Preguntas APREF

RESOLUCION DE PROBLEMAS. CONCEPTOS Y DEFINICIONES

1- ¿Qué es la informática?

La informática es la ciencia que estudia el análisis y resolución de problemas utilizando computadoras.

2- ¿Cuáles son las etapas en la resolución de un problema?

- Análisis del problema: Se analiza el problema en su contexto con el mundo real. Deben obtenerse los requerimientos del usuario y el resultado de esta etapa es un modelo preciso del ambiente del problema y del objetivo a resolver.
- Diseño de una solución: En esta etapa se define una estructura de sistema de hardware y software que lo resuelva. El primer paso es la modularizacion del problema, es decir, la descomposición del mismo en partes que tendrán una función bien definida y datos propios. Estas partes deben estar comunicadas.
- Implementación (escritura del programa): Un algoritmo es una especificación simbólica que debe convertirse en un programa real sobre un lenguaje de programación concreto. Debe ser traducido (automáticamente) al lenguaje de la maquina que lo vaya a ejecutar, esta traducción, denominada compilación, permite detectar y corregir los errores sintácticos que se cometan en la escritura del programa.
- Verificación: en esta etapa se verifica que la ejecución del programa conduzca al resultado deseado, con datos representativos del problema real. Sin embargo, en los casos reales es muy difícil realizar una verificación exhaustiva de todas las posibles condiciones de ejecución de un sistema de software.

3- ¿Cómo especificarías las tareas involucradas en cada etapa?

Para el análisis y el diseño podría especificar tareas como entender el problema, modularizarlo, y escribir los algoritmos. Y para la implementación y verificación entraría tareas como codificar los algoritmos en algún lenguaje de programación, compilarlo, hacer prueba de ejecución y verificar que cumpla con lo deseado.

4- ¿Qué es un algoritmo?

Un algoritmo es la especificación rigurosa de la secuencia de pasos (instrucciones) a realizar sobre un autómata para alcanzar un resultado deseado en un tiempo finito.

Especificación rigurosa significa que debemos expresarnos en forma clara y univoca. El termino autómata viene de computadora. Y que sea en tiempo finito significa que un algoritmo comienza y termina.

5- ¿Qué es un programa?

Un programa es un conjunto de instrucciones, ejecutables sobre una computadora, que permite cumplir una función específica (dichas ordenes están expresadas en un lenguaje de programación concreto).

6- ¿Cuáles son las partes que componen un programa?

Los componentes básicos de un programa son las instrucciones y los datos.

Las instrucciones representan las operaciones que ejecutara la computadora al interpretar el programa. Y los datos son los valores de información de los que se necesita disponer, y en ocasiones transformar para ejecutar la función del programa.

7- ¿Cómo definimos el concepto de lenguaje de programación?

Es un lenguaje formal diseñado para expresar procesos que pueden ser llevados a cabo por maquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento lógico y físico de una maquina, para expresar algoritmos con precisión, o como modo de comunicación humana.

8- ¿A que llamamos ambiente de programación?

Es el entorno en el cual el programador desarrolla sus aplicaciones. Hay dos ambientes reconocidos, uno visual donde se manipulan los objetos y se generan conexiones. Y otro de tipo código, donde se generan las instrucciones de programación a base de ir escribiendo y compilando.

9- ¿Qué significa que un lenguaje sea fuertemente tipado?

Significa que no permite violaciones de los tipos de datos, es decir, dado el valor de una variable de un tipo concreto, no se puede usar como si fuera de otro tipo distinto a menos que se haga una conversión.

10- ¿Considera que Pascal es un lenguaje fuertemente tipado?

Si, se caracteriza por eso ya que el tipo de datos de todas las variables debe ser declarado previamente para que su uso quede habilitado.

11- ¿Qué es una precondición y una postcondicion?

Una precondición es una información que se conoce como verdadera antes de iniciar el programa. Y una postcondicion es una información que debiera ser verdadera al concluir el programa, se se cumple adecuadamente el requerimiento pedido.

12- ¿Cómo ejemplificaría cada una de ellas?

Un ejemplo para la precondición seria calcular el factorial de un número (solo está definido para valores positivos o cero), por lo tanto, en el algoritmo la precondición exigirá que dicho número sea mayor o igual que cero.

Un ejemplo para la postcondicion seria calcular el resultado de un factorial (es siempre un entero mayor o igual que uno), por lo tanto, en el algoritmo la postcondicion exigirá que el resultado debe ser un entero y que este debe ser mayor o igual que uno.

ESTRUCTURAS DE CONTROL

1- ¿A que llamamos estructura de control?

Conjunto mínimo de instrucciones que permiten especificar el control del algoritmo que se quiere implementar. Este conjunto debe contener como mínimo una secuencia, una decisión, una selección, una repetición o una iteración.

2- ¿Qué estructuras de control conoces?

