Conceptos y Paradigmas de Lenguajes de Programación

Lenguajes Asignados:

- C
- Matlab

Grupo 3

- 14859/2 - Adrogué Benas María Noel
- 15190/3 - Farinella Robertino

Referencias

- Primera Parte
    - *The C programming language* - Brian W. Kernighan y Dennis M. Ritchie.
    - *The GNU C Reference Manual*. link
    - Documentación oficial de Matlab. link
    - *Matlab Reference Manual*. link *PDF*

Primera Parte

A.

> Enuncie y compare distintos aspectos semánticos (tanto de la semántica estática y dinámica) de cada uno de los lenguajes asignados.

Semántica estática

- **Declaración de Variables**

  - C
    En C se deben declarar todas las variables antes de su uso, generalmente al principio de la función y antes de cualquier proposición ejecutable, esto se denomina *tipado estático*. Una variable en C consta de un nombre de tipo, de los cuales C provee muchos, seguido por una lista de variables, por ejemplo:

```
int aux;
```

  - Matlab
    Al contrario de C, Matlab utiliza un solo tipo de variable, el arreglo. Este arreglo puede ser unidimensional, como un vector e incluso un valor (en caso de poseer un solo elemento), o multidimensional, como una matriz.

```
A = 5 (matriz unaria de enteros)
B = ['a' 'b' 'c'] (matriz unidimensional de caracteres)
```

  Una variable puede ser declarada sin su tipo y el lenguaje identifica de qué tipo es por los valores de inicialización de la misma, esto se denomina *tipado dinámico*.

- **Asignación y cambio de variables a distintos tipos**

  - C
    En este lenguaje, una variable no puede cambiar de tipo, razón por la cual querer asignar un string a una variable de tipo entero, por ejemplo, generará un error. Uno puede realizar, sin embargo, lo que se denomina un *casting* para poder utilizarla como si fuera de otro tipo. Un *casteo* crea una copia en el tipo indicado de la variable, no modifica la variable original.

  - Matlab
    Por el contrario en Matlab una variable, al ser todos arreglos, puede cambiar de tipo de elemento en cualquier momento. Una matriz unidimensional de enteros puede pasar a ser una matriz multidimensional de caracteres sin ninguna sintaxis particular. No puede, eso si, combinar operaciones de elementos numéricos con otro tipo de elementos o comparar dos

arreglos de distintas dimensiones (a no ser que uno sea un arreglo de un elemento, el cual se comporta como un solo valor).

- **Signatura o Firma de una función**

  - C
    Una función recibe una cantidad de elementos con sus respectivos tipos y puede devolver 0 o 1 valor de un tipo específico. Al momento de compilación se hace un chequeo de los llamados a funciones y de que contengan todos los parámetros de la definición y que sean de los tipos correctos. Si una función termina con "…" puede recibir una mayor cantidad de argumentos, pero nunca menos y debe poseer por lo menos un parámetro en la definición.

  - Matlab
    En Matlab en cambio, gracias a los argumentos `varargin` y `varargout`, una función puede recibir y devolver una cantidad variable (tanto mayor como menor) de argumentos bajo una misma definición. Estos argumentos no necesariamente son del mismo tipo.

  Tanto Matlab como C no soportan pasaje por referencia, solo por valor. La forma de simular un pasaje por referencia en los lenguajes es pasar un puntero al valor que se quiere simular como referencia, y así se usa a lo que apunta ese puntero para poder ser modificado.

Semántica dinámica

Matlab, o mas específicamente lenguaje M, es un lenguaje interpretado, esto quiere decir que las sentencias en M se ejecutan instantáneamente. Debido a esto, todo tipo de chequeo o error es de tipo dinámico, ya que no hay una compilación a priori de la ejecución del programa y por lo tanto, compatibilidad de tipos, indices fuera de límites, etc. se realizan en ejecución, a diferencia de C el cual es un lenguaje compilado y por lo tanto estos chequeos se realizan en compilación.

- **Indexación**

  - C
    En C la indexación siempre comienza por el 0 y se extiende a n-1 donde n es la cantidad de elementos dentro del arreglo declarado. Debido a esto es muy usual ver errores como querer acceder al valor en la posicion n del arreglo, cuando hay n valores pero hay n-1 posiciones.

