# Laboratorio de Programación

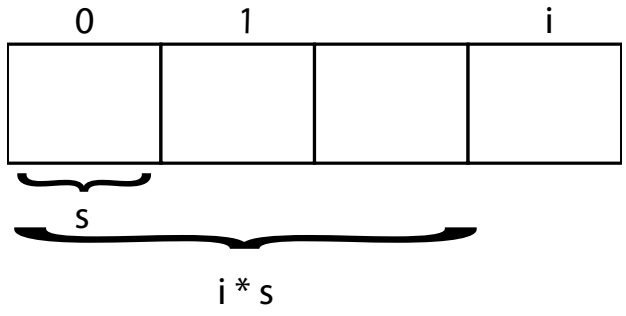
## Parte 2: Manipulación de Vectores en C++

Algoritmos y Estructuras de Datos I

Departamento de Computación, FCEyN, Universidad de Buenos Aires.

# Qué hace C++ internamente

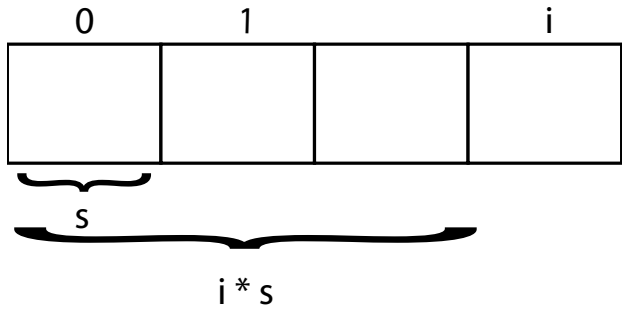
Los elementos de un vector se guardan en una porción de memoria consecutiva que se reserva de forma dinámica durante la ejecución del programa.




- Si un elemento de tipo $T$ ocupa $s$ bytes, entonces el elemento en la posición $i$ se encuentra en la posición $i \times s$ después del inicio.

# Qué hace C++ internamente

Los elementos de un vector se guardan en una porción de memoria consecutiva que se reserva de forma dinámica durante la ejecución del programa.




- Si un elemento de tipo $T$ ocupa $s$ bytes, entonces el elemento en la posición $i$ se encuentra en la posición $i \times s$ después del inicio.
- Luego, obtener un elemento cualquiera tiene un tiempo de ejecución constante, independientemente del tamaño del vector.

# Qué hace C++ internamente

Si los elementos se guardan en forma consecutiva,
¿Qué pasa si agrandamos el vector mediante un `push_back()`?

# Qué hace C++ internamente

Si los elementos se guardan en forma consecutiva,
¿Qué pasa si agrandamos el vector mediante un `push_back()`?

- Si los próximos $s$ bytes NO están reservados por otra entidad del programa (variable, función, etc.) se asignan al vector en cuestión.

# Qué hace C++ internamente

Si los elementos se guardan en forma consecutiva,
¿Qué pasa si agrandamos el vector mediante un `push_back()`?

- Si los próximos $s$ bytes NO están reservados por otra entidad del programa (variable, función, etc.) se asignan al vector en cuestión.
- En caso de que SI estén reservados, no puedo ocupar esta porción de la memoria. Entonces, de forma dinámica, se realizarían las siguientes acciones:
  1. Buscar un bloque de memoria suficiente para albergar la nueva dimensión del vector.
  2. Copiar todos los elementos del vector de la posición de memoria original a la nueva.
  3. Liberar la porción de memoria original.

  Esto, por supuesto, puede acarrear un costo de ejecución proporcional al tamaño del vector.

# Posiciones inválidas

¿Qué ocurre si accedemos a una posición no definida?

```
1  #include <vector>
2  using namespace std;
3
4  int main() {
5    vector<int> cuenta; // <>
6    cuenta.push_back(1); // <1>
7    cuenta[2000] = 10; // ?
8    int valor = cuenta[2000]; // ?
9    return 0;
10 }
```

# Posiciones inválidas

¿Qué ocurre si accedemos a una posición no definida?

```
1  #include <vector>
2  using namespace std;
3
4  int main() {
5    vector<int> cuenta; // <>
6    cuenta.push_back(1); // <1>
7    cuenta[2000] = 10; // ?
8    int valor = cuenta[2000]; // ?
9    return 0;
10 }
```

- C++ no define qué ocurre cuando accedemos a posiciones fuera de rango, el comportamiento está **indefinido**.

