ITCR – ANÁLISIS Y DISEÑO DE ALGORITMOS
Tarea 2: prácticas para examen 1

INSTRUCCIONES:

- Requisito para revisar los ejercicios: deben cumplir con estas instrucciones.
- Nombre del programa fuente en C++: **tarea2_su_nombre.cpp. (envíe solo el .cpp)**
- Ponga documentación interna al inicio de cada función, al menos lo que hace la función, entradas y salidas.
- Recuerde usar buenas prácticas de programación como nombres significativos, reutilización de código (usar funciones que ya fueron desarrolladas en este trabajo), documentación interna, etc.
- Para los estatutos de repetición se debe usar el while excepto que el ejercicio indique explícitamente otro tipo de estatuto de repetición.
- Los datos ocupados por las funciones se leen en la función principal para luego enviarlos a la función respectiva. En cada ejercicio se indica si debe hacer algún proceso adicional en la función **main**.
- Las funciones deben tener el nombre que se da en cada ejercicio.
- Dentro de la función usted puede estructurarla según su conveniencia (por ejemplo puede usar otras funciones) pero cumpliendo con lo solicitado.
- Considere que los vectores y las matrices nativas se definen con un tamaño máximo. Los tamaños en cada ejercicio no exceden estos tamaños máximos: vectores con un máximo de 20 elementos y las matrices 20 x 20.
- No valide las restricciones.
- Enviar al tecDigital / EVALUACIONES / TAREAS. Verifique el envío al tecDigital.
- 10 puntos cada ejercicio.
- Fecha de entrega: 29 de agosto, 11pm

RECOMENDACIONES:

- Lea cuidadosamente los problemas que tiene que resolver (primer paso de la metodología de solución de problemas: entender el problema).
- En la etapa de diseño del algoritmo se sugiere hacer un esquema donde determine el comportamiento del algoritmo para luego proceder con su desarrollo en la computadora.

EJERCICIO 0: función main

En la función **main** del programa implemente un menú con estas opciones:

1. Sumatoria de cuadrados
2. Multiplicatoria
3. Pares amigables
4. Calcular factores primos
5. Determinar si un numero es abundante
6. Sumar y restar vectores
7. Multiplicar vectores
8. Sumar diagonal de una matriz cuadrada
9. Números primos y palindromos
10. Triangulo de Pascal
11. Reemplazar números
12. Igualdad de conjuntos
13. Máximos y mínimos de la matriz
14. Extraer números
15. Contar palabras
16. Complemento de conjuntos
17. Trozos iguales de strings
18. FIN

    Opción __

Use un while para manejo del menú, el cual se debe desplegar cada vez que el usuario ocupe ejecutar alguna función.
Use un switch para procesar cada opción que el usuario seleccione.

1. **Sumatoria de cuadrados (sumatoria_cuadrados)**

Esta función implementa una sumatoria de cuadrados empezando desde i = m hasta n.

$$\sum_{i=m}^{n} i^2 = i^2 + (i+1)^2 + (i+2)^2 + \ldots + (i+n)^2$$

En el main lea m y n, luego los envía a la función para que retorne el resultado y también lo imprima en el main. Use la estructura for. Ejemplo del funcionamiento:

m: 2
n: 4
Sumatoria de cuadrados: 29 ← $2^2 + 3^2 + 4^2$

2. Multiplicatoria (multiplicatoria)

Esta función implementa el operador de multiplicatoria. En lugar de sumar cada término, estos se multiplican. Empieze desde k=m hasta n.

$$\prod_{k=m}^{n} a_k = a_m \cdot a_{m+1} \cdot \quad \ldots \quad \cdot a_n$$

En el main lea m y n, luego los envía a la función para que retorne el resultado y también lo imprima en el main. Use la estructura for. Ejemplo del funcionamiento:

m: 3
n: 6
Multiplicatoria: 360 ← 3 * 4 * 5 * 6

3. Pares amigables (pares_amigables)

En matemáticas un par de números *m* y *n* es llamado par amigable (o números amistosos), si la suma de todos los divisores de *m* (excluyendo a *m*) es igual al número *n*, y la suma de todos los divisores del número *n* (excluyendo a *n*) es igual a *m* (donde $m \neq n$).

