Conceptos y Paradigmas de Lenguajes de Programación

Lenguajes Asignados:

- C
- Matlab

Grupo 3

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

Referencias

- Documentación oficial de Matlab. link
- *Matlab Reference Manual*. link (*PDF*)
- *MATLAB Programming for Engineers* - Stephen J. Chapman
- *Cornell University MATLAB Programming Virtual Workshop*. link
- *Computer Science: an Interdisciplinary Approach* - Robert Sedgewick, Kevin Wayne.
- *The C programming language* - Brian W. Kernighan, Dennis M. Ritchie.
- *The GNU C Reference Manual*. link
- *C Recipes: A Problem-Solution Approach* - Shirish Chavan.

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:

```c
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 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.

Segunda Parte

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	**Matlab**
Tipado	
C es un lenguaje de tipado estatico y fuerte. Esto tiene diferentes implicaciones en el lenguaje. La primera, y probablemente la mas notable, es que en la declaracion de una variable, el programador debe declarar explicitamente el tipo de la variable. El tipo de la variable no cambia durante la ejecucion, solo el tipo del valor mediante conversiones, pero la variable se mantiene del mismo tipo. Como la ligadura se realiza durante la compilacion, el tipado estatico elimina la necesidad de repetir el cheque de tipos cada vez que el programa es ejecutado. Sin embargo, C no es tan flexible en operaciones y converciones implicitas permitidas entre tipos como otros lenguajes.	MATLAB, por el contrario, es un lenguaje dinamico y debilmente tipado, por lo tanto, uno no tiene que explicitamente declarar el tipo de una variable. La declaracion de la misma variable en 2 tipos distintos es perfectamente aceptable, por ejemplo `x = 5; x = 'foo'`. Por otro lado, ya que los valores tienen tipos, no las variables, el tipo de la variable es dado por su valor y puede cambiar constantemente, al ser dinamicamente tipado, y por lo tanto la ligadura entidad/variable se realiza durante su ejecucion. Esto hace de MATLAB un lenguaje mas flexible pero a su vez mas propenso a errores
Tipos de Datos	
En C encontramos distintos tipos de datos. Por un lado, las constantes, las cuales pueden ser numericos, de caracter, flotantes, etc. Luego, existen los tipos basicos (int, double, float, etc.) y por otro los tipos derivados, construidos a partir de los tipos fundamentales (arreglos, funciones, apuntadores, estructuras, uniones, etc.). El tipo void se usa como el tipo regresado por funciones que no generan un valor.	MATLAB tiene 15 tipos de datos fundamentales: 8 tipos de enteros (de acuerdo a su almacenamiento y precision),single, double, logical, char, cell, structure y function. Todos estos elementos son arreglos, desde un minimo de 0x0 en dimensiones, hasta arreglos n-dimensionales, por lo que se dice que en Matlab *"Todo es un arreglo".* MATLAB tambien posee clases creadas por el usuario. Desde string, siendo arreglos de chars hasta matrices de double, arreglos de tipo cell y timetables. Como hemos mencionado anteriormente, estos tipos pueden variar constantemente durante la ejecucion del programa.

Conversion	
En C la coversion puede ser de 2 tipos: conversion implicita o explicita. En la coversion implicita, la conversion del tipo de origen(derecha del programador) al tipo destino(izquierda del programador) se realiza de manera automatica. Si el rango de destino es mayor al de origen, se denomina conversion ancha, cuando el rango de destino es menor, se denomina conversion estrecha. El compilador realiza conversiones anchas sin ningun problema, pero al realizar conversiones estrechas puede ocurrir perdida de precision (por ejemplo de tipo double a tipo entera se producira un truncamiento.) En la conversion explicita, el programador indica explicitamente el tipo destino de la conversion, se dice que el programador *castea* a tipo destino. Ejemplos de esto es de double a entero, de char a entero, de entero a apuntador, etc. Es importante notar que, si el casteo se realiza sobre una variable, primero se copia el valor de la variable origen y luego realiza la conversion al tipo destino, para luego asignarlo a la variable destino en caso de haber una, dejando sin modificar el tipo de la variable origen, manteniendo asi su tipado fuerte y estatico.	Al igual que C, la conversion en MATLAB tambien puede ser explicita o implicita. La conversion implicita se realiza cuando se realiza concatenacion o asignacion suscripta en arreglos, ya que todos sus elementos deben de ser del mismo tipo. En este caso, MATLAB sigue ciertas reglas para entender cual es el tipo destino y cual el origen. En caso de concatenacion, los tipos definidos por el usuario son dominantes sobre predefinidos como double, si no hay prioridad entre los elementos, se utiliza el tipo del dato mas a la izquierda. En caso de asignaciones suscriptas, siempre se utiliza el tipo del dato de la izquierda. La conversion explicita, a diferencia de C, no se realiza con una forma de casteo general (en caso de C *(tipoDestino)*) sino que hay distintas funciones de casteo para distintos pares de tipos de datos. Ejemplos de esto es `cellstr int2str bin2dec array2table`, etc. Esto permite muchos mas casos explicitos que los provistos mediante casteo explicito en C donde muchas conversiones explicitas no se encuentran definidas.
Inferencia de Tipo	
C no permite inferencia de tipos debido a que siempre debe estar definido el tipo de un variable. La inferencia de tipos se refiere a la posibilidad de lenguajes de detectar o inferir el tipo de una variable o funcion analizando su contexto, y esto no es posible en C. Lo es en lenguajes derivados como C++ o C# gracias a variables de tipo var o auto.	En MATLAB la inferencia de tipo es algo constantemente presente. Cuando declaramos una funcion por ejemplo, nunca se expresa el tipo de retorno o el tipo de los argumentos que esa funcion recibe, sin embargo el lenguaje puede inferir el resultado de esa funcion. Difiere de la conversion implicita en que justamente no se utilizan los tipos de argumentos de los argumentos para saber el tipo de la funcion. Al ser MATLAB dinamico, tambien complica la distincion entre ambas, la inferencia es mas distintiva en lenguajes estaticos.

	La inferencia tiene sus desventajas, sin embargo. Desde la posibilidad de que haya errores en la inferencia y la dificultad de encontrar dichos errores, hasta la falta de comprension de lo que se esta haciendo en una funcion dependiendo de sus argumentos.
Polimorfismo	
Si bien C no soporta polimorfismo de manera directa, hay ciertas formas de simular polimorfismo en C mediante punteros a funciones y macros.	MATLAB por otro lado si soporta polimorfismo. Utiliza clases de tipo abstractas para poder sobrecargar funcionalidades y heredar caracteristicas en comun para grupos de objetos. Tambien se puede realizar sobrecarga de funciones directamente mediante la definicion de funciones con la misma definicion pero distintas implementaciones.

Ejemplos de Codigo

Tipado

