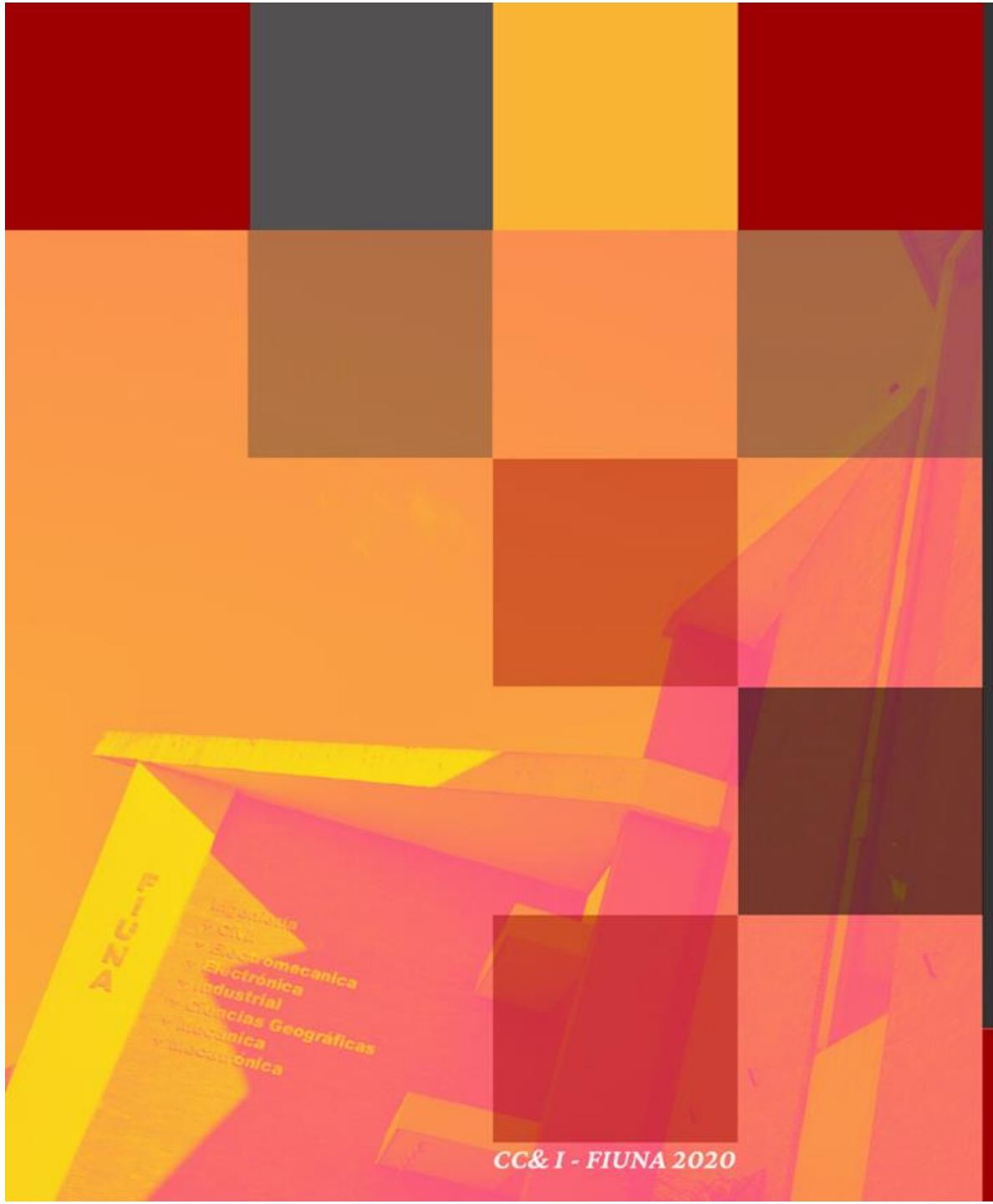

UNIVERSIDAD NACIONAL DE ASUNCIÓN
FACULTAD DE INGENIERÍA

COMPUTACIÓN

Vectores
(Arreglos Unidimensionales)

Aux. Ing. Cristhian Colbes

2020

ARREGLOS UNIDIMENSIONALES

Resolución “paso a paso” de Ejercicios

En los ejercicios que veremos a continuación, se proponen situaciones en los enunciados, y mostraremos la resolución paso a paso, si bien como ya en este punto sabemos, los caminos para llegar a cumplir lo que se propone pueden ser varios.