- Secuencia: representada por una sucesión de operaciones (asignaciones, etc.), en la que el orden de ejecución coincide con el orden físico de aparición de las instrucciones.
- Decisión: instrucción no secuencial que permite tomar decisiones en función de los datos del problema. Estructura IF (cond.) THEN, si la condición es verdadera ejecuta las instrucciones correspondientes, si es falsa se va por el ELSE y ejecuta las instrucciones.
- Selección: permite realizar una o más acciones, dependiendo de cual de todas las condiciones evaluada es verdadera. Estructura CASE (variable) OF, (se puede utilizar también el IF). La variable debe ser de tipo ordinal.
- Repetición: consiste en repetir un número fijo y conocido de veces una o más acciones. Estructura FOR (índice:valor_inicial) DO/DOWNTO (valor_final) DO. El índice debe ser de tipo ordinal (entero, boolean, char) y del tipo del identificador, no debe modificarse del lazo. Los incrementos o decrementos son implícitos. Al terminar el ciclo, la variable índice no tiene un valor definido.
- Iteración: ejecuta un bloque de instrucciones desconociendo el número exacto de veces que se ejecutan.
  Pre condicionales: evalúa la condición y luego ejecuta el bloque de instrucciones. Se ejecuta de 0 a más veces, mientras la condición sea verdadera. Estructura WHILE (cond.) DO
  Post condicionales: ejecuta el bloque de acciones y luego evalúan la condición. Se ejecuta de 1 a más veces, mientras la condición sea falsa. Estructura REPEAT (bloque de acciones) UNTIL (cond.).

3- ¿Cómo clasificarías las estructuras de control que conoces en repetitivas, iterativas, precondicional y postcondicional?

DATOS Y TIPOS DE DATOS. ALCANCE DE LOS DATOS

1- ¿Qué es un dato?

Es una representación de un objeto en el mundo real mediante la cual podemos modelizar aspectos del problema que se quiere resolver con un programa sobre una computadora (en forma binaria).
Conceptualmente pueden ser constantes (no cambian su valor durante la ejecución) o variables (pueden cambiar).

2- ¿Cómo diferencias el concepto de variable y de constante?

En las constantes su valor no cambia con la ejecución del programa y el valor se indica en su declaración. En cambio en las variables su valor va cambiando con la ejecución del programa y su valor se indica en el programa.

3- **¿A que llamamos variables locales y variables globales?**
Las variables globales son las que pueden ser accedidas en cualquier parte del programa.
En cambio, las variables locales solo pueden ser accedidas por el modulo en las que se le declara.

4- **¿Cómo definirías el concepto de tipo de datos?**
Un tipo de dato es una clase de objetos de datos ligados a un conjunto de operaciones para crearlos y manipularlos. Se caracterizan por un rango de valores posibles, su representación interna y por un conjunto de operaciones realizables sobre ese tipo.
Se clasifican en datos simples y datos compuestos:
- Los datos simples son los que toman solamente un único valor y están acotados por el lenguaje, pueden ser de tipo numérico (integer y real), de tipo lógico (boolean) o de tipo carácter (char).
- Los datos compuestos son los que toman más de un valor y están definidos por el usuario, por ejemplo los registros, arreglos, listas, árboles o conjuntos.

5- **¿Por qué crees que es útil tener tipos de datos?**
Porque generan flexibilidad (solo debemos modificar una declaración en lugar de una serie de declaraciones de variables), ayuda en la documentación (facilitando el entendimiento y la lectura del programa) y trasmite seguridad (se reducen los errores de correspondencia entre el valor que se pretende asignar a una variable y el previamente declarado).

6- **¿Qué diferencias hay entre los tipos de datos estándar de un lenguaje, los tipos definidos por el usuario y los tipos de datos abstractos?**

7- **¿Qué entiendes por ocultamiento y protección de datos?**
Conocido como Data Hidding se utiliza para significar que todo dato que es relevante para un modulo debe ocultarse de los otros módulos. De esta manera se evita que en el programa principal se declares datos que solo son relevantes para algún modulo en particular y, además, se protege la integridad de los datos.

8- **¿Cómo defines el concepto de alcance de los datos?**
Así como se declaran módulos dentro del programa principal, puede declararse módulos dentro de otros módulos. Por lo tanto, deben aplicarse de reglas que gobiernan el alcance de los identificadores:
- El alcance de un identificador es el bloque de programa donde se lo declara.
- Si un identificador declarado en un bloque es nuevamente declarado en un bloque interno al primero, el segundo bloque es excluido del alcance de la primera sección.

MODULARIZACION. PARAMETROS. POSIBILIDADES EN PASCAL

1- **¿Cómo defines el concepto de modularizacion?**
Significa dividir un problema en partes funcionalmente independientes, que encapsulen operaciones y datos. En otras palabras seria separar en funciones lógicas con datos propios y datos de comunicación perfectamente especificados.

2- **¿Cuáles son las características y ventajas de la modularizacion?**
- Mayor productividad: al dividir un sistema de software en módulos funcionalmente independientes, un equipo de desarrollo puede trabajar simultáneamente en varios módulos, incrementando la productividad.
- Reusabilidad: posibilidad de utilizar repetidamente el producto de software desarrollado. Naturalmente la descomposición funcional que ofrece la modularizacion favorece el re-uso.
- Facilidad de mantenimiento correctivo: la división lógica de un sistema en módulos permite aislar los errores que se producen con mayor facilidad. Esto significa poder corregir los errores en menor tiempo y disminuye los costos de mantenimiento de los sistemas.

- Facilidades de crecimiento del sistema: los sistemas de software reales crecen (aparecen con el tiempo nuevos requerimientos del usuario). La modularizacion permite disminuir los riesgos y costos de incorporar nuevas prestaciones a un sistema en funcionamiento.
- Mayor legibilidad: mayor claridad para leer y comprender el código fuente. El ser humano maneja y comprende con mayor facilidad un número limitado de instrucciones directamente relacionados.

3- ¿Qué relación encuentras entre el diseño top down y la modularización?

El diseño top down y la modularización son similares, las dos se encargan de dividir un programa en módulos más pequeños para reducir su complejidad. Ir de lo general o lo particular. Dividir, conectar y verificar.

4- ¿Una buena modularización asegura la corrección de un programa? ¿Por qué?

No, facilita su corrección pero no nos asegura de que el programa cumpla con su especificación. Para saber si un programa es correcto hay varias formas de comprobarlo, una seria verificar su comportamiento viendo si cumple con las pre y pos condiciones del programa íntegramente.

5- ¿Una buena modularización asegura la eficiencia de un programa? ¿Por qué?

No, porque la eficiencia se mide a través del tiempo de ejecución y la memoria utilizada del algoritmo. Podemos tener dos algoritmos que cumplen la misma función y están los dos modularizados pero uno es más eficiente que el otro.

6- ¿Cómo defines el concepto de modulo?

Es un tarea específica bien definida, se comunican entre si adecuadamente y cooperan para conseguir un objetivo común. Cada módulo encapsula acciones, tareas o funciones. En otras palabras, es un conjunto de instrucciones.

7- ¿Qué módulos reconoce Pascal?

Pascal reconoce dos tipos de módulos, los procedimientos y las funciones.

8- ¿Qué diferencias hay entre dichos módulos?

Procedimiento: es un módulo que realiza una tarea específica y retorna 0, 1 o más valores por parámetros. La comunicación es a través de parámetros por valor o referencia. Sin parámetros no devuelve nada.
Función: es un módulo que realiza una tarea específica y retorna siempre un valor de tipo simple (real, integer, char, boolean). Para devolver el resultado se asigna el nombre de la función como ultima instrucción. Puede recibir solo parámetros de entrada.

9- ¿Todo procedimiento puede ser transformado en una función? ¿Por qué?

No. Porque la tarea que realiza un procedimiento puede devolver más de un valor y puede tener parámetros por referencia.

10- ¿Toda función puede ser transformada en un procedimiento? ¿Por qué?

Sí. Porque la tarea que realiza una función puede devolver un valor y puede tener parámetros por valor.

11- Defina el concepto de pasaje de parámetros.

El concepto de pasajes de parámetros hace referencia a la manera que tienen los módulos de comunicarse, estos parámetros se envían desde la invocación del módulo (parámetros actuales) y se comunican con los parámetros formales (son los declarados en el encabezado del módulo). Pueden ser parámetros por valor o parámetros por referencia.

12- ¿A que llamamos parámetros por valor y por referencia?

Parámetros por valor: es llamado parámetro IN y significa que el modulo recibe (sobre una variable local) una copia de un valor proveniente de otro modulo o del programa principal. Con él puede realizar operaciones y/o cálculos, pero no producirá ningún cambio ni tampoco tendrá incidencia fuera del módulo.
Parámetros por referencia: es un dato que contiene la dirección de memoria donde se encuentra la información compartida con otro módulo o programa que lo invoca. El modulo puede operar con ella y su valor original dentro del módulo, y las modificaciones que se produzcan se reflejan en los demás módulos que conocen la variable.

13- ¿Qué argumento puedes dar para justificar el hecho de que en las funciones todos los parámetros deberían pasarse por valor?

Una función puede usarse solo parámetro por valor porque es módulo que realiza una única tarea y devuelve un solo valor de tipo simple.

14- **¿A que llamamos parámetros formales y actuales?**
Parámetros actuales: son los parámetros que se definen en el llamado del módulo.
Parámetros formales: son los parámetros descritos en el encabezado del módulo invocado (procedimiento o función).

RECURSION

1- **¿Cómo defines el concepto de recursión?**
Resuelve problemas por resolución de instancias más pequeñas del mismo problema, es decir, el problema siempre es el mismo pero de menor tamaño. La recursividad se da cuando se obtienen soluciones a problemas en los que una función o procedimientos se llaman a sí mismo para resolver el problema, entonces se obtienen subprogramas recursivos. Tiene un caso base o degenerado.
2- **¿Qué características debe reunir un problema para que su solución pueda ser resuelta utilizando recursión?**
Debe existir al menos un valor del parámetro sobre el que se hace la recursión que no provoca un nuevo cálculo recursivo: caso base. Un conjunto de casos llamados recurrentes que son los que si producen un nuevo cálculo recursivo.
3- **¿Qué características debe reunir un problema para que su solución pueda ser resuelta utilizando recursión?**
En una solución recursiva se debe tener en cuenta 3 características:
    - Como definir el problema en términos de un problema más pequeño del mismo tipo.
    - Como será disminuido el tamaño del problema en cada llamado recursivo.
    - Qué instancia del problema servirá como caso base.
4- **¿Cuáles son los elementos que deben aparecer en toda solución recursiva?**
Los elementos en una recursión son:
    - Caso base.
    - Auto invocación.
    - Alguna situación que cumpla que se alcance el caso base.
5- **¿Crees que toda solución recursiva puede ser escrita en forma iterativa? ¿Por qué?**
Cualquier cálculo recursivo puede ser expresado como una iteración, porque una solución iterativa se puede resolver de la misma manera que una recursiva pero sin respetar sus operaciones de auto invocación, caso base y alguna situación que garantice que se alcanzó el caso base.
6- **¿El código de una solución recursiva es más corto que el de una solución iterativa?**
El código de una solución recursiva ocupa más memoria y requiere más tiempo de ejecución que una solución iterativa, pero en cuanto a líneas de código es más corto uno recursivo.
7- **¿Una solución recursiva es más legible que una solución iterativa?**
Si, un problema recursivo posee una solución más legible si se resuelve mediante un algoritmo recursivo.
8- **¿Todos los algoritmos recursivos pueden ser resueltos también en forma iterativa?**
Se puede escribir cualquier algoritmo recursivo en forma iterativa usando un bucle y el caso base como corte del bucle. Lo que no se puede hacer siempre es escribir una solución iterativa en forma recursiva.
9- **¿Puede existir más de un caso base, puedes ejemplificar?**
Si verdadero, ejemplo en las búsquedas dicotómicas y capicúa de vectores.

