Conceptos y paradigmas de lenguajes de programación
Trabajo integrador 2019 – Parte 2

- **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/**
  - **Libro: "PHP 6 Sitios Dinámicos Con El Lenguaje Mas Robusto " Editorial: Users.**
  - **Material ofrecido por la cátedra.**
  - **Libro: Concepts oF 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.**
  - **Libro: The C programming language, second edition – Brian W. Kernighan & Dennis M. Ritchie**
  - **http://www.itnuevolaredo.edu.mx/Takeyas/**

C- Realice una tabla comparativa permitiendo visualizar las diferencias más relevantes respecto del sistema de tipos de los lenguajes asignados. Justifique las características mencionadas con ejemplos de código (al menos para 3 de las características. Concepto de lenguaje fuertemente tipado.

	C	**PHP**
Seguridad (con respecto al tipo de datos)	Fuertemente tipado, dado una variable de un tipo concreto, no se puede usar como si fuera una variable de otro tipo distinto a menos que se haga una conversión.	Débilmente tipado. Por defecto, PHP fuerza a los valores de un tipo erróneo a ser del tipo escalar esperado si es posible. Por ejemplo, una función a la que se le pasa un integer para un parámetro que se prevé sea un string obtendrá una variable de tipo string. Es posible habilitar el modo estricto (a partir de php7) en función de cada fichero. En el modo estricto solamente será aceptada una variable del tipo exacto de la declaración de tipo, o será lanzada una TypeError. La única excepción a esta regla es que se podría proporcionar un integer a una función que espere un float. Las llamadas a funciones desde dentro de funciones internas no se verán afectadas por la declaración strict_types.
Tipos de ligadura	Estática (compilación)	Dinámica (en tiempo de ejecución)
Conversión de tipos (typecasting)	Permite hacer una conversión explícita de un tipo de dato a otro, a criterio del programador siempre y cuando estos tipos sean compatibles.	Casting implícito a partir de operaciones sobre las variables, donde los tipos deben de ser compatibles.
Tipos de datos	Primitivos, Derivados y definidos por el usuario	Escalares, compuestos y especiales

Seguridad (tipos de datos):

Lenguaje C:
Fuertemente tipado, donde se presenta restricción de operaciones ante los tipos asignados al momento de la declaración.

Ejemplo de C:
int i; /* variable i de tipo int */
i = 0; /* asignación de valor 0 */
i = “texto n/”; /* error semántico en compilación */

Lenguaje PHP:
Débilmente tipado, donde la asignación del tipo de las variables se hace en tiempo de ejecución, permitiendo a una variable tomar valores de distintos tipos.

Ejemplo de php:
<?php
$a = 1; //a es un entero
$a = "texto"; //a es una cadena de caracteres
?>

Tipos de Ligadura
Lenguaje C:

Estático: Cuando la comprobación de tipificación se realiza durante la compilación del programa, esto permite que los errores de tipificación sean detectados antes y así promover a que la ejecución del programa sea más eficiente y segura.

Ejemplo de C:

```c
// Example program
#include <stdio.h>

int main()
{
   int a;

   a = 4;
   a = "texto";

   printf( "%d", a );

   return 0;
}
```

options | compilation | execution

```
 In function 'int main()':
10:6: error: invalid conversion from 'const char*' to 'int' [-fpermissive]
```

Lenguaje PHP:
Ligadura dinámica ya que la tipificacion de datos ocurre durante la ejecución del programa y puede cambiar durante la ejecución, lo que hace que el lenguaje sea más flexible.

Se específica a través de la declaración de asignación:

```
<?php
$a = “hola”;
$b = 10;
?>
```

En tiempo de ejecución, a medida que se interpreta el código se liga la variable con el dato en memoria.

