Punteros

Parte 2: Aritmética, Vectores y Estructuras

Aritmética de punteros

Sobre las variables de tipo puntero es posible utilizar los operadores +, -, ++ y --.

Estos operadores incrementan o decrementan la posición de memoria a la que “apunta” la variable puntero.

El incremento o decremento se realiza de acuerdo al tipo base de la variable de tipo puntero, de ahí la importancia del tipo del que se declara.

Aritmética de punteros

Cada operación sobre el puntero se rige por su tipo:

```
tipo* puntero;
tipo variable[10];
puntero = &variable[0];
puntero = puntero + incremento;
```

dirección que contiene “puntero” + (incremento * sizeof(tipo))

Uso de aritmética de punteros

	Posición Actual	puntero++;	puntero - -;	puntero += 2;	puntero -= 3;
char* puntero	0xF000A080	0xF000A081	0xF000A07F	0xF000A082	0xF000A07D
int* puntero	0xF000B080	0xF000B084	0xF000B07C	0xF000B088	0xF000B06E
float* puntero	0xF000C080	0xF000C084	0xF000C07C	0xF000C088	0xF000C06E

Uso de aritmética de punteros

	Posición Actual	puntero++;	puntero - -;	puntero += 2;	puntero -= 3;
char* puntero	0xF000A080	0xF000A081	0xF000A07F	0xF000A082	0xF000A07D
int* puntero	0xF000B080	0xF000B084	0xF000B07C	0xF000B088	0xF000B06E
float* puntero	0xF000C080	0xF000C084	0xF000C07C	0xF000C088	0xF000C06E

sizeof(int) == 4

Vectores y punteros

Existe una estrecha relación entre los punteros y los vectores, ya que el nombre de un vector se puede utilizar como un puntero y un puntero puede ser utilizado como un vector, teniendo en consideración que un vector es en realidad un puntero constante y por lo tanto no se puede alterar su valor.

Vectores y punteros

```
char    arrayChar[ ] = “HOLA MUNDO”;
```

Dirección de Memoria	Valor
0x00C0	‘H’
0x00C1	‘O’
0x00C2	‘L’
0x00C3	‘A’
0x00C4	0x20
0x00C5	‘M’
0x00C6	‘U’
0x00C7	‘N’
0x00C8	‘D’
0x00C9	‘O’
0x00CA	‘\0’

Vectores y punteros

```
char    arrayChar[ ] = “HOLA MUNDO”;
char*   punteroChar;
punteroChar = arrayChar;
```

Dirección de Memoria	Valor
0x00C0	‘H’
0x00C1	‘O’
0x00C2	‘L’
0x00C3	‘A’
0x00C4	0x20
0x00C5	‘M’
0x00C6	‘U’
0x00C7	‘N’
0x00C8	‘D’
0x00C9	‘O’
0x00CA	‘\0’

Vectores y punteros

```
char    arrayChar[ ] = “HOLA MUNDO”;
char*   punteroChar;
punteroChar = arrayChar;
printf(“Dir: %p”, punteroChar);
```

>Dir: 0x00C0

Dirección de Memoria	Valor
0x00C0	‘H’
0x00C1	‘O’
0x00C2	‘L’
0x00C3	‘A’
0x00C4	0x20
0x00C5	‘M’
0x00C6	‘U’
0x00C7	‘N’
0x00C8	‘D’
0x00C9	‘O’
0x00CA	‘\0’

Vectores y punteros

```
char    arrayChar[ ] = “HOLA MUNDO”;
char*   punteroChar;
punteroChar = arrayChar;
punteroChar[ 4 ] = ‘@’ ; // => “HOLA@MUNDO”
```

Dirección de Memoria	Valor
0x00C0	‘H’
0x00C1	‘O’
0x00C2	‘L’
0x00C3	‘A’
0x00C4	‘@’
0x00C5	‘M’
0x00C6	‘U’
0x00C7	‘N’
0x00C8	‘D’
0x00C9	‘O’
0x00CA	‘\0’

Vectores y punteros

```
char    arrayChar[ ] = “HOLA MUNDO”;
char*   punteroChar;
punteroChar = arrayChar;
punteroChar[ 4 ] = ‘@’ ; // => “HOLA@MUNDO”
*(arrayChar+4) = ‘#’ ; // => “HOLA#MUNDO”
```

Dirección de Memoria	Valor
0x00C0	‘H’
0x00C1	‘O’
0x00C2	‘L’
0x00C3	‘A’
0x00C4	‘#’
0x00C5	‘M’
0x00C6	‘U’
0x00C7	‘N’
0x00C8	‘D’
0x00C9	‘O’
0x00CA	‘\0’

Punteros y estructuras

Un puntero puede apuntar a una estructura y acceder a sus campos.

```
struct Dato x;
struct Dato *ptr;
...
ptr = &x;
(*ptr).campo1 = 33;
```

El operador flecha ->

Para hacer menos incómodo el trabajo con punteros a estructuras, C tiene el operador flecha.

```
struct Dato x;
struct Dato *ptr;
...
ptr = &x;
ptr->campo1 = 33; // (*ptr).campo1 = 33;
```

Punteros como parámetro de funciones

Existen casos donde se requiere que una función modifique una variable que no pertenece a ella, es decir, que se encuentra fuera de su ámbito.

Como en C solo se puede pasar el valor de una variable a una función, debemos pasar como parámetro la dirección de memoria para luego en la función utilizar el operador de indirección (*) a fin de acceder al contenido de la variable original y poder así leerla/modificarla.

```
#define LONGITUD_ARRAY 100
```

```
#define LONGITUD_ARRAY 100
void initArray(int* array , int longitud );
```

```
#define LONGITUD_ARRAY 100
void initArray(int* array , int longitud );

void main()
{
	int auxArrayInt [LONGITUD_ARRAY] ;
	initArray(auxArrayInt, LONGITUD_ARRAY);
	…
}
```

```c
#define LONGITUD_ARRAY 100
void initArray(int* array , int longitud );

void main()
{
	int auxArrayInt [LONGITUD_ARRAY] ;
	initArray(auxArrayInt, LONGITUD_ARRAY);
	…
}
void initArray(int* array , int longitud )
{
	int i;
	for(i=0; i < longitud ; i++)
	{
		*(array + i) = 0;
		// Tambien es valido  array[i] = 0;
	}
}
```