Debemos enfocarnos en comprender el objetivo de cada porción de código, y cómo es que contribuye a lograr la resolución final.

Mostraremos 2 (dos) ejercicios con características distintas, lo más detalladamente posible planteando todas las cuestiones que puedan ser relevantes, para ilustrar el uso y capacidades de los arreglos unidimensionales, que habitualmente conocemos como “vectores”.

ARREGLOS UNIDIMENSIONALES

Ejercicio 1)

Elaborar un programa en C++ que permita leer un vector de n componentes de tipo flotante (siendo "n" un número entero y positivo), y que cree una función auxiliar para transformar dicho vector en otro cuyas componentes sean el doble de cada componente del inicial. Imprimir el vector antes y después del cambio.

Resolución

Primeramente, debemos declarar siempre los archivos de cabecera que nos permitan declarar el flujo de entrada y salida de variables, además de cada función auxiliar que pensemos sea necesaria.

Ahora bien, aparte de una de las funciones lógicas a crear para no tener que utilizar una compleja rutina de impresión (con un vector o arreglo se trabaja por lo general “componente a componente”, no olvidar esto) la cual llamaremos “vimprimir”, tal vez conviene fijarnos en el pedido del enunciado.

ARREGLOS UNIDIMENSIONALES

Pero vamos paso a paso:

```
#include<iostream>
using namespace std;
void vimprimir(float x[], int n){
        int i;
        for (i=0;i<n;i++) cout<<x[i]<<"\t";
        cout<<endl;
}
```

Declaraciones previas

Definición de la función

Podríamos haber *prototipado* primero la función (o sea, apenas declarar el nombre, tipo y argumentos) y definirla después de creada la función **main**, pero en estos casos por practicidad conviene definirlas de antemano para evitar líneas de código, pero es una elección libre.

Tener en cuenta que el ejercicio plantea un vector de entrada de tipo flotante, con una dimensión genérica "n", así que lo que hace la función es recorrer con un índice (representado por "i") todas las posiciones del vector de entrada, e imprimirlas en pantalla "al momento". NO devuelve valor, por lo que se prescinde de la sentencia de "return". Por ello mismo, también se define como tipo "void".

ARREGLOS UNIDIMENSIONALES

```
void vimprimir(float x[], int n){
        int i;
        for (i=0;i<n;i++) cout<<x[i]<<"\t";
        cout<<endl;
}
```

Se imprime cada valor de la posición de x[i], recorriendo todas las posiciones lógicamente se imprime todo el vector.

En este caso, el vector en sí viene a la función en el primer argumento, en el cual se declara un vector genérico "x[]"; mediante esta función en particular se permite pasar como argumentos tanto el nombre del vector así como su dimensión. Por ejemplo podemos pasar como argumentos un vector dado "V" y su dimensión ya sea genérica ("n") o definida (un número, digamos "5"), y la función trabajará con ellos para empezar un recorrido en la primera posición del vector, e ir subiendo con un contador las posiciones indicadas por el índice i para las acciones necesarias.

Pero no debemos preocuparnos aún por entender detalladamente esto, sino debemos ocuparnos en saber que así podemos pasar un vector entero y modificarlo según sea necesario, *modificando los valores del vector en sí*, como veremos.

Arreglos Unidimensionales

Ahora bien, todo lo que hicimos hasta aquí es la parte de “rutina”, no hay nada en sí, aparte de la previsión de que el vector sea de componentes flotantes, que sea particular y único de este problema.