ESTRUCTURAS DE DATOS. DEFINICION. CLASIFICACION

1- **¿Qué es una estructura de datos?**
Es un conjunto de variables (no necesariamente del mismo tipo) relacionadas entre sí de diversas formas y que se pueden operar como un todo, bajo un nombre único.
2- **¿Qué estructura de datos conoces?**
Registro, Arreglos, Listas, Arboles, Pila y Cola.
3- **¿Cómo clasifica a las estructura de datos que conoce?**

Por tipo de dato: simples (entero, carácter, etc.), compuestas (registros, pilas, etc.)
Por contenido: homogéneas (pilas, colas, etc.), heterogéneas (registro, etc.)
Por alocación en memoria: estáticas (arreglos, etc.), dinámicas (listas, arboles, etc.)
Por tipo de acceso: indexado/directo (arreglos, etc.), secuencia (listas, etc.)
Por la relación e/ sus elem: Lineales (listas, pilas, colas etc.), no lineales (arboles, grafos, etc.).

4- **¿Cuándo se dice que una estructura de datos es dinámica y estática?**
Estructura de datos estática: cuando la cantidad de elementos que contienen es fija, es decir, si la cantidad de memoria que se utiliza no varía durante la ejecución de un programa.
Estructura de datos dinámica: cuando el número de componentes y, por lo tanto, la cantidad de memoria, puede variar durante la ejecución de un programa.

5- **¿A que llamamos estructuras de datos homogéneas y heterogéneas?**
Estructura de datos homogénea: cuando los datos que la componen son todos del mismo tipo.
Estructura de datos heterogénea: cuando los datos que la componen son de distinto tipo.

6- **¿A que llamamos estructuras de datos lineales y no lineales?**
Estructura de datos lineales: los elementos tienen una relación 1 a 1. Por ejemplo listas, pilas y colas.
Estructura de datos no lineales: los elementos tienen una relación 1 a muchos, muchos a 1 o muchos a muchos. Por ejemplo árboles y grafos.

REGISTRO

1- **¿Cómo define el tipo de dato registro?**
Un registro es un conjunto de valores con estas características básicas: 1) es una estructura heterogenea. 2) los valores almacenados en un registro son llamados campos, y cada uno de ellos tiene un identificador; los campos son nombrados individualmente, como variables ordinarias. 3) es una estructura estática. 4) el acceso a sus campos es directo.

2- **¿Cómo es el acceso a los campos?**
Para acceder al campo de un registro es necesario especificar tanto el nombre del registro como el del campo al que se desea hacer referencia. Esto es lo que se denomina calificar el campo. La sentencia en Pascal seria: {nombre_variable_registro.nombre_campo} donde nombre_variable_registro es el nombre de una variable de un tipo registro y nombre_campo es el campo del registro que se quiere acceder.

3- **¿Es importante el orden de los campos?**
No, ya que el acceso se hace a través de su nombre y no por su posición.

4- **¿Qué tipos de datos pueden contener los campos?**
Pueden tener datos de tipo entero, real, string, etc.

ARREGLOS

1- **¿Cómo defines la estructura de datos arreglo?**
Es una colección de elementos que se guardan consecutivamente en la memoria, con las siguientes características: 1) es una estructura homogénea. 2) los elementos pueden recuperarse en cualquier orden, simplemente indicando la posición que ocupan dentro de la estructura; por este motivo es indexada. 3) es una estructura dinámica.

2- **¿Cuáles son sus operaciones?**
Las operaciones de los arreglos son lectura/escritura, recorrido, asignación, actualización (añadir, eliminar, insertar), ordenación, búsqueda.

3- **¿Cómo describirías la operación de agregar, insertar y borrar un elemento de un vector?**
Cuando realizamos la operación de agregar un nuevo elemento a continuación del último valor, la única condición necesaria es comprobar que haya espacio para el nuevo elemento.
La operación de insertar un elemento consiste en introducir dicho elemento en el interior del vector, es una posición i dada, de forma tal que los elementos ubicados en las siguientes posiciones sean desplazados a las posiciones i + 1 respectivas.

La operación de eliminar un elemento al final no presenta ningún problema, en cambio, si el borrado se realiza en el interior del mismo esto provoca un corrimiento contrario al de insertar (i – 1) para reorganizar el vector.

4- ¿A que llamamos dimensión lógica y física del arreglo?

La dimensión lógica hace referencia al tamaño máximo ocupado de elementos dentro del arreglo, puede variar dependiendo si se agregan o eliminan elementos.

