Arreglos

3

Objetivos de aprendizaje
Dominando los temas del presente capitulo Usted podrá.

1. Conocer la particularidad de los tipos de datos.
2. Definir y comprender claramente las estructuras indexadas
3. Dar valor a la eficiencia en las soluciones
4. Introducirse en el manejo de las estructuras conociendo mecanismos de recorrido, búsquedas y ordenamientos

Tipos de Datos

Identifica o determina un dominio de valores y el conjunto de operaciones aplicables sobre esos valores.

1. Primitivos.
2. Derivados.
3. Abstractos.

Los algoritmos operan sobre datos de distinta naturaleza, por lo tanto los programas que implementan dichos algoritmos necesitan una forma de representarlos.
Tipo de dato es una clase de objeto ligado a un conjunto de operaciones para crearlos y manipularlos, un tipo de dato se caracteriza por

1. Un rango de valores posibles.
2. Un conjunto de operaciones realizadas sobre ese tipo.
3. Su representación interna.

Al definir un tipo de dato se esta indicando los valores que pueden tomar sus elementos y las operaciones que pueden hacerse sobre ellos.
Al definir un identificador de un determinado tipo el nombre del identificador indica la localización en memoria, el tipo los valores y operaciones permitidas, y como cada tipo se representa de forma distinta en la computadora los lenguajes de alto nivel hacen abstracción de la representación interna e ignoran los detalles pero interpretan la representación según el tipo.

Como ya vimos, los tipos de datos pueden ser.

1. **Estáticos:** Ocupan una posición de memoria en el momento de la definición, no la liberan durante el proceso solamente la liberan al finalizar la aplicación.
    a. **Simples**: Son indivisibles en datos mas elementales, ocupan una única posición para un único dato de un único tipo por vez.

i. **Ordinales**: Un tipo de dato es ordinal o esta ordenado discretamente si cada elemento que es parte del tipo tiene un único elemento anterior (salvo el primero) y un único elemento siguiente (salvo el ultimo).
    1. **Enteros**: Es el tipo de dato numérico mas simple.
    2. **Lógico** o booleano: puede tomar valores entre dos posibles: verdadero o falso.
    3. **Carácter**: Proporcionan objetos de la clase de datos que contienen un solo elemento como valor. Este conjunto de elementos esta establecido y normatizado por el estándar ASCII.
ii. **No ordinales**: No están ordenados discretamente, la implementación es por aproximación
    1. Reales: Es una clase de dato numérico que permite representar números decimales.

b. **Cadenas**: Contienen N caracteres tratados como una única variable.

c. **Estructuras**: Tienen un único nombre para mas de un dato que puede ser del mismo tipo o de tipo distinto. Permiten acceso a cada dato particular y son divisibles en datos mas elementales.

Una estructura es, en definitiva, un conjunto de variables no necesariamente del mismo tipo relacionadas entre si de diversas formas.

Si los datos que la componen son todas del mismo tipo son homogéneas, heterogéneas en caso contrario.

Una estructura es estática si la cantidad de elementos que contiene es fija, es decir no cambia durante la ejecución del programa

i. **Registro**: Es un conjunto de valores que tiene las siguientes características:

   Los valores pueden ser de tipo distinto. Es una estructura heterogénea.

   Los valores almacenados se llaman campos, cada uno de ellos tiene un identificador y pueden ser accedidos individualmente.

   El operador de acceso a cada miembro de un registro es l operador punto (**.**)

   El almacenamiento es fijo.

ii. **Arreglo**: Colección ordenada e indexada de elementos con las siguientes características:

   Todos los elementos son del mismo tipo, un arreglo es una estructura homogénea.

   Los elementos pueden recuperarse en cualquier orden, simplemente indicando la posición que ocupa dentro de la estructura, esto indica que el arreglo es una estructura indexada.

   El operador de acceso es el operador []

   La memoria ocupada a lo largo de la ejecución del programa es fija, por esto es una estructura estática.

   El nombre del arreglo se socia a un área de memoria fija y consecutiva del tamaño especificado en la declaración.

   El índice debe ser de tipo ordinal. El valor del índice puede verse como el desplazamiento respecto de la posición inicial del arreglo.

   Los arreglos pueden ser de varias dimensiones. Esta dimensión indica la cantidad de índices necesarias para acceder a un elemento del arreglo.

   El arreglo lineal, con un índice, o una dimensión se llama vector.

El arreglo con 2 o mas índices o dimensiones es una matriz. Un grupo de elementos homogéneo con un orden interno en el que se necesitan 2 o mas índices para referenciar a un elemento de la estructura.

iii. **Archivos**: Estructura de datos con almacenamiento físico en memoria secundaria o disco.

Las acciones generales vinculadas con archivos son

Asignar, abrir, crear, cerrar, leer, grabar, Cantidad de elementos, Posición del puntero, Acceder a una posición determinada, marca de final del archivo, definiciones y declaraciones de variables.

Según su organización pueden ser secuenciales, indexados.

