Conceptos de algoritmos, datos y programas (CADP)

Guia teoria

Clase 1: Definiciones fundamentales

Informática

Es la ciencia que estudia el análisis y resolución de problemas utilizando computadoras. Cuyo objetivo es resolver problemas del mundo real utilizando una computadora (utilizando un software).

Algoritmo

- Los algoritmos son secuencias de instrucciones que indican cómo realizar operaciones específicas.
- Un conjunto de instrucciones forma un algoritmo, que guía a la computadora en sus acciones.
- Los algoritmos deben ser específicos, precisos y capaces de alcanzar un resultado deseado en un tiempo finito.
- La cantidad de instrucciones en un algoritmo también debe ser finita.

Datos

- Los datos son valores de información necesarios para que un programa funcione.
- Pueden ser constantes o variables y representan objetos del mundo real.
- Los datos pueden ser números, lógicos (verdadero o falso) o caracteres.

Operaciones y Programas

- Los programas consisten en una secuencia de operaciones que una computadora ejecuta.
- Las operaciones incluyen operaciones aritméticas y lógicas, como sumar, restar y verificar la veracidad.
- Los programas deben ser legibles, organizados y documentados para facilitar su comprensión y mantenimiento.

Diseño de Variables y Constantes

- Las variables son zonas de memoria que pueden cambiar su valor durante la ejecución del programa.
- Las constantes también son zonas de memoria, pero su valor no cambia durante la ejecución del programa.
- Las variables y constantes deben ser declaradas y asociadas a un tipo de dato.

Tipos de Datos y Verificación

- Los tipos de datos incluyen simples (un único valor) y compuestos (varios valores bajo un nombre).
- Tipos de datos comunes incluyen numéricos (enteros y reales), lógicos y caracteres.
- Los lenguajes de programación pueden ser fuertemente tipados (verifican tipos estrictamente) o dinámicamente tipados (permiten cambios de tipo).

Precondición y Postcondición

- La precondición se refiere a la información verdadera antes de iniciar un programa o módulo.
- La postcondición se refiere a la información que debería ser verdadera al finalizar un programa o módulo si se siguen los pasos especificados.

Conceptos Básicos en Estructuras de Control

Las estructuras de control en la programación son esenciales para definir cómo se ejecutan las instrucciones en un algoritmo. Estas estructuras permiten organizar la lógica del programa, tomar decisiones y repetir acciones de manera controlada. Existen tres estructuras fundamentales: secuencia, decisión e iteración.

Secuencia: Flujo Lógico

- La estructura de control más simple es la secuencia, donde las operaciones se ejecutan en el mismo orden en que aparecen.
- Las instrucciones se ejecutan una tras otra, siguiendo el flujo lógico del programa.

Decisión: Elecciones basadas en Datos

- Las decisiones son cruciales en algoritmos que resuelven problemas reales.
- La estructura de decisión permite elegir entre dos alternativas basadas en ciertas condiciones.
- Se utiliza un condicional que determina la dirección del flujo según la evaluación de una condición.

Iteración: Repetición Controlada

- La iteración es útil cuando se necesita repetir un bloque de instrucciones un número específico de veces o mientras se cumpla una condición.
- Existen dos tipos principales de iteración: precondicional y postcondicional.

Iteración Precondicional: Mientras se Cumple la Condición

- La condición se evalúa antes de entrar al bloque de acciones.
- Si la condición es verdadera, se ejecutan las acciones y luego se vuelve a evaluar la condición.
- El bloque de acciones se repite O, 1 o más veces mientras la condición sea verdadera.

Iteración Postcondicional: Ejecutar y Luego Evaluar

- Las acciones se ejecutan primero y luego se evalúa la condición.
- Si la condición es falsa, el bloque de acciones se repite O, 1 o más veces.
- Esta estructura asegura que las acciones se ejecuten al menos una vez antes de evaluar la condición.

Precondicionales vs. Postcondicionales

- Las estructuras precondicionales evalúan la condición antes de ejecutar el bloque de acciones.
- Las estructuras postcondicionales ejecutan el bloque de acciones antes de evaluar la condición.
- Ambas estructuras permiten repetir acciones bajo ciertas condiciones.

Repetición: Ejecución Fija y Conocida

- La repetición es una extensión de la secuencia, donde un bloque de acciones se repite un número fijo de veces.
- El número de repeticiones es conocido de antemano y se ejecutan de forma sucesiva.

Clase 2: Estructuras de Control en Programación

Las estructuras de control son instrucciones esenciales en la programación, incluyendo secuencia, decisión e iteración.

Estructuras de Control de Repetición (Bucles)

Estas estructuras permiten ejecutar bloques de código repetidamente hasta que se cumpla una condición.

Estructuras de Control de Selección (Switch)

Permiten tomar decisiones basadas en el valor de una variable ordinal. Requieren que las opciones sean disjuntas.

Maximizar y Minimizar Valores

Para encontrar el máximo y el mínimo en un conjunto de datos:

- Inicializar una variable como el valor máximo (inicialmente bajo) y otra como el mínimo (inicialmente alto).
- Actualizar las variables según los datos leídos.
- Esto permite encontrar los valores máximos y mínimos en el conjunto de datos.

Tipos de Datos Definidos por el Programador

Estos tipos permiten crear nuevas representaciones de datos a partir de tipos simples. Son fundamentales para manejar datos no estándar en programación, brindando mayor seguridad y flexibilidad.

Ventajas de los Tipos de Datos Definidos por el Usuario (TDDU)

- **Flexibilidad:** Modificar la representación de datos es sencillo, al cambiar una sola declaración en lugar de varias de variables.
- **Documentación:** Utilizar nombres autoexplicativos para tipos mejora la legibilidad del código.
- **Seguridad:** Evita errores al limitar las operaciones permitidas para cada tipo de dato.

Subrango

Un subrango es una sucesión de valores de un tipo ordinal base (predefinido o definido por el usuario). Es simple, ordinal y común en la mayoría de los lenguajes.

Operaciones Permitidas en Subrango

Se permiten asignación, comparación y todas las operaciones del tipo base. Las operaciones no permitidas dependen del tipo base.

Clase 3: Modularización

La modularización implica dividir un problema en partes independientes que encapsulan operaciones y datos. No es solo subdividir código, sino separar funciones lógicas con datos específicos y de comunicación.

Características de la Modularización:

- Subproblemas independientes con datos y objetivos propios.
- Nivel de detalle uniforme en subproblemas.
- Solución independiente de subproblemas.
- Combinación de soluciones para el problema principal.

Módulo: Encapsulando Funcionalidades

Un módulo es una unidad funcional específica que coopera para lograr un objetivo común. Encapsula acciones o funciones y representa los objetivos del problema.

Metodología Top-Down

Desarrollo en capas, facilitando trabajo en paralelo, reusabilidad y adaptación a cambios.

Ventajas de la Modularización:

- Aumenta la productividad al trabajar en paralelo.
- Favorece la reusabilidad del software.
- Facilita la incorporación de nuevas prestaciones.
- Mejora la claridad y comprensión del código.

Alternativas a los Módulos:

Se usan subrutinas, módulos, procedimientos, funciones y clases para modularizar el código.

Procedimiento y Función:

Los procedimientos ejecutan tareas y pueden retornar valores. Las funciones retornan un único valor de tipo simple. Características incluyen el tipo de valor a retornar y la asignación del nombre de función a la variable que la recibe.

Alcance de las Variables en Programación

Las variables en programación tienen diferentes alcances según su nivel de declaración. Estos alcances se dividen en:

Variables Globales:

Pueden usarse en todo el programa, incluyendo los módulos. Son accesibles desde cualquier parte del código.

Variables Locales al Programa:

Solo pueden usarse en el cuerpo del programa donde están declaradas. No son visibles fuera del programa.

Variables Locales al Proceso:

Son accesibles solo en el proceso (o función) en el que se declaran. No pueden ser usadas fuera de ese proceso.

Resolución de Variables:

Si una variable se utiliza en un proceso, se busca en orden:

- Como variable local al proceso.
- Como parámetro del proceso.
- Como variable global al programa.

Si una variable se usa en un programa, se busca en orden:

- Como variable local al programa.
- Como variable global al programa.

Consideraciones:

La elección del alcance adecuado para las variables es crucial para evitar confusiones y problemas en el código. Variables globales pueden ser convenientes, pero deben usarse con cuidado debido a su alcance extendido.

Clase 4: Comunicación entre modulos

La comunicación efectiva entre módulos es crucial para un diseño de software sólido. Las opciones incluyen:

Variables Globales:

- **Problemas de Identificación:** Exceso de identificadores y conflictos de nombres pueden surgir.
- **Pérdida de Integridad:** Cambios inadvertidos en variables pueden afectar otros módulos.

Parámetros:

- **Solución a Problemas:** Uso de parámetros y ocultamiento de datos para evitar conflictos y pérdida de integridad.

Parámetros por Valor:

- **Pasaje de Valor:** El módulo recibe un valor de otro módulo y opera con él localmente sin afectar su origen.

Parámetros por Referencia:

- **Comunicación Directa:** El módulo opera con una variable conocida en otros módulos, reflejando cambios en todos ellos.

Invocación de Módulos:

- **Coincidencia de Argumentos:** El número y tipo de argumentos deben coincidir con los parámetros del módulo.
- **Parámetros por Valor:** Se trata como una copia local, modificaciones solo afectan al módulo actual.

Una combinación de ocultamiento de datos y parámetros permite una comunicación eficiente y segura entre módulos en un programa.

Clase 5: Tipos y Clasificación de Datos Estructurados en Programación

Las estructuras de datos son fundamentales para organizar y gestionar información en programas. Se pueden clasificar en:

Homogéneas y Heterogéneas:

- **Homogéneas:** Elementos del mismo tipo.
- **Heterogéneas:** Elementos de tipos diferentes.

Tamaño Estático y Dinámico:

- **Estáticas:** Tamaño fijo durante la ejecución.
- **Dinámicas:** Tamaño puede cambiar en ejecución.

Acceso Secuencial y Directo:

- **Secuencial:** Acceso siguiendo un orden predefinido.
- **Directo:** Acceso directo a elementos, sin recorrer otros.

Linealidad:

- **Lineal:** Elementos relacionados adyacentemente.
- **No Lineal:** Elementos no siguen relación adyacente.

Registros:

- Tipo de datos estructurado.
- **Heterogéneos:** Pueden contener diferentes tipos de datos.
- **Estáticos:** Tamaño fijo.
- **Campos:** Representan datos dentro del registro.

Las estructuras de datos permiten la organización eficiente y el manejo de información en los programas, brindando flexibilidad y claridad en la gestión de datos.

Clase 6: Tipos de Datos Arreglos

Los arreglos (ARRAYs) son estructuras de datos que permiten acceder a elementos mediante un índice. En el contexto de PASCAL, se pueden dividir en diferentes tipos:

Arreglo:

- Estructura compuesta con acceso mediante un índice.
- Permite almacenar y acceder a elementos por posición.

Vector:

- Arreglo con elementos consecutivos en memoria.
- **Homogéneo:** Todos los elementos del mismo tipo.
- **Estático:** Tamaño fijo en la compilación.
- **Indexado:** Acceso mediante índices ordinales.

Recorridos de Vectores:

- **Recorrido Total:** Analiza todos los elementos del vector, explorando completamente la estructura.
- **Recorrido Parcial:** Analiza elementos hasta encontrar el que cumple cierta condición. Puede requerir recorrer todo el vector..

Clase 7: Operaciones con vectores

Las operaciones con vectores, esenciales en PASCAL, involucran aspectos relacionados con sus dimensiones y manipulación de datos:

Dimensiones de un Vector:

- **Dimensión Física:** Determina la cantidad máxima de memoria ocupada. Establecida en la declaración, no cambia durante la ejecución.
- **Dimensión Lógica:** Indica las posiciones ocupadas con contenido real. Nunca excede la dimensión física.

Agregar:

- Añadir un nuevo elemento al final del vector.
- Puede ser imposible si el vector está lleno.

```
procedure agregarAlVector(var v:vector;var dimL:integer;t);
begin
    if ((dimL+1)<dimF) then begin
        dimL:=dimL+1;
        v[dimL]:=t;
    end;
end;
```

Insertar:

- Agregar un elemento en una posición específica.
- Puede ser imposible si el vector está lleno o la posición no es válida.

```
procedure insertarAlVector(var v:vector;var dimL:integer;t;pos);
var
    i:integer;
begin
    if ((dimL+1)<dimF) and (pos>=1) and (pos<=dimL) then begin
        for i:= dimL downto pos do {corre los elementos para hacer espacio}
            v[i+1]:=a[i];
        v[pos]:=t;
        dimL:=dimL+1;
    end;
end;
```

Eliminar:

- Borrar un elemento en una posición determinada o con un valor específico.
- Puede ser imposible si la posición es inválida o el elemento no existe.

```
procedure eliminarDelVector(var v:vector;var dimL:integer;pos);
var
    i:integer;
begin
    if ((pos>=1) and (pos<=dimL)) then begin
        for i:= pos to (dimL-1)) do {corre los elementos para eliminar el espacio}
            v[i]:=a[i+1];
        dimL:=dimL-1;
    end;
end;
```

Buscar:

- Recorrer el vector en busca de un valor específico.
- Diferencias entre vectores desordenados y ordenados.

Búsqueda en Vectores Desordenados:

- Requiere recorrer todo el vector en el peor caso.

- Puede detenerse al encontrar el valor o al agotar el vector.

Búsqueda en Vectores Ordenados:

- Aprovecha el orden para reducir comparaciones.
- Dos enfoques: búsqueda mejorada y búsqueda dicotómica.

Búsqueda Mejorada:

- Aplicable a elementos con orden.
- Promedio de comparaciones: (dimL+1)/2.

Búsqueda Dicotómica:

- Aplicable a elementos ordenados.
- Promedio de comparaciones: (1 + 10^(log2(dimL+1)/2)).

Clase 8: Punteros, memoria y listas

Los punteros son elementos cruciales en PASCAL para la manipulación dinámica de memoria y gestión de datos. Aquí presentamos los aspectos más relevantes:

Alocación Estática y Dinámica:

- **Variables Estáticas:** No cambian su tamaño en tiempo de ejecución. La memoria se reserva en la declaración y perdura durante el programa.
- **Variables Dinámicas:** Permiten cambios en tiempo de ejecución.

Concepto de Puntero:

- Un puntero es una variable que almacena direcciones de memoria.
- Contiene la dirección donde reside un valor (de cualquier tipo: char, boolean, integer, real, string, registro, arreglo u otro puntero).
- Es un tipo de dato simple.

Características:

- Almacena dirección donde reside el valor real.
- Apunta solo a direcciones en memoria dinámica (heap).
- Puede apuntar a un solo tipo de dato.
- Se representa con ^ y ocupa 4 bytes (stack) en Pascal.

Operaciones con Punteros:

- **Creación:** Reserva memoria dinámica para asignarle contenido. (new(puntero))
- **Eliminación:** Libera la memoria dinámica de un puntero. (dispose(puntero))
- **Liberación:** Rompe el enlace con la memoria dinámica (queda inaccesible).
- **Asignación:** Asigna dirección de un puntero a otro del mismo tipo.
- **Contenido:** Accede al valor contenido en la dirección apuntada (^).

Notas Importantes:

- if (p = nil) compara si el puntero p no tiene dirección.
- if (p = q) compara si p y q apuntan a la misma dirección.
- if (p^ = q^) compara si p y q tienen el mismo contenido.
- No se puede hacer read(p) o write(p) si p es un puntero.
- No se asigna una dirección manualmente (p:= ABCD).
- No se compara por mayor o menor dirección de punteros (p>q).

Listas y memoria

Las listas son estructuras fundamentales para gestionar datos de manera dinámica en PASCAL. Aquí se presentan sus aspectos más destacados:

Memoria de un Programa:

- La memoria necesaria para la ejecución del programa se divide en memoria estática y dinámica.
- **Memoria Estática:** Incluye variables locales y globales del programa.
- **Memoria Dinámica:** Engloba la reserva y liberación de memoria en tiempo de ejecución.

Tabla de Ocupación:

- **char** (1 byte)
- **boolean** (1 byte)
- **integer** (4 bytes)
- **real** (8 bytes)
- **string** (tamaño + 1 byte)
- **subrango** (depende del tipo)
- **registro** (suma de sus campos)
- **arreglos** (dimFísica * tipo de elemento)
- **puntero** (4 bytes).

Listas:

- Una colección de nodos, donde cada nodo almacena un elemento y la dirección del siguiente nodo.
- Los nodos no ocupan posiciones contiguas en memoria, pero mantienen un orden lógico.
- Cada nodo se representa con un puntero.

Características de las Listas:

- **Homogénea:** Todos los elementos son del mismo tipo.
- **Dinámica:** La cantidad de nodos puede variar durante la ejecución.
- **Lineal:** Cada nodo tiene un único antecesor y sucesor.
- **Acceso:** El acceso a cada elemento es secuencial.

Operaciones con Listas:

- **Crear una Lista:** Marca que la lista no tiene una dirección inicial.
- **Recorrer una Lista:** Se posiciona al comienzo de la lista y pasa por cada elemento secuencialmente hasta llegar al final.

Gestión de Memoria Dinámica en Listas:

- Para agregar un nodo, se reserva memoria dinámica (new).
- Para eliminar un nodo, se libera memoria dinámica (dispose).

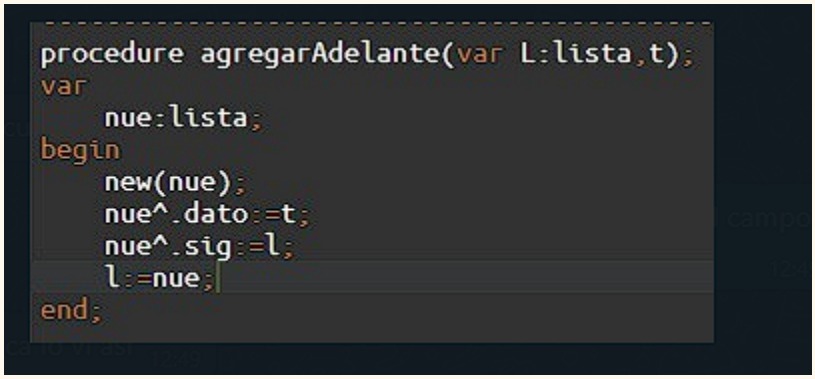
Las listas permiten una gestión eficiente de datos dinámicos y son fundamentales en la programación para administrar información de manera flexible.

Clase 9: Operaciones con listas

Las listas en PASCAL se pueden manipular mediante una serie de operaciones esenciales que permiten gestionar los datos de manera eficiente:

Agregar al Frente en una Lista:

- Implica crear un nuevo nodo y agregarlo como el primer elemento de la lista.
- El nuevo nodo se convierte en el inicio de la lista.



```
procedure agregarAdelante(var L:lista,t);
var
    nue:lista;
begin
    new(nue);
    nue^.dato:=t;
    nue^.sig:=l;
    l:=nue;
end;
```

Agregar al Final en una Lista:

- Involucra generar un nuevo nodo y colocarlo como el último elemento de la lista.
- El nuevo nodo se convierte en el último de la lista.

