EIF207 – Estructuras de Datos
Proyecto de programación #1

Prof. Georges E. Alfaro Salazar

OBJETIVOS DEL PROYECTO

El objetivo del proyecto es aplicar los conceptos teóricos, principios y técnicas estudiadas en clase. Es importante que al desarrollar el proyecto se escriba código correctamente estructurado y encapsulado. El proyecto busca en particular que los estudiantes implementen estructuras de datos no elementales y que realicen un análisis adecuado de la eficiencia de los algoritmos utilizados (tanto en tiempo como en espacio).

DESCRIPCIÓN DEL PROYECTO

Se construirá una biblioteca de clases para hacer cálculos aritméticos con valores enteros de precisión arbitraria (precisión múltiple).

Los tipos elementales provistos por la mayoría de los lenguajes de programación para el manejo de valores enteros (`short`, `int`, `long`) están limitados por la arquitectura del computador. Así, por ejemplo, en C++ los valores declarados de tipo `int` son representados en memoria utilizando 32 o 64 bits de memoria, según el tamaño de los registros generales provistos por la CPU del computador. Esto restringe el valor de los números enteros que pueden ser representados. Con registros de 32 bits, pueden representarse valores entre -2^{31} y $2^{31}-1$.

Las clases por construir permitirán manejar valores enteros (positivos y negativos) sin ninguna restricción en la cantidad de dígitos permitidos (limitados únicamente por la cantidad de memoria disponible).

FUNCIONALIDADES POR IMPLEMENTAR

La clase para representar un número entero de longitud arbitraria se denominará **`Integer`**.

La clase (o clases) a desarrollar implementará las siguientes funcionalidades:

- Constructor de conversión para objetos de tipo `int` y `long`.
- Operaciones aritméticas básicas (suma, resta, multiplicación y división)
- Operaciones de comparación
- Sobrecarga de los operadores aritméticos (+, - *, /)
- Sobrecarga de los operadores de asignación (=, +=, -=, *=, /=)
- Sobrecarga de los operadores de comparación (==, !=, <, <=, >, >=)
- Declaración de constantes para los valores 0 (`ZERO`) y 1 (`ONE`)
- Conversión de hileras a instancias de la clase (`parse()`)
- Método `toString()`, para convertir una instancia en una hilera imprimible. Observe que el método `toString()` es el inverso del método de conversión `parse()`. Es decir, para una

variable *x* de tipo `Integer`:

```
x == Integer::parse(x.toString())
```

No deberá implementar operadores unarios ni operadores lógicos (&&, ||, !) o de patrones de bits (&, |, ~, >>, ^). Puede sobrecargar el operador << para poder convertir directamente instancias de la clase a hileras (`string`). Observe que, ya que la clase se implementará utilizando una estructura de datos con memoria asignada de manera dinámica, es necesario definir el destructor, el constructor de copia y sobrecargar el operador de asignación.

Se deben incluir métodos de prueba para verificar cada una de las operaciones solicitadas.

Escriba además funciones para calcular:

Función factorial:	$n! = \begin{cases} 1 & \text{si } n = 0 \\ n(n-1)! & \text{si } n > 0 \end{cases}$
Secuencia de Fibonacci:	$F(n) = \begin{cases} n & \text{si } n < 2 \\ F(n-1) + F(n-2) & \text{si } n \geq 2 \end{cases}$
Número de combinaciones de k elementos de un conjunto de tamaño n: https://es.khanacademy.org/math/statistics-probability/counting-permutations-and-combinations	$C(n,k) = \binom{n}{k} = \frac{n!}{k!\,(n-k)!}$

La implementación de estas funciones puede ser iterativa o recursiva. Investigue sobre las propiedades de cada función para ver si existen formulaciones alternativas que permitan efectuar el cálculo de manera más eficiente.

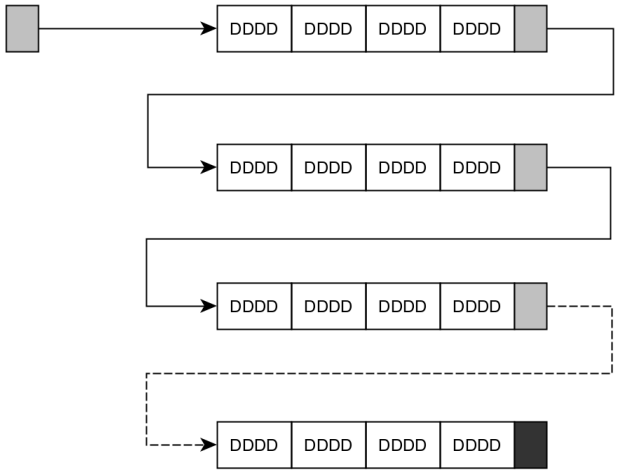
Aplicando las funciones anteriores, calcule:

```
1000!
F(1000)
C(1000, 350), C(1000,500), C(1000,650)
```

Adjunto al enunciado, encontrará un archivo con los valores correctos para cada cálculo, que puede utilizar como referencia para comprobar sus resultados.

