# Estructuras de datos más "complejas"

En este documento comenzamos a trabajar con estructura de datos más "complejas", que permiten agrupar valores en una única variable y/o mantener una serie de valores en memoria y acceder a estos en cualquier momento durante la ejecución del programa.

En los próximos documentos iremos profundizando y trabajando con estas estructuras.

Temas a desarrollar:

- Registros o estructuras
- Introducción a arreglos: vectores, patrones de búsqueda y recorrido
- Resolución de problemas

# Los registros o estructuras en C++

# Uso de registros en C++

El lenguaje C++ así como otros permiten “agrupar” variables (también llamadas campos) de forma tal de manipularlas juntas. Hemos trabajado hasta ahora en varios ejemplos de esta característica:

**fechas**: está compuesta por día, mes y año. O puede ser en formato AAAAMMDD, o se le puede agregar el día de la semana

**estudiante**: está compuesto por su nombre y fecha de nacimiento

**mayor de una lista**: está compuesto por el valor y su ubicación en la lista

**Y así podemos pensar un sinnúmero de ejemplos tantos como programadores haya….y como algoritmos desarrollen.**

Pero, cuál es la ventaja. Pregunto esto porque me parece que necesito tener todas esas variables, caso contrario no veo dónde guardamos los datos.

Verdaderamente las ventajas son muchas y variadas, pero me voy a detener solo en una (por ahora).

Piensen en el ejercicio de buscar el mayor y el menor y mostrar su fecha (la de cada uno) de nacimiento y su nombre.

Necesitamos –por ejemplo para el mayor- las siguientes variables: diaMayor, mesMayor, anioMayor, nombreMayor, ordenMayor … y lo mismo para el menor … y lo mismo para ingresar los datos … y lo mismo … y lo mismo ….

Entonces, debemos escribir mucho, con nombres ligeramente modificados (dia, diaMayor, diaMenor, mes, mesMayor, mesMenor,…etc).

Y lo más incomodo **DEBEMOS PASAR MUCHOS PARÁMETROS A LAS FUNCIONES.**

He aquí el quid de la question, ver cómo organizar las variables con el afán de mejorar la calidad del código para trabajar más ordenadamente.

**Vaya, apareció (o está por aparecer) un nuevo TIPO DE DATOS**

## ¿Qué es un tipo de datos?

Preguntarnos esto ahora parece raro, si desde hace varias semanas estamos escribiendo algoritmos y usamos variables. Y como todos recordamos (o deberíamos recordar) una variable está asociada a un tipo de datos.

Si para números puede ser entera (int) o real (float), entero largo (long int) o real doble precisión (doublé), de carácter (char), o se cadena de caracteres (char x[10] o string), lógicas (bool)… Un montón de tipos de datos que ya existen y permiten que definamos las variables a utilizar en el programa.

Pero, cómo se relaciona el **nuevo tipo de datos** con lo anterior. O preguntado de otra manera ¿habrá alguna forma que el desarrollador de algoritmos defina sus propios tipos de datos?.

**VAYA SI LO HAY, claro, los lenguajes de programación permiten definir nuevos tipos de datos a partir de los preexistentes, y lo mejor, de una manera sencilla**

Entonces, manos a la obra, trabajemos en el armado de los nuevos tipos de datos a partir de los que conocemos (conocemos los que nos provee el lenguaje más los que vamos desarrollando).

Debemos tener en cuenta que primero definimos los tipos de datos y luego los usamos. Para ser claros, si vamos a definir una variable el tipo de datos debe existir tanto que sea provisto por el lenguaje o sea definido por el desarrollador.