Ejemplo: los números 220 y 284 son un par amigable. La explicación es la siguiente:

Los únicos números que dividen de forma exacta a 220 (excluyéndolo) son 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 y 110, y la suma de ellos es: 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284

Los únicos números que dividen de forma exacta a 284 (excluyéndolo) son 1, 2, 4, 71 y 142, y la suma de ellos es: 1 + 2 + 4 + 71 + 142 = 220

Desarrolle la función **pares_amigables** que reciba un vector de 10 enteros (cada entero es > 1 y no se repiten en el vector) e imprima la lista con los pares amigables que contiene el vector de entrada. La impresión debe ser similar a lo mostrado en el ejemplo del funcionamiento:

Datos del vector:
100
1184
220
15
18
2
1210
284
3
25

Pares amigables:
1184 y 1210
220 y 284

4. Calcular factores primos (factores_primos)

En teoría de números la factorización de enteros o factorización de primos consiste en expresar un número compuesto (número no primo) como un producto de factores primos. Un procedimiento para obtener esta factorización consiste en tomar el número y dividirlo por el menor número primo posible (2 o 3 o 5, etc.), luego el cociente lo dividimos por el mismo primo de ser posible, sino es posible seguimos probando con el siguiente primo, y así sucesivamente hasta que el cociente sea igual al valor 1. Los divisores calculados son los factores primos. Ejemplos:

120	2
60	2
30	2
15	3
5	5
1	

2 x 2 x 2 x 3 x 5

33	3
11	11
1	

3 x 11

Desarrolle la función **factores_primos** que reciba un entero (>= 2) y calcule su factorización según el procedimiento explicado. La impresión del resultado debe ser igual a lo mostrado en los ejemplos del funcionamiento:
Calcular los factores primos de: 120
2 x 2 x 2 x 3 x 5

Calcular los factores primos de: 33
3 x 11

Calcular los factores primos de: 11
11

Defina esta función después de la función principal del programa. Use la estructura do-while.

5. Determinar si un numero es abundante (numero_abundante)

En matemáticas un número n es abundante si cumple la siguiente propiedad: la suma de los divisores positivos de n incluyéndole es mayor al producto de 2 * n. La abundancia se calcula así: suma de divisores - 2 * n.
Haga la función numero_abundante para determinar si un número es abundante. En el main se lee un entero n (>= 1) el cual se envía a esta función para imprimir los siguientes resultados:

Divisores de n (IMPRESOS EN UNA MISMA LÍNEA)
Suma de los divisores
Es abundante o No es abundante (este mensaje depende de los cálculos)
Abundancia: Cálculo de la abundancia o el mensaje “no hay” en caso de no haber abundancia

Ejemplos del funcionamiento:

Número para el cálculo: 24 (DATO LEÍDO EN MAIN)
Divisores: 1 2 3 4 6 8 12 24
Suma de los divisores: 60
Es abundante // porque la suma de los divisores es mayor a 2 * número
Abundancia: 12 // cálculo de la abundancia

Número para el cálculo: 19
Divisores: 1 19
Suma de los divisores: 20
No es abundante // no cumple la propiedad de abundante
Abundancia: no hay

6. Sumar y restar vectores (sumar_restar_vectores)

Un vector es un conjunto de n elementos continuos que puede ser representado por un arreglo de una dimensión con la forma:

$$v = [\, v_0, v_1, v_2, \ldots, v_{n-1} \,]$$

El acceso a sus elementos es por medio del índice que va de 0 a n-1 (indexación base 0).

