# UNIVERSIDAD NACIONAL AUTÓMATA DE MÉXICO

## FACULTAD DE INGENIERÍA

## ESTRUCTURA DE DATOS Y ALGORITMO I

## ACTIVIDAD ASÍNCRONA #1 VIERNES

PÉREZ AGUILAR ROBERTO

(04/03/2021)

# ACORDEÓN DE LENGUAJE C

- **COMENTARIOS:** el comentario por línea inicia con los símbolos "//" y termina hasta donde finaliza el renglón, los comentarios por bloque inicia con "/* y finaliza con "*/" este puede abarcar varios renglones
- **DECLARACIÓN DE VARIABLES:** se hace por medio de la siguiente sintaxis "[modificadores] tipoDeDato identificador [= valor]. Los tipos de datos son caracteres, enteros sin punto decimal, flotantes que son números reales de precisión normal y los dobles que son reales de doble precisión. El especificador de formato nos ayuda a guardar o imprimir, para enteros en base 10 %i o %d (%li o %ld si son más largos, si es para imprimir un entero base 8 se utiliza %o y para base 16 es %x, para valor real %f, para reales de doble precisión %lf, %e para notación científica, %g redondea a 3 cifras significativas, %c un carácter y %s cadena de caracteres.
- **IDENTIFICADOR:** es el nombre con el que se va almacenar un tipo de dato, puede contener palabras de a-z mayúsculas y carácter de guion bajo en lugar de espacios, se recomienda la notación del camello ejemplo tipoDeDato.
- **MODIFICADORES DE ALCANCE:** "const" impide que el valor cambie durante la ejecución del programa, "static" permanece estática en la memoria
- **OPERADORES:**

| *Operador* | *Operación* | *Uso* | *Resultado* |
|---|---|---|---|
| + | Suma | 125.78 + 62.5 | 188.28 |
| - | Resta | 65.3 - 32.33 | 32.97 |
| * | Multiplicación | 8.27 * 7 | 57.75 |
| / | División | 15 / 4 | 3.75 |
| % | Módulo | 4 % 2 | 0 |

| *Operador* | *Operación* | *Uso* | *Resultado* |
|---|---|---|---|
| >> | Corrimiento a la derecha | 8 >> 2 | 2 |
| << | Corrimiento a la izquierda | 8 << 1 | 16 |
| & | Operador AND | 5 & 4 | 4 |
| \| | Operador OR | 3 \| 2 | 3 |
| ~ | Complemento ar-1 | ~2 | 1 |

- **MOLDEO O CAST:** las operaciones da un tipo de dato diferente y es necesario moldearlo.
- **EXPRESIONES LÓGICAS:**

| *Operador* | *Operación* | *Uso* | *Resultado* |
|---|---|---|---|
| == | Igual que | 'h' == 'H' | Falso |
| != | Diferente a | 'a' != 'b' | Verdadero |
| < | Menor que | 7 < 15 | Verdadero |
| > | Mayor que | 11 > 22 | Falso |
| < = | Menor o igual | 15 <= 22 | Verdadero |
| > = | Mayor o igual | 20 >= 35 | Falso |

- **DEPURACIÓN DE PROGRAMAS:** es cuando no falla la ejecución del programa.
- **ESTRUCTURA DE CONTROL SELECTIVA IF:** sintaxis if expresión_logica){ // bloque de código}, evalúa la expresión lógica si se cumple se ejecuta el bloque de código, sino se sigue el flujo normal del programa, si el bloque de código solo consta de una línea no son necesarias la llaves
- **ESTRUCTURA DE CONTROL SELECTIVA IF-ELSE:** if (expresión_logica) {// bloque de código si es verdadera} else {// bloque de código si la condición

es falsa}. Si la condición es verdadera se ejecuta el código de la primera llave y si es falsa se ejecuta el segundo código de la segunda llave, es posible anidar la misma estructura.

- **ESTRUCTURA DE CONTROL SELECTIVA SWITCH-CASE:** switch (opcion_a_evaluar) {case valor1: /* Código a ejecutar*/ break; case valor2: /* Código a ejecutar*/ break;… case valorN: /* Código a ejecutar*/ break; default: /* Código a ejecutar*/}. Evalúa y compara de acuerdo a cada case y ejecuta su código, es importante usar break al finalizar cada case ya que si no se hace ejecutara el siguiente código.
- **ENUMERACIÓN:** enum identificador(VALOR1, VALOR 2, …VALORN);, enum boolean (FALSE, TRUE); esto nos puede ayudar en las estructuras de control selectivas
- **ESTRUCTURA DE CONTROL REPETITIVA WHILE:** while (expresión_lógica) { // Bloque de código a repetir mientras que la expresión lógica sea verdadera.}, primero valida la expresión y si se cumple ejecutara el código hasta que no sea válida.
- **ESTRUCTURA REPETITIVA DO-WHILE:** do { /* Bloque de código que se ejecuta por lo menos una vez y se repite mientras la expresión lógica sea verdadera. */ } while (expresión_lógica); ejecuta el primer bloque de código hasta que ya no sea válido.
- **ESTRUCTURA DE CONTROL DE REPETICIÓN FOR:** for (inicialización; expresión_lógica; operaciones por iteración) {/* Bloque de código a ejecutar */}, primero se establece el valor, después la condición y por último la operación, deja de ejcutarse hasta cumplir la condición.
- **DEFINE:** #define nombre valor, es una macro que sirve para el definir un valor
- **BREAK:** provoca que el ciclo se cierre inmediatamente.
- **ARREGLOS UNIDIMENSIONALES:**


