

FUNDAMENTOS DE PROGRAMACIÓN

TEORÍA N.º 5

UNIDAD 3: ESTRUCTURAS DE DATOS BÁSICAS VECTORES



**Facultad de Ingeniería
Universidad Nacional de Jujuy**

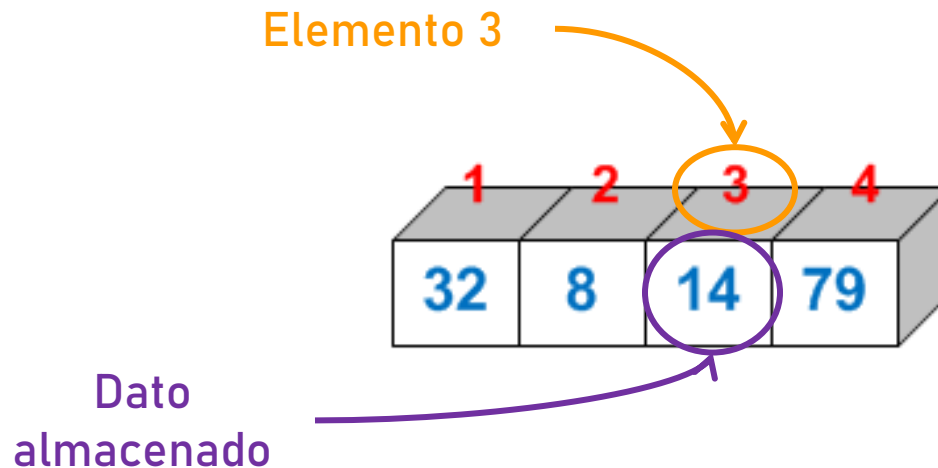
Datos Simples y Compuestos

- Las variables de tipos de datos simples permiten almacenar un valor específico.
- Las variables de tipos de datos compuestos permiten almacenar un grupo de valores.



Arreglos. Definición

- Un arreglo es un **conjunto finito** de elementos del mismo tipo cuyo acceso se realiza a través de *índices*.
- Disposición contigua en memoria: los casilleros están consecutivos.



Índices

- Los índices permiten referenciar posiciones específicas de un arreglo.



- Los índices pueden ser constantes, variables o expresiones de tipo ordinal.
- En C++ estándar, la indexación SIEMPRE comienza en 0 y finaliza en tamaño - 1.
- Los índices permiten identificar en forma individual cada elemento del arreglo.

Índices: La Regla de Base Cero (0 a N-1)

Estructura de Índices en Memoria

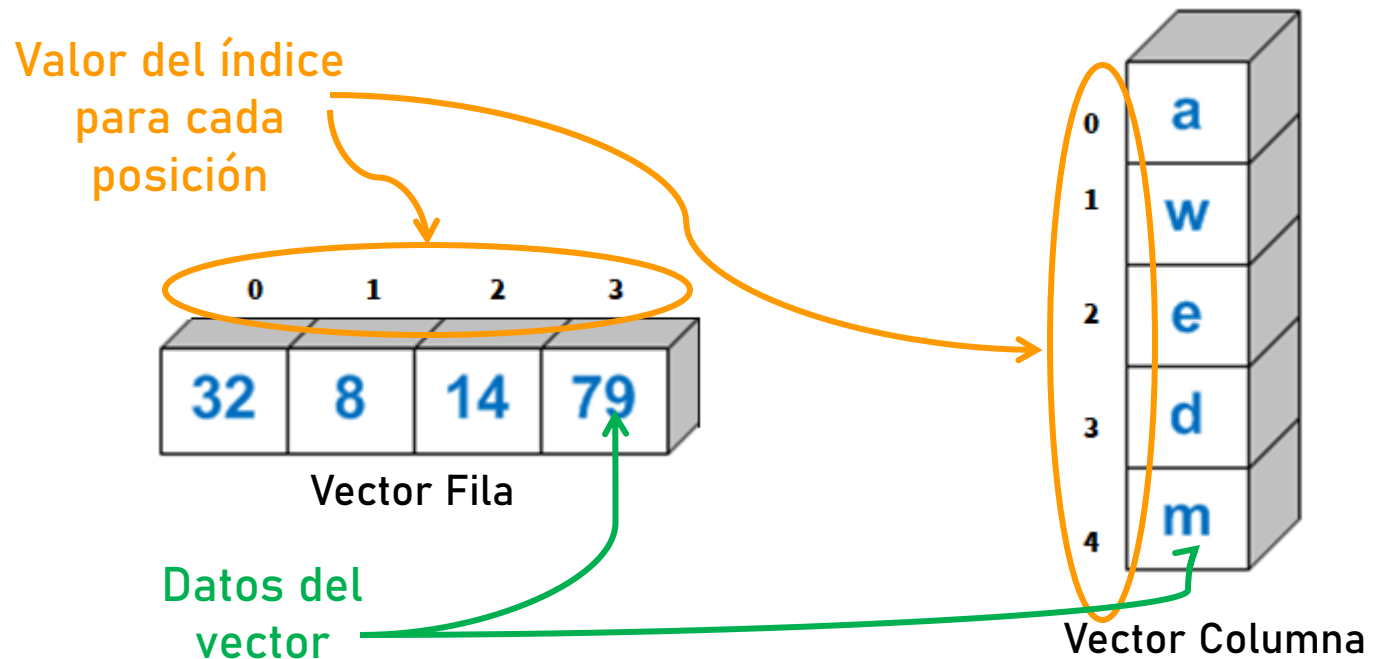
Posición (Índice):	[0]	[1]	[2]	[3]	[4]	
Dato almacenado:	32	8	14	79	51	
Total de elementos: $N = 5$			Primer elemento: $v[0]$			Último elemento: $v[4]$ ($N-1$)

Reglas Críticas de Acceso

- El primer casillero SIEMPRE es el índice 0.
- El último casillero SIEMPRE es el índice tamaño - 1.
- Intentar acceder al índice N (ejemplo: $v[5]$ en vector de 5) provoca acceso fuera de rango.

Vectores

- El vector es el tipo más simple de arreglo ya que sus elementos se disponen en una única fila o columna y pueden accederse utilizando un solo índice.



¿Cómo se declara?

- En general, se puede aplicar la siguiente sintaxis:

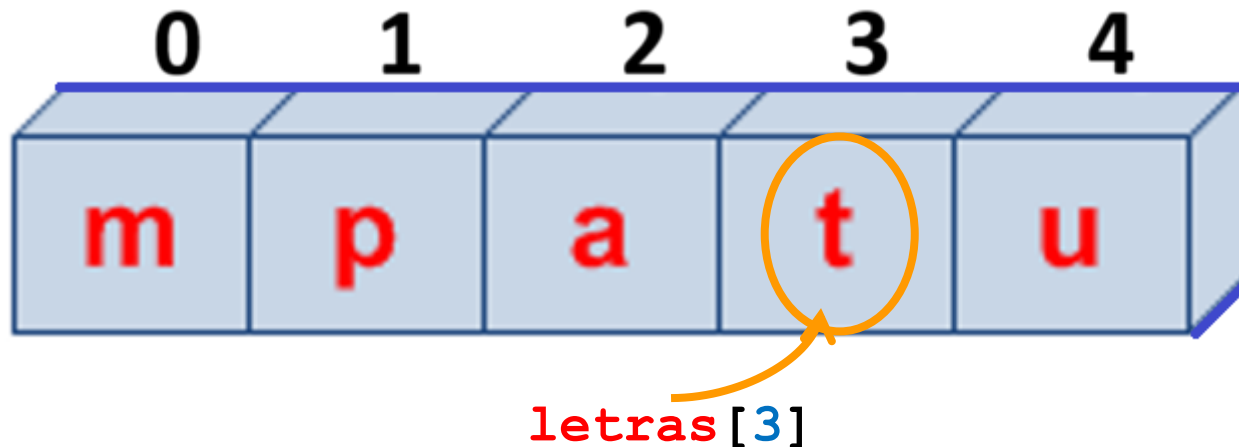
```
#include <iostream>
#include <vector> // Inclusion de la cabecera necesaria

using namespace std;

int main() {
    // Sintaxis y declaracion: vector<tipo> nombre;
    vector<int> v; //vector llamado v de tipo int, sin tamaño ni valor inicial
    vector<float> num(5,1.5); //vector llamado num de tipo float, con tamaño 5
    //y componentes iniciados en 1.5
    return 0;
}
```

¿Cómo se accede?

