Capítulo I

Introducción a las Estructuras De Datos

El pato y la Serpiente

A orillas de un estanque, diciendo estaba un pato:

-¿A qué animal dio el cielo los dones que me ha dado? Soy de agua, tierra y aire; cuando de andar me canso, si se me antoja, vuelo; si se me antoja, nado.

Una serpiente astuta, que le estaba escuchando, le llamó con un silbido y le dijo:

-No hay que ser tan arrogante señor pato, pues ni anda como el ciervo, ni vuela como el halcón, ni nada como el pez; y así tenga sabido que lo importante y raro no es entender de todo, sino ser diestro en algo.

de Tomás de Iriarte

1.1. Introducción

Para procesar información en un computador es necesario hacer una abstracción de los datos que tomamos del mundo real, abstracción en el sentido de que se ignoran algunas propiedades de los objetos reales, es decir, se simplifican. Se hace una selección de los datos más representativos de la realidad a partir de los cuales pueda trabajar el computador para obtener unos resultados.

Cualquier lenguaje suministra una serie de tipos de datos simples, como son los números enteros, caracteres, números reales. En realidad suministra un subconjunto de éstos, pues la memoria del ordenador es finita. Los punteros (si los tiene) son también un tipo de datos. El tamaño de todos los tipos de datos depende de la máquina y del compilador sobre los que se trabaja.

1.2. Abstracción

Una abstracción es un proceso mental donde se extraen rasgos esenciales de algo para representarlos por medio de un lenguaje gráfico o escrito.

1.3. Abstracción de datos

Técnica o metodología que permite diseñar estructuras de datos, permite representar bajo ciertos lineamientos las características esenciales de las estructuras de datos.

1.4. Definición de Estructuras de Datos

Una estructura de datos es cualquier colección de datos organizados de tal forma que tengan asociados un conjunto de operaciones para poder manipularlos.

Ejemplo: Estructura de Datos: ⇨ Arreglo

Sus operaciones: ⇨
- Comparación
- Asignación
- Escritura
- Lectura

Otros ejemplos de estructuras de datos:

- ✓ Lista de elementos
- ✓ Archivos (conjunto de registros)
- ✓ Vector (conjunto de elementos)
- ✓ Registros

OPERACIONES

- Ordenar
- Buscar
- Eliminar
- Modificar
- Insertar
- desplegar

1.5. T.D.A. (Tipo de Dato Abstracto)

Al diseñar una estructura de datos con la técnica de abstracción pasa a ser un TDA, que:

- ✓ Puede implementarse en cualquier lenguaje
- ✓ Puede aplicarse en cualquier concepto

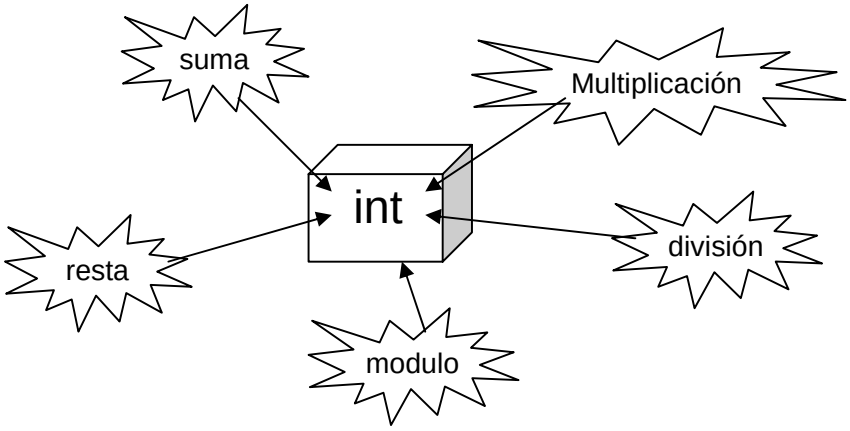
1.6. Especificación lógica de un T.D.A. (Tipo de Dato Abstracto)

1. Elementos que conforman la estructura
2. Tipo de organización en que se guardarán los elementos :
   - Lineal
   - Jerárquica
   - Red
   - sin relación

3. Dominio de la estructura
4. Descripción de las operaciones de la estructura:
    - Nombre de la operación
    - Descripción breve,
    - Datos de entrada
    - Datos de salida,
    - Precondición
    - Poscondición

Ejemplo:

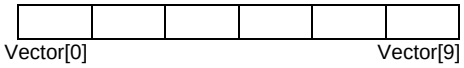
TDA: Matriz, int x



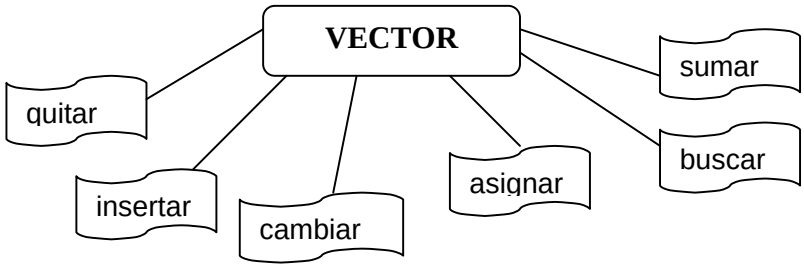
Ejemplo:

TDA: VECTOR: $<x_1, x_2, x_3, \ldots, x_n>$

int vector[10] ; ← Declaración de un vector



A continuación vemos en la siguiente figura el TDA vector con sus diferentes operaciones



1.7. Niveles de datos

1. ***Nivel lógico o abstracto*:** Se define abstractamente la estructura de datos y las operaciones relacionadas con ella. Independientemente del lenguaje en el que se implementará.
2. ***Nivel físico o de implementación***: En que lenguaje, qué tipos de datos ya definidos servirán y se implementa como un módulo cada una de las operaciones del TDA.
3. ***Nivel de aplicación o de uso*:** Llamar a las operaciones de la estructura, siguiendo las reglas especificadas.

1.8. Encapsulamiento de datos

El usuario no tiene que usar información solamente trata con los datos de su descripción lógica.

1.9. Estructuras Fundamentales

Los datos a procesar por una computadora se clasifican en:

- ✓ Simples
- ✓ Estructurados

Los ***datos simples*** ocupan sólo una casilla de memoria, por tanto una variable simple hace referencia a un único valor a la vez.

Los ***datos Estructurados*** se caracterizan por el hecho de que con un nombre (identificador de variable estructurada) se hace referencia a un grupo de casillas de memoria. Tiene varios componentes.

