Fundamentos de Programación

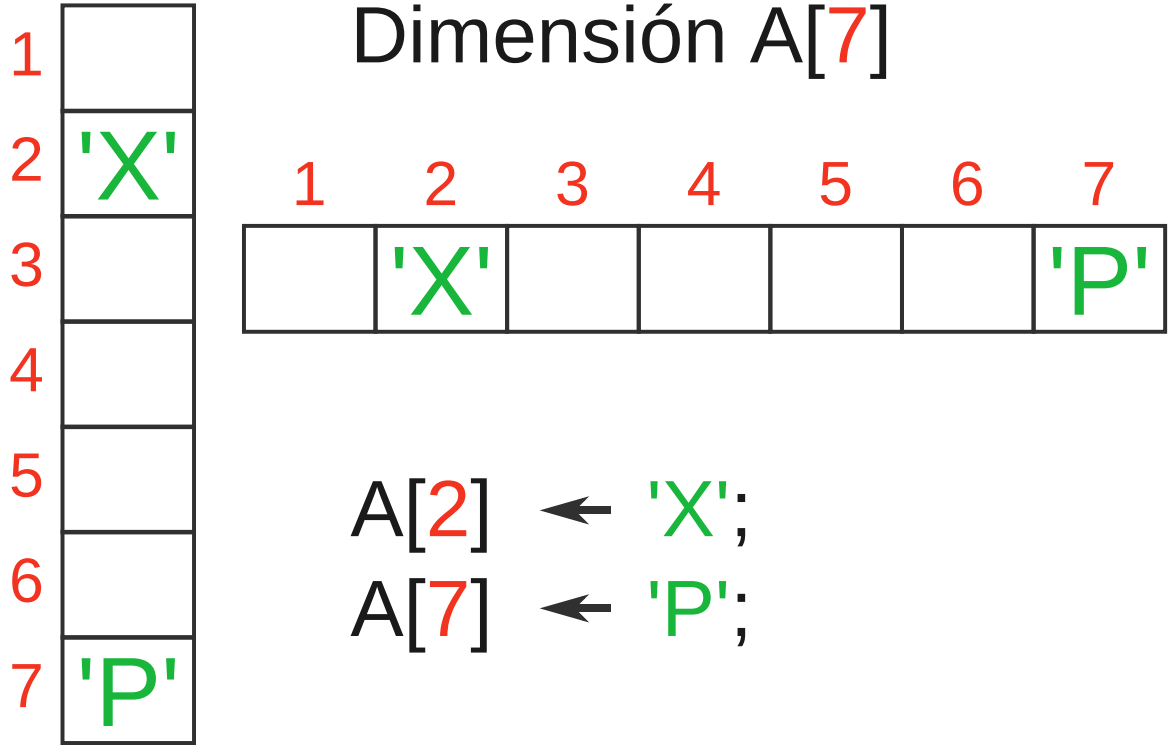
Unidad 9: Arreglos y Structs

Pablo Novara

FICH UNL
Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas

ARREGLOS EN PSEUDOCÓDIGO

Secuencia de datos **homogeneos** donde cada elemento tiene una **posición relativa** dentro de la misma que puede ser establecida por uno o más **indices**.

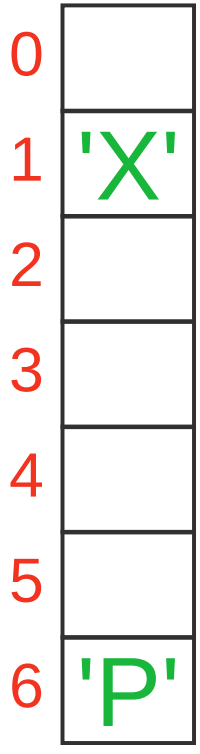


ARREGLOS EN C++

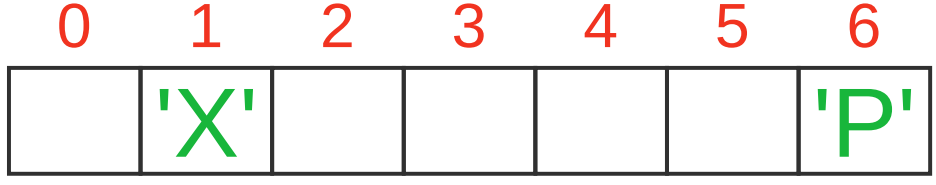
- Datos **homogeneos** (mismo tipo).
- Los elementos se almacenan de forma contigua en memoria.
- Se acceden individualmente mediante **índices** (enteros).
- Los **índices** se toman en **base 0**.
  - En un arreglo `v` de `10` elementos, el primero es `v[0]` y el décimo es `v[9]`
- **C++ NO controla la validez de los índices.**
- Hay varias formas de representar arreglos en C++.
  - En esta materia vamos a usar **`std::vector`**

ARREGLOS EN C++

Secuencia de datos **homogeneos** donde cada elemento tiene una **posición relativa** dentro de la misma que puede ser establecida por uno o más **indices**.




```
vector<char> v(7);
```




```
v[1] ='X';
v[6] ='P';
```

EL TIPO `std::vector`

- Vector *vacío*:

```
vector<int> v;
```

- Idem, con sus 10 elementos inicializados todos en 5:

```
vector<int> v(10,5);
```

✅ *el valor inicial (5) es opcional*

- Vector de 4 elementos inicializados con una lista de valores específicos:

```
vector<int> v = { 8, 3, 7, 1 };
```

EJEMPLO

1. Generar y mostrar un vector con 10 enteros:

```
#include <iostream>
#include <vector>
```
se requiere un #include
```
using namespace std;

int main() {
    vector<int> v(10);
    // cargar en cada pos. un num. aleatorio
    for(int i = 0; i<10; ++i)
        v[i] = rand()%101;
    // mostrar el vector
    for(int i = 0; i<10; ++i)
        cout<<"Pos "<<i<<":  " <<v[i]<<endl;
}
```

EJEMPLO V2

```
vector<int> generar_vector(int n, int max) {
    vector<int> v(n);
    for(int i = 0; i<n; ++i)
        v[i] = rand()%(max+1);
    return v;
}

int main() {
    // generar un vector con nums aleatorios
    vector<int> v = generar_vector(10,100);
    // mostrar el vector
    for(int i = 0; i<10; ++i)
        cout<<"Pos "<<i<<":  "<<v[i]<<endl;
}
```

El tamaño no es parte del tipo de dato

Se puede asignar/copiar un vector completo

EJEMPLO V3

```
vector<int> generar_vector(int n, int max) {
    vector<int> v(n);
    for(int i = 0; i<n; ++i)
        v[i] = rand()%(max+1);
    return v;
}

void mostrar_vector(const vector<int> &v) {
    for(int i=0; i<v.size(); ++i)
                        Se puede consultar el tamaño
        cout<<"Pos "<<i<<":  "<<v[i]<<endl;
}

int main() {
    vector<int> v = generar_vector(10);
    mostrar_vector(v);
}
```

EJEMPLO V4

```
vector<int> generar_vector(int n, int max) {
    vector<int> v(n);
    for(int &x : v)
        x irá tomando "por referencia" cada valor de v
        x = rand()%(max+1);
    return v;
}

void mostrar_vector(const vector<int> &v) {
    for(int x : v)
        x irá tomando "por copia" cada valor de v
        cout<<"Pos "<<i<<":  "<<x<<endl;
                ya no conozco el indice i
}

int main() {
    vector<int> v = generar_vector(10);


    mostrar_vector(v);
}
```

RESUMEN `std::vector`

- Declarar e inicializar:

```
vector<string> v1; // vacío
vector<float>  v2(10, 0.0); // 10 ceros
vector<int>    v3 = { 8, 3, 7, 1 };
```

- Acceder a un elemento:

```
cin>>v[i]; cout<<v[i]; v[i]=42;
```

- Consultar tamaño:

```
for(size_t i = 0; i<v.size(); i++) { ...
```

para evitar un warning usar `size_t` en lugar de `int`

- Acceder a todos los elementos:

```
for (int &x:v) cout<<x<<endl;
```

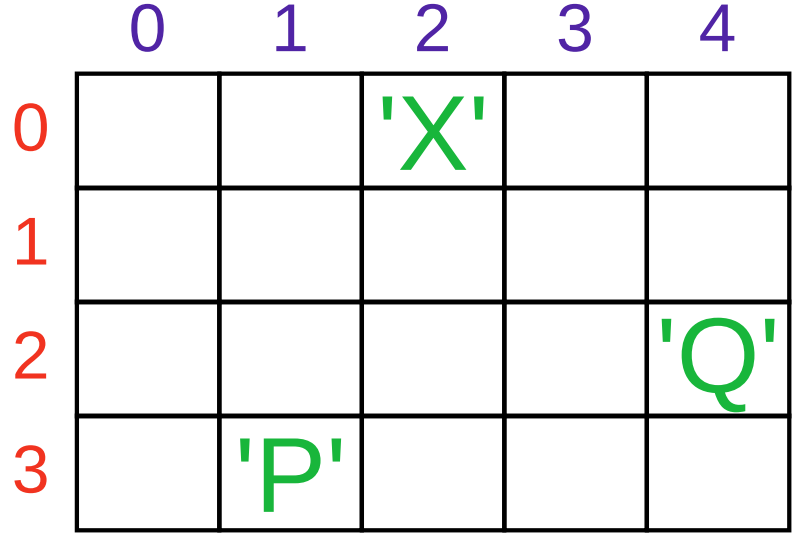
EJEMPLOS

2. Se poseen los resultados de una evaluación de un curso de 60 estudiantes. Se desea informar:
    1. la calificación media.
    2. la mejor calificación del curso
    3. qué porcentaje aprobó la evaluación

MATRICES EN C++

Vamos a utilizar un tipo `matrix` **no estandar** que presenta una interface muy similar a la de `vector`.

```
matrix<char> A(4,5);
```




```
A[0][2] = 'X';
A[3][1] = 'P';
A[2][4] = 'Q';
```

Deben descargar desde el Moodle e instalar en Zinjal el complemento para `matrix`

EJEMPLO

3. Escriba funciones para:
    1. Generar una matriz con valores aleatorios
    2. Mostrar una matriz en pantalla

```
#include <matrix>
...
int main() {
    int n, m;
    cin >> n >> m;
    matrix<int> A = generar_matriz(n,m);
    mostrar_matriz(A);
}
```

EJEMPLO

```
matriz<int> generar_matriz(int n, int m) {
    // declarar matriz de n filas x m cols
    matrix<int> A(n,m);
    // recorrer y llenar
    for(size_t i = 0; i<A.size(0); ++i)
        for(size_t j = 0; j<A.size(1); ++j)
            A[i][j] = rand()%101;
    return A;
}
void muestra_matriz(const matrix<int> &A) {
    for(size_t i = 0; i<A.size(0); ++i) {
        for(size_t j = 0; j<A.size(1); ++j)
            cout << setw(5) << A[i][j];
        cout << endl;
    }
}
```

Notar el uso de índices como `[i][j]` (dos pares de corchetes) en lugar de `[i,j]` (separados por coma)

RESUMEN `matrix`

- Declarar e inicializar:

```
matrix<char>  m1; // vacia
matrix<float> m2(3,2, 0.0); // 3x2 ceros
matrix<int>   m3 = { { 8, 3, 7 } ,
                     { 1, 5, 4 } }; // 2x3
```

- Acceder a un elemento:

```
cin>>m[i][j]; cout<<m[i][j]; m[i][j]=42;
```

- Consultar tamaño (filas y columnas):

```
for(size_t i = 0; i<m.size(0); i++) {
    for(size_t j = 0; j<m.size(1); j++) {
        ...
```

EJEMPLO

4. Escriba funciones para:
    1. Generar una matriz con valores aleatorios
    2. Mostrar una matriz en pantalla
    3. Obtener la posición del máximo valor de una matriz.

OPERACIONES ESPECIALES

- Copia/asignación:

```
v1 = v2; // entre vectores
m1 = m2; // entre matrices
```

No importa si no tenían el mismo tamaño

- Cambio de tamaño:

```
v.resize(10); // 10 es el nuevo tamaño
m.resize(8,12); // 8x12 es el nuevo tamaño
```

- Agregar un elemento nuevo al final:

```
v.push_back(x);
```

Solo para vector

EJEMPLOS

4. Modifique el ejemplo nro 2 para el caso en que no se conoce a priori la cantidad de estudiantes.

5. Escriba una función para buscar y eliminar un elemento de un vector.

6. Escriba una función para agregar a una matriz de ventas una fila y una columna de totales.

STRUCTS

Estructura de datos **heterogenea**.

Cada componente (atributo) se declara individualemente y se refiere a travez de un identificador propio.

```
struct Alumno {
    string nombre;
    int dni;
    float prom;
};
```

DECLARACIÓN Y ACCESO A MIEMBROS

```
struct Alumno {
    string nombre;
    int dni;
    float prom;
};
```

en este punto todavía no existe ningún alumno

- `Alumno` es ahora un nuevo tipo de dato:

```
Alumno x; // x sí es un Alumno
Alumno y; // y es otro Alumno
```

DECLARACIÓN Y ACCESO A MIEMBROS

```
struct Alumno {
    string nombre;
    int dni;
    float prom;
};
```

- `Alumno` es ahora un nuevo tipo de dato:

```
Alumno x = { "Adan", 1, 10.0 };
Alumno y = { "Eva",  2, 10.0 };
```

se puede inicializar mediante una "lista" de valores

- Se accede por atributo utilizando el `.` (punto):

```
cout << "Nombre: " << x.nombre << endl;
cout << "DNI: " << x.dni << endl;
cout << "Promedio: " << x.prom << endl;
```

EJEMPLOS

7. Declare un struct para representar una fecha.

8. Escriba un función edad que permita obtener la edad de una persona a partir de la fecha actual y la fecha de nacimiento.