- Para acceder a una posición específica de un vector debe indicarse el nombre del vector y el índice del elemento requerido.



- *letras*: nombre del vector
- 3: posición del vector

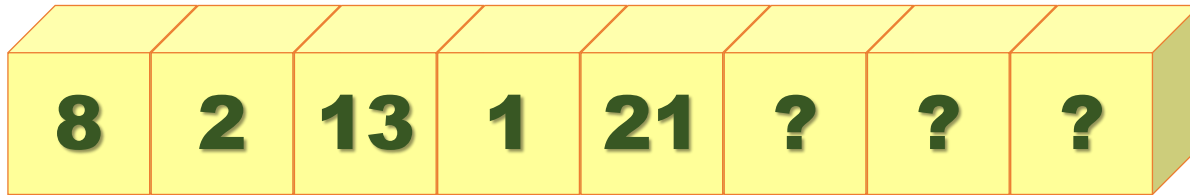
```
cout << letras.at(3);  
// Si i está fuera de rango,  
// detiene el programa con error claro.
```

push_back

- La operación de *push_back* permite añadir al final del vector un valor introducido por teclado.
- Añade un nuevo elemento al final del vector e incrementa automáticamente su `size()` en 1.

```
nombre_vector.push_back(v) ;
```

```
num.push_back(3) ;
```

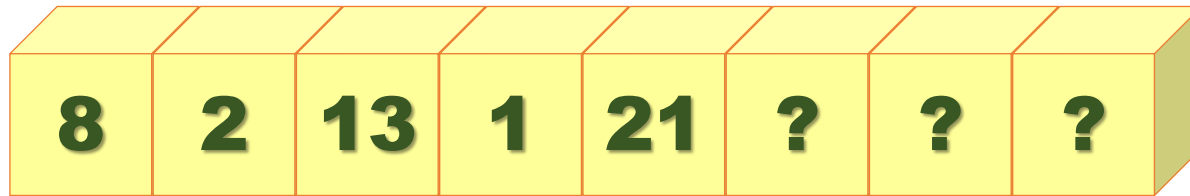
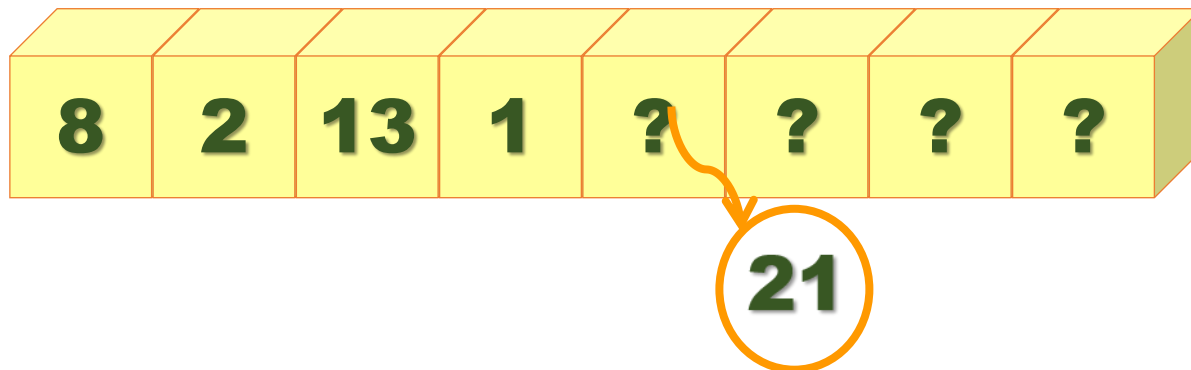


pop_back

- La operación de *pop_back* permite eliminar el contenido de la última posición del vector.
- Elimina el último componente del vector y decrementa su `size()` en 1.

```
nombre_vector.pop_back();
```

```
num.pop_back();
```



Consulta y Gestión: `size()`, `empty()`, `clear()`

`v.size()`

- Devuelve la cantidad actual de componentes.
- Retorna tipo `size_t`.
- Control ideal para ciclos `for` de 0 a `size()-1`.

`v.empty()`

- Retorna `true` si el vector no tiene elementos (`size == 0`).
- Ideal para bucles `while`.

`v.clear()`

- Elimina todos los elementos.
- Deja `v.size()` en 0 de inmediato.
- Reinicia contenedores para nuevos lotes.

Recorrido de Arreglos

- Las operaciones de asignación, lectura y escritura, se aplican a elementos individuales de un arreglo.
- Para procesar todos los elementos de un arreglo es necesario recorrerlo.
- El acceso sucesivo a las posiciones un arreglo se denomina recorrido del arreglo.
- Para recorrer un arreglo se usan estructuras repetitivas que incluyen sentencias para la asignación, lectura o escritura.

```
// Recorrido de lectura / escritura:  
for (int i = 0; i < v.size(); i++) {  
    cout << v[i] << " ";  
}  
  
// Recorrido inverso (fin a principio):  
for (int i = v.size() - 1; i >= 0; i--) {  
    cout << v[i] << " ";  
}
```

Ordenamiento de Vectores: Función sort()

Inclusión Obligatoria

```
#include <algorithm>
// Sintaxis: sort(iterador_inicio, iterador_fin);
```

Orden Ascendente (Menor a Mayor)

```
vector<int> v = {9, 2, 7, 1};
sort(v.begin(), v.end());

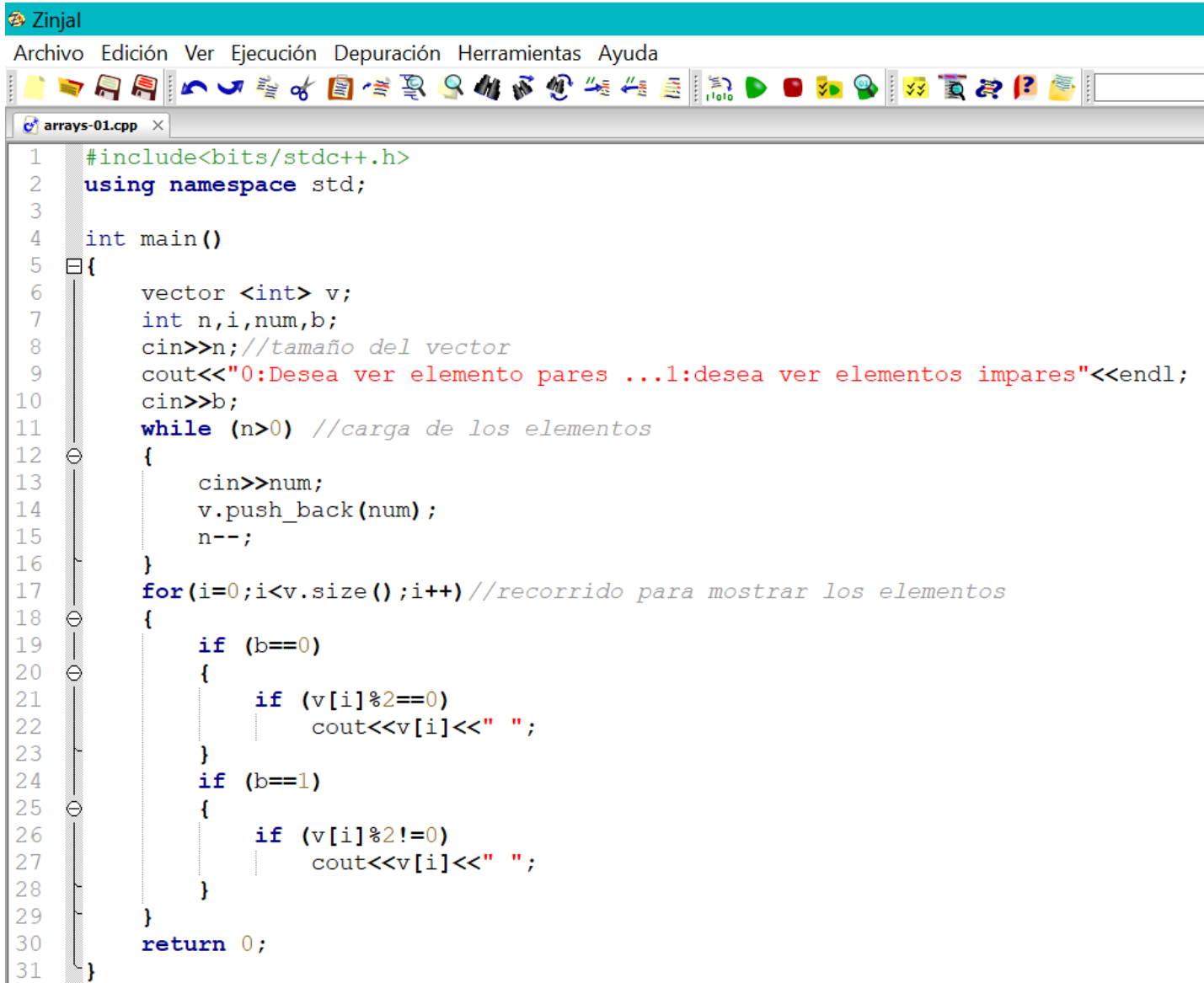
// Resultado en v:
// [1, 2, 7, 9]
```

