

AyED: Arrays en C++

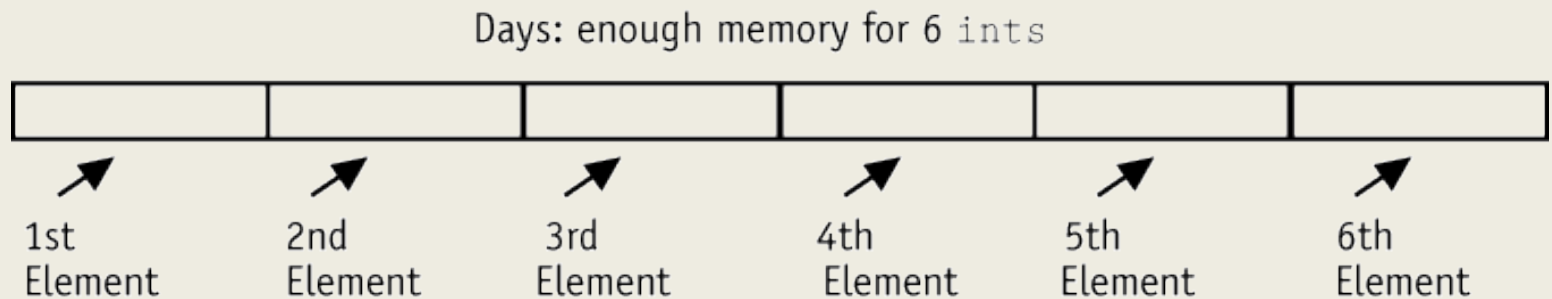
Ing. Pablo Méndez

Los arrays permiten guardar múltiples valores

A diferencia de las variables simples o atómicas que guardan un único valor, los arrays pueden guardar muchos valores de un mismo tipo.

<code>int cont</code>	<div>12345</div>
<code>float temp</code>	<div>56.981</div>
<code>char c</code>	<div>A</div>

Un array con seis elementos



Declaración de arrays

Array Declaration	Number of Elements	Size of Each Element	Size of the Array
char letters[25];	25	1 byte	25 bytes
short rings[100];	100	2 bytes	200 bytes
int miles[84];	84	4 bytes	336 bytes
float temp[12];	12	4 bytes	48 bytes
double Distance[1000];	1000	8 bytes	8000 bytes

Acceso a través del índice de un array

Se puede acceder a un elemento individual y tratarlo como una variable atómica. Para ello se indexa el array en la posición que se desea acceder.

Ejemplo de programa

```
int main()
{
    int temp[7];
    cout << "Ingrese la temp del lunes:" << endl;
    cin >> temp[0];
    cout << "Ingrese la temp del martes:" << endl;
    cin >> temp[1];
    cout << "Ingrese la temp del miércoles:" << endl;
    cin >> temp[2];
    cout << "Ingrese la temp del jueves:" << endl;
    cin >> temp[3];
    cout << "Ingrese la temp del viernes:" << endl;
    cin >> temp[4];
    cout << "Ingrese la temp del sábado" << endl;
    cin >> temp[5];
    cout << "Ingrese la temp del domingo:" << endl;
    cin >> temp[6];
    //Se listan todas las temperaturas
    cout << "Las temperaturas fueron:" << endl;
    for (int i = 0; i < 7; i++)
    {
        cout << temp[i] << endl;
    }
    return 0;
}
```

Verificación de límites

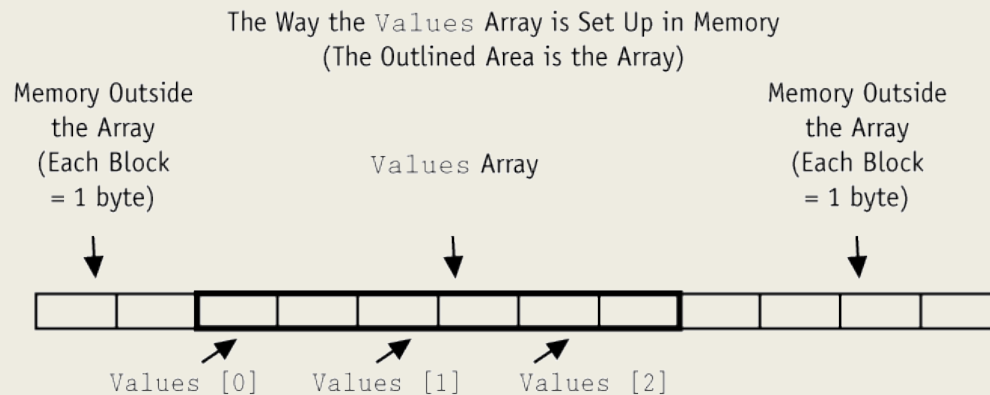
- C++ deja al usuario el manejo libre de memoria. Tener especial cuidado, ya que podemos pasarnos de los límites del array. El programa compilará, pero es probable que rompa en tiempo de ejecución. Que no rompa, no asegura tampoco que los valores hayan sido almacenados.

