Vectores

Curso de Estructura de datos y algoritmos 1 – Universidad ORT Uruguay

Conceptos generales

1. ¿Qué son los vectores?

Se le llama vectores a un conjunto de memoria lineal y continua que contiene datos de un mismo tipo, en particular nos referimos a un array unidemensional. Cada dato almacenado en el vector tiene una posición asociada. Sabiendo esa posición, se puede acceder un dato específico dentro del vector.

En C la cantidad de elementos que podrá contener un vector es fijo, y en principio se define cuando se declara el vector. Los vectores se pueden declarar de la siguiente forma:

```
tipo_elemento nombre[largo];
```

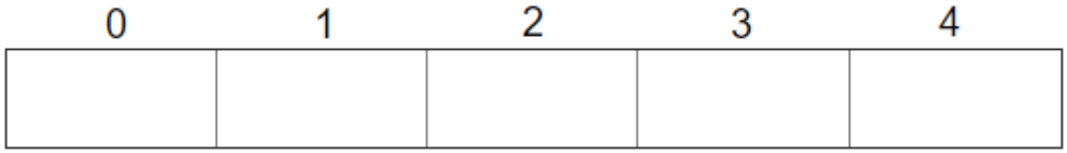
Esto declara la variable ***nombre*** como un vector de ***tipo_elementos,*** que podrá contener ***largo*** cantidad de elementos, y cada uno de estos elementos podrá contener un valor de tipo ***tipo_elemento***.

Por ejemplo:

```
int vectorDeEnteros[5];
```

En el ejemplo se declaró un vector que puede contener hasta 5 enteros.

Las posiciones empiezan a contar desde 0, por lo que, en el ejemplo anterior, se podrían acceder desde las posiciones 0 a 4 inclusive:



2. Vectores Estáticos

¿Se podría crear un vector con **largo dinámico** (que se asigne durante la ejecución) utilizando la sintaxis anterior?
La respuesta es **NO.**

Por lo que el siguiente código no compilaría:

```
int largo = 5;
int vectorDeEnteros[largo];
```

Sin embargo, si es posible hacerlo si la variable es **constante**.

Hay dos formas de hacerlo:

1.
```
int main(){
   const int largo = 5;
   int vectorDeEnteros[largo];
}
```

2.
```
#define LARGO 5

int main() {
   int vectorDeEnteros[LARGO];
}
```

Se busca utilizar la segunda opción ya que se le asigna el valor a la variable *LARGO* antes de compilar.

Veremos más sobre la directiva *#define* más adelante.

¿Cómo asignar datos al vector?
Es posible hacerlo como en los siguientes ejemplos:

1.
```
#define LARGO 5

int main(){
   int vectorDeEnteros[LARGO] = {10, 24, 3, 1, 12}
}
```

2.
```
int main(){
   int vectorDeEnteros[] = {10, 24, 3, 1, 12}
}
```

3.
```
#define LARGO 5

int main(){
   int vectorDeEnteros[LARGO];
   vectorDeEnteros[0] = 10;
   vectorDeEnteros[1] = 24;
   vectorDeEnteros[2] = 3;
   vectorDeEnteros[3] = 1;
   vectorDeEnteros[4] = 12;
}
```

3. Vectores Dinámicos

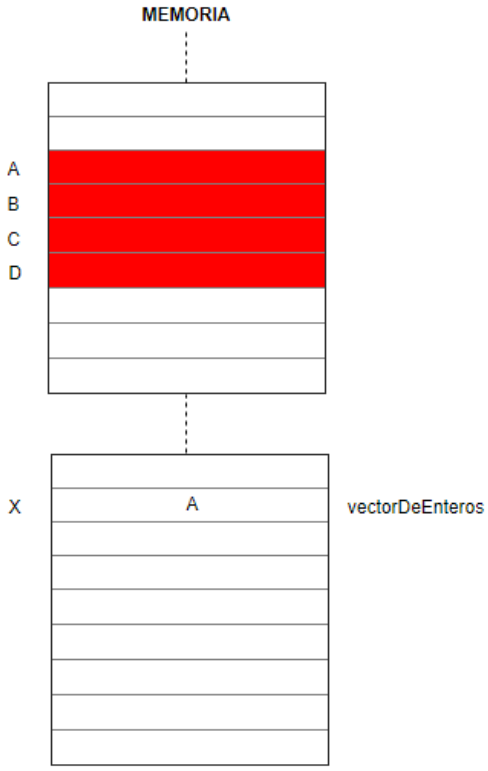
Para lograr declarar un vector con un largo dinámico, es necesario utilizar **punteros**:

```
int main(){
   int largo = 4;
   int* vectorDeEnteros = new int[largo];
}
```

Lo que hace el código mostrado es:

- **Reservar memoria** para 4 enteros, siendo 4 direcciones de memoria contiguas.
- Asignar la **primera direccion de memoria** reservada a la variable *vectorDeEnteros*.

En la siguiente ilustración, se hará una representación (a grandes rasgos) de como es utilizado el puntero para crear y utilizar un vector.



Esta tabla representaría la memoria de nuestro sistema y cada celda es un lugar en la memoria donde se guardará un valor (las líneas suspensivas están para representar que hay muchas más celdas que las indicadas).

Siguiendo el ejemplo anterior, al declarar la variable *vectorDeEnteros*, se asigna el lugar de memoria ***X*** para esa variable.

Al asignarle a esa variable ***new int[largo]*** donde ***largo*** es 4, se reservan 4 lugares en la memoria (marcados en rojo), donde cada lugar **debe** contener un entero, ya que se hizo *new **int***, entonces se reservan lugares de memoria para enteros.

Como *vectorDeEnteros* es un **puntero** debe guardar un **lugar de memoria**, en este caso, como mencionamos anteriormente, guarda el primer lugar de la memoria reservada, o sea, ***A.***

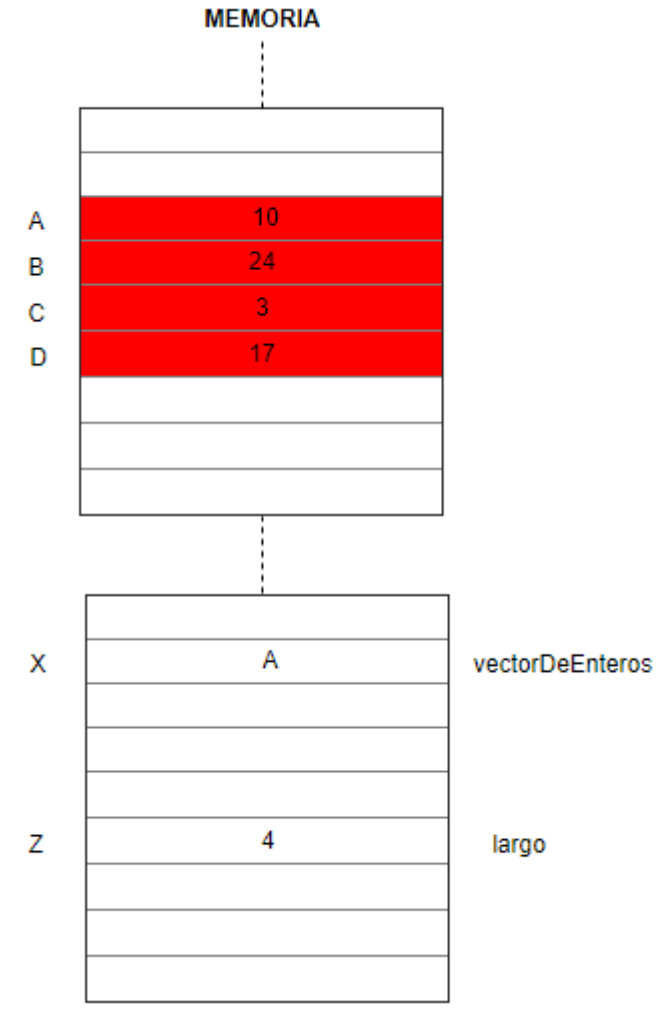
¿Cómo esto podría ser un vector?
El lenguaje nos permite usar la sintaxis de vector para los punteros (la misma de java).
Por lo que, si queremos acceder a un valor de una posición dada, la sintaxis es:

```
vectorDeEnteros[0]
```

Continuando con el ejemplo anterior, podemos asignar valores a cada posición del array de la siguiente manera:

```
int main(){
      int largo = 4;
      int* vectorDeEnteros = new int[largo];
      vectorDeEnteros[0] = 10;
      vectorDeEnteros[1] = 24;
      vectorDeEnteros[2] = 3;
      vectorDeEnteros[3] = 17;
}
```

Una representación de como afectaría la memoria este código seria la siguiente:



Si quisiéramos mostrar todos los elementos del vector, podríamos hacer lo siguiente:

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

int main() {
	int largo = 4;
	int* vectorDeEnteros = new int[largo];
	vectorDeEnteros[0] = 10;
	vectorDeEnteros[1] = 24;
	vectorDeEnteros[2] = 3;
	vectorDeEnteros[3] = 17;

	for (int i = 0; i < largo; i++) {
		cout << vectorDeEnteros[i] << endl;
	}
}
```

Cuando accedemos a la posición ***i*** de un vector, lo que nos devuelve es el valor que se encuentra ***i*** posiciones por debajo del primer lugar de memoria reservado.

IMPORTANTE: ***SI RESERVO, LIBERO***.
Cada vez que usamos la palabra clave ***new***, significa que estamos reservando memoria.
En el ejemplo anterior, reservamos 4 lugares de memoria. Al reservar esos 4 lugares de memoria, lo que nos asegura el lenguaje es que esos 4 lugares de memoria no van a ser utilizados por otro proceso.

Entonces, luego de utilizar la variable, hay que **liberar** esa memoria reservada para que pueda volver a ser utilizada.

Esto se logra mediante la palabra clave ***delete[] nombre_variable;***

Siguiendo el ejemplo anterior:

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

int main() {
	int largo = 4;
	int* vectorDeEnteros = new int[largo];
	vectorDeEnteros[0] = 10;
	vectorDeEnteros[1] = 24;
	vectorDeEnteros[2] = 3;
	vectorDeEnteros[3] = 17;

	for (int i = 0; i < largo; i++) {
		cout << vectorDeEnteros[i] << endl;
	}
	delete[] vectorDeEnteros;
}
```

No siempre es necesario liberar la memoria dentro de la misma función, a veces la responsabilidad de la función es devolver un vector nuevo (reservando memoria), pero no es su responsabilidad liberar esa memoria, sino de quien la utiliza.

Por ejemplo:

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

//PRE: largo > 0
//POS: Retorna un nuevo vector de tamaño 'largo'
int* crearVector(int largo){
	int* nuevoVector = new int[largo];
	return nuevoVector;
}

int main() {
	int largo = 4;
	int* vectorDeEnteros = crearVector(largo);
	vectorDeEnteros[0] = 10;
	vectorDeEnteros[1] = 24;
	vectorDeEnteros[2] = 3;
	vectorDeEnteros[3] = 17;

	for (int i = 0; i < largo; i++) {
		cout << vectorDeEnteros[i] << endl;
	}

	delete[] vectorDeEnteros;
}
```

En este caso, el ***new*** se hizo en la función ***crearVector***, pero ésta no la liberó porque el vector no se utiliza en esa función, se utiliza en el ***main***, entonces es éste quien debe encargare de liberarla.

NOTA:
No es posible obtener el largo de un vector, por lo que cada vez que una función recibe un vector, también debe recibir su largo.

Ejercicios

1. Implementar una función *sumaTotal* que, dado un vector de enteros y su largo, retorne la suma de todos los elementos.
   Ejemplo:
   [10,20,30,40,50] -> 150
   [10, 40] -> 50
   [5, -10, 2, 3] -> 0

   ```
   //PRE: Recibe un vector de enteros y su largo
   //POS: Retorna la suma de todos los elementos de 'vec'
   int sumaTotal(int *vec, int largo);
   ```

2. Implementar una función *maximo* que, dado un vector de enteros y su largo, retorne el máximo elemento del vector.

   ```
   //PRE: Recibe un vector de enteros y su largo
   //POS: Retorna el máximo número del vector
   int maximo(int *vec, int largo);
   ```

3. Implementar una función *cantidadImpares* que, dado un vector de enteros y su largo, retorne la cantidad de números impares en el vector

   ```
   //PRE: Recibe un vector de enteros y su largo
   //POS: Retorna la cantidad de impares en 'vec'
   int cantidadImpares(int *vec, int largo);
   ```

4. Implementar una función *obtenerImpares* que, dado un vector de enteros y su largo, retorne un **nuevo** vector de enteros solamente con los impares del vector.
   Ejemplo:
   [1,2,3,4,5,6] -> [1,3,5]
   [10, 3, 15, 2, 9] -> [3,15,9]
   [2,4,6,8,10] -> NULL

   SUGERENCIA: Utilizar la función anterior para este ejercicio.

   ```
   //PRE: Recibe un vector de enteros y su largo
   //POS: Retorna un nuevo vector con los impares de `vec`.
   int* obtenerImpares(int *vec, int largo);
   ```