Rwanda Coding Academy

Nyabihu District
rca.ac.rw
@info@rca.ac.rw
0788-548-000

Data Structures and algorithms

Using C++ Perform the following with your own functions.

PART A: EASY

1. Write a C++ program that will display the calculator menu. The program will prompt the user to choose the operation choice (from 1 to 5). Then it asks the user to input two integer values for the calculation. The function must be grouped in arithmetic namespace. See the sample below.

   MENU
   1. Add
   2. Subtract
   3. Multiply
   4. Divide
   5. Modulus

   Enter your choice: 1
   Enter your two numbers: 12 15
   Result: 27

   Continue? y

   The program also asks the user to decide whether he/she wants to continue the operation. If he/she input 'y', the program will prompt the user to choose the operation again. Otherwise, the program will terminate.

2. Write a program in C++ to print the sum of two numbers.
   Example of the output :The sum of 29 and 30 is : 59
3. Write a program in C++ to add two numbers accepted through the keyboard.
4. Write a program in C++ to swap two numbers.

   Example of output

   Input 1st number : 25

Input 2nd number : 39
After swapping the 1st number is : 39
After swapping the 2nd number is : 25

5. Write a program that calculates the volume of the sphere.
6. Compute the sum of all even numbers less than 100
7. Compute the product of all odd numbers between 1 and 20.
8. Compute the product of prime numbers between 1 and 100.
9. Write a function that prints all integer pairs(a,b) greater than 1 and less than 100 that can satisfy the hypotenuse rule. Hypotenuse should be also integer. Display the number of pairs found. Treat (3,4) and (4,3) as one pair.
10. Given a, b,c indices of a quadratic function , find the roots x1, x2 of the equation using the following:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

11. Write a program in C++ to find the Area and Perimeter of a Rectangle. You should get the width and length from the user through the keyboard.

12. Write a program in C++ with a function to convert temperature in Celsius to Fahrenheit.

Hints

How to convert Celsius temperatures to Fahrenheit
Multiply the Celsius temperature by 9/5.
Add 32° to adjust for the offset in the Fahrenheit scale.

13. Write a program with a function to compute quotient and remainder of two numbers accepted from keyboard.

14. Write a program in C++ with a function to check whether a number is positive, negative or zero.

15. Write a C++ program with a function to display the current date and time
16. Write a program in C++ with a function which accepts the user's first and last name and print them in reverse order with a space between them.

Sample output

Input First Name: Mugisha
Input Last Name: Sam
Name in reverse is: Sam Mugisha

17. Write a program which accepts the radius of a circle from the user and compute the area and circumference. The area and circumference must be in separate functions.
18. Write a function to compute the volume of the cube on user input.
19. Write a function to read three integers and to print them in ascending order
20. A function is required that checks strings to see if they are palindromes. A palindrome string that reads the same, backwards, or forwards. For example,
    I was able no on elba saw I

21. John is 1,000,000,000 seconds old; how old is John in terms of Years.
22. Build a function that checks the number of spaces in a given string.
23. Build a function that check if an entered character is a digit or alpha letter.
24. Build a word count function which display the following through input provided by the user:
    i. Number of characters without spaces
    ii. Number of characters with spaces
    iii. Number of words
25. Build a function that reverse a given string
26. Build a function that uses a while loop to calculate the length of a string without using built-in functions.
27. Write a function that checks if a given string is a palindrome with and without built in functions.
28. Write a function that converts the entered string to Uppercase
29. Write a function that removes all spaces in a string and print the resulting string.

PART B: MEDIUM

30. Given a number n, determine what the nth prime is. By listing the first six prime numbers: 2, 3, 5, 7, 11, and 13, we can see that the 6th prime is 13. If your language provides methods in the standard library to deal with prime numbers, pretend they don't exist and implement them yourself.

31. Write a function named **minDistance** that returns the smallest positive distance between two factors of a number. For example, consider 13013 = 1*7*11*13. Its factors are 1, 7, 11, 13 and 13013. **minDistance**(13013) would return 2 because the smallest positive distance between any two factors is 2 (13 - 11 = 2). As another example, **minDistance** (8) would return 1 because the factors of 8 are 1, 2, 4, 8 and the smallest positive distance between any two factors is 1 (2 – 1 = 1).

Hint: Converting a number into a string or using any built-in function.

32. Write a function named **countDigit** that returns the number of times that a given digit appears in a positive number. For example countDigit(32121, 1) would return 2 because there are two 1s in the number 32121. Other examples:

    countDigit(33331, 3) returns 4

    countDigit(33331, 6) returns 0

    countDigit(3, 3) returns 1

    The function should return -1 if either argument is negative, so

    countDigit(-543, 3) returns -1.