Para esto vamos a usar dos palabras reservadas (propias del lenguaje) que lo definen todo: **typedef** y **struct**

# Vamos a los ejemplos

**Tipos de datos para cadenas de caracteres (permiten asignar nombres, descripciones, colores, etc:**

```
// definición de tipos de datos
 typedef char str30[30];     // para cadenas de 30 caracteres
 typedef char str100[100]; // para cadenas de 100 caracteres

// definición de variables
str30 color;          // para un color de hasta 30 caracteres
str30 nombre;       //  para un nombre de hasta 30 caracteres
str100 email;        // para una dirección de mail de hasta 100 caracteres
```

**SIN FUNCIONES**

```
#include<iostream>
using namespace std;
typedef char str30[30] ;
typedef char str100[100] ;

int main() {
  str30 nombre ;
  str100 email;
  cout<<"ingresar nombre y mail"<<endl;
  cin>>nombre>>email;
  cout <<endl<<nombre<<"  "<<email<<endl;
  return 0;
}
```

**CON FUNCIONES**


```
#include<iostream>
using namespace std;
typedef char str30[30] ;
typedef char str100[100] ;
// prototipo
void pedirDatos(str30 & nbe, str100 & mail) ;
int main() {
  str30 nombre ;
  str100 email;
  pedirDatos(nombre,  email) ;
  cout <<endl<<nombre<<"  "<<email<<endl;
  return 0;
}

void pedirDatos(str30 & nbe, str100 & mail) {
   cout<<"ingresar nombre y mail"<<endl;
   cin>>nbe>>mail;
}
```

## Tipos de datos para fechas, con día, mes, año y AAAAMMDD:

En el siguiente ejemplo en C++ se piden dos fechas, se agrupan los datos de cada una, se las convierte a AAAAMMDD y se las compara

```
#include<iostream>
using namespace std;
struct tFecha{
  int dia;
  int mes;
  int anio;
  long AAAAMMDD;
};
void pedirFecha(tFecha & f) ;
void convertirAAAAMMDD(tFecha & f);
int main() {
  tFecha fecha1, fecha2;
  pedirFecha(fecha1) ;
  pedirFecha(fecha2) ;
  if (fecha1.AAAAMMDD>fecha2.AAAAMMDD)
    cout<<"la fecha 2 es anterior" ;
  else
    if (fecha1.AAAAMMDD<fecha2.AAAAMMDD)
      cout<<"la fecha 1 es anterior";
    else
      cout<<"las fechas son iguales";
  return 0;
}
```

```
void pedirFecha(tFecha & f) {
  cout<<"dia, mes y anio"<<endl;
  cin>>f.dia>>f.mes>>f.anio;
   convertirAAAAMMDD(f);
}
void convertirAAAAMMDD(tFecha & f){
  f.AAAAMMDD = f.dia + 100*f.mes +
10000*f.anio;
}
```

```
dia, mes y anio
29
11
2014
dia, mes y anio
23
3
2010
la fecha 2 es anterior
--------------------------------
```

**Mediante la palabra reservada "struct" se define el tipo de datos. Presten atención a los símbolos {} que se utilizan para "encerrar los campos del tipo.**

**Observen también las funciones, donde se pasa solamente una variable que agrupa las 4 campos de la fecha (día, mes, año y en formato AAAAMMDD.**