Lo interesante es que siempre un problema se puede resolver por partes; la parte realmente trascendente de un problema suele ser la más “corta” (ojo que no implica menos líneas de código necesariamente) pero sí la que más pienso suele llevar para llegar a bosquejar su solución.

En este caso debemos plantearnos: cómo crear una función auxiliar para transformar el vector en otro cuyas componentes sean el doble de cada componente del inicial? Y en el proceso imprimir el vector original así como el vector modificado después del cambio.

Ojo, se pide “transformar”, no crear otro vector desde cero, duplicarlo y luego trabajar, etc. Cómo lo hacemos?

Arreglos Unidimensionales

De nuevo, así como previmos en el caso de la impresión del vector, sabemos que para modificar un vector hay que trabajar componente a componente, y si pasamos el vector como argumento, similar de nuevo al algoritmo de impresión, podríamos cambiar cada componente con el doble de la inicial!

Definición de la función que llamamos “vdoble”:

```
void vdoble(float x[], int n){
        int i;
        for (i=0;i<n;i++) x[i]=2*x[i];
}
```

Se recorre con un índice cada posición del vector genérico recibido como argumento, y se cambia por el doble de esa misma componente!

Luego de este proceso, el vector que se recibió YA NO ES EL MISMO. Fue reemplazado según esta función por el doble de cada componente del vector inicial.

Esta lógica se comprenderá mejor más adelante en la materia, en la clase de Punteros; por ahora utilizar el hecho de que en esta forma se puede pasar un vector a una función y modificarlo, es clave pasar el nombre del vector y su dimensión.

ARREGLOS UNIDIMENSIONALES

Por tanto ya tenemos una idea de cómo plantear una función **main**, que utilizando las dos funciones creadas, pueda cumplir con lo solicitado por el problema!

La consideración a tener en cuenta es que debemos imprimir el vector original con la función de impresión “vimprimir” antes de cambiarlo, luego utilizar la función de duplicación “vdoble”, y como se debería comprender en este punto, el vector YA cambió, por lo que al aplicar la función de impresión va a imprimir esta vez el vector ya modificado.

Una función main, con sus declaraciones y lectura de datos que cumpla con lo solicitado y analizado, la vemos en la siguiente diapositiva.

ARREGLOS UNIDIMENSIONALES

```
int main() {
        int i,n;
        cout<<"\nIngrese el numero de componentes del vector:\n";
        cin>>n;
        float a[n];
        for (i=0;i<n;i++){
                cout<<"\nIngrese componente de la posicion "<<i<<" del vector: ";
                cin>>a[i];
        }
        cout<<"\nEl vector original es: \n";
        vimprimir(a,n);
        vdoble(a,n);
        cout<<"\nEl vector modificado con el doble de cada componente es: \n";
        vimprimir(a,n);
        return 0;
}
```

Carga de dimensión y declaración del vector

Carga del vector

Impresión del vector original, se pasa el nombre del vector como argumento junto a su dimensión

Modificación de componentes del vector, fijarse que es el mismo nombre del vector

Impresión del vector modificado, sigue con el mismo nombre

UNIVERSIDAD NACIONAL DE ASUNCIÓN
FACULTAD DE INGENIERÍA

ARREGLOS UNIDIMENSIONALES

```cpp
//Poner como parámetros de una función a un vector y su dimensión
/*Elaborar un programa en C++ que permita leer un vector de n componentes de tipo flotante (siendo "n" un número entero y positivo),
y que cree una función auxiliar para transformar dicho vector en otro cuyas componentes sean el doble de cada componente del inicial.
Imprimir el vector antes y después del cambio*/
#include<iostream>
using namespace std;
void vimprimir(float x[], int n){
    int i;
    for (i=0;i<n;i++) cout<<x[i]<<"\t";
    cout<<endl;
}
void vdoble(float x[], int n){
    int i;
    for (i=0;i<n;i++) x[i]=2*x[i];
}
int main(){
    int i,n;
    cout<<"\nIngrese el numero de componentes del vector:\n";
    cin>>n;
    float a[n];
    for (i=0;i<n;i++){
        cout<<"\nIngrese componente de la posicion "<<i<<" del vector: ";
        cin>>a[i];
    }
    cout<<"\nEl vector original es: \n";
    vimprimir(a,n);
    vdoble(a,n);
    cout<<"\nEl vector modificado con el doble de cada componente es: \n";
    vimprimir(a,n);
    return 0;
}
```

