Trabajo Práctico

Grupo Nº 27:

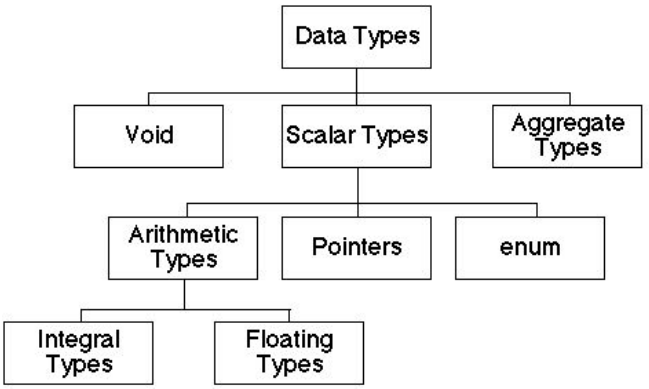
- Pablo Kepes 14726/9
- Santiago Farias 14379/1
- Martin Tribiño 03736/2
- Joaquin Diaz Churria 16225/0

Lenguajes: C y PHP

A. Tipos de datos en C:

Un tipo de datos agregado consta de uno o más tipos de datos escalares. Puede declarar los siguientes tipos de datos agregados:

- Array
- Union
- Struct



En PHP, los tipos de datos son:

1. 4 tipos escalares:
    a. boolean $variable = True;
    b. integer $variable = 1;
    c. float $variable = 1.234;
    d. string $variable = “valor”;
2. 4 tipos compuestos:
    a. array $variable = array(“key” => “val”);
    b. Object Class ClassName { }
    c. Callable call_user_func(“funcion de retorno”);
    d. iterable
3. 2 tipos especiales:
    a. resource
    b. NULL $variable = NULL;

PHP no requiere (ni soporta) la definición explícita de tipos en la declaración de variables; el tipo de la variable se determina por el contexto en el cual se emplea la variable, PHP usa tipado dinámico. Es decir, una variable será del tipo de su contenido en cualquier momento.

En C el tipado es estático, lo cual obliga a definir desde el principio el tipo de una variable.

El forzado de tipos (type casting) en PHP funciona de la misma manera que en C:

```
$var = 10;
$res = (boolean) $var;
```

En C cuando cualquier valor escalar se convierte a _Bool, el resultado es 0 si el valor se compara igual a 0; de lo contrario, el resultado es 1.

```
Int var =10;
_Bool res=(_Bool) var;
```

En la variable **res** queda el valor que se interpreta como **true** en ambos casos

PHP es un lenguaje de script ejecutado del lado del servidor con tipado dinámico y que originalmente no tenía soporte para programación orientada a objetos. No soporta clases abstractas o interfaces, destructores o controles de acceso para miembros de clase.

Diferencias notables:

	C	PHP
Paradigmas	Procedural y estructural	Funcional y Orientado a objetos
Tipo de lenguaje	De proposito general.	De script, orientado a web.
Tipado	Estático	Dinámico
Ambito variables	Global, local	Global, local
Valores por defecto en parámetros formales	No	Si
Chequeo de tipos en parámetros	La versión original no requería.	Si

Punteros	Si	Solo referencias.
IO y llamadas de sistema	Si	No

B. Cuando hablamos de sintaxis, en C se debe tomar en cuenta que es un lenguaje fuertemente tipado, esto quiere decir que para poder utilizar una variable está debe estar declarada previamente, además el nombre de la variable debe comenzar con una letra o un guión bajo.
Al analizar la sintaxis de PHP, nos encontramos un lenguaje débilmente tipado: No es necesario declarar el tipo de variable que es, cambia de acuerdo a la sentencia en la que es usada y para reconocerse como una variable el nombre debe comenzar con un signo dólar ($) seguido del nombre de ésta. ($var)

Tanto C como PHP tienen varias similitudes a la hora de analizar estos lenguajes sintácticamente, por ejemplo, la sintaxis de la estructura de control de selección múltiple es similar (también en ambos si no se encuentra la sentencia break al final del segmento seleccionado, se continua por el siguiente), las instrucciones se separan con ; y los comentarios se escriben de igual forma. Por ejemplo:

En C:
```
/* estructura de selección múltiple */
switch ( expresion ) {
  case const1:
     Bloque1;
     [break;]
   …
  case constn:
     bloquen;
     [break;]
   [default: sentencia_default] }

/* estructura de repeticion for */
for ( var:=Exp; condición; incrementado ) {
   Código; }

/* estructura if. En ambos el else es opcional */
if ( a == b ) { printf("es igual"); }
if ( a == b ) { printf("es igual");
} else { printf("es distinto"); }
```

En PHP:
```
/* estructura de selección múltiple */
switch ( expresion ) {
  case const1:
     Bloque1;
     [break;]
   …
  case constn:
     bloquen;
     [break;]
   [default: sentencia_default] }

/* estructura de repeticion for */
for ($var:=Exp; condición; incrementado){
   Código; }

/* estructura if. En ambos el else es opcional */
if ( $a == $b ) { echo "es igual"; }
if ( $a == $b ) { echo "es igual";
} else { echo "es distinto"; }
```

Como así también los operadores:

- Aritmeticos (Suma +,Resta -,Multiplicación *,División /,Resto o módulo %)
- Lógicos (And &&,Or ||,Not !)
- Relacionales(Igual ==,Distinto !=,Menor <,Mayor >,Menor o igual <=,Mayor o igual >=)
- a nivel bit (O inclusivo |,O exclusivo ^,Not ~, Desplazar a izquierda <<, Desplazar a derecha >>)
- Asignación(=).

También se usa en ambos el operador ternario Exp1 ? Exp2 : Exp3;

Tanto en C como en PHP , una función puede llamarse a sí misma recursivamente.
Definiciones de variables fuera de las funciones en un archivo crean variables globales,tanto en C como en PHP, sin embargo su referencia dentro de funciones es diferente como se verá luego.

Por otra parte, pueden observarse diferencias relacionadas a la sintaxis, como por ej. en relación a la estructura array, en C debe declararse la longitud del array en el momento de su declaración de la forma: `int x[100];` y en PHP tienen una naturaleza dinámica `array();` . El lenguaje C tiene la posibilidad de usar punteros, poderosa herramienta para referenciar posiciones de memoria, que PHP no tiene.
Respecto a la sintaxis de las funciones, la diferencia radica en que en C se debe declarar el tipo de retorno de la función. Por ejemplo:

En C:	En PHP:
void function () { *Código;* }	function () { *Código;* }

En C se dispone del tipo Struct, estructura que permite agrupar variables de tipos distintos, y también dispone del tipo enumerativo , por ej : `enum MONEDAS{ pesos, dólares, euros} monedas;`

C. Un mecanismo de abstracción fundamental en cualquier lenguaje de programación es el uso de *nombres* o identificadores, para denotar entidades o construcciones del lenguaje y, para describir la semántica de un lenguaje, es necesario describir las convenciones que determinan el significado de cada nombre utilizado en un programa, significado que está determinado por las propiedades o atributos asociados con el nombre. El proceso de asociar un atributo con un nombre se llama enlace (Binding). Suponga que se tienen los archivos main.c y Alpha.c. Veamos main.c :

```
#include <stdio.h>
#include "Alpha.h" // incluye en este archivo, (para que esté disponible) el contenido de Alpha
int main() { // asocia al nombre main el atributo función, tipo retorno entero y cuerpo de la función
        puts(__FILE__); //Se imprime la constante __FILE__, la cual es asignada por el preprocesador
        puts(__func__); //imprime la constante __func__
        printFile(); // Definido en Alpha.c "int printFile(){ puts(__FILE__); }"
        printFunc(); //Definido en Alpha.c "int printFunc(){ puts(__func__); }"
        return 0; }
```

```
Salida:     main.c
            main
            Alpha.c
            printFunc
```

Estos son ejemplos de la semántica estática de C, es decir al binding en tiempo de compilación.

Tanto __FILE__ como __func__ están ligados a un valor en tiempo de compilación en el contexto del archivo y de la función respectivamente. Por este motivo las funciones definidas en Alpha no hacen referencia al contexto actual en tiempo de ejecución.

En PHP el binding de las variables con sus atributos se realiza en tiempo de ejecución.

```
<?php  // asocia al nombre Alpha el atributo class y cuerpo de la clase
class Alpha {
// asocia a getClassname los atributos public static function y el cuerpo de la función
   public static function getClassname(){  echo __CLASS__.' '; }
   public static function printClass() { // idem getClassname
      static::getClassname();
      self::getClassname(); }
}
// asocia al nombre Beta el atributo class y el atributo extends
class Beta extends Alpha {   }
Beta::printClass(); ?>
```

```
Salida: Beta Alpha
```

Debido a que las referencias estáticas a la clase actual como self :: o __CLASS__ se resuelven en ejecución usando la cadena estática, se hace referencia a la clase donde está contenida la llamada a self. Si se desea hacer una referencia a la clase actual siguiendo la cadena dinámica, se deberá usar una implementación de PHP llamada "late static binding". un "late static binding" funciona almacenando el nombre de clase de quien hizo la última llamada que no tenga «propagación». En el caso de llamadas a métodos estáticos, se trata de la clase nombrada explícitamente *(Una llamada con propagación es una llamada estática que va precedida por self::, parent::, static:: .).*

static :: getClassname () traerá un binding tardío y hará referencia a la clase que inicialmente hizo la llamada, que en nuestro caso es la clase Beta (Beta::printClass();)

"tardío" proviene del hecho de que static :: no se resolverá utilizando la clase donde se define el método, sino que se calculará utilizando información de tiempo de ejecución.

Además de los *nombres*, una descripción de la semántica de la mayoría de los lenguajes de programación incluye conceptos de ubicación ,valor y enlace(tema que no ahondaremos).

Mediante la operación de asignación se realiza el enlace entre la ubicación de la variable y su valor.

En C y en PHP existe la posibilidad de abreviar las asignaciones de expresiones, esto es un caso de binding en tiempo de implementación, se asocia el operador a la operación correspondiente como se ilustra en la Tabla.

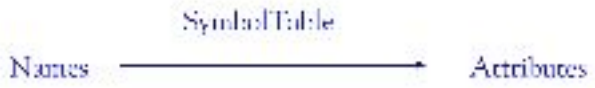
x += exp	Suma	x = x + exp
x -= exp	Resta	x = x - exp
x *= exp	Producto	x = x * exp
x /= exp	División	x = x / exp
x &= exp	And bit a bit	x = x & exp
x |= exp	Or bit a bit	x = x | exp
x >>= exp	Desplazamiento a la derecha	x = x >> exp
x <<= exp	Desplazamiento a la izquierda	x = x << exp

En ambos lenguajes la operación de asignación puede formar parte de una expresión que sea parte de otra asignación. El valor de una asignación será siempre el de su parte derecha.

En C: Int x=y=c=z=1; En PHP: $x=$y=$c=$z=1;

En PHP también se abrevia la concatenación de strings: $cadena .= 'extra';

Los Bindings deben ser mantenidos por un traductor para que se den los significados apropiados a los nombres durante traducción y ejecución. Un traductor hace esto creando una estructura de datos, que se llama tabla de símbolos, y matemáticamente, es una función de nombres a atributos, que podríamos escribir como se muestra en la Figura:



D. Los contenidos de una celda de memoria son una representación codificada de un valor. Los lenguajes introducen la noción de **variables** como una abstracción de la noción de celdas de memoria.

Formalmente, una **variable** es un 5-tupla <nombre, alcance, tipo, l_valor, r_valor>, donde

- **Nombre:** Cadena de caracteres utilizada por las instrucciones del programa para identificar la variable; una abstracción de la dirección de la celda de memoria.
- **Alcance:** es el rango de instrucciones del programa sobre el cual se conoce el nombre;
- **Tipo:** Especificación del conjunto de valores que pueden asociarse con la variable y las operaciones permitidas para crear, acceder y modificarlos.
- **L_valor:** Área de almacenamiento vinculada a la variable. La acción que adquiere un área de almacenamiento para una variable y establece la vinculación se denomina *asignación de memoria*. El tiempo de vida se extiende desde el punto de asignación al punto en el que se recupera el almacenamiento asignado (desasignación de memoria);
- **R_valor:** Valor codificado almacenado en la ubicación asociada a la variable. La representación codificada se interpreta según el tipo de variable.