Typecasting

En lenguaje C:
Uso de un cast poniendo un *tipo de dato* delante de la expresión o variable a convertir.
Ejemplo:

```
int a = 121;
//ejemplo cast de a en printf
Printf(a, (float)a, (double)a, (char)a):
```

```
Imprime:
a:121
<float>a: 121.000
<double>a: 121.000
<char>a: y
```

Valor a convertido en float y double (mostrándolo con 3 cifras decimales) y a convertido en char donde al hacer la conversión de "a" a char se toma a como el código ascii del char y lo muestra; en nuestro ejemplo 121 es el ascii de la letra "y".

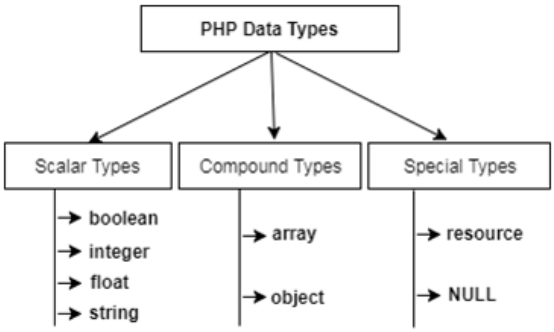
En lenguaje PHP:

```
<?php
$a= 1;    //a es un entero
$b= 0.3;     //b es un float
$a += $b;      //a es ahora float
Echo $a;      //imprime 1.3
?>
```

Un ejemplo de conversión de tipos automática de PHP es el operador ‘+’. Si al menos uno de los operando es float, entonces ambos operando son evaluados como floats y el resultado será float.
Hay que tener en cuenta que esto no implica que se cambien los tipos de los propios operando; el único cambio es en cómo se valúan los operandos y en el tipo de expresión en sí mismo.

Tipo de dato

Lenguaje PHP: El tipo de datos PHP se utiliza para mantener los diversos tipos de datos o valores. Hay tres tipos de tipos de datos en PHP.



En PHP, hay alguna función para verificar el tipo de datos y su valor.

1. gettype () : se utiliza para verificar solo el tipo de datos.
2. var_dump () : se utiliza para verificar el tamaño y el tipo de datos

Ejemplo:

```php
<?php
$var = 60;
echo "60 is: ".gettype($var);
echo "<br>";
// ------------------------------
$var = 1000.56;
echo "1000.56 is: ".gettype($var);
echo "<br>";
// ------------------------------
$var = "PHP";
echo "PHP is: ".gettype($var)
?>
```

Imprime:

60 is: integer
1000.56 is: double
PHP is: string

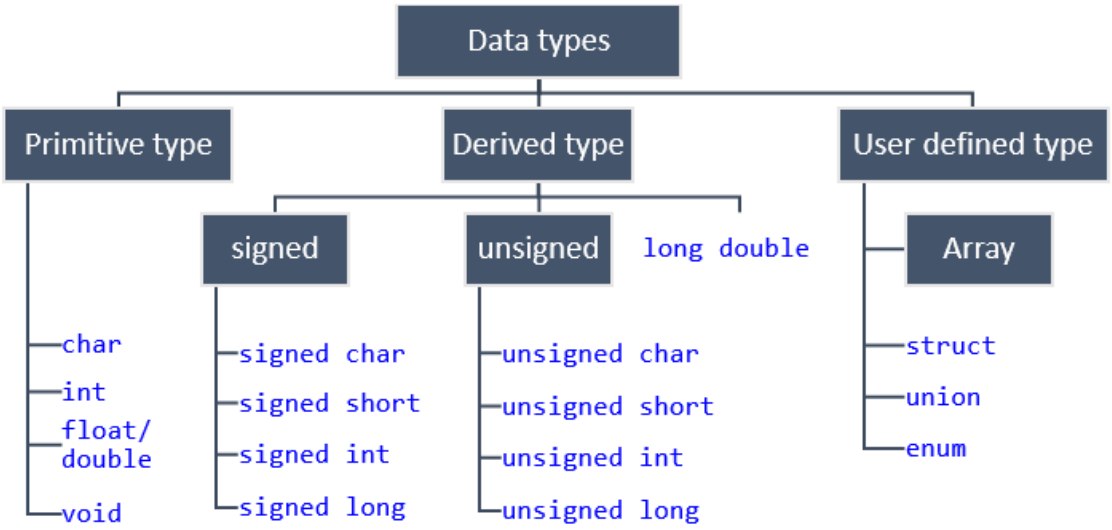
Lenguaje C:

El tipo de datos es un sistema para definir varias propiedades básicas sobre los datos almacenados en la memoria. Propiedades tales como, tipo de datos, rango de datos, bytes ocupados, cómo se interpretan estos bytes, etc.

Los tipos de datos en C se clasifican en tres categorías amplias.

1. Tipo de datos primitivos: Lenguaje C es compatible con cuatro tipos primitivos - char, int, float, void. Los tipos primitivos también se conocen como tipos de datos predefinidos o básicos.
2. Tipo de datos derivados: se define utilizando una combinación de calificadores junto con el tipo de datos primitivo. Los tipos derivados se crean utilizando tipos de datos básicos con comportamiento y propiedad modificados.
3. Tipo de datos definido por el usuario: el lenguaje C admite la función para definir nuestro tipo personalizado en función de nuestra necesidad. El tipo

definido por el usuario incluye matriz, puntero, estructuras, uniones, tipos de enumeración, etc.



Ejemplo tipos de datos primitivos:

```
int a=90;  //  valid
a=90;  //   invalid
```

Ejemplo tipo de datos derivados:

```
int mark[5]={10, 20, 30, 40};
int *a;
```

Ejemplo tipos de datos definidos por el usuario:

```
struct student
{
int id;
char name[10];
float mob;
};
```

D- Enuncie las características más importantes del manejo de excepciones que presentan los lenguajes asignados.

En C, no se manejan las excepciones de forma nativa, es entonces donde aparece la posibilidad de generar una simulación de manejo de excepciones, esto puede ser a través de la librería <signal.h>, donde se logran manejar las excepciones que surjan durante la ejecución.
Antes de definir un nuevo manejador de excepciones se deben de tener en cuenta ciertos aspectos del control de excepciones: debe ser simple de usar y de entender, debe separar el código del tratamiento de excepciones del código normal, implementar un tratamiento uniforme de las excepciones, debe permitir que las acciones de recuperación sean programadas.
Cuando el manejador termina se pueden hacer dos cosas:

- Reanudar la ejecución del bloque
- Terminar la ejecución del bloque y devolver el control al punto de invocación.

```
...
try
{
   ...
   / * Bloque de código que puede causar errores * /
   ...
}
catch (SomeExceptionType e)
{
   ...
   / * Qué hacer si sucede un error * /
   ...
}
...
```

- Cada bloque catch captura una clase de excepción.
- Un bloque try puede tener un bloque catch general que capture excepciones no captadas.
- Un bloque try no puede capturar una excepción derivada de una clase capturada en un bloque catch anterior.

setjmp.h es un encabezado definido en la biblioteca estándar de C (#include <setjmp.h>) para proporcionar "saltos no locales": control de flujo que se desvía de la llamada a subrutina habitual y la secuencia de retorno. Las funciones complementarias *setjmp* y *longjmp* proporcionan esta funcionalidad.
Un uso típico de setjmp/longjmp es la implementación de un mecanismo de excepción que utiliza la capacidad de longjmp para restablecer el estado del programa o hilo, incluso a través de múltiples niveles de las llamadas a funciones. Un uso menos común de setjmp es crear una sintaxis similar a co-rutinas.

Definición:

```
int setjmp (jmp_buf env);
void longjmp (jmp_buf env, int val);
```

Los mecanismos setjmp y longjmp funcionan de la siguiente manera: cuando se invoca setjmp la primera vez, devuelve 0 y llena la estructura jmp_buf con el entorno de llamada y la máscara de señal . El entorno de llamada representa el estado de los registros y el punto en el código donde se llamó la función. Cuando se llama longjmp, el estado guardado en la variable jmp_buf se copia nuevamente en el procesador y el cómputo comienza desde el punto de retorno de la función setjmp , pero el valor devuelto es el que se pasa como segundo argumento a la función longjmp .

Los sistemas de excepciones reales tienen la posibilidad de definir varios tipos de excepciones. Estos tipos se asignan sobre tipos y las declaraciones de captura interceptan excepciones utilizando estos tipos.
Debe de usarse una instrucción switch en lugar de una declaración if para lograr albergar más de una sentencia Catch (con parametros).

Existe la posibilidad de agregar al final del bloque la sentencia *finally*. El bloque finally es código que se ejecutará haya o no excepciones.

```
...
try
{
  ...
  / * Bloque de código que puede causar errores * /
  ...
}
catch (SomeExceptionType e)
{
  ...
  / * Qué hacer si sucede un error * /
  ...
}
catch (SomeExceptionType e)
{
  ...
  / * Qué hacer si sucede un error * /
  ...
}

catch (SomeExceptionType e)
{
  ...
  / * Qué hacer si sucede un error * /
  ...
}
finally
{
  ...
  / * De cualquier manera, hacer lo siguiente… * /
  ...
}

...
```

La instrucción throw se utiliza para señalizar la aparición de una situación anómala (excepción) durante la ejecución del programa.
Se puede utilizar una instrucción throw en el bloque catch para volver a producir la excepción, la cual ha sido capturada por la instrucción catch.
Disparar una excepción mediante: → throw new Exception("Error: …….");

```
static void ProcesarCadena (string s)
{
    if (s == null)
    { throw new ArgumentNullException(); }
}
static void Main() {
try {
       string s = null;
       ProcesarCadena(s);
}
//Mas específico
catch (ArgumentNullException e)
{      Console.WriteLine("{0} First exception caught.", e); }
//Menos específico
catch (Exception e)
{      Console.WriteLine("{0} Second exception caught.", e);
} }
```

Otra manera en la que se puede definir un manejador de excepciones para C es tomando similitudes en la definición de excepciones en ADA, como lo es el uso de un RAISE para elevar o propagar una excepción.
En C se utiliza la sentencia **#define** para la declaración de constantes, es decir, que mantendrá el mismo valor a lo largo de todo el programa.
Cabe destacar que en éste planteo de un posible manejador también se utilizan, al igual que en el sistema try-catch, las funciones complementarias setjmp y longjmp (las cuales permiten guardar el contexto de ejecución actual de su programa y reanudarlo en un punto arbitrario en el futuro).

```
#define NEW_EXCEPTION (name) ...
    /* declaración de excepción */
#define BEGIN ...
    /* comienzo de ámbito */
#define EXCEPTION ...
    /* comienzo de lista de manejadores */
#define END ...
    /* fin de ámbito */
#define RAISE (name) ...
    /* elevar excepción */
#define WHEN (name) ...
    /* manejador */
#define OTHERS ...
    /* manejador por defecto */
```

Esquema donde ocurre el llamado a la excepción:

```
typedef char *exception;
exception error="error";
/* comienzo del ámbito de la excepción */
    if (current_exception = (exception)setjmp(save_area) == 0)
    {
        /* instrucciones donde se pueden producir excepciones */
        if (error_detected) {
            longjmp(save_area, (int)error);
        };
    }
    else {
```

```
        if (current_exception == error) {
            /* manejador de error */
        }
        else {
        /* propagar la excepción */
        }
    }
/* final del ámbito de la excepción */
```

Ejemplo que implementa lo definido anteriormente:

```
NEW_EXCEPTION (sensor_high);
NEW_EXCEPTION (sensor_low);
NEW_EXCEPTION (sensor_dead);
...
BEGIN
...
RAISE (sensor_high);
...
EXCEPTION
WHEN (sensor_high)
    /* manejador */
...
END;
```

Excepciones en Lenguaje PHP:

PHP tiene un modelo de excepciones similar al de otros lenguajes de programación. Una excepción puede ser lanzada ("thrown"), y atrapada ("catched") dentro de PHP. El código puede estar dentro de un bloque try para facilitar la captura de excepciones potenciales. Cada bloque try debe tener al menos un bloque catch o finallycorrespondiente.
El objeto lanzado debe ser una instancia de la clase Exception o una subclase de Exception. Intentar lanzar un objeto que no lo sea resultará en un Error Fatal de PHP.