Arreglos Unidimensionales

Un ejemplo de programa ejecutado con el algoritmo y código compilado que planteamos se ve así:

```
Ingrese el numero de componentes del vector:
4

Ingrese componente de la posicion 0 del vector: 1

Ingrese componente de la posicion 1 del vector: -2.2

Ingrese componente de la posicion 2 del vector: 3.45

Ingrese componente de la posicion 3 del vector: -4

El vector original es:
1       -2.2    3.45    -4

El vector modificado con el doble de cada componente es:
2       -4.4    6.9     -8

--------------------------------
Process exited after 95.01 seconds with return value 0
Presione una tecla para continuar . . .
```

ARREGLOS UNIDIMENSIONALES

*“Por qué no hice una función *aparte* para la carga del vector también?”*

“No podría haber aprovechado la función de creación del vector modificado para imprimir ya ahí en el mismo proceso ese vector nuevo?”

“Por qué no validé el valor de la dimensión n, restringiendo el ingreso de enteros positivos únicamente?”

.

.

.

Quedan muchas preguntas, pero es saludable que así sea, porque nos estamos planteando diversas formas de solución! El que las llevemos a cabo depende de si es más práctico para el problema en cuestión, o si me puede servir para generalizar porciones de código con miras a uso futuro, etc.

En este caso esta fue ***una*** solución nada más del problema, que cumple con lo solicitado ☺

ARREGLOS UNIDIMENSIONALES

Ejercicio 2)

Elaborar un programa en C++ que permita leer un vector de n componentes de tipo entero (siendo "n" un número entero y positivo), y que realice lo siguiente: encuentre e imprima la cantidad y el promedio de las "componentes" del vector que sean pares, así como que encuentre e imprima el promedio de las componentes de las posiciones "impares" del vector. Imprimir además el vector original.

Resolución

Primeramente de nuevo, debemos declarar siempre los archivos de cabecera que nos permitan declarar el flujo de entrada y salida de variables, además de cada función auxiliar que pensemos sea necesaria.

En este caso a nivel de funciones solamente necesitaríamos la función básica de impresión (“vimprimir”) de vectores que creamos antes, o bien también si quisiéramos una para carga de datos.

ARREGLOS UNIDIMENSIONALES

Pero vamos paso a paso:

```
#include<iostream>
using namespace std;
void vimprimir(int x[], int n){
        int i;
        for (i=0;i<n;i++) cout<<x[i]<<"\t";
        cout<<endl;
}
```

Declaraciones previas

Definición de la función de impresión

Tener en cuenta que en este caso el ejercicio plantea un vector de entrada de tipo **entero**, con una dimensión genérica “n”.

Así como dijimos en el ejercicio anterior, nos falta la parte central que nos resuelva el problema principal, aunque en este caso tenemos “dos” problemas separados:

*1) Cómo puedo encontrar las componentes que sean pares, contarlas y promediarlas?
*2) Cómo puedo promediar las componentes de las *posiciones* impares?

Arreglos Unidimensionales

*1) Cómo puedo encontrar las componentes que sean pares, contarlas y promediarlas?

Todas las componentes que sean pares cumplen una simple condición: su resto de división entre 2 debe ser nulo. Si podemos expresar esto en código, podremos sumar todos los elementos que sean apropiados en una variable auxiliar llamada comúnmente “sumador” o acumulador, que va llevando cuenta de los valores para después promediarlos.

Por cierto, también necesitaremos la cantidad de elementos que cumplen el criterio para hallar ese promedio, para lo cual usaremos otra variable auxiliar que generalmente se denomina en programación como “contador” ya que cuenta de a uno a aquellos que van cumpliendo cierta condición.