Cabe hacer notar que en el presente texto, se tomará como herramienta para los programas y representar las diferentes estructuras de datos el Lenguaje C++.

Ejemplos:

Dato Simple

Declaramos una variable A de tipo entero y asignamos el valor 25.

A ← Identificador

```
int  A;
A = 25;
```

25

Dato Estructurado

Declaramos un dato compuesto o estructurado A que tendrá 5 elementos de tipo entero.

A ← Identificador

```
Int A[5] ;
A = {20,30,40,50,60};
```

20	30	40	50	60

1.10. Arreglos

Con frecuencia se presentan problemas cuya solución no resulta fácil de implementar (a veces es imposible) si se utilizan datos simples.

A continuación se presentará un problema y dos posibles soluciones del mismo utilizando tipos simples de datos. El objetivo de este ejemplo es ilustrar lo complejo que resulta un algoritmo de solución para ciertos problemas, sin usar tipos estructurados de datos.

Problema. *Se tienen calificaciones de un grupo de 50 alumnos. Se necesita saber cuántos alumnos tienen una calificación superior al promedio del grupo.*

¿Cómo resolver este problema?

Primera Solución: DOBLE LECTURA
Vea el programa N°1

Segunda Solución: MUCHAS VARIABLES

Esta solución, resuelve el problema planteado utilizando múltiples variables.
Vea el programa N°2

Se observa que ninguna de las dos soluciones resulta práctica ni eficiente. Es necesario un nuevo tipo de datos que permita tratar estos problemas de una manera más adecuada. Los tipos de datos estructurados que ayudan a resolver problemas como éste son los arreglos.

Tercera Solución: UTILIZAR UN ARREGLO

Esta solución, resuelve el problema planteado utilizando un arreglo.

Vea el programa N°3

```
//PROGRAMA N°1
// Este programa resuelve el problema planteado por medio de una doble lectura

#include <iostream.h>
#include <conio.h>

void main (void)
{
        int I, CONT;
        float AC, PROM, C;

        clrscr();
        AC = 0;  I = 1;
        while ( I < = 50)
        {       cout << " Introduzca la calificación del alumno " << I;
                cin >> C;
                AC = AC + C;
                I ++;
        }
        PROM = AC / 50;
//Como se necesita decir cuántos alumnos obtuvieron una calificación superior al promedio, se deberá //volver a
leer las 50 calificaciones para poder comparar cada una de ellas con el promedio

CONT = 0;     I = 1;
while ( I < = 50 )
{
        cout << " Introduzca la calificación del alumno " << I;
        cin >> C;
        if ( C > PROM )    CONT = CONT + 1;          I + +;
}
cout<<  "el numero de alumnos cuya calificación es mayor al promedio son " << CONT;
getch ();    }

 }
```

```
//PROGRAMA N°2  SEGUNDA SOLUCION

#include <iostream.h>
#include <conio.h>
void main (void)
{
        int CONT, C1,C2, C3, …, C50;
        float PROM, AC, C;

        clrscr();
        cout << " introduzca la calificación del alumno 1 "; cin >> C1;
        cout << " introduzca la calificación del alumno 2 "; cin >> C2;
        .
        .
        cout << " introduzca la calificación del alumno 50 "; cin >> C50;

        // las calificaciones correspondientes a los 50 alumnos
        AC = C1 + C2 + … + C50;
        PROM = AC / 50;                    CONT = 0;
        If ( C1 > PROM )  CONT = CONT + 1;
        If ( C2 > PROM )  CONT = CONT + 1;
        .
        .
        If ( C50 > PROM )  CONT = CONT + 1;
        cout<<  "el numero de alumnos cuya calificación es mayor al promedio son " << CONT;
        getch();
}
}
```

1.10.1. Definición De Arreglo

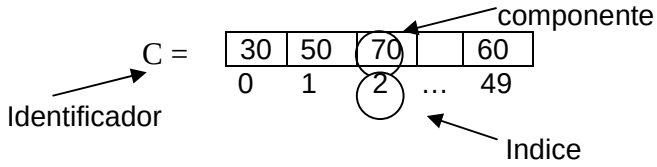
Un arreglo se define como una colección finita, homogénea y ordenada de elementos.

Finita: Todo arreglo tiene un límite; es decir, debe determinarse cuál será el número máximo de elementos que podrán formar parte del arreglo.

Homogénea: Todos los elementos del arreglo son del mismo tipo (todos enteros, todos boléanos, etc., pero nunca una combinación de distintos tipos)

Gráficamente podemos representar un arreglo como sigue:

Ejemplo: Sea un arreglo C que almacena las 50 calificaciones



1.10.2. Declaración de vectores

Para declarar un vector en el lenguaje C, tenemos la siguiente sintaxis:

Tipo_dato nombre_vector[tamaño];

Ejemplo: Un vector **a** de 10 enteros de tipo int se declara:

```
int a[10];
```

El vector a comprende los elementos a[0], a[1], a[2], . . . , a[9], todos de tipo **int .**

En una misma línea se puede declarar más de un vector, siempre que todos compartan el mismo tipo de datos para sus componentes.

Ejemplo. En esta línea se declaran dos vectores de **double**, uno con 20 componentes y otro con 100:

```
double a[20], b[100];
```

Preferiblemente se usa constantes para declarar el tamaño o dimensión de un vector:

```
#define TAM 80
...
int vector[TAM];
```

1.10.3. Inicialización de los vectores

Los arrays pueden ser inicializados en la declaración.

Ejemplos:

```
float R[10] = {2, 32, 4.6, 2, 1, 0.5, 3, 8, 0, 12};
float S[] = {2, 32, 4.6, 2, 1, 0.5, 3, 8, 0, 12};
int N[] = {1, 2, 3, 6};
int M[][3] = { 213, 32, 32, 32, 43, 32, 3, 43, 21};
char Mensaje[] = "Error de lectura";
```

En estos casos no es obligatorio especificar el tamaño para la primera dimensión, como ocurre en los ejemplos de las líneas 2, 3, 4 y 5. En estos casos la dimensión que queda indefinida se calcula a partir del número de elementos en la lista de valores iniciales.

En el caso 2, el número de elementos es 10, ya que hay diez valores en la lista.

En el caso 3, será 4.

En el caso 4, será 3, ya que hay 9 valores, y la segunda dimensión es 3: 9/3=3.

Y en el caso 5, el número de elementos es 17, 16 caracteres más el cero de fin de cadena

Una vez creado un vector, sus elementos presentan valores arbitrarios. Es un error suponer que los valores del vector son nulos tras su creación.

