Memoria dinámica

Reservar memoria

```
new tipo_de_dato(valor_a_asignar);
```

si tipo_de_dato es un struct (los valores deben ir en órden):

```
new tipo_de_dato {valor_variable_1, valor_variable_2, ... };
```

Ver nodo en ListaEnlazada para un ejemplo.

Liberar memoria

```
delete variable_puntero;
```

Es una buena práctica asignar el puntero a "nullptr" luego de liberar la memoria.

Puntero a struct o class

Lo siguiente:

```
(*puntero_a_instancia).metodo() (o variable interna)
```

es equivalente a:

```
puntero_a_instancia->metodo() (o variable)
```

Templates

Para hacer una clase paramétrica:

```
template <typename T>
class NombreDeLaClase {...
```

Esta clase se puede instanciar como;

```
NombreDeLaClase<tipo_de_dato> mi_instancia(...
```

Luego podemos escribir la implementación de un método que toma un tipo genérico T:

```
template <typename T>
tipo_de_dato NombreDeLaClase<T>::nombre_del_metodo(T nombre_param) {...
```

Nota: la definición de un template debe estar en archivos .h, no es posible separar el código en definición (.h) e implementación (.cpp) como acostumbramos.