```
int main() {
  int i = 0;
  double b = 1.5;
  int result = i + b;
  return 0;
}
```

Notese como en cada variable, incluso la funcion `main()`, se encuentra expresado su tipo.

```
function avg = average(nums)
global TOTAL
avg = sum(nums)/TOTAL;
end
```

En MATLAB no se declaran tipos, en el ejemplo incluso se denota una variable global, pero no se dice que tipo es.

Tipos de Datos

```
void imprimir(const char *str){
  printf("%s",str);
```

```
}
```

Aqui podemos ver varios ejemplos de tipos de datos en C, desde string que es una cadena de caracteres, hasta el tipo void de la funcion y puntero a la cadena str.

```
str = 'Hello World!'
n = 2345
d = double(n)
un = uint32(789.50)
rn = 5678.92347
c = int32(rn)
```

Varios tipos de datos en MATLAB, esto imprimira:

```
str = Hello World!, n =  2345, d =  2345, un = 790, rn = 5678.9 y c =  5679
```

Recordar que todos estos tipos son arreglos de 1 elemento, a excepcion de str que es un arreglo de caracteres.

Conversion

Conversion en C

```c
#include <stdio.h>
int main()
{
    int x = 10.0;    // int x
    char y = 'a';  // character c

    x = x + y; // Valor ASCII de 'a' es 97

    float z = x + 1.0; // x es convertido implicitamente a float

    double sum = (double)x + 10.0; // x es convertido explicitamente de int a double

    printf("x = %d, z = %f, sum = %d", x, z, sum);
    return 0;
}
```

Esto imprimira: `x = 107, z = 108.000000, sum = 118.000000`

Conversion en MATLAB

```
chr = '37.294e-1';

val = str2num(chr) // conversion explicita de string a valor numerico
A = [1.0 2.0 4.0];
B = 3;
A(2) = B; // conversion implicita de int a double
```

Esto imprimira:

```
val = 3.7294
A =   1.0   3.0   4.0
```

D.

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

Las excepciones son interrupciones al flujo normal de la ejecucion del programa debido a errores en el codigo.

Matlab

Cuando un metodo no puede recuperaarse de un error por su cuenta, se recolecta informacion del error (en detalle mas adelante), crea un objeto de tipo MException y luego lanza la excepcion. Un MException contiene la siguiente informacion del error:

- **identifier**: string unico que identifica el error mediante el nombre del elemento que causo la excepcion y una regla mnemonica para identificar el tipo de error. El identificador es unico para cada excepcion.
- **message**: descripcion del error.
- **stack**: arreglo de estructuras que describen el path a la locacion donde ocurrio el error, el nombre de la funcion, y el numero de linea donde ocurrio.
- **cause**: informacion sobre excepciones secundarias relacionadas con la principal, en caso de haberlas.

`ME.getReport()` imprime la informacion del error de manera comprensible.
El programador puede crear sus propios objetos de excepciones mediante bloques try catch para atrapar errores. Esto permite examinar informacion sobre el error, recolectar aun mas informacion para reportar,tratar de realizar la tarea de alguna otra forma y limpiar comportamiento no deseado causado por el error.

C

El manejo del desbordamiento, errores de division y otras condiciones de error dentro de la evaluacion de errores no esta definido por el lenguaje. El trato de las condiciones excepcionales puede ajustarse mediante el uso de funciones no estandar de biblioteca. Estas funciones estan definidas en `<signal.h>`, la cual da facilidades para manejar excepciones, tal como señales de interrupcion de una fuente externa o un error durante la ejecucion. Las excepciones se manejan mediante funciones del estilo `void (* signal(int sig, void (*handler)(int)))(int)`. Si handler es `SIG_DFL` se usa el comportamiento definido por la

implantacion, si es `SIG_IGN`, la señal se ignora, de otra manera se llama a la funcion apuntada por el handler, con lor argumentos de tipo señal, las cuales pueden ser, entre otras:

- SIGABRT: terminacion anormal.
- SIGFPE: error aritmetico, por ej. division por 0 u overflow.
- SIGILL: imagen de funcion ilegal, por ej. instruccion ilegal.
- SIGTERM: solicitud de terminacion enviada al programa.
  etc.

Cuando ocurre subsecuentemente una señal sig, la señal se regresa a su comportamiento predeterminado, luego se llama a la funcion manejadora. Si este regresa, la ejecucion continuara donde se encontraba cuando ocurrio la excepcion.
Para enviar la señal sig al programa se utiliza la funcion `int raise(int sig)` la cual devuelve 0 si fue exitoso o cualquier otro valor en caso contrario.