**Observen también que mediante el símbolo punto se accede a cada uno de los campos de la variable de tipo struct.**

**PRESTEN ATENCIÓN: No se puede asignar datos a toda la estructura en una sola instrucción.**

## Tipos de datos para personas con nombre, mail y fechas de nacimiento (con día, mes, año y AAAAMMDD):

En el siguiente ejemplo en C++ se piden datos de dos personas y se compara su fecha de nacimiento

```
#include<iostream>
using namespace std;
typedef char str30[30] ;
typedef char str100[100] ;
struct tFecha{
  int dia;
  int mes;
  int anio;
  long AAAAMMDD;
};
struct tPersona {
   str30 nombre;
   str100 email;
  tFecha fechaNac;
};
// prototipo
void pedirNombreMail(str30 & nbe, str100 & mail) ;
void pedirFecha(tFecha & f) ;
void convertirAAAAMMDD(tFecha & f);
void pedirDatosPersona(tPersona & per) ;
int main() {
   tPersona per1, per2;
   pedirDatosPersona(per1)  ;
   pedirDatosPersona(per2)  ;
   if (per1.fechaNac.AAAAMMDD>per2.fechaNac.AAAAMMDD)
     cout<<per2.nombre<<" es mayor que "<<per2.nombre;
   else
    if (per1.fechaNac.AAAAMMDD<per2.fechaNac.AAAAMMDD)
      cout<<per1.nombre<<" es mayor que "<<per2.nombre;
    else
      cout<<per1.nombre<<" y "<<per2.nombre<<" tienen la misma edad";
 return 0;
}
```

```
void pedirFecha(tFecha & f) {
 cout<<"dia, mes y anio"<<endl;
 cin>>f.dia>>f.mes>>f.anio;
 convertirAAAAMMDD(f);
}
void convertirAAAAMMDD(tFecha & f){
  f.AAAAMMDD = f.dia + 100*f.mes +
10000*f.anio;
}
void pedirNombreMail(str30 & nbe, str100 &
mail) {
 cout<<"ingresar nombre y mail"<<endl;
 cin>>nbe>>mail;
}
void pedirDatosPersona(tPersona & per) {
  pedirNombreMail(per.nombre, per.email) ;
  pedirFecha(per.fechaNac);
}
```

```
ingresar nombre y mail
Julieta juli@mail.com
dia, mes y anio
23 3 2010
ingresar nombre y mail
Facundo facu@mail.com
dia, mes y anio
29 11 2014
Julieta es mayor que Facundo
--------------------------------
```

**Presten atención:**

**Primero se definen los tipos str y fecha y luego el tipo tPersona (así el compilador "conoce" los tipos usados en tpersona.**

**Los prototipos son los encabezdos de las funciones, que se definen para que el compilador los conozca cuando se los menciona, aunque aun no "sepa" cómo están resueltos.**

**Miren el ejemplo, se pueden separar los datos a ingresar con caracteres en blanco**

# Los arreglos de una dimensión (vectores) en C++

## Tenemos un problema:

Venimos resolviendo algoritmos que procesan series de datos, calculamos promedios, buscamos el más grande, también el más chico, quisimos saber a quién correspondía cada uno de estos y hasta el orden en que habían llegado.  Bah, hicimos un montón.

Pero, al menos a mi, nos quedó un sabor amargo.  Parece todo muy incompleto, ya sé cómo calcular el promedio, pero ¿podré saber cuántos estudiante superan ese ese promedio? Y cuántos están por debajo del promedio.

Y hay más….

Ordernamos dos números, luego hicimos lo mismo con tres números, y hasta nos animamos a ordenar cuatro números.  Al final de  todo los desafié a ordenar cinco y hasta seis.  Para quienes lo intentaron les podría preguntar si luego de esa tarea titánica se animan a ordena diez números y luego cien y mil,y…,y,….,etc.

No parece muy difícil, si ya comprendimos el algoritmos, para ordenar 100, primero ordenamos 99 y luego ordenamos el último, y para 99, primero 98, y para 98, primero 97 y así sucesivamente.  Es "algorítmico" pero muy laborioso. Se debe escribir mucho y siempre lo mismo. Es rutinario.

Y podemos pensar más, si en una lista de personas queremos saber cuántas tienen 12 años, pues sumamos a una variable cant12, y para 13, a una variable cant13,…, y asi agregamos algunos valores y variables más, hasta que se torna "inmanejable".  Como en el problema de ordenar, es algorítmico, pero noy tedioso de escribir.

**Entonces tenemos un problema!!!,  o más bien, CREEMOS QUE TENEMOS UN PROBLEMA**

Seamos pacientes y avancemos con la lectura (y la clase), vamos a estudiar un nuevo tipo de datos que facilita la resolución de problemas de esta característica. Y luego si, podríamos decir que estamos en condiciones de resolver una familia muy grande de algoritmos.

