1- A positive integer n is said to be ***prime*** **(or, "a prime")** if and only if n is *greater than* 1 and is divisible only by 1 and n . For example, the integers 17 and 29 are prime, but 1 and 38 are not prime. Write a function named "is_prime" that takes a *positive* integer argument and returns as its value the integer 1 if the argument is prime and returns the integer 0 otherwise.
Thus, for example,

```
cout << is_prime(19) << endl; // will print 1
cout << is_prime(1) << endl;   // will print 0
cout << is_prime(51) << endl; // will print 0
cout << is_prime(-13) << endl; //will print 0
```

2- Write a function named "digit_name" that takes an integer argument in the range from 1 to 9 , inclusive, and prints the English name for that integer on the computer screen. No newline character should be sent to the screen following the digit name. The function should not return a value. The cursor should remain on the same line as the name that has been printed. If the argument is not in the required range, then the function should print "digit error" without the quotation marks but followed by the newline character.

Thus, for example,
the statement *digit_name(7);* should print seven on the screen;
the statement *digit_name(0);* should print digit error on the screen and place

3- Write a function grade2number that takes a student's letter grade without \+/- (A,B,C,D,F) as a char and returns the corresponding numerical score in the GPA (4.0,3.0,2.0,1.0,0.0). You should handle the case when the student enters an invalid character (e.g. G) by return the number -1.0. Use a switch statement. The function will be called as in the code below.

```
char grade;
cout << "Enter your letter grade (no +/-): ";
cin >> grade;
cout << "The corresponding numeric grade is: "
     << grade2number(grade);
```

4- Write a function named *"reduce"* that takes two positive integer arguments, call them "num" and "denom", treats them as the numerator and denominator of a fraction, and reduces the fraction. That is to say, each of the two arguments will be modified by dividing it by the greatest common divisor of the two integers. The function should return the value 0 (to indicate failure to reduce) if either of the two arguments is zero or negative, and should return the value 1 otherwise.

Thus, for example, if m and n have been declared to be integer variables in a program, then

```
m = 25;
n = 15;
if (reduce(m,n))
cout << m << '/' << n << endl;
else
cout << "fraction error" <<endl;
//will produce the following output:
5/3
```

Note that the values of m and n were modified by the function call. Similarly,

```
m = 63;
n = 210;
if (reduce(m,n))
cout << m << '/' << n << endl;
else
cout << "fraction error" <<endl;
//will produce the following output:
3/10
```

5- Write a function named "swap_floats" that takes two floating point arguments and interchanges the values that are stored in those arguments. The function should return no value.

To take an example, if the following code fragment is executed

```
float x = 5.8, y = 0.9;
swap_floats (x, y);
cout << x << " " << y << endl;
```

then the output will be

```
0.9 5.8
```

6- Write a function named "sort3" that takes three floating point arguments, call them "x" , y" , and "z" , and modifies their values, if necessary, in such a way as to make true the following inequalities: $x \leq y \leq z$.

The function should return no value. To take an example, if the following code fragment is executed

```
float a = 3.2, b = 5.8, c =0.9;
```

```
sort3 (a, b, c);
cout << a << " " << b << " " << c <<endl;
```

then the output will be

0.9 3.2 5.8

7- Write a function that takes an array of integers and a "target" number as arguments and then returns the number of occurrences of the target in that array.

8- Write a function named **sumOfProduct** which takes two 1D integer arrays and an integer **n** as parameters and returns the sum of products of the first **n** elements of the two arrays.

For example, given the following two 1D arrays:

```
int[] arr1 = {1, 2, 3, 4, 5};
int[] arr2 = {6, 7, 8, 9, 10};
```

Invoking **sumOfProduct(arr1, arr2, 4)** will return the sum of products of the first 4 elements of **arr1** and **arr2**, that is, 1 * 6 + 2 * 7 + 3 * 8 + 4 * 9 = 80 as the result.

9- Write a function named **"reverse"** that takes as its arguments the following:

(1) an array of floating point values;

(2) an integer that tells how many floating point values are in the array.

The function must reverse the order of the values in the array. Thus, for example, if the array that's passed to the function looks like this:

5.8 | 2.6 | 9.0 | 3.4 | 7.1

then when the function returns, the array will have been modified so that it looks like this: 7.1 | 3.4 | 9.0 | 2.6 | 5.8

The function should not return any value.

10- Write a function named **"sum"** that takes as its arguments the following:

(1) an array of floating point values;

(2) an integer that tells how many floating point values are in the array.

The function should return as its value the sum of the floating point values in the array. Thus, for example, if the array that's passed to the function looks like this:

5.8 | 2.6 | 9.0 | 3.4 | 7.1

then the function should return the value 27.9 as its value.

11- Write a function named **"location_of_largest"** that takes as its arguments the following:

(1) an array of integer values;

(2) an integer that tells how many integer values are in the array.

The function should return as its value the subscript of the cell containing the largest of the values in the array.

Thus, for example, if the array that's passed to the function looks like this:

58 | 26 | 90 | 34 | 71

then the function should return 2. If there is more than one cell containing the largest of the values in

the array, then the function should return the *smallest* of the subscripts of the cells containing the largest values. For example, if the array that's passed to the function is

58 | 26 | 91 | 34 | 70 | 91 | 88

then the largest value occurs in cells 2 and 5 , so the function should return the integer value 2 .

12-

Write a function named "`subtotal`" takes as its arguments the following:

(1) an array of floating point values;

(2) an integer that tells the number of cells in the array.

The function should replace the contents of each cell with the sum of the contents of all the cells in the original array from the left end to the cell in question. Thus, for example, if the array passed to the function looks like this:

0	1	2	3	4
5.8	2.6	9.1	3.4	7.0

then when the function returns, the array will have been changed so that it looks like this:

0	1	2	3	4
5.8	8.4	17.5	20.9	27.9

because $5.8 + 2.6 = 8.4$ and $5.8 + 2.6 + 9.1 = 17.5$ and so on. Note that the contents of cell 0 are not changed. The function should not return a value.

13- Write a function named **"rotate_right"** that takes as its arguments the following:

(1) an array of floating point values;

(2) an integer that tells the number of cells in the array;

The function should shift the contents of each cell one place to the right, except for the contents of the last cell, which should be moved into the cell with subscript 0 . Thus, for example, if the array passed to the function looks like this:

5.8 | 2.6 | 9.1 | 3.4 | 7.0

then when the function returns, the array will have been changed so that it looks like this:

7.0 | 5.8 | 2.6 | 9.1 | 3.4

The function should not return a value.

14- Write a function that takes an array of integers and size of that array as argument and returns the mode, which is the number that appears most frequently in the array.

-