Ambas variables auxiliares deben inicializarse, para evitar cualquier tipo de ambigüedad en la porción de memoria asignada o datos basura en ellas: en este caso, como ambos “suman” (los *sumadores* suman los componentes en sí, los *contadores* suman una unidad cada vez que se cumple la condición) pueden inicializarse a “0”.

ARREGLOS UNIDIMENSIONALES

Una porción de código que cumple dicho criterio (inicialmente, luego le haremos un par de mejoras) sería:

```
for (i=0;i<n;i++) {
        if (a[i]%2==0){
                cant1+=1;
                suma1+=a[i];
        }
}
prom1=suma1/cant1;
```

Siendo a[n] un vector de “n” componentes con el que vayamos trabajando en el programa **main**, y al contador llamamos “cant1” y al sumador “suma1” (recordar que al definirlos antes deberían estar inicializados a “0”, tal como veremos al juntar todo el código”).

Lógicamente, al final del proceso de recorrido del vector recién podremos calcular el promedio de elementos que cumplen la condición (ser pares) si los hubiere.

ARREGLOS UNIDIMENSIONALES

Podemos ahora mejorar este código con una consideración: Qué pasaría en el caso de que NO hubiera elementos que sean pares? En este caso, no habría promedio a considerar, y tendríamos una división indeterminada en la línea de programación de promedio (suma elementos:0, dividido entre cantidad de elementos pares: 0) por lo cual agregamos una bandera que indique si hay elementos (ejemplo: ban!=0), entonces proceder con esta división; si no, indicar que no hay elementos pares.

```
int ban=0;
for (i=0;i<n;i++) {
        if (a[i]%2==0){
                cant1+=1;
                suma1+=a[i];
                ban=1;
        }
}
```

Veremos como imprimir con esta lógica.

Arreglos Unidimensionales

Con este agregado a la impresión, solucionamos definitivamente el problema.

```
if (ban!=0) {
        prom1=suma1/cant1;
        cout<<"\nLa cantidad de componentes pares es "<<cant1<<", y su promedio respectivo es
"<<prom1<<".\n";
}
else cout<<"\nNo hay componentes pares en el vector\n";
```

Ahora bien, queda una cuestión más a resolver: cómo hacer que una operación cuyo resultado y naturaleza son de coma flotante pero que involucra variables de tipo entero (como el promedio -flotante- entre las componentes del vector -enteras-) muestre adecuadamente los dígitos decimales que pueden resultar de la operación?

ARREGLOS UNIDIMENSIONALES

Si pudiéramos probar el código en este punto, encontraríamos que si un vector ejemplo tuviera de componentes a: { 66, 20, 2, 1 }

```
Ingrese el numero de componentes del vector:
4

Ingrese componente de la posicion 0 del vector: 66

Ingrese componente de la posicion 1 del vector: 20

Ingrese componente de la posicion 2 del vector: 2

Ingrese componente de la posicion 3 del vector: 1

El vector ingresado es:
66      20      2       1

La cantidad de componentes pares es 3, y su promedio respectivo es 29.
```

Vemos que el promedio de las componentes pares (66+10+2=88, dividido por 3) debería ser 29.33333…., pero el promedio nos muestra apenas 29, porque si bien la variable prom1 está definida como float, el vector en sí está definido como int.

Arreglos Unidimensionales

Esto se soluciona echando mano de la conversión puntual de tipo de datos (cast), que nos sirven para convertir una variable de un tipo a otro, para una operación dada, como ya se vio anteriormente en la materia. Con hacer conversión de apenas una de las variables involucradas en la operación de promedio, bastará.

Con esto el código quedará:

```
if (ban!=0) {
        prom1=(float)suma1/cant1;
        cout<<"\nLa cantidad de componentes pares es "<<cant1<<", y su promedio respectivo es "<<prom1<<".\n";
}
else cout<<"\nNo hay componentes pares en el vector.\n";
```

En este caso el promedio ya mostrará como 29.3333 para los mismos componentes { 66, 20, 2, 1 }.