Un **Tipo** puntero es aquel en el que las variables tienen un rango de valores que consiste en direcciones de memoria y un valor nulo. El valor nulo no es una dirección válida y se usa para indicar que un puntero no se puede usar actualmente para hacer referencia a una celda de memoria.

La aparición de una variable de puntero en una expresión se puede interpretar de dos maneras:

1. como una referencia a su contenido(r_valor), que es una dirección de memoria.
2. como una referencia al valor en la celda de memoria a la que apunta(resultado de desreferenciar el puntero)

Veamos un ejemplo:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int *p;// puntero de alcance global
4  void bye (){printf ("\np vale %d \n", p[0] ); free (p);}// *p global vale 9 al finalizar el main
5  int main(){
6          p=(int *)malloc(sizeof(int));
7          p[0]=9;
8          atexit(bye);
9          static int *p;// p local enmascara a p global, *p contiene basura
10         p=(int *)malloc(sizeof(int)*3);//*p inicializa en 0
11         for (int i=0;i<3;i++) p[i] = (i+1) * 2;// se asigna 2 4 y 6 a variable anonima que p apunta
12         free (p);//p* local muere
13         printf ("\n%p \n", p[0] );// nil
14         p=NULL;// p local muere
15         printf ("\n%p \n", p );//nil
16         return 0; }
```

Una variable que se declara con el especificador static está vinculada estáticamente al alcance de la unidad donde se declara y al almacenamiento. Por lo tanto, su alcance es estático y local para la función, pero su vida útil se extiende a lo largo de toda la ejecución del programa del que forma parte, salvo que, explícitamente se desasigne su L_valor, como es este caso en la línea 14.

Presentamos los atributos de las variables para su mejor entendimiento:

Nombre	Alcance	Tipo	L_valor	R_valor	Tiempo de vida
global p	4 - 10	Puntero a int	Automática	Dirección de memoria	1 - 16+
global p*	4 -10	int	Dinámica	0, 9	6 - 16+
Local p	10-16	Puntero a int	Estática	Dirección de memoria	1 - 14
Local p*	10 -14	int	Dinámica	0, 2, 4, 6	11-13

La variable global **p** ve enmascarado su alcance cuando se declara la variable local con el mismo nombre, pero su tiempo de vida continua hasta el fin del programa, por eso es que podemos acceder a su contenido luego de terminada la función main, ya que termina el alcance de la variable que la enmascara. A la variable anónima *p, se le asigna una dirección de memoria en tiempo de ejecución, por eso se dice que es dinámica, y su tiempo de vida dura mientras dure esa asignación.

El alcance de las variables anónimas a las que apunta p local, normalmente sería el mismo de p local, pero en este caso, cuando p local muere, termina el alcance de las mismas, por lo que intentar accederlas generaría error en tiempo de ejecución.

En PHP a diferencia de C no existe el tipo puntero, pero si es posible el uso de referencias.

```
1 <?php
2 $v="global";
3         function vFun(){
4             global $v; $g=&$v;//v es la variable global y g hace una referencia a ella
5             echo $GLOBALS['g'];//noticia: g no es una variable global
6             static $v;//se enmascara la variable global
7             $v += 10;
```

```
8           $v=&$g;//se pierde la referencia a la variable local
9           return $v;
10      }
11 echo vFun();// output "global"  ?>
```

Nombre	Alcance	Tipo	L_valor	R_valor	Tiempo de vida
global v	3 - 6, 9 -11	String	Automática	“global”	1 - 11
local v	7 - 8	int	Estática	10	3 - 8
g	5 - 10	String	Automática	“global”	3 - 10

References

Ghezzi, C., & Jazayeri, M. (2011). Programming language concepts. Wiley India.

Louden, K. C., & Lambert, K. A. (2012). Programming languages: Principles and practice. Course Technology.

Sebesta, R. W. (2012). Concepts of programming languages - 10th edition. Pearson Addison Wesley.

PHP Manual - © 1997-2020 the PHP Documentation Group - https://www.php.net/manual/