CONSIDERACIONES DE IMPLEMENTACIÓN

La estructura de datos a emplear será una lista enlazada (simple o doble) de arreglos, donde el tipo base del arreglo será una clase que permita almacenar una cierta cantidad de dígitos. El valor de cada elemento será un valor entero binario, es decir, no se guardarán caracteres para los dígitos de cada celda.



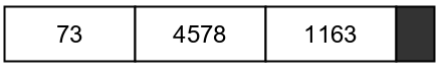
Por ejemplo, cada celda del arreglo podría contener un valor de tipo `long`. Una variable `long` tiene un tamaño de al menos 32 bits (4 bytes). Esto quiere decir que se puede representar en una variable de este tipo un número entero entre -2^{31} y $2^{31}-1$, o un valor entero positivo entre 0 y $2^{32}-1$. Esto permitiría guardar hasta 9 dígitos decimales en cada celda del arreglo. Si se usaran caracteres para representar cada dígito, se necesitaría un poco más del doble de la memoria empleada de la otra forma (9 bytes).

Observe el siguiente ejemplo:

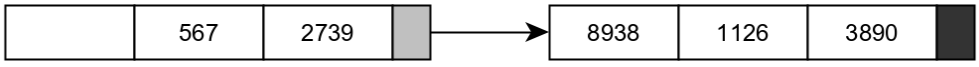
Suponga que en cada celda del arreglo se almacenarán 4 dígitos decimales. Para esto, puede usarse una variable de tipo entero (int), que necesita solamente 2 bytes, en lugar de usar 4 caracteres (4 bytes). Una variable unsigned int podría almacenar un valor en el rango:

$$[0 \cdots 2^{16} - 1] = [0..65535]$$

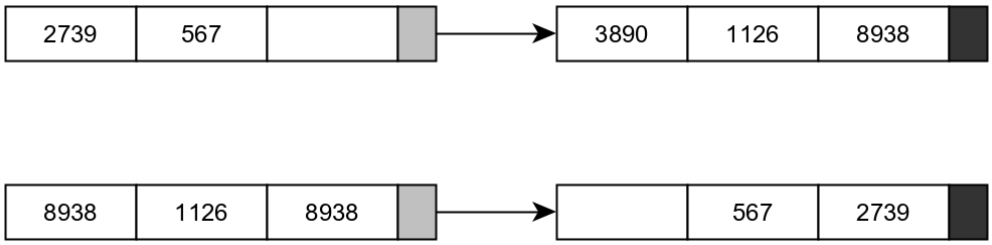
En un arreglo de 3 posiciones, se representarían hasta 12 dígitos. Si se emplea una lista enlazada de arreglos, cada uno con 3 posiciones, un número como 7345781163 quedaría en memoria de la siguiente manera:



Pero un número como 5672739893811263890 no podría guardarse en solamente 3 posiciones. La idea es, entonces, crear una lista enlazada de arreglos para contener el resto de los dígitos:



Observe que la posición de los elementos del arreglo o de cada uno de los arreglos dentro de la lista es arbitraria. Cualquiera de las siguientes representaciones podría ser válida, por ejemplo:



La posición relativa de los elementos puede interpretarse de la manera en que sea conveniente para la implementación.

Hay que considerar el uso óptimo de la memoria para definir los parámetros de la estructura: si se almacenan pocos elementos en cada arreglo, se necesitará asignar más memoria a los punteros de la lista. Pero si se hacen arreglos muy grandes, es posible que se desperdicie espacio cuando haya entradas en el arreglo que no se utilicen. De la misma manera, es importante determinar que tipo de variable se utilizará para almacenar el valor en cada posición del arreglo.

Para definir la estructura de datos puede basarse en las clases de plantilla vistas en clase para implementar una pila o cola por medio de listas enlazadas. Si necesita una implementación alternativa, usando listas doblemente enlazadas o arreglos, pude construir las plantillas correspondientes. Observe que la estructura usada en la clase `Integer` NO es una plantilla, sino una instancia que depende del tipo de elemento seleccionado para resolver el problema.

Debe incluir un breve análisis para determinar el tamaño óptimo de:

- El tamaño (número de bytes) de cada celda (b)
- El número de celdas en cada arreglo (n)

También debe escribir una fórmula $d(b,n) =?$ para saber cuál es el desperdicio (en promedio) o espacio libre en la estructura.

