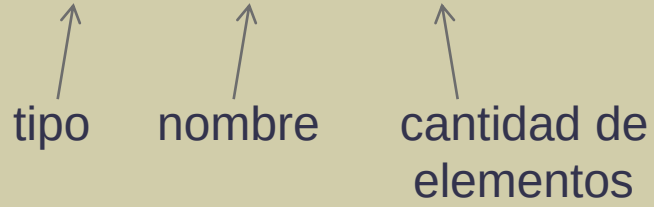
Introducción al Cómputo

Arreglos/vectores et al

Arrays nativos

- Conjunto ordenado de datos de un mismo mismo tipo.
- El conjunto completo está identificado con un nombre.
- A cada dato individual se accede mediante un índice.
- Definición de un array:

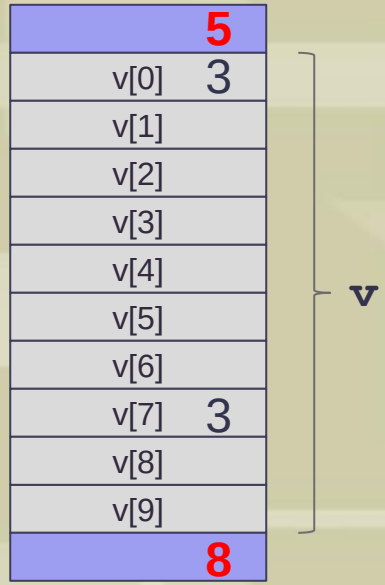
```
int array[50];
```



Arrays nativos: Acceso a elementos

- El índice del primer elemento de un array es 0.
- El índice del último elemento de un array de **`N`** elementos es **`N-1`**.
- C/C++ no verifican límites en los arreglos nativos.

```
int v[10];

v[0] = v[7] = 3;

v[10] = 8;
v[-1] = 5;
```



Arrays nativos: Acceso a elementos

- Acceso a los elementos:

```
int iarray[50];
double darray[100];
int i = 25;

iarray[0]  = 23;
iarray[49] = i;

for( i = 0; i < 100; i++ ) {
    darray[i] = i * sqrt(i);
    cout << i << " " << darray[i] << '\n';
}

i = iarray[0];
iarray[i] = i * i;
```

Arrays nativos de estructuras

- Acceso a los elementos y a campos de elementos:

```
struct Punto2D {
    double x;
    double y;
    void print() { cout << "(" << x << "," << y << ")\n"; }
};
...
Punto2D parray[10];
...
parray[i]  = parray[i+1];  // copia un elemento completo
...
parray[i].x = parray[i+1].x;   // copia el campo "x"
...
parray[i].print();
```

Ejemplo con Arrays nativos

ejem6_1.cpp

```
#include "icom_helpers.h"
// generación de Fibonacci
const int N = 20;

int main(void)
{
    int Fibonacci[N], i;

    Fibonacci[0] = 0;
    Fibonacci[1] = 1;

    for( i = 2; i < N; ++i )
        Fibonacci[i] = Fibonacci[i-2] + Fibonacci[i-1];

    for( i = 0; i < N; ++i )
        cout << Fibonacci[i] << '\n';

    return 0;
}
```

Inicialización de Arrays nativos

```
int noInit[5];

int integers[5]  = { 0, 1, 2, 3, 4 };

char vocales[]   = { 'a', 'e', 'i', 'o', 'u' };

float datos[500] = { 100.0, 300.0, 505.5 };

double x[500] = { [2] = 505.5, [0] = 100.0, [1] = 300.0 };

int main(void)
{
    const int SIZE = 3;
    Punto2D parray[SIZE] = {{2.0, 3.0}, {4.0, 5.0}, {6.0, 7.0}};

    for( int i = 0; i < SIZE; ++i )
        parray[i].print();

    return 0;
}
```

Arrays multidimensionales

```
const int N = 5;

double mat[N][N];
double tr = 0;
int i, j;

for( i = 0; i < N; ++i )
    for( j = 0; j < N; ++j )
        mat[i][j] = (i == j ? 1 : 0);

for( i = 0; i < N; ++i )
    tr += mat[i][i];

for( i = 0; i < N; ++i ) {
    for( j = 0; j < N; ++j )
        cout << mat[i][j] << " ";

    cout << '\n';
}
```

Arrays nativos como argumentos

ejem6_2.cpp

```
#include "icom_helpers.h"

void func(int a[]) {
    cout << "En func &a[0] = " << &a[0] << '\n';
    cout << "En func a     = " << a << '\n';
}

int main(void)
{
    int v[20] = {0};

    cout << "En main &v[0] = " << &v[0] << '\n';
    cout << "En main v     = " << v << '\n';

    func(v);

    return 0;
}
```

```
$ ./ejem6_2
En main &v[0] = 0xffffcbd0
En main v     = 0xffffcbd0
En func &a[0] = 0xffffcbd0
En func a     = 0xffffcbd0
```

Arrays nativos como argumentos

```
double sumaVec(double vec[], int n) {
    double s = 0;
    int i;
    for( i = 0; i < n; i++ )
        s += vec[i];
    return s;
}

int main(void) {
    const int SIZE = 4;
    double val[SIZE] = { 2.3, 3.14, 0.125, 8.9 };

    double result = sumaVec(val, SIZE);

    cout << "Suma: " << result << '\n';

    return 0;
}
```

Arrays nativos como argumentos

```
const int N = 3;
double traza(double mat[N][N]) {
    double tr = 0;
    int i;
    for( i = 0; i < N; i++ )
        tr += mat[i][i];
    return tr;
}

int main(void) {
    double result;
    double mat[][N] = {{2.3, 3.14, 0.12},{2.1, 8.9, 0},
                       {0,   2,    4.56}};
    result = traza(mat);
    cout << "Traza: " << result << '\n';

    return 0;
}
```

std::array

- Entre los problemas de los arreglos nativos podemos citar:
  - Pérdida de información de tamaño del mismo, excepto en el ámbito de su declaración.
  - No chequeo de rango cuando se accede a un elemento por indexado.

- C++ (11) introduce **`std::array`** dentro de sus bibliotecas (tipo no nativo) que provee la funcionalidad de un array de tamaño fijo, pero con facilidades que permiten evitar los problemas anteriores. Este tipo está definido en el header file **`<array>`** (ver **`icom_helper.h`**)

Declaración:

```
array<int, 5> myIntArray;      // declara un array de 5 enteros, llamado myIntArray
array<Punto2D, 10> myP2DArray; // array de 10 Punto2D

array<int, 3> v1 {1, 2, 3};    // declara e inicializa
```

std::array

Acceso a elementos:

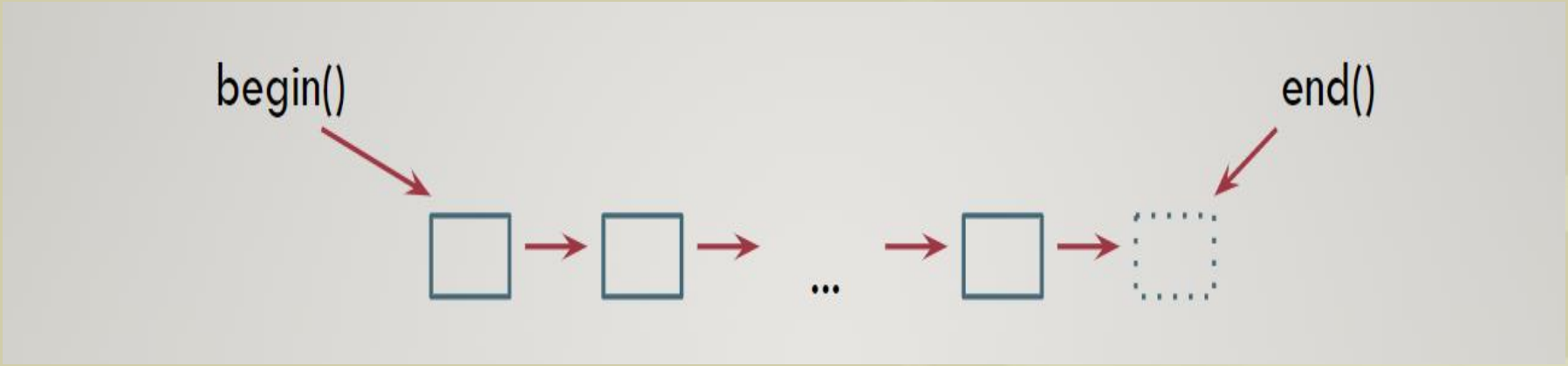
```
array<int, 5> iArray;

iArray[3] = 32;                  // pone 32 en el cuarto element de iArray

// al igual que con los array nativos, no hay chequeo de indices...
// pero hay un método que da acceso chequeado:

iArray.at(4) = 60;
iArray.at(8) = 15;  -> este acceso disparará una excepcion del tipo out_of_range

// es posible recuperar el tamaño a traves del método size()

cout << "Tamaño de iArray: " << iArray.size() << '\n';
```