1. **Archivos de texto**: Secuencia de líneas compuestas por cero uno o mas caracteres que finalizan con un carácter especial que indica el final de la línea. Los datos internos son representados en caracteres, son mas portables y en general mas extensos.
2. **Archivos de tipo o binarios**: secuencia de bytes en su representación interna sin interpretar. Son reconocidos como iguales si son leídos de la forma en que fueron escritos. Son menos portables y menos extensos.

2. **Dinámicos:** Ocupan direcciones de memoria en tiempo de ejecución y se instancian a través de punteros. Esta s instancias pueden también liberarse en tiempo de ejecución. El tema de puntadores y estructuras enlazadas (estructuras relacionadas con este tipo de dato se analizan en detalle en capítulos siguentes)
    a. **Listas simplemente enlazadas**: cada elemento sólo dispone de un puntero, que apuntará al siguiente elemento de la lista o valdrá NULL si es el último elemento.
    b. Pilas: son un tipo especial de lista, conocidas como listas LIFO (Last In, First Out: el último en entrar es el primero en salir). Los elementos se "amontonan" o apilan, de modo que sólo el elemento que está encima de la pila puede ser leído, y sólo pueden añadirse elementos encima de la pila**.**
    c. **Colas**: otro tipo de listas, conocidas como listas FIFO (First In, First Out: El primero en entrar es el primero en salir). Los elementos se almacenan en fila, pero sólo pueden añadirse por un extremo y leerse por el otro.
    d. **Listas circulares**: o listas cerradas, son parecidas a las listas abiertas, pero el último elemento apunta al primero. De hecho, en las listas circulares no puede hablarse de "primero" ni de "último". Cualquier nodo puede ser el nodo de entrada y salida.
    e. **Listas doblemente enlazad**as: cada elemento dispone de dos punteros, uno a punta al siguiente elemento y el otro al elemento anterior. Al contrario que las listas abiertas anteriores, estas listas pueden recorrerse en los dos sentidos.
    f. **Árboles**: cada elemento dispone de dos o más punteros, pero las referencias nunca son a elementos anteriores, de modo que la estructura se ramifica y crece igual que un árbol.
    g. **Árboles binarios**: son árboles donde cada nodo sólo puede apuntar a dos nodos.
    h. **Árboles binarios de búsqueda** (ABB): son árboles binarios ordenados. Desde cada nodo todos los nodos de una rama serán mayores, según la norma que se haya seguido para ordenar el árbol, y los de la otra rama serán menores.
    i. **Árboles AVL**: son también árboles de búsqueda, pero su estructura está más optimizada para reducir los tiempos de búsqueda.

j. **Árboles B**: son estructuras más complejas, aunque también se trata de árboles de búsqueda, están mucho más optimizados que los anteriores.
k. **Tablas HASH**: son estructuras auxiliares para ordenar listas.
l. **Grafos**: es el siguiente nivel de complejidad, podemos considerar estas estructuras como árboles no jerarquizados.
m. **Diccionarios**.

Registros y vectores

Registro: Es un conjunto de valores que tiene las siguientes características:
Los valores pueden ser de tipo distinto. Es una estructura heterogénea.
Los valores almacenados se llaman campos, cada uno de ellos tiene un identificador y pueden ser accedidos individualmente.
El operador de acceso a cada miembro de un registro es l operador punto (.)
El almacenamiento es fijo.
Declaracion
Genérica
NombreDelTipo = TIPO < $TipoDato_1$ $Identificador_1$; …; $TipoDato_N$ $Identificador_N$>

TipoRegistro = TIPO <Entero N; Real Y> //declara un tipo
TipoRegistro Registro; // define una variable del tipo declarado
En C

```
 struct NombreTipo {
        Tipo Identificador;
        Tipo Identificador;
}
struct TipoRegistro {
        int N;
        double Y;
};                      //      declara un tipo
TipoRegistro Registro; //       define una variable
```

Ejemplo de estructuras anidadas en C

```
struct TipoFecha {
        int     D;
        int     M;
        int     A;
};                      //      declara un tipo fecha

struct TipoAlumno {
        int             Legajo;
        string          Nombre;
        TipoFecha       Fecha
};                      //      declara un tipo Alumno con un campo de tipo Fecha
TipoAlumno Alumno;
```

Alumno es un registro con tres miembros (campos) uno de los cuales es un registro de TipoFecha.
El acceso es:

Nombre	Tipo dato	
Alumno	Registro	Registro total del alumno
Alumno.Legajo	Entero	Campo legajo del registro alumno que es un entero
Alumno.Nombre	Cadena	Campo nombre del registro alumno que es una cadena
Alumno.Fecha	Registro	Campo fecha del registro alumno que es un registro
Alumno.Fecha.D	Entero	Campo dia del registro fecha que es un entero
Alumno.Fecha.M	Entero	Campo mes del registro fecha que es un entero
Alumno.fecha.A	Entero	Campo anio del registro alumno que es un entero

Arreglo: Colección ordenada e indexada de elementos con las siguientes características:

- Todos los elementos son del mismo tipo, un arreglo es una estructura homogénea.
- Los elementos pueden recuperarse en cualquier orden, simplemente indicando la posición que ocupa dentro de la estructura, esto indica que el arreglo es una estructura indexada.
- El operador de acceso es el operador []
- La memoria ocupada a lo largo de la ejecución del programa es fija, por esto es una estructura estática.
- El nombre del arreglo se socia a un área de memoria fija y consecutiva del tamaño especificado en la declaración.
- Al elemento de posición genérica i le sigue el de posición i+1 (salvo al ultimo) y lo antecedede el de posición i-1 (salvo al primeo.
- El índice debe ser de tipo ordinal. El valor del índice puede verse como el desplazamiento respecto de la posición inicial del arreglo. La posición del primer elemento en el caso particular de C es 0(cero), indica como se dijo el desplazamiento respecto del primer elemento, para este caso es nana. En un arreglo de N elemento, la posición del ultimo es N – 1, por la misma causa.
- Los arreglos pueden ser de varias dimensiones. Esta dimensión indica la cantidad de índices necesarias para acceder a un elemento del arreglo.
- El arreglo lineal, con un índice, o una dimensión se llama vector.
- El arreglo con 2 o mas índices o dimensiones es una matriz. Un grupo de elementos homogéneo con un orden interno en el que se necesitan 2 o mas índices para referenciar a un elemento de la estructura.
- En el caso de C, no hay control interno para evitar acceder a un índice superior al tamaño físico de la estructura, esta situación si tiene control en C++, mediante la utilización de at pata el acceso (se vera mas adelante).

Declaración

Genérica
Nombre del Tipo = TABLA [Tamaño] de Tipo de dato;
ListaEnteros = TABLA[10] de Enteros; // una tabla de 10 enteros
ListaRegistros = TABLA[10] de TipoRegistro; // una tabla de 10 registros
En C:

```
TipoDeDato Identificador[Tamaño];
int VectorEnteros[10]// declara un vector de 10 enteros
TipoAlumno VectorAlum[10] // declara un vector de 10 registros.
```

Accesos

Nombre	Tipo dato	
VectorAlum	Vector	Vector de 10 registros de alumnos
VectorAlum[0]	Registro	El registro que esta en la posición 0 del vector
VectorAlum[0].Legajo	Entero	El campo legajo del registro de la posición 0
VectoeAlum[0].Fecha	Registro	El campo fecha de ese registro, que es un registro
VectorAlum[0].Fecha.D	Entero	El campo dia del registro anterior

En C++, incluye la clase <array>

```
#include <array>

// Declaración generica array<tipo de dato, cantidad elementos> identificador;
array<int,10> ArrayEnteros; // declara un array de 10 enteros
array<TipoAlumno, 10> ArrayRegistros //declara array de 10 registros

Iteradores
begin           Return iterador al inicio
end             Return iterador al final
 std::array<int,5> miarray = { 2, 16, 77, 34, 50 };
 for ( auto it = myarray.begin(); it != myarray.end(); ++it )

Capacidad
size            Return tamaño
 std::array<int,5> miarray;
 std::cout << miarray.size() << std::endl;
 // retorna 5

Elementos de  acceso
operator[]      Acceso a un elemento
at              Acceso a un elemento
front           Acceso al primero de los elementos
back            Acceso al ultimo de los elementos
data

 std::array<int,4> myarray = {10,20,30,40};
   std::cout << myarray[1];//muestra 10
   std::cout << myarray.at(2);// muestra 20
   std::cout << myarray.front();// muestra 10
   std::cout << myaray.back(;// muestra 40
```

Acciones y funciones para vectores

BusqSecEnVector

(Dato V: Tvector; Dato N: Entero; Dato Clave:Tinfo; Dato_resultado Posic: Entero): una accion

Usar este algoritmo si alguna de las otras búsquedas en vectores mas eficientes no son posibles, recordando que búsqueda directa tiene eficiencia 1, búsqueda binaria es logarítmica y búsqueda secuencial es de orden N

```
 PRE:    V: Vector en el que se debe buscar
         Clave : Valor Buscado
         N: Tamaño lógico del vector
         U: = N -1 posición del ultimo, uno menos que el tamaño del vector
POS:     Posic: Posición donde se encuentra la clave, -1 si no esta.
LEXICO
j : Entero;
ALGORITMO
         Posic = 0;
         j = 0; //Pone el indice en la primera posición para recorrer el vector//
         MIENTRAS (j  <=  MAX_FIL  y  j <= U  y  V[j] <> Clave) HACER
                  Inc (j) //Incrementa el indice para avanzar en la estructura//
         FIN_MIENTRAS;
         SI (j > N)
         ENTONCES
                  Posic = -1 // No encontró la clave buscada
         SI_NO
                  Posic = j // Encontró la clave en la posición de índice j
FIN_SI;

FIN. // Búsqueda secuencial En Vector
```

Controla No superar el tamaño fisico del vector j<= MAX_FIL

No leer mas alla del ultimo elemento logico cargado j <= N

BusqMaxEnVector

(Dato V: Tvector; Dato N: Entero; Dato_resultado Maximo :Tinfo; Dato_resultado Posic: Entero): una acccion

```
PRE:     V: Vector en el que se debe buscar (sin orden)
         N : Tamaño lógico del vector
         U = N-1 Posicion del ultimo elemento
POS:     Posic: Posición donde se encuentra el máximo
Maximo : Valor máximo del vector.
LEXICO
j : Entero;
ALGORITMO
         Posic = 0;
         Maximo = V[1];
         PARA j [1, U] HACER
                  SI (v[j] > Maximo)
                  ENTONCES
                           Posic = j;
                           Maximo = v[j];
                  FIN_SI;
         FIN_PARA;
```