## Estructuras multidimensionales : una dimensión (vectores), más de una dimensión (matrices)

Es una estructura de datos estática que almacena bajo el mismo nombre (variable) a una colección de datos del mismo tipo.
Toda vez que se necesita "almacenar" en memoria un conjunto de datos de la misma característica (edades de 100 personas, temperatura de todas las provincias en todos los días del año) es necesario contar con un tipo de variable especial, que permita de forma organizada acceder a todos sus datos de manera sencilla.

**Los vectores (arreglos de una dimensión) se caracterizan por:**

- Son un conjunto de variables que se identifican con el mismo nombre. A cada una de las variables las podemos denominar "celdas"
- Se almacenan los elementos (las celdas) en posiciones contiguas de memoria.
- Para hacer referencia a esos elementos (cada celda) se necesita utilizar un índice que especifica el lugar que ocupa cada elemento dentro del vector.
- Permite acceder a todos los elementos secuencialmente (desde el primero al último)
- Permite acceder a los elementos en forma aleatoria (a cualquier contenido, sin necesidad de recorrer secuencialmente)
- Permite buscar el menor valor y el mayor valor
- Permite ordenar los elementos de menor a mayor o de mayor a menor

**Las matrices** (arreglos de dos dimensiones) permiten representar tablas de doble entrada, por ejemplo representar en las filas a los estudiantes y en las columnas cada día de clase (1ra, 2da, …), y el contenido un valor lógico (verdadero o falso) en caso de asistencia al curso del estudiante en cada día.
Se caracterizan por

- Mediante un mismo nombre se puede acceder a todas las variables
- Para hacer referencia a una celda debemos fijar los índices correspondientes a fila y a columna
- Permite acceder en forma secuencial fila por fila y dentro de cada fila, columna por columna
- Permite acceder en forma directa a una celda, fijante la fila y la columna
- Se puede utilizar de igual forma que las tablas en las planillas de cálculo, por ejemplo:
  - Sumar el contenido de cada fila o columna
  - Contar contenido por fila y/o columna

**¡¡¡Pensemos en los ejemplos vistos hasta ahora!!!**

**Si podemos leer las edades y mantenerlas en memoria, podremos realizar varias procesos con esos datos. Si se puede acceder a todos los datos con el mismo nombre, entonces se puede generalizar el ordenamiento de las edades.**

## En síntesis

Un arreglo es  un conjunto de variables que se pueden agrupar, esto es usar todas con el mismo identificador, permitiendo de esa manera generalizar problemas "de repetición".

Todas las variables del arreglo son del mismo tipo y se accede a cada una de estas mediante su posición dentro del arreglo, comúnmente denominado "índice".

Permite resolver problemas del tipo "calcular promedio de todas las edades de los alumnos de curso y cuántos superan el mismo", donde se deben solicitar las edades, calcular su promedio y luego comparar todas las edades con el promedio (sin volver a pedir las edades)

## Algunos ejemplos en C++

| **Definir una variable para 60 edades** | **Acceder a los elementos de una variable con 60 edades** |
|---|---|
| int edadesPersonas[60];<br><br>Donde:<br>El tipo de cada elemento de **edadesPersonas** es **int**<br>La cantidad de edades es **60**<br>El identificador de las edades es **edadesPersonas** | edadesPersonas[5] = 30 ;<br>Int i = 0;<br>edadesPersonas[i] = 12;<br>Donde:<br>En el primer caso se asigna 30 al sexto elemento del vector<br>En el segundo caso se asigna 12 al primer elemento del vector |

**Para definir un vector se debe indicar el tipo de datos, el identidicador (tener presente que es una variable) y la cantidad de elementos del vector (TOPE).**
**Los elementos están entre la "posición" 0 y la posición TOPE-1 del vector**

**Recordemos que en C++ se pueden definir tipos de datos y la única restricción es que cada identificador se defina de un tipo resuelto anteriormente.**