Ejemplo

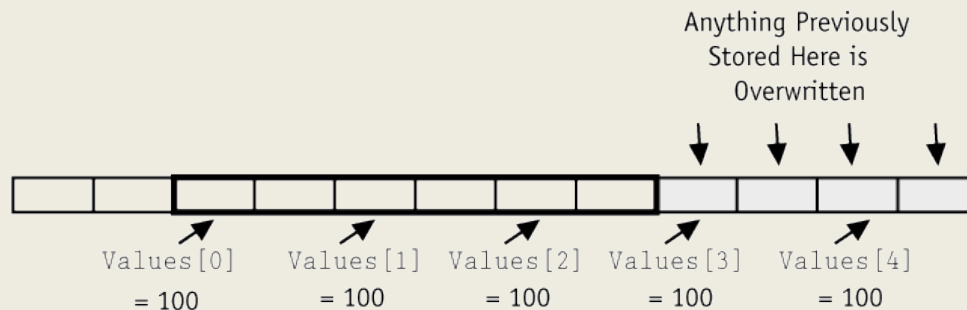
```
#include <iostream>
/* Este es un ejemplo que se excede de los límites del array
CUIDADO: Si compila y ejecuta este programa, puede romper en
tiempo de ejecución */
using namespace std;
int main()
{
    short values[3];    // Un array de 3 short integers.

    cout << "Almacenaremos 5 valores en un array de 3!" << endl;
    for (int count = 0; count < 5; count++)
        values[count] = 100;
    cout << "Si se lee esto, no rompió" << endl;
    for (int count = 0; count < 5; count++)
        cout << values[count] << endl;
    return 0;
}
```


Almacenamiento en memoria



How the Number Assigned to the Elements Overflow the Array's Boundaries
(The Shaded Area is the Section of Memory Written To)



Los arrays se almacenan en posiciones de memoria contiguas

Inicialización

Como todo tipo de variable, contienen basura al momento de su creación, y deben inicializarse.

Pueden inicializarse:

- Luego de su declaración. Para hacer esto hay que acceder a todas sus posiciones una por una.
- En la declaración:
 - Inicialización total.
 - Inicialización parcial.
 - Inicialización total con declaración implícita del tamaño.

Inicialización luego de la declaración

```
#include <iostream.h>

void main(void)
{
    int dias[12];
    dias[0] = 31; // Enero
    dias[1] = 28; // Febrero
    dias[2] = 31; // Marzo
    dias[3] = 30; // Abril
    dias[4] = 31; // Mayo
    dias[5] = 30; // Junio
    dias[6] = 31; // Julio
    dias[7] = 31; // Agosto
    dias[8] = 30; // Septiembre
    dias[9] = 31; // Octubre
    dias[10] = 30; // Noviembre
    dias[11] = 31; // Diciembre
    for (int cont = 0; cont < 12; cont++)
    {
        cout << "El mes " << (cont + 1) << " tiene ";
        cout << dias[cont] << " dias.\n";
    }
}
```

Inicialización total en la declaración

```
#include <iostream.h>

using namespace std;
int main()
{
    int dias[12] = {31, 28, 31, 30,
                    31, 30, 31, 31,
                    30, 31, 30, 31};
    for (int cont = 0; cont < 12; cont++)
    {
        cout << "El mes " << (cont + 1) << " tiene " <<
        dias[cont] << " dias." << endl;
    }
    return 0;
}
```

Otro ejemplo

/* Este programa toma las primeras 10 letras del alfabeto, y devuelve el código ascii de cada una */

```
int main()
{
    char letras[10] = {'A', 'B', 'C', 'D', 'E',
                       'F', 'G', 'H', 'I', 'J'};
    cout << "Caracter" << "\t" << "Cod. ASCII \n";
    cout << "-----" << "\t" << "-----\n";
    for (int cont = 0; cont < 10; cont++)
    {
        cout << letras[cont] << "\t\t" << int(letras[cont]) <<
endl;
    }
    return 0;
}
```

Salida del programa

Character	ASCII Code
-----	-----
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74

Inicialización parcial en la declaración

En C++, a diferencia de otros lenguajes, no hace falta inicializar todas las posiciones del array si se opta por inicializar en la declaración

```
int sietenums[7] = {1, 1, 1, 1};
```

Inicialización parcial - Ejemplo

```
#include <iostream.h>
using namespace std;

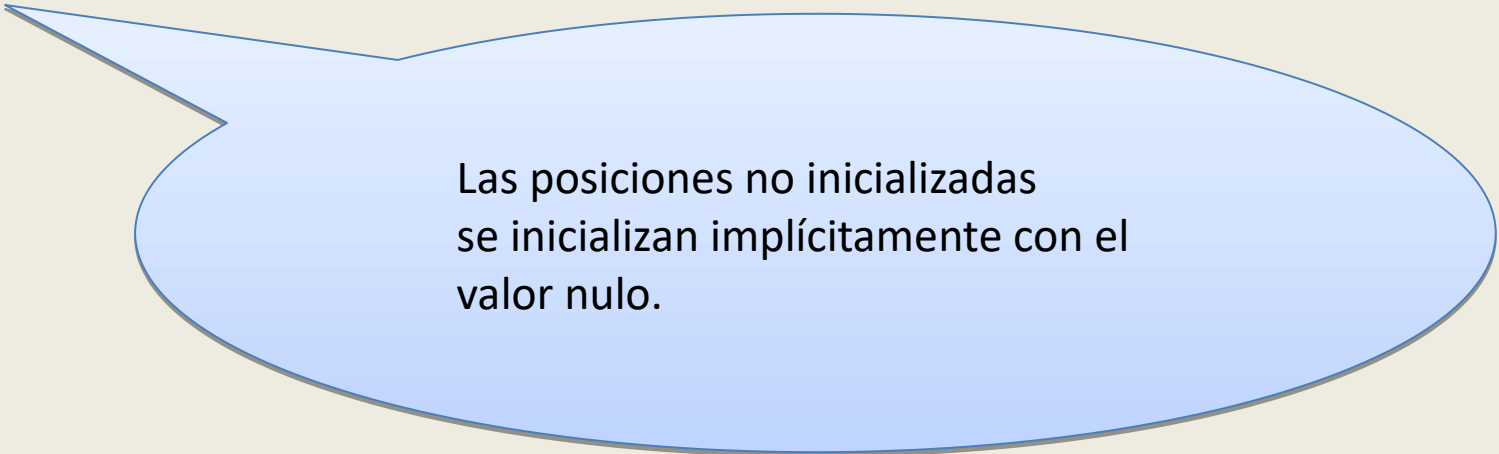
int main()
{
    int sietenums[7] = {1, 1, 1, 1};

    cout << "Contenido del array:" << endl;
    for (int index = 0; index < 7; index++)
        cout << sietenums[index] << endl;
    return 0;
}
```


Salida del programa

Contenido del array:

1
1
1
0
0
0



Las posiciones no inicializadas
se inicializan implícitamente con el
valor nulo.

Declaración implícita del tamaño del array

Es posible declarar el array sin especificar su tamaño, siempre y cuando se declare la cantidad de elementos en la lista de inicialización (sólo sirve para inicialización total en la declaración).

```
float ratings[] = {1.0, 1.5, 2.0, 2.5, 3.0};
```

Arrays de caracteres

Para el manejo de texto hay dos herramientas:

- Variable de tipo string (c++): Facilitan mucho el manejo de cadena de caracteres, pero tienen limitaciones cuando se tienen que usar en archivos.
- Array de caracteres:

```
char name[] = "Warren";
```

Arrays de caracteres - Almacenamiento

```
char Name [7] = "Warren";
```

Null Terminator



'W'	'a'	'r'	'r'	'e'	'n'	'\0'
Name [0]	Name [1]	Name [2]	Name [3]	Name [4]	Name [5]	Name [6]

Ejemplo – Array de caracteres

```
using namespace std;
int main()
{
    char saludo[] = "Hola";
    char nombre[] = {'J', 'i', 'm', 'e', 'n', 'a', '\0'};
    //Notar que se puede hacer un cout de todo el vector
    //sin necesidad de acceder índice por índice
    cout << saludo << endl;
    cout << nombre << endl;
    return 0;
}
```

Procesando contenido de arrays

- Los elementos de un array son procesados como cualquiera otra variable de su tipo, pero debe indexarse el array para hacerlo.
- Ejemplo: Realice un programa que permita el ingreso de la cantidad de horas trabajadas de 5 empleados. También debe solicitar el valor hora de trabajo.
- Luego, el programa debe informar el sueldo total de cada empleado.

