UNIVERSIDAD NACIONAL DE LA PLATA

Facultad de Informática

Conceptos y paradigmas de lenguajes de programación
Trabajo integrador 2019 – Parte 1

- Número de grupo: 26.
- Integrantes: Faraone Negri Camila, 12549/2.
  González Iriart Gabriela, 12125/0.
- Lenguajes asignados: C – PHP.
- Ayudante asignado: Anahí.
- Bibliografía:
  - http://php.net/
  - http://php.net/manual/es/language.types.type-juggling.php
  - Libro: "PHP 6 Sitios Dinámicos Con El Lenguaje Mas Robusto " Editorial: Users.
  - Material ofrecido por la cátedra.
  - Libro: Concepts od Programming Languages – Robert W. Sebesta.
  - Libro: Programming Languages Principales and Practice – Kenneth C. Louden & Kenneth A. Lambert.
  - Libro: Introducción a la programación en C – Marco A. Peña Basurto & José M. Cela Espín.
  - https://www.cprogramming.com/tutorial/c-tutorial.html
  - https://en.cppreference.com/w/cpp/language/function#Function_definition

A. Enuncie los aspectos semánticos más relevantes respecto de la asignación e inicialización por defecto de variables. Ejemplifique con los lenguajes asignados.

C es un lenguaje de programación compilado, es decir, es un lenguaje fuerte y estáticamente tipado, que al **declarar** una variable se reserva un espacio en memoria en la compilación, mientras que el contenido de dicha variable es definido recién en la asignación, sea éste valor uno ya *predefinido* o por *inicialización*. Con esto se pueden evitar muchos errores en tiempo de compilación, sin necesidad de esperar a la ejecución para verlos. Tenemos dos formas de declarar y asignar una variable, por un lado se puede hacer en dos sentencias (una de declaración y otra de asignación), y sino puede declararse y definirse en una sola sentencia.

Se utilizan como principio el tipo primitivo (boolean, char, int, etc.), con un valor por defecto, lo que permite realizar operaciones con funciones que fueron declaradas pero no inicializadas. En consecuencia, se corre el riesgo de cometer algún error semántico debido a que podríamos utilizar valores que no son los esperados.

C:

```c
#include <stdio.h>

int main()
{
    //Por separado
    int num;
    num = 69;
    //Todo en una linea
    char c = 'a';

    printf("%i\n", num);
    printf("%c\n", c);
    return 0;
}
```

Por otra parte, PHP es un lenguaje de tipado dinámico, por lo tanto no se declara su tipo; esta ligadura es dinámica, se asignará el tipo cuando se haya asignado un valor a la misma en tiempo de ejecución. Su ventaja es la flexibilidad. Si por ejemplo definimos una variable entre comillas, la variable será considerada de tipo cadena. Además, en PHP se puede asignar tanto por valor como por referencia. Lo que se refiere a la semántica estática PHP no necesita declarar un tipo a sus variables y en ejecución se le puede asignar valores de distintos tipos sin que esto genere error alguno.

```php
<?php
$foo = "0";  // $foo es string (ASCII 48)
$foo += 2;   // $foo es ahora un integer (2)
$foo = $foo + 1.3;  // $foo es ahora un float (3.3)
$foo = 5 + "10 Cerditos pequeñitos"; // $foo es integer (15)
$foo = 5 + "10 Cerdos pequeños";     // $foo es integer (15)
?>
```

Algo a tener en cuenta en C sobre las variables de tipo numérico, si bien el lenguaje no tiene asignación dinámica, existe un orden parcial entre los distintos tipos, esto quiere decir en ejemplo que una variable de tipo long puede ser asignada a otra variable de un tipo mayor sin pérdida de magnitud ni errores semánticos. Por su contraparte esta variable no podría ser asignada a otra de tipo menor, ya que sí habría pérdida de precisión en el valor almacenado.

Los errores semánticos son "errores racionales, deductivos", la lógica detrás del código escrito no hace lo que el programador cree que hará.

Para provocar un error semántico dinámico en PHP (en ejecución) podemos hacer lo siguiente:

```php
<?php
$dividendo=10;
$divisor=0;
$resto= $dividendo / $divisor;
?>
```

Sintácticamente no hay ningún error en lo descripto en el código, pero semánticamente sí, ya que no es posible dividir por 0 lo que provoca un error en ejecución.

En el caso de C:

// Expectativa: Sumar uno 1 a la variable X

x -= 1;

En este caso estamos restándole 1 a la variable X.

// Expectativa: Sumar 1 a la variable X

y += 1;

En este caso estamos sumándole 1 a la variable Y (no a X, que es la que necesitamos).

// Actualizar todos los elementos de A

for (size_t i=1; i update(A[i]);

En este caso nos estamos olvidando de actualizar al arreglo A en su posición 0, es decir A[0], por lo que se trata de un error semántico.

