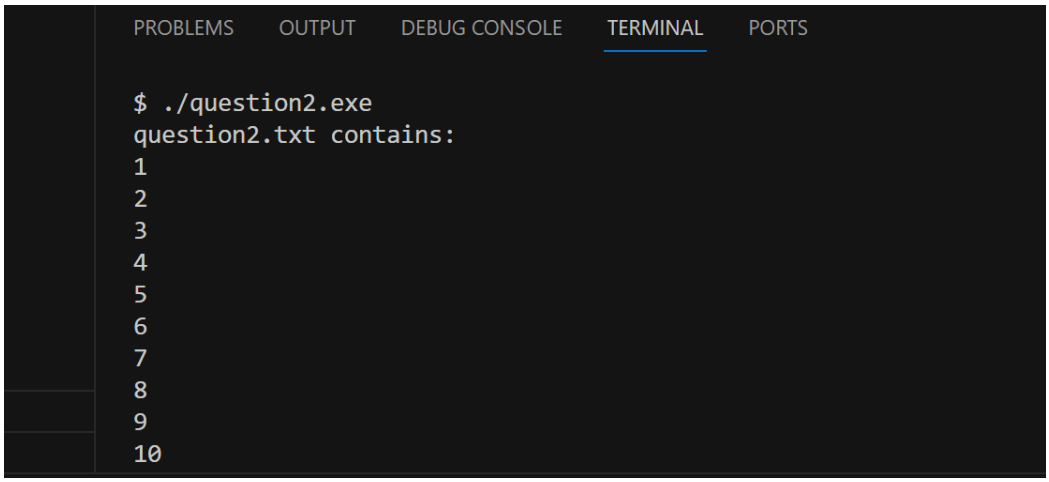
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash

```
danielamas@Linux22:~/school/opsystems/tutorials/tut3$ gcc question1.c -o que
stion1
danielamas@Linux22:~/school/opsystems/tutorials/tut3$ ./question1
Input your first name: Daniel
Input your age: 20
Input your height: 6.1
Your name is Daniel
Your age is 20
Your height is 6.1
danielamas@Linux22:~/school/opsystems/tutorials/tut3$
```

2. Create a program that does the following
    - Reads the ten integers from the included file **question2.txt**
    - Stores each integer **read from the file** in an array
    - Prints the contents of the array to the terminal
    - You will need to review the **fopen** and **fscanf** documentation