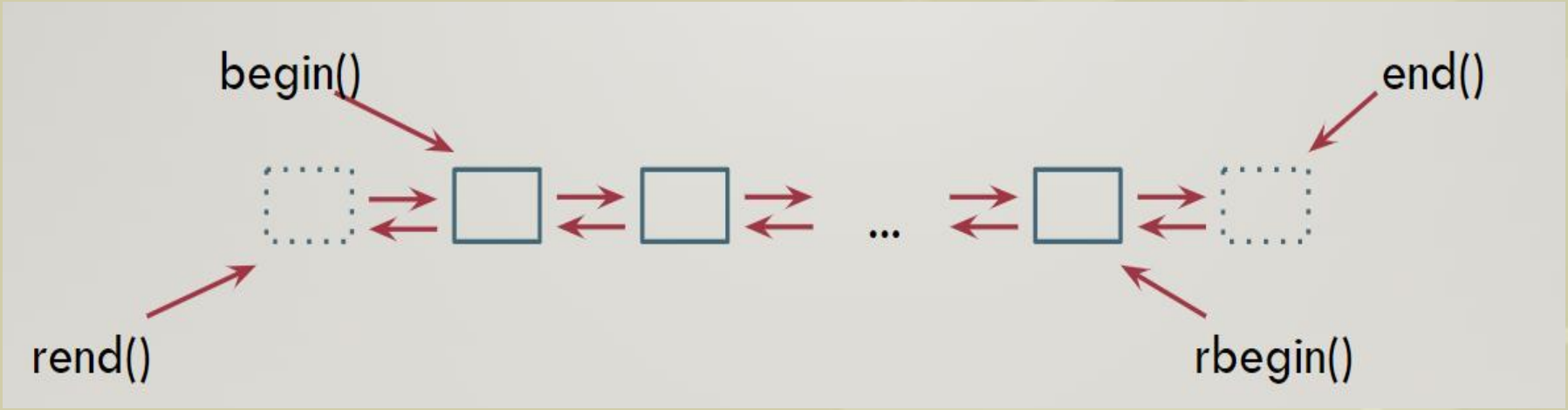
Ejemplo con std::array

Ejem6_3.cpp

```
#include "icom_helpers.h"
// ejemplo holístico con std::array y sort...

int main()
{
    array<int, 5> myArray { 7, 3, 1, 9, 5 };

    sort(myArray.begin(), myArray.end()); // ordena creciente
//    sort(myArray.rbegin(), myArray.rend()); // ordena decreciente

    for (auto element : myArray)
        cout << element << ' ';

    return 0;
}
```

Ejemplo con std::array





std::vector

- C++ introduce **`std::vector`** dentro de sus bibliotecas (tipo no nativo) que prove la funcionalidad de un array dinámico

Declaración:

```
vector<int> myIntVector;          // declara un vector de enteros, llamado myArray
vector<Punto2D> myP2DVector;      // vector de Punto2D

vector<double> v1 {1.1, 2.2, 3.3};       // declara e inicializa
```