// Expectativa: calcular el área del rectángulo

int CalcularArea(int ancho, int largo)

{

return ancho + largo;

}

En la función anterior, la sintaxis en C es correcta, pero hay un error semántico. En realidad, para calcular el área del rectángulo se debería multiplicar en lugar de sumar.

Otra característica semántica de C es la sentencia ’break’ que se utiliza para la ’ruptura’ de bucles tales como la sentencia ’for’, ’while’ y también para realizar una salida de estructuras como ’swich’, la sentencia ’break’ debe estar anidado en alguna de dichas estructuras, este es su contexto semántico.

En el caso de PHP, dicha sentencia ‘break’ maneja el mismo contexto semántico que el llevado a cabo por C.

B- Defina una porción de código donde pueda apreciarse las características más relevantes de las variables en cuanto a sus atributos. Elija alguno de los lenguajes asignados que presente mayores posibilidades para mostrar estas características y desarrolle el ejercicio de la misma forma que se realiza en la práctica. Luego, si es necesario realice las explicaciones que permitan una mayor comprensión del ejercicio.

```php
<?php
$a=1; /*ambito global*/
$e=2;
$i=3;
$o=4;
function ejuno(){
    echo $a; /*referencia a una variable del ámbito local */
    global $a, $e; /*referencia a las variables globales desde ahora hasta el final de la funcion*/
    echo "<br>";
    print ($a + $e);
}
function ejdos(){
    static $u=5;
    function tres(){
        echo "funcion tres";
    }
}
function ejcuatro(){
    $o=10;
    function cinco(){
        echo "<br>";
        echo $o;
        global $o; /*alcance estatico usa la del programa principal*/
        echo "<br>";
        echo $o;
    }
    cinco();
}
ejuno();
ejcuatro();
tres(); /*no lo conoce*/
ejdos();
tres(); /*si lo conoce*/
?>
```

Identificador	L-valor	R-valor	Alcance	T.vida
a	automática	NULL	1 – 5 y 7 - 34	1 - 34
e	automática	NULL	2 - 34	1 - 34
i	automática	NULL	3 - 34	1 - 34
o	automática	NULL	4 – 18 y 23 -34	1 - 34
u	automática	NULL	13 - 17	12 - 17
a y e	automática	Indef-3	7- 11	5 - 11
ejuno			5 - 34	5 - 11
ejdos			12 - 34	12 - 17
tres			*	*
a' (línea 6)	automática	NULL	6 - 7	5 - 11
o' (línea 19)			19 - 22 y 27	18 - 28

*Si bien la función “tres” se conocería solo dentro del ámbito de la función “dos”, luego de realizar un llamado a la función “dos” que es la que lo contiene, se carga en memoria y es posible llamarla en cualquier parte del programa luego de dicho llamado.

Algunas características de C:

- El operador de asignación es =
- El operador de comparación es ==
- Un bloque está comprendido por las sentencias encerradas entre { }
- Los identificadores son separados por caracteres de espaciado
- Las sentencias son separadas por ;
- Para los comentarios se utiliza el símbolo // para indicar comentarios de linea, y los símbolos /* comment */ para encerrar comentarios de párrafo.
- La indentación es irrelevante para la evaluación de sentencias
- El operador de invocación es ()

Los **R-Valores** son aquellos que solo pueden estar a la derecha de una expresión y que carecen de sentido a la izquierda. Veamos el siguiente ejemplo:

```
int x;
1 = x;
&x = 3;
```

- Es obvio que eso no se puede hacer, no podemos asignar un valor a un literal ni tampoco a un objeto temporal como es en este caso la dirección de memoria que creamos.

Cualquier **L-Value** se puede comportar siempre como un **R-Value**.

```
int x, y = 4;
x = y;
```

- En este ejemplo en la primera línea tanto ***x*** como ***y*** son L-Values ya que están en la parte izquierda de la expresión, pero en la segunda línea actúa como R-Value ya que está a la derecha.

Se pueden definir variables que tengan ámbito local, pero de existencia permanente.

Para declararlas se utiliza la palabra **static**.

static *declaración*;

Por ejemplo, si se declara una variable **static** dentro de una función, aunque la función retorne la variable permanece con el último valor que se asignó:

```
int contador (void)
{
        static int cuenta = 0;
        return cuenta++;
}
```

Esta función devuelve un número que se va incrementando cada vez que se la llama. La variable **cuenta** es local a la función **contador**, pero no desaparece con la salida de la función.

En la inicialización de una variable *static* se realiza una sola vez, al comienzo de la ejecución del programa. Por eso el ejemplo anterior funciona (*cuenta* se inicializa a cero una sola vez).