PRÁCTICA

INSTRUCCIONES:

- Ponga documentación interna de cada función (por ejemplo: lo que hace la función, entradas, salidas).
- Usar buenas prácticas de programación como nombres significativos, reutilización de código (funciones), documentación interna, eficiencia, etc.
- No es necesario leer los datos que se van a procesar, puede ponerlos directamente en el programa principal como se hizo en el examen 2 y en otros trabajos.

RECOMENDACIONES:

- Lea cuidadosamente el problema a 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.

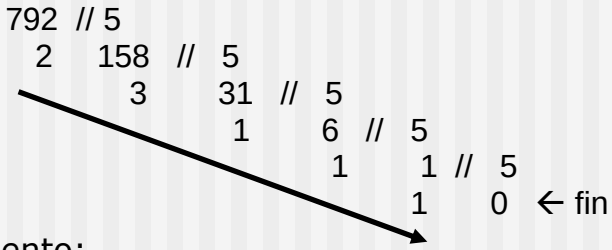
1) Programa base10_a_otras_bases.

Desarrollar la función **base10_a_otras_bases** para convertir número expresado en base decimal a a otras bases numéricas

La función recibe estos parámetros numéricos naturales en base decimal:

- un número
- una posición inicial
- una posición final
- base final (2, 3, 4, 5, 6, 7, 8, 9)

Proceso: debe extraer del número (primer parámetro), aquel valor representado por los dígitos que están entre la posición inicial y la posición final inclusive. El dígito de más a la izquierda en el primer parámetro está en la posición 1, el dígito que le sigue a su derecha en la posición 2 y así sucesivamente. Ese número extraído hay que convertirlo a la base numérica indicada por el parámetro base final y retornarlo. Para hacer la conversión de base 10 a otra base usaremos el método de divisiones sucesivas: consiste en realizar divisiones del número decimal entre la base final hasta que el cociente sea cero. El primer residuo se convierte en el dígito menos significativo y el último residuo en el más significativo del número convertido. Por ejemplo el número decimal 792 se convierte en el número quinario (base 5) 11132 de la siguiente forma:



Ejemplos del funcionamiento:

Numero: 187924
Posición inicial: 3
Posición final: 5
Base final: 5
Resultado de la conversión: 11132 (conversión de 792 a base 5)

Numero: 61309
Posición inicial: 2
Posición final: 4
Base final: 8
Resultado de la conversión: 202 (conversión de 130 a base 8)

Imprima el resultado en el main.

2) Programa clase_Estudiante.

Implementación de TDA usando clases. Usando la clase Estudiante que fue revisada en las lecciones:

- Definir interfaz: copie el encabezado de la clase Estudiante y genere el archivo de encabezado: Estudiante.h

- Definir implementación: copie la implementación de la clase Estudiante y genere el archivo de implementación de la clase: Estudiante.cpp

- Desarrollar estos pasos en la función main para usar la clase:
  - ✓ Crear objetos (al menos 4 objetos)
  - ✓ Asignar valores
  - ✓ Desplegar valores
  - ✓ Almacenar los objetos en un vector nativo.
  - ✓ Desarrollar una función en este main –NO UN MÉTODO DE LA CLASE-que tome los objetos del vector y los imprima accediendo directamente las variables miembro de Estudiante.

3) Programa numeros.

- Desarrollar la función **extraer_numeros** que reciba un string (use <string> de la biblioteca estándar de C++) y extraiga los números presentes en ese string. Cada número será un elemento de una lista (use <forward_list>). Los números no tienen signo, pueden ser enteros o flotantes. En caso de ser flotantes tienen al menos un dígito a la izquierda y otro a la derecha del punto decimal. A los lados de lo números puede encontrar cualquier símbolo no numérico. La función retorna ese vector.

- Desarrollar la función **extraer_numeros_imprimir** que reciba ese vector y lo imprima. En caso de que el vector esté vacío, se imprime el mensaje “No hay números”.