Acceso: Igual que con **`std::array`** el acceso se puede hacer a traves del operador **`[]`** en forma no-chequeada, o a traves del método **`at()`** en forma chequeada.

```
double x = v1[1];

double y = v1.at(3);  -> dispara una excepción out_of_range
```

std::vector

Redimensionamiento:

```
vector<int> myIntVector {0, 1, 2};

myIntVector.resize(5);

cout << "El tamaño es: " << myIntVector.size() << '\n';

for (auto v: myIntVector)
        std::cout << v << ' ';
```

```
El tamaño es:  5
0 1 2 0 0
```

Si se agregan elementos, los mismos son inicializados con el valor por defecto para el tipo (0 para enteros)

std::vector

Redimensionamiento:

```
vector<int> v;

v.reserve(128);

for (int i = 0; i < 100; i++)
    v.push_back(i);

cout << "size: " << v.size() << "\n";
cout << "capacity: " << v.capacity() << "\n";
```

Ejemplo con std::vector

ejem6_4.cpp

```
#include "icom_helpers.h"
// eliminar palabras duplicadas, otro ejemplo holístico...
int main()
{
    vector<string> words, w2;
    for (string s; cin >> s && s != "salir"; )
        words.push_back(s);

    sort(words.begin(), words.end());

    if (words.size()) {
        w2.push_back(words[0]);
        for (int i=1; i<words.size(); ++i)
            if(words[i-1]!=words[i])
                w2.push_back(words[i]);
    }
    cout<< "se encontraron " << words.size()-w2.size() <<  " duplicadas\n";
    for (string s : w2)
        cout << s << "\n";
}
```

Replay: Arrays nativos como argumentos

ejem6_2.cpp

```
void func(int a[]) {
    cout << "En func &a[0] = " << &a[0] << '\n';
    cout << "En func a     = " << a << '\n';
    a[4] = 32;
    cout << "En func a[4]     = " << a[4] << '\n';
}

int main(void) {
    int v[20] = {0,1,2,3,4,5};

    cout << "En main &v[0] = " << &v[0] << '\n';
    cout << "En main v     = " << v << '\n';

    func(v);

    cout << "En main v[4]     = " << v[4] << '\n';

    return 0;
}
```

```
En main &v[0] = 0x61fdc0
En main v     = 0x61fdc0
En func &a[0] = 0x61fdc0
En func a     = 0x61fdc0
En func a[4]     = 32
En main v[4]     = 32
```

`std::vector` como argumento

ejem6_5.cpp

```
#include "icom_helpers.h"

void func(vector<int> a) {
    cout << "En func &a[0] = " << &a[0] << '\n';
    cout << "En func &a    = " << &a << '\n';
    a[4] = 32;
    cout << "En func a[4] = " << a[4] << '\n';
}

int main(void){
    vector<int> v {0, 1, 2, 3, 4, 5, 6};
    cout << "En main &v[0] = " << &v[0] << '\n';
    cout << "En main &v    = " << &v << '\n';

    func(v);
    cout << "En main v[4] = " << v[4] << '\n';

    return 0;
}
```

```
En main &v[0] = 0x651730
En main &v    = 0x61fdb0
En func &a[0] = 0x651aa0
En func &a    = 0x61fdf0
En func a[4] = 32
En main v[4] = 4
```

`std::vector` como argumento

ejem6_5.cpp

```
#include "icom_helpers.h"

void func(vector<int> &a) {
    cout << "En func &a[0] = " << &a[0] << '\n';
    cout << "En func &a    = " << &a << '\n';
    a[4] = 32;
    cout << "En func a[4] = " << a[4] << '\n';
}

int main(void){
    vector<int> v {0, 1, 2, 3, 4, 5, 6};
    cout << "En main &v[0] = " << &v[0] << '\n';
    cout << "En main &v    = " << &v << '\n';

    func(v);
    cout << "En main v[4]     = " << v[4] << '\n';

    return 0;
}
```

```
En main &v[0] = 0xe11730
En main &v    = 0x61fdd0
En func &a[0] = 0xe11730
En func &a    = 0x61fdd0
En func a[4] = 32
En main v[4]     = 32
```