ARREGLOS UNIDIMENSIONALES

*2) Cómo puedo promediar las componentes de las *posiciones* impares?

En este caso ni siquiera deberemos discriminar idealmente ninguna condición dentro del recorrido del vector (en realidad se puede, pero usaremos el camino de manejarnos con el incremento de la variable de control del ciclo repetitivo)

Las posiciones pares estarán dadas por el índice (en este caso usamos “i”) empezando en la primera posición impar: la posición “1” (y NO en “cero”, que es par), y los incrementos, en vez de ser unitarios, pasarán a ser de 2 en 2, lo cual asegura que recorra solamente y todas las posiciones impares. De nuevo, estos elementos deberemos guardarlos en otro sumador, la cantidad podemos contarla también en otro contador (hay otros métodos como por ejemplo calcular la cantidad de posiciones pares o impares dependiendo de si la dimensión del vector en sí es par o impar, pero no hace falta complicarse innecesariamente).

De nuevo, ambas variables auxiliares deben inicializarse, para evitar cualquier tipo de ambigüedad o datos basura en ellos: en este caso, como ambos “suman” (los *sumadores* suman los componentes en sí, los *contadores* suman una unidad cada vez que se cumple la condición) pueden inicializarse a “0”.

ARREGLOS UNIDIMENSIONALES

Una porción de código que cumple dichos criterios (y tiene en cuenta la conversión de variables ya antes analizada para los promedios) es:

```
for (i=1;i<n;i+=2) {
                    cant2+=1;
                    suma2+=a[i];
}
prom2=(float)suma2/cant2;
```

Fijarse que en la condición de incremento, vamos pasando de 2 en 2, para asegurarnos recorrer desde el 1 a todas las posiciones impares.

Lógicamente, otra vez al final del proceso de recorrido del vector recién podremos calcular el promedio de elementos que cumplen la condición (los elementos de las posiciones impares).

Arreglos Unidimensionales

Similarmente al caso anterior, cabe preguntarnos: Qué pasaría en el caso de que NO hubiera posiciones impares en el vector? En este caso, hay una sola posibilidad, de hecho: que el vector tenga 1 solo elemento (posición “0”). Si se cae en este caso, de nuevo tendríamos que considerar la posibilidad de una división indeterminada.)suma elementos:0, dividido entre cantidad de elementos de las posiciones impares: 0) pero ya no hace falta una bandera auxiliar: básicamente, para la impresión y el cálculo si la dimensión “n” es mayor a 1, se puede calcular este promedio, y si no, se menciona que no hay posiciones impares y queda solucionado el asunto.

```
if (n>1) {
        prom2=(float)suma2/cant2;
        cout<<"\nEl promedio de componentes de las posiciones impares del vector es "<<prom2<<".\n";
}
else cout<<"\nNo hay posiciones impares en el vector.\n";
```

Arreglos Unidimensionales

Similarmente al caso anterior, cabe preguntarnos: Qué pasaría en el caso de que NO hubiera posiciones impares en el vector? En este caso, hay una sola posibilidad, de hecho: que el vector tenga 1 solo elemento (posición “0”). Si se cae en este caso, de nuevo tendríamos que considerar la posibilidad de una división indeterminada.)suma elementos:0, dividido entre cantidad de elementos de las posiciones impares: 0) pero ya no hace falta una bandera auxiliar: básicamente, para la impresión y el cálculo si la dimensión “n” es mayor a 1, se puede calcular este promedio, y si no, se menciona que no hay posiciones impares y queda solucionado el asunto.

```
if (n>1) {
        prom2=(float)suma2/cant2;
        cout<<"\nEl promedio de componentes de las posiciones impares del vector es "<<prom2<<".\n";
}
else cout<<"\nNo hay posiciones impares en el vector.\n";
```

ARREGLOS UNIDIMENSIONALES

Un ejemplo que cumple ya con todas las previsiones es:

```
Ingrese el numero de componentes del vector:
1

Ingrese componente de la posicion 0 del vector: 3

El vector ingresado es:
3

No hay componentes pares en el vector.

No hay posiciones impares en el vector.
```

Por tanto ya tenemos una idea de cómo plantear una función **main**, que utilizando los elementos y porciones de código planteados pueda llegar a la solución del problema.

