Vectores

En las unidades anteriores hemos utilizado variables simples, esto es, espacios de memoria identificados con un nombre, que pueden almacenar sólo un valor al mismo tiempo. Estas estructuras de datos nos permitieron resolver los ejercicios de las guías anteriores, ya que es en las variables donde escribimos o leemos los datos que necesitamos procesar. Las variables simples seguirán siendo necesarias en nuestro trabajo, pero a medida que los ejercicios crezcan en complejidad, será necesario utilizar estructuras de datos más complejas también.

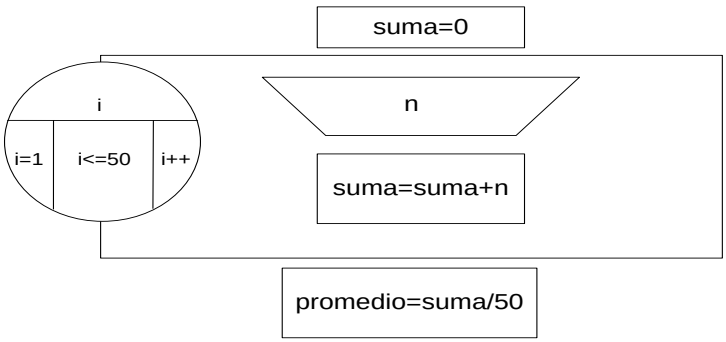
Analicemos como podríamos resolver el siguiente problema:

Dados un lista de 50 números que se ingresan por teclado, informar cuántos de los valores ingresados son mayores que el promedio.

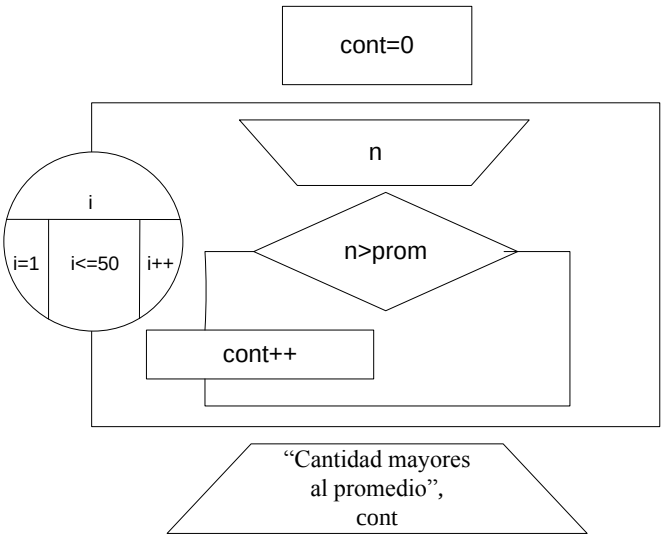
La resolución la podríamos dividir en dos acciones:

- Calcular el promedio
- Comparar cada valor con el promedio; si el valor es mayor sumar 1 a un contador previamente puesto en 0. Por último informar el valor del contador.

El diagrama de flujo para el cálculo del promedio es el siguiente



Para contar lo mayores al promedio



En la resolución del ejercicio se apela a un recurso que no es aceptable: se exige que el operador ingrese 2 veces los 50 valores. En ningún caso –por múltiples razones – se puede pedir al usuario que ingrese más de una vez los mismos valores, por lo que el programa del diagrama anterior es incorrecto, independientemente que su resultado sea el que corresponda.

Ahora bien, si esta forma de resolución no es correcta, la única alternativa que nos queda (con lo que sabemos hasta el momento) es almacenar los valores en 50 variables distintas, hacer 50 ingresos independientes (no podemos usar un ciclo exacto para el ingreso ya que en cada ingreso se debe usar una variable distinta), y 50 compararaciones también independientes antes de informar el resultado. Un programa de esas características sería absurdamente largo (¿y si en vez de 50 fueran 500 los valores a ingresar?), por lo que será necesario una estructura de datos distinta, una variable más adecuada para la resolución de este tipo de problemas, y esa estructura de datos se denomina **vector**, o array unidimensional. Veamos su definición:

Un vector es una estructura de datos que permite agrupar bajo un mismo nombre de variable un conjunto definido de datos de un mismo tipo, y acceder a cada elemento individual del conjunto–para leer o escribir- utilizando un subíndice entre corchetes, que representa la ubicación de ese elemento o componente individual dentro del vector.