**Entonces, hasta se podría definir un tipo STRUCT y luego un tipo o una variable del tipo estructura.**

Podemos pensar en el tipo **struct tFecha** definido anteriormente.

```
struct tFecha{
   int dia;
   int mes;
   int anio;
}
```

**Ahora definamos un vector del tipo tFecha que permita "guardar" 30 fechas.**

```
tFecha vFecNac[30];
```

**Veamos ahora cómo se accede a los elementos**

Asignemos el día, mes y año a la cuarta posición del vector

```
vFecNac[3].dia = 23;
vFecNac[3].mes = 3;
vFecNac[3].anio = 2010;
```

Leamos los valores de una fecha y asignamos a la tercera posición del vector

```
cin>> vFecNac[2].dia>> vFecNac[2].mes>> vFecNac[2].anio;
```

Mostremos la fecha de la tercera posición

```
Cout<< vFecNac[2].dia<<'/'<< vFecNac[2].mes<< <<'/'<<
vFecNac[2].anio;
```

**PRESTEN ATENCIÓN: No se puede asignar datos a todo el vector en una sola instrucción.**

**También estamos viendo que se pueden definir tipos de datos (unas pocas páginas atrás trabajamos con estructuras), entonces podríamos definir un tipo de datos vector y luego variables de ese tipo de datos.**

Podemos pensar en el tipo **struct tFecha** definido anteriormente.

```
struct tFecha{
   int dia;
   int mes;
   int anio;
}
```

**Ahora definamos un tipo de datos vector del tipo tFecha que permita "guardar" 30 fechas.**

```
typedef tFecha tVectorFecNac[30];  // es un tipo de datos

tVectorFecNac vecFechaNac;
```

**Veamos ahora cómo se accede a los elementos**

Asignemos el día, mes y año a la cuarta posición del vector

```
vecFechaNac[3].dia = 29;
vecFecNac[3].mes =11;
vecFecNac[3].anio = 2014;
```

Leamos los valores de una fecha y asignamos a la decima posición del vector

```
cin>> vFechaNac[3].dia>> vecFechaNac[3].mes>>
vecFechaNac[3].anio;
```

Mostremos la fecha de la tercera posición

```
Cout<< vecFechaNac[3].dia<<'/'<< vecFechaNac[3].mes<< <<'/'<<
vecFechaNac[3].anio;
```

**Esta forma de desarrollo es muy útil cuando se trabaja con funciones. Definimos los parámetros de las funciones usando los tipos de datos. Así se evita cualquier tipo de incompatibilidad de tipos.**

# Resolución de problemas con registros y vectores en C++

# Armar un vector con los primeros diez números pares.

```
#include<iostream>
using namespace std;
/*
Creamos y mostramos un vector
con los 10 primeros números
pares
*/
typedef int tVector[10];
void mostrarVector(tVector v);

int main() {
  tVector vector;
  // se crea el vector
  for (int i=0; i<10;i++) {
    vector[i] = 2 * (i+1) ;
  }
  mostrarVector(vector);
}
```

```
void mostrarVector(tVector v){
   cout<<"Los primeros 10 numeros
pares"<<endl;
   for (int i=0; i<10;i++){
      cout<<"en la celda "<<i<<" esta el
valor "<<v[i]<<endl;
   }
}
```

```
Los primeros 10 numeros pares
en la celda 0 esta el valor 2
en la celda 1 esta el valor 4
en la celda 2 esta el valor 6
en la celda 3 esta el valor 8
en la celda 4 esta el valor 10
en la celda 5 esta el valor 12
en la celda 6 esta el valor 14
en la celda 7 esta el valor 16
en la celda 8 esta el valor 18
en la celda 9 esta el valor 20
```

**En la sentencia "for" se puede definir la variable de "control".**
**A cada elemento del vector se le asigna el doble del contenido de (i+1).**

**¿Porqué (i+1)?, observen que i va de 0 a 9, entonces hay que sumarle 1 para llevarlo de 1 a 10.**
**Multiplicarlo por 2 permite llevarlo a un número par.**

