Task-1:

Suppose you and your friends go to have breakfast in a restaurant. Each of you orders Paratha, Vegetable, and Mineral Water. Treat each of the ordered items as structures and each of the structures will have two properties which are: quantity and unit price. Each property of the structure will be taken as input from the user. After taking all the inputs calculate what is the total bill and also, take input from the user on how many people are there in total. Lastly calculate how much each person will have to pay and print it (Note: This value will be a float).

```
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/lab_A2$ gedit 1st_task1.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/lab_A2$ gcc -o 1st_task1_output 1st_task1.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/lab_A2$ ./1st_task1_output
Enter quantity and unit price for Paratha:
25
10
Enter quantity and unit price for Vegetable:
5
20
Enter quantity and unit price for Mineral Water:
20
20
Enter the number of people:
6
Individual people will pay: 125.00 tk
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/lab_A2$
```

Task-2:

Write a C program to print perfect numbers between given intervals using a function. A perfect number is a positive integer equal to the sum of its positive divisors, excluding the number itself.

```
10000
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/lab_A2$ gedit 1st_task2.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/lab_A2$ gcc -o 1st_task2_output 1st_task2.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/lab_A2$ ./1st_task2_output
lower bound: 1
upper bound: 10000
6
28
496
8128
```

The following questions are related to system call [5X2= 10 marks]

Task-1:

Write a c program that will open a file given from the command line argument and then it will ask the user to input strings that will be written to that file. It will continue to ask the user to enter a string as long as the user enters "-1". If the given file does not exist in the directory, then your program will automatically create the file.

```
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$ gedit task1.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$ gcc -o task1 task1.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$ ./task1 output1.txt
Death
War
kkkkkkkkkkkkkk
generate
-1
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$ cat output1.txt
Death
War
kkkkkkkkkkkkkk
generate
```

Task-2:

Write a program that will create a child and another grandchild process. Every process will print a line.

The parent process will print, "I am parent"

Child process will print, "I am child"

The grandchild process will print, "I am grandchild"

```
generate
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$ gedit task2.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$ gcc -o task2 task2.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$ ./task2
I am grandchild
I am child
I am parent
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$
```

Task-3:

Consider the following code snippet in your main function -

```
a = fork();

b = fork();

c = fork();
```

Now, write the full program, that will check the children's PID and if it is odd then the process will create another child process. Lastly, print how many processes have been created considering the first parent process.

```
Total: 11
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/system_call$ gedit task3.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/system_call$ gcc -o task3 task3.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/system_call$ ./task3
Parent : 21239, Child : 21674
Parent : 21674, Child : 21677
Parent : 21674, Child : 21676
Parent : 21676, Child : 21679
Parent : 21674, Child : 21675
Parent : 21675, Child : 21680
Parent : 21675, Child : 21678
Parent : 21678, Child : 21681
Total: 12
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/system_call$ ./task3
```

```
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/system_call$ ./task3
Parent : 21239, Child : 21931
Parent : 21931, Child : 21935
Parent : 21931, Child : 21933
Parent : 21931, Child : 21932
Parent : 21932, Child : 21934
Parent : 21933, Child : 21936
Parent : 21934, Child : 21938
Parent : 21932, Child : 21937
Total: 11
```

Task-4:

Write a program named "sort.c" where you will give some number from the command line argument and the program will print the sorted array in descending order. Then, write another program named "oddeven.c" which will take some numbers from the command line, then check and print whether the numbers in the array are odd or even.

Now, you have to write a program that will create a child process and the child process will first sort the array that you have declared in this program. And then, the parent process will print the odd/even status for each number in the array.

```
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~$ gedit sort.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~$ gcc -o sort sort.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~$ ./sort 12 5 76 2 55 32
76 55 32 12 5 2
```

```
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~$ gedit oddeven.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~$ gcc -o oddeven oddeven.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~$ ./oddeven 12 5 76 2 55 32
12 : even
5 : odd
76 : even
2 : even
55 : odd
32 : even
```

```
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~$ gedit task4.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~$ gcc -o task4 task4.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~$ ./task4 12 5 76 2 55 32
76 55 12 5 2
12 : even
5 : odd
76 : even
2 : even
55 : odd
```

Task -5:

Write a program in c that the parent process will create one child process and 3 grandchild processes and print their IDs

Output: 1. Parent process ID : 0

2. Child process ID: ….

```
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$ gcc -o task5 task5.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$ ./task5
1. Parent process ID : 18525
2. Child process ID : 18526
3. Grandchild process ID : 18527
4. Grandchild process ID : 18528
5. Grandchild process ID : 18529
```

The following questions are related to Threading [3X2= 6 marks]

Task-1:

Write a c program that creates 5 threads and prints which thread is running and after the thread is closed, a new thread starts its execution. Each thread should run sequentially one by one.

OUTPUT:
thread-1 running
thread-1 closed
thread-2 running
thread-2 closed
…

```
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$ gcc -o task1 task1.c -lpthread
task1.c: In function ‘open_th’:
task1.c:10:5: warning: implicit declaration of function ‘sleep’ [-Wimplicit-function-decla
   10 |     sleep(1);
      |     ^~~~~
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$ ./task1
thread-1 running
thread-1 closed
thread-2 running
thread-2 closed
thread-3 running
thread-3 closed
thread-4 running
thread-4 closed
thread-5 running
thread-5 closed
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$
```

Task-2:

Write a program in c using 5 threads where each thread will print 5 integers

The outputs will look like this:

Output:

Thread 0 prints 1

Thread 0 prints 2

Thread 0 prints 3

```
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$ gedit task2.c
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$ gcc -o task2 task2.c -lpthread
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$ ./task2
thread-1 prints 1
thread-1 prints 2
thread-1 prints 3
thread-1 prints 4
thread-1 prints 5
thread-2 prints 6
thread-2 prints 7
thread-2 prints 8
thread-2 prints 9
thread-2 prints 10
thread-3 prints 11
thread-3 prints 12
thread-3 prints 13
thread-3 prints 14
thread-3 prints 15
thread-4 prints 16
thread-4 prints 17
thread-4 prints 18
thread-4 prints 19
thread-4 prints 20
thread-5 prints 21
thread-5 prints 22
thread-5 prints 23
thread-5 prints 24
thread-5 prints 25
ubuntu@ubuntu-HP-280-Pro-G8-Microtower-PC:~/Downloads$
```

Task-3:

Write a program in c that has a function that takes the name of the user and adds all the ASCII value of the characters and returns it. Now create 3 threads that run the function using 3 different user names. Now print "Youreka" if all the returned values are equal, print "Miracle" if the 2 returned values are equal, and print "Hasta la vista" if the values don't match using another thread.