la primera localidad las compone el número 0 y la ultima la compone el elemento n-1, La sintaxis para un la siguiente: tipoDeDato nombre [tamaño], también se puede usar las estructura de control repetitiva como son el for y while.

- **APUNTADORES:** es una variable que contiene la dirección de una variable, la sintaxis es la siguiente TipoDeDato *apuntador, variable; apuntador = &variable;

- **ARREGLOS MULTIDIMENSIONALES:** su estructura es la siguiente tipoDato nombre [tamaño] [tamaño]… [tamaño]; la primera dimensión corresponde a los renglones, la segundo a las columnas, la tercera al plano. Se pueden recoger arreglos con apuntadores.
- **FUNCIONES:** valorRetorno nombre (parámetros) {// bloque de código de la función}, nos permite dividir un problemas en varios bloques de código.
- **ABRIR Y CERRAR ARCHIVOS:** *FILE fopen (char *nombre_archivo, char *modo); int fclose (FILE *apArch); r: Abre un archivo de texto para lectura. w: Crea un archivo de texto para escritura. a: Abre un archivo de texto para añadir. r+: Abre un archivo de texto para lectura / escritura. w+: Crea un archivo de texto para lectura / escritura. a+: Añade o crea un archivo de texto para lectura / escritura. rb: Abre un archivo en modo lectura y binario. wb: Crea un archivo en modo escritura y binario.
- **FUNCIONES FGETS Y FPUTS:** Las funciones fgets() y fputs() pueden leer y escribir, cadenas sobre los archivos.
- **FUNCIONES FSCANF Y FPRINTF:** fprintf() y fscanf() se comportan exactamente como printf() (imprimir) y scanf() (leer), excepto que operan sobre archivo. int fprintf(FILE *apArch, char *formato, ...); int fscanf(FILE *apArch, char *formato, ...);.
- **FUNCIONES FREAD Y FWRITE:** permiten trabajar con elementos de longitud conocida. fread permite leer uno o varios elementos de la misma longitud a partir de una dirección de memoria determinada (apuntador). int fread(void *ap, size_t tam, size_t nelem, FILE *archivo). fwrite permite escribir hacia un archivo uno o varios elementos de la misma longitud almacenados a partir de una dirección de memoria determinada. int fwrite(void *ap, size_t tam, size_t nelem, FILE *archivo).

# ACORDEÓN DE RUBY

QUE ES RUBY: es un lenguaje multiplataforma, interpretado y orientado a objetos.

**NORMA DE CÓDIGO:** los paréntesis son opcionales cuando usamos un método como puts, en Ruby todo número o palabra es un objeto, acabando de usar el método se pone ".".

**NÚMEROS:** son los mismos que se manejan, más que aquí Fixnum maneja números pequeños, mientras que Bignum es lo contrario.

## OPERADORES Y PROCEDENCIAS:

| operador | significado |
|---|---|
| : | Alcance (scope) |
| [] | Índices |
| ** | Exponentes |
| + - ! ~ | Unarios: pos/neg, no,... |
| * / % | Multiplicación, División,... |
| + - | Suma, Resta,... |
| « » | Desplazadores binarios,... |
| & | 'y' binario |
| , ^ | 'or' y 'xor' binarios |
| > >= < <= | Comparadores |
| == === <=> != =~ !~ | Igualdad, inegualdad,... |
| && | 'y' booleano |
| \|\| | 'o' booleano |
| .. ... | Operadores de rango |
| = (+=, -=,...) | Asignadores |
| ?: | Decisión ternaria |
| not | 'no' booleano |
| and, or | 'y', 'o' booleano |

**EL OPERADOR MODULO:** el operador nos da el resto de una división, ejemplos

Puts (5%3) #imprime 2

Puts (-5%3) #imprime 1

Puts (5%-3) #imprime -1

Puts (-5%-3) #imprime -2

**STRINGS:** son secuencias de caracteres entre comillas simples o dobles, también pueden haber vacíos.

**EL ACENTO GRAVE:** son los strings delimitados por este símbolo, este enviado al sistema operativo como un comando para ser ejecutado.