Ejemplo de arrays – Entrada y salida de datos

```
using namespace std;
int main()
{
    int horas[5];
    float valorhora;

    cout << "Ingrese la cantidad de horas trabajadas de los \
empleados que cobran la misma razon $/hr" << endl;
    for (int cont = 0; cont < 5; cont++)
    {
        cout << "Empleado Nro" << (cont + 1) << ": ";
        cin >> horas[cont];
    }
    cout << "Ingrese el precio hora de los empleados: ";
    cin >> valorhora;
    cout << "***** TOTAL POR EMPLEADO *****" << endl;
    cout.precision(2);
    //ios::fixed Se muestran exactamente la cantidad de decimales establecidos en precision
    //ios::showpoint: Muestra 2 decimales siempre, aunque sea X.00
    cout.setf(ios::fixed | ios::showpoint);
    float TotalEmpleado;
    for (int cont = 0; cont < 5; cont++)
    {
        TotalEmpleado = horas[cont] * valorhora;
        cout << "Empleado Nro" << (cont + 1);
        cout << ": $" << TotalEmpleado << endl;
    }
    return 0;
}
```

Salida del programa

```
Ingrese la cantidad de horas trabajadas de los empleados que cobran la misma ra
zon $/hr
Empleado Nro1: 45
Empleado Nro2: 20
Empleado Nro3: 78
Empleado Nro4: 90
Empleado Nro5: 49
Ingrese el precio hora de los empleados: 62.5
***** TOTAL POR EMPLEADO *****
Empleado Nro1: $2812.50
Empleado Nro2: $1250.00
Empleado Nro3: $4875.00
Empleado Nro4: $5625.00
Empleado Nro5: $3062.50

Process returned 0 (0x0)   execution time : 20.236 s
Press any key to continue.
```


Ejemplo, continuación

- Extenderemos el programa anterior para considerar horas extra. Se considera hora extra cada hora trabajada que exceda las 40 hs semanales.
- De este modo, ahora, además de ingresar el valor de hora, el usuario también debe ingresar el valor de la hora extra.

Agregamos horas extra (más de 40 hs es extra)

```
using namespace std;
int main()
{
    int horas[5];
    float valorhora, valorhoraextra;

    cout << "Ingrese la cantidad de horas trabajadas de los \
empleados que cobran la misma razon $/hr" << endl;
    for (int cont = 0; cont < 5; cont++)
    {
        cout << "Empleado Nro" << (cont + 1) << ": ";
        cin >> horas[cont];
    }
    cout << "Ingrese el precio hora de los empleados: ";
    cin >> valorhora;
    cout << "Ingrese el valor de hora extra ";
    cin >> valorhoraextra;
    cout << "***** TOTAL POR EMPLEADO *****" << endl;
    cout.precision(2);
    //ios::fixed Se muestran exactamente la cantidad de decimales establecidos en precision
    //ios::showpoint: Muestra 2 decimales siempre, aunque sea X.00
    cout.setf(ios::fixed | ios::showpoint);
    float TotalEmpleado;
    for (int cont = 0; cont < 5; cont++)
    {
        if (horas[cont]<=40)
            TotalEmpleado = horas[cont] * valorhora;
        else
            TotalEmpleado = 40 * valorhora + (horas[cont] -40) * valorhoraextra;
        cout << "Empleado Nro" << (cont + 1);
        cout << ": $" << TotalEmpleado << endl;
    }
    return 0;
}
```

Un enfoque a ingeniería de software: Arrays paralelos con relación implícita por su índice

Si quisiéramos que el usuario ingrese la cantidad de horas extra trabajadas por cada empleado y además el valor de hora extra, podríamos tener dos arrays, uno para las horas extra trabajadas (horas) y otro para el valor hora de cada empleado.

