Unidad 08 Estructuras indexadas

8

Contenidos analíticos

Parte III: Estructura de datos estáticos y almacenamiento logico

Definición y declaración de array. Array de tipo de dato simple. Array de tipos de datos derivados. Tamaño lógico y tamaño físico. Estructura unidimensional o multidimensional. Array de caracteres. Concepto de puntero constante, desplazamiento.

Algoritmos y Estructura de datos

1. Algoritmos, que refiere al "…*conjunto de finito de reglas, ordenadas de forma lógica y precisa para la solución de un problema, con utilización o no de un computador*…" y…
2. Estructura de datos. identifican un dominio de valores y operaciones aplicables sobre esos valores.

Tipos de datos

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

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 declarar 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.

Tipos de datos pueden ser.

1. **Estáticos:**
    a. **Simples**:
        i. **Ordinales**:
            1. **Enteros**: int a;
            2. **Lógico** o booleano:
            3. **Carácter**: char c;
        ii. **No ordinales**:
            1. Reales:
    b. **Cadenas**: char s[5]

A	B	c	\0	

c.

i. **Estructuras**:
   1. **Registro**:
   2. **Arreglo**:
   3. **Archivos**:

2. **Dinámicos:**
   a. **Listas enlazadas**:
   b. Pilas:
   c. **Colas**:
   d. Otros ….

Ya vimos en la unidad anterior

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

- Los valores pueden ser de tipo distinto, no necesariamente homogeneos.
- Se define como posiciones contiguas de memoria de tipos no homogéneos. Es, entonces, 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 el operador punto (.)
- El almacenamiento es fijo.

unAlumno

Nombre	dni	Legajo

alumno.nombre
alumno.dni
Declaracion Genérica

```
struct Alumno{
        char Nombre[20+1];
        int dni;
        int Legajo;
}
Alumno unAlumno;
```

En C, su declaración es:

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

12

```
a=3*4;

TipoRegistro r1,r2; r1.N=12;r1.Y=10.5;
```

r1 r1.N = 12; campo a campo o estructura completa r2=r1;

N→int→ 12	Y→double→ 10.5

cout<<r1.N; solamente campo a campo
cin>>r1.N>>r1.Y; solamente campo a campo
r2

N→int→ 12	Y→double→ 10.5

Asignación interna
r1.N = 3*4 →campo a campo
r1.Y = 10.5
r2.N = r1.N
r2 = r1 → estructura completa
asignación externa
entrada
cin>>r1.N → debe ser campo a campo
~~cin>>r1~~ no es posible
salida
cout<<r1.N → debe ser campo a campo
~~cout<<r1~~