**INTERPOLACIÓN:** es insertar un resultado en un string mediante # {expresión}.

**COMILLAS VS COMILLAS DOBLES:** se podría decir que el proceso es más rápido cuando se utilizan las comillas simples, ya que cuando hay una comilla doble es posible que exista una interpolación.

**STRING#LENGTH:** devuelve el número de bytes de una cadena.

**VARIABLES:** sirven para almacenar un número o un string, las locales empieza con letras minúsculas o un guion bajo, están formadas por letras, números o guiones. Se puede apoyar en el signo de igualdad.

**INTERPRETACIÓN DINÁMICA:** no es necesario explicar el tipo de variable.

**ALCANCE:**

- Alcance global y variable global: el alcance es el que menos usa y las variables tienen el signo de dólar, son muy poco usadas.
- Variables globales por defecto: son aquellas que tienen información útil que puede ser usada en cualquier momento.
- Alcance local: acaba al alcance de las variables locales.

**LOS MÉTODOS BANG (¡):** acaban con el signo de exclamación y son peligrosos ya que pueden modificar el objeto (número, caracteres, cadenas, etc.).

**ALIAS:** cambia el nombre a un método existente.

**MÉTODOS PERDIDOS:** son útiles para evitar errores en los programas.

**VALORES POR DEFECTO:** estos son usados si no se especifica un valor explícitamente por lo cual se hace por medio del signo de igualdad.

**INTRODUCIENDO DATOS GETS:** en lugar de sacar datos por la pantalla nos ayuda a introducirlos.

**CONSTANTES:** empiezan por una letra mayúscula y después se utiliza el signo de igualdad para asignarle un valor.

**RANGOS:** sirven para representar una secuencia estas tienen un punto inicial y un punto final.

**ARRAYS:** es una lista o conjunto ordenado, se puede usar diferentes elementos.

```
# Arrays (o vectores)

# array vacío
vec1 = []

# Los índices empiezan desde el cero (0,1,2,...)
nombres = ['Satish', 'Talim', 'Ruby', 'Java']
puts nombres[0]
puts nombres[1]
puts nombres[2]
puts nombres[3]
# si el elemento no existe, se devuelve nil
puts nombres[4]
# pero podemos añadir a posteriori más elementos
nombres[3] = 'Python'
nombres[4] = 'Rails'
```

**%W:** puede volver más fácil la elaboración de un arreglo ya que quita las comillas. Se pueden extraer elementos del arreglo con el método each (| |).

```
nombres1 = [ 'ann', 'richard', 'william', 'susan', 'pat' ]
puts nombres1[0] # ann
puts nombres1[3] # susan

# esto es más sencillo y más rápido:
nombres2 = %w{ ann richard william susan pat }
puts nombres2[0] # ann
puts nombres2[3] # susan
```

**BLOQUES:** tienen la misma funcionalidad que el lenguaje c, estos están separados por llaves “{}”.

**EXPRESIONES REGULARES:** son usada para reconocer patrones y procesos de texto para iniciar una expresión se utiliza diagonales para iniciar y finalizar.

**EL COMODÍN . (PUNTO):** se usa para buscar un carácter en una posición determinada.

**MÉTODOS CON STRINGS:**

- Reverse: invierte los caracteres.
- Length: indica los números de caracteres y espacios.
- Upcase: pone todos los caracteres en mayúscula.
- Downcase: pone todos los caracteres en minúsculas.
- Swapcase: pone los caracteres en minúsculas a mayúsculas y viceversa.
- Capitalize: pone el primer carácter de un string en mayúsculas y los demás en minúsculas.
- Slice: da una parte de un string.

**IF-ELSE:** tiene el mismo funcionamiento que en c.

**ELSIF:** puede ayudar a tomar más decisiones que la función if-else.

**CASE:** se crean una serie de condiciones y se ejecuta la que se cumpla primero.

**WHILE:** tiene la misma estructura que el while de c.

**OBJECT ID, RESPOND_TO?:** sirve para ejecutar una dicha acción mediante la búsqueda de un número que está en un objeto.

**CLASS, INSTANCE_OF?:** podemos a que clase pertenece un objeto mediante este método.

**CONSTRUCTORES LITERALES:**

| Clase | Constructor Literal | Ejemplo |
|---|---|---|
| String | ' ó " | "nuevo string" o 'nuevo string' |
| Símbolo | : | :símbolo o :"símbolo con espacios" |
| Array | [ ] | [1, 2, 3, 4, 5] |
| Hash | { } | {"Nueva Yor" => "NY", "Oregon" => "OR' |
| Rango | .. ó ... | 0...10 ó 0..9 |
| Expresiones regulares | // | /([a-z]+)/ |

**FICHEROS: LECTURA/ESTRUCTURA:** File.open abre un nuevo fichero si no hay un bloