Date: 14/06/2021

Exercise 4

Name: *Krithika Swaminathan*
Roll No.: *205001057*

1. Sums of Odds and Evens

Aim:

To get the number of inputs required as input from the user; then, after identifying each number entered as input as odd or even, to add all the even numbers together, the odd together, and display the sums.

Algorithm:

Step 1: Start
Step 2: Define a function, say, *CheckOddEven*, to check whether a given number is odd or even. The function should be as follows:

- The function should be of datatype **int**.
- It takes an integer parameter, say, *num*.
- Initialise a flag variable to 0.
- If *num* is divisible by 2, the value of flag is changed to 1.
- Return the value of flag.

Step 3: In the main function, read the number of integer entries as input from the user.
Step 4: Initialise two variables to calculate the sums (say, *sum_even* and *sum_odd*) to zero.
Step 5: Run a for loop as many times as the number of integer entries and read each entry.
Step 6: For each entry, call the function *CheckOddEven* and check if the return value is 0 or 1.
Step 7: If the return value is 1, add the integer entry to *sum_even*. If the return value is 0, add the integer entry to *sum_odd*.
Step 8: After all the iterations are complete, print *sum_even* and *sum_odd*.
Step 9: End

Code:

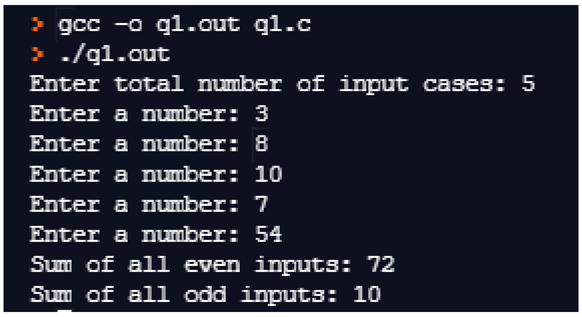
```c
//to check if the entered numbers are odd or even and to sum the odds and evens separately
#include <stdio.h>

//function to check if odd or even
int CheckOddEven(int num){
  int flag=0;
  if (num%2==0) flag=1;
  return flag;
}

//main function
int main(void) {
  int N, n, sumEven=0, sumOdd=0;
  printf("Enter total number of input cases: ");
  scanf("%d",&N);
  for (int i=1; i<=N; i++){
    printf("Enter a number: ");
    scanf("%d",&n);
```

Exercise 4

Name: *Krithika Swaminathan*
Roll No.: *205001057*

```
    if (CheckOddEven(n)==1) sumEven+=n;
    else sumOdd+=n;
  }

  printf("Sum of all even inputs: %d\n",sumEven);
  printf("Sum of all odd inputs: %d\n",sumOdd);

  return 0;
}
```

Output:



```
> gcc -o q1.out q1.c
> ./q1.out
Enter total number of input cases: 5
Enter a number: 3
Enter a number: 8
Enter a number: 10
Enter a number: 7
Enter a number: 54
Sum of all even inputs: 72
Sum of all odd inputs: 10
```

Result:

A program for adding all the even numbers and odd numbers separately is written and executed.

Exercise 4

Name: *Krithika Swaminathan*
Roll No.: *205001057*

2. Reverse of a Number

Aim:

To get any integer as input from the user and print the reverse of that number.

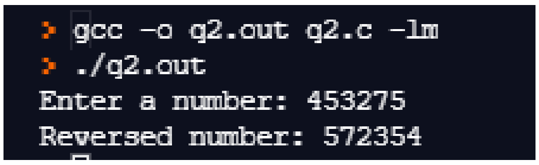
Algorithm:

```
Step 1: Start
Step 2: Define a function, say, ReverseNum, to reverse a given integer. The function should be as follows:
    • The function should be of datatype int.
    • It should accept an integer, say, num, as a parameter.
    • Initialise a counter to -1 and a "reversed number" to 0.
    • Run a for loop and let the loop variable be i. Initialise i as num and run the loop for as long
      as i is positive. The value of i is altered to i/10 after each iteration.
    • For each iteration, increment the value of the counter. This gives us the number of digits in
      the integer.
    • Run a for loop and let the loop variable be i. Initialise i as num and run the loop for as long
      as i is positive. The value of i is altered to i/10 after each iteration.
    • In each iteration, add the product of the last digit of i and 10 raised to the counter to the
      "reversed number". Then, decrement the counter by 1.
    • Return the reversed number.
Step 3: In the main function, read an integer as input from the user.
Step 4: Call the ReverseNum function with the integer as the argument and print the return value.
Step 5: End
```

Code:

```c
//to reverse a given number using functions
#include <stdio.h>
#include <math.h>

//function to reverse a number
int ReverseNum(int num){
  int ctr=-1,rev=0;
  for (int i=num; i>0; i/=10)
    ctr++;
  for (int i=num; i>0; i/=10){
    rev+=(i%10)*pow(10,ctr);
    ctr--;
  }
  return rev;
}

int main(void) {
  int n;
  printf("Enter a number: ");
  scanf("%d",&n);

  printf("Reversed number: %d\n",ReverseNum(n));

  return 0;
}
```

Output:



```
> gcc -o q2.out q2.c -lm
> ./q2.out
Enter a number: 453275
Reversed number: 572354
```

Result:

A program for reversing a given number is written and executed.

Date: 14/06/2021

Exercise 4

Name: *Krithika Swaminathan*
Roll No.: *205001057*

3. Evaluating the Power

Aim:

To get a base and an integer exponent as input from the user and to evaluate the power.

Algorithm:

Step 1: Start
Step 2: Define a function, say, *Power*, to calculate the power of a given number raised to a given integer. The function should be as follows:

- The function should be of datatype **float.**
- It should take two parameters – the base (**float**) and the exponent (**int**).
- Use the power function to calculate the value of the base raised to the exponent.
- Return the power.

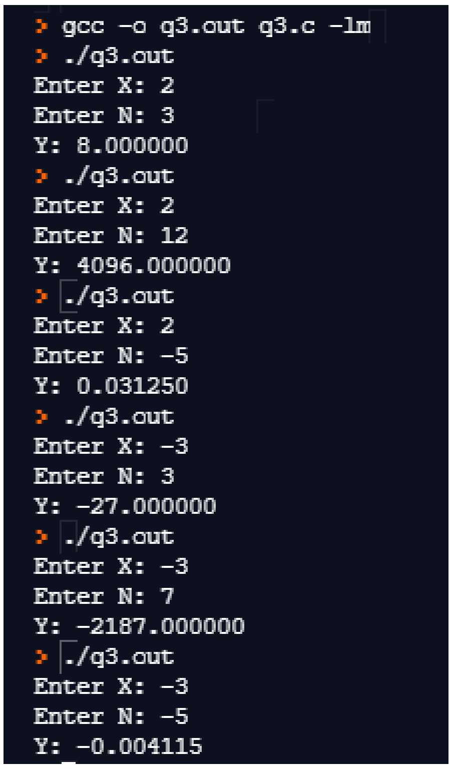
Step 3: In the main function, read the base and exponent as input from the user.
Step 4: Call the function *Power* with the base and exponent as arguments and print the return value.
Step 5: End

Code:

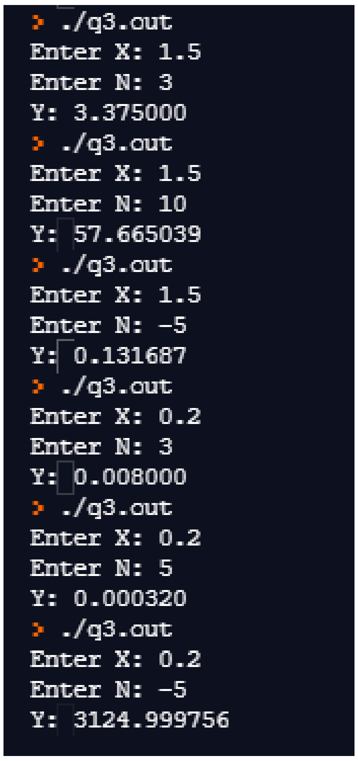
```c
//to find the power of a number using functions
#include <stdio.h>
#include <math.h>

float power(float x, int n){
  float power;
  power=pow(x,n);
  return power;
}

int main(void) {
  float x;
  int n;
  printf("Enter X: ");
  scanf("%f",&x);
  printf("Enter N: ");
  scanf("%d",&n);

  printf("Y: %f\n",power(x,n));

  return 0;
}
```

Exercise 4

Output:



```
> gcc -o q3.out q3.c -lm
> ./q3.out
Enter X: 2
Enter N: 3
Y: 8.000000
> ./q3.out
Enter X: 2
Enter N: 12
Y: 4096.000000
> ./q3.out
Enter X: 2
Enter N: -5
Y: 0.031250
> ./q3.out
Enter X: -3
Enter N: 3
Y: -27.000000
> ./q3.out
Enter X: -3
Enter N: 7
Y: -2187.000000
> ./q3.out
Enter X: -3
Enter N: -5
Y: -0.004115
```



```
> ./q3.out
Enter X: 1.5
Enter N: 3
Y: 3.375000
> ./q3.out
Enter X: 1.5
Enter N: 10
Y: 57.665039
> ./q3.out
Enter X: 1.5
Enter N: -5
Y: 0.131687
> ./q3.out
Enter X: 0.2
Enter N: 3
Y: 0.008000
> ./q3.out
Enter X: 0.2
Enter N: 5
Y: 0.000320
> ./q3.out
Enter X: 0.2
Enter N: -5
Y: 3124.999756
```

Result:

A program for evaluating the power when the base and exponent are given is written and executed.

Exercise 4

Name: *Krithika Swaminathan*
Roll No.: *205001057*

4. Product of N numbers

Aim:

To calculate the product of N floating point numbers input by the user, using recursive functions.

Algorithm:

Step 1: Start
Step 2: Define a function, say, *Product*, to find the product of the given list of numbers using recursion. The function should be as follows:

- The function should be of datatype **float**.
- It should accept an integer (N) as the parameter.
- Read a number as input from the user.
- Set the base case as: if N is 1, return the user-input number.
- If N is not 1, return the product of **the number and the return value of *Product*(*N-1*)**.

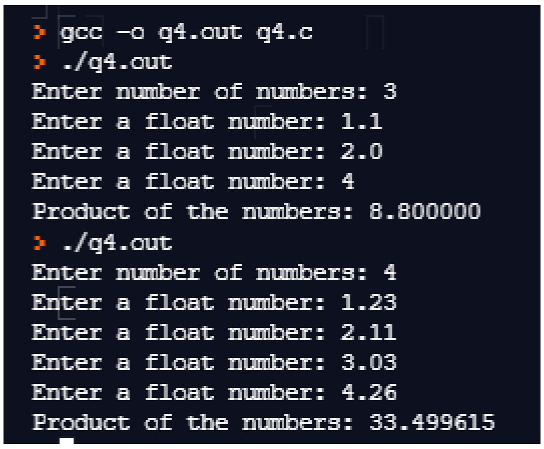
Step 3: In the main function, read the number of entries as input from the user.
Step 4: Call the *Product* function with the number of entries as the argument and print the return value.
Step 5: End

Code:

```c
//to find the product of N float numbers using recursion
#include <stdio.h>

float prod(int N){
  float x;
  printf("Enter a float number: ");
  scanf("%f",&x);

  if (N==1)
    return x;
  else
    return x*prod(N-1);
}

int main(void) {
  int n;
  printf("Enter number of numbers: ");
  scanf("%d",&n);

  printf("Product of the numbers: %f\n",prod(n));

  return 0;
}
```

Exercise 4

Output:



```
> gcc -o q4.out q4.c
> ./q4.out
Enter number of numbers: 3
Enter a float number: 1.1
Enter a float number: 2.0
Enter a float number: 4
Product of the numbers: 8.800000
> ./q4.out
Enter number of numbers: 4
Enter a float number: 1.23
Enter a float number: 2.11
Enter a float number: 3.03
Enter a float number: 4.26
Product of the numbers: 33.499615
```

Result:

A program to find the product of the given numbers using recursion is written and executed.

Exercise 4

Name: *Krithika Swaminathan*
Roll No.: *205001057*

5. Printing the Predecessors

Aim:

To get a number as input from the user and to print all the numbers preceding it in order until 0 is reached.

Algorithm:

Step 1: Start
Step 2: Read a global variable, say, *num* and set *num* to be 0.
Step 3: Define a function, say, *Print*, to print all the numbers preceding a given number (till 0) using recursion. The function should be as follows:

- The function should be of datatype **long**.
- It should take an integer (N) as the parameter.
- Change the value of *num* using *num=(num*10)+(N-1)*.
- Set the base case to be: if N is 1, return *num*.
- If N is not 1, then return the return value of *Print*(*N-1*).

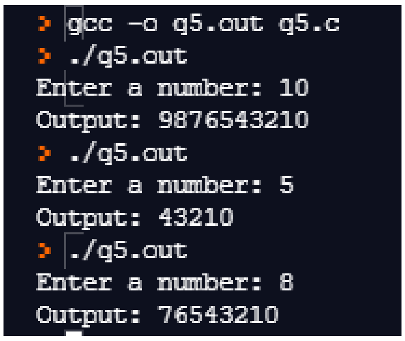
Step 4: In the main function, read an integer as input from the user.
Step 5: Call the *Print* function and print its return value.
Step 6: End

Code:

```
//to print all the numbers preceding the given number till 0
#include <stdio.h>

long num=0;
long print(int N){
  num=(num*10)+(N-1);
  if (N==1)
    return num;
  else
    return print(N-1);
}

int main(void) {
  int n;
  printf("Enter a number: ");
  scanf("%d",&n);

  printf("Output: %ld\n",print(n));

  return 0;
}
```

Exercise 4

Output:



```
> gcc -o q5.out q5.c
> ./q5.out
Enter a number: 10
Output: 9876543210
> ./q5.out
Enter a number: 5
Output: 43210
> ./q5.out
Enter a number: 8
Output: 76543210
```

Result:

A program for printing all the numbers before the given number till zero is written and executed.

Name: *Krithika Swaminathan*
Roll No.: *205001057*

6. Right-most Non-zero Digit of a Factorial

Aim:

To get a number as input from the user and to find the right-most non-zero digit of the factorial of that number, using recursion to find the factorial.

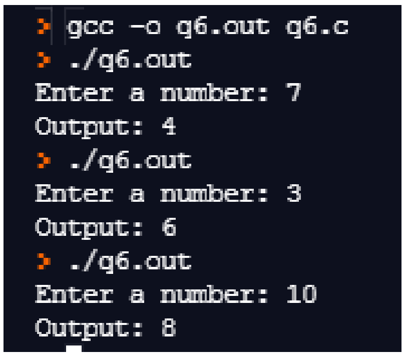
Algorithm:

```
Step 1: Start
Step 2: Define a function, say, Factorial, to find the factorial of a number using recursion. The function should be as follows:
        • The function should be of datatype long.
        • It should take an integer (num) as the parameter.
        • Set the base case to be: if num is 0, return 1.
        • If num is not 0, then return the product of num and the return value of Factorial(num-1).
Step 4: In the main function, read an integer (n) as input from the user.
Step 5: Run a for loop, initialising the loop variable i as the value of Factorial(n), altering it to i/10 after each iteration, and terminating the loop when i becomes less than 0.
Step 6: For each iteration, if the last digit of i is not 0, print the last digit of i and break the loop.
Step 7: End
```

Code:

```c
//to find the right-most non-zero digit of the factorial of a number
#include <stdio.h>

//function to find the factorial of a given number
long fact(int num){
  if (num==0)
    return 1;
  else
    return num*fact(num-1);
}

int main(void) {
  int n;
  printf("Enter a number: ");
  scanf("%d",&n);

  for (long i=fact(n);i>0;i/=10){
    if (i%10){
      printf("Output: %ld\n",i%10);
      break;
    }
  }
  return 0;
}
```

Output:



```
> gcc -o q6.out q6.c
> ./q6.out
Enter a number: 7
Output: 4
> ./q6.out
Enter a number: 3
Output: 6
> ./q6.out
Enter a number: 10
Output: 8
```

Result:

A program to find the right-most non-zero digit of the factorial of a given number is written and executed.