```
Program final; - 1er llamado fecha febrero 2023
type
    feeha = record
     dia: mteger;
     mes: string[50];
     end;
     Cliemte = record
               dmi: imteger;
               f: fecha;
               monto: real;
              end;

    Lista1 = ^modo;
    modo = record
           mfo: Clieute;
           Sig: Lista1;
           end;

  registro = reeord
           Dmi: imreger;
           CantCompras: integer;
           MONTOCompras: real;
           end;
  Lista2 = ^ modo;
  modo = record
           info: registro;
           sig: Lista2;
           end;
Procedure LeerCliemte (var C: Cliente); ll se dispome.
Procedure Insertar por Dmi (var L: Lista1; C: Cliemte); ll se dispome
```

```
Procedure AgregarAtras ( var l2, ult: Lista2; r: registro);
var aux: lista2;
Begin
        new (aux);
        aux^ . info := r;
        aux^ . sig := nil;
        If ( L2 = nil) then L2 := aux
        Else ult^. sig := aux;
        ult := aux;
end;

Procedure recorrer L1 (L1: Lista1; Var L2: Lista2);
var r: registro;
Begin
        while (L1 < nil) do begin
        r: dmi := L1^. info. dmi; r. montoCompras := 0; r. cantCompras := 0;

        while (( L1 <> nil) and ( L1^. info . dmi = r. dmi)) do begin
            r. montoCompras := r. montoCompras + L1^. info . monto;
            r. cantCompras := r. cant Compras + 1;
            L1 := L1^. sig;
        end;
       if (r. cantCompras > 5) then AgregarAtras (L2, ult, r);
       end;
end;

var L1: lista1; L2: lista2; C: Cliente;
Begin
    L1 := nil
    L2 := nil
    leerCliente (c); insertar por dmi (L1, C);
    recorrer L1 (l1,l2);
end;
```

HOJA Nº

FECHA

2.

A: reserva la memoria dinámica e incrementa sobre basura.

B: suma sobre la variable puntero en lugar de memoria dinámica.

3.

Memoria estática	Memoria dinámica
alumno: 26 + 4 bytes	for: 10 × 30 (10 ocupa alumno)
i: 4 bytes	dispose - 30 (un alumno menos)
Vector 4 × 40 bytes	Total 270
Total: 194	

Tiempo de ejecución:

For 3. 11 del for + 2 + 11 2 de asignación

1 asignación fuera del for

Tiempo = 85 UT

Verdadero o falso:

a. No siempre. Si es un vector que no esta lleno, considero la dimensión lógica. (Puede no estarlo)

b. Falso El Div y Mod solo se realiza entre enteros

c. El vector referenciado o apuntado lo que otorgará un puntero, dirección de memoria que es el acceso a la variable. La lista copiara su valor que también es un puntero. Los dos valen lo mismo.

d. Verdadero. Si se mantiene un puntero al final de la lista no hace falta recorrerla secuencialmente

NOTA

e. Invocar si,
se puede comunicar mediante variables globales pero es una mala práctica.

f.
La lista debería ser pasada por referencia (siendo una mala practica) pero podría ya que retorna un tipo de dato simple.
Aunque si modifico el valor al que apunta en memoria y la lista es por valor, como no modifico el puntero quedaría guardado.
No quedarían cosas como modificar los punteros.

g. Verdadero. es más legible, pensable. Buena practica
Solo si es necesario. Si tuviese que imprimir una línea sería absurdo.

h. Los tipos de datos deberían estar declarados en el type.

NOTA

```
Program final 9/08/2022;

Const dimf = 5000; Añito: 2023;
type
    sub = 1..5
    participante = record                    fecha = record
                   dni: string[20];                  dia: integer
                   Apy Nombre: string[20];           mes: integer
                   Categoria: sub;                   año: Añito;
                   f: fecha                          end;

    porCate = Array[sub] of integer (El vector contador)
    lista = ^ nodo;
    nodo = record
           info: participante;
           sig: lista;
           end;
    inscriptos = Array [1.. dimF] of participante;
Procedure almacenar (var ins: inscriptos; dimL: integer); // Dispuesto

Procedure inicializar (var C: porCate);
Var i: integer
begin
        for i:= 1 to 5 do begin C[i]:=0;
end;
Procedure Contar (var C: porCate; ins: inscriptos; dimL: integer);
Var
    x: integer
Begin       for x:= 1 to dimL do C[ins[x]. categoria]:= C[ins[x]. categoria] + 1;
end;
```

```
Procedure AgregarAdelante ( p: participante ; var l: lista );
var nue: lista;
begin
        if (l <> nil) then begin
            new (nue);
            nue^.sig := L;
            nue^. info: = p;
            L:= nue;
        end
        else begin
            l^. info := p;
            l^. sig := nil;
        end;
end;
Procedure filtrar (var l: lista; c: tipoCate; ins: incriptos ; diml: integer);
var j: integer;
begin
            for j:= 1 to diml do
            if((c[ins[j]. categoria]) <= 50) then AgregarAdelante (ins[j], L);
end;
var diml: integer; l: lista; ins: inscriptos; c: tipoCate;
begin
        diml:= 0;
            L:= nil;
        almacenar (ins, diml);
        inicializar ( c);
        contar (c, ins, diml);
        filtrar (l, c, ins, diml);
end.
```

