



ASSIGNMENT BRIEF

HTU Course No:

40303130

HTU Course Name:

Fundamentals of Computing

BTEC UNIT No:

BTEC UNIT Name:

# Version: 1

1

**Assignment Brief**

|  |  |
| --- | --- |
| Student Name/ID Number/Section |  |
| HTU Course Number and Title | **40303130: Fundamentals of Computing** |
| BTEC Unit Number and Title | **Not Available** |
| Academic Year | Fall 2022/2023 |
| Assignment Author | Course Instructors |
| Course Tutor | Eng. Ashraf Al-Smadi Eng. Bassam Al-Kasasbeh Dr. Hala Hamadah  Eng. Malek Allouzi Dr. Mohammad Yahya Eng. Samer Suleiman  Eng. Umar Alkafaween |
| Assignment Title | Prime Numbers |
| Assignment Ref No | **Assignment 1** |
| Issue Date | 26th December 2022 |
| Formative Assessment dates | Milestone 1: 2nd January 2023 – 5th January 2023  Milestone 2: 15th January 2023 – 19th January 2023 |
| Submission Date | Milestone 1: 7th January 2023  Milestone 2: 26th January 2023 |
| IV Name & Date |  |

# Submission and Assessment Format

This assignment is a project design. You must complete it and submit it according to the following guidelines (Failing to follow these guidelines may result in an ‘Unclassified’ grade):

# What to submit

You are required to submit 3 files for this project called as follows:

**code.c**: Contains the code you write to answer tasks: 1, 2, 3, 5, 6, and 7.

**write\_up.pdf**: Contains the answers for tasks 4 and 8 and the **student declaration form** after you sign it (you can find it attached at the end of these instructions).

**make\_up.pdf**: Contains the answers for any make-up questions you choose to solve to make-up for criteria you did not achieve in the midterm.

For each milestone, you need to submit the 3 files above, and they should contain the answers as follows:

|  |  |
| --- | --- |
| Milestone 1 | Answers to tasks 1, 2, 3, and 4, in addition to any make-up tasks you solve. |
| Milestone 2 | Answers to **ALL** tasks – including milestone 1’s tasks, in addition to make-up tasks you solve. |

Notes about your submission:

* Only soft-copy submissions are allowed. You are required to upload your submission files to the university’s eLearning system through (https://elearning.htu.edu.jo) within the submission dates and times stated above. **NO SUBMISSIONS by EMAIL or TEAMS and NO LATE SUBMISSIONS WILL BE ACCEPTED.** Failing to follow this guideline may result in an ‘Unclassified’ grade.
* If you commit any kind of plagiarism, HTU policies and regulations will be applied.
* This is a strictly individual assignment and no collaboration amongst students is allowed. Working with your colleagues is not considered teamwork, but rather plagiarism.



**In-class Oral Discussion:**

* In-class assessment will be one to one online or in-person oral discussion between you and your instructor after the 2nd milestone. It includes designing, developing, debugging, analyzing, and evaluating the code and algorithms developed in this project.
* The attendance of the oral assessment is mandatory in the date and time determined by your instructor. Be ready to **open your camera** throughout the assessment in the case of an online discussion.
* You should be prepared to answer any questions about the assignment’s tasks during the oral assessment. Any questions not answered in the oral assessment with the required level of detail means that you might lose the criteria related to it, even if you have already completed the task correctly in your submission.
* You must sign the witness form that your instructor will fill up during the discussion to complete the oral assessment process.