La suma y la resta de dos vectores produce como resultado un vector. Para hacer estas operaciones los vectores deben tener el mismo tamaño y se realizan entre los elementos que tienen la misma posición:

$$v = [\, v_0, v_1, v_2, \ldots, v_{n-1} \,]$$
$$w = [\, w_0, w_1, w_2, \ldots, w_{n-1} \,]$$
$$v \pm w = [\, v_0 \pm w_0, v_1 \pm w_1, v_2 \pm w_2, \ldots, v_{n-1} \pm w_{n-1} \,]$$

Haga una función para sumar y restar vectores. En el main lea los siguientes datos y retorne el vector resultante:

- Tamaño de los vectores
- Los valores de cada vector (flotantes)
- Operación (1-Suma, 2-Resta).
- Vector resultante (ESTE VECTOR LO VAMOS A ENVIAR COMO UN PARÁMETRO PARA QUE SEA DEVUELTO CON EL RESULTADO Y LUEGO IMPRESO EN EL MAIN)

Sugerencia:
Haga una función para leer vectores, puede que se ocupe para otros ejercicios. Entradas a la función: tamaño del vector, vector que se completará con los datos leídos.
Desarrolle una función para imprimir vectores, puede que se ocupe para otros ejercicios. Entradas a la función: tamaño del vector, vector a imprimir.

Ejemplo del funcionamiento:
Tamaño de los vectores: 4
Valores del vector:
20
15
30
80
Valores del vector:
70
15
50
45
Operación (1-Suma, 2-Resta): 1
Vector resultante:
90 30 80 125

7. Multiplicar vectores (multiplicar_vectores)

La multiplicación de dos vectores produce como resultado un único valor. Deben tener el mismo tamaño. Multiplicamos los elementos que están en las mismas posiciones dentro de los vectores y luego se suman para dar el resultado lo cual representamos así:

$v = [\, v_0,\ v_1,\ v_2, \ldots ,\ v_{n-1} \,]$

$w = [\, w_0, w_1, w_2, \ldots , w_{n-1} \,]$

$v * w = v_0 * w_0 + v_1 * w_1 + v_2 * w_2 + \ldots + v_{n-1} * w_{n-1} \,]$

Haga una función para multiplicar vectores y retorne el resultado. En el main lea los siguientes datos:

- Tamaño de los vectores
- Los valores de cada vector (flotantes)

Imprima el resultado en el main.

Ejemplo del funcionamiento:

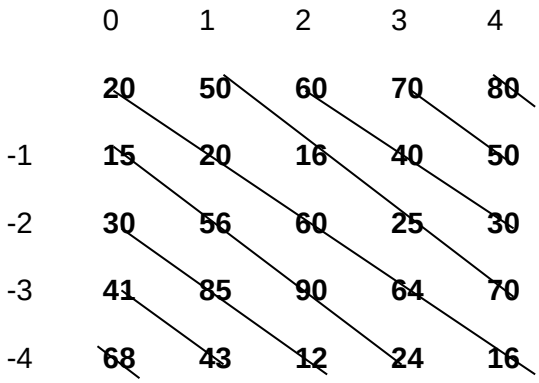
Tamaño de los vectores: 3
Valores del vector:
5
3
2
Valores del vector:
5
5
4
Multiplicacion: 48

8. Sumar diagonal de una matriz cuadrada (sumar_diagonal)

Esta función recibe tres parámetros:

- Tamaño de la matriz cuadrada (entero)
- Una matriz cuadrada (números enteros)
- Un número de diagonal (entero).

La función debe retornar la suma de los elementos que están en el número de diagonal dado en el tercer parámetro. La diagonal principal es la diagonal 0, las diagonales arriba de la principal se numeran de 1 en adelante y las que están debajo de -1 en adelante tal como se muestra a continuación:



Ejemplo del funcionamiento:
Tamaño de la matriz cuadrada: 5
Valores de la matriz:
Fila 1 columna 1: 20
Fila 1 columna 2: 50
Fila 1 columna 3: 60
Fila 1 columna 4: 70
Fila 1 columna 5: 80
Fila 2 columna 1: 15
…
Numero de diagonal: 2
Suma de la diagonal: 130 // IMPRIMIR EN MAIN EL RESULTADO

9. Numeros primos y palíndromos (primos_y_palindromos)

Esta función recibe dos números enteros: el inicio y el fin de un rango (inicio >0, fin > 0, inicio <= fin). La función debe hacer lo siguiente:

- Para los números que están en el rango de entrada hay que imprimir solamente aquellos cuya representación decimal cumpla con estas dos condiciones:
    - que sea un número primo y
    - palíndromo.