Orden Descendente (Mayor a Menor)

```
vector<int> v = {9, 2, 7, 1};
// Con iteradores reversos:
sort(v.rbegin(), v.rend());

// Resultado en v:
// [9, 7, 2, 1]
```

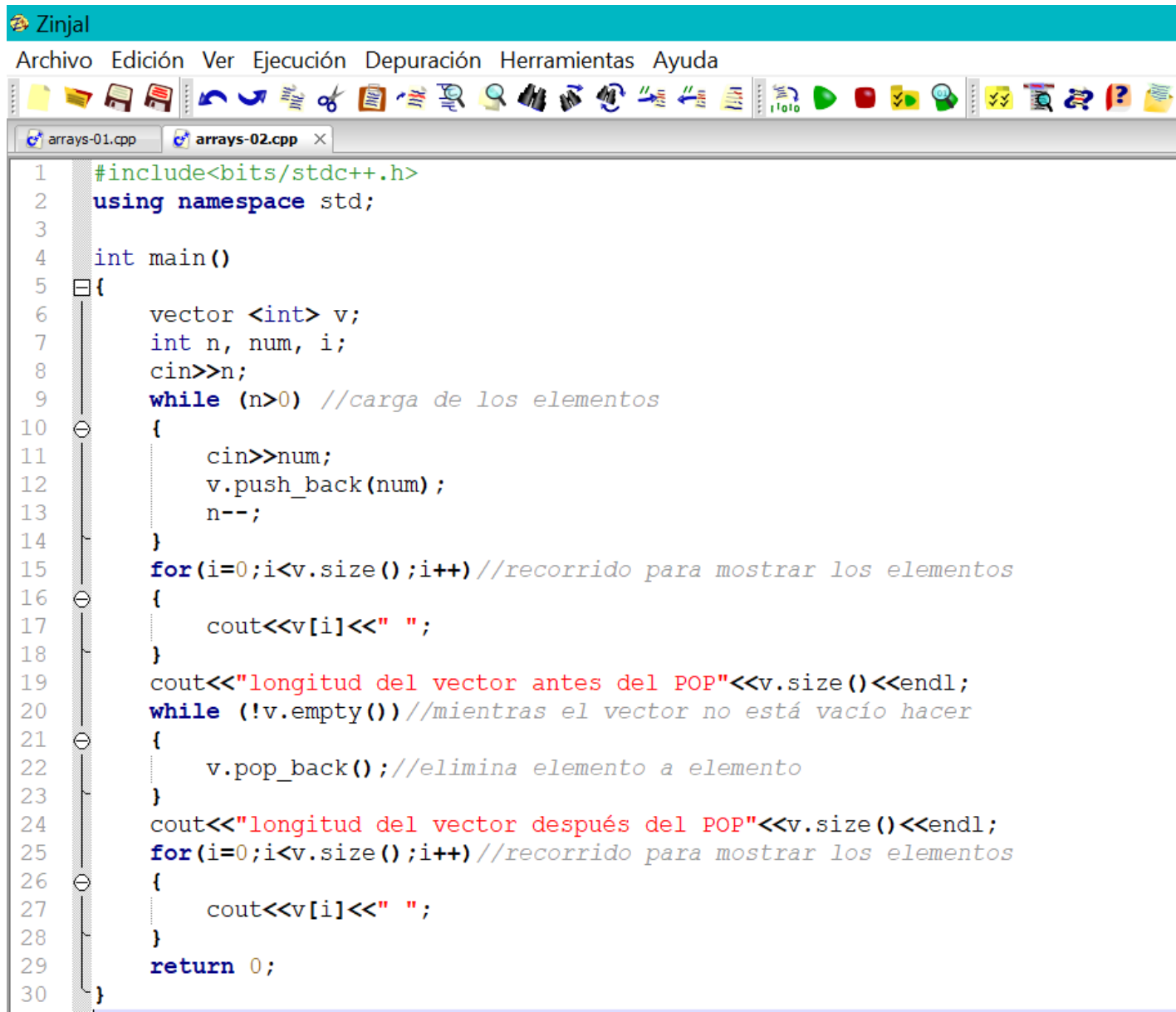
Ejercicios desarrollados en Clase (1)



The screenshot shows the Zinjal IDE interface. The title bar is teal with the Zinjal logo. The menu bar includes Archivo, Edición, Ver Ejecución, Depuración, Herramientas, and Ayuda. The toolbar contains various icons for file operations, editing, and execution. The active window is titled 'arrays-01.cpp'. The code is as follows:

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  int main()
5  {
6      vector <int> v;
7      int n,i,num,b;
8      cin>>n;//tamaño del vector
9      cout<<"0:Desea ver elemento pares ...1:desea ver elementos impares"<<endl;
10     cin>>b;
11     while (n>0) //carga de los elementos
12     {
13         cin>>num;
14         v.push_back(num);
15         n--;
16     }
17     for(i=0;i<v.size();i++)//recorrido para mostrar los elementos
18     {
19         if (b==0)
20         {
21             if (v[i]%2==0)
22                 cout<<v[i]<<" ";
23         }
24         if (b==1)
25         {
26             if (v[i]%2!=0)
27                 cout<<v[i]<<" ";
28         }
29     }
30     return 0;
31 }
```

Ejercicios desarrollados en Clase (2)



The screenshot shows a C++ IDE window titled 'Zinjal'. The menu bar includes 'Archivo', 'Edición', 'Ver', 'Ejecución', 'Depuración', 'Herramientas', and 'Ayuda'. The toolbar contains various icons for file operations, editing, and execution. Two tabs are open: 'arrays-01.cpp' and 'arrays-02.cpp'. The active tab, 'arrays-02.cpp', displays the following C++ code:

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  int main()
5  {
6      vector<int> v;
7      int n, num, i;
8      cin>>n;
9      while (n>0) //carga de los elementos
10     {
11         cin>>num;
12         v.push_back(num);
13         n--;
14     }
15     for(i=0;i<v.size();i++)//recorrido para mostrar los elementos
16     {
17         cout<<v[i]<<" ";
18     }
19     cout<<"longitud del vector antes del POP"<<v.size()<<endl;
20     while (!v.empty())//mientras el vector no está vacío hacer
21     {
22         v.pop_back();//elimina elemento a elemento
23     }
24     cout<<"longitud del vector después del POP"<<v.size()<<endl;
25     for(i=0;i<v.size();i++)//recorrido para mostrar los elementos
26     {
27         cout<<v[i]<<" ";
28     }
29     return 0;
30 }
```