La dimensión física hace referencia al tamaño fijo del arreglo, declaro al principio del programa, y es la cantidad de elementos que pueden entrar dentro del arreglo. No varía.

5- ¿De qué tipo de dato se pueden declarar los elementos y el índice de un arreglo?

Los elementos de un arreglo pueden ser de tipo entero, real, lógico, carácter, enumerativo, registro, etc.
Y su índice tiene que ser un tipo de dato ordinal (entero, carácter, enumerado, etc.)

6- ¿Cómo es el acceso a los elementos de un arreglo?

Para poder acceder a un elemento de un arreglo debemos hacerlo a través de su índice. Acceso indexado.

7- ¿Cuáles son los distintos tipos de arreglos y sus características? ¿Cómo se declaran en Pascal?

Arreglos unidimensionales o vector: su característica es que tiene solamente un índice.

```
{Type
        Vector = array [rango] of tipo;}
```

Arreglos bidimensionales o matriz: su característica es que tiene dos índices.

```
{Type
        Matriz = array [fila, columna] of tipo;}
```

Arreglos multidimensionales: más de dos índices.

8- ¿Qué métodos de ordenación conoces?

Método de selección: para ordenar un vector de n elementos, a[1] .. a[n], busca el elemento menor según el criterio adoptado y lo ubica al comienzo. Para mantener la dimensión del vector lo intercambia con a[1]. A continuación se busca el segundo elemento menor del vector y se intercambia con a[2]. Continúa así sucesivamente buscando el próximo menor y lo coloca en su lugar, hasta que todos los elementos quedan ordenandos. Para ordenar un vector completo es necesario realizar n-1 pasadas por el vector.

Método de intercambio o burbujeo: procede de una manera similar al de selección, en el sentido de que realiza n-1 pasadas. Pero a diferencia del anterior, realiza correcciones locales de distancia 1, es decir, compara ítems adyacentes y los intercambia si están desordenados. Al final de la primera pasada, se habrán comparado n-1 pares y el elemento más grande habrá sido arrastrado, como una burbuja, hacia el final del vector. Al final de la pasada, el segundo máximo habrá sido ubicado en la posición n-1.

Método de inserción: ordena el vector de entrada insertando cada elemento a[i] entre los i-1 anteriores que ya están ordenados. Para realizar esto comienza a partir del segundo elemento, suponiendo que el primero ya está ordenando. Si los dos primeros elementos están desordenados, los intercambia. Luego, toma el tercer elemento y busca su posición correcta con respecto a los dos primeros.

9- ¿Qué tipo de búsquedas se pueden aplicar?

Búsqueda lineal: es el proceso de buscar desde el principio de la estructura, analizando los elementos que contienen uno a uno hasta encontrarlo o hasta llegar al final. Requiere excesivo consumo de tiempo y es ineficiente en estructuras muy grandes.

Búsqueda dicotómica o binaria: el vector tiene que estar ordenado y se basa en la estrategia de "divide y vencerás". Los pasos son, calcula la primer y última posición para obtener el medio, luego compara si el elemento buscado es igual al que se encuentra en el medio, en caso de no ser el buscado, si el elemento buscado es menor se continúa la búsqueda en la primera mitad del vector, en caso contrario sobre la segunda mitad. El proceso termina cuando se encuentra el elemento buscado o cuando ya no hay elementos por comparar.

LISTAS

1- ¿Cómo defines la estructura de datos lista?

Una lista es un conjunto de elementos de tipo homogéneo, donde los mismos no ocupan posiciones secuenciales o contiguas de memoria, es decir, los elementos pueden aparecer físicamente dispersos en la memoria, si bien mantienen un orden lógico interno. Es una estructura lineal.

2- ¿Cuáles son sus operaciones?

Recorrer una lista.
Acceder al k-esimo elemento de la lista.
Intercalar un nuevo elemento de la lista.
Agregar un elemento al final de la lista.
Combinar dos listas, para formar una sola.
Localizar un elemento de la lista para conocer su posición dentro de ella.
Borrar un elemento de la lista.

3- ¿Cuáles son los distintos tipos de listas que conoces y sus características? ¿Cómo se declaran en Pascal?

Lista enlazada: tiene un enlace por nodo. Este enlace apunta al siguiente nodo en la lista, o al valor NULL o a la lista vacía, si es el último nodo.

```
{Type
        Lista=^nodo;
        Nodo=record
                Dato:tipo_elem;
                Sig:lista; (estructura recursiva)
                End;
Var
        Lis:lista;(memoria estática reservada)}
```

Listas circulares: es una lista enlazada en la cual el último elemento apunta al primero o principio de la lista. Las ventajas que tiene es que cada nodo de la lista es accesible desde cualquier otro nodo de ella. Y las operaciones de concatenación y división son más eficaces. La desventaja es que se pueden producir bucles o lazos infinitos.

Listas doblemente enlazadas: en estas se puede recorrer en ambos sentidos. En este tipo de listas, además del campo información se cuenta con dos variables del tipo puntero que apuntan al nodo anterior y siguiente. Ocupan más memoria por nodo.

```
{Type
        Lista=^nodo;
        Nodo=record
                Dato:tipo_elem;
                Ant,sig:lista;
                End;}
```

ARBOLES

1- ¿Cómo defines el tipo de dato árbol?