# Posiciones inválidas

¿Qué ocurre si eliminamos elementos de un vector vacío?

```
1  #include <vector>
2  #include <iostream>
3  using namespace std;
4
5  int main() {
6    vector<int> v; // <>
7    v.push_back(1); // <1>
8    v.pop_back(); // <>
9    v.pop_back(); // ?
10   cout << "largo de v: " << v.size() << endl;
11   return 0;
12 }
```

# Posiciones inválidas

¿Qué ocurre si eliminamos elementos de un vector vacío?

```
1  #include <vector>
2  #include <iostream>
3  using namespace std;
4
5  int main() {
6    vector<int> v; // <>
7    v.push_back(1); // <1>
8    v.pop_back(); // <>
9    v.pop_back(); // ?
10   cout << "largo de v: " << v.size() << endl;
11   return 0;
12 }
```

- C++ no define qué ocurre cuando queremos eliminar elementos de vectores vacios, el comportamiento está **indefinido**.

# Posiciones inválidas

¿Qué ocurre si eliminamos elementos de un vector vacío?

```
1  #include <vector>
2  #include <iostream>
3  using namespace std;
4
5  int main() {
6    vector<int> v; // <>
7    v.push_back(1); // <1>
8    v.pop_back(); // <>
9    v.pop_back(); // ?
10   cout << "largo de v: " << v.size() << endl;
11   return 0;
12 }
```

- C++ no define qué ocurre cuando queremos eliminar elementos de vectores vacios, el comportamiento está **indefinido**.Que imprimirá el cout??

# Posiciones inválidas

La **precondición** de leer o escribir una posición (`[...]`) es que la posición haya sido previemente definida

- Algunos posibles resultados al leer o escribir una posición fuera de rango en C++:
  - Puede ser correcto
  - Puede dar un error (exception) durante la ejecución
  - Puede generar un `segmentation fault` y terminar la ejecución del programa.
  - Puede colgarse

# Copiar vectores

¿Cómo copiamos un vector a en otro vector b?

Opción 1: Copiar elemento a elemento.

```
1 vector<double> b;
2 for(int i=0; i<a.size(); i=i+1) {
3   b.push_back(a[i]);
4 }
```

# Copiar vectores

¿Cómo copiamos un vector a en otro vector b?

Opción 1: Copiar elemento a elemento.

```
1 vector<double> b;
2 for(int i=0; i<a.size(); i=i+1) {
3   b.push_back(a[i]);
4 }
```

Opción 2: Usar el operador de asignación =.

```
1 vector<double> b;
2 b = a;
```

- Ambas opciones tienen el mismo resultado.

# Funciones con vectores

¿Cómo declaramos una función que retorne un vector?

```
1 vector<int> funcionQueRetornaVector(vector<int> v){
2     vector<int> res;
3     ...
4     return res;
5 }
```

# Funciones con vectores

¿Cómo declaramos una función que retorne un vector?

```
1 vector<int> funcionQueRetornaVector(vector<int> v){
2     vector<int> res;
3     ...
4     return res;
5 }
```

- Internamente: el vector `v` pasado como parámetro se recibe por copia.
- Dentro del código: la función devuelve por el vector `res` declarado dentro de su scope.

# Retorno de vectores (por copia)

```
1  #include <vector>
2  using namespace std;
3
4  vector<int> crearVector(int n) {
5    vector<int> res;
6    for (int i=1; i<=n;i=i+1) {
7      res.push_back(i);
8    }
9    return res;
10 }
11
12 int main() {
13   vector<int> cuenta = crearVector(5); // <1,2,3,4,5>
14   return 0;
15 }
```

# Vectores como parámetros (por copia)

Atención: los vectores, como cualquier parámetro, siempre se reciben por copia a no ser que pidamos lo contrario.

```
1  #include <vector>
2  using namespace std;
3
4  void modificarVector(vector<int> a) {
5    a[0]=35;
6  }
7
8  int main() {
9    vector<int> v(3,10); // <10,10,10>
10   modificarVector(v); // ?
11   return 0;
12 }
```

# Vectores como parámetros (por copia)

Atención: los vectores, como cualquier parámetro, siempre se reciben por copia a no ser que pidamos lo contrario.

```
1  #include <vector>
2  using namespace std;
3
4  void modificarVector(vector<int> a) {
5    a[0]=35;
6  }
7
8  int main() {
9    vector<int> v(3,10); // <10,10,10>
10   modificarVector(v); // ?
11   return 0;
12 }
```

