

**Universidad de las Américas, Puebla**

**Asignatura:**

Lenguajes y Paradigmas de Programación

**Tarea 3:** “Investigación de conceptos clave”

**Profesor:**

Juan Carlos Galán Hernández

**Alumno:** Raúl González Cruz

**ID:** 151211

**Fecha:** 2/10/2015

Otoño 2015

**Introducción**

Se tiene como objetivo de esta tarea investigar algunos conceptos clave que se utilizan con frecuencia al elaborar programas, debido a que existen ocasiones en las que uno no sabe identificar o interpretar el significado de ciertas líneas de código que se utilizan dentro de un programa. Por otra parte, esta tarea nos permitirá saber más acerca de cómo y cuándo utilizar ciertas técnicas que nos permiten elaborar códigos eficientes; poniendo este conocimiento en práctica cuando el desarrollador lo requiera.

Finalmente, es importante que como desarrollador, no sólo se sepa resolver un problema en específico, sino que además se conozcan teóricamente los conceptos básicos que permiten llevar a cabo programas útiles para la vida diaria.

**Investigación**

* Variables estáticas

También llamadas como variables de clase, son variables únicas para todas las instancias (objetos) de una clase. Es decir, una declaración de una variable estática se refiere a que existe un único espacio de memoria, compartido por todos los objetos de una misma clase, que contiene un valor invariable.

Las variables estáticas se declaran con la palabra reservada *static.* Cabe destacar que estas variables se inicializan automáticamente a “cero” en la asignación de la memoria dada; además de que pueden ser especificadas como variables:

* Automáticas. Siguen existiendo incluso después de que el bloque en el que están definidas termina, por lo que el valor de una variable se conserva entre llamadas repetidas a la misma función, sin olvidar que el almacenamiento asignado se convierte en “permanente” durante la ejecución del programa. La declaración ha de realizarse **obligatoriamente dentro de un método que involucra el uso de llaves.**
* Externas. La única diferencia es que las variables externas estáticas se acceden solamente en el archivo en el que se definen, es decir, ningún otro archivo de origen puede acceder a las variables externas estáticas que se definen en otro. La declaración ha de realizarse **obligatoriamente en la cabecera de la clase.**

Las variables estáticas pueden ser inicializadas en sus declaraciones; sin embargo, estas inicializaciones deben ser expresiones constantes.

* Ciclo de vida de las variables

Se refiere al intervalo de tiempo que trascurre desde que se crea la variable hasta que se destruye y deja de existir. El tiempo durante el que una variable conserva su valor se denomina duración; sin embargo, el valor de una variable puede cambiar durante su duración, pero conservará algún valor.

Finalmente, cuando una variable pierde ese valor, es porque su ciclo de vida ha terminado, porque toda variable viva tiene siempre un valor.

El ciclo de vida de algunas variables es:

* + Variables de clase

1. Se crean cuando la clase se usa por primera vez.
2. Se inicializan por defecto si no se hace de modo explícito:
   1. 0 para variables numéricas
   2. "false" para variables booleanas
   3. "null" para objetos
3. Suelen existir para el resto del programa (salvo que no esté sobrecargado).
4. Se destruyen cuando el programa termina su ejecución.
   * Variables de método
5. Creadas en la sentencia en la que están definidas.
6. No se inicializan por defecto. Contienen datos imprevisibles.
7. Se destruyen al salir del bloque, es decir, en la llave final.
   * Variables de instancia u objeto
8. Se crean cuando se crea el objeto que las contiene.
9. Se inicializan por defecto si no se hace de modo explícito:
   1. 0 para variables numéricas
   2. "false" para variables booleanas
   3. "null" para objetos
10. Se destruyen cuando no se encuentran referencias activas para el objeto, es decir, no tiene un valor y consecuentemente perdió ámbito.

* Memoria dinámica (Lenguaje C)

Básicamente, es memoria que se reserva en tiempo de ejecución de un programa. Su principal ventaja es que su tamaño puede variar durante la ejecución del programa. Particularmente en C, el programador es el encargado de liberar esta memoria cuando no la utilice más.

Por lo regular cuando se diseña un algoritmo, se deben conocer los elementos de entrada y de salida; sin embargo, en algunas ocasiones no se sabe de forma exacta el número de variables que requerirá nuestro algoritmo o programa. Entonces el uso de memoria dinámica es necesario cuando no conocemos el número de datos/elementos a tratar.

Como se ha descrito antes, todos los objetos tienen un tiempo de vida. En C, existen 3 tipos de duración: estática (se crean antes de que el programa inicie su ejecución y se destruyen cuando el programa termina), automática (se crean al entrar al bloque en el que fueron declaradas y se destruyen al salir de ese bloque) y asignada (asignada se refiere a los objetos cuya memoria se reserva de forma dinámica). Como se mencionó antes, la memoria dinámica se crea y se debe liberar de forma explícita.

La biblioteca estándar de C proporciona las funciones *malloc, calloc, realloc y free* para el manejo de memoria dinámica. Estas funciones están definidas en el archivo de cabecera *stdlib.h.* A continuación se describen las funciones mencionadas:

* Malloc

Reserva un bloque de memoria y devuelve un puntero *void* al inicio de la misma. Tiene la siguiente definición: *void \*malloc(size\_t size);* donde el parámetro *size* especifica el número de bytes a reservar.

En caso de que no se pueda realizar la asignación, devuelve el valor nulo (definido en la macro NULL), lo que permite saber si hubo errores en la asignación de memoria.

* Calloc

Funciona de modo similar a *malloc*, pero además de reservar memoria, inicializa a 0 la memoria reservada. Se usa comúnmente para arreglos y matrices. Está definida de esta forma: *void \*calloc(size\_t nmemb, size\_t size);* donde el parámetro *nmemb* indica el número de elementos a reservar, y *size* el tamaño de cada elemento.

* Realloc

Redimensiona el espacio asignado de forma dinámica anteriormente a un puntero. Tiene la siguiente definición: *void \*realloc(void \*ptr, size\_t size); d*onde *ptr* es el puntero a redimensionar, y *size* el nuevo tamaño, en bytes que tendrá.

Si el puntero que se le pasa tiene el valor nulo, esta función actúa como *malloc*. Si la reasignación no se pudo hacer con éxito, devuelve un puntero nulo, dejando intacto el puntero que se pasa por parámetro.

Al usar *realloc*, se debería usar un puntero temporal; de lo contrario, podríamos tener una fuga de memoria, si es que ocurriera un error en *realloc*. Cuando se redimensiona la memoria con *realloc*, si el nuevo tamaño (*size*) es mayor que el anterior, se conservan todos los valores originales, quedando los bytes restantes sin inicializar. Si el nuevo tamaño es menor, se conservan los valores de los primeros *size* bytes. Los restantes también se dejan intactos, pero no son parte del bloque regresado por la función.

* Free

Sirve para liberar memoria que se asignó dinámicamente. Si el puntero es nulo, *free* no hace nada. Se define como: *void free(void \*ptr);* donde el parámetro *ptr* es el puntero a la memoria que se desea liberar. Una vez liberada la memoria, si se quiere volver a utilizar el puntero, primero se debe reservar nueva memoria con *malloc* o *calloc*.

Finalmente, las regiones de memoria que reservan y liberan estas funciones, además de reservar la memoria dinámica como tal, son almacenadas en el almacenamiento libre o *heap*. Cabe destacar que el manejo de memoria dinámica es la base del poder del lenguaje C, puesto que le da la capacidad de crear programas complejos que emplean grandes cantidades de memoria, manejados de manera eficiente.

* Clase

Es simplemente una representación de un tipo de objeto; es decir, una clase permite la descripción de un objeto. Así como un plano puede utilizarse para construir varios edificios, una clase puede utilizarse para crear varias copias de un objeto. Para crear una clase se usa la palabra reservada ***class*.**

Cabe destacar que definir clases permite trabajar con código reutilizable, puesto que desde una clase se puede crear una instancia (un objeto que deriva de otro objeto). La instancia toma el patrón de la clase padre; sin embargo, las variables son independientes.

En otras palabras, una clase define un tipo de objeto especificando qué propiedades o atributos tiene y qué operaciones o métodos disponibles posee.

* Objeto

Es una entidad existente abstracta en la memoria del ordenador que tiene propiedades, como atributos, variables o datos sobre sí mismo almacenados por el objeto; y unas operaciones o procedimientos disponibles específicas referentes al mismo objeto, conocidas también como métodos.

En programación, un objeto permite separar los diferentes componentes de un programa, simplificando así su elaboración, depuración y posteriores mejoras. Además, a los objetos se les otorga ciertas características de la vida real.

Los objetos se componen de 3 partes fundamentales: métodos, eventos y atributos. Para crear un objeto de una clase se usa la palabra reservada ***new*.**

* Instanciación

Se llama instancia a todo objeto que derive de algún otro. De esta forma, todos los objetos son instancias de algún otro, menos la clase Object que es la madre de todas.

Cada vez que creamos una nueva instancia, ésta adquiere las propiedades, métodos y eventos de la clase a la que pertenece, lo que permite decir en lenguaje humano: “*Es un*…”. Sin embargo, cada instancia es independiente de las otras; por lo que existen ciertas ventajas como:

1. Si se realiza algún cambio en la clase, todas las instancias de esta clase se actualizarán automáticamente; lo cual permite hacer cambios sin tener que ir a cada una de las instancias (se aplica el mismo principio de herencia, aunque a un nivel diferente).
2. Al ser independientes una instancia de las otras, se puede asignar valores diferentes sin que afecten a las demás. Aunque comparten la misma estructura, pueden programarse individualmente.