Es una estructura de datos que satisface tres propiedades: 1) Cada elemento del árbol (nodo) se puede relacionar con cero o más elementos, los cual los llama “hijos”. 2) Si el árbol no está vacío, hay un único elemento el cual se llama raíz y que no tiene padre (predecesor), es decir, no es hijo de ningún otro. 3) Todo otro elemento del árbol posee un único padre y es un descendiente de la raíz.

2- ¿A que llamamos árbol binario de búsqueda?

Es un árbol de orden 2 en los que se cumple que para cada nodo, el valor de la clave de la raíz del subárbol izquierdo es menor que el valor de la clave del nodo y que el valor de la clave de la raíz del subárbol derecho es mayor que el valor de la clave del nodo.

3- ¿Cuál es la representación de este tipo de datos?

```
{Type
        tipoElem: …;
```

```
        arbolBinario=^nodo;
        nodo=record
                clave:tipoElem;
                izq,der:arbolBinario;
                end;
var
        árbol:arbolBinario;}
```

4- ¿Por qué crees que es útil contar con este tipo de datos?

Porque los datos pueden ser accedidos y procesados de manera mucho más rápida que en otras estructuras.

5- ¿Cuáles son sus operaciones más frecuentes?

- Buscar un elemento.
- Insertar un elemento.
- Borrar un elemento.
- Movimientos a través del árbol (izquierda, derecha, raíz).
- Información (comprobar si esta vacío, calcular el número de nodos, comprobar si el nodo es hoja, calcular la altura del nodo, calcular la altura de un árbol).

6- ¿Cómo harías la operación de inserción en un árbol binario de busqueda?

Cuando se llega a un árbol vacio se crea el nodo en el puntero que se pasa como parámetro por referencia, de esta manera los nuevos enlaces mantienen la coherencia. Si el elemento a insertar ya existe entonces no se hace nada.

```
{procedure insertar (var a:arbolBinario; elem:integer);
        Begin
                If a = NIL then begin
                        New(a);
                        a^.clave := elem;
                        a^.izq := NIL;
                        a^.der := NIL;
                end;
                else
                        if a^.clave < elem then
                                insertar(a^.der, elem);
                else
                        if a^.clave > elem then
                                insertar(a^.izq, elem);
        end;}
```

7- ¿Cómo harías la operación de borrado en un árbol binario de busqueda?

Pueden darse tres casos:

1) El nodo no tiene descendentes. Simplemente se borra.
2) El nodo tiene al menos un descendiente por una sola rama. Se borra dicho nodo, y su primer descendiente se asigna como hijo del padre del nodo borrado.
3) El nodo tiene al menos un descendiente por cada rama. Al borrar dicho nodo es necesario mantener la coherencia de los enlaces, además de seguir manteniendo la estructura como un árbol binario de búsqueda. La solución consiste en sustituir la información del nodo que se borra por el de una de las hojas, y borrar a continuación dicha hoja (la mayor de las claves menores al nodo que se borra o la menor de las claves mayores al nodo que se borra).

```
{procedure borrar (var a:arbolbinario; elem:integer; var ok:boolean);
Var
    Aux:Arbolbinario;
Begin
```

```
If (a = NIL) then
        Ok:=false;
Else begin
        If (elem < a^.dato) then
                Borrar (x,a^.izq,ok);
        Else
                If (elem > a^.dato) then
                        Borrar (x, a^.der, ok);
        Else begin {valor encontrado analizar los 3 casos}
                If (a^.izq = NIL) then begin
                        Aux:=a;
                        A:=a^.der;
                        Dispose(aux);
                End;
                Else
                        If (a^.der = NIL) then begin
                                Aux:=a;
                                A:=a^.izq;
                                Dispose(aux);
                        End;
                        Else begin {2 hijos. Reemplazo con el mas pequeño de la derecha}
                                Aux:=minimo (a^.der);
                                A^.dato:= aux^.dato;
                                Borrar(a^.dato,a^.der,ok);
                                End;
                End;
        End;
    End;}
```

8- ¿En las operaciones sobre arboles es lo mismo hacer una implementación recursiva o iterativa?
Desde el punto de vista algorítmico las operaciones sobre árboles son más claras y eficientes con una implementación recursiva. Por lo tanto no es lo mismo.

9- ¿Cómo realizarias la búsqueda de un minimo en un árbol binario de búsquedas de forma iterativa y recursiva?

Forma recursiva:

```
{function minimo (a:arbolBinario):árbol;
   Begin
      If (a = NIL) then
         Minimo := nil;
      Else
         If (a^.izq = NIL) then
            Minimo:= a;
         Else
            Minimo:=minimo (a^.izq);
   End;
```

REUSABILIDAD

1- ¿Cómo defines la reusabilidad?
La reusabilidad hace referencia a poder volver usar parte de dicho software o algoritmo en otro proyecto o algoritmo.

CORRECCION

1- **¿Cómo defines el concepto de corrección?**
Un programa es correcto cuando cumple con la función especificada (requerimientos propuestos). Para determinar cuáles son esos requerimientos se debe tener una especificación completa, precisa y no ambigua del problema a resolver antes de escribir el programa. Para medir si un programa es correcto se debe probar con datos reales que permitan verificar su función.