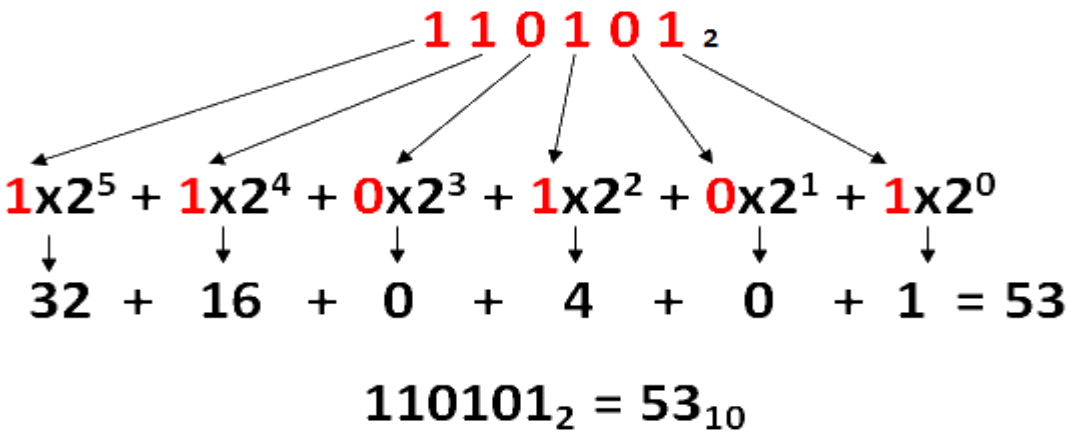
En caso de no haber números con estas condiciones imprimir el mensaje “NO HAY NÚMEROS CUYA REPRESENTACIÓN DECIMAL SEA UN PRIMO Y PALÍNDROMO A LA VEZ”. La impresión debe ser según ejemplos del funcionamiento:

Inicio del rango: 110 // 6 en decimal
Fin del rango: 1101 // 13 en decimal
Números que son primos y palíndromos a la vez
7
11

Inicio del rango: 101101001 // 361 en decimal
Fin del rango: 1011101110 // 750 en decimal
Números que son primos y palíndromos a la vez
373 es primo y palíndromo
383 es primo y palíndromo
727 es primo y palíndromo

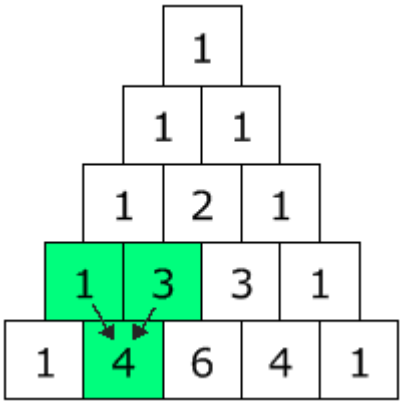
Inicio del rango: 1100 // 12 en decimal
Fin del rango: 1110 // 14 en decimal
Números que son primos y palíndromos a la vez
NO HAY NÚMEROS CUYA REPRESENTACIÓN DECIMAL SEA UN PRIMO Y PALÍNDROMO A LA VEZ

La conversión de un número binario a su representación decimal se debe realizar con el siguiente procedimiento: cada dígito del número a convertir se multiplica por una potencia de 2. El último dígito del número, es decir el que está más a la derecha, se multiplica por 2 con potencia 0, el penúltimo dígito se multiplica por 2 con potencia 1 y así sucesivamente se va aumentando la potencia de 2 según la posición de los dígitos. La suma de todas las multiplicaciones da el número convertido. Esto se muestra en el ejemplo:



10. Triangulo de Pascal (triangulo_de_pascal)

La función **triángulo_de_pascal**. recibe un entero n (> 0 y < tamaño máximo de las matrices para esta práctica) y retorna una matriz con el triángulo de pascal (Blaise Pascal, representan coeficientes binomiales ordenados en forma triangular). Este triángulo empieza con un 1 en la cima. También se pone un 1 en los extremos de cada línea: a la izquierda y a la derecha. Los números dentro del triángulo representan la suma de los dos números adyacentes que están encima según se muestra en la figura siguiente para n = 5. Otra forma de ver los números dentro del triángulo: la suma horizontal de dos números da el número debajo de los mismos.



n
1
2
3
4
5

Ejemplo del funcionamiento:

Nivel del triángulo: 5
Retorna una matriz como esta que se imprime en el main. Cada nivel es una fila de la matriz y se imprime en una línea separada.
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1

11. Reemplazar números (reemplazar)

Desarrolle la función **reemplazar**. Recibe tres parámetros numéricos enteros desde el main: busca todas las apariciones (de izquierda a derecha) del primer número en el segundo número y las cambia por el tercer número. Retorna el resultado para imprimir en el main. Ejemplos del funcionamiento:

Buscar: 457
En el numero: 28457
Reemplazar por: 30
Resultado: 2830

Buscar: 22
En el numero: 2228
Reemplazar por: 4
Resultado: 428

Buscar: 5580
En el numero: 55808055804
Reemplazar por: 9
Resultado: 98094

Buscar: 64
En el numero: -647648164
Reemplazar por: 395
Resultado: -395739581395

Buscar: 12345
En el numero: 123
Reemplazar por: 30
Resultado: 123

12. Igualdad de conjuntos (igualdad)

Un conjunto matemático está formado por un grupo de elementos (números, letras, colores, figuras, etc.) no repetidos. En este caso vamos a representar conjuntos de dígitos por medio de números enteros. Ejemplos de estos conjuntos: 62841, 1029. Desarrolle esta función booleana que recibe dos números enteros (> = 0) y retorna el valor booleano true si esos conjuntos son iguales o false si son diferentes. La igualdad de conjuntos está definida así: dos conjuntos (en este caso dos números) son iguales si y solo si tienen los mismos elementos sin interesar su orden. Ejemplos del funcionamiento:
Conjunto 1: 62841
Conjunto 2: 14682
Conjuntos iguales

Conjunto 1: 62841
Conjunto 2: 362841
Conjuntos NO iguales

Conjunto 1: 82614
Conjunto 2: 62841
Conjuntos iguales

Conjunto 1: 64812
Conjunto 2: 81624
Conjuntos iguales

Conjunto 1: 1029
Conjunto 2: 108
Conjuntos NO iguales

13. Máximos y mínimos de una matriz (maximos_y_minimos)

La función recibe estos parámetros:
- Cantidad de filas de la matriz (entero)
- Cantidad de columnas de la matriz (entero)
- Matriz del tamaño anterior (enteros).

En la función se imprime por cada fila y por cada columna el número máximo y el número mínimo.

Ejemplo del funcionamiento:

Filas: 3
Columnas: 5

Suponiendo esta matriz:
10 15 25 40 20
13 98 45 60 51
30 30 78 99 39

Los resultados serían:
Fila 1: máximo 40 minimo 10
Fila 2: máximo 98 minimo 13
Fila 3: máximo 99 minimo 30

Columna 1: máximo 30 minimo 10
Columna 2: máximo 98 minimo 15
Columna 3: máximo 78 minimo 25
Columna 4: máximo 99 minimo 40
Columna 5: máximo 51 minimo 20

14. Extraer números (extraer_numeros)

Desarrolle la función **extraer_numeros** que reciba un string e imprima los números presentes en el string recibido. Cada número presente debe formarlo en una variable y cuando completa el número esa variable es impresa. Los números no tienen signo y son enteros. En caso de no encontrar números imprimir solo la palabra “Ninguno”. Los resultados se imprimen en la función.

Ejemplos del funcionamiento:

String a analizar: xy 225pa.b0g89iou ab789 25
Numeros encontrados:
225
0
89
789
25

String a analizar: abcd n950x8 c76n
Numeros encontrados:
950
8
76

String a analizar: abc.d n c mn
Numeros encontrados:
Ninguno

15. Contar palabras (contar_palabras)

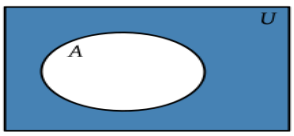
Desarrolle la función **contar_palabras** que reciba dos vectores (arreglos de 1 dimensión) de strings: uno con palabras (5 elementos) y otro con frases (3 elementos). Debe retornar la cantidad de veces que aparece cada palabra en el vector de frases. Imprima los resultados en la función principal según el ejemplo del funcionamiento:

Vector de palabras: aprender
computadoras
lenguaje
C++
De

Vector de frases: en este curso aprendemos el lenguaje de programación C++
C++ es un lenguaje de programación de computadoras
C++ tiene una clase de nombre string

Resultados:
Cantidad de veces que aparece cada palabra en las frases:
aprender: 0
computadoras: 1
lenguaje: 2
C++: 3
de: 4

16. Complemento de conjuntos (complemento)



Recibe dos strings que representan conjuntos de símbolos: el primer string es un conjunto de caracteres (U de conjunto universo) y el segundo string es un subconjunto de U (A). La función debe retornar el complemento de A, es decir, aquellos caracteres que están en U pero no están en A y el valor booleano true. Hacer esta validación: el conjunto A debe ser un subconjunto de U, es decir, todos los elementos de A tienen que estar en U, de lo contrario retorna un string con los símbolos de A que no pertenecen a U y el valor false.

Ejemplos del funcionamiento:

Conjunto universo: tpabzeyx
Subconjunto: ay
Complemento: tpbzex
true ← significa que pudo aplicar el complemento

Conjunto universo: abc12
Subconjunto: xa1y5
Símbolos que no estan en el conjunto universo: xy5
false ← significa que no pudo aplicar el complemento porque A no es subconjunto de U

17. Trozos iguales de strings (trozos_iguales)

Desarrolle la función **trozos_iguales** que reciba un vector de strings y retorne de ese vector todos los trozos que tengan la longitud de trozo máxima . Un trozo es un grupo de elementos seguidos que tienen el mismo valor. El vector de strings termina cuando se encuentre un elemento con el valor “fin”.

Por ejemplo, si tenemos el vector de entrada:

{ “xyz”, “xyz”, “m”, “a”, “a”, “a”, “a”, “j”, “j”, “x1”, “x1”, “x1”, “x1”, “fin” }

aquí hay 5 trozos (1-xyz xyz, 2-m, 3-a a a a, 4- j j 5- x1 x1 x1 x1), de ellos hay 2 que tienen la máxima cantidad de elementos que es 4 (el trozo 3 y el trozo 5), así el valor retornado es el vector { “a”, “a”, “a”, “a”, “x1”, “x1”, “x1”, “x1” }

Los trozos se retornan en el mismo orden de aparición de izquierda a derecha. Si el trozo está repetido solo se retorna el primero. Imprimir el resultado en la función main.

Ejemplos del funcionamiento:

Vector de strings: xyz, xyz, m, a, a, a, a, j, j, x1, x1, x1, x1, fin
Resultados:
a, a, a, a
x1, x1, x1, x1

Vector de strings: z, xb, xb, xb, m, a, a, a, w, a, a, x, x, x, j, j, j, a, fin
Resultados:
xb, xb, xb
a, a, a
j, j, j

Última línea