2. a. Falso. Es posible pero no una buena práctica. Las variables globales pueden ser utilizadas por todos los módulos. Están visibles y disponibles.

b. Falso. No siempre. La posición especifica puede ser no válida para un vector con dimensión lógica. También, el elemento a eliminar puede no estar en el vector.

C. Verdadero. puede no cumplir con el objetivo, respecto a lo que se espera que haga. Puede no necesitar modularización (una sola línea de código).

d. Falso. Se puede a través de por Ej el indice de vectores, de forma especifica.

3. 1 no especifica el campo del registro al cual se le asigmará el valor que entre por teclado.

2 C NO es un puntero

3. Es válido. Libera la posición en memoria a la que apunta aunque esta seguirá ocupada. No podra referenciarse ni utilizarse de nuevo.

4 Reserva un espacio en memoria a la que direccionará el puntero.

5 Copia la dirección en memoria a la que apunta Cli en Cli-esp.

6. Libera la posición en memoria a la que apunta pero puede volver a utilizarse y el contenido quedará inaccesible para Cli-esp.

7 eli-esp^ no podra acceder a Dmi sin acceder primero al campo del nodo. Ej Cli-esp^.dato.Dmi. Aunque estuviese bien escrito no tendría a qué acceder.

8 Lo mismo con 7. No encontrará el acceso a la variable a imprimir.

4. Variables locales. Solo pueden ser manipuladas en aquel módulo en el que se declaró.

Variables globales. El problema es que los nombres asignados a ellos pueden ser utilizados por diferentes personas. Conflictivo con respecto a los identificadores. Al referenciar y modificarla se verá reflejado luego en el resto del programa.

5.

Memoria Estatica

VECTOR p: 20 * 4 bytes

i : 4 bytes

categorias: 4 bytes (un integer de 1 al 5)

Ayn: 31 bytes

TOTAL 119

Memoria Dinámica

1er for

m: (10 - 1) + 1

10 x lo que pesa cliente

5 bytes del subrango

31 de cadena

8 de real

Primer for: 440 bytes

2do for → (10 NEW - 5 dispose) + 1.

→ 440 − 6× participante ?

→ − 440 / 264 / 176 bytes.

Tiempo de ejecución

1er for

m: (10 - 1) + 1 = 10

→ 3m + 2 + m. asignaciones = 3 . 10 + 2 + 10 . 4 = 168.

2do for m: 3 [(10 - 5) + 1] + 2 + [(10 - 5) + 1] . asignaciones =

= 20 + 6. asignaciones

Final 05/07/2022.

```
1. Program supermercado;
  type
      Rubro = 1..20;
      Producto= record
                  codigo: integer;
                  nombre: string[30];
                  ru:Rubro;
                  precio: real;
                  end;
      Lista1 = ^ nodo1;
      nodo1= reeord
               dato :producto;
               sig: lista1;
               end;
     Por Rubro = Array [rubro] of integer;
      lista2 = ^ nodo2;
      nodo2= Recoed
               dato: producto;
               sig: lista2;
               end;
Procedure leerProductos (var p: producto);

Procedure AgregarAtras (var L1, ult: lista2; p: producto);      se dispone
Procedure Cargar lista (var L1: lista1);
Inicializar Vector (var v: porRubro); var i: integer; begin for i:=1 to 20 do v[i]:=0 ; end;
Procedure Contar (var R: porRubro; L1 :lista1);
Begin    while (l1<>nil) do begin
                 r[ l1^.dato.ru] := r[ L1^.dato.ru ] + 1;
                 l1:= L1^.sig;
         end;
end;
```

```
Procedure insertarOrdenado (var L2: lista2; P: producto);
var amt, aux, act: lista2;
begin
        new(aux);
        aux^.dato := p;
        act := L2; amt := L2;
        while (act <> nil) and (act^.dato.cubro < aux^.dato.cubro)
        do begin
                amt := act;
                act := act^.sig;
        end;
        if (amt = act) then L2 := aux
        else amt^.sig := aux;
        aux^.sig := act
    end;
```