    Hint: Converting a number into a string or using any built-in function yield in zero.

33. Find the difference between the square of the sum and the sum of the squares of the first N natural numbers.

The square of the sum of the first ten natural numbers is $(1 + 2 + ... + 10)^2 = 55^2 = 3025$.

The sum of the squares of the first ten natural numbers is $1^2 + 2^2 + ... + 10^2 = 385$.

Hence the difference between the square of the sum of the first ten natural numbers and the sum of the squares of the first ten natural numbers is 3025 - 385 = 2640.

34. An array with an odd number of elements is said to be centered if all elements (except the middle one) are strictly greater than the value of the middle element. Note that only arrays with an odd number of elements have a middle element. Write a function that accepts an integer array and array size and returns 1 if it is a centered array, otherwise it returns 0.

Examples

If the input array is	**Then the function returns**
{1, 2, 3, 4, 5}	0 (the middle element 3 is not strictly less than all other elements)
{3, 2, 1, 4, 5}	1 (the middle element 1 is strictly less than all other elements)
{3, 2, 1, 4, 1}	0 (the middle element 1 is not strictly less than all other elements)
{1, 2, 3, 4}	0 (no middle element)
{}	0 (no middle element)

35. **Write a function that takes an array of integers as an argument and returns the difference of the sum of odd numbers and the sum of even numbers in the array.** Let X = the sum of the odd numbers in the array and let Y = the sum of the even numbers. The function should return X – Y

The signature of the function is

int difference(int[ ] a)

Examples

if input array is	**return**
{1}	1
{1, 2}	-1
{1, 2, 3}	2
{1, 2, 3, 4}	-2
{3, 3, 4, 4}	-2
{3, 2, 3, 4}	0
{4, 1, 2, 3}	-2
{1, 1}	2
{}	0

36. Write a function that reverse an integer.

if the input integer is	**return**
1234	4321
12005	50021
1	1
1000	1
0	0

-12345	-54321

37. Convert a number into a string that contains raindrop sounds corresponding to certain potential factors. A factor is a number that evenly divides into another number, leaving no remainder. The simplest way to test if a one number is a factor of another is to use the modulo operation. The rules of raindrops are that if a given number:
    - has 3 as a factor, add 'Pling' to the result.
    - has 5 as a factor, add 'Plang' to the result.
    - has 7 as a factor, add 'Plong' to the result.
    - *does not* have any of 3, 5, or 7 as a factor, the result should be the digits of the number.

Examples

- 28 has 7 as a factor, but not 3 or 5, so the result would be "Plong".
- 30 has both 3 and 5 as factors, but not 7, so the result would be "PlingPlang".
- 34 is not factored by 3, 5, or 7, so the result would be "34".

38. **Write a function to return an array containing all elements common to two given arrays containing distinct positive integers. You should not use any inbuilt methods. You are allowed to use any number of arrays.**
    The signature of the function is:
    int[] f(int[] first, int[] second)

Examples

if input parameters are	**return**
{1, 8, 3, 2}, {4, 2, 6, 1}	{1, 2}

{1, 8, 3, 2, 6}, {2, 6, 1}	{2, 6, 1}
{1, 3, 7, 9}, {7, 1, 9, 3}	{1, 3, 7, 9}
{1, 2}, {3, 4}	{}
{}, {1, 2, 3}	{}
{1, 2}, {}	{}
{1, 2}, null	null
null, {}	null
null, null	null

39. **Write a function named *countOnes* that returns the number of ones in the binary representation of its argument.**

For example, countOnes(9) returns 2 because the binary representation of 9 is 1001.

Some other examples:

countOnes(5) returns 2 because binary 101 equals 5

countOnes(15) returns 4 because binary 1111 equals 15.

You may assume that the argument is greater than 0.

The function prototype is

int countOnes(int n);

Hint: Use modulo and integer arithmetic to count the number of ones.

40. A **Daphne array** is an array that contains either all odd numbers or all even numbers. For example, {2, 4, 2} (only even numbers) and {1, 3, 17, -5} (only odd numbers) are Daphne arrays but {3, 2, 5} is not because it contains both odd and even numbers. Write a function named *isDaphne* that returns 1 if its array argument is a Daphne array. Otherwise it returns 0.

the function prototype is

int isDaphne (int a[ ], int len) where len is the number of elements in the array.