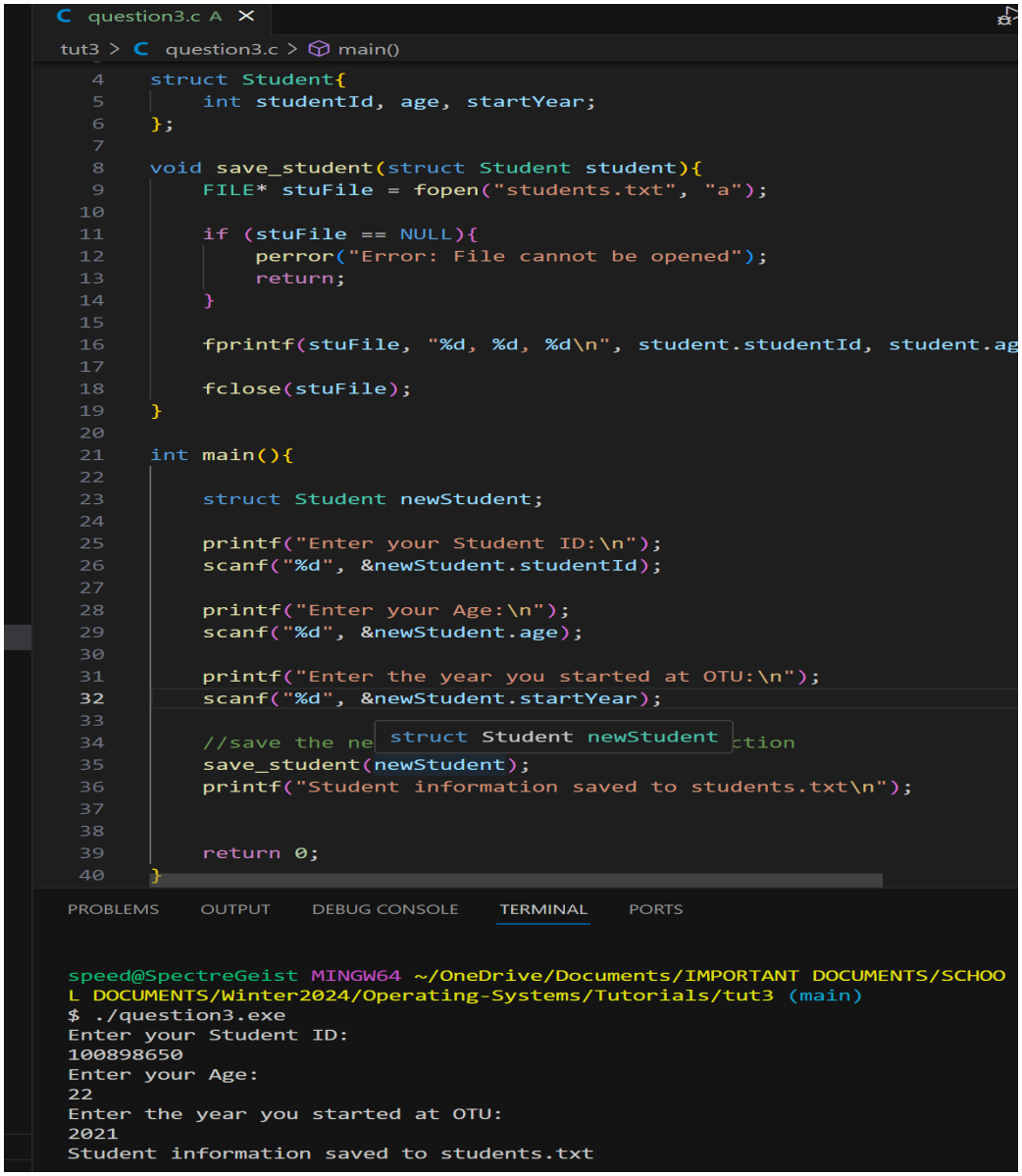


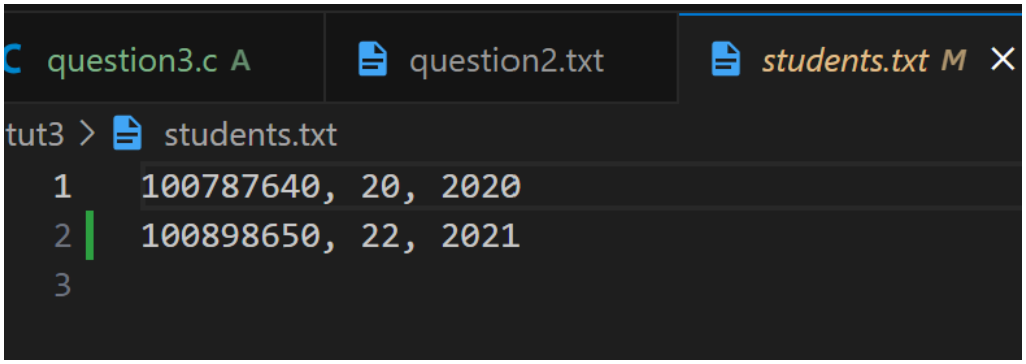
```c
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    //file pointer variable. stores the value returned by fopen
    FILE* q2ptr;
    q2ptr = fopen("question2.txt", "r");
    int q2Numbers[10];

    //check if the file opens successfully
    if (q2ptr == NULL){
        printf("The file is not opened. The program will now exit.");
        exit(0);
    }

    //read input from the question2.txt into an q2Numbers array
    for(int i=0; i < sizeof(q2Numbers)/sizeof(q2Numbers[0]); i++){
        fscanf(q2ptr, "%d", &q2Numbers[i]);
    }

    //close file
    fclose(q2ptr);
    printf("question2.txt contains:\n");
    for(int i = 0; i < sizeof(q2Numbers)/sizeof(q2Numbers[0]); i++){
        printf("%d\n", q2Numbers[i]);
    }


    return 0;
}
```