|  |
| --- |
| **Unit Learning Outcomes** |
| **LO1** Discuss the basic concepts of computer hardware, software, and operating systems, with the basic relation between them, taking C commands as an example.  **LO2** Implement a full program in C with identification of the three kinds of control structures: sequence, selection, and repetition, understanding the capabilities of implementing C codes that capable to deal with functions.  **LO3** Understanding the basic concepts of pointers in C with implementation of arrays and dealing with files. In addition to the role of C programming as a powerful tool to communicate between devices and manipulate their data. |
| **Assignment Brief and Guidance** |
| You are given 8 tasks that you must complete over 2 milestones. You will find the description of all the tasks in this section.  **Assignment Files**  This assignment contains 7 files that you will be using:   * **Instructions.pdf**, This file, it contains the instructions of the assignment. * **Makeup.pdf**, which contains questions you need to solve some of to make up for any criteria you lost in the midterm. * **code.c**, which contains the starting code for the assignment. You will need to implement functions in this file to complete some of the tasks. * **image.in**, which contains values for pixels of an image. * **original\_image.jpg**, the same image in image.in but can be displayed in your computer image viewer so you could check what your code’s output should look like. * **foc\_fa23.h**, the header file of the library that allows us to draw pictures from 2D arrays. This file needs to be included inside code.c. * **foc\_fa23.o**, the object file of the library that allows us to draw pictures from 2D arrays.   This file needs to be compiled with code.c. |

|  |
| --- |
| **Getting Started**  Start by looking at code.c, and image.in, then reading all the instructions before you start coding.  **code.c**  This C file contains the starting code. You will update this file to implement 7 essential functions that will be completely explained in the tasks below, but here is a quick summary:  **mainMenu**: Prints the main menu and invokes all the other essential functions.  **isPrime**: Checks whether a number is prime or not.  **primesInRange**: Prints prime numbers from 1 to a number specified by the user.  **sieve**: Implements an algorithm called Sieve of Eratosthenes. The algorithm is fully explained below.  **primesInRangeSieve**: Prints the prime numbers that were generated by the sieve function from 1 to a number specified by the user.  **flipImageVertically**: Flips an image vertically.  **rotateImageClockwise**: Rotates an image clockwise. |

The file also contains the following helper functions, these are functions that you **need** to implement in order to use them in other functions and make your job easier. **It is recommended that you start with implementing these functions before the essential functions they are related to.**

**extractImageRows**: Extracts the number of rows of an image from the image file. Check the description of the image file below to understand how this function should work.

**extractImageColumns**: Extracts the number of columns of an image from the image file. Check the description of the image file below to understand how this function should work.

**readImageFromFile**: Reads an image from a file and stores it in the passed array.

**drawImageFromFile**: Reads an image from a file and draws it using the showArray function from the foc\_fa33.h header.

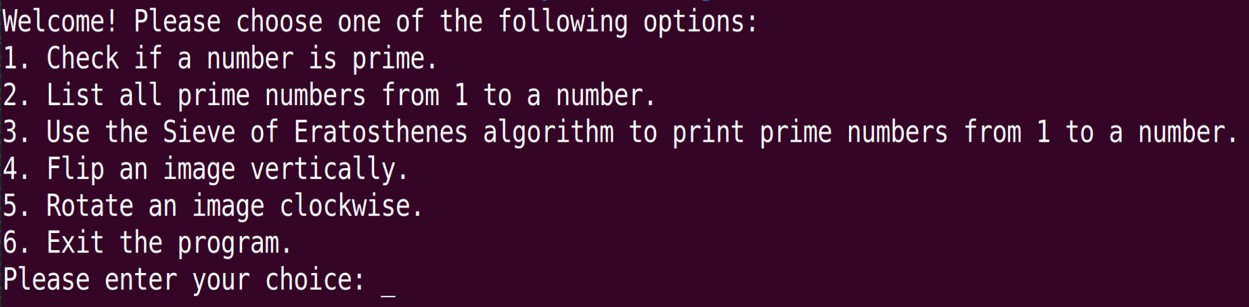
**storeImageInFile**: Writes an image to a file.

Note 1: All the functions have comments above them in code.c to remind you of what they do. Note 2: **Do NOT** modify the names, return types, or arguments of the provided functions.

Note 3: You are allowed to create additional functions if you need to, but you **HAVE TO**

implement all the provided functions.

The file also contains one global variable that is defined at the beginning of the file. Global variables are variables that have a “global scope”, and therefore can be used anywhere in your program. They are usually used to store values will be accessed from different parts of the code. You will not need to define any other global variables to solve this project.