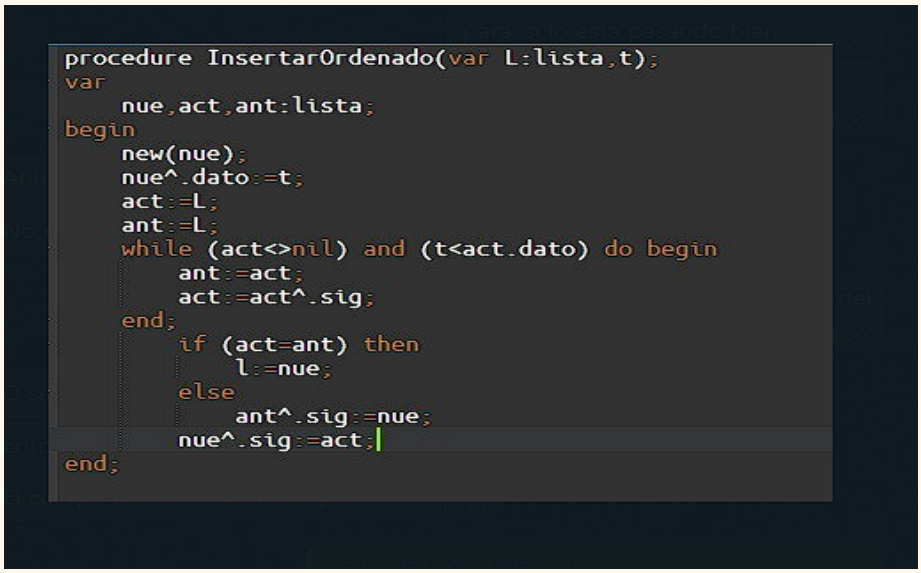
```
procedure agregarAtras(var L:lista,t);
var
    nue,ult:lista;
begin
    new(nue);
    nue^.dato:=t;
    nue^.sig:=nil;
    if (L=nil) then
        L:=nue;
    else
        ult^.sig:=nue;
    ult:=nue;
end;
```

Buscar un Elemento en una Lista:

- Se recorre la lista desde el principio, nodo por nodo, hasta encontrar el elemento deseado o llegar al final de la lista.
- Es una operación fundamental para encontrar información específica en la lista.

Insertar un Elemento en una Lista:

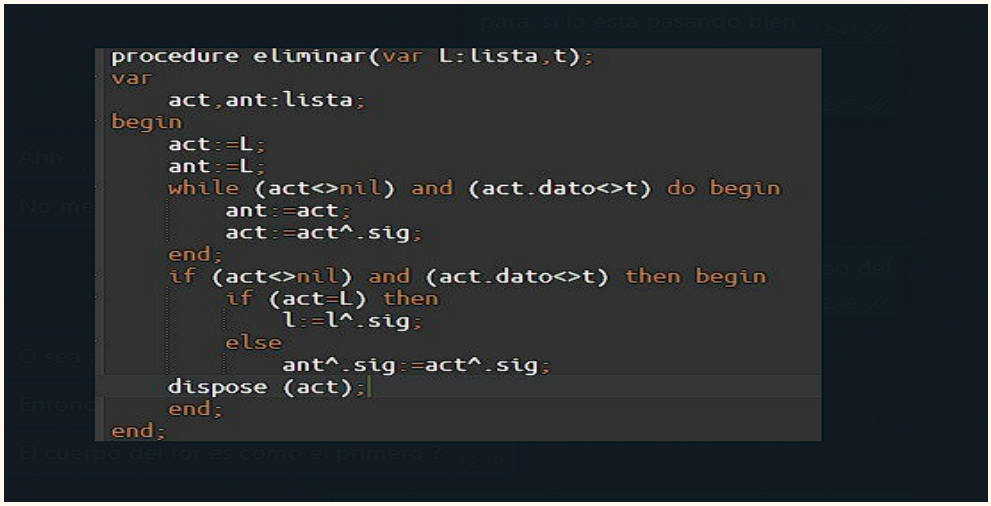
- Requiere que la lista tenga un orden definido.
- Implica agregar el elemento de manera que la lista se mantenga ordenada.
- Se inserta el elemento en la posición adecuada según su orden.



```
procedure InsertarOrdenado(var L:lista,t);
var
    nue,act,ant:lista;
begin
    new(nue);
    nue^.dato:=t;
    act:=L;
    ant:=L;
    while (act<>nil) and (t<act.dato) do begin
        ant:=act;
        act:=act^.sig;
    end;
        if (act=ant) then
            l:=nue;
        else
            ant^.sig:=nue;
        nue^.sig:=act;
end;
```

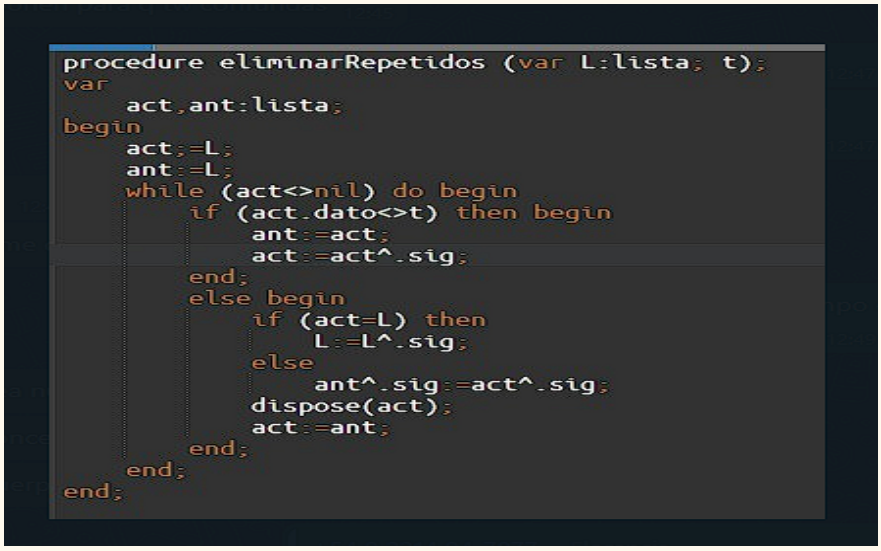
Eliminar un Elemento de una Lista:

- Involucra recorrer la lista desde el inicio, nodo por nodo, hasta encontrar el elemento que se desea eliminar.
- En ese momento, se elimina el nodo correspondiente (utilizando la función "dispose").
- Es importante destacar que el elemento a eliminar puede no estar presente en la lista.



```
procedure eliminar(var L:lista,t);
var
    act,ant:lista;
begin
    act:=L;
    ant:=L;
    while (act<>nil) and (act.dato<>t) do begin
        ant:=act;
        act:=act^.sig;
    end;
    if (act<>nil) and (act.dato<>t) then begin
        if (act=L) then
            l:=l^.sig;
        else
            ant^.sig:=act^.sig;
    dispose (act);
    end;
end;
```

Eliminar repetidos



```
procedure eliminarRepetidos (var L:lista; t);
var
    act,ant:lista;
begin
    act;=L;
    ant:=L;
    while (act<>nil) do begin
        if (act.dato<>t) then begin
            ant:=act;
            act:=act^.sig;
        end;
        else begin
            if (act=L) then
                L:=L^.sig;
            else
                ant^.sig:=act^.sig;
            dispose(act);
            act:=ant;
        end;
    end;
end;
```

Estas operaciones básicas son fundamentales para gestionar y manipular listas en programación, permitiendo la adición, búsqueda, inserción y eliminación de elementos de manera controlada y ordenada.

Clase 10: Corrección y eficiencia

Al desarrollar algoritmos en programación, es crucial considerar dos conceptos fundamentales:

Corrección: Un programa se considera correcto si se desarrolla de acuerdo con sus especificaciones y cumple con su propósito.

Técnicas para Corrección de Programas:

Testing: El testing busca proporcionar pruebas convincentes de que un programa realiza el trabajo esperado.

Plan de Pruebas:

- Define los aspectos del programa a ser probados y establece casos de prueba para cada aspecto.
- Determina los resultados esperados para cada caso de prueba, prestando atención a casos límite.
- Diseña casos de prueba basados en lo que el programa debe hacer, no solo en lo que está escrito.

Proceso de Pruebas:

- Se analizan los casos de prueba y se corrigen los errores detectados.
- Este proceso se repite hasta que no haya más errores.

Debugging: Es el proceso de identificar y corregir las causas de los errores en un programa.

- Involucra agregar sentencias para monitorear el comportamiento del programa.
- Diseñar pruebas adicionales para identificar la naturaleza de los errores.

Tipos de Errores:

- **Errores Sintácticos:** Detectados en la etapa de compilación.
- **Errores Lógicos:** Se manifiestan durante la ejecución y generalmente se deben a problemas en el diseño o en la implementación.

- **Errores de Sistema:** Son raros y pueden originarse en el entorno de ejecución.

Walkthrough: Es un proceso de revisar un programa frente a una audiencia sin preconceptos, lo que permite descubrir errores u omisiones.

- Al leer un programa ante otra persona, es posible identificar errores que el programador no haya notado.

Verificación: Verificar un programa implica asegurarse de que se cumplan las precondiciones y postcondiciones definidas en el mismo.

Eficiencia

La eficiencia en programación se refiere al análisis del tiempo y la memoria que requiere un algoritmo para ejecutarse de manera óptima. La eficiencia está influenciada por diversos factores:

Factores de Influencia:

Datos de Entrada:

- Tamaño y cantidad de los datos de entrada.

Calidad del Código:

- La calidad del código generado por el compilador.

Instrucciones de Máquina:

- La naturaleza y velocidad de ejecución de las instrucciones de máquina.

Tiempo del Algoritmo Base:

- El tiempo base que el algoritmo requiere para su ejecución.

Tiempo de Ejecución: El tiempo de ejecución de un algoritmo puede definirse como una función de entrada:

- Algunos algoritmos tienen un tiempo de ejecución que no depende de las características de los datos, sino de la cantidad o tamaño de los datos de entrada.

- En otros casos, el tiempo de ejecución es específico para ciertas entradas, como el "peor caso", proporcionando una cota superior para cualquier entrada.

Análisis Empírico: Consiste en implementar el programa y medir el tiempo consumido durante su ejecución.

- Fácil de realizar y obtiene valores exactos para una configuración específica.
- Altamente dependiente de la máquina en la que se ejecuta y de los datos utilizados.
- Requiere ejecutar el algoritmo repetidamente para obtener resultados confiables.

Análisis Teórico: Implica encontrar una cota máxima para expresar el tiempo del algoritmo sin ejecutarlo.

Método de Análisis Teórico:

- Se calcula el tiempo de ejecución de cada instrucción elemental del algoritmo.
- Considera solo las instrucciones fundamentales: asignación, operaciones aritmético/lógicas.
- Se asume que cada operación elemental se ejecuta en una unidad de tiempo constante (1UT).

Ejemplo de Cálculo de Tiempo: Supongamos que tenemos un algoritmo con tres instrucciones:

- Asignación (1UT)
- Operación aritmética (1UT)
- Asignación (1UT)

El tiempo total para este algoritmo sería: 1UT + 1UT + 1UT = 3UT.

La eficiencia en programación busca optimizar el tiempo y la memoria utilizados por los algoritmos. El análisis empírico y teórico son herramientas clave para lograr una programación eficiente.