Así, la cantidad de horas trabajadas por el empleado 1 estará en la posición 1 del array “horas” y su valor hora estará en la posición 1 del array valorHora.

```

#include <iostream>
#include <conio.h>
using namespace std;
// Observar en dónde definimos el tamaño del array
const int numEmps = 5;

int main()
{
    int horas[numEmps];
    float valorHora[numEmps];
    cout << "Por favor, ingrese las horas trabajadas por los empleados y el valor hora de cada uno" << endl;
    for (int index = 0; index < numEmps; index++)
    {
        cout << "Horas trabajadas por el empleado Nro " << (index + 1) << ": " << endl;
        cin >> horas[index];
        cout << "Valor hora para el empleado Nro " << (index + 1) << ": " << endl;
        cin >> valorHora[index];
    }
    cout << "**** Valor pagado por empleado ****" << endl;
    cout.precision(2);
    cout.setf(ios::fixed | ios::showpoint);
    for (int index = 0; index < numEmps; index++)
    {
        float total = horas[index] * valorHora[index];
        cout << "Empleado Nro " << (index + 1) << ": $" << total << endl;
    }
    getch();
}

```

IMPORTANTE

- Nunca usar el operador de asignación para copiar un array en otro. La operación de copia debería tener la siguiente forma:

```
for (int count=0; count < 4; count++)  
    newVal[count] = oldVal[count];
```

Mostrar todos los elementos de un array

- Para mostrar los elementos de un array (todos) también se suele usar estructuras iterativas:

```
int array[5] = { 10, 20, 30, 40, 50 };  
for (int count = 0; count < 5; count++)  
    cout << array[count] << endl;
```

Arrays como argumentos

- Para pasar un array como argumento de un subprograma sólo debe pasarse el nombre del mismo, sin indicar su dimensión.

```
#include <iostream>

using namespace std;

//Muestra el vector por pantalla
void mostrar(const int v[], int dim)
{
    for (int i = 0; i < dim; i++)
    {
        cout << v[i] << endl;
    }
    return;
}

int main()
{
    int vec[] ={34,54,12,78};
    mostrar(vec,4);
    return 0;
}
```

Arrays como argumentos (II) Pasaje por referencia

- Por defecto, los arrays se pasan por referencia, aunque no tengan el operador “&” especificado en el prototipo del subprograma.

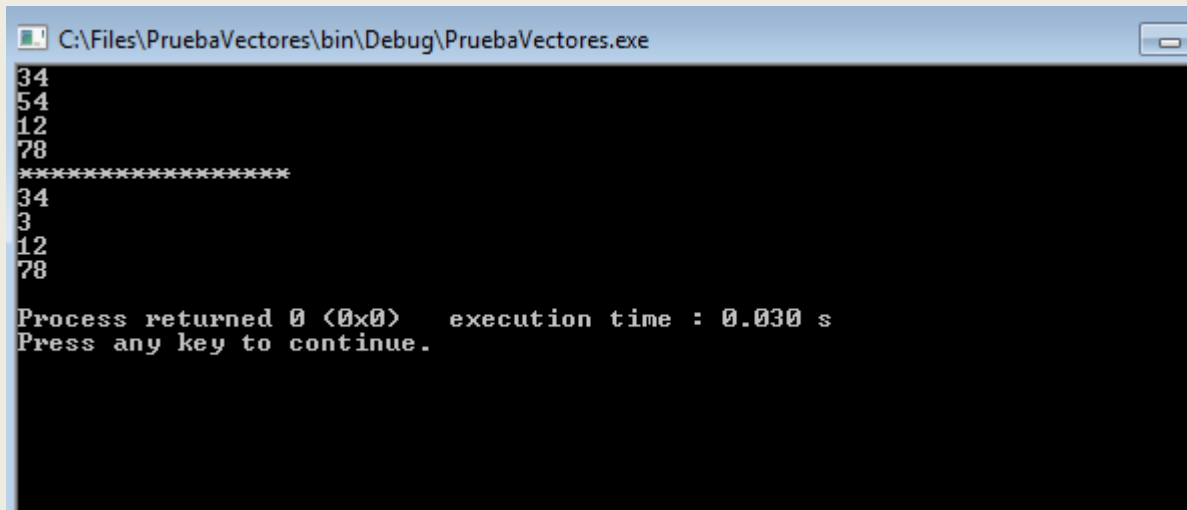
```
void insertar(int dato, int v[], int pos,int dim)
{
    if (pos>=dim)
        cout << "Error: la posición excede los limites del array" << endl;
    else
    {
        v[pos]= dato;
    }
    return;
}
```



```

int main()
{
    int vec[] = {34,54,12,78};
    mostrar(vec,4);
    cout << "*****" << endl;
    insertar(3,vec,1,4);
    mostrar(vec,4);
    return 0;
}

```



```

C:\Files\PruebaVectores\bin\Debug\PruebaVectores.exe
34
54
12
78
*****
34
3
12
78
Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.