Las estructuras pueden ser anidadas. Ejemplo de estructuras anidadas en C

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


struct TipoAlumno {
        int             Legajo;
        char            Nombre[9+1];
        TipoFecha       Fecha
};                      //      declara un tipo Alumno con un campo de tipo Fecha
TipoAlumno Alumno;  //      define un identificador con la estructura declarada.
```

Legajo	Nombre	Fecha		
		D	M	A

TipoFecha Fecha;

D	M	A

Fecha.D

En el caso de la definición precedente, Alumno es un registro (struct para C) con tres miembros (campos) uno de los cuales es un registro (struct) 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 año 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, en principio, 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 antecededе 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. 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 lista o vector.
- El arreglo con 2 o más índices o dimensiones es una tabla o matriz. Un grupo de elementos homogéneo con un orden interno en el que se necesitan 2 o más índices para referenciar a un elemento de la estructura.
- En el caso de C, no hay control interno m para evitar acceder a un índice superior al tamaño físico de la estructura, esta situación si tiene cotrol en C++, mediante la utilización de at para el acceso (se vera mas adelante).

indexada→ índice acceso a cada miembro s[5]
char s[5] = “abc”

A	b	c	\0	

s[0]

S es una cadena de caracteres en este caso se inicializa con los caracteres del literal cadena agregando el carácter centinel \0 para indicar el final de la misma
Al contener el carácter centinela esta estructura que es un vector de caracteres se considera una cadena y la asignación externa puede ser por estructura completa, al ser conceptualmente un vector de caracteres también es posible mostrarlo elemento por elemento como ocurre con los vectores

1 indice (1 dim) → vector → int v[5]

V[0]	V[1]	V[2]		

V es un vector de enteros y la asignación interna y externa debe hacerse elemento por elemento. Salvo en el caso particular de la definición en el momento de la declaración, cosa que veremos mas adelante

2º mas (+1 dim)→ matriz → int M[3][4]

M[0][0]	M[0][1]	M[0][2]	M[0][3]
M[1][0]	M[1][1]	M[1][2]	M[1][3]
M[2][0]	M[2][1]	M[2][2]	M[2][3]

M es una matriz de dos dimensiones o un vector de vectores (vector de N filas, en cada una de las cuales contiene un vector de m columnas la declaración presentada facilita esta particularidad, El lo referido a la asignación sigue las reglas definidas anteriormente

Estas estructuras permiten Acceso directo y su tamaño físico es fijo tiempo de ejecución
int vector[10];

V[0]	V[1]								

Carga o asignacion
define TOPE 5

int main(){

int v[TOPE]

¿?	¿?	¿?	¿?	¿?

La declaracion o garantiza, en general un valor de inicialización, por lo que una declaración solo hace reserva de memoria en posiciones contiguas sin precisar el valor asignado

int v[TOPE]={1,2,3,4,5};

1	2	3	4	5

Si en cabio en la declaración se definen el conjunto de valores iniciales queda inicializado con ellos. Los elementos del conjunto son una lista de elementos (recuerde que la , separa listas) estos elementos, por ser un conjunto se encuentran entre { y }. En este caso al asignarse valores a todas las posiciones las mismas adoptaran, el el orden de la lista los mismos desde el comienzo de la estructura

int v[TOPE] = {1,2,3};

1	2	3	0	0

Si el subconjunto de asignación no refiere a la totalidad de elementos del array la lista comenzara a asignas desde la primera posición en posiciones contiguas completando las posiciones faltantes con ceros

int v[TOPE] = {0}:

0	0	0	0	0

Llevando a un extremo lo expresado anteriormente si el subconjunto de asignación es uno solo pondrá ese valor en la primera posición y el resto lo completara con ceros. Si ese valor también es cero, como en el ejemplo, toda la estructura, independientemente del tamaño físico quedara inicializada en cero

La asignación interna también puede ser elemento por elemento utilizando una composición iterativa

```
for(i=0; i<TOPE; i++)
        v[i]=0;
```

Lo mismo es valido para la asignación externa de entrada

```
for(i=0; i<TOPE; i++)
        cin>>v[i];
```

también para la asignación externa de salida

```
for(i=0; i<TOPE; i++)
        cout<<v[i];
```

Carga directa: Como la estructura es indexada se puede acceder a una posicion en forma directa.
v[j] = expresion; j debe estar en el rango [0..TOPE-1]

int m[2][3]={ {1,4,5 },{4,8,32 } }; en forma similar se puede definir los valores de una matriz en la declaracion

1	4	5
4	8	32

La asignación interna requiere tantas composiciones iterativas anidadas como dimensiones tenga la estructura

```
for (i=0;i<2;i++)
        for(j=0;j<3;j++)
                cin>>m[i][j];
```

Combinando estructuras

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

La declaracion anterior refiere a una estructura tipo fecha

```
struct TipoAlumno {
        int             Legajo;
        char            Nombre[40];
        TipoFecha       Fecha
};                      //      declara un tipo Alumno con un campo de tipo Fecha
```

Esta declaración tiene una combinadion di tipos de datos, un entero, un vector de caracteres y una struct de tipo fecha

TipoAlumno Vector[4];

El array aquí declarado tiene cuatro registros cada uno de los cuales tiene un entero, un vector de caracteres y un regidtro que tiene, a su vez tres campos de tipo entero

legajo	nombre	Fecha		
		D	m	A

Vector
Vector[0]
Vecto[0].legajo→accede al campo legajo del registro que esta en la posición cero del vector
Vector[0].nombre→ accede al capo nombre del mismo registro
Vector[0].nombre[2]→accede al tercer carácter del campo nombre del mismo registro
Vector[0].fecha→accede al campo fecha del mismo registro que a su vez es un registro de tipo fecha
Vector[0].fecha.d→accede al campo dia del registro fecha que es un campo del registro de la pos 0 de V
Vector[0].fecha.m→similar a lo anterior para el mes
Vector[0].fecha.a.-->similar a lo anterior para el año

Declaración

Genérica

En C:

TipoDeDato Identificador[Tamaño];

int VectorEnteros[10]// declara un vector de 10 enteros

TipoAlumno VectorAlum[10] // declara un vector de 10 registros.

TipoAlumno MatrizAlumno[10][5] //declara matriz o tabla de 10 filas y y columnas

Caracteristicas de los array

Almacenamiento	Electronico
Procesamiento	Rapido
Tamaño T.E.	Fijo en tiempo de ejecución
Persistencia	Sin persistencia
Proposito	Procesar con rapidez conjunto de datos homogéneos Uso como estructura auxiliar para optimizar proceso
Declaracion	Tipo de dato Nombre[índice][índice]----
Acceso	Nonbre[índice][índice]….
Carga	En la declaración Secuencial Ordenada Sin repetición de clave Directa → según cantidad de índices definidos se selecciona vector o matriz
Busquedas	Directa Secuencial Binaria
Ordenamiento	PUP Metodos
Recorridos	Totales Parciales Con Corte de control Apareando

```
struct TR{
        int     a;
        float   b;
};
```

TR V[5];

V vector de tipo TR → vector de registros

V[0] → registro pos 0 vector 5 registros

V[0].a → el int campo a del registro pos 0 del vector

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

TipoAlumno VectorAlum[10];
```

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
VectorAlum[0].Fecha	Registro	El campo fecha de ese registro, que es un registro
VectorAlum[0].Fecha.D	Entero	El campo dia del registro anterior

Vector de 5 componentes
int V[5]

V[0]
V[1]
V[2]
V[3]
V[4]

Matriz de 5 filas y tres columnas
int M[5] [3]

M[0][0]	M[0][1]	M[0][2]
M[1][0]	M[1][1]	M[1][2]
M[2][0]	M[2][1]	M[2][2]
M[3][0]	M[3][1]	M[3][2]
M[4][0]	M[4][1]	M[4][2]

Busqueda lineal en un vector

int tope = 8;
int N=0;

Ve[0]	6
1	3
2	9
3	14
4	1
5	11
6	25
7	96

Buscado	i	v[i]	buscado != v[i]	
cuando el buscado esta índice [0..N-1]				
14		0	6	v
		1	3	v
		2	9	v
		3	14	f
cuando no está el buscado → índice toma el valor de N				
19		0	6	v
		1	3	v
		2	9	v
		3	14	v
		4	1	v
		5	11	v
		6	25	v
		7	96	v
		8[N]	???	

```
#define TOPE 8

int busquedaSecuencial(int  buscado, int v[], int N)
{
        int i = 0;
        while(i<N&&v[i]!=buscado)
                i++;
        if(i<N) return i;
        return -1;
};
```

```
void cargarAlFinal(int b, int v[].int &N)
{ //control de tope antes de invocar a la función
    v[N] = b;
    N++;
    return;
  }
```

12	35	8	0	0	0	0	0

```
int main(){
        int Vector[TOPE] = {12, 35, 8};       //TOPE tamaño fisico del vector
        int N = 3;                            // N tamaño lógico
        int aBuscar, esta;
        cin>>aBuscar;
        esta = busquedaSecuencial(aBuscar, Vector, N)//
        if(esta == -1){Vector[N]=aBuscar; N++ };

return 0;
}
```

Interrogantes
Como controlar para no superar el tope?
Que hacer si el buscado no esta?
Como se pasa un vector como argumento?
Como se declara como parámetro? Se protege de modificación? Y en una matriz?

Busqueda directa PUP, clave posicional
saber nombre del equipo Numero 101 → V[101-101].nombre →V[0].nombre
saber nombre del equipo Numero 104 → V[104-101].nombre →V[3].nombre

Pos	Numero	Nombre
0	101	Huracán
1	102	Boca
2	103	River
3	104	Racing
4	105	sl
5	106	Chaca
6	107	Platense
7	108	Indep

V[clave-valor - pos inicial]

Carga

1. **Declaración**
   a. int v[8] = {1,4,2,6,8,9,2,0};→ inicializa todas las posiciones con la lista
   b. int v[8] = {2,6,8};→ inicializa todas las posiciones, tres con la lista el resto con cero.
   c. int v[8] = {0} → inicializa todas las posiciones con cero
   d. equipo equipos[8] = ???? {{101 , "huracán"} , { , } , { , } }

2. **Secuencial → elemento por elemento**
   a. **asignación interna** → for(i=0;i<8;i++) V[i] = valor;
   b. **asignación externa →** for(i=0;i<8;i++) cin>> V[i];
3. **Directa** → Clave numérica, PUP→ pos V[Clave-valor posición inicial]
4. **ordenada o no sin repetición de clave**

búsqueda

1. **Directa** Clave numérica, PUP→ Ef(1)→ V[Clave – valor 1ra posición]
2. **Secuencial** {} valores sin orden → recorrer desde la primera posición hasta que lo encuentre o hasta que se termine el vector→ Ef(N)

```
int búsquedasecuencial(int v[], int buscado, int n){
vector, valor buscado, tope del vector
i = 0; → índice para recorrer desde el principio
while(i<n&&buscado!= v[i])
        i++; → para acceder a la sgte posicion

if(i<n) return i;
else return -1;//arbitrario indica dato buscado no esta
retorna la posicion donde el buscado esta o -1 si el buscado no esta
}
```

3. **Binaria → deben estar ordenados**→ Ef(logarítmica)

512 256 128 64 32 16 8 4 2 1

2^{10}

Condiciones de los datos para la búsqueda		
Directa	PUP	Ef(1)
Binaria		ordenados Ef(log)
Secuencial		datos están sin orden Ef (N)

Busqueda Binaria

0	14
1	25
2	29
3	32
4	45
5	54
6	68
7	92

N 8

Pos 1er 0

pos ultimo 7→N-1

buscado	pos 1ro	pos ultimo	mitad(p+u)/2	V[mitad]
32	0	7	3	32
54	0	7	3	32
	4	7	5	54
25	0	7	3	32
	0	2	1	25
34	0	7	3	32
	4	7	5	54
	4	4	4	45
	4	3	¿??	¿???

cuando el valor de la posición del primero supera el valor de la posicion del ultimo es indicador que el dato no está.

```
int busquedaBinaria(int v[], int buscado, int N, int& primero){
//retorna la posición donde está el dato o menos uno si no lo encuentra
        int ultimo = N-1;
        int medio;
        primero = 0;
        while (primero <= ultimo){
                medio = (primero + ultimo)/2;
                if(v[medio] == buscado) return medio; //la posición donde lo encontró
                if(buscado > v[medio]) primero = medio + 1;
                else ultimo = medio – 1;
        };
        return -1;// si sale del while es que el primero es mayor que ultimo ese es el indicio que el dato
buscado no esta por eso retorna -1
}
```

Cargar sin repetir la carga

Elementos sin orden

If (busquedaSecuencial(buscado, vector, N) == -1){vector[N]=buscado; N++)

……..