2- **Técnicas para medir corrección.**
Testeo de un programa (Testing): es el proceso de proveer evidencias convincentes respecto de si el programa hace el trabajo esperado. Se basa en una batería de pruebas, diseñada adecuadamente, la cual debe ser planteada y no implementada aleatoriamente. Lo que cuenta es la calidad de prueba y no la cantidad. Si la prueba sale mal, el programa de ser corregido y probado con la misma batería de test.
Debugging: el proceso de descubrir un error en el código y repararlo. Los errores pueden aparecer en el programa de distintas formas:
- Problemas en el diseño del programa, será necesario corregirlo o rehacerlo.
- La utilización de un proceso erróneo.
- Codificación incorrecta.
- Errores sintácticos, semánticos o lógicos, es decir errores de diseño.
- Errores de hardware o errores del sistema (poco probables)
- El programa puede ser aplicado a situaciones no previstas produciendo resultados incorrectos.

A veces hace falta poner checkpoints o breakpoints, o incluso hacer un seguimiento paso a paso (trace), inspeccionando los valores de las variables hasta hallar un error.
Walkthroughs: la idea es seguir o leer el código del programa intentado comprender, o haciendo comprender a otro como funciona. Un programa bien estructurado es fácil de leer.
Verificación: es el proceso de analizar las postcondiciones en función de las precondiciones establecidas. Cuando probamos nuestro programa, observamos y analizamos su comportamiento. Sobre la base de este análisis podemos hacer ciertas afirmaciones del tipo. Si nuestro programa es utilizado con ciertos datos, produce ciertos resultados.

3- **¿Cuándo consideras que un programa es correcto?**
Un programa es correcto cuando considero que sus requerimientos son completos, precisos y no erróneos.

4- **Si un programa es correcto, ¿también es eficiente?**
No, un programa puede ser correcto y no ser eficiente, es decir, puede estar entregando un resultado correcto pero haciendo un mal uso de recursos.

5- **¿Un programa libre de errores es correcto?**
No falso, puede no tener errores un programa pero a la vez puede no ser preciso y completo lo cual no significa que sea correcto.
Siempre y cuando cumpla con la función especificada, si está libre de errores, es correcto.

6- **¿La corrección de un programa depende de la elección del lenguaje de programación elegido?**
No falso, podemos elegir cualquier lenguaje de programación, porque la corrección depende que se cumpla con sus requerimientos de ser completo, preciso y no erróneo.

7- **¿La cantidad de líneas de código de un programa determinan su eficiencia en cuanto a tiempo de ejecución?**
No falso, la líneas de código de un programa no determinan la eficiencia, lo que determina la eficiencia son las cantidad de operaciones.

8- **¿Si has utilizado todas las técnicas y determinas que el programa es correcto, puedes asegurar que ese programa será siempre correcto?**
Si, puedo asegurar que el programa va ser correcto porque en cada técnica puedo comprobarlo.

9- **¿Si eliges una solución modularizada, esto te asegura que el programa es correcto?**
No falso, porque puede no cumplir con los requerimientos de ser completo, preciso y no erróneo.

10- **¿Un programa correcto asegura legibilidad?**

No falso, porque puede ser muy legible un programa pero puede no cumplir con los requerimientos de ser completo, preciso y no erróneo.

11- **¿Un programa bien documentado asegura corrección?**
No falso, un programa puede estar bien documentado pero si no cumple con todos los requerimientos de ser completo, preciso y no erróneo no es correcto.

12- **¿La correcta utilización de variables locales y globales ayuda a la corrección de una solución?**
No, porque la utilización de variables globales lleva a posibles errores, y en una corrección de un algoritmo es importante que no sea erróneo.

13- **¿Crees que existe una única solución correcta a un problema planteado?**
No falso, puede haber varias soluciones correctas para un mismo problema.

14- **¿Las estructuras de datos elegidas determinan que una solución sea correcta o no?**
No falso, la corrección no depende de la elección de una estructura de datos podes elegir cualquier estructura, mientras se respete que los requerimientos sean completos, precisos y no errónea va ser correcto la solución.

EFICIENCIA

1- **¿Cómo defines el concepto de eficiencia?**
Un algoritmo es eficiente si realiza una administración correcta de los recursos del sistema en el cual se ejecuta. La eficiencia mide el tiempo que tarda un algoritmo en ejecutarse y la memoria requerida, en otros aspectos.

2- **¿Cómo puedes medir la eficiencia de una solución dada?**
La eficiencia de una solución dada se logra midiendo el tiempo de ejecución y el uso de memoria utilizada.

3- **Describa los métodos que permiten medir la eficiencia de un algoritmo respecto del tiempo de ejecución.**
   - Realizar un análisis teórico: se busca tener una medida de trabajo realizado por el algoritmo a fin de obtener una estimación teórica del tiempo de ejecución. Esto permite comparar algoritmos y seleccionar la mejor implementación.
   - Realizar un análisis empírico: es necesario ejecutar el programa y medir los recursos utilizados. Tiene la ventaja de ser muy fácil de implementar, pero no tiene en cuenta algunos factores como la velocidad de la máquina y los datos con lo que se ejecuta el algoritmo.

4- **¿Existe una relación directa entre eficiencia y corrección?**
Existe una relación pero no es directa, podríamos decir que es indirecta, ya que un programa puede ser correcto pero no eficiente.

5- **¿Una solución eficiente es siempre correcta?**
Sí, porque para verificar si es eficiente o no, primero tenemos que comprobar que sea correcto.

6- **¿Desde el punto de vista de eficiencia, una solución recursiva es más o menos eficiente que una iterativa?**
Es menos eficiente por causa del overhead (sobrecarga) de tiempo y memoria asociado con la llamada a subprogramas.

TAD

1- **¿Cómo defines Tipo Abstracto de Datos?**
Un tipo abstracto de datos (TAD) es un tipo definido por el usuario que:
- Tiene un conjunto de valores y un conjunto de operaciones.
- Cumple con los principios de abstracción, ocultación de la información y se puede manejar sin conocer la representación interna.

Es decir, los TADs ponen a disposición del programador un conjunto de objetos junto con sus operaciones básicas que son independientes de la implementación elegida.

2- **¿Cuáles son sus características y principales ventajas?**
Características:
   - Es un tipo de dato para representar objetos del mundo real que no existen en el lenguaje.

- Tiene un parte pública (INTERFACE) en donde se declara el nombre del tipo y las posibles operaciones.
- Tiene un parte privada (IMPLEMENTACION) donde se implementa con alguna estructura el tipo declarado y todas las operaciones de la interface.
- El usuario solo conoce la parte pública, es decir, no conoce cómo es la estructura utilizada ni como se implementaron las operaciones.

Ventajas: La independencia facilita la re-usabilidad del código.

El programa o modulo que referencia al TAD, lo utiliza como "caja negra", esto permite que las modificaciones internas del TAD no afecten a quienes lo utilizan, siempre que su interfaz no se modifique.

Se diferencian dos etapas:

- En el momento de diseñar y desarrollar el TAD no interesa conocer la aplicación que lo utilizara. Esto permite concentrarse únicamente en la implementación del nuevo tipo.
- En el momento de utilizar el TAD no interesa saber cómo funcionara internamente, solo bastara con conocer las operaciones que permiten manejarlo.

PILAS (TAD PILA)

1- Definición. Características. Operaciones.

El tipo de dato pila (stack) es una colección ordenada de elementos, con tres características básicas:

1) Es una estructura homogénea.
2) Los elementos pueden recuperarse en orden inverso al que fueron almacenados (LIFO: last in first out)
3) Es una estructura dinámica.
4) Lineal.

Sus operaciones son:

- Crear pila.
- Agregar elemento.
- Sacar elemento.
- Saber cuál elemento está en el tope.
- Cantidad de elementos contenidos.
- Saber si está vacía.

COLAS (TAD COLA)

1- Definición. Características. Operaciones.

El tipo de dato cola es una colección ordenada de elementos, con tres características básicas:

1) Es una estructura homogénea.
2) Los elementos pueden recuperarse en el orden en que fueron almacenados (FIFO: first in first out).
3) Es una estructura dinámica
4) Lineal.

Sus operaciones son:

- Crear cola.
- Agregar elemento.
- Sacar elemento.
- Saber cuál elemento está en el tope.
- Saber cuál elemento esta al final.
- Cantidad de elementos contenidos.
- Saber si está vacía.

PROGRAMACION ORIENTADA A OBJETOS

1- **¿Cómo define una clase?**
   Una clase se puede definir como un modelo que define los atributos y métodos comunes a todos los objetos que comparten las mismas características. Es una "plantilla genérica" para un conjunto de objetos de similares características. Compuesta por un conjunto de atributos, la estructura de los atributos y un conjunto de métodos para el comportamiento.
2- **¿Cómo define un objeto?**
   Un objeto es una unidad que contiene atributos específicos (datos) y responde a los métodos (comportamiento) de su clase, para operar sobre esos atributos. Es una instancia de una clase y se comunican entre sí mediante mensajes.
3- **¿Qué son los métodos?**
   Un método es una subrutina cuyo código es definido en una clase y puede pertenecer tanto a una clase, como es el caso de los métodos de clase o estáticos, como a un objeto, como es el caso de los métodos de instancia. Consiste generalmente de una serie de sentencias para llevar a cabo una acción.
4- **¿Cómo explica el concepto de herencia?**
   Es un mecanismo que le permite a un objeto heredar propiedades de otra clase de objetos. La herencia permite a un objeto contener sus propios procedimientos o funciones y heredar los mismos de otros objetos.
5- **¿Cómo explica el concepto de polimorfismo?**
   Se refiere a la propiedad por la que es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos. El único requisito que deben cumplir los objetos que se utilizan de manera polimórfica es saber responder al mensaje que se les envía.

PROGRAMACION CONCURRENTE

1- **¿Qué entiende por programación concurrente?**
   Concurrencia es la característica de los sistemas que indica que múltiples procesos/tareas pueden ser ejecutados al mismo tiempo y pueden cooperar y coordinarse para cumplir la función del sistema.
2- **¿Cómo se comunican los procesos en un entorno concurrente?**
   Pasaje de mensajes → enviar y recibir.
   - Es necesario establecer un canal (lógico o físico) para transmitir información entre procesos.
   - También el lenguaje debe proveer un protocolo adecuado.
   - Para que la comunicación sea efectiva los procesos deben "saber" cuándo tienen mensajes para leer y cuando deben trasmitir mensajes.

   Memoria compartida → depositar y sacar.
   - Los procesos intercambian información sobre la memoria compartida o actúan coordinadamente sobre datos residentes en ella.
   - Lógicamente no pueden operar simultáneamente sobre la memoria compartida, lo que obliga a bloquear y liberar el acceso a la memoria.
   - La solución más elemental es una variable de control que habilite o no el acceso de un proceso a la memoria compartida