```

Búsqueda - Contextualización

- Arrays
- **Métodos de ordenamiento:**
 - BubbleSort
 - BubbleSort mejorado
 - QuickSort
- **Método de búsqueda secuencial:**

Es de fácil implementación, pero cuando los arrays son muy grandes se vuelve muy lento.

Búsqueda Binaria - Concepto

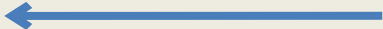
Este método se basa en el principio “Divide y Triunfarás”. Para compararlo con un caso de la vida real podemos imaginarnos cuando buscamos una palabra en un diccionario. ¿Comenzamos a buscar desde el principio, o dividimos el diccionario en partes y luego subdividimos esas partes hasta encontrar la palabra buscada?

Búsqueda Binaria – Concepto (II)

- El método binario se utiliza con la condición de tener ordenado previamente el vector donde se desea buscar un valor.

Veamos el funcionamiento con un caso práctico:

4
32
35
47
51
65
78
87



Valor a buscar

Búsqueda Binaria – Concepto (III)

4	<- Primero
32	
35	
47	<- Central
51	
65	
78	<- Buscado
87	<- Último

Para empezar a pensar en el algoritmo debemos considerar la necesidad de usar 3 índices:

- Primero: que indica el primer elemento desde donde comienza la búsqueda.
- Último: Marca la última posición del vector dentro del rango de búsqueda.
- Central: Marca la posición central del rango de búsqueda.

Necesitamos por supuesto el valor buscado.

Caso práctico – Primera iteración

4	<- Primero
32	
35	
47	<- Central
51	
65	
78	<- Buscado
87	<- Último

- Evaluemos:

¿ El valor del vector en la posición central, es igual al valor buscado?

Como no lo es, y además el valor buscado es mayor, busquemos ahora a partir de la segunda mitad (actualizando los índices para la próxima vuelta):

Primero = Central + 1 (porque el valor buscado es mayor al actual central)

Central = (primero + último) / 2

Caso práctico – Segunda iteración

Verifiquemos nuevamente, esta vez con los índices actualizados.

¿El valor en la posición central, es igual al valor buscado?

Nuevamente, no lo es, y es mayor:

Primero = Central + 1 (porque el valor buscado es mayor al actual central)

Central = (primero + último) / 2

4	
32	
35	
47	
51<- Primero	
65<- Central	
78<- Buscado	
87<- Último	

Caso práctico – Tercera iteración

4		
32		
35		
47		
51		
65<- Primero		
78<- Buscado	<- Central	
87<- Último		

¿El valor en la posición central, es igual al valor buscado?

¡Esta vez sí! Hemos encontrado el valor!

Atención – Valor no encontrado

Si el valor a buscar no estuviera en el vector se debe cortar el proceso cuando con una condición de corte.

Esa condición podría ser que el índice que apunta al primero sea igual al que apunta al último elemento.

Implementación en c++

```
int busquedaBinaria(const mi_tdato vec[], mi_tdato clave)
{
    bool encontrado = false;
    int larriba = cantElementos(vec)-1;
    int labajo = 0;
    int lcentro;
    int result = -1;
    while (labajo <= larriba && !encontrado)
    {
        lcentro = (larriba + labajo)/2;
        if (vec[lcentro] == clave)
        {
            result= lcentro;
            encontrado= true;
        }
        else
        {
            if (clave < vec[lcentro])
                larriba=lcentro-1;
            else
                labajo=lcentro+1;
        }
    }
    return result; // si no encontrado devuelve -1
}
```

Parametros:

valbusc: el valor a buscar en el vector.

vec: el vector donde buscar.

n: la cantidad de elementos del vector.

Pre: El vector que contiene los datos debe estar previamente ordenado con algún metodo de ordenamiento visto.