* Herencia

Es un mecanismo que permite compartir automáticamente métodos y datos entre clases, subclases y objetos. Además, permite crear nuevas clases introduciendo ciertas variaciones con respecto a su clase base, si es que el desarrollador del programa así lo decide.

Para los lenguajes de programación orientada a objetos, la herencia permite obtener una clase a partir de otra, lo que implica que ésta obtenga los métodos y atributos de la clase que hereda; facilitando además la instanciación. Por otra parte, la herencia permite la existencia de clases abstractas, cuya particularidad es la ausencia de instancias específicas.

Existen lenguajes que permiten al diseñador controlar qué miembros de las clases base son visibles o no a las clases derivadas. Para ello, se utilizan las palabras reservadas como:

* *Private*: Ningún miembro privado de la superclase es visible en la subclase.
* *Protected*: Los miembros protegidos de la superclase son visibles en la subclase, pero no visibles para el exterior.
* *Public*: Los miembros públicos de la superclase siguen siendo públicos en la subclase.

Además, existen dos tipos principales de herencia:

1. Herencia simple. Una subclase puede heredar datos y métodos de una clase simple así como añadir o sustraer ciertos comportamientos.
2. Herencia múltiple: Posibilidad de adquirir métodos y datos de varias clases simultáneamente.

* Sobrecarga

Como tal, sobrecarga permite nombrar de una misma forma diferentes variables y métodos. Para programación orientada a objetos, conocido también como sobrecarga de métodos, es un mecanismo que permite asignar el mismo nombre a dos o más funciones que llevan a cabo acciones distintas, destacando el uso de distintos argumentos entre cada uno de ellos. Es el compilador el encargado de saber cuál usar dependiendo de los parámetros o argumentos utilizados.

El desarrollador debe ser cuidadoso al usar esto, puesto que la sobrecarga no admite métodos con argumentos iguales.

* Ensombrecimiento (*shadowing*)

Como tal, representa un problema que se produce cuando se inicializa una constante de manera global con nombre idéntico a una inicialización de una variable local. El siguiente ejemplo ilustra esta situación:

#include <stdio.h>

int numero = 10;

void funcion()

{

int numero; /\* Colisiona con variable global \*/

numero = 20; /\* ¿A qué variable se le asigna el valor? \*/

}

int main(int argc, char \*argv[]) {

funcion();

printf("%d\n", numero); /\* ¿Qué valor imprime? \*/

return 0;

}

Cuando se ejecuta la función se produce el ensombrecimiento: la variable global “numero” no puede ser accedida, pues ese nombre se refiere a la variable local del método “función”. El valor que se imprime es 10 porque la variable global ya no está ensombrecida.

**Bibliografía**

[s.a]. *Ámbito y Acceso*. Consultado el 30 de septiembre de 2015. Recuperado de <http://mit.ocw.universia.net/1.00/s02/class-sessions/lecture-11/lecture-11.pdf>

[s.a]. *Static Variables*. Documento electrónico. Consultado el 28 de septiembre de 2015. Recuperado de <http://ee.hawaii.edu/~tep/EE160/Book/chap14/subsection2.1.1.6.html>

Franco, E. *Memoria dinámica y su uso en C.* Archivo PPT. Consultado el 29 de septiembre de 2015. Recuperado de <http://eafranco.com/docencia/algoritmia/files/29_30/Clase_29_30.pdf>

Frick, C. *Programación Orientada a Objetos- Clase e Instancia.* Consultado el 29 de septiembre de 2015. Recuperado de <https://thefricky.wordpress.com/2008/01/15/programacion-orientada-a-objetos-clase-e-instancia/>

Microsoft. *¿Qué es una clase?* Consultado el 29 de septiembre de 2015. Recuperado de <https://msdn.microsoft.com/es-es/library/ms172601(v=vs.90).aspx>

Microsoft. *Comprensión del ciclo de vida de las variables.* Consultado el 30 de septiembre de 2015. Recuperado de <https://msdn.microsoft.com/es-es/library/office/gg278427.aspx>

Wikilibros. *Manejo dinámico de memoria.* Programación en C. Consultado el 30 de septiembre de 2015. Recuperado de <https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C/Manejo_din%C3%A1mico_de_memoria>

[s.a]. *Capítulo 3. Declaración de variables.* Consultado el 28 de septiembre de 2015. Recuperado de <http://www.it.uc3m.es/labas/course_notes/variables_es.html>