Supone que el maximo es el primer valor del vector por lo que le asigna ese valor a maximo y la posición 0 a la posición del maximo. Al haber leido solo un elemento supone ese como maximo

Recorre ahora las restantes posiciones del vector, a partir de la segunda y lcada vez que el valor leido supera al maximo contiene ese valor como maximo y el indice actual como posición del maximo

FIN. // Búsqueda máximo En Vector

BusqMinDistCeroEnVector

(Dato V: Tvector; Dato N: Entero; Dato_resultado Minimo :Tinfo; Dato_resultado Posic: Entero): una acccion

PRE: V: Vector en el que se debe buscar (sin orden)
N : Tamaño lógico del vector, existe al menos un valor <> de cero
U = N-1 Posicion del ultimo elemento

POS: Posic: Posición donde se encuentra el minimo distinto de cero

Minimo : Valor minimo distinto de cero del vector.

```
LEXICO
i,j  : Entero;
ALGORITMO
        //
        J = o;
Mientras (J<=U) Y (V[j] = 0) Hacer
        Incrementar[j];
Posic = J;
        Minimo = V[j];
        PARA j [Posic.+1 , N] HACER
                SI (v[j]<> 0 Y v[j] < Minimo)
                ENTONCES
                        Posic = j;
                        Minimo = v[j];
                FIN_SI;
        FIN_PARA;
```

Recorre el vector hasta encontrar el primero distinto de cero. Al encontrarlo supone ese valor como minimo y el valor del indice como posición del minimo

Recorre el vector desde la posición inmediata siguiente hasta la ultima desplazando el minimo solo si el valor es distinto de cero y, ademas, menor aue el minimo

FIN. // Búsqueda minimo distinto de cero En Vector

BusqMaxySiguienteEnVector

(Dato V: Tvector; Dato N: Entero; Dato_resultado Maximo :Tinfo; Dato_resultado Posic: Entero, Dato_resultado Segundo :Tinfo; Dato_resultado PosicSegundo: Entero): una accion

PRE: V: Vector en el que se debe buscar
N : Tamaño lógico del vector mayor o gual a 2
U = N - 1

POS: Posic: Posición donde se encuentra el máximo, PosicSegundo: Posición donde se encuentra el siguiente al máximo

Maximo : Valor máximo del vector. Segundo : Valor del siguiente al máximo del vector

```
LEXICO
j : Entero;
ALGORITMO
        SI V[0] > V[1]
ENTONCES
        Posic = 0;
        Maximo = V[0];
        PosicSegund = 1;
        Segundo = V[1];
```

Se tiene como precondicion que al menos hay dos valores. Se verifica el valor que esta en la primera posición y se lo compara con el que esta en la segunda posición, en el caso de ser mayor, el maximo es ese valor, posición del máximo es cero, el segundo el valor que esta en segundo lugar y posición del segundo es 1. En caso contrario se establece como maximo el valor de la segunda posición y segundo el de la primera.

```
SINO
        Posic = 1;
        Maximo = V[1];
        PosicSegund =0;
        Segundo = V[0];

FIN_SI

PARA j [2, U] HACER
        SI (v[j] > Maximo)
        ENTONCES
                Segundo = Maximo;
                PosicSegundo = Posic;
                Posic = j;
                Maximo = v[j];
        SINO
                SI Maximo>Segundo
                ENTONCES
                        Segundo = V[j];
                        PosicSegundo = j
                FIN_SI
        FIN_SI;
FIN_PARA;

FIN. // Búsqueda máximo En Vector
```

Se verifica luego desde la tercera posición hasta el final. En el caso que el nuevo valor sea mayor que el maximo, se debe contener el anterior maximo en el segundo y al maximo se le asigna el nuevo valor. Cosa similar hay que hacer con las posiciones. Si esto no ocurriera se debe verificar si el

CargaSinRepetirEnVectorV1

(Dato_Resultado V: Tvector; Dato_Resultado N: Entero; Dato Clave:Tinfo; Dato_resultado Posic: Entero; Dato_resultado Enc : Booleano): una accion

Utilizar este algoritmo si la cantidad de claves diferentes es fija, se dispone de memoria suficiente como para almacenar el vector y la clave no es posicional(es decir clave e índice no se corresponden directamente con posición única y predecible.Si la posición fuera unica y predecible la busqueda debe ser directa

PRE: V: Vector en el que se debe buscar
Clave : Valor Buscado
N : Tamaño lógico del vector
U = N-1 posicion del ultimo elemento

POS: Posic: Posición donde se encuentra la clave, o donde lo inserta si no esta. Retorna -1 (menos 1) en caso que el vector esta completo y no lo encuentra
Enc : Retorna True si estaba y False si lo inserto con esta invocación
Carga vector sin orden

```
LEXICO
j : Entero;
ALGORITMO/
        Posic = -1;
        J = 0;
        MIENTRAS (j  <=  MAX_FIL  y  j <= U  y  V[j] <> Clave) HACER
                Inc (j)
```

Controla No superar el tamaño fisico del vector j<= MAX_FIL

No leer mas alla del ultimo elemento logico cargado j <= N

```
        FIN_MIENTRAS;
        SI j > MAX_FIL
        ENTONCES
                Posic = -1
        SI_NO
                Posic = j:
                SI (j > N)
                ENTONCES
                        Enc =FALSE; // No encontró la clave buscada
                        Inc(N);
                        V[N] = Clave;
                SI_NO
                        Enc = True // Encontró la clave en la posición de índice j
                FIN_SI;
        FIN_SI
FIN. // Carga sin repetir en vector
```

Si debio superar el tamaño fisico maximo del vector no pudo cargarlo y retorna cero como señal de error

Si encontro un dato o lo debe cargar esto es en el indice j por lo que a pos se se asigna ese valor. En el caso que j sea mayor que n significa que recorrio los n elemntos cargados del vector, tiene etpacio por lo que debe cargar el nuevo en la posición j. En este caso, y al haber un elemento nuevo debe incrementar n que es el identificador que controla el tamaño logico

BusquedaBinariaEnVectorV1(Dato V: Tvector; Dato N: Entero; Dato Clave:Tinfo; Dato_resultado Posic: Entero; Dato_resultado Pri : Entero): una accion

Utilizar este algoritmo si los datos en el vector están ordenados por un campo clave y se busca por ese campo. Debe tenerse en cuenta que si la clave es posicional se deberá utilizar búsqueda directa ya que la diferencia en eficiencia esta dada entre 1, para la búsqueda directa y $log_2 N$ para la binaria

PRE: V: Vector en el que se debe buscar con clave sin repetir
Clave : Valor Buscado
N : Tamaño lógico del vector

POS: Posic: Posición donde se encuentra la clave, o -1 (menos 1) si no esta
Pri : Retorna la posición del limite inferior

```
LEXICO
j : Entero;
u,m : Entero;
ALGORITMO
        Posic = -1;
        Pri = ;
        U = N-1;
        MIENTRAS (Pri < = U  y  Pos = -1) HA
                M = (Pri + U ) / 2
                SI V[M] = Clave
                ENTONCES
                        Posic = M;
                SI_NO
                        SI Clave > V[M]
                        ENTONCES
                                Pri = M+1
                        SI_NO
                                U = M – 1
                        FIN_SI
```

Establece valores para las posiciones de los elementos del vector, Pri contiene el indice del primero, es decir el valor 1, U el indice del ultimo elemento logicio, es decir N. Ademas se coloca en Posic. El valor cero, utilizando este valor como bandera para salir del ciclo cuando encuentar el valor buscado

Permanece en el ciclo mientras no encuentre lo buscado, al encontrarlo le asigna a pos el indice donde lo encontro, como es un valor > que cero hace false la expresión logica y sale del ciclo. Si no lo encuentra, y para evirtar ciclo infinito verifica que el primero no tome un valor mayor que el ultimo. Si eso ocurre es que el dato buscado no esta y se debe salir

Si el dato buscado lo encuentra le asigna a posición el indice para salir. Si no lo encuentra verifica si esmayor el dato buscabo a lo que se encuentra revisa en la mitad de los mayores por lo que le asigna al primero el indice siguiente al de la mitad dado que alli no estab y vuelve a dividir el conjunto de datos en la mitas, de ser menor pone como tome ultimo el anterior al de la mitad actual

```
            FIN_SI
        FIN_MIENTRAS;

FIN. // Búsqueda binaria en vector
```

BusquedaBinariaEnVectorV2

(Dato V: Tvector; Dato N: Entero; Dato Clave:Tinfo; Dato_resultado Posic: Entero; Dato_resultado Pri : Entero): una acccion

PRE: V: Vector en el que se debe buscar clave puede estar repetida
Clave : Valor Buscado
N : Tamaño lógico del vector

POS: Posic: Posición donde se encuentra la primera ocurrencia de la clave.
0 (cero) si no esta.
Pri : Retorna la posición del limite inferior

```
LEXICO
j : Entero;
u,m : Entero;
ALGORITMO
        Posic = -1;
        Pri = 1;
        U = N-1;
        MIENTRAS (Pri <  U ) HACER
                M = (Pri + U ) / 2
                SI V[M] = Clave
                ENTONCES
                        Posic = M;
                        Pri = M;
                SI_NO
                        SI Clave > V[M]
                        ENTONCES
                                Pri = M+1
                        SI_NO
                                U = M – 1
                        FIN_SI
                FIN_SI
        FIN_MIENTRAS;

FIN. // Búsqueda binaria en vector
```

La busqueda es bastante parecida a lo desarrollado anteriormente, pero en pos debe tener la primera aparicion de la clave buscada que puede repetirse.