  - Matlab
    En Matlab, por el contrario, la indexación de los elementos comienza en la posición 1 y se extiende hasta la n. De forma inversa a lo anterior el error será querer acceder a la posición 0 de un arreglo sin haber especificado anteriormente de manera explícita que dicho arreglo contiene un indice 0.

- **Estructuras de Control**

- C
  Operacionalmente, ciertas estructuras de control no se comportan de la misma forma en ambos lenguajes, por ejemplo, la sentencia *for*.
  En C se utiliza un índice el cual se va actualizando hasta que la condicion de control no se cumpla. Este indice puede ser modificado dentro del bucle y de ese modo infiere en el ciclo de ejecución del for.

- Matlab
  En M, si bien también poseemos esta posibilidad de utilizar un índice el cual se va actualizando, éste no puede ser modificado dentro del bucle, siempre se sobreescribe el valor del indice al que le tocaría en esa iteración. Por otro lado también existen otras formas de realizar un for, por ejemplo `for v = [1 5 8 17]` realiza las instrucciones declaradas en el bucle para cada uno de los valores del arreglo.

- **Alocación y Liberación de Memoria**

  - C
    C aloca memoria estática dependiendo de los tipos de variables declarados y sus dimensiones, en caso de ser arreglos. Esta memoria se encuentra reservada durante el tiempo de ejecución del bloque (en caso de variables estáticas, persiste), y no puede ser liberada ni alocada manualmente. Esto imposibilita el uso de *Sparse Arrays*, arreglos donde al ser en su mayoría valores iguales a 0, esto no se aloca y solo se alocan los valores no nulos a medida que van siendo utilizados. Otra desventaja de la imposibilidad de variar la memoria alocada es los espacios nulos en bloque de memoria que quedan inaccesibles. Por ejemplo, un arreglo de tamaño fijo al que se le elimina un elemento del mismo. Los elementos deben luego realizar un corrimiento para no desperdiciar memoria ni correr el riesgo de apuntar a un puntero inválido.
    Al no poseer C un control de la memoria inaccesible, por lo tanto, puede ocurrir lo que se denomina *Memory leak*.

  - Matlab
    Matlab posee lo que se denomina un *Garbage Collector*. Éste controla la memoria inutilizada y va durante la ejecución liberando memoria de forma eficiente, reduciendo dimensiones de vectores con elementos eliminados o haciendo uso de Sparse Arrays. Esto no elimina la amenaza de Memory leaks, debido a memoria que puede ser alocada manualmente, pero reduce considerablemente el desperdicio de memoria en caso de una programación "mas irresponsable" con respecto al uso de memoria.

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.

Código

```c
#include<stdio.h>

void func_1();
int a, b = 10;

int main()
{
  func_1();
  func_1();
 func_1();

  return 0;
}

void func_1()
{
  int a = 1;
  static int b = 100;
  printf("a = %d\n", a);
  printf("b = %d\n\n", b);
  a++;
  b++;
}
```

Identificador	Tipo	r-valor	alcance	tiempo de vida
a (línea 4)	automática	10	4-17	3-23
b (línea 4)	automática	10	4-18	3-23
a (línea 17)	automática	1	17-23	17-23
b (línea 18)	estática	100	18-23	todo el programa

Tanto a y b declaradas en la la línea 4 son variables globales. Por otro lado, a y b declaradas en `func_1()` son variables locales e incluso b es declarada como estática. Cuando `func_1()` es llamada por primera vez toma el valor 100 (r-valor al momento de declaración) y a toma el valor 1, luego son aumentadas en 1 e impresas; sin embargo cuando `func_1()` es llamada nuevamente, a vuelve a tomar valor 1 pero b retiene su valor anterior de 101 dado que es estática y esto hace que retenga su valor entre llamadas a la función `func_1()`, terminando con valor 102. No obstante, esto no extiende su alcance, b es sólo válida dentro de su función.
También notar que en C el alcance de una variable es estática y por lo tanto a toma el valor 1 en vez de 10 ya que se encuentra dentro del bloque de `func_1()` y por lo tanto, esta declaración toma prioridad sobre la global.