Una función main, con sus declaraciones y lectura de datos que cumpla con lo solicitado y analizado, la vemos en la siguiente diapositiva.

ARREGLOS UNIDIMENSIONALES

```cpp
//Uso de contador y acumulador dentro de un arreglo, así como del manejo del índice incremental/decremental para recorrido del vector
/*Elaborar un programa en C++ que permita leer un vector de n componentes de tipo entero (siendo "n" un número entero y positivo), y
que realice lo siguiente: encuentre e imprima la cantidad y el promedio de las "componentes" del vector que sean pares, así como que
encuentre e imprima el promedio de las componentes de las posiciones "impares" del vector. Imprimir además el vector original*/
#include<iostream>
using namespace std;
void vimprimir(int x[], int n){
    int i;
    for (i=0;i<n;i++) cout<<x[i]<<"\t";
    cout<<endl;
}
int main (){
    int i,n,cant1=0,suma1=0,cant2=0,suma2=0, ban=0;
    float prom1,prom2;
    cout<<"\nIngrese el numero de componentes del vector:\n";
    cin>>n;
    int a[n];
    for (i=0;i<n;i++){
        cout<<"\nIngrese componente de la posicion "<<i<<" del vector: ";
        cin>>a[i];
    }
    for (i=0;i<n;i++) {
        if (a[i]%2==0){
            cant1+=1;
            suma1+=a[i];
            ban=1;
        }
    }
    for (i=1;i<n;i+=2) {
        cant2+=1;
        suma2+=a[i];
    }
    cout<<"\nEl vector ingresado es: \n";
    vimprimir(a,n);
    if (ban!=0) {
    prom1=(float)suma1/cant1;
    cout<<"\nLa cantidad de componentes pares es "<<cant1<<", y su promedio respectivo es "<<prom1<<".\n";
    }
    else cout<<"\nNo hay componentes pares en el vector.\n";
    if (n>1) {
    prom2=(float)suma2/cant2;
    cout<<"\nEl promedio de componentes de las posiciones impares del vector es "<<prom2<<".\n";
    }
    else cout<<"\nNo hay posiciones impares en el vector.\n";
    return 0;
}
```

ARREGLOS UNIDIMENSIONALES

Un ejemplo de programa ejecutado con el algoritmo y código compilado que planteamos se ve así:

```
Ingrese el numero de componentes del vector:
4

Ingrese componente de la posicion 0 del vector: 66

Ingrese componente de la posicion 1 del vector: 20

Ingrese componente de la posicion 2 del vector: 2

Ingrese componente de la posicion 3 del vector: 1

El vector ingresado es:
66      20      2       1

La cantidad de componentes pares es 3, y su promedio respectivo es 29.3333.

El promedio de componentes de las posiciones impares del vector es 10.5.

--------------------------------
Process exited after 6.593 seconds with return value 0
Presione una tecla para continuar . . .
```

Arreglos Unidimensionales

De nuevo quedan muchas preguntas, como dijimos, hay elecciones que se toman pero por didáctica algunas cosas las hacemos así.

Por ejemplo (hay muchas cuestiones más!) se podría plantear de distinta forma:

"No podría haber aprovechado la carga del vector para calcular ahí mismo los elementos que eran pares y contarlos y sumarlos?"

"Y en esa misma carga del vector, no podría haber validado alguna condición del índice para encontrar las posiciones impares ya ahí mismo y contar dichos elementos y sumarlos?"

"Por qué defino dos variables para contadores, dos para sumadores y dos para promedios? Si es que utilizo un conjunto sumador-contador por vez y luego un promedio, puedo reutilizar las mismas variables en otra parte de código, después ya claro de imprimirlas y sacar el valor útil digamos?"

De nuevo, este código presentado fue ***una*** solución nada más del problema, que cumple con lo solicitado ☺

(FIN)
☺

Gracias por la atención!