41. An array is defined to be **odd-valent** if it meets the following two conditions:

    a. It contains a value that occurs more than once

    b. It contains an odd number

    For example {9, 3, 4, 9, 1} is odd-valent because 9 appears more than once and 3 is odd. Other odd-valent arrays are {3, 3, 3, 3} and {8, 8, 8, 4, 4, 7, 2}

    The following arrays are **not** odd-valent:

    {1, 2, 3, 4, 5} - no value appears more than once.

    {2, 2, 2, 4, 4} - there are duplicate values but there is no odd value.

    Write a function name ***isOddValent*** that returns 1 if its array argument is odd-valent, otherwise it returns 0.

    The function prototype is:

    int isOddValent (int a[ ], int len) where len is the number of elements in the array.

42. A **normal** number is defined to be one that has no odd factors, except for 1 and possibly itself. Write a method named *isNormal* that returns 1 if its integer argument is normal, otherwise it returns 0.

Examples: 1, 2, 3, 4, 5, 7, 8 are normal numbers. 6 and 9 are not normal numbers since 3 is an odd factor. 10 is not a normal number since 5 is an odd factor. The function signature is

int isNormal(int n)

43. A non-empty array of length n is called an array of ***all possibilities***, if it contains all numbers between 0 and n - 1 inclusive. Write a method named *isAllPossibilities* that accepts an integer array and returns 1 if the array is an array of all possibilities, otherwise it returns 0.

Examples {1, 2, 0, 3} is an array of all possibilities, {3, 2, 1, 0} is an array of all possibilities, {1, 2, 4, 3} is not an array of all possibilities, (because 0 not included and 4 is too big), {0, 2, 3} is not an array of all possibilities, (because 1 is not included), {0} is an array of all possibilities, {} is not an array of all possibilities (because array is empty).

The function signature is

int isAllPossibilities(int a[ ], int len) where len is the number of elements in the array.

44. An array is defined to be a **Filter array** if it meets the following conditions

a. If it contains 9 then it also contains 11.

b. If it contains 7 then it does **not** contain 13.

So {1, 2, 3, **9**, 6, **11**} and {3, 4, 6, **7**, 14, 16}, {1, 2, 3, 4, 10, 11, 13} and {3, 6, 5, 5, 13, 6, 13} are Filter arrays. The following arrays are **not** Filter arrays: {9, 6, 18} (contains 9 but no 11), {4, 7, 13} (contains both 7 and 13)

Write a function named isFilter that returns 1 if its array argument is a Filter array, otherwise it returns 0.

The function signature is

int isFilter(int a[ ], int len) where len is the number of elements in the array.

45. Write a function named **isDigitSum** that returns 1 if sum of all digits of the first argument is **less than** the second argument and 0 otherwise. For example isDigitSum(32121,10) would return 1 because 3+2+1+2+1 = 9 < 10.

    More examples:

    isDigitSum(32121,9) returns 0, isDigitSum(13, 6) returns 1, isDigitSum(3, 3) returns 0

    The function should return -1 if either argument is negative, so isDigitSum(-543, 3) returns -1.

    The function signature is

    int isDigitSum(int n, int m)

46. A twin is a prime number that differs from another prime number by 2. A ***Fine array*** is defined to be an array in which every prime in the array has its twin in the array. So {4, 7, 9, 6, 5} is a Fine array because 7 and 5 occur. Note that {4, 9, 6, 33} is a Fine array since there are no primes. On the other hand, {3, 8, 15} is not a Fine array since 3 appears in the array but its twin 5 is not in the array.

    Write a function named **isFineArray** that returns 1 if its array argument is a Fine array, otherwise it returns 0.

    The function signature is

    int isFineArray (int a[ ], int len) where len is the number of elements in the array.

You may assume that there exists a function isPrime that returns 1 if its argument is a prime, otherwise it returns 0. **You do not have to write this function.**

47. A balanced array is defined to be an array where for every value n in the array, -n also is in the array. For example {-2, 3, 2, -3} is a balanced array. So is {-2, 2, 2, 2}. But {-5, 2, -2} is not because 5 is not in the array.

    Write a function named ***isBalanced*** that returns 1 if its array argument is a balanced array. Otherwise it returns 0.

    The function signature is

    int isBalanced(int a[ ], int len) where len is the number of elements in the array.

48. An **Evens** number is an integer whose digits are all even. For example 2426 is an Evens number but 3224 is not.
    Write a function named *isEvens* that returns 1 if its integer argument is an Evens number otherwise it returns 0.