9. Escriba un programa cliente que permita cargar ambas fechas y mostrar la edad.

ARREGLO DE STRUCTS

```
struct Alumno {
    string nombre;
    int dni;
};

vector<Alumno> x(30);
for (size_t i=0;i<x.size();i++) {
    cin >> x[i].nombre >> x[i].dni;
}
```

- `x` es un **arreglo** (de Alumnos)
- `x[i]` es el i-ésimo **Alumno**
- `x[i].nombre` es el **nombre** del i-ésimo Alumno

STRUCTS CON ARREGLOS

```
struct Curso {
    int codigo;
    vector<int> dni;
};
Curso c;
cin >> c.codigo;
c.dni.resize(30);
for (int i=0;i<10;i++)
    cin >> c.dni[i];
```

el arreglo estaba vacío

- `c` es un **Curso**
- `c.codigo` es el **código** del Curso c
- `c.dni` es el **arreglo** de DNIs del Curso c
- `c.dni[i]` es el **i-ésimo DNI** del Curso c

STRUCTS CON ARREGLOS DE STRUCTS CON ARREGLOS

```
struct Alumno {
    int dni;
    vector<float> notas;
};

struct Curso {
    int codigo;
    vector<Alumno> a;
};
```

Curso es un struct...

...que tiene por atributo un arreglo a...

...cuyos elementos son structs de tipo Alumno...

...y cada uno guarda un arreglo con sus notas.

```
struct Alumno {
    int dni;
    vector<float> notas;
};
struct Curso {
    vector<Alumno> a;
};
Curso c;
```

- `c` es un **Curso**
- `c.a` es el **arreglo** de Alumnos del Curso c
- `c.a[i]` es el i-ésimo Alumno del Curso c
- `c.a[i].dni` es el **dni** del i-ésimo Alumno de c
- `c.a[i].notas` es el **arreglo** de notas del Alumno i
- `c.a[i].notas[j]` es la j-ésima **nota** del Alumno i

STRUCTS CON ARREGLOS DE STRUCTS CON ARREGLOS

```
struct Alumno {
    int dni;
    vector<float> notas;
};
struct Curso {
    int codigo;
    vector<Alumno> a;
};

Curso c;
cin >> c.codigo;
c.a.resize(30);
for (int i=0; i<c.a.size(); i++) {
    cin >> c.a[i].dni;
    c.a[i].notas.resize(5);
    for (int j=0;j<c.a[i].notas.size();j++)
        cin >> c.a[i].notas[j];
}
```

USO DE ALIAS Y FOR ABREV. PARA SIMPLIFICAR

Antes:

```
for (size_t i=0; i<c.a.size(); i++) {
    cin >> c.a[i].dni;
    c.a[i].notas.resize(5);
    for (size_t j=0;j<c.a[i].notas.size();j++)
        cin >> c.a[i].notas[j];
}
```

Despues:

```
for (Alumno &un_alumno : c.a) {
        me "olvido" del vector, y me concentro en un alumno
    cin >> un_alumno.dni;
    un_alumno.notas.resize(5);
    for (int &una_nota : un_alumno.notas)
        cin >> una_nota;
}
```

USO DE FUNCIONES PARA SIMPLIFICAR

```
void mostrar_alumno(const Alumno &a) {
    cout << "  DNI: " << a.dni << endl;
    cout << "  Notas:";
    for(int x:a.notas)
        cout << " " << x;
    cout << endl;
}

void mostrar_curso(const Curso &c) {
    cout << "Curso " << c.codigo << endl;
    for(size_t i=0;i<c.a.size();i++) {
        cout << "Alumno " << i+1 << endl;
        mostrar_alumno(c.a[i]);
    }
}

int main(int argc, char *argv[]) {
    ...

    mostrar_curso(c);
    ...
}
```

MÁS EJEMPLOS

10. En una competencia de programación regional gana el equipo que más problemas resuelve.

    Si hay empate, gana el que menos "penalización" tenga. La penalización se forma sumando:

    - el tiempo que tardó en resolver cada problema resuelto
    - 20 minutos adicionales por cada intento fallido de un problema resuelto

    Los 2 mejores clasifican para la instancia nacional.

    Escriba una función que reciba los resultados de todos los equipos y retorne los nombres de los clasificados.