Elementos ordenados conservando el orden

Pos	Valor
0	8
1	25
2	32
3	46
4	85
5	92

N = 6

Pos	Valor
0	8
1	25
2	32
3	40
4	46
5	85
6	92

N = 7

Buscado = 40

Primero = 3

v[6] = v[5] → 5 represenrta N-1 6 representa N

v[5] = v[4]

v[4] = v[3] →3 represnenta primero 4 es el primer mayor a primero

el índice del valor a desplazar varia entre 5, que es el valor de N-1 y 3 que es primero

```
if(busquedaBinaria(vector, buscado, N, primero) == -1&&N<Tope)
{               //i toma 5, 4, 3                    i toma 6, 5, 4
        for(i=N-1; i>=primero; i--)                 for(i=N;i>primero, i--)
                v[i+1] = v[i]; // el desplazamiento      v[i] = v[i-1];

        v[primero] = buscado;     //coloca el nuevo
        N++;                      //incrementa el tamaño logico
}
```

Ordenamiento → mas simple → menos eficiente → burbuja

0	5									
1	2								2	
2	3						3		3	
3	4				4		4		4	
4	1		5		5		5		5	
Paso	→ i	1		2		3		4		
Comp.	→j	4		3		2		1		
P+C	→N	5		5		5		5		

i	j	v[j-1]	v[j]	variación de j	
1	1	v[0]	v[1]	[j-1][j]	→ índices que compara en cada paso
	2	v[1]	v[2]		
	3	v[2]	v[3]		
	4	v[3]	v[4]	1..N-i	→1..5-1 →1..4
2	1	v[0]	v[1]		
	2	v[1]	v[2]		
	3	v[2]	v[3]	1..N-i	→1..5-2 →1..3
3	1	v[0]	v[1]		
	2	v[1]	v[2]	1..N-i	→1..5-3 →1..2
4	1	v[j]	v[j-1]	1..N-i	→1..5-4 →1..1

```
void ordenarVector(int v[], int N) {// v el vector a ordenar N cantidad de componentes
        int i,j, aux;
        for(i = 1; i < N; i++){// pasos → 1..N-1
                for( j = 1; j <= N – i;j++){//comparaciones en cada paso → 1..N-i
                        if(v[j-1] > v[j]{
                                aux = v[j];
                                v[j] = v[j -1];
                                v[j – 1] = aux;
                        }
                } // fin ciclo interno
        }// fin ciclo externo
return;
} // fin de la funcion
```

Recorridos

1. Completo
   fo(i = 0; i<N, i++)
   cout << v[i];

2. Con corte de control
   Precondición → Al menos un campo que se repite, agrupado por ese campo, que conforman un subconjunto
   Pos condición → Información de cada subconjunto y/o del conjunto general
3. Apareo
   Precondición → Al menos dos estructuras con al menos un campo en común y ordenados.
   Pos condición → manejo conjunto intercalando y manteniendo el orden

Ordenar por más de un campo → solo cambia el criterio de selección

	c1	**c2**
0	20	11
1	8	2
2	15	18
3	20	6
4	3	11
5	3	3
6	20	4
7	15	9

0	3	3
1	3	11
2	8	2
3	15	9
4	15	18
5	20	4
6	20	6
7	20	11

Cambiar la lógica Vs cambiar la algoritmia

```
void ordenarVector(int v[], int N) {// v el vector a ordenar N cantidad de componentes
        int i,j, aux;
        for(i = 1; i < N; i++){
                for( j = 1; j <= N – i;j++){
                        if(v[j-1].c1 > v[j].c1|| (v[j-1].c1== v[j].c1&& v[j-1].c2> v[j].c2)){
                                aux = v[j];
                                v[j] = v[j -1];
                                v[j – 1] = aux;
                        }
                } // fin ciclo interno
        }// fin ciclo externo
return;
} // fin de la función
```

```
struct prueba{
        int c1;
        int c2;
};
prueba v[8];
```

carga ordenada sin repetir la clave

v

0	20
1	25
2	31
3	34
4	54
5	82
6	
7	

pos=busquedaBinaria(V,34,5,primero)
pos=-1; primero=3

Cantidad física → declaración → 8
cantidad lógica → actual → 5

```
N = 0;
while(N<8){
        cin >> valor;
        pos = busquedaBinaria(V, valor, N, primero);
        if(pos == -1) {cargarvalor(V,valor,N,primero); N++};
};
```

