

PROGRAMACIÓN I

PARCIAL 1

**EVALUACIÓN PARCIAL – RESOLUCIÓN DE CASOS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| FACULTAD: | **Tecnología Informática** | | | | |
| CARRERA: | **Analista Programador** | | | | |
| ALUMNO/A: |  | | | | |
| SEDE: | **Buenos Aires** | | LOCALIZACIÓN: | **Online** | |
| ASIGNATURA: | **Programación I** | | | | |
| CURSO: | **1O** | | TURNO: |  | |
| PROFESOR: | **Hunt, Alejandro** | | FECHA: |  | |
| TIEMPO DE RESOLUCIÓN: | | **Según cronograma** | EXAMEN PARCIAL NRO: | | **1** |
| MODALIDAD DE RESOLUCIÓN: | | | Virtual Domiciliario | | |

**Criterios de calificación:** Para acreditar los saberes deberá obtener, al menos, el 60% de los aspectos conceptuales, y al menos, el 60% de los aspectos procedimentales.

**Criterios de resolución:** Los alumnos recibirán la consigna del examen en la fecha de evaluación prevista por el cronograma de la asignatura. Deberán resolver y entregar este examen previo al vencimiento indicado en la actividad. En caso de que el mismo sea entregado luego del vencimiento, se procederá a una penalización en el puntaje final, la cual dependerá del tiempo de demora.

**Criterios de evaluación:** Se evaluará la claridad en el planteamiento de los aspectos conceptuales y procedimentales. En los aspectos procedimentales, no solo se evaluará que el desarrollo provea una solución correcta a la consigna planteada, sino también la claridad, eficiencia y aplicación de los conceptos vistos en la cursada.

**Resultados de aprendizaje:**

* Comprender los conceptos más importantes sobre la programación visual.
* Conocer todos los aspectos relacionados al IDE de Visual Studio.
* Poder desarrollar programas informáticos en C#
* Poder resolver un problema por medio de la utilización de listas enlazadas, pilas y colas, comprendiendo cual es el tipo de estructura dinámica más adecuada para utilizar.
* Comprender e implementar las operaciones que aplican a las estructuras dinámicas según su tipo.
* Comprender el concepto de TDA y poder aplicarlo en la resolución de problemas.
* Comprender el concepto de recursividad y poder aplicarlo en la resolución de problemas.

La evaluación se hará a partir de la siguiente grilla:

|  |  |  |
| --- | --- | --- |
| **Aspectos Conceptuales** | | |
| Pregunta 1 |  |  |
| Pregunta 2 |  |  |
| Pregunta 3 |  |  |
| Pregunta 4 |  |  |
| Pregunta 5 |  |  |
| **Aspectos procedimentales** | | |
| Desarrollo del punto 1 |  |  |
| Desarrollo del punto 2 |  |  |
|  | | |
| Calificación final |  |  |

# Forma de entrega del examen

Se deberá entregar un archivo comprimido (.zip, .rar, .7z) que contenga, un documento con la resolución de todos los aspectos conceptuales y una solución con dos proyectos con la resolución de los aspectos procedimentales. En este último punto, ambos proyectos deberán incluir el código fuente desarrollado, el cual deberá compilar con la posibilidad de ser ejecutado para realizar pruebas.

El archivo entregado deberá tener el siguiente formado:

* Materia\_comisión\_2021\_1P\_Apellido\_Nombre.ZIP E**jemplo**:
* Programacion1\_1O\_2021\_1P\_Hunt\_Alejandro.ZIP

# Aspectos conceptuales

1. En Windows Forms, todos los controles tienen 3 características principales, enumérelas y explique como se relacionan entre sí.

Eventos, Métodos, Propiedades. El cómo se relacionan entre sí viene dado por lo que es cada uno. Esta explicación la va a entender aquél que ha visto funcionar controles de usuario (de otro modo, resulta abstracta).

Hagamos una analogía entre lo que se conoce de procedimientos, funciones y variables:

Los eventos y métodos son equivalentes a los procedimientos y funciones. La diferencia es que están estrechamente vinculados a un objeto específico.

Desde este punto de vista, los eventos (a su vez) también son métodos, pero con la diferencia de que el evento tiene sus propios disparadores (que llaman a la ejecución automática del código pertinente).

Finalmente, las propiedades son simplemente variables que, a diferencia de las variables comunes, están estrechamente vinculadas a un control (de alguna manera, son su estado en análogía a los componentes de react).

**Resumiendo:** éstas 3 características de los controles de usuario se relacionan entre sí por el objeto que tienen en común: el control de usuario mismo (y es porque cada control de usuario ya viene instanciado de la clase que les provee por default con este “paquete” de eventos, métodos y propiedades).

**Los métodos definen el comportamiento**

1. ¿¿Que es un TDA?? Explique su principal característica y que ventajas posee.

Los programadores han tenido que luchar con el problema de la complejidad durante mucho tiempo. Mejor dicho, desde el nacimiento mismo de la informática. Los programadores (hasta donde sabemos) son humanos, y los humanos han desarrollado una técnica excepcional y potente para tratar la complejidad: la abstracción (y que es: ante la incapacidad de dominar en su totalidad los objetos complejos, se ignoran los detalles no esenciales tratando en su lugar con el modelo ideal del objeto y centrándose en el estudio de sus aspectos esenciales).

En ello que re-definimos las cosas así, un TDA viene a ser un modelo matemático de los objetos de datos que constituyen un tipo de dato, así como de las funciones que operan sobre estos objetos (es necesario recalcar que las operaciones que manipulan los objetos están incluidas en las especificaciones del TAD). ¿Abstracto, no?

Es decir: un Tipo de Dato Abstracto (un TDA) es un conjunto de datos (u objetos) al cual se le asocian operaciones. El TDA provee de una interfaz con la cual es posible realizar las operaciones permitidas, abstrayéndose de la manera en como estén implementadas dichas operaciones (pero eso es otro tema, uno de la POO: encapsulamiento, y que, en última instancia, significa que un mismo TDA puede ser implementado utilizando distintas estructuras de datos mientras provee de la misma funcionalidad).

**En resumen:** un Tipo de Dato Abstracto (TDA) es un modelo que define valores y las operaciones que se pueden realizar sobre ellos.

1. ¿¿Qué ventajas y desventajas podemos encontrar en la utilización de Listas enlazadas sobre la utilización de Vectores??

Los elementos que se pueden insertar en listas enlazadas son en número indefinido, cosa que no es posible en un vector que, una vez lleno, se debe agrandar de tamaño, que es una operación costosa (y que puede no ser posible si la memoria está fragmentada).

Por otro lado, un vector del cual se eliminan muchos elementos puede quedar inútilmente vacío o hacerlo más pequeño (y ahí estamos en el mismo escenario descripto anteriormente).

Sin embargo, los vectores (y matrices) permiten el acceso aleatorio, mientras que las listas enlazadas solo permiten el acceso secuencial. Incluso el acceso secuencial en los arreglos (vectores y matrices) también es más rápido que en las listas enlazadas debido a la ubicación de las cachés de referencia y datos. Las listas enlazadas casi no reciben ningún beneficio del caché.

Las listas enlazadas necesitan de un almacenamiento adicional para las referencias. Esto las hace, a menudo, poco prácticas para listas de elementos de datos pequeños (como caracteres o valores booleanos).

1. ¿¿Que significa que una función sea recursiva y a que tipo de estructura suele ser una alternativa?? Indique a su vez las ventajas y desventajas de utilizar una función recursiva sobre su alternativa.

En pocas (poquísimas) palabras, las funciones recursivas son funciones que se llaman a sí mismas mientras son ejecutadas.

Deduzco que “estructura” se refiere a estructura de control del paradigma estructurado. Así que contestaré basado en entender que la pregunta se refiere al uso de la recursión en lugar de la iteración. En ese sentido, la respuesta no es tan sencilla como parece pues pertenece a un plexo propio de las ciencias computacionales. Trataré de exponerlo en forma breve y sencilla:

En el paradigma funcional se prioriza el uso de recursividad y aplicación de funciones de orden superior para resolver problemas que en otros lenguajes se resolverían mediante estructuras de control (por ejemplo, iteración). ¿Por qué? Porque se busca eliminar el efecto secundario (en oposición, la programación imperativa enfatiza los cambios de estado mediante la mutación de variables). Esto significa que dos expresiones sintácticas idénticas pueden producir resultados distintos si dependen de algo que ha mutado: es decir, carecen de transparencia referencial. En el paradigma funcional, el valor generado por una función depende exclusivamente de los argumentos alimentados a la función: al eliminar los efectos secundarios, se puede entender y predecir el comportamiento de un programa mucho más fácilmente. Esta es una de las principales motivaciones para utilizar la programación funcional (y de que se respeten tanto sus posiciones teóricas).