- catch
  Se pueden usar múltiples bloques catch para atrapar diferentes clases de excepciones. La ejecución normal (cuando no es lanzada ninguna excepción dentro del bloque try) continuará después del último bloque catchdefinido en la sencuencia. Las excepciones pueden ser lanzadas ("thrown") (o relanzadas) dentro de un bloque catch.

Cuando una excepción es lanzada, el código siguiente a la declaración no será ejecutado, y PHP intentará encontrar el primer bloque catch coincidente. Si una excepción no es capturada, se emitirá un Error Fatal de PHP con un mensaje "Uncaught Exception ..." ("Excepción No Capturada"), a menos que se haya definido un manejador con set_exception_handler().

- finally
  En PHP 5.5 y posterior, se puede utilizar un bloque finally después o en lugar de los bloques catch. El código de dentro del bloque finally siempre se ejecutará después de los bloques try y catch, independientemente de que se haya lanzado una excepción o no, y antes de que la ejecución normal continúe.

Ejemplo: lanzar una excepción

```
<?php
function a($x) {
    if (!$x) {
        throw new Exception('División por cero.');
    }
    return 1/$x;
}

try {
    echo a(5) . "\n";
    echo a(0) . "\n";
} catch (Exception $e) {
    echo 'Excepción capturada: ',  $e->getMessage(), "\n";
}

// Continuar la ejecución
echo 'Hola\n';
?>
```

El resultado sería:
0.2
Excepción capturada: División por cero.
Hola

E. Conclusión sobre los lenguajes: Realice una conclusión exponiendo las diferencias, características, virtudes y defectos de los lenguajes asignados. Esta conclusión no debe superar una carilla.

C es un lenguaje bastante rápido por estar cercano a los lenguajes de bajo nivel, es decir que, aunque es un lenguaje de alto nivel (es estructurado y posee sentencias y funciones que simplifican su funcionamiento) tenemos la posibilidad de programar a bajo nivel (Como en el Assembler tocando los registros, memoria etc.). Se implementa para el desarrollo de sistemas operativos, bases de datos, programación móvil, entre otros, teniendo en cuenta que la principal desventaja de este lenguaje es sus sintaxis que resulta ser compleja y trabajoso de implementar.
PHP (acrónimo recursivo de PHP: Hypertext Preprocessor) es un lenguaje de código abierto muy popular especialmente adecuado para el desarrollo web y que puede ser incrustado en HTML. Lo mejor de PHP es su extrema simplicidad para el principiante, pero a su vez ofrece muchas características avanzadas para los programadores profesionales. Es un lenguaje que está muy bien documentado.

Una característica de C al igual que PHP, es que ambos lenguajes tienen incluidas librerías de funciones que pueden ser incluidas haciendo referencia la librería que las incluye.

F. Conclusión sobre el trabajo: Realice una conclusión mencionando los aportes que le generó la realización del trabajo como grupo.

El trabajo de investigación no solo nos brindó un mayor marco de información en relación a los lenguajes que nos tocaron analizar (C y PHP), sino también de que cada etapa de la entrega forman una parte de la constitución de lo que es un lenguaje de programación, abriéndonos la perspectiva a que un lenguaje no es solo sintaxis correcta y compilación exitosa, sino a poder ver más en profundidad la forma en que se manejan ante el uso de variables, el tratamiento de excepciones, versatilidad en el código y lograr comprender que la semántica juega un rol muy importante en el desarrollo de software.
En cuanto a los lenguajes, ambos son conocidos y de uso masivo, por lo que el trabajo nos brindó un espacio extra para poder investigar acerca de ellos, a rasgos generales, características propias y distintivas de cada lenguaje, como por ejemplo, su propia sintaxis, la forma en que utilizan pasaje de parámetros, etc.