CONCEPTOS BÁSICOS CLAVES RESUMIDOS

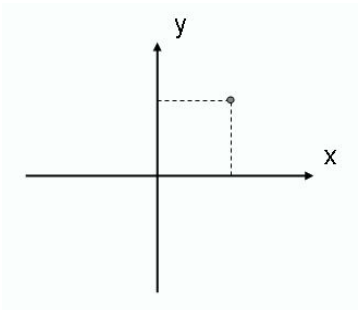
Todos los conceptos expuestos son tomados de manera muy resumida y machetera de los ejemplos de: The GNU C Reference Manual. Se colocan a modo de tabla (con traducciones mal hechas, ahi disculparan) con el objetivo de rapida consulta. La idea es que comprendan los ejemplos y los sepan relacionar con la reference card de C (vendida previamente en el laboratorio por $200). O si necesitan una versión mas completa pueden ver el siguiente enlace:

Arrays	
Declaring Arrays	`int my_array[10];`
Initializing Arrays	`int my_array[5] = { 0, 1, 2, 3, 4 };` `int my_array[5] = { 0, 1, 2 };` `/*  ISO C99, or C89 with GNU extensions */` `int my_array[5] = { [2] 5, [4] 9 }; // int my_array[5] = { 0, 0, 5, 0, 9 };` `int my_array[5] = { [2] = 5, [4] = 9 }; // int my_array[5] = { 0, 0, 5, 0, 9 };` `/* GNU extensions → [first] ... [last] */` `int new_array[100] = { [0 ... 9] = 1, [10 ... 98] = 2, 3 };` `int my_array[] = { 0, 1, 2, 3, 4 };` `int my_array[] = { 0, 1, 2, [99] = 99 };`
Accessing Array Elements	`my_array[0] = 5;`
Multidimensional Arrays	
Declaring Multidimensional Arrays	`int two_dimensions[2][5] { {1, 2, 3, 4, 5}, {6, 7, 8, 9, 10} };`
Declaring Multidimensional Arrays	`two_dimensions[1][3] = 12;`

Arrays as Strings	
Initializing Strings	`char blue[26];` `char yellow[26] = {'y', 'e', 'l', 'l', 'o', 'w', '\0'};` `char orange[26] = "orange";` `char gray[] = {'g', 'r', 'a', 'y', '\0'};` `char salmon[] = "salmon";` **Warning**: After initialization, you cannot assign a new string literal to an array using the assignment operator. For example, this will not work: `char lemon[26] = "custard";` `lemon = "steak sauce";  /* Fails! */`
Accessing individual string	`char name[] = "bob";` `name[0] = 'r';  // Accessing individual string`

Pointers	
Declaring Pointers	`int *ip;` `int *foo, *bar;  /* Two pointers. */` `int *baz, quux;  /* A pointer and an integer variable. */`
Initializing Pointers	`int i;` `int *ip = &i;`
	`int i, j;` `int *ip = &i;  /* 'ip' now holds the address of 'i'. */` `ip = &j;  /* 'ip' now holds the address of 'j'. */` `*ip = &i;  /* 'j' now holds the address of 'i'. */`

Estructuras

Defining Structures	 `struct point` `  {` `    int x, y;` `  };` That defines a structure type named **`struct point`**, which contains two members, **`x`** and **`y`**, both of which are of type int.
Declaring Structure Variables	**Declaring Structure Variables at Definition** `struct point` `  {` `    int x, y;` `  } first_point, second_point;` **Declaring Structure Variables After Definition** `struct point` `  {` `    int x, y;` `  };` `struct point first_point, second_point;`
Initializing Structure Members	**Caso 1**: `struct point`

```
  {
    int x, y;
  };
struct point first_point = { 5, 10 };
```

Caso 2:

```
struct point first_point = { .y = 10, .x = 5 };
```

Caso 3:

```
struct point first_point = { y: 10, x: 5 };
```

Caso 4:

```
struct point
  {
    int x, y;
  } first_point = { 5, 10 };
```

Otros casos:

Caso en el que no todos los miembros son inicializados:

```
struct pointy
  {
    int x, y;
    char *p;
  };
struct pointy first_pointy = { 5 };  // Here, x is initialized with 5, y is initialized with 0, and p is initialized
with NULL.
```

Inicializando una estructura donde los miembros son otras estructura:

```
struct point
  {
    int x, y;
```

	`` }; struct rectangle { struct point top_left, bottom_right; }; struct rectangle my_rectangle = { {0, 5}, {10, 0} }; ```
Accessing Structure Members	Con el operador punto (.) muy similar a como se hace en lenguajes como java. ``` struct point { int x, y; }; struct point first_point; first_point.x = 0; first_point.y = 5; ``` Cuando una estructura posee otra estructura como miembro: ``` struct rectangle { struct point top_left, bottom_right; }; struct rectangle my_rectangle; my_rectangle.top_left.x = 0; my_rectangle.top_left.y = 5; my_rectangle.bottom_right.x = 10; my_rectangle.bottom_right.y = 0; ```

<table>
<tr><th colspan="2">Combinando conceptos</th></tr>
<tr><td>Arrays of Structures</td><td>

creating an array of a structure type

```
struct point
  {
    int x, y;
  };
struct point point_array [3];
```

Creacion e inicializacion de un array de 3 elementos tipo struct point

```
struct point point_array [3] = { {2, 3}, {4, 5}, {6, 7} };
```

Otra forma:

```
struct point point_array [3] = { {2}, {4, 5}, {6, 7} };
```

Accediendo a los miembros despues de la inicialización:

```
struct point point_array [3];
point_array[0].x = 2;
point_array[0].y = 3;
```

</td></tr>
<tr><td>Pointers to Structures</td><td>

Creando un puntero a una estructura:

```
struct fish
  {
    float length, weight;
  };
struct fish salmon = {4.3, 5.8};
struct fish *fish_ptr = &salmon;
```

Accediendo a los miembros: En vez de el operador punto (.) emplado para variables normales se usa el operador flecha cuando la variable asociada a la estructura es un puntero.

</td></tr>
</table>

	`fish_ptr -> length = 5.1;` `fish_ptr -> weight = 6.2;`
Funciones y estructuras	// Supongase la siguiente estructura. `struct foo` `{` `  int x;` `  float y;` `  double z;` `};` // Una funcion como la siguiente toma un apuntador (a para el caso) a esta estructura para hacer lo que tenga que hacer. A cntinuacion se muestra la declaracion (cabecera) `void bar (const struct foo *a);`
Funciones y arrays	**Forma 1:** `// Declaracion` `void foo (int a[]);` ... `// Invocacion` `int x[100];` `foo (x);` **Forma 2**: `// Declaracion` `void foo (int *a);` ... `// Invocacion` `int x[100];` `foo (x);`