**image.in**

This file contains the dimensions and the pixels of an image. It has a very specific format.

It starts with one line that contains 2 integers separated by a space. The integers represent the dimensions of the image. The first integer is the length (rows) of the image, and the second integer is the width (columns).

After the first line, there will be several lines equal to the length of the image; each line will contain a number of space-separated integers that is equal to the width of the image. Each number is between 0 and 255 and it represents a grayscale image’s pixel.

For example, if the image starts with “100 200”, then it will have next 100 lines, and each line of the 100 will have 200 integers in it.

**The Tasks**

# Task One

Implement the function int mainMenu().

This function is called from the main function. It should print out the main menu of the program, it looks like figure 1 below.

Figure 1

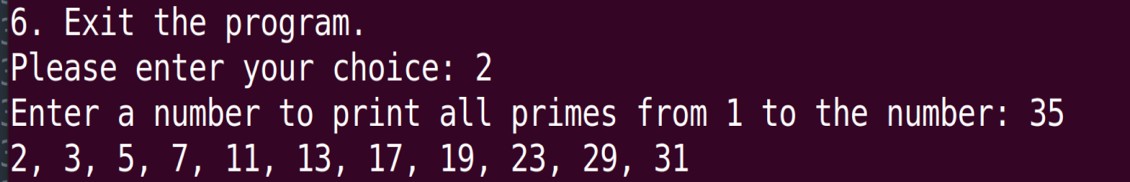
It then accepts the user’s choice, which is an integer from 1 to 6. Depending on the choice of the user, it should call the appropriate function(s) and pass any values it needs to it. The following choices should run the following functions **and do the following actions before and after running the functions**:



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
|  | Choice | Function To Call | Before Calling It (**do this in** | After Calling It (**do** |  |
|  |  | **mainMenu**) | **this in mainMenu**) |
| 1 | int isPrime(int number) | Ask the user to input a | If the number was |
|  |  | number and pass it to the | prime, print “X is |
|  |  | function as an argument. | prime”, if the number |
|  |  |  | was not prime |
|  |  |  | (composite), print “X |
|  |  |  | is not prime”. |
|  |  |  | Replace X with the |
|  |  |  | number when you |
|  |  |  | print it. |
|  |  |  | e.g., “5 is prime”. |
| 2 | void primesInRange(int  rangeEnd) | Ask the user to input the  range to print primes until | Do nothing |
|  |  | and pass it to the function as |  |
|  |  | an argument. |  |
| 3 | int sieve(int numbers[], int  arraySize) | Ask the user to input the  range to generate and print | Call the  primesInRangeSieve |
|  |  | primes until and create an | function and pass the |
|  |  | array that has the size of the | prepared array to it. |
|  |  | range, then pass the array and |  |
|  |  | its size to the function as |  |
|  |  | arguments. |  |
| 3 | int primesInRangeSieve (int numbers[], int arraySize) | Call the sieve function to prepare the numbers array, then pass it to this function  along with its size so that the | Do nothing |
|  |  | function can print the prime |  |
|  |  | numbers from it. |  |
| 4 | void flipImageVertically(int rows, int columns, int image[rows][columns]) | Extract the number of rows and the number of columns of the image using the helper  functions extractImageRows | Call the drawImageFromFile function to draw and  display the image |
|  |  | and extractImageColumns, | after this function |
|  |  | then create an array with the | flipped it. |
|  |  | extracted dimensions, use |  |
|  |  | readImageFromFile to read |  |
|  |  | the image from the file and |  |
|  |  | fill it in the array then pass |  |
|  |  | the filled array to this |  |
|  |  | function along with the |  |
|  |  | dimensions. |  |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 5 | void rotateImageClockwise (int rows, int columns, int image[rows][columns]) | Extract the number of rows and the number of columns of the image using the helper  functions extractImageRows | Call the drawImageFromFile function to draw and  display the image |  |
|  |  | and extractImageColumns, | after this function |
|  |  | then create an array with the | rotated it. |
|  |  | extracted dimensions, use |  |
|  |  | readImageFromFile to read |  |
|  |  | the image from the file and |  |
|  |  | fill it in the array then pass |  |
|  |  | the filled array to this |  |
|  |  | function along with the |  |
|  |  | dimensions. |  |
| After each choice from 1 to 5, mainMenu should return 0 to the main function. If the user chooses 6 (Exit) the function should return 1 to the main function so it could exit. Any other choice should print “Invalid Choice” and return 0. Check the implementation for the main function if you need to.  **Task Two**  Implement the function int isPrime(int number).  This function checks whether an integer is a prime number or not. A prime number is defined as: **a number that is greater than 1 and has only 2 divisors: 1 and itself.** For example, 2, 5, 7, 11, 73, and 97 are all prime numbers. The opposite of a prime number is a composite number, which is a number that can be formed by multiplying a prime number with another number (e.g., 28 is the product of 7 and 4 or 2 and 14).  The function accepts one integer as a parameter, the number that will be checked for primality. It **returns** 1 if the number is prime, or 0 if the number is not composite.    Figure 2 | | | | | |