- La función `cambiarVector()` no afecta el estado del vector en el `main()`

# Vectores como parámetros (por referencia)

Para modificar el vector hay que pasarlo por referencia &.

```
1  #include <vector>
2  using namespace std;
3
4  void modificarVector(vector<int>& a) {
5    a[0]=35;
6  }
7
8  int main() {
9    vector<int> v(3,10); // <10,10,10>
10   modificarVector(v); // <35,10,10>
11   return 0;
12 }
```

# Sumar los elementos de una secuencia

$$\text{proc } \mathit{suma}(\text{in } s : \mathit{seq}\langle\mathbb{Z}\rangle, \text{out } \mathit{res} : \mathbb{Z}\ )\{$$
$$\quad \text{Post } \{\mathit{res} = \textstyle\sum_{i=0}^{|s|-1} n[i]\}$$
$$\}$$

Solución usando `while`:

# Sumar los elementos de una secuencia

proc *suma*(in $s : seq\langle\mathbb{Z}\rangle$, out $res : \mathbb{Z}$ ){
  Post $\{res = \sum_{i=0}^{|s|-1} n[i]\}$
}

Solución usando `while`:

```
1  int suma(vector<int> v) {
2    int res = 0;
3    int i =0;
4    while( i < v.size() ) {
5      res = res + v[i];
6      i = i + 1;
7    }
8    return res;
9  }
```

# Calcular promedio de los elementos de una secuencia - uso de `const`

proc *promediar*(in $s : seq\langle\mathbb{R}\rangle$, out $res : \mathbb{R}$ ){

Post $\{res = \frac{\sum_{i=0}^{|s|-1} n[i]\}}{|s|}$

}

Solución usando `while`:

# Calcular promedio de los elementos de una secuencia - uso de `const`

proc *promediar*(in $s : seq\langle\mathbb{R}\rangle$, out $res : \mathbb{R}$ ){

Post $\{res = \frac{\sum_{i=0}^{|s|-1} n[i]\}}{|s|}$

}

Solución usando `while`:

```
1  float promediar(vector<float> & v) {
2    float res = 0;
3    int i =0;
4    while( i < v.size() ) {
5      res = res + v[i] / v.size();
6      i = i + 1;
7    }
8    return res;
9  }
```

# Calcular promedio de los elementos de una secuencia - uso de `const`

proc *promediar*(in $s : seq\langle\mathbb{R}\rangle$, out $res : \mathbb{R}$ ){

Post $\{res = \frac{\sum_{i=0}^{|s|-1} n[i]\}}{|s|}$

}

Solución usando `while`:

# Calcular promedio de los elementos de una secuencia - uso de `const`

proc *promediar*(in $s : seq\langle\mathbb{R}\rangle$, out $res : \mathbb{R}$ ){

Post $\{res = \frac{\sum_{i=0}^{|s|-1} n[i]\}}{|s|}$

}

Solución usando `while`:

```
1  float promediar(vector<float> & v) {
2    float res = 0;
3    int i =0;
4    while( i < v.size() ) {
5      res = res + v[i] / v.size();
6      i = i + 1;
7      // error a proposito!!
8      v[0] = i;
9    }
10   return res;
11 }
```

# Calcular promedio de los elementos de una secuencia - uso de `const`

```
1  float promediar(const vector<float> & v) {
2    float res = 0;
3    int i =0;
4    while( i < v.size() ) {
5      res = res + v[i] / v.size();
6      i = i + 1;
7      // error a proposito!!
8      v[0] = i;
9    }
10   return res;
11 }
```

El `const` convierte al vector a "solo-lectura", con lo cual no se puede utilizar del lado izquierdo de un operador de asignación. Esto permite ser robusto a posibles errores.

# Resumen: Vectores en C++

| | |
|---|---|
| `vector<int> v;` | Declara un vector sin elementos |
| `vector<int> v(n);` | Declara un vector con `n` elementos |
| `vector<int> v(n,x);` | Declara un vector con `n` elementos con el valor `x` |
| `v.size();` | Informa la longitud del vector |
| `v.push_back(x);` | Almacena el valor `x` al final del vector |
| `v.pop_back();` | Elimina la última posición del vector |
| `int a = v[i];` | Lee la posición `i` del vector `v` (y la guarda en `a`) |
| `v[i] = x;` | Reemplaza la posición `i` del vector con el valor `x` |
| `v1 = v2 ;` | Vacía `v1` y copia en `v1` todos los elementos de `v2` (`v1` y `v2` deben tener el **mismo tipo**). |