- Ejemplos del funcionamiento:

String a analizar: xy 225pa.b3.1415g89iou1000 ab78.9 25
Numeros encontrados:
225
3.1415
0
89
1000
78.9
25

String a analizar: a0.25bcd n950x8 c76n
Numeros encontrados:
0.25
950
8
76

String a analizar: abc.d n c mn
Numeros encontrados:
No hay números

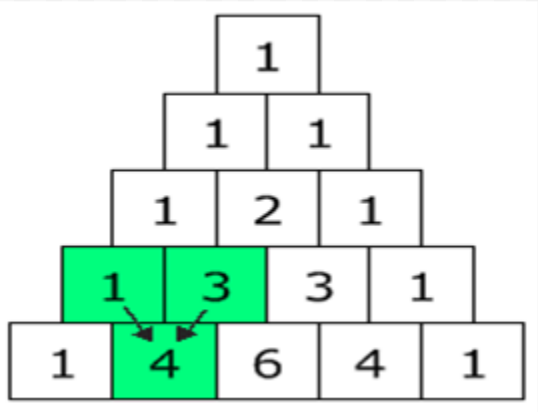
4) Programa triangulo_de_pascal. Use el componente <vector>.

- Desarrollar la función **triangulo_de_pascal**. Recibe la cantidad de niveles requeridos del triángulo (entero >= 1) y retorna un vector representando el triángulo. Cada elemento del vector retornado es un nivel del triángulo de Pascal (Blaise Pascal, representan coeficientes binomiales ordenados en forma triangular) y a la vez cada nivel es otro vector con los valores de dicho nivel. Este triángulo empieza con dos niveles fijos: el nivel 1 tiene solo el número 1, el nivel 2 tiene los números 1 y 1. Los siguientes niveles se calcularán basados en el nivel anterior: se pone un 1 en los extremos del nuevo nivel, a la izquierda y a la derecha, luego la suma horizontal de cada par de números adyacentes del nivel anterior dan los números debajo de ellos para el nuevo nivel. Esto se muestra en el ejemplo del funcionamiento para 5 niveles.

- Desarrollar la función **triangulo_de_pascal_imprimir**. Recibe un vector con la estructura anterior e imprime el triángulo representado según ejemplo.

Niveles del triángulo de pascal: 5

```
Nivel 1:                                1
Nivel 2:                      1                   1
                                      1 + 1
Nivel 3:                1                   2                   1
                                 1 + 2               2 + 1
Nivel 4:          1                   3                   3                   1
                           1 + 3               3 + 3               3 + 1
Nivel 5:     1                   4                   6                   4                   1
```



Una posible solución:

```
// definir dos estructuras de datos

vector<int> nivel;   // vector para numeros de un nivel, cuando el vector es completado para un nivel

vector < vector<int> > triangulo;  // este se agrega al vector con todos los niveles

// repetir por cada nivel:
//            empezar el nivel sin elementos, vacío, se puede hacer con nivel.clear()
//            construir el nivel en el vector nivel, se basa en el nivel anterior
//            agregar el vector nivel al vector triangulo

// imprimir vector triángulo usando el for basado en rango
```

5) Programa semiprimos.

Los números primos y semiprimos son altamente utilizados en los campos de la criptografía (codificación de datos) y de la teoría de números. Los números primos son los enteros >= 2 que tienen exactamente dos divisores: el 1 y él mismo, entre ellos: 2, 3, 5, etc. Los números semiprimos (o biprimos) son los números que se forman al multiplicar dos números primos, ejemplos:

4 es semiprimo, se forma al multiplicar los primos 2 y 2

6 es semiprimo, se forma al multiplicar los primos 2 y 3

39 es semiprimo, se forma al multiplicar los primos 3 y 13, etc.

Desarrollar la función **semiprimos** que reciba un intervalo (2 enteros >= 2, el primero menor que el segundo) y retorne una lista (use el componente <list>) donde cada elemento va a tener información de cada semiprimo encontrado en ese intervalo. La información de cada semiprimo contiene tres datos como en el ejemplo:

+ el número semiprimo encontrado
+ el primer número primo que lo conforma
+ el segundo número primo que lo conforma

En caso de no haber semiprimos retorna una lista vacía.

Desarrollar otra función para imprimir el resultado retornado anteriormente.

Ejemplo del funcionamiento: suponiendo que recibe el intervalo de 14 a 24, la lista retornada sería:

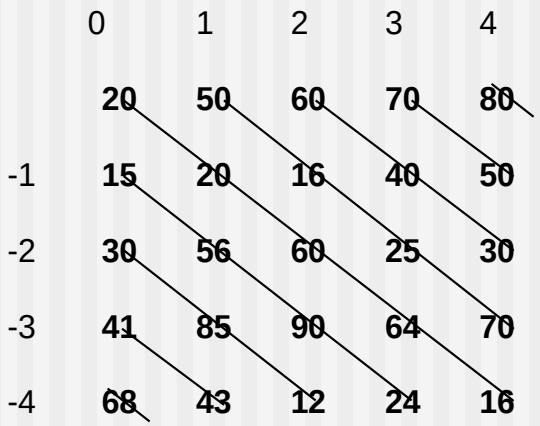
{ {14, 2, 7}, {15, 3, 5}, {21, 3, 7}, {22, 2, 11} }

6) Programa extrae_diagonal.

Desarrollar la función **extrae_diagonal** que reciba:

- Una matriz cuadrada de tamaño m x m, la matriz es un arreglo nativo o un arreglo del componente <array> de dos dimensiones, usted lo decide.
- El tamaño de esta matriz.
- Un número de diagonal.

La función debe retornar un vector (use componente <vector>) con los valores de la diagonal indicada. 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:



Ejemplos del funcionamiento usando esta matriz para un tamaño 5 x 5:

Para la diagonal 0 el vector resultante es:
{ 20, 20, 60, 64, 16 }

Para la diagonal 2 el vector resultante es:
{ 60, 40, 30 }

Para la diagonal -3 el vector resultante es:
{ 41, 43 }

Si la diagonal no existe retorna un vector vacío:
{ }

Aunque este ejemplo es para una matriz de tamaño 5 x 5, el programa debe funcionar para tamaños máximos de 100 x 100.

7) Programa numeros_largos.

Para representar números enteros naturales muy largos (no sabemos la cantidad de dígitos) vamos a usar el componente <list>. Una lista de este tipo almacenará un número donde cada elemento de la lista representará un dígito del número. El primer elemento (de la izquierda) contiene el dígito más a la izquierda del número y el último elemento contiene el dígito más a la derecha.

Desarrollar la función **sumar_números_largos** que reciba dos enteros naturales, los convierta a estas listas y calcule su suma usando otra lista. Ejemplo:

Número 1: 875

Número 2: 12348

Lista del número 1:	8	7	5		
Lista del número 2:	1	2	3	4	8
Lista resultante:	1	3	2	2	3

Para hacer la operación debemos ir recorriendo las listas de atrás hacia adelante e ir generando a la vez otra lista con los dígitos del resultado, es decir, debemos empezar por los elementos que tienen los dígitos más a la derecha de cada número. Para este ejemplo son los elementos que contienen los valores 5 y 8 que sumados dan 13, el 3 se guarda inmediatamente en la lista resultante como dígito más a la derecha y se acarrea 1 a la siguiente suma de dígitos, que en este caso son los elementos que contienen los valores 7 y 4, a los cuales se les suma 1 que viene de la operación anterior lo cual daría 12, el 2 se guarda en la lista resultante en el elemento respectivo y se acarrea 1 a la siguiente suma, y así sucesivamente hasta completar la suma de todos los dígitos. Los números de entrada pueden tener diferentes longitudes.

Para no repetir código haga también estas funciones:

convertir_a_lista. Recibe un número y lo convierte a una lista. Llame esta función 2 veces: para convertir cada número recibido.

imprimir_lista. Recibe una lista y la imprime. Llame esta función 3 veces: para desplegar las listas de los 2 números recibidos y para la lista resultante.

- Revisar otras prácticas para el examen:
  - Ejercicios y ejemplos en materiales del curso.
  - Evaluaciones (tareas, quices, prácticas, ejemplos, exámenes, programas, etc.).
  - Otros materiales pertinentes, digitales o impresos.

“En la vida hay algo peor que el fracaso: el no haber intentado nada”

Frankling Roosevelt