En la busqueda anterior utilizabamos esta pos como bandera, para saber cuando Sali si lo encontro. En este caso si lo utilizamos con el mismo proposito saldria cuando encuentra un valor oincidente con la clave que no necesariamente es el primero, por lo que esa condicion se elimina. Al encontrarlo en m se le asigna ese valoe a pos, alli seguro esta. No sabemos si mas arriba vuelve a estar por lo que se asigna tambien esa posición al ultimo para seguir iterando y ver si lo vuelve a encontrar. Debe modificarse el operador de relacion que compara primero con ultimo para evitar un ciclo infinito, esto se hace eliminando la relacion por igual. Insisto en el concepto de los valores de retorno de la busqueda binaria. Una particularidad de los datos es que si lo que se busca no esta puede retornal en el primero la posición donde esa clave deberia estar

CargaSinRepetirEnVectorV2

(Dato_Resultado V: Tvector; Dato_Resultado N: Entero; Dato Clave:Tinfo; Dato_resultado Posic: Entero; Dato_resultado Enc : Booleano): una acccion

PRE: V: Vector en el que se debe buscar ordenado por clave
Clave : Valor Buscado
N : Tamaño lógico del vector

La busqueda es bastante parecida a lo desarrollado anteriormente, pero en pos debe tener la primera aparicion de la clave buscada que puede repetirse.
En la busqueda anterior utilizabamos esta pos como bandera, para saber cuando Sali si lo encontro. En este caso si lo utilizamos con el mismo proposito saldria cuando encuentra un valor oincidente con la clave que no necesariamente es el primero, por lo que esa condicion se elimina. Al encontrarlo en m se le asigna ese valoe a pos, alli seguro esta. No sabemos si mas arriba vuelve a estar por lo que se asigna tambien esa posición al ultimo para seguir iterando y ver si lo vuelve a encontrar. Debe modificarse el operador de relacion que compara primero con ultimo para evitar un ciclo infinito, esto se hace eliminando la relacion por igual. Insisto en el concepto de los valores de retorno de la busqueda binaria. Una particularidad de los datos es que si lo que se busca no esta puede retornal en el primero la posición donde esa clave deberiaestar

POS: Posic: Posición donde se encuentra la clave, o donde lo inserta si no esta. Retorna -1 (menos 1) en caso que el vector esta completo y no lo encuentra
Enc : Retorna True si estaba y False si lo inserto con esta invocación
Carga vector Ordenado

LEXICO
j : Entero;
U = N-1;
ALGORITMO
Enc = True;
BusquedaBinariaEnVectorV(V; N; Clave; Posic; Pri)
SI (Posic = -1)
ENTONCES
Enc = False ;
Posic = Pri;
PARA j [U, Pri](-) HACER
V[j+1] = V[j];
FIN_PARA;
V[Pri] = Clave;
Inc(N);
FIN_SI
FIN. // Carga sin repetir en vector Versión 2. con vector ordenado

Al estar el vector ordenadola busqueda puede ser binaria, si lo encuentra retorna en posición un valor mayor a cero. Si no lo encuentra el valor de posición sera -1. En este caso, se conoce que en pri es en la posición donde este valor debe estar

Se produce un desplazamiento de los valores desde el ultimo hasta el valor de pri corriendolos un lugar para poder insertar en la posición pri el nuevo valos. Al pasar por aquí se inserto un nuevo elemento por lo que n, que contiene la cantidad de elementos del vector debe incrementarse en uno

OrdenarVectorBurbuja

(Dato_Resultado V: Tvector; Dato N: Entero): una acccion
Pre: V: Vector en el que se debe ordenar, se supone dato simple
N : Tamaño lógico del vector
U = N-1 // posición del ultimo elemento del vector.
POS: Vector ordenado por clave creciente

Usar este algoritmo cuando los datos contenidos en un vector deben ser ordenados. Se podría por ejemplo cargar los datos de un archivo al vector, ordenar el vector recorrerlo y generar la estructura ordenada. Para esto la cantidad de elementos del archivo debe ser conocida y se debe disponer de memoria suficiente como para almacenar los datos. Una alternativa, si la memoria no alcanza para almacenar todos los datos podría ser guardar la clave de ordenamiento y la referencia donde encontrar los datos, por ejemplo, la posición en el archivo.

LEXICO
I,J, : Entero;

La idea general es ir desarrollando pasos sucesivos en cada uno de los cuales ir dejando el mayor de los elementos en el último lugar. En el primer paso se coloca el mayor en la ultima posición, en el paso siguiente se coloca el que le sigue sobre ese y asi hasta que queden dos elemntos. En ese caso al acomodar el segundo el otro queda acomodado el primer ciclo cuenta los pasos, son uno menos que la cantidad de elementos porque el ultimo paso permite acomodar 2 elementos, por eso el ciclo se hace entre 1 y N – 1 siendo n la cantidad de elementos

```
Aux : Tinfo;
ALGORITMO
        PARA i [1, U - 1] HACER
                PARA j [1, U - i] HACER
                 SI (v[j] > v[j + 1])
                   ENTONCES
                     Aux = v[j];
                     V[j] = v[j + 1];
                     V[j + 1] = Aux;
                 FIN_SI;
                FIN_PARA;
        FIN_PARA;
FIN
```

Para poder colocar el mayor al final es necesario hacer comparaciones. Se compara el primero con el segundo y si corresponde se intercambian. Asi hasta llegar al ante ultimo eleemento que se lo compara con el últim.