Ejercicios desarrollados en Clase (3)

```
Zinjal
Archivo Edición Ver Ejecución Depuración Herramientas Ayuda
arrays-01.cpp arrays-02.cpp arrays-03.cpp x
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main()
5 {
6     vector<int> v, v2, m;
7     int n,n2,num, i,p;
8     cin>>n;
9     n2=n;
10    while (n>0) //carga de los elementos
11    {
12        cin>>num;
13        v.push_back(num);
14        n--;
15    }
16    while (n2>0) //carga de los elementos
17    {
18        cin>>num;
19        v2.push_back(num);
20        n2--;
21    }
22    for(i=0;i<v.size();i++) //recorrido para mostrar los elementos
23    {
24        cout<<"v " << i<< " " <<v[i]<< " " <<endl;
25        cout<<"v2 " << i<< " " <<v2[i]<< " " <<endl;
26    }
27    for(i=0;i<v.size();i++) //completamos el tercer vector
28    {
29        p=v[i] * v2[i];
30        m.push_back(p);
31    }
32    for(i=0;i<m.size();i++) //recorrido para mostrar los elementos
33    {
34        cout<<"m " << i<< " " <<m[i]<< " " <<endl;
35    }
36    return 0;
37 }
38
```

Ejercicios desarrollados en Clase (4)

Zinjal

Archivo Edición Ver Ejecución Depuración Herramientas Ayuda

arrays-01.cpp arrays-02.cpp arrays-03.cpp arrays-04.cpp*

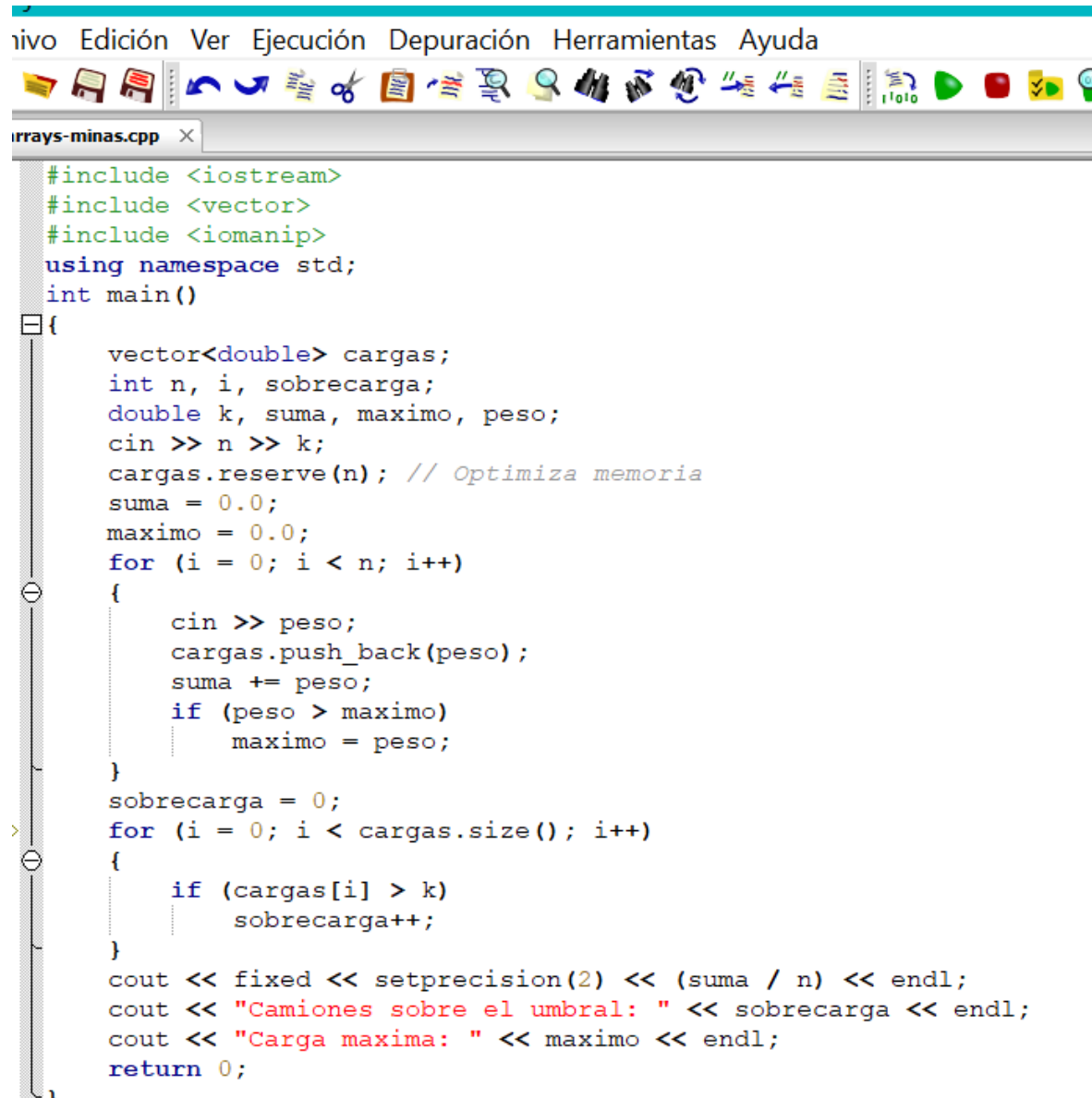
```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  int main()
5  {
6      vector<int> v;
7      int i,n,nro,val, c=0;
8      cin>>n;
9      cin>>nro;
10     while (n>0)
11     {
12         cin>>val;
13         v.push_back(val);
14         n--;
15     }
16     for(i=0;i<v.size();i++)//tener en cuenta el signo correto es <; NO <=
17     {
18         if(v[i]==nro)
19             c++;
20     }
21     cout<<" el número "<<nro<<" aparece "<<c <<" veces";
22     return 0;
23 }
```

Ejercicio desafío (1)

Problema Aplicado (Ingeniería de Minas / Industrial)

- Contexto: En una explotación minera de Jujuy se registran los pesajes de N camiones en toneladas.
- Consigna:
 1. Cargar las N cargas en un vector dinámico `vector<double> cargas` mediante `push_back()`.
 2. Calcular la carga promedio de la jornada y mostrarla con 2 decimales (`setprecision`).
 3. Dado un umbral de sobrecarga K (ejemplo: 100 toneladas), contar cuántos camiones lo superaron.
 4. Indicar el valor de la carga máxima registrada.

Ejercicio desafío (2)



The image shows a screenshot of a C++ IDE with a menu bar (Archivo, Edición, Ver, Ejecución, Depuración, Herramientas, Ayuda) and a toolbar. The active window is 'arrays-minas.cpp'. The code is as follows:

```
#include <iostream>
#include <vector>
#include <iomanip>
using namespace std;
int main()
{
    vector<double> cargas;
    int n, i, sobrecarga;
    double k, suma, maximo, peso;
    cin >> n >> k;
    cargas.reserve(n); // Optimiza memoria
    suma = 0.0;
    maximo = 0.0;
    for (i = 0; i < n; i++)
    {
        cin >> peso;
        cargas.push_back(peso);
        suma += peso;
        if (peso > maximo)
            maximo = peso;
    }
    sobrecarga = 0;
    for (i = 0; i < cargas.size(); i++)
    {
        if (cargas[i] > k)
            sobrecarga++;
    }
    cout << fixed << setprecision(2) << (suma / n) << endl;
    cout << "Camiones sobre el umbral: " << sobrecarga << endl;
    cout << "Carga maxima: " << maximo << endl;
    return 0;
}
```

Bibliografía

- Sznajdleder, Pablo Augusto. Algoritmos a fondo. Alfaomega. 2012.
- López Román, Leobardo. Programación estructurada y orientada a objetos. Alfaomega. 2011.
- De Giusti *et al.* Algoritmos, datos y programas, conceptos básicos. Editorial Exacta, 1998.
- Joyanes Aguilar, Luis. Fundamentos de Programación. Mc Graw Hill. 1996.
- Joyanes Aguilar, Luis. Programación en Turbo Pascal. Mc Graw Hill. 1990.