Clase 13/6

**Nota:** ASCII Math | Programa para cálculos matemáticos.  
 Latex | Programa para cálculos matemáticos.

Tuplas(También conocido como “par ordenado”):

Operación acceso: es la típica operación realizada sobre una tupla.

Por posición: base+offset p0,p1   
 Por nombre: px,py

Terminología en programación:

C++: Struct.  
Pascal: Record.  
  
---------------------------------------------------------------------

Struct Punto {  
double x, y;  
};  
  
Punto p;  
p.x //acceso al elemento ‘a’ perteneciente a la struct T.

---------------------------------------------------------------------

Struct Punto {  
 double x, y;  
 };  
  
 Punto p{7, 13};  
 cout << p.x //Aparece “7” en pantalla  
  
 p.x = 4;  
  
---------------------------------------------------------------------

Una estructura podría tener n objetos, las cuales a su vez podrían tener m estructuras dentro de cada ‘enésimo’ objeto.

* En una invocación, se copia el objeto completo en el parámetro de la función y se utiliza ese parámetro. Esto hace que no importe si el objeto y el parámetro tienen nombres diferentes.
  + Esto trae dos problemas: mayor tiempo y mayor espacio.

Para evitar esto, podemos **pasar el objeto por referencia.**

En la invocación:  
void EnviarPunto(Punto&) //Acá espera una referencia a Punto.  
  
En la definición:  
  
 void EnviarPunto(Punto& unPunto) { … }

De esta manera ganamos tiempo porque ‘unPunto’ hace referencia o “apunta”(no confundir con puntero) a la variable “p” declarada anteriormente.

Esto **nos permite modificar variables de manera global**, ya que de la otra manera la modificación de la variable sería local y no tendría ningún efecto en el main.

Si queremos pasar una referencia **pero no queremos que sea modificable**, debemos declarar a la variable como un tipo ‘*const*’.

En la invocación:  
void EnviarPunto(**const** Punto&) //Acá espera una referencia a Punto de valor constante.  
  
En la definición:  
  
 void EnviarPunto(**const** Punto& unPunto) { … }

De esta manera, obtenemos la rapidez y eficiencia de pasar por referencia, y logramos que no se modifique el valor dentro de la función establecida.

Si queremos conocer la dirección de memoria de una variable, debemos mostrar en pantalla lo siguiente:

std::cout << **&p**; //Esto nos imprime una dirección de memoria en Hexadecimal

Si queremos conocer el tamaño en bytes de las variables que estamos manejando entonces escribimos:

std::cout << **sizeof** p; //Tamaño de la variable p  
 std::cout << **sizeof** p.x; //Tamaño de la variable p  
 std::cout << **sizeof** p.y; //Tamaño de la variable p

p

p.x

p.y