Al ir recorriendo los distintos pasos, y dado que en cada uno se acomoda un nuevo elemento corresponde hacer una comparación menos en cada avance, como los pasos los recorremos con i, las comparaciones seran U – i. disminuye en 1 en cada paso

OrdenarVectorBurbujaMejorado

(Dato_Resultado V: Tvector; Dato N: Entero): una acccion
PRE: V: Vector en el que se debe ordenar, se supone dato simple
N : Tamaño lógico del vector
U = N - 1
POS: Vector ordenado por clave creciente

```
LEXICO
I,J, : Entero;
Aux : Tinfo;
Ord : Boolean;
ALGORITMO
        I = 0;
REPETIR
        Inc(i);
                Ord = TRUE;
                PARA j [1, U - i] HACER
                        SI (v[j] > v[j + 1])
                        ENTONCES
                                Ord = False;
Aux = v[j];
                                V[j] = v[j + 1];
                                V[j + 1] = Aux;
                        FIN_SI;
                FIN_PARA;
        HASTA ( Ord   o I = U – 1); //si esta ordenado o llego al final
FIN
```

El algoritmo es similar al anterior, solo que el ciclo de repetición externo no lo hace si es que tiene la certeza, en el paso anterior que los datos ya estan ordenados. Es por eso que agrega una bandera para verificar si ya esta ordenado y cambia el cilo exactp por un ciclo pos condicional. Es decir reemplaza la composición para por la composición repetir hasta

OrdenarVectorInserion

(Dato_Resultado V: Tvector; Dato N: Entero): una accion
PRE: V: Vector en el que se debe ordenar, se supone dato simple
N : Tamaño lógico del vector
POS: Vector ordenado por clave creciente

Este algoritmo consta de los siguientes pasos
El primer elemento A[0] se lo considera ordenado; es decir se considera el array con un solo elemento.
Se inserta A[1] en la posicion correcta, delante o detras de A[0] segun sea mayor o menor.
Por cada iteracion, d i desde i=1 hasta n-1, se explora la sublista desde A[i-1] hasta A[0],buscando la posicion correcta de la insercion ; a la vez se mueve hacia abajouna posicion todos los elementos mayores que el elemento a insertar A[i] para dejar vacia la posicion.
Insertar el elemento en l posicion correcta.

```
LEXICO
I,J, : Entero;
Aux : Tinfo;

ALGORITMO
        PARA I[1..N-1]
                J = I;
                Aux = A[i] ;
                MIENTRAS (J >0 Y AUX < A[J - 1])HACER
                        A[J] = A[J - 1] ;
                        Dec(J) ;
                FIN MIENTRAS ;
        A[J] = Aux ;
        FIN PARA;
FIN
```

OrdenarVectorShell

(Dato_Resultado V: Tvector; Dato N: Entero): una acccion
PRE: V: Vector en el que se debe ordenar, se supone dato simple
N : Tamaño lógico del vector
POS: Vector ordenado por clave creciente
Este algoritmo consta de los siguientes pasos
Dividir la lista original en n/2 grupos de dos, considerando un incremento o salto entre los elementos en n/2.
Analizar cada grupo por separado comparandolas parejas de elementos, y si no estan ordenados, se intercambian .
Se divide ahora la lista en la mitad n/4, con un incremento tambien en n/4 y nuevamente se clasifica cada grupo por separado.
Se sigue dividiendo la lista en la mitad de grupos que en el paso anterior y se clasifica cada grupo por separado.
El algoritmo termina cuando el tamaño del salto es 1.

```
ALGORITMO
        Intervalo = n / 2;

MIENTRAS Intervalo > 0 HACER
                PARA I [Intervalo + 1 .. N] HACER
                        MIENTRAS (J > 0) HACER
```

```
K = J + Intervalo;
                                    SI (A[J] <= A[K]
                                    ENTONCES
                                            J = -1
                                    SINO
                                            Intercambio(A[J] ,A[K]) ;
                                            J = J – Intervalo ;
                                    FINSI ;
                        FIN PARA ;
                FIN MIENTRAS;
FIN.
```

CorteDeControlEnVector

(Dato V:Tvector; Dato N: Entero): una acccion

Usar este procedimiento solo si se tienen los datos agrupados por una clave común y se requiere procesar emitiendo información por cada subconjunto correspondiente a cada clave.

PRE: V: Vector en el que se debe Recorrer con corte de control
Debe tener un elemento que se repite y estar agrupado por el.
N : Tamaño lógico del vector
U = N-1

POS: Recorre agrupando por una clave

```
LEXICO
I : Entero;

ALGORITMO
        I = 0;
        Anterior = TipoInfo;
        // Inicializar contadores generales
      MIENTRAS (I<=U) Hacer
          //inicializar contadores de cada sublote
         Anterior = V[i]
          MIENTRAS (I<=U Y Anterior = V[i] HACER
                  // Ejecutar acciones del ciclo

                  I = I+1 // avanza a la siguiente posición
         FIN_MIENTRAS
         // Mostrar resultados del sublote
     FIN_MIENTRAS
     // Mostrar resultados generales
FIN
```

ApareoDeVectores

(Dato V1,v2:Tvector; Dato N1,N2: Entero): una acccion

Utilizar este procedimiento si se tiene mas de una estructura con un campo clave por el que se los debe procesar intercalado y esas estructuras están ORDENADAS por ese campo común.