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

$ ./question2.exe
question2.txt contains:
1
2
3
4
5
6
7
8
9
10
```

3. Create a program that does the following
    - Create a struct called **student** containing their **student id**, **age**, and the year they started at UOIT.
    - Create a function called **save_student** which does the following:
        - Takes as its argument the **student** struct and returns **void**
        - Opens a file called **students.txt** in **append** mode
        - Saves the student id, age, and year from the **students** struct to the file on one line delimited by commas (e.g. **100123456,19,2014**).
    - In the console prompt the user for their **student id**, **age**, and the **year** they start at UOIT.
    - Store the values entered by the user in the **student** struct.
    - Call the function **save_student** with the student struct to save the data to the **students.txt** file.




question3.c A
question2.txt
students.txt M
tut3 > students.txt
1 100787640, 20, 2020
2 100898650, 22, 2021
3

4. Create a program that does the following
    - Creates three pointers, a character pointer **professor**, and two integer pointers **student_ids**, **grades**
    - Using dynamic memory, use calloc to allocate 256 characters for the professor pointer
    - Prompts the professor for their **name**, and the **number of students** to mark.
    - Stores the professor's name using the **professor** pointer and in an integer the number of students to mark.
    - Using dynamic memory, use **malloc** to allocate memory for **student_ids** and **grades** to hold the number of students the professor needs to mark.
    - The program does not need to do anything else, ensure that you **free** your memory before terminating.
    - You will need to review the **malloc**, **calloc**, and **sizeof** documentation.

```
it3 > question4.c > main()
#include <stdio.h>
#include <stdlib.h>

int main(){
#include <stdio.h>
    // create pointers
    int num_students;
    char *professor;
    int *student_ids, *grades;

    //allocate 256 characters to prof pointer using calloc
    professor = (char*)calloc(256, sizeof(char));

    if (professor == NULL){
        perror("Error: Could not allocate memory for professor");
        return 1;
    }

    //Prompt professor for name and number of students to mark
    printf("Professor Name:\n");
    fgets(professor, 256, stdin);
    printf("Enter number of students to mark: ");
    scanf("%d", &num_students);

    //allocate memory for student ids and grades
    student_ids = (int*)malloc(num_students * sizeof(int));
    grades = (int*)malloc(num_students * sizeof(int));

    // Check if malloc was successful
    if (student_ids == NULL || grades == NULL) {
        perror("Error allocating memory for student_ids or grades");
        free(professor);  // Free the memory allocated for professor
        return 1;
    }

```

```
    //free all allocated memory
    free(professor);
    free(student_ids);
    free(grades);