Podemos inicializar todos los valores de un vector a cero con un bucle **for:**

```
#include <stdio.h>
#define TAM 10

void main(void )
{
    int i, a[TAM];
    for (i = 0; i < TAM; i++)
            a[i] = 0;
}
```

Los vectores pueden ser inicializados también, al momento de su declaración:

```
int b[TAM] = {1, 2, 3, 4, 5};
```

```
//PROGRAMA N°3 SOLUCION AL PROBLEMA DE LAS CALIFICACIONES DE 5O
//ALUMNOS
#include <iostream.h>
#include <conio.h>
void main (void)
{
       int  C[51];  //declara un vector C para 50 elementos
       int I;
       float AC, PROM;
       for ( I =1 ; I < = 50 ; I++)
       {
          cout<< " Introduzca la calificación del alumno " << I ;
          cin >> C [ I ];
          AC = AC + C [I];
       }
       PROM = AC / 50;
       for ( I =1 ; I < = 50 ; I++)
              if ( C [ I ] > PROM ) CONT ++;
       cout<<  "el numero de alumnos cuya calificación es mayor al
       promedio son " << CONT;
       getch();
}
```

Ejercicio:

Queremos efectuar estadísticas con una serie de valores (las edades de 15 personas). En una primera versión, solicitaremos las edades de todas las personas y, a continuación, calcularemos y mostraremos por pantalla la edad media, la desviación estándar, la moda y la mediana. Las fórmulas para la media y la desviación estándar son:

$$\bar{x} = \frac{\sum_{i=1}^{15} x_i}{15} \qquad \delta = \sqrt{\frac{\sum_{i=1}^{15} (x_i - \bar{x})^2}{15}}$$

donde Xi es la edad del individuo número i. La moda es la edad que más veces aparece (si dos o más edades aparecen muchas veces con la máxima frecuencia, asumiremos que una cualquiera de ellas es la moda). La mediana es la edad tal que el 50% de las edades son inferiores o iguales a ella y el restante 50% son mayores o iguales.

Empezamos por la declaración del vector que albergará las 15 edades y por leer los datos:

```
#include <conio.h>
#include <iosteam.h>
#include <math.h>

#define PERSONAS 6

void IngresarDatos(int edad[PERSONAS])
{
 //Lectura de edades
 for (i=0; i<PERSONAS; i++)
 {
   cout<<"Por favor, introduce la edad de la persona nïumero"<< i;
   cin>>edad[i];
  }  //fin del for
}  // fin de IngresarDatos

void main(void )
{
 int edad[PERSONAS], i;
 IngresarDatos(edad);
 getch();
}
```

Pasamos ahora a calcular la edad media y la desviación estándar:

```
#include <conio.h>
#include <iostream.h>
#include <math.h>
#define PERSONAS 6

void IngresarDatos(int edad[PERSONAS])
{
 //Lectura de edades
 for (int i=0; i<PERSONAS; i++) {
   cout<<"Por favor, introduce la edad de la persona numero"<< i;
   cin>>edad[i];
  }
}

// Calculo de la media
double Media(int edad[PERSONAS])
{
 double sumaedad = 0, media;
 for (int i=1; i<=PERSONAS; i++)
    sumaedad = sumaedad +edad[i];
 media =(double)  sumaedad / PERSONAS;
 return media;
}

// Calculo de la desviación estándar

double Desviacion(int edad[PERSONAS],double media)
{
  double desviacion, sumadesviacion = 0.0;
  for (int i=0; i<PERSONAS; i++)
  sumadesviacion = sumadesviacion+(edad[i]-media)*(edad[i] - media);
  desviacion = sqrt(sumadesviacion / PERSONAS );
  return desviacion;
}
```

```
void main(void )
{
 int edad[PERSONAS], i;
 double med, des;
 IngresarDatos(edad);
 med=Media(edad);
 des=Desviacion(edad,med);
 cout<<"La media es "<<med;
 cout<<"La desviación estándar es "<<des;
 getch();
}
```

El cálculo de la moda (la edad más frecuente) resulta más problemática. La solución propuesta consiste en ordenar el vector de edades y contar la longitud de cada secuencia de edades iguales. La edad cuya secuencia sea más larga es la moda, la ordenación del vector se deja como ejercicio al lector, el siguiente segmento de programa se realizó bajo el supuesto que el vector está ordenado:

```
   frecuencia = 0;
   frecuenciamoda = 0;
   moda = -1;
   for (i=0; i<PERSONAS-1; i++) // Búsqueda de la serie de valores idénticos más
larga.
    if (edad[i] == edad[i+1])
      {
        frecuencia++;
        if (frecuencia > frecuenciamoda)
        {
            frecuencia_moda = frecuencia;
            moda = edad[i];
        }
      }
      else
        frecuencia = 0;
```

1.10.4. Cadenas de Caracteres

Los arreglos de caracteres tienen varias características únicas.

Un arreglo de caracteres puede inicializarse de la siguiente forma:

```
char CADENA1[ ] = "rosa";
```

- El tamaño de CADENA1 está determinado por el compilador, y generalmente es de 256 caracteres máximo.

- La cadena "rosa" contiene cuatro caracteres, *más* un caracter especial de terminación denominado *caracter nulo* (' \0 ').

- Los arreglos de caracteres pueden inicializarse también con constantes individuales dentro de una lista.

```
char CADENA1[ ]= {' r ', ' o ', ' s ', ' a ', ' \0 ' };
```

- También podemos colocar una cadena de caracteres directamente dentro de un arreglo de caracteres desde el teclado. Por ejemplo:

```
char CAD2[20];
cin >> CAD2;
```

EJERCICIO 1.
El programa puede parecer a primera vista muy sencillo. En primer lugar vamos a leer y escribir una cadena. La primera solución intuitiva:

```
#include <iostream> // USAMOS: cin, cout

void main()
{
    char s[20]; // Cadena de hasta 19 caracteres
    cin >> s; // Leer la primera palabra
    cout << endl << s // Escribir en nueva línea la cadena
         << endl; // Y pasar a la línea siguiente
}
```

EJERCICIO 1.
Programa para leer caracteres desde teclado hasta que el caracter sea nulo

```
#include <iostream.h>             // Usamos: ios, cin, cout
#include <iomanip.h>              // Usamos: resetiosflags

void Leer_Cadena(char * s)
{
  cin >> resetiosflags(ios::skipws); // Para que no pase los caracteres blancos.
  for (int i= 0; cin >> s[i]; i++)  // Leer caracteres hasta el caracter nulo.
    if (s[i] == '\n') break;
    s[i]= '\0';                   // Pone el caracter de fin de cadena.
}

void main()
{
  char s[100];    // Buffer de capacidad de hasta 99 caracteres y '\0'. Si
                  // metemos mas caracteres hay posibilidad de 'cuelgue'

  cout << endl << "Introduce una cadena (termina con CTRL-Z o INTRO)" << endl;
  Leer_Cadena(s);
  cout << endl << "La cadena es: " << s << endl;
}
```

EJERCICIO 3.

Bien, ahora veamos un ejemplo tradicional, se trata de leer caracteres desde el teclado y contar cuántos hay de cada tipo. Los tipos que deberemos contar serán: consonantes, vocales, dígitos, signos de puntuación, mayúsculas, minúsculas y espacios. Cada carácter puede pertenecer a uno o varios grupos. Los espacios son utilizados frecuentemente para contar palabras.

```
// Cuenta letras
#include <iostream.h>
#include <ctype.h>
int main()
{
   int consonantes = 0;
   int vocales = 0;
   int digitos = 0;
   int mayusculas = 0;
   int minusculas = 0;
   int espacios = 0;
   int puntuacion = 0;
   char c; // caracteres leídos desde el teclado

   cout << "Contaremos caracteres hasta que se pulse ^Z" << endl;
   while((c = getchar()) != EOF)
   {
      if(isdigit(c)) digitos++;
      else if(isspace(c)) espacios++;
      else if(ispunct(c)) puntuacion++;
      else if(isalpha(c))
      {
```

```
if(isupper(c)) mayusculas++; else minusculas++;
         switch(tolower(c)) {
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u':
               vocales++;
               break;
             default:
               consonantes++;
         }
      }
   }
   cout << "Resultados:" << endl;
   cout << "Dígitos:     " << digitos << endl;
   cout << "Espacios:    " << espacios << endl;
   cout << "Puntuación:  " << puntuacion << endl;
   cout << "Alfabéticos: " << mayusculas + minusculas<< endl;
   cout << "Mayúsculas:  " << mayusculas << endl;
   cout << "Minúsculas:  " << minusculas << endl;
   cout << "Vocales:     " << vocales << endl;
   cout << "Consonantes: " << consonantes << endl;
   cout << "Total: " << digitos + espacios + vocales +
      consonantes + puntuacion << endl;
   return 0;
}
```

1.10.5. Arreglos Multidimensionales

Existe en la mayoría de los lenguajes una estructura de arreglos multidimensionales. El número de dimensiones (índices) permitido depende del lenguaje elegido.

1.10.5.1. Matrices

Una matriz es un arreglo de dos dimensiones, y para especificar cualquier elemento, debemos hacer referencia a dos índices (que representan la posición como renglón y columna). Aunque no se justificará aquí, es conveniente mencionar que la representación matricial es puramente conceptual y con el único fin de facilitar al programador el manejo de los elementos, ya que la computadora almacena los datos en una forma totalmente diferente.

Se analizarán primero los **arreglos bidimensionales** (caso especial de los multidimensionales) por ser los más utilizados.

- C++ soporta hasta arreglos con 12 dimensiones. En arreglos de dos dimensiones, el primer elemento representa el *renglón* y el segundo la *columna*.

- Cada elemento de un arreglo bidimensional puede referenciarse de la siguiente manera: arreglo [i] [j].

- Un arreglo multidimensional puede inicializarse desde su declaración. Por ejemplo, un arreglo bidimensional b[2][2] puede declararse e inicializarse así:

```
int  b[2][2] = { {1,2} , {3,4} };
```

- Los valores son agrupados en renglones.

✓ *Declaración de una Matríz*

La sintaxis en el lenguaje C++ es el siguiente:

```
tipo nombre_de_variable [rango1][rango2];
```

donde:

- tipo puede ser cualquier tipo de dato (int, float, char, etc.).
- nombre_de_variable es el nombre del arreglo.
- rango 1 corresponde al número de renglones que conforman el arreglo.
- rango 2 corresponde al número de columnas.

Podemos trabajar con cada uno de los elementos de la matriz:

```
X[3][5]  = 20;
```

✓ *Operaciones con arreglos bidimensionales*

Las operaciones que pueden realizarse con arreglos bidimensionales son las siguientes:

- Lectura/escritura
- Asignación
- Actualización: Inserción
    - Eliminación
    - Modificación
- Ordenación

- Búsqueda

En general los arreglos bidimensionales son una generalización de los unidimensionales, por lo que se realizará un ejemplo con algunas de estas operaciones a continuación.

Ejemplo: La Tabla contiene gastos que registra una ama de casa correspondientes a los 12 meses del año anterior.

Meses/Gastos	Agua	Luz	Telefono	Mercado
Enero	23	57	840	250
Febrero	28	60	560	280
Marzo	34	55	400	275
Abril	24	87	700	340
Mayo	29	80	450	310
Junio	34	65	670	320
Julio	45	67	560	325
Agosto	48	78	570	323
Septiembre	32	54	540	290
Octubre	33	50	250	300
Noviembre	35	70	330	350
Diciembre	38	62	300	430

Es posible interpretar esta tabla de la siguiente manera: dado un mes, se conocen los gasto realizados por la ama de casa; y dado un gasto se conocen los gastos mensuales.

El programa será el siguiente:

```
#iclude<iostream.h>
#include<conio.h>

typedef int MATRIZ[13][5]; //definimos un tipo de dato MATRIZ de números
enteros

//PROCEDIMIENTO PARA REGISTRAR LOS GASTOS EN LA MATRIZ G
//DE DIMENSIONES m filas y g columnas

void Lee_gastos(MATRIZ G, int m, int g)
{
   int i; //contador de meses
   int j; //contador de columnas de gasto

   for ( i= 1; i<=m ; i++)
   for ( j=1; j<=g; j++)
   {
      cout<<"\n Ingrese el gasto del mes  "<< i << "gasto "<< j<<;
      cin>>G[i][j];
   }
}
```

```
// PROCEDIMIENTO PARA IMPRIMIR TODOS LOS GASTOS DE LA
// MATRIZ G

void Imprime_Gastos(MATRIZ G, int m, int g)
{
      for (int i=1; i<=m; i++)
      {
       for(int j=1; j<=n; j++)
            cout<<G[i][j]<<"   ";
       cout<<"\n";
      }
}
//PROCEDIMIENTO PARA LISTAR SUMAR TODOS LOS GASTOS DE TODO EL AÑO

void Suma_Gastos (MATRIZ G, int m, int g)
{
      int TotalGastos=0;

      for (int i=1; i<=m; i++)
       for(int j=1; j<=n; j++)
            TotalGastos = TotalGastos + G[i][j];

     Cout<<"\n \n EL TOTAL DE GASTOS DEL AÑO ES..."<<TotalGastos;
}

//PROGRAMA PRINCIPAL
void main(void)
{     MATRIZ GASTOS ;

      clrscr() ;
      Lee_Gastos (GASTOS, 12, 4) ; //lee gastos para 12 meses y 4 gastos
      Imprime_Gastos (GASTOS, 12,4); //Imprime los gastos de la matriz
      Suma_Gastos(GASTOS, 12,4);
      getch();
}
```

1.11. Registros (Estructuras)

Cuando se habló de los arreglos se mencionó que se trataba de una colección de datos, todos del mismo tipo, que era un tipo estructurado de datos, y que son ellos se podía solucionar un gran número de problemas. Sin embargo, en la práctica a veces se necesitan estructuras que permitan almacenar distintos tipos de datos (característica con la cual no cuentan los arreglos).

Ejemplo

Una compañía tiene por cada empleado los siguientes datos:

- Nombre (cadena de caracteres)
- Dirección (cadena de caracteres)
- Edad (entero)
- Sexo (carácter)
- Antigüedad (entero)

Si lo vemos gráficamente estos datos tenemos;

Nombre			Direccion					
Nom	Pat	Mat	Calle	Nro	Zona	Edad	Sexo	Antig
Juan Maria	Rodríguez Alvarez	Salas Vargas	Av. Arce Calle 12	123 1345	Central Obrajes	25 29	M F	2 3

Si se quisiera almacenar estos datos no sería posible usar un arreglo, ya que sus componentes deben ser todos del mismo tipo.

1.11.1. Definición De Registro y declaración de la variable de registro.

Un registro es un dato estructurado, donde cado uno de sus componentes se denomina campo. Los campos de un registro pueden ser todos de diferentes tipos. Por lo tanto también podrán ser registros o arreglos. Cada campo se identifica por un nombre único (el identificador de campo). No se establece orden entre los campos.

Para definir un registro en el lenguaje C, utilizamos la siguiente sintaxis:

```
struct Nom_Registro
{
      tipo_dato1 campo1;
      tipo_dato1 campo2;

      tipo_daton campon;
};
//declaración de la variable Variable_registro de tipo
// Nom_Registro

Nom_Registro Variable_registro ;
```

Donde:

Nom_Registro: Es el nombre o identificador del registro.

campoi: Es el nombre del campo i
tipo_datoi: es el tipo de dato del campo i.

EJEMPLO
Definir la estructura para los datos de un empleado y luego declarar la variable de registro.

```
struct EMPLEADO
{
      char nombre[20];
      char direccion[30];
      int edad;
      int antigüedad:
      char sexo;
};

EMPLEADO E ;
```

1.11.2. Acceso a los Campos De Un Registro

Como un registro es un dato estructurado no puede accecsarse directamente como un todo, sino que debe especificarse qué elemento (campo) del registro interesa. Para ello, en la mayoría de los lenguajes se sigue la siguiente sintaxis:

```
Variable_registro.campo
```

Ejemplo:

Escribir un programa para leer los registros de N empleados de una empresa.

```
#include<iostream.h>
#include<conio.h>

struct EMPLEADO
{
      char nombre[20];
      char direccion[30];
      int edad;
      int antiguedad:
      char sexo;
};

//PROCEDIMIENTO PARA LEER LOS DATOS DE UN EMPLEADO
EMPLEADO LeeEmpleado (void)
{
      EMPLEADO E;

      cout<<"\n Nombre del Empleado "; cin>> E.nombre;
      cout<<"\n Dirección del Empleado "; cin>> E.direccion;
      cout<<"\n Edad del Empleado "; cin>> E.edad;
      cout<<"\n Antigüedad del Empleado "; cin>> E.antiguedad;
      cout<<"\n Sexo del Empleado "; cin>> E.sexo;
 return ( E );
}

// PROCEDIMIENTO PARA IMPRIMIR LOS DATOS PARA UN EMPLEADO
void ImprimeEmpleado (EMPLEADO E)
{
      cout<<"\n NOMBRE : "<<E.nombre;
      cout<<"\n DIRECCION : "<<E.direccion;
      cout<<"\n EDAD : "<<E.edad;
      cout<<"\n ANTIGUEDAD : "<<E.antiguedad;
      cout<<"\n SEXO : "<<E.sexo;
}

// PROCEDIMIENTO LEER LOS DATOS DE LOS EMPLEADOS EN UN VECTOR
void Lee_N_Empleados (EMPLEADO VE[50], int N)
{ //llena el vector de empleados VE con registros de N empleados
      for (int i=1; i<= N; i++)
      {     cout<<"\n INTRODUZCA LOS DATOS DEL EMPLEADO "<<i<<endl;
           VE[i]= LeeEmpleado();
      }

}
// PROCEDIMIENTO IMPRIMIR LOS DATOS DE LOS EMPLEADOS EN UN VECTOR
void Imprime_N_Empleados (EMPLEADO VE[50], int N)
{     for (int i=1; i<= N; i++)
      {     cout<<"\n EMPLEADO "<<i<<endl;
            ImprimeEmpleado( VE[i]);
      }
getch();
}
```

```
//PROGRAMA PRINCIPAL
void main (void)
{
      EMPLEADO ve[50];
      int N;
      clrscr( );

      cout<<"\n Cuantos empleados desea registrar ?";
      cin >>N;
            Leer_N_Empleados (ve ,  N);
            Imprimir_N_Empleados (ve, N);
      getch( );
}
```

1.11.3. Funciones en el interior de estructuras:

C++, al contrario que C, permite incluir funciones en el interior de las estructuras. Normalmente estas funciones tienen la misión de manipular los datos incluidos en la estructura.

Aunque esta característica se usa casi exclusivamente con las clases, como veremos más adelante, también puede usarse en las estructuras.

Dos funciones muy particulares son las de inicialización, o constructor, y el destructor. Veremos con más detalle estas funciones cuando asociemos las estructuras y los punteros.

1.11.4. Asignación de estructuras:

La asignación de estructuras está permitida, pero sólo entre variables del mismo tipo de estructura, salvo que se usen constructores, y funciona como la intuición dice que debe hacerlo.