0	20	20
1	25	25
2	31	31
3	54	34
4	82	54
5		82
6		
7		

valor 34 N=5 primero=3

desplazamiento
V[5]=V[4]
v[4]=V[3]

i
4 → V[5]=V[4]
3 → v[4]=V[3]

```
void cargavalor(int V[].int valor, int N, int &primero){
for(i = N-1 ; i >= primero ;i--)
        V[i+1]=v[i];
v[primero]= valor;
return;
}
```

Preguntas
Si se dispone de un vector y la clave de identificación es posicional:

1. Cual es la búsqueda adecuada?
2. Cual es la eficiencia de la búsqueda?
3. Como se accede a la posición de búsqueda?

Si la clave fuera numérica y los datos están ordenados cuales serian sus respuestas?

Si la clave fuera una cadena de caracteres?

En caso de estar sin orden, como respondería? Existe diferencia si la clave es numérica o cadena?- Justifique-

En caso de estar desordenado, puede plantear alternativas a la búsqueda;

Que estrategias conoce para buscar u ordenal par mas de un campo?

Ejercicios con vectores y matrices

1. Ingresar un valor N (< 25). Generar un arreglo de N componentes en el cual las mismas contengan los primeros números naturales pares e imprimirlo.
2. Ingresar un valor entero N (< 30) y a continuación un conjunto de N elementos. Si el último elemento del conjunto tiene un valor menor que 10 imprimir los negativos y en caso contrario los demás.
3. Ingresar un valor entero N (< 20). A continuación ingresar un conjunto VEC de N componentes. A partir de este conjunto generar otro FACT en el que cada elemento sea el factorial del elemento homólogo de VEC. Finalmente imprimir ambos vectores a razón de un valor de cada uno por renglón
4. Ingresar un valor entero N (< 25). A continuación ingresar un conjunto VEC de N componentes. Si la suma de las componentes resulta mayor que cero imprimir las de índice impar, sino los otros elementos.
5. Ingresar un valor entero N (< 30). A continuación ingresar un conjunto UNO y luego otro conjunto DOS, ambos de N componentes. Generar e imprimir otro conjunto TRES intercalando los valores de posición impar de DOS y los valores de posición par de UNO.
6. Ingresar un valor entero N (< 40). A continuación ingresar un conjunto VALOR de N elementos. Determinar e imprimir el valor máximo y la posición del mismo dentro del conjunto. Si el máximo no es único, imprimir todas las posiciones en que se encuentra.
7. Ingresar un valor entero N (< 15). A continuación ingresar un conjunto DATO de N elementos. Generar otro conjunto de dos componentes MEJORDATO donde el primer elemento sea el mayor valor de DATO y el segundo el siguiente mayor (puede ser el mismo si está repetido). Imprimir el conjunto MEJORDATO con identificación.
8. Ingresar un valor entero N (< 25). A continuación ingresar un conjunto GG de N elementos. Imprimir el arreglo en orden inverso generando tres estrategias para imprimir los elementos a razón de: a) Uno por línea, b) Diez por línea, c) Cinco por línea con identificación
9. Ingresar un valor entero N (< 40). A continuación ingresar un conjunto A y luego otro conjunto B ambos de N elementos. Generar un arreglo C donde cada elemento se forme de la siguiente forma: C[1] = A[1]+B[N] C[2] = A[2]+B[N-1]
10. Ingresar dos valores enteros M (< 10) y N (< 15). A continuación ingresar un conjunto A de M elementos y luego otro B de N elementos. Generar e imprimir:
    a) Un conjunto C resultante de la anexión de A y B.
    b) Un conjunto D resultante de la anexión de los elementos distintos de cero de A y B.