Agregar Atras o Adelante. No me pidieron orden.

```
Procedure recorrerL1 (Var l2: lista2; l1: lista1; e: Poecubro);
var
  Begin
        while (l1 <> nil) do begin

            if (e[L1^.dato.cu] = 10) then
                insertarordenado (L2, L^.dato);
        L1 := L1^.sig;
end;
end;

var L1: lista1; L2: lista2; e: Poecubro;
Begin
      cargar lista (L1);
      inicializarVectore (e);
      contar (e; l1);
      recorrer l1 (l2, l1, e);
end.
```

2. a. Verdadero. Es necesario, ya que, en el proceso de analizar, identificar y corregir errores, los casos límites podrían ser un Problema.

b. Falso. No siempre. Si, ya que, la cantidad de elementos es tanta como el vector permite, mediante la dimensión física no es necesario.

C. Falso. La función devuelve tipo de datos simples. El registro es un tipo de dato compuesto.

d. Falso. La modularización es una buena práctica. Puede see utilizado igualmente aunque haya variables globales.

3. 1. NO especifica el campo del registro al cual se le asigmará el valor que entra por teclado.

3. Falso. No se puede leer un puntero. La dirección podría see inválida.

2. C no es un puntero.

4. C no es un puntero.

5. Elimina el enlace. Deja de apuntar a la posición em memoria. a la que apunta, aunque esta seguirá ocupada. No podría utilizarse mi referenciarse. más tarde.

6. Se libera la posición en memoria y libera el enlace pero la posición podrá reutilizarse. Aunque el dispose sobre mil no tendría sentido.

7. Por el dispose em 6 el contenido quedó inaccesible para Cli. Así como la posición en memoria para guardar.

8. Imprimirá basura. No hay mada guardado en C. codigo. Pero es válida.

4. Es una estructura homogénea, Lineal. estática e indexada.

Esto ultimo significa que para acceder a cada elemento se debe utilizar una variable indice que es de tipo ordinal (subrango, boolean, integer, char).

homogénea porque los elementos que la componen son del mismo tipo.

Estática porque el tamaño no varía durante la ejecución del programa pero puede cambiar el valor de lo que se guarda en la posición de memoria. También es lineal porque c/ elemento le precede otro y uno le sigue.

La busqueda depende del orden del arreglo.

Si esta desordenado. Se recorre hasta que encuentre lo que busco o se termine el arreglo. De forma que mientras estas condiciones no sean cieetas, si lo que busco no esta en la posición actual, seguiré por las posiciones hasta encontrarlo. Al utilizar un vector devolverá finalmente un bodeano.

Si el arreglo esta ordenado se utilizará una busqueda dicotomica o mejorada. Consiste en calcular el elemento que esta en la posición del medio, si no encontré, seguiré por la mitad del arreglo que convenga según la condición

Memoria dinamica

For (10-1)+1 : 10.

10 x alumno: 10.25 : 250 bytes.

Memoria estática

vector: 10 x 4 bytes : 40. bytes

i : 4 bytes

sum: 4 bytes

nota: 4 bytes

Apenom: 21 bytes

Total 73.

Tiempo de ejecución

For 3N + 2 + N . operaciones N: (10-1) + 1 : 10.

3.10 + 2 + 10.3 : 62 UT.

sum := 0 1 UT

While N + 1 + N . operaciones (evaluar la ultima condición)

N+1 + N . 2 (suma y asigmación) UT : 3N+1.

Total : 62 UT + 1 UT + 3N + 1 UT.

El repeat until

N + N. Operaciones.