    The function signature is

    int isEvens (int n)

49. An array is defined to be a **Magic** array if the sum of the primes in the array is equal to the first element of the array. If there are no primes in the array, the first element must be 0. So {21, 3, 7, 9, 11 4, 6} is a Magic array because 3, 7, 11 are the primes in the array and they sum to 21 which is the first element of the array.{13, 4, 4, 4, 4} is also a Magic array because the sum of the primes is 13 which is also the first element. Other Magic arrays are {10, 5, 5}, {0, 6, 8, 20} and {3}. {8, 5, -5, 5, 3} is **not** a Magic array because the sum of the primes is 5+5+3 = 13. Note that -5 is not a prime because prime numbers are positive.

Write a function named *isMagicArray* that returns 1 if its integer array argument is a Magic array. Otherwise it returns 0.

The function signature is

int isMagicArray (int a[ ], int len) where *len* is the number of elements in the array.

You may assume that a function named *isPrime* exists that returns 1 if its int argument is a prime, otherwise it returns 0. **You do not have to write this function!** You are allowed to use it.

50. An array is defined to be **complete** if the conditions (a), (d) and (e) below hold.
    a. The array contains even numbers

    b. Let *min* be the smallest even number in the array.

    c. Let *max* be the largest even number in the array.

    d. *min* does not equal *max*

    e. All numbers between *min* and *max* are in the array

    For example {-5, 6, 2, 3, 2, 4, 5, 11, 8, 7} is complete because

    a. The array contains even numbers

    b. 2 is the smallest even number

    c. 8 is the largest even number

    d. 2 does not equal 8

    e. the numbers 3, 4, 5, 6, 7 are in the array.

    Examples of arrays that are **not** complete are:

    {5, 7, 9, 13} condition (*a)* does not hold, there are no even numbers.

    {2, 2} condition (*d*) does not hold

    {2, 6, 3, 4} condition (*e)* does not hold (5 is missing)

Write a function named *isComplete* that returns 1 if its array argument is a complete array. Otherwise it returns 0.

The function signature is

int isComplete (int a[ ], int len) where *len* is the number of elements in the array.

51. A **prime product** number is a positive integer that is the product of exactly two primes greater than 1. For example, 22 is a prime product since 22 = 2 times 11 and both 2 and 11 are primes greater than 1. Write a function named *isPrimeProduct* with an integer parameter that returns 1 if the parameter is a prime product, otherwise it returns 0. Recall that a prime number is a positive integer with no factors other than 1 and itself.
    You may assume that there exists a function named *isPrime(int m)* that returns 1 if its m is a prime number. You have to write *isPrime before the isPrimeProduct*.

The function signature

int isPrimeProduct(int n)

52. Write a function *fill* with signature

    int[ ] fill(int[ ] arr, int k, int n)

    which does the following: It returns an integer array arr2 of length n whose first k elements are the same as the first k elements of arr, and whose remaining elements consist of repeating blocks of the first k elements. You can assume array arr has at least k elements. The function should return null if either k or n is not positive.

    Examples:

    fill({1,2,3,5, 9, 12,-2,-1}, 3, 10) returns {1,2,3,1,2,3,1,2,3,1}.

    fill({4, 2, -3, 12}, 1, 5) returns {4, 4, 4, 4, 4}.

    fill({2, 6, 9, 0, -3}, 0, 4) returns null.

53. An array is said to be **hollow** if it contains 3 or more zeroes in the middle that are preceded and followed by the same number of non-zero elements. Write a function named *isHollow* that accepts an integer array and returns 1 if it is a hollow array, otherwise it returns 0

    Examples: isHollow({1,2,4,0,0,0,3,4,5}) returns 1. isHollow ({1,2,0,0,0,3,4,5}) returns 0. isHollow ({1,2,4,9, 0,0,0,3,4, 5}) returns 0. isHollow ({1,2, 0,0, 3,4}) returns 0.

    The function signature is

    int isHollow(int[ ] a, int len)

    where *len* is the number of elements in the array.

54. An array is defined to be a **Bean array** if it meets the following conditions

    a. If it contains a 9 then it also contains a 13.

    b. If it contains a 7 then it does **not** contain a 16.

    So {1, 2, 3, **9**, 6, **13**} and {3, 4, 6, **7**, 13, 15}, {1, 2, 3, 4, 10, 11, 12} and {3, 6, 9, 5, 7, 13, 6, 17} are Bean arrays. The following arrays are **not** Bean arrays:

    a. { 9, 6, 18} (contains a 9 but no 13)

    b. {4, 7, 16} (contains both a 7 and a 16)

    Write a function named isBean that returns 1 if its array argument is a Bean array, otherwise it returns 0.

    The function signature is

    int isBean (int a[ ], int len) where len is the number of elements in the array.

==========

https://www.doc.ic.ac.uk/~wjk/c++intro/RobMillerE5.html
https://exercism.io/tracks/cpp/exercises