    return 0;
}
```

```
$ ./question4.exe
Professor Name:
Masoud
Enter number of students to mark: 9
```

5. Building upon the previous questions you will create a marking system for professors at UOIT.
    - Structs can be used the same as any other data type in C, instead of having two arrays for the grades and student ids create a struct called **grade** that contains two integers: **student_id** and **mark**.
    - Create a function **grade_students** which takes the following arguments: a **pointer** to the **grade** struct called **grades**, and an integer **num_students**. The function returns **void** and does the following:
        - Opens the file **grades.txt** in **write** mode
        - Using the **num_students** parameter iterates through all of the grade structs pointed to by the **grades** parameter (remember arrays are pointers, you can treat pointers like arrays).
        - **For each** grade structure **adds** the mark member of the struct to a variable called sum that holds the **sum** of all student's grades.
        - **For each** grade structure **write** to the file **grades.txt** the **student id** and the **mark** on a single line.
        - After adding every student's **mark** to the **sum** variable, calculate the **average** (mean) and **standard deviation**, you will need to use `<math.h>` don't forget when you compile to add **-lm**
        - **Write** to the file **grades.txt** the **average** and **standard deviation** that you calculated.
    - Create two pointers, a character pointer **professor**, and a pointer for the **grade** struct you created and call it **grades**, it will hold an array of grade structures.
    - Using dynamic memory, use **calloc** to allocate 256 characters for the **professor** pointer.
    - Prompts the professor for their **name**, and the **number of students** to mark.
    - Store the professor's name using the **professo**r pointer and in an integer **num_students** the number of students to mark.
    - Using dynamic memory, use **malloc** to allocate memory for the **grades** pointer to hold the number of students the professor needs to mark.
    - For each student to mark (**num_students**) **prompt the professor** for the student id and the mark and store it in the **grade** struct in **grades** (you can use grades just like an array).
    - After getting all of the student IDs and marks from the professor call the **grade_students** function to grade the students, calculate grade statistics, and store all the results to **grades.txt**
    - Don't forget to **free** all of your dynamic memory

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

struct Grade {
    int student_id;
    int mark;
};

void grade_students (struct Grade *grades, int num_students){
    //open or create grades.txt
    FILE * filePointer = fopen("grades.txt", "w");
    if(filePointer == NULL){//check if file was opened
        perror("Error: could not open file");

    }

    int sum = 0;
    double avg, std_deviation = 0.0;

    //iterate through Grade struct
    for(int i = 0; i < num_students; i++){
        //write student id and mark to the file
        sum += grades[i].mark;

        fprintf(filePointer, "%d %d\n", grades[i].student_id, grades[i].mark);
    }

    //calculate average
    avg = (double)sum/num_students;

    //calculate std_deviation
    for (int i = 0; i< num_students; i++){
        std_deviation += pow(grades[i].mark - avg, 2);
    }
```

```c
    std_deviation /= num_students;
    std_deviation = sqrt(std_deviation);

    // Write average and standard deviation to the file
    fprintf(filePointer, "Average: %lf\nStandard Deviation: %lf\n", avg, std_deviation);

    // Close the file
    fclose(filePointer);

}
```

```c
int main(){
    //create char pointer for professor's name and struct pointer for the grade struct
    char *professor = (char *)calloc(256, sizeof(char));
    struct Grade *grades;

    //Prompt professor for name and number of students to mark
    printf("Enter Professor Name:\n");
    fgets(professor, 256, stdin);
    int num_students;
    printf("Enter number of students to mark: ");
    scanf("%d", &num_students);

    //Allocate memory for the grades
    grades = (struct Grade *)malloc(num_students * sizeof(struct Grade));

    // Check if memory allocation was successful
    if (grades == NULL) {
        printf("Memory allocation failed.\n");
        free(professor);
        return 1; // Exit with an error code
    }

    // Prompt professor for student id and mark for each student
    for (int i = 0; i < num_students; i++) {
        printf("Enter student %d id: ", i + 1);
        scanf("%d", &grades[i].student_id);

        printf("Enter struct Grade *grades + 1);
        scanf("%d", &grades[i].mark);
    }

    // Call the function to grade students and calculate statistics
    grade_students(grades, num_students);

    // Free allocated memory before terminating
    free(professor);
    free(grades);

    return 0;
}
```

```
Enter student 1 mark: 100
Enter student 2 id: 100898765
Enter student 2 mark: 65
Enter student 3 id: 100998764
Enter student 3 mark: 77
Enter student 4 id: 100676340
Enter student 4 mark: 10
Enter student 5 id: 100765434
Enter student 5 mark: 55
Enter student 6 id: 100666421
Enter student 6 mark: 89
Enter student 7 id: 100898774
Enter student 7 mark: 95
Enter student 8 id: 100656449
```

question5.c U × | grades.txt U × | ques

tut3 > grades.txt

```
1   100787640 100
2   100898765 65
3   100998764 77
4   100676340 10
5   100765434 55
6   100666421 89
7   100898774 95
8   100656449 57
9   100887465 64
10  100760636 73
11  Average: 68.500000
12  Standard Deviation: 24.446881
13
```