Final 04/06/2019.

```
program final;
Const      corte 9999;
type
     venta = record
               codVenta: integer;
               codProducto: integer;
               cant vendida: integer;
               end;
     Lista = ^nodo;
     nodo = record
               dato: venta;
               sig: lista;
               end;
Procedure  LeerVenta (var v: venta);
Begin
     With v do begin
        Readln(codventa);
        if (codVenta <> corte) then begin
        Readln(codProducto);
        Readln(cantvendida);
        end;
   end;

end;

Procedure  insertarOrdenado (var L:lista; v: venta);
Var ant, aux, act: lista;
begin
    new(aux);
    aux^.dato:= v;
    act:= L;
    ant:= L;
    while (act<>nil) and (act^.dato < aux^.dato) do begin
    ant:= act; act:= act^.sig; end;
    if (ant = act) then p:= aux. else ant^.sig:= aux;
    aux^.sig:= act
end;
```

```
Procedure CargarLista (var L:Lista);
var v: venta;
Begin
    leerVenta(v);
    while (v.codVenta <> Corte) do begin
        inseetarOrdenado (L, v);
        leer venta (v);
    end;
end;

Procedure recorrer lista ( L:lista ; CodProducto : integer);
var sumaProductos : integer; MismoCodproducto: integer;
Begin
    sumaProductos := 0;
    while (L<>nil) do begin
        while((L<>nil) and ( L^.dato.CodProducto <> CodProducto)) do
            L := L^.sig;
        if (L = nil) then writeln("El Codigo no existe");
        else begin
            while ( L^.dato.CodProducto = CodProducto) do begin
                sumaProductos := sumaProductos + L^.dato.CantVendida;
            end;
        end;
        writln ( sumaProductos);
end
var L: lista; Cod: integer
Begin

    cargarLista (L);
    readln( Cod );
    recorrerlista (L, Cod);
end.
```

HOJA N°

FECHA

2. Ambas (while y repeat) estructuras iterativas. En las dos, el n° de veces que se ejecutara un bloque de código se desconoce.
La diferencia es que while es precondicional (evaluara condición antes de ejecutarse) y el repeat PosCondicional (después). El repeat debido a esto se ejecutará al menos una vez, Mientras que el while podría nunca hacerlo. A demás while repetirá los pasos hasta que la sentencia sea falsa, poe el contrario, repear hasta que sea verdadera.
El foe es una estructura de control iterativa que se repite conociendo previamente m (cantidad de veces que ejecutará el bloque de código).

Problema en el caso de foe.

se necesita que se escriba en pantalla 5 veces.

Problema en el caso de while

Escribir en pantalla hasta que ingrese el número 5 el cual no debe procesarse.

" " until.

" " el cuál debe procesarse.

NOTA

4. Como el vectoe esta ordenado utilizaria una que devuelva la posición
bus que di cotomica, invocaría el modulo em
el proceso 3.

```
Procedure Dicotómica (var V: vector; diml: integer; dato: integer;
var posi integer);
Var pri, ult, medio: integer;
begin
    Pri:= 1; ult:= diml; medio:=(Pri + ult) div 2;
    while((pri <= ult) and (dato <> v[medio])) do. begin
        if (dato < v [medio]) then begin
            ult:= medio - 1; end
        else pri:= medio + 1;
        medio:= (Pri + ult). div 2;
    end;
    if (Pri = ult) then Pos:= -1. else Pos := medio;
end;
```

Teniendo ya la posición puedo elimina directamente el elemento.

```
Procedure Tres (var V: vectoe; Var diml: integer; dato: integee);
Var i: integee;
Begin
        dicotómica (V, diml, dato);
        foe i:= (pos + 1) to diml do.
            V[i-1]:= V[i];
        diml:= diml - 1;
end;
```

5. Final del 05/07/2022. Punto 4.

6. En La teoria del resumen.

Pregunta suelta Memoria y tiempo de ejecución.

Estática.

V: 10 × empleado [Empleado: ape-nom: 51 bytes. edad: 4 bytes. sueldo: 8 bytes.]

Como Empleado es la suma de sus campos.

V: 10 * 63 bytes : 630 bytes

TOTAL : 630 bytes + 63 bytes (por la var empleado) + 4 bytes (integer) + 4 bytes (Puntero) = 701.

Dinámica

1er for: [(8-1)+1] * Empleado : 8 * 63 bytes

For: 504 bytes.

2do for: [(4-2)+1] * e ¿0?

TOTAL : 504 bytes.

Tiempo de ejecución.

1er For.

For: 3.N + 2 + N. operaciones.

N: (8-1)+1 : 8.

⇒ For: 3.8 + 2 + 8. operaciones.

Operaciones: 1.

⇒ For: 3.8+2+8.1 : 34 UT

2do For.

For: 3.N+2+N. Operaciones.

N: (4-2)+1 : 3

⇒ For: 3.3 + 2 + 3. Operaciones.

Operaciones: 3.

⇒ For: 9 + 2 + 9 : 20. UT

TOTAL : 54. UT.

Si hay new o dispose afecta a la mem. dimámica.

Pregunta suelta 2.

Explicarlo mejor, no tan desordenado.

b. Verdadero. El subrango es un tipo de dato simple que toma un único valor (entero) dentro de los límites del subrango.

* Simple porque es un conjunto finito que guarda una relación. Al tener un maximo y mínimo en un grupo finito y ordenado. también es ordinal. ¿Hace falta decir esto?