Un aspecto inherente de la recursividad es el costo de gestión. Consideremos las implicancias: cuando tiene lugar el llamado de una función, la secuencia de ejecución (flujo de control) cambia, y se da un salto hacia el código que representa dicha función, pero este cambio no es tan simple como parece: el ámbito de una función es un nuevo contexto en todos los sentidos: hay nuevas variables, hay nuevas instrucciones y sentencias a procesar, etc. Todas ellas se ejecutan en el mismo microprocesador y utilizan el mismo conjunto de registros, por lo que antes de ejecutar la primera línea de código de una función, se debe respaldar el estado de todos los registros y la dirección de retorno (y todo otro elemento que se utilizará para procesar el nuevo contexto).

Cuando la función termina, antes de regresar el control a donde estaba antes del llamado (ámbito del invocador), se deben restaurar todos los datos respaldados en los registros para que no exista ninguna afectación en la lógica de ejecución y de procesamiento de datos del ámbito del invocador. Es decir, la ejecución debe continuar como si nunca se hubiera cambiado el flujo de ejecución.

Aunque la descripción anterior es una versión muy simplista y resumida de lo que pasa con cada llamado de función, lo importante es tener en mente que atrás del llamado de una función sucede mucho, y a eso es a lo que se le llama (desde el punto de vista computacional) costo de gestión de funciones (overhead).

Las implementaciones recursivas son en general ineficientes en tiempo, pero elegantes y simples una vez que se tiene identificada la relación de recurrencia. En este sentido, siempre que se pueda optar por un enfoque iterativo que sea sencillo de implementar, es mejor quedarse con él.

Pero por otro lado, el problema de Las Torres de Hanoi es un ejemplo ineludible de que, por medio de la recursividad, es posible solucionar problemas complejos de manera bastante sencilla, aunque el precio a pagar sea la eficiencia en tiempo (y recursos).

Basta intentar resolver el problema de Las Torres de Hanoi de manera iterativa para tener una mejor comprensión de lo anterior y experimentar en carne propia el eterno conflicto (tradeoff) computacional entre espacio versus tiempo, eficiencia versus simplicidad.

1. Está realizando el desarrollo de un juego donde el personaje principal debe encontrar la salida de un laberinto, y se desean guardar los jugadores que fueron saliendo, manteniendo el orden de salida de los mismos y con la posibilidad de obtener en algún momento cual fue el primero que salió y los que vinieron después. Solo tiene la posibilidad de guardar los jugadores en una estructura dinámica de las que se vieron en la materia, cual utilizaría y porqué?? (indique que ventajas posee la utilización de dicha estructura)

Por su definición misma: la cola, que justamente se define por “primero entra, primero sale” (FIFO). Las razones, nuevamente, vienen dada por su particularidad inherente: en una estructura de datos de cola, solo podemos acceder al primer y al último elemento de la estructura. Asimismo, los elementos solo se pueden eliminar por el principio y solo se pueden añadir por el final de la cola.

La función primordial de una queue es la de mantener una secuencia ordenada (la fila de un banco, por ejemplo). En términos sencillos, se podría ilustrar con un ul versus un ol del html. O bien una lista como esta:

1. Nodo 1
2. Nodo 2
3. Nodo 3
4. (etcétera)

Técnicamente, las colas brindan su mayor beneficio cuando se tienen editores y suscriptores separados, pero en términos prácticos, (de nuevo) la principal ventaja es que la adición o eliminación de elementos se puede hacer de forma rápida y eficiente: simplemente se agregan elementos al final de la cola o se eliminan del principio de la cola.

# Aspectos procedimentales

1. Desarrolle un programa que posea un control para cargar un nombre, un control para cargar un número, un radiobutton que muestre 2 opciones (rojo, azul), y un botón. Al presionar el botón, el nombre cargado debe aparecer en un label, el cual será del color seleccionado si el número es positivo y del color no seleccionado si el número es negativo. En caso de que el número ingresado sea cero, se debe mostrar una ventana diciendo que no se puede ingresar el valor cero y que seleccione otro número.
2. Desarrolle un programa en windows forms, que posea un control para poner un nombre, y un botón para agregar dicho nombre a un TDA que usted desarrolle. Se podrán agregar tantos nombres como se deseen al TDA, y además existirá en pantalla un contador que indicará la cantidad de nombres agregados. Los nombres repetidos se contarán como nombres distintos. Además, existirá un botón que permitirá mostrar en una ventana, cual fue el último nombre agregado al TDA. Seleccione el TDA que considere más eficiente para la resolución del ejercicio.