Veamos un ejemplo:

```
struct Punto {
   int x, y;
   Punto() {x = 0; y = 0;}
} ;

Punto Punto1, Punto2;

int main()
{
   Punto1.x = 10;
   Punto1.y = 12;
   Punto2 = Punto1;
}
```

La línea:

```
Punto2 = Punto1;
```

equivale a:

```
Punto2.x = Punto1.x;
Punto2.y = Punto1.y;
```

1.11.5. Arrays de estructuras:

La combinación de las estructuras con los arrays proporciona una potente herramienta para el almacenamiento y manipulación de datos.

Ejemplo:

```
struct Persona
{
   char Nombre[65];
   char Direccion[65];
   int AnyoNacimiento;
};
Persona Plantilla[200];
```

Vemos en este ejemplo lo fácil que podemos declarar el array Plantilla que contiene los datos relativos a doscientas personas.

Podemos acceder a los datos de cada uno de ellos:

```
cout << Plantilla[43].Direccion;
```

O asignar los datos de un elemento de la plantilla a otro:

```
Plantilla[0] = Plantilla[99];
```

1.11.6. Estructuras anidadas:

También está permitido anidar estructuras, con lo cual se pueden conseguir superestructuras muy elaboradas.

Ejemplo:

Definiremos una estructura para persona con los datos: Nombre (nombre de pila, paterno, materno), Direccion (Calle, nro y zona) y telefono.

```
struct stDireccion
{
   char Calle[64];
   int Nro;
   char Zona[32];
};

Struct stNombre
{
   char nom[20];
   char pat[20];
   char mat[25];
};

struct stPersona
{
   stNombre Nomb;
   stDireccion Direccion;
   long int Telefono;
};
...
```

En general, no es una práctica corriente definir estructuras dentro de estructuras, ya que resultan tener un ámbito local, y para acceder a ellas se necesita hacer referencia a la estructura más externa.
Por ejemplo para declarar una variable del tipo stNombre hay que utilizar el operador de acceso (::):

```
stPersona::stNombre NombreAuxiliar;
```

Sin embargo para declarar una variable de tipo stDireccion basta con declararla:

```
stDireccion DireccionAuxiliar;
```

1.12. Conjuntos

El conjunto es también un tipo de dato estructurado. Puede definirse un conjunto como una colección de objetos del mismo tipo base. El tipo base puede ser solamente un tipo ordinal (enteros, caracteres, enumerados y subrangos).

1.12.1. Definición De Conjuntos

Los conjuntos serán definidos de la siguiente manera:

Ident_conjunto = CONJUNTO DE tipo_base

Donde:

Tipo_base es cualquier tipo ordinal de dato.

Ejemplo:
Se definen los conjuntos NUMEROS, MAYUSCULAS y ALUMNOS. NUMEROS es el tipo conjunto formado por todos los números enteros comprendidos entre el 1 y el 50 inclusive. MAYUSCULAS es el tipo conjunto formado por las letras mayúsculas, y finalmente ALUMNOS es el tipo conjunto formado por todos los elementos del tipo enumerado NOMBRES.

```
NOMBRES    = (Juan, Jose, Julio,Javier)
NUMEROS    = CONJUNTO DE 1..50
MAYUSCULAS = CONJUNTO DE 'A'..'Z'
ALUMNOS    = CONJUNTO DE nombres
```

1.13. Matrices Poco Densas

Matriz es un término matemático utilizado para definir un conjunto de elementos organizados por medio de renglones y columnas, equivalente al término arreglo bidimensional utilizado en computación.

Poco Densa indica una proporción muy alta de ceros entre los elementos de la matríz. Es decir una matriz poco densa es aquella que tiene gran cantidad de elementos ceros.

Ejemplo:
La matriz A de 4 filas por 4 columnas, del total de elementos que es 16, solo 4 de ellos son diferentes de cero.

A =

0	1	1	0
0	0	1	0
0	1	0	0
0	1	0	0

Existen diversos métodos para almacenar los valores diferentes de cero de una matríz poco densa. A continuación presentamos uno de ellos.

Arreglo de Registros

Se utiliza un arreglo unidimensional, donde cada elemento es un registro formado por tres campos: uno para guardar la fila donde se encontró el valor diferente de cero, otro para guardar la columna, y el tercero para guardar el valor del elemento distinto de cero de la matriz.

El siguiente programa nos permite registrar una matriz poco densa

```
#include<iostream.h>
#include<conio.h>

struct PocoDensa
{
    int fila;
    int col;
    int valor;
};

//PROCEDIMIENTO PARA LLENAR LA MATRIZ POCO DENSA
Void LeePocoDensa( PocoDensa PD[20])
{
      int  F, C, k;
      int e;
      cout<< "\n Cual es la dimensión de la matríz poco densa
(filas y columnas)  "
      cin>>M>>N;
      k=1;
     for (int i= 1; i<= F; i++)
     for (int j=1; j<=C;j++)
     {   cout<< "\n Ingrese el elemento "<<i<<"  "<<j<<"  ";
          cin>>e;
           if (e != 0)
           {
                PD[k].fila = i;
          PD[k].col =j;
                PD[k].valor = e;
                k++;
            }
       PD[0].fila = F ;  PD[0].col = C;  PD[0].valor= k-1;
  }
//PROGRAMA PRINCIPAL
Void main (void)
{
     PocoDensa M[20]; //M es un vector de registros
     clrscr() ;
     LeePocoDensa(M) ;
     getch() ;
}
```

1.14. Archivos

- El almacenamiento de datos en variables y arreglos es temporal; al terminar un programa todos estos datos se pierden. Para la conservación permanente de grandes cantidades de datos se utilizan los archivos.

- Las computadoras almacenan los archivos en dispositivos de almacenamiento secundario, especialmente en discos.

- C ve cada uno de los archivos simplemente como un flujo secuencial de bytes. Cada archivo termina con un marcador de fin de archivo.

- Abrir un archivo regresa un apuntador a una estructura FILE (definida en <stdio.h>) que contiene información utilizada para procesar dicho archivo.

Modos de apertura de archivos

w :Para crear un archivo, o para descartar el contenido de un archivo antes de escribir datos.
r :Para leer un archivo existente.
a :Para añadir registros al final de un archivo existente.
r+ :Para abrir un archivo de tal forma que pueda ser escrito y leído (abre un archivo de lectura y escritura para actualizar)
w+ :Genera un archivo para lectura y escritura. Si el archivo ya existe, se abre y el contenido se descarta
a+ :Abre un archivo para lectura y escritura - toda escritura se efectuará al final del archivo. Si no existe, será creado.

Si al abrir un archivo en cualquiera de los modos anteriores ocurre un error, fopen regresará NULL.

EJEMPLO
ARCHIVO DE TEXTO CREACION Y LECTURA. Lee y escribe en un archivo de texto de n números sus cubos.

```
//ARCHIVOS DE TEXTO
#include<iostream.h>
#include<conio.h>
#include<process.h>
#include<stdio.h>

FILE *Miarch;

void crear(void)
{
      int a,n;

      if ( (Miarch=fopen("C:\\prueba.dat","w"))==NULL)
      {
            cout<<"no se puede abrir el archivo \n";
            exit(1);
      }
      else
      {
            cout<<"Cuantos numeros quieres introducir al archivo "<<endl;
            cin>>n;
            for (int i=1; i<=n; i++)
            {
                  a=i*i*i;
                  fprintf(Miarch,"\n %d  %d",i,a);
            }
            fclose(Miarch);
      }
}
void leer(void)
{  int a,b;

      if ((Miarch=fopen("c:\\prueba.dat","r"))==NULL)
      {
            cout<<"no se puede abrir el archivo "<<endl;
            exit(1);
      }
      else
      {     cout<<"\n  n     n^3";
            fscanf(Miarch,"%d%d",&a,&b);
            while (!feof(Miarch))
            {
            cout<<"\n "<<a<<" "<<b;
            fscanf(Miarch,"%d%d",&a,&b);
            }
            fclose(Miarch);
      }
}
void main(void)
{
      clrscr();
      crear();
      leer();
}
```

```
// Ejemplo de ficheros de acceso aleatorio.
#include <stdio.h>
#include <stdlib.h>

struct stRegistro
 {
   char valido;  // Campo que indica si el registro es válido S->Válido, N->Inválido
   char nombre[34];
   int dato[4];
};

int Menu();
void Leer(struct stRegistro *reg);
void Mostrar(struct stRegistro *reg);
void Listar(long n, struct stRegistro *reg);
long LeeNumero();
void Empaquetar(FILE **fa);

int main()
{
   struct stRegistro reg;
   FILE *fa;
   int opcion;
   long numero;
   fa = fopen("alea.dat", "r+b");          // Este modo permite leer y escribir
   if(!fa) fa = fopen("alea.dat", "w+b");  // si el fichero no existe, lo crea.
   do {
      opcion = Menu();
      switch(opcion) {
         case '1': // Añadir registro
            Leer(&reg);
            // Insertar al final:
            fseek(fa, 0, SEEK_END);
            fwrite(&reg, sizeof(struct stRegistro), 1, fa);
            break;
         case '2': // Mostrar registro
            system("cls");
            printf("Mostrar registro: ");
            numero = LeeNumero();
            fseek(fa, numero*sizeof(struct stRegistro), SEEK_SET);
            fread(&reg, sizeof(struct stRegistro), 1, fa);
            Mostrar(&reg);
            break;
         case '3': // Eliminar registro
            system("cls");
            printf("Eliminar registro: ");
            numero = LeeNumero();
            fseek(fa, numero*sizeof(struct stRegistro), SEEK_SET);
            fread(&reg, sizeof(struct stRegistro), 1, fa);
            reg.valido = 'N';
            fseek(fa, numero*sizeof(struct stRegistro), SEEK_SET);
```

```
            fwrite(&reg, sizeof(struct stRegistro), 1, fa);
            break;
         case '4': // Mostrar todo
            rewind(fa);
            numero = 0;
            system("cls");
            printf("Nombre                                Datos\n");
            while(fread(&reg, sizeof(struct stRegistro), 1, fa))
               Listar(numero++, &reg);
            system("PAUSE");
            break;
         case '5': // Eliminar marcados
            Empaquetar(&fa);
            break;
      }
   } while(opcion != '0');
   fclose(fa);
   return 0;
}

// Muestra un menú con las opciones disponibles y captura una opción del
usuario
int Menu()
{
   char resp[20];
   do {
      system("cls");
      printf("MENU PRINCIPAL\n");
      printf("--------------\n\n");
      printf("1- Insertar registro\n");
      printf("2- Mostrar registro\n");
      printf("3- Eliminar registro\n");
      printf("4- Mostrar todo\n");
      printf("5- Eliminar registros marcados\n");
      printf("0- Salir\n");
      fgets(resp, 20, stdin);
   } while(resp[0] < '0' && resp[0] > '5');
   return resp[0];
}

// Permite que el usuario introduzca un registro por pantalla
void Leer(struct stRegistro *reg)
{
   int i;
   char numero[6];
   system("cls");
   printf("Leer registro:\n\n");
   reg->valido = 'S';
   printf("Nombre: ");
   fgets(reg->nombre, 34, stdin);
```

```
   // la función fgets captura el retorno de línea, hay que eliminarlo:
   for(i = strlen(reg->nombre)-1; i && reg->nombre[i] < ' '; i--)
      reg->nombre[i] = 0;
   for(i = 0; i < 4; i++) {
      printf("Dato[%1d]: ", i);
      fgets(numero, 6, stdin);
      reg->dato[i] = atoi(numero);
   }
}

// Muestra un registro en pantalla, si no está marcado como borrado
void Mostrar(struct stRegistro *reg)
{
   int i;
   system("cls");
   if(reg->valido == 'S') {
      printf("Nombre: %s\n", reg->nombre);
      for(i = 0; i < 4; i++) printf("Dato[%1d]: %d\n", i, reg->dato[i]);
   }
   system("PAUSE");
}

// Muestra un registro por pantalla en forma de listado,
// si no está marcado como borrado
void Listar(long n, struct stRegistro *reg)
{
   int i;
   if(reg->valido == 'S') {
      printf("[%6ld] %-34s", n, reg->nombre);
      for(i = 0; i < 4; i++) printf(", %4d", reg->dato[i]);
      printf("\n");
   }
}

// Lee un número suministrado por el usuario
long LeeNumero()
{
   char numero[6];
   fgets(numero, 6, stdin);
   return atoi(numero);
}

// Elimina los registros marcados como borrados
void Empaquetar(FILE **fa)
{
   FILE *ftemp;
   struct stRegistro reg;

   ftemp = fopen("alea.tmp", "wb");
```

```
   rewind(*fa);
   while(fread(&reg, sizeof(struct stRegistro), 1, *fa))
      if(reg.valido == 'S')
         fwrite(&reg, sizeof(struct stRegistro), 1, ftemp);
   fclose(ftemp);
   fclose(*fa);
   remove("alea.bak");
   rename("alea.dat", "alea.bak");
   rename("alea.tmp", "alea.dat");
   *fa = fopen("alea.dat", "r+b");
}
```