Please note that this function does not print anything. All printing should be done in the

mainMenu function as explained in Task 1.

# Task Three

Implement the function void primesInRange(int rangeEnd).

This function accepts one parameter rangeEnd and prints all the prime numbers from 1 up until rangeEnd. It should use the isPrime function to determine whether a number is prime or not.

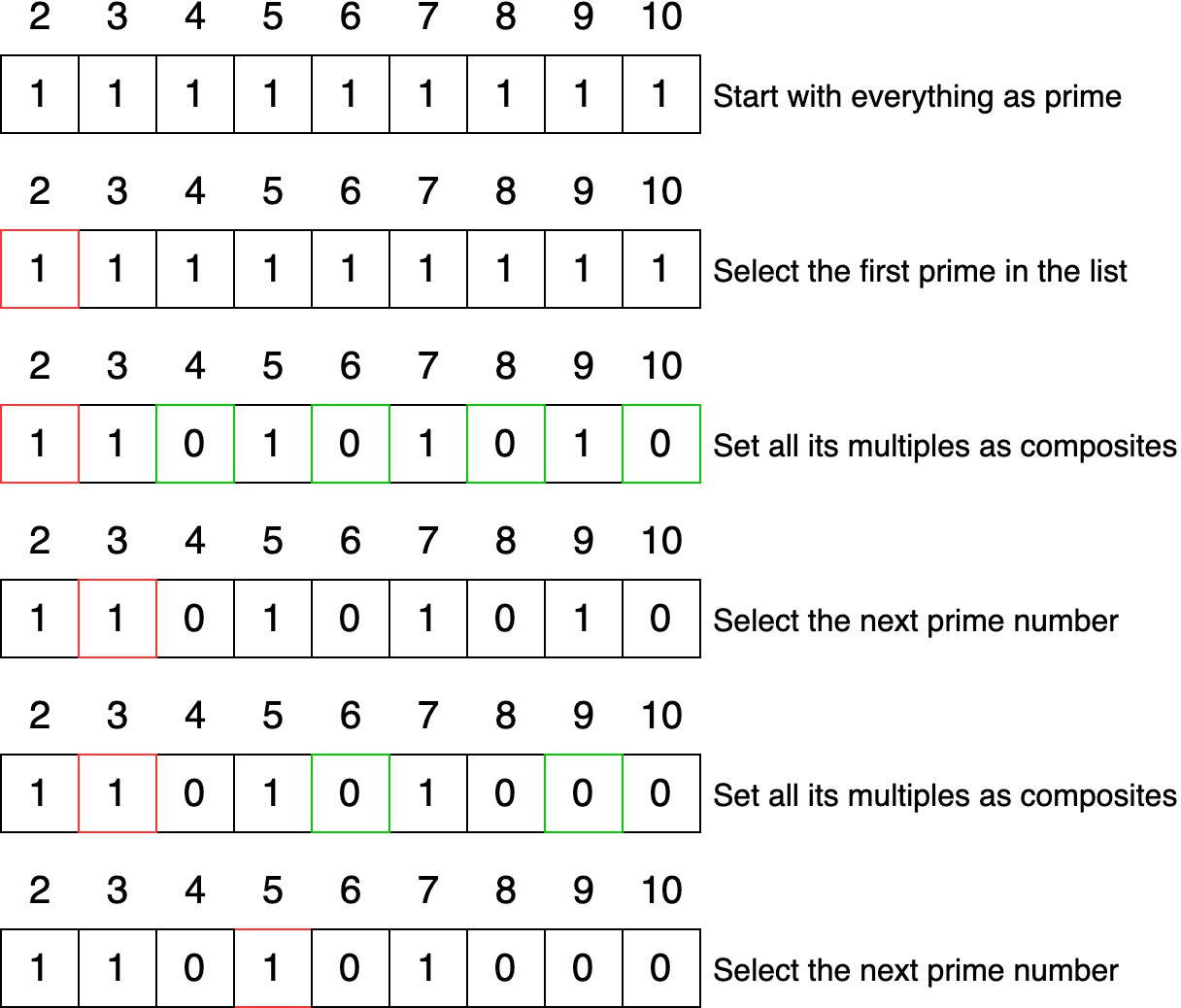
For example, if rangeEnd is 35, the function will print all the prime numbers less than or equal to 35:

Figure 3

The function does not return anything. It **prints** the numbers directly, separated by commas.

# Task Four

Design a flowchart for the function in **Task 3**. You can draw the flowchart on paper and take pictures of it, but you need to be extra careful to make sure the drawing and the pictures are clear. Alternatively, you can use any programs or free websites to build the flowchart using basic shapes (e.g., draw.io). Make sure your flowchart is correct. The flowchart needs to show how both functions isPrime and primesInRange work.



# Task Five

Implement the functions:

int sieve(int numbers[], int arraySize).

int primesInRangeSieve(int numbers[], int arraySize).

# Preparing the array of primes:

The first function implements an algorithm called Sieve of Eratosthenes that was developed by a Greek mathematician called Eratosthenes of Cyrene in 250 BCE (اﻟﻤﯿﻼد ﺒﻗﻞ).

The algorithm is simple. It is used to find prime numbers in a range from 1 to a number. We pick a prime number and mark all its multiples as composite – since they can be divided by that prime number, they are composite. Once done, we pick the next prime number and do the same. When we finish, all the prime numbers will be marked.

Here is a visual representation for finding prime numbers from 1 to 10 (1 means prime, and 0 means composite). We start from 2 because we know that 1 is not a prime:

Figure 4

The above visualization keeps going until all the numbers in the range are selected. Once done, all the composite numbers will be marked with 0, and only the prime numbers will remain marked with 1.

The following is a pseudo-code for the Sieve of Eratosthenes algorithm, it uses an integer array to keep track of the primality of each number. The index of the array represents the number, and the value of the array is either 1 or 0 to represent whether the number in the index is prime or not:

sieve(array, size):

set all values in array to 1 (prime) set array[0] to 0 (composite)

set array[1] to 0

for each index i in array starting from 2: for each multiple of i:

set array[this multiple of i] to 0

When you write the function sieve, it will accept the array and its size as input. The function will then prepare the array according to the algorithm described in the pseudo code above.

# Printing the primes from the prepared array:

The second function accepts an array that was prepared by the sieve function and its size. And simply **prints** all the numbers marked as prime inside it. The numbers are separated by commas when printed.

# Improving Sieve:

Once you have implemented the two functions **and tested them**, you will be ready to improve your algorithm and make it faster by reducing the number of times your loops iterate in the function sieve.

Let us look at the following figure that shows the Sieve of Eratosthenes method done on numbers from 2 to 120.