PRE: V1,V2: Vectores a Recorrer mezclados o intercalados

```
        Los vectores deben estar ordenados.
          N1,N2 : Tamaño lógico de los vectores
|         U1 = N1-1; U2 = N2 – 1 //posición de los últimos elementos de V1 y V2
POS:    Muestra la totalidad de los datos con el orden de las estructuras


LEXICO
I,J : Entero;
ALGORITMO
        I = 0;
        J = 0;

      MIENTRAS (I<=U1 o J<=U2) Hacer
        SI((J > U2) o ((I<=U1) y (V1[I]<V2[J])) HACER
        ENTONCES
                Imprimir (V1[I]);
                I = I + 1;
        SINO
                Imprimir(V2[J]);
                J = J + 1;
        FIN_SI;
      FIN_MIENTRAS

FIN
```

CargaNMejoresEnVector

```
(Dato_Resultado V: Tvector; Dato_Resultado N: Entero; Dato Clave:Tinfo): una acccion
PRE:    V: Vector en el que se debe buscar e insertar los mejores
        Clave: Valor Buscado
        N: Tamaño lógico del vector
        U = N-1 //p0sicion del ultimo elemento en el vector
POS:    Vector con los N mejores sin orden
LEXICO
j : Entero;
Maximo: Tinfo
Posic: Entero;
ALGORITMO
        SI (U < MAX-FIL)
        ENTONCES
                Inc (U);
                V[U] = Clave
        SI_NO
        BusqMaxEnVector(V;  N; Maximo :Tinfo; Posic: Entero);
                SI (Clave > Maximo)
                ENTONCES
                        V[Posic] = Clave;
                FIN_SI;
        FIN_SI;
FIN. // Carga los N mejores en vector
```

Implementaciones C C++
Operaciones sobre arrays

Agregar un elemento al final de un array

La siguiente función agrega el valor `v` al final del array `arr` e incrementa su longitud `len`.

```
void agregar(int arr[], int& len, int v)
{
      arr[len]=v;
      len++;
      return;
}
```

Recorrer y mostrar el contenido de un array

La siguiente función recorre el array `arr` mostrando por consola el valor de cada uno de sus elementos.

```
void mostrar(int arr[], int len)
{
      for(int i=0; i<len; i++){
        cout << arr[i] << endl;
      }
  return;
}
```

Busqueda secuencial en un vector

La siguiente función permite determinar si el array `arr` contiene o no al elemento `v`; retorna la posición que `v` ocupa dentro de `arr` o un valor negativo si `arr` no contiene a `v`.

```
int buscar(int arr[], int len, int v)
{
      int i=0;
      while( i<len && arr[i]!=v ){
       i++;
      }
  return i<len?i:-1; }
```

Eliminar el valor que se ubica en una determinada posición del array

La siguiente función elimina el valor que se encuentra en la posición `pos` del array `arr`, desplazando al i-ésimo elemento hacia la posición i-1, para todo valor de i>`pos` y i<`len`.

```
void eliminar(int arr[], int& len, int pos)
{
      for(int i=pos; i<len-1; i++ ){
       arr[i]=arr[i+1];
      }
      // decremento la longitud del array
      len--;
  return;
}
```

Insertar un valor en una determinada posición del array

La siguiente función inserta el valor `v` en la posición `pos` del array `arr`, desplazando al i-ésimo elemento hacia la posición i+1, para todo valor de i que verifique: i>=`pos` e i<`len`.

```
void insertar(int arr[], int& len, int v, int pos)
{
      for(int i=len-1; i>=pos; i--){
        arr[i+1]=arr[i];
      }
      // inserto el elemento e incremento la longitud del array
      arr[pos]=v;
      len++;
  return;
}
```

Insertar un valor respetando el orden del array

La siguiente función inserta el valor `v` en el array `arr`, en la posición que corresponda según el criterio de precedencia de los números enteros. El array debe estar ordenado o vacío.

```
int insertarOrdenado(int arr[], int& len, int v)
{
      int i=0;
      while( i<len && arr[i]<=v ){
       i++;
      }
      // inserto el elemento en la i-esima posicion del array
      insertar(arr,len,v,i); // invoco a la funcion insertar
      // retorno la posicion en donde se inserto el elemento
return i;}
```

Insetar un valor respetando el orden del array, sólo si aún no lo contiene

La siguiente función busca el valor `v` en el array `arr`; si lo encuentra entonces asigna `true` a `enc` y retorna la posición que `v` ocupa dentro de `arr`. De lo contrario asigna `false` a `enc`, inserta a `v` en `arr` respetando el orden de los números enteros y retorna la posición en la que finalmente `v` quedó ubicado.

```
int buscaEInserta(int arr[], int& len, int v, bool& enc)
{
      // busco el valor
      int pos = buscar(arr,len,v);  // determino si lo encontre o no
      enc = pos>=0;      // si no lo encontre entonces lo inserto ordenado
      if( !enc ){
        pos = insertarOrdenado(arr,len,v);
      }     // retorno la posicion en donde se encontro el elemento o en donde se inserto
return pos;}
```