Podríamos representar gráficamente un vector de nombre **v** de la siguiente manera:

10	v[0]
8	v[1]
66	v[2]
7	v[3]

20	v[4]
50	v[5]
40	v[6]
33	v[7]
15	v[8]
2	v[9]

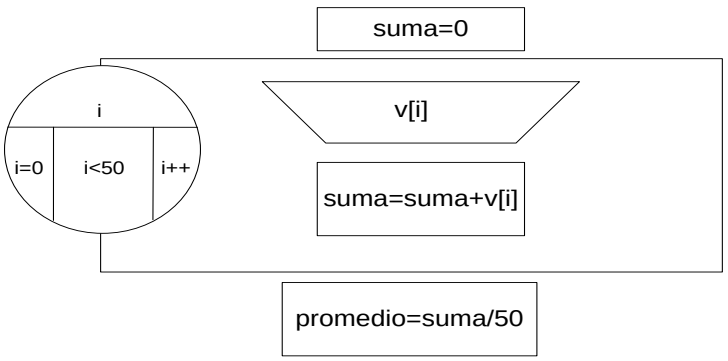
v es un vector que puede almacenar 10 números enteros (recordar que todos los valores deben ser de un mismo tipo de datos). Cada uno de sus elementos individuales o componentes se identificará con el nombre del vector, seguido de los corchetes y un número que identifica su posición dentro del vector. Nótese que la primera de todas esas posiciones se identifica con 0, y la úlltima con la cantidad definida -1.
Luego, podremos realizar las mismas acciones que hacíamos con las variables simples: cada elemento del vector puede pensarse como una variable de las que utilizabamos hasta ahora.

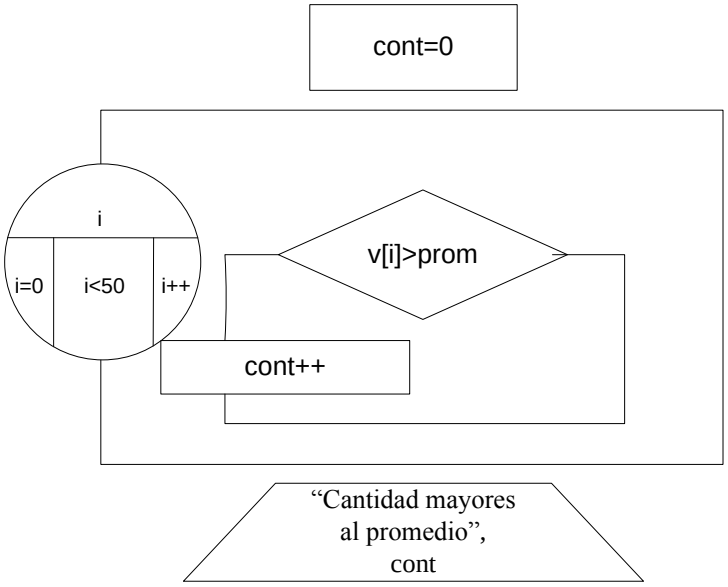
Por ejemplo, si queremos asignar el número 10 en la primer posición del vector:
v[0]=10

Si quisieramos sumar las dos últimas componentes:
suma=v[8]+v[9]

Esta nueva estructura de datos nos permite resolver problemas más complejos que los que utilizan sólo variables simples. La única limitación es que debemos conocer previamente la cantidad de valores a procesar, ya que la longitud de un vector es fija.

Veamos entonces como resolver el ejercicio anterior con vectores:





En el primer ciclo exacto se hizo el ingreso de datos directamente en cada una de las posiciones del vector, utilizando para eso la misma variable i que maneja el ciclo. Como i varía de 0 a 49 el primer valor se almacena en v[0], el segundo en v[1], y asi sucesivamente hasta el último en v[49]; luego se acumuló en la variable suma la sumatoria de lo que contiene cada una de las posiciones del vector.

Al finalizar el primer for, los 50 valores ingresados permanecen almacenados en el vector v, y seguirán allí hasta una nueva asignación o un nuevo ingreso (p.e. si se hace v[0]=10 el vector pierde su valor original para contener el 10).

Por esa razón y luego del cálculo del promedio, en el segundo ciclo exacto cada posición del vector puede compararse con el promedio para hallar los mayores. Nuevamente vuelve a utilizarse el valor de i para recorrer cada una de las posiciones del vector.

Los vectores se utilizan frecuentemente cuando es necesario manejar un conjunto determinado de valores de manera conjunta. Una vez ingresados los valores se pueden realizar múltiples operaciones sobre ese mismo conjunto de datos. Supongamos que sobre los 50 valores del ejercicio anterior quisieramos averiguar el valor máximo, o el mínimo, la diferencia entre los valores positivos y los negativos, etc., : sólo sería necesario recorrer con un for el vector para obtener los resultados.

El código de ejercicio es el siguiente:

```
# include<iostream>
# include<cstdlib>
# include<cstdio>
```

```
using namespace std;
int main(){
  int v[50];
  int i, suma, cont=0;
  float promedio;
  for(i=0;i<50;i++){
    cout<<"INGRESE UN NUMERO: ";
    cin>>v[i];
    suma=suma+v[i];
    }
  promedio=(float)suma/50;
  for(i=0;i<50;i++){
    if(v[i]>promedio){
      cont++;
      }
    }
  cout<<"CANTIDAD DE VALORES MAYORES AL PROMEDIO: "<<cont<<endl;
  system("pause");
  return 0;
}
```

Otro ejemplo clásico del trabajo con vectores, puede ser el siguiente:

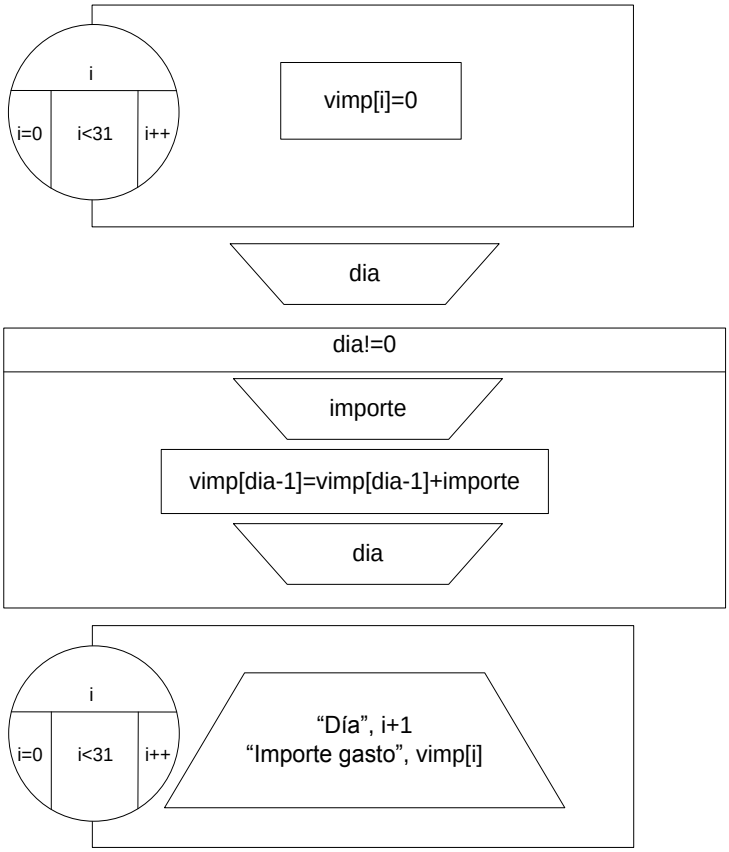
Se dispone de un lote de registros con los gastos realizados por una persona durante un mes, en el que se indica el día del mes y el importe del gasto. El lote termina con un número de día igual a 0. Puede haber más de un registro para cada día. Se pide que se calcule e informe el gasto total diario.

Para resolver el problema debemos analizar cada registro ingresado, y sumar al día correspondiente el importe del gasto. De esta manera tendremos, al finalizar el ingreso de datos, que informar 31 valores: cada uno de ellos representará el gasto de uno de los días del mes en particular.

Nuevamente se nos presenta el problema de tener que trabajar con demasiadas variables individuales: por cada registro de día e importe ingresado, deberíamos hacer 30 comparaciones distintas para decidir en cual de las variables acumular el importe.

Para evitar esta complicación, podemos reemplazar las 31 variables simples por un vector de 31 posiciones (1 para cada día); luego cada importe deberá sumarse al importe existente para ese día. En este caso utilizaremos como subíndice del vector el número de día-1, con lo que en la posición 0 del vector se acumularán todas los importes del día 1, en la posición 1 todos los importes del día 2, etc.

Previamente deben ponerse en 0 todas las posiciones del vector, y por último mostrar su contenido completo. El diagrama de flujo del programa es el siguiente:



En el primer ciclo exacto se asigna un 0 en cada una de las posiciones del vector.

Luego se ingresa el primer valor para la variable dia, se chequea que sea distinto de 0, y se ingresa al ciclo inexacto. En él se ingresa el valor correspondiente a un importe, y luego se acumula ese valor al existente en la posición correspondiente al día-1. Por ejemplo:

Si se ingresa dia 7e importe 100, la asignación sería
vimp[7-1]=vimp[7-1]+importe

como en el primer ingreso todas las posiciones del vector contienen un 0, y el valor del importe es 100 nos queda:

vimp[6]=0+100

Si se repitiera el valor de día 7, el importe ingresado se sumaría a los 100 existentes en vimp[6] .

El programa continúa hasta que se ingresa un valor de día igual a 0. En este caso se corta el ciclo inexacto, y se ejecuta el segundo ciclo exacto.

En el segundo ciclo exacto se muestra el contenido de cada una de las posiciones del vector, utilizando para eso como subíndice la variable que maneja el for.

```
# include<iostream>
# include<cstdlib>
# include<cstdio>

using namespace std;
int main(){
  int i, dia;
  float vimp[31], importe;
  for(i=0;i<31;i++){
    vimp[i]=0;
    }
  cout<<"INGRESE EL DIA: ";
  cin>>dia;
  while(dia!=0){
    cout<<"INGRESE EL IMPORTE: ";
    cin>>importe;
    vimp[dia-1]=vimp[dia-1]+importe;
    cout<<"INGRESE EL DIA: ";
    cin>>dia;
    }
  for(i=0;i<31;i++){
    cout<<"DIA: "<<i+1<<endl;
    cout<<"IMPORTE GASTO: "<<vimp[i]<<endl;
    }
  system("pause");
  return 0;
}
```