La función retorna un único valor de tipo simple.

c. La variable a evaluar ha de ser un valor con equivalencia ordinal para la estructura Case. Es decir, un conjunto finito que guarda una relación. Teniendo un sucesor y predecesor.
Por ej. variable tipo entera o caracter (numero asociado por tabla Ascii).
En el caso de un String implicaría evaluar multiples expresiones, cosa que no sería válida, pero como st solo contiene un valor tipo Char es posible utilizar a st como variable de decisión.

Pregunta suelta 4

```
Const   dimF = 500;
type
    vector = Array[1..dimF] of integer
    Lista = ^nodo
    nodo = Record
        dato: integer
        sig: lista
    end;
Procedure modificar (Var v: vector, L: lista, var dimL: integer);
var
Begin
    dimL := 0;
    while((L<>nil) And (dimL<=dimF) do begin
        if(L^.dato mod 3 = 0) then begin

            v[dimL] := L^.dato;
            dimL := dimL + 1;

        end;
        L := L^.sig;
    end;
end;
```

HOJA Nº

FECHA

Final Reducido 2.

Ej. 4.

Dinámica	Estática
New en Repeat until	V: 10 * ^emple (puntero): 10 * 4 bytes : 40 bytes
N * empleado :	e: ape-nom + edad + sueldo: 12 bytes + 2 bytes + 6 bytes
N * 20 bytes.	: 20 bytes
	i : 2 bytes
	Total. 40 bytes + 20 bytes + 2 bytes : 62 bytes.

Tiempo de ejecución.

Repeat until : N + N. Operaciones.

Las operaciones son 3.

N + 3.N : 4N UT

while : N + 1 + N. Operaciones.

Operaciones. 5

⇒ : N + 1 + N. 5 : 6N + 1. UT

Total : 4N + 6N + 1 UT : 10N + 1. UT.

NOTA

HOJA Nº

FECHA

Final de julio 2022.

Si es un puntero válido o tiene memoria asignada. Si L es igual a mil B no modifica nada.

1. ~~Dependiendo si la lista se dispone~~ B, C, D estaría realizando la modificación del primer dato de la lista siempre. Mientras que A reserva espacio en memoria, guardaría el valor en la memoria y sobreescribiría el puntero. El C modificaría en memoria dinámica igual que con D pero la diferencia es que los cambios y modificaciones en punteros se verían reflejados en la lista por referencia. Así que para modificar el valor entero de la variable "dato" son correctas B, C y D.

2. Para minimizar el tiempo de ejecución debería utilizarse una lista. sería más rápido así redireccionar punteros que mover todos los elementos del vector para agregar al comienzo el nuevo empleado llevando una variable para limitar los empleados incrementando en 1 hasta que sean 100, por ejemplo.

3.

Memoria estática	Memoria dinámica
vector = 10 * 4 bytes.	M: (8 - 3) + 1 : 6
a: 16 string + 4 puntero	
i: 4 integer.	new (V[i]) : 20 bytes
Total 64 bytes	Total 6 * 20 : 120
	dispose - 20
	Total 100

Tiempo de ejecución.

M: (8-3) + 1 : 6

→ 3 . m + 2 + m : 3 . 6 + 2 + 6 asignaciones: 20 + 6 . 3 = 38

+ una asignación fuera del for: 39 UT

NOTA

4. utilizaría una función, pasaría la lista por valor así al recorrerla no perderé el puntero al primer/último modo.

un Booleano para devolver si hubo exito al encontrarlo.

El booleano inicializado en falso.
Mientras el ~~actual~~ no se mil y el valor del campo del registro al que apunta L sea mayor que el valor que busco... seguiré recorriendo la lista.
actual copiará la dirección del puntero siguiente
finalmente asigno la función el valor de la condición de si L no es nil y el dato es el valor que buscaba.

5. a. verdadero
La función devuelve una variable simple, y al ser invocada la es asignada a una variable del mismo tipo que devuelve 20+y dan como resultado otro entero ya que "y" es entera y el parámetro de la función admite un entero.

b. /if evalua condiciones no relacionadas mientras que case evalua una unica condición contra equivalentes posibles. Además para comparar varias condiciones se debería concatenar los if. mientras que en el case no. Falso/
X Era verdadero se definiría una variable booleana con la condicion del if
condicion := (x and y), case (condición).

c. - ya lo resolví. Verdadero

d. Falso si se tomase en cuenta la dimensión física y no la lógica el elemento podría no estar y se accedería en medio de la búsqueda a posiciones no válidas.

e. verdadero. La comunicación no sería posible ya que no habría datos que intercambiar. pero la invocación lo sería.

F. ya lo hice. Falso

NOTA