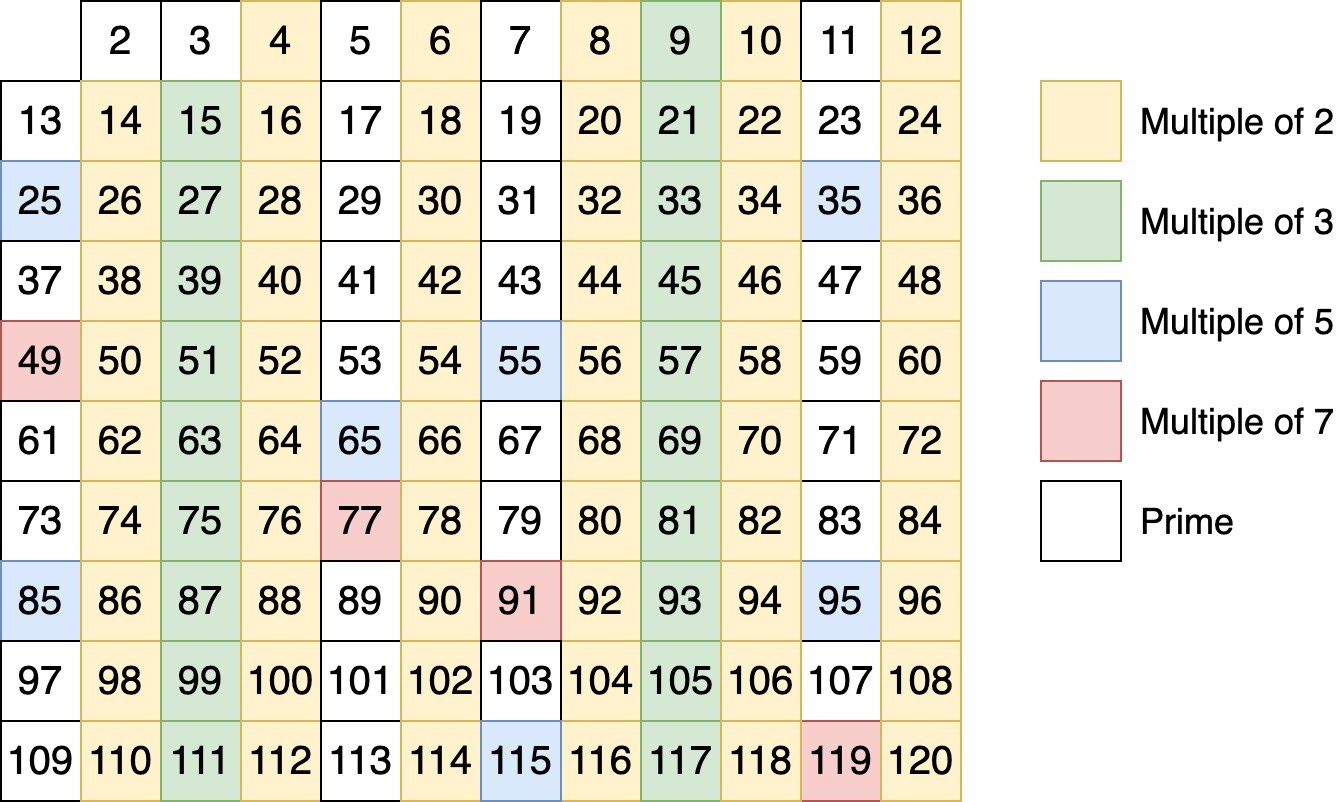


Figure 5

# Improving the outer loop:

Let us consider the composite number 120, it can be formed by multiplying the two numbers in each of the following pair: (1, 120), (2, 60), (3, 40), (4, 30), (5, 24), (6, 20), (8, 15), (10,

12).

Notice that √120 is around 10.95, and how all the numbers in the pairs above are spread around its square root:

1 2 3 4 5 6 8 10 √120 12 15 20 24 30 40 60 120

For each of the factors that are less than or equal to √120, there is a factor greater than or equal to √120 that can be multiplied with it to result in 120. This can help us improve our code because we can be sure that 120 will be marked as composite by one of the numbers that are less than or equal to its square root. Therefore, we do not need to loop until 120, only until

√120.

Of course, in this example 120 will be marked composite when we mark the multiples of 2, but this is not always the case. For example, if we had 121 numbers, then 121 will only be marked as composite when we mark the multiples of 11 (the square root of 121). So it is always enough to iterate until the square root of the number, but not less than that.

This will change our code to look like this:

sieve(array, size):

set all values in array to 1 (prime) set array[0] to 0 (composite)

set array[1] to 0

**for each index i in array from 2 to sqrt(size):**

for each multiple of i:

set array[this multiple of i] to 0

# Improving the inner loop:

Let us now look at the prime number 5, notice how the first multiple of 5 that is not yet marked as composite is 25. All of 10, 15, and 20 were marked composite by previous smaller primes. The same is true if we look at any other prime number, for example 7, the first multiple of 7 that is not yet marked as composite is 49. All of 14, 21, 28, 35, and 42 are marked composite by previous smaller primes. This means that instead of looping through the multiples of the prime number starting with the next multiple (2 × 𝑝𝑟𝑖𝑚𝑒) we can start with the square of the prime (𝑝𝑟𝑖𝑚𝑒2).

This will change our code one more time to look like this:

sieve(array, size):

set all values in array to 1 (prime) set array[0] to 0 (composite)

set array[1] to 0

for each index i in array from 2 to sqrt(size):

**for each multiple of i starting from i2:**

set array[this multiple of i] to 0

Note that i2 might be a huge number, so it is advised to use long long to store it instead of int.

To test the Sieve method against the original method in Task 3, run the program and ask it to output the prime numbers up to 1 million (1,000,000). Notice how the Sieve method is much faster than the original method in Task 3.



# Task Six

Implement the function

void flipImageVertically(int rows, int columns, int image[rows][columns]).

This function accepts the number of rows and columns of an image, along with the image as a 2D array, and flips the image vertically inside the array and writes it into the same file by calling the storeImageInFile function.

Flipping an image vertically means that the top row of pixels becomes the last row, the 2nd row becomes the 2nd to last, and so on.

The following figure shows the result of flipping the original image.in file vertically.

Figure 6: Original image on the left and vertically flipped image on the right

As explained in the table in Task 1, you should read the image dimensions and the actual image from the file before calling this function so you could pass them to it. When this function is done, you should call the appropriate function to draw the image from within mainMenu as well.

Before you implement flipImageVertically, it is recommended that you implement the functions extractImageRows, extractImageColumns, readImageFromFile, and drawImageFromFile.

extractImageRows: Accepts a file name of an image and reads and returns the number of rows of that image. Read the description of the file image.in to understand the structure of the image.

extractImageColumns: Accepts a file name of an image and reads and returns the number of columns of that image. Read the description of the file image.in to understand the structure of the image.

readImageFromFile: Accepts a file name of an image, the number of rows and columns of an image, and a 2D array of the passed dimensions. It reads the pixels of the image from the file and stores them inside the passed 2D array.

drawImageFromFile: Accepts a file name of an image, and the number of rows and columns of an image. It reads the image from the file and draws it using the showArray function from the foc\_fa33.h header.

storeImageInFile: Accepts a file name of an image, the number of rows and columns of an image, and a 2D array of the passed dimensions, and a file name. It writes the pixels of the image in the array into a file called by the passed name. Remember to preserve the structure of the image files so that readImageFromFile can read them again.

# Task Seven

Implement the function

void rotateImageClockwise(int rows, int columns, int image[rows][columns]).

This function accepts the number of rows and columns of an image, along with the image as a 2D array, and rotates the image clockwise by 90 degrees and writes it into the same file by calling the storeImageInFile function.



The following figure shows the result of rotating the original image.in file clockwise one time.

Figure 6: Original image on the left and vertically flipped image on the right

As explained in the table in Task 1, you should read the image dimensions and the actual image from the file before calling this function so you could pass them to it. When this function is done, you should call the appropriate function to draw the image from within mainMenu as well.