**La función "mostrarVector" recibe el vector, lo recorre y va mostrando el índice (la posición de cada celda) y el contenido de la celda**

## Tarea: modificar el programa para mostrar los 10 primeros números impares

# Ingresar una fecha (día, mes y año) y mostrar la misma con el nombre del mes.

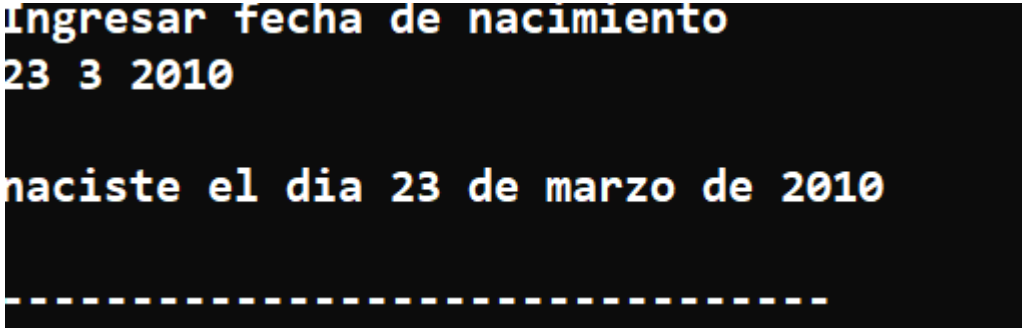
```
#include<iostream>
using namespace std;
/*
se pide una fecha, y se la muestra
con el nombre del mes
*/

// se declara una constante "global" con el nombre de los meses

const  string meses[12]={"Enero",
"febrero","marzo","abril","mayo","junio","julio","agosto",
"septiembre","octubre","noviembre","diciembre"};

int main() {
  int dia, mes, anio;
  cout<<"Ingresar fecha de nacimiento"<<endl;
  cin>>dia>>mes>>anio;
  cout<<endl;
  cout<<"naciste el dia "<<dia<<" de "<<meses[mes-1]<<"
de "<<anio<<endl;

}
```




**Se define una constante (const) de tipo vector de caracteres y se inicializa con el nombre de cada mes.**

**Cuando se muestra la fecha, el índice que se utiliza es el valor de (mes-1).**

**Tener en cuenta que los meses son del 1 al 12 mientras que las celdas de un vector van del 0 al 11 (dimensión – 1)**

**Tarea: en los programas publicados van a encontrar uno que muestra el día de la semana. Modificarlo para armar un vector con el nombre de los días.**

# Para una serie de números que finalizan en 0, indicar cuántos superan el promedio.

```
#include<iostream>
using namespace std;
/* promediar hasta 60 edades */
typedef int tVecEdades[60];
// prototipos
void inicializarEdades(tVecEdades, int &);
void pedirEdades(tVecEdades, int &) ;
float calculoPromedio(tVecEdades, int );
int calculoMayores(tVecEdades, int, float);
int main() {
  tVecEdades vecEdades;
  int cantEdades=60;
  int cantMayores ;
  float prom;
  inicializarEdades(vecEdades, cantEdades);
  pedirEdades(vecEdades, cantEdades) ;
  if (cantEdades!=0) {
    prom = calculoPromedio(vecEdades, cantEdades);
    cantMayores = calculoMayores(vecEdades, cantEdades,
prom);
   cout<<"El promedio es "<<prom<<" y lo superan
"<<cantMayores<<" personas"<<endl;
  }
  return 0;
}
```

```
void inicializarEdades(tVecEdades v, int & c){
  for (int i=0; i<c ; i++)  v[i] = 0 ;
  c = 0 ;
}
```

```
void pedirEdades(tVecEdades v, int & c)  {
  int edad;
  cout<<"Ingresar edad (0=fin)"<<endl;
  cin>>edad ;
  while ((c<60)&& (edad!=0)) {
    v[c] = edad ;
    c++ ;
    cout<<"Ingresar edad (0=fin)"<<endl;
    cin>>edad ;
  }
}
```

```
float calculoPromedio(tVecEdades v, int c){
  int suma=0 ;
  for (int i=0; i<c ; i++)   suma = suma + v[i] ;
  return suma / c ;
}
```

```
int calculoMayores(tVecEdades v, int c, float prom)
{
  int cantMayores = 0 ;
  for (int i=0; i<c ; i++) {
     if(v[i] >= prom) antMayores++ ;
  }
  return cantMayores;
}
```

```
Ingresar edad (0=fin)
20
Ingresar edad (0=fin)
18
Ingresar edad (0=fin)
22
Ingresar edad (0=fin)
24
Ingresar edad (0=fin)
16
Ingresar edad (0=fin)
0
El promedio es 20 y lo superan 3 personas

--------------------------------
```