Ejercicios Propuestos

CONTESTA LAS SIGUIENTES PREGUNTAS DE OPCIÓN MÚLTIPLE:

1. ¿Cuál de las siguientes descripciones de C++ es verdadera?
   a) Es un lenguaje pequeño (de instrucciones) como C.
   b) Es un lenguaje para la enseñanza como Pascal.
   c) Es un lenguaje seguro, con recolección de basura, como Java.
2. Los compiladores siguen la regla de leer el mayor número de carácteres que pueden ensamblarse en una construcción sintáctica válida. Entonces, ¿qué ocurre con el siguiente código?
   a) a++++++++b;
   b) No compila.
   c) Imprime el valor de *a* + *b*.
   d) No hace ninguna operación con *a* y *b* e imprime los valores originales.
   e) Indica un *warning*.
3. ¿Cuál de las siguientes listas de tipos de datos de C++ están ordenadas ascendentemente por tamaño, calculado con *sizeof()*?.
   a) char, int, short, long
   b) long, int, char, short
   c) char, long, int, short
   d) char, short, int, long

ARREGLOS DE UNA DIMENSIÓN

1. Declare e inicialice un vector de N elementos de modo que los componentes de indice par valgan 0 y los de indice impar valgan 1. Ejm. V(1,0,1,0, ...)

2. Escriba un programa que almacene en un vector los N primeros números de Fibonacci. Una vez calculados, el programa los mostrará por pantalla en orden inverso.

3. Escriba un programa que almacene en un vector los N primeros números de Fibonacci. Una vez calculados, el programa pedirá al usuario que introduzca un número y dirá si es o no es uno de los N primeros números de Fibonacci.

4. Hallar la mediana, en el anterior planteado en la pagina 6 del texto
5. Modifica el programa anterior para que permita efectuar cálculos con N personas.
6. Modifica el programa del ejercicio anterior para que muestre, además, cuántas edades hay entre 0 y 9 años, entre 10 y 19, entre 20 y 29, etc. Considera que ninguna edad es igual o superior a 150. Ejemplo: si el usuario introduce las siguientes edades correspondientes a 12 personas:

   10 23 15 18 20 18 57 12 29 31 78 28

   el programa mostrará (además de la media, desviación estándar, moda y mediana), la siguiente tabla:

```
  0 - 9: 0
10 - 19: 5
20 - 29: 4
30 - 39: 1
40 - 49: 0
50 - 59: 1
60 - 69: 0
70 - 79: 1
80 - 89: 0
90 - 99: 0
100 - 109: 0
110 - 119: 0
120 - 129: 0
130 - 139: 0
140 - 149: 0
```

4 Modifica el programa para que muestre un histograma de edades. La tabla anterior se mostrará ahora como este histograma:

```
0 - 9:
10 - 19: *****
20 - 29: ****
30 - 39: *
40 - 49:
50 - 59: *
60 - 69:
```

```
70 - 79: *
80 - 89:
90 - 99:
100 - 109:
110 - 119:
120 - 129:
130 - 139:
140 - 149:
```

Como puedes ver, cada asterisco representa la edad de una persona.

5 Modifica el programa anterior para que el primer y último rangos de edades mostrados en el histograma correspondan a tramos de edades en los que hay al menos una persona. El histograma mostrado antes aparecerá ahora así:

```
10 - 19: *****
20 - 29: ****
30 - 39: *
40 - 49:
50 - 59: *
60 - 69:
70 - 79: *
```

6 Modifica el programa del ejercicio anterior para que muestre el mismo histograma de esta otra forma:

```
| ####### |         |          |          |          |          |          |
| ####### | ####### |          |          |          |          |          |
| ####### | ####### |          |          |          |          |          |
| ####### | ####### |          |          |          |          |          |
| ####### | ####### | ######## |          | ######## |          | ######## |
+---------+---------+----------+----------+----------+----------+----------+
 | 10 – 19 |   20 - 29 | 30 – 39 | 40 – 49 | 50 – 59 | 60 - 69  | 70 – 79 |
```

7 Diseñe un programa que pida el valor de N números enteros distintos y los almacene en un vector. Si se da el caso, el programa advertirá al usuario, tan pronto sea posible, si introduce un número repetido y solicitará nuevamente el número hasta que sea diferente de todos los anteriores. A continuación, el programa mostrará los N números por pantalla

8 Diseñe un programa C que lea y almacene en un vector N números enteros asegurándose de que sean positivos. A continuación, el programa pedirá que se introduzca una serie de números enteros y nos dirá si cada uno de ellos está o no en el vector. El programa finaliza cuando el usuario introduce un número negativo. Luego ordenar el vector, por el método de la burbuja.

9 En un arreglo se ha almacenado el número total de toneladas de cereales cosechadas durante cada mes del año anterior. Se desea la siguiente información:

i. El promedio anual de toneladas cosechadas
ii. ¿Cuántos meses tuvieron una cosecha superior al promedio anual?
iii. ¿Cuántos meses tuvieron una cosecha inferior al promedio anual?Escriba un programa que proporcione estos datos.

ARREGLOS MULTIDIMENSIONALES

20. Escriba un programa que intercambie por renglón los elementos de un arreglo bidimensional. Los elementos del renglón 1 deben intercambiarse con los del renglón N, los del renglón 2 con los del N-1, y así sucesivamente.

21. Escriba un programa que asigne valores a A, a partir de B teniendo en cuenta los siguientes criterios:

iv. Aij = (bi) si i <= j
v. Aij = 0 si i >j

REGISTROS

22. Una compañía distribuye N productos a distintos comercios de la ciudad. Para ellos almacena en un arreglo toda la información relacionada con su mercancía:

- Clave
- Descripción
- Existencia
- Mínimo a mantener de existencia
- Precio unitario

Escriba un programa que pueda llevar a cabo las siguientes opraciones:

a) Venta de un producto: se deben actualizar los campos que correspondan, y verificar que la nueva existencia no esté por debajo del mínimo. (datos: clave, cantidad_vendida)
b) Reabastecimientos de un producto: se deben actualizar los campos que correspondan. (Datos: clave, cantidad comprada)
c) Actualizar el precio de un producto: se deben proporcionar todos los datos relacionados con un producto. (Dato: clave)