Plantillas para vectores

agregar → Agrega un nodo al final del vector, el control de no superar el valor físico debe hacerse antes de invocar a la función, actualiza el tamaño lógico

```
void agregar(int arr[], int& n,
int x){
   arr[n]=x;
   n++;

   return;
}

agregar(V, 10, 123)
```

```
template <typename T> void
agregar(T arr[], int& n, T v){
   arr[n]=v;
   n++;

   return;
}
agregar<tipoReg>(V, 10, reg)
agregar<int> (V, 10, 123)
```

mostrar → muestra el contenido de un vector, en el caso de plantillas de agrega el concepto de puntero a función.

```
void mostrar(int v[], int n){

   for(int i=0; i<n; i++){
      cout <<(v[i]<<endln;
   }

   return;
}
```

```
template <typename T> void
mostrar(T v[], int n),void
(*ver(T))){
   for(int i=0; i<n; i++){
      ver(v[i];
   }

   return;
}
void ver_reg(TipoReg r){
 cout << r.c1…….<<endl;
 return;
}
mostrar<tipoReg>(v, 10,ver_reg);
```

ordenar → ordena los elementos de un vector, creciente en enteros, según el criterio de ordenamiento y miembro con plantilla y puntero a funcion

```
void ordenar(int v[], int n,){

   int aux;
   int i,j,ord=0;
   for(i=0,i<n-1&&ord==0,i++){
      ord=1;
      for(int i=0; i<n-1; i++){

      if(v[j]>v[j+1]){
            aux = v[j];
            v[j] = v[j+1];
            v[j+1] = aux;
            ord = 1;
         }
      }
   }

   return;
}



Ordenar(v,n);
```

```
template <typename T> void
ordenar(T v[], int n, int
(*criterio)(T,T)){
   T aux;
   int i,j,ord=0;
   for(i=0,i<n-1&&ord==0,i++){
      ord=1;
      for(int i=0; i<n-1; i++){

if(criterio(v[j],v[i+1])>0){
            aux = v[j];
            v[j] = v[j+1];
            v[j+1] = aux;
            ord = 1;
         }
      }
   }

   return;
}
int enteroCrec(int e1,inte2){
    return e1>e2?1:0;
 }
ordenar<int>(v, n, enterocrec);
```

buscar→ busca secuencialmente un elemento en un vector retornando el índice donde lo encuentra, si el dato buscado esta o el valor -1 silo buscado no esta en el vector

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



buscar(v,n, 123);
```

```
template <typename T, typename K>
int buscar(T v[], int n, K x, int
(*criterio)(T,K)){
   int i=0;
   while( i<n &&
criterio(v[i],x)>0 ){
      i++;
   }
   return i<n?i:-1;
}

int Legajo(tr r, int leg){
return r.leg!=leg?1:0;

buscar<tr,int>(v,n, 123, legajo);
```

busquedaBinaria→ busca dicotomicamente un elemento en un vector retornando el índice donde lo encuentra, si el dato buscado esta o el valor -1 silo buscado no esta en el vector

```
int busquedaBinaria(int a[], int
n, int v){
   int i=0;
   int j=n-1;
   int k;

  while( i<=j ){
      k=(i+j)/2;
      if( v > a[k] ){
         i=k+1;
      }
      else
      {
         if(v<a[k]){
            j=k-1;
         }
         else{
            return k;
         }
      }
   }
   return -1;
}
```

```
template<typename T, typename K>
int busquedaBinaria(T v[], int n,
K v, int (*criterio)(T, K)){
   int i=0;
   int j=n-1;
   int k;

  while( i<=j ){
      k=(i+j)/2;
      if( criterio(v,a[k])>0 ){
         i=k+1;
      }
      else
      {
         if(criterio(v,a[k])<0){
            j=k-1;
         }
         else{
            return k;
         }
      }
   }
   return -1;
}
```