**Se deben ingresar todos los números e ir asignándolos a un vector.**

**Se debe calcular el promedio sumando todos los valores que contiene el vector.**

**Se debe recorrer el vector contando cuántos están por encima del promedio**

**Tarea: modificar el programa para que también cuente cuántos están por debajo del promedio.**

# Ejemplo de declaración y uso de un vector de registro.

## Definimos una estructura de empleados y luego un vector de empleados

Se define la estructura **tRegEmp**;

```
struct     tRegEmpl {
   long int DNI;
   string apellido;
   string nombre;
}
```

Se accede a los datos del vector de la siguiente manera:

```
cin>>vemp[2].DNI;
cin>>vemp[2].apellido;
cin>>vemp[2].nombre
```

Se define un vector para 10 empleados y una variable ese tipo:

```
typedef tRegEmpl tVectorEmpl[10];

tVectorEmpl  vemp
```

Se define variable de tipo empleado y se asigna valores:

```
tRegEmpl emp;

cin>>emp.DNI;
cin>>emp.apellido;
cin>>emp.nombre
```

**Tarea: modificar el programa del promedio para pedir la edad y el nombre y mostrar el nombre de aquellos que superan el promedio.**

# Pedir datos de tres empleados y mostrarlos en pantalla.

## Pequeña aplicación con la estructura definida en la página anterior

```
#include <iostream>
using namespace std;
#define _CANTEMPLEADOS 3

struct tRegEmp {
  long int DNI;
  string apellido ;
  string nombre ;
}

typedef tRegEmp
tVectorEmp[_CANTEMPLEADOS];
void incializarV(tVectorEmp vEmp, int) ;
void cargarEmp(tVectorEmp vEmp, int ) ;
void mostrarEmp(tVectorEmp vEmp, int) ;
int main(){
   tVectorEmp  vEmp ; // se define un
vector empleado
   incializarV(vEmp,_CANTEMPLEADOS) ;
   cargarEmp(vEmp,_CANTEMPLEADOS) ;
   mostrarEmp(vEmp,_CANTEMPLEADOS) ;
}
```

```
void incializarV(tVectorEmp v, int cant) {
  for (int i=0;i<cant;i++) {
    v[i].DNI = 0 ;
    v[i].nombre =  "";
    v[i].apellido =  "";
  } ;
};
```

```
void cargarEmp(tVectorEmp v, int cant) {
  for (int i=0;i<cant;i++) {
     cout<<"Ingresar DNI, nombre y
apellido separados por enter"<<endl;
    cin>>v[i].DNI ;
   cin>>v[i].nombre ;
   cin>>v[i].apellido ;
  }
};
```

```
void mostrarEmp(tVectorEmp v, int cant) {
   for (int i=0;i<cant;i++) {
     cout<<"DNI,    nombre,  apellido"<<endl;
      cout<<v[i].DNI<<"   "<<v[i].nombre<<"
"<<v[i].apellido<<endl ;
   }
};
```

```
Ingresar DNI, nombre y apellido sepa

22122332 Marcelo Lipkin
DNI,    nombre,  apellido
123456   Julieta   Lipkin
DNI,    nombre,  apellido
234567   Facundo   Lipkin
DNI,    nombre,  apellido
22122332   Marcelo   Lipkin


-----------------------------
```

**Este ejemplo está resuelto mediante funciones. Es importante que el main tenga poco código, esto es invocar funciones. De esta forma el programa está modularizado, con funciones específicas, reutilizables y probadas.**

**Se logra así un código más claro.**

**Tarea: modificar el  programa para que pida datos hasta 30 empleados o cuando ingresa un DNI en 0.**

# Cálculo de un polinomio.

## Se piden los coeficientes y potencias de un polinomio y luego se calcula para un valor de X

```
#include<iostream>
using namespace std;
/*  cálculo de un polinomio
  se ingresan pares de valores correspondientes a
coeficientes y potencias
  para armar el poliniomio

  se ingresan valores de x para reolver el polinomio
*/
#define _CANTTERMINOS 5
struct tRegTermino {
   int coeficiente;
   int potencia;
};
typedef tRegTermino tVecPolinomio[10] ;
Void  inicializarPolinomio(tVecPolinomio) ;
Void  ingresarPolinimio(tVecPolinomio);
Void  mostrarPolinomio(tVecPolinomio);
long int calcularPolinomio(tVecPolinomio, int) ;
long int fPot(int, int) ;
int main() {
  tVecPolinomio polinomio;
  long int resultado;
  inicializarPolinomio(polinomio) ;
  ingresarPolinimio(polinomio);
  mostrarPolinomio(polinomio);
  // La siguiente linea resuelve para el X = 10
  int valor = 10 ;
  resultado = calcularPolinomio(polinomio, valor) ;
  cout<<"El resultado de f("<<valor<<") es
"<<resultado<<endl;
  return 0;
}
```

```
Void inicializarPolinomio(tVecPolinomio pol) {
   for (int i=0; i<_CANTTERMINOS;i++){
     pol[i].coeficiente = 0;
    pol[i].potencia = 0 ;
  }
}
```

```
long int fPot(int x, int p) {
    long int resul=1 ;
    for (int i=0; i<p; i++)
        resul = resul * x;
   return resul;
}
```

```
ingresar pares de coeficientes y potencias
4 4
3 3
2 2
1 1
0 0
4 * X (4) + 3 * X (3) + 2 * X (2) + 1 * X (1) + 0 * X (0)
+
El resultado de f(10) es 43210

--------------------------------
```

```
Void ingresarPolinimio(tVecPolinomio
pol){
    cout<<"ingresar pares de coeficientes y
potencias"<<endl;
 for (int i=0; i<_CANTTERMINOS;i++){
    cin>>pol[i].coeficiente ;
   cin>>pol[i].potencia  ;
 }
}
```

```
long int calcularPolinomio(tVecPolinomio pol,
int x) {
  long int resul = 0 ;
  for (int i=0; i<_CANTTERMINOS;i++)
      resul = resul + pol[i].coeficiente  * fPot(x,
pol[i].potencia) ;
 return resul;
}
```

```
 void  mostrarPolinomio(tVecPolinomio pol){
   for (int i=0; i<_CANTTERMINOS;i++)
     cout<<pol[i].coeficiente <<" * X
("<<pol[i].potencia<<") + " ;
     cout<<endl;
}
```

**Observar:**

- Se define una estructura con el par: coeficiente, potencia
- Se define un vector donde cada elemento está compuesto por un par coeficiente, potencia
- Se inicializa todo el vector (poner valores iniciales a todos los campos de todas las celdas)
- Se piden los pares de valores, los cuales se representan en cada celda.
- Se muestra el polinomio ingresado: se construye a partir de los pares coeficiente, potencia
- Se calcula el resultado del polinomio para un valor específico.
- Se usa la función fPot (calcula la potencia) para resolver el polinomio en el valor especificado.

**Tarea: modificar el  programa para que resuelva el polinomio para diferentes valores de X pedidos al usuario.**