Before you implement rotateImageClockwise, it is recommended that you implement the functions extractImageRows, extractImageColumns, readImageFromFile, and drawImageFromFile as described in the previous task.

# Task Eight

Assume that your image manipulation code was developed further to modify live videos while they are being streamed. If the live stream is being modified and broadcasted from computer A, and the audience are watching it from multiple computers B, C, D, and so on, answer the following two questions:

1. What type of connection(s) do you prefer to use to connect the source computer to the other computers?

2. Which communication protocol would you use? Identify **all** the available communication protocols that can be used to implement the connection, then select one of them and justify your selection. Use a block diagram to support your answer.

THE END



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STUDENT ASSESSMENT SUBMISSION AND DECLARATION** | | | | |
| When submitting evidence for assessment, each student must sign a declaration confirming that the work is their own. | | | | |
| **Student name: Student ID:** | | | | **Assessor name:** |
| **Issue date:** | **Submission date:** | | | **Submitted on:** |
| **Programme:** | | | | |
| **HTU Course Name:**  Fundamentals of Computing | | **BTEC UNIT Title: NA** | | |
| **HTU Course Code:** 40303130 | | **BTEC UNIT Code: NA** | | |
| **I AM REPEATING THIS UNIT:** ☐ **Yes** ☐ **No** | | | | |
| **Plagiarism:** Plagiarism is a particular form of cheating. Plagiarism must be avoided at all costs and students who break the rules, however innocently, may be penalized. It is your responsibility to ensure that you understand **correct referencing practices**. As a university level student, you are expected to use appropriate references throughout and keep carefully detailed notes of all your sources of materials for material you have used in your work, including any material downloaded from the Internet. Please consult the relevant unit lecturer or your course tutor if you need any further advice. | | | | |
| **Student declaration**  I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice. | | | | |
| **Student Signature** | | | **Date:** | |



|  |  |  |  |
| --- | --- | --- | --- |
| **Learning Outcomes and Assessment Criteria** | | | |
| **Learning Outcome** | **Pass** | **Merit** | **Distinction** |
| **LO1**: Discuss the basic concepts of computer hardware, software and operating systems, with the basic relation between them, taking C commands as an example. | **P1** Identify the basic hardware of the computer and how to deal with it.  **P2** Identify the basic software of the computer, and able to interact with Ubuntu operating system through its terminal. | **M1** Analyse the basic structure of simple C program.  **M2** Express any arithmetic or logical expression to its equivalent in C language. | **D1** Critically analyse basic C commands and relate it with the machine cycle. |
|  |  | **M3** Apply C language programming basic concepts  to trace and debug simple C programs. |  |
| **LO2**: Implement a full program in C with identification of the three kinds of control structures: sequence, selection, and repetition, understanding the capabilities of implementing C codes that capable to deal with functions. | **P3** Implement problem solving decisions in algorithms and introduce structure charts as a system documentation tool.  **P4** Identify the C syntax of sequence, selection, and repetition and when to use each statement type.  **P5** Identify the structure of functions with passing information and get data if  needed. | **M4** Solve real life problems using control structures in C language integrated with functions. | **LO 2 & 3**  **D2** Critically evaluate the source code that solve real life problem implementing the functions, arrays, and files following the code standards  and best practices. |
| **LO3**: Understanding the basic concepts of pointers  in C with implementation of arrays and dealing with files. In addition to the role of C programming as a powerful tool to communicate between devices and manipulate their data. | **P6** Explore pointer’s role in referencing variables, and how it is reflected in the memory.  **P7** Implement arrays as one dimensional or two dimensional and dealing with any operation on them.  **P8** Explore the fundamentals of data communication between devices. | **M5** Implement the effectiveness of arrays in real applications with usage of functions.  **M6** Identify streams in C and their relationship to deal with files.  **M7** Introduce a block diagram to solve a real-life problem in communication between devices as a practical application of C  language. |  |



This page is left empty intentionally