El código del proyecto debe estar estructurado adecuadamente, respetando principios básicos de diseño.

En la entrega, se deberán incluir los diagramas de clase y los archivos de proyecto correspondientes. Cada profesor indicará cual es el ambiente de desarrollo (IDE) a utilizar. El proyecto se desarrollará en el lenguaje C++. Es particularmente importante que el manejo de memoria dinámica sea correcto (asignación y recuperación).

Los archivos de proyecto tienen que entregarse en el formato adecuado para su revisión. No se recibirán proyectos en un formato diferente al indicado. Los diagramas de clase podrían elaborarse usando UMLet (http://www.umlet.com/) u alguna otra aplicación apropiada y se guardarán usando el formato propio de la aplicación y en PDF.

Los diagramas de clase no son adecuados para mostrar la estructura de datos utilizada en el proyecto, así que incluya también los **diagramas de objetos** correspondientes.

https://www.tutorialspoint.com/uml/uml_object_diagram.htm

https://www.geeksforgeeks.org/unified-modeling-language-uml-object-diagrams/

En particular:

- NO se puede utilizar ninguna biblioteca de terceros que implemente la funcionalidad solicitada.
- **NO deberá usar ninguna de las colecciones implementadas en la biblioteca estándar de plantillas de C++** (*Standard Template Library* – STL).
- Las clases deben separar correctamente la declaración de la interfaz y su implementación, usando archivos de cabecera (archivos .h) y código fuente (archivos .cpp) por aparte. Los archivos de cabecera deberán siempre usar guardas (se puede especificar la directiva `#pragma once` cuando se utilice Visual Studio).
- Las clases de entidad no deberán contener código de entrada/salida (como salida a la consola usando `cout`, por ejemplo)
- El manejo de la interfaz con el usuario debe hacerse por medio de una o varias clases diseñadas para tal efecto.

Es importante que el diseño general del programa considere la reutilización de código. En todos los casos, las colecciones y otras estructuras deben implementarse de la forma más general posible. En particular, la implementación de las colecciones (pilas, colas y listas) deberá usar plantillas (*templates*).

REFERENCIAS.

Puede consultar las siguientes referencias:

https://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic

https://en.wikipedia.org/wiki/Division_algorithm

https://cp-algorithms.com/algebra/big-integer.html

https://en.cppreference.com/w/cpp/language/types

ENTREGA Y EVALUACIÓN

El proyecto debe entregarse **por medio del aula virtual, en el espacio asignado para ello, o de la manera solicitada por el profesor del curso**. La fecha de entrega es el día **viernes 30 de septiembre de 2022**. No se aceptará ningún proyecto después de esa fecha, ni se admitirá la entrega del proyecto por correo electrónico. **El proyecto puede completarse en grupos de 2 a 3 personas. Alguno de los integrantes de grupo entregará un archivo de texto con el detalle de los participantes, a más tardar el día 13 de septiembre de 2022. No se permitirá que ninguna persona se integre posteriormente a cualquier grupo de trabajo.**

Incluya comentarios en el código de los programas y describa cada una de las clases y métodos utilizados cuando sea útil y conveniente.

El código del programa debe compilar correctamente. Si el código no puede compilarse, el trabajo será calificado con una nota de 0 (cero). Para ser evaluado, tiene que cumplirse al menos un 50% de la funcionalidad solicitada. De lo contrario, el proyecto será considerado como no entregado y se penalizará la nota correspondiente.

Durante la revisión del proyecto, es muy importante defender adecuadamente la solución propuesta.

Se evaluarán los siguientes aspectos:

Documentación	**(5%)**
Aspectos formales (presentación general, estructura, redacción y ortografía)	
Descripción de la solución	
Correctitud y completitud de los diagramas	
Estructura	**(30%)**
Clases e implementación de estructuras (listas)	
Funcionalidad (general)	**(20%)**
Control de errores	
Manejo correcto de memoria	
Funcionalidad (específica)	**(45%)**
Representación, constructores y métodos de conversión	
Operaciones aritméticas y lógicas. Sobrecarga de operadores.	
Operadores de asignación.	
Cálculo de funciones (factorial, Fibonacci, combinaciones)	

Observaciones generales:

- Los proyectos deben entregarse con toda la documentación, diagramas, código fuente y cualquier otro material solicitado.
- Los trabajos no se copiarán de ninguna llave USB u otro dispositivo en el momento, sino que se deben entregar en el formato adecuado.
- **Cualquier trabajo práctico, que no sea de elaboración original del estudiante y haya sido copiado o adaptado de otro origen (plagio), de manera parcial o total, se calificará con nota 0 (cero) y se procederá como lo indiquen los reglamentos vigentes de la universidad.**