Post: Si se encuentra el elemento la función devuelve el índice donde se encontro. Sino, devuelve cero.

Ejercicio en laboratorio de PC

Poner un breakpoint en el ciclo while del procedimiento busqbin en el programa principal provisto, ejecutar el programa y analizar línea a línea la actualización de los índices.

4<- Primero	
32	
35	
47<- Central	
51	
65	
78<- Buscado	
87<- Último	

4	
32	
35	
47	
51<- Primero	
65<- Central	
78<- Buscado	
87<- Último	

4		
32		
35		
47		
51		
65<- Primero		
78<- Buscado	<- Central	
87<- Último		

Eficiencia de la búsqueda binaria

En este algoritmo, cada vez que se hace una comparación , el vector se divide a la mitad. Si n es el tamaño del vector, las sucesivas búsquedas serán con subvectores de:

$n/2, n/4, n/8 \dots$

Y el proceso finaliza cuando:

$$n / 2^k \leq 1$$

Por lo tanto, despejando y aplicando logaritmo:

$$\text{Log}_2 n < k$$

Eficiencia de la búsqueda binaria y comparación con búsqueda lineal

Eficiencia de la búsqueda binaria:

$$O(\log_2 n)$$

Citando un ejemplo para un caso de 40000 elementos la búsqueda lineal en el peor de los casos requiere 40000 comparaciones y 20000 en promedio, mientras que la búsqueda binaria nunca requerirá más que $\log_2 40000$

Arrays de dos dimensiones

- Supongamos que queremos almacenar en una variable las ventas de 3 productos durante el primer cuatrimestre del año. Intuitivamente pensaríamos en una tabla:

	Enero	Febrero	Marzo	Abril
Producto 1	[0][0]	[0][1]	[0][2]	[0][3]
Producto 2	[1][0]	[1][1]	[1][2]	[1][3]
Producto 3	[2][0]	[2][1]	[2][2]	[2][3]

```

#include <iostream>
using namespace std;

int main()
{
    float ventas[3][4];
    float ventasTotales =0;
    cout << "Este programa calcula las ventas totales sumando los tres productos
del primer cuatrimestre" << endl;
    cout << "Ingrese los datos de ventas" << endl;

    for (int prod = 0; prod < 3; prod++)
    {
        for (int mes = 0; mes < 4; mes++)
        {
            cout << "Ventas del producto " << (prod + 1);
            cout << ", mes " << (mes + 1) << ": $";
            cin >> ventas[prod][mes];
        }
        cout << endl;
    }
    for (int prod = 0; prod < 3; prod++)
    for (int mes = 0; mes < 4; mes++)
        ventasTotales += ventas[prod][mes];
    cout.precision(2);
    cout.setf(ios::fixed | ios::showpoint);
    cout << "Las ventas fueron: $" << ventasTotales << endl;
    return 0;
}

```

Arrays de dos dimensiones como parámetros

- Cuando se pasa por parámetro a un subprograma un array de dos dimensiones (tabla), el parámetro formal debe tener indicada la cantidad de columnas.

```
int array[10][10];  
/*Definición*/  
void passFunc(int a[][10], int qfilas, int qcols)  
{ // ... }
```

```
passFunc(array, filas, cols); // Invocación
```


Arrays de Strings

- Un array bidimensional de chars es un array de C-strings.

```
#include <iostream>
using namespace std;

int main()
{
    char meses[12][50] = {"Enero", "Febrero", "Marzo","Abril",
                          "Mayo", "Junio","Julio", "Agosto",
                          "Septiembre","Octubre", "Noviembre","Diciembre"};
    int days[12] = { 31, 28, 31, 30, 31, 30, 31, 31,30, 31, 30, 31};
    for (int count = 0; count < 12; count++)
    {
        cout << meses[count] << " tiene " << days[count] << " dias." << endl;
    }
    return 0;
}
```

```
Enero tiene 31 dias.  
Febrero tiene 28 dias.  
Marzo tiene 31 dias.  
Abril tiene 30 dias.  
Mayo tiene 31 dias.  
Junio tiene 30 dias.  
Julio tiene 31 dias.  
Agosto tiene 31 dias.  
Septiembre tiene 30 dias.  
Octubre tiene 31 dias.  
Noviembre tiene 30 dias.  
Diciembre tiene 31 dias.  
  
Process returned 0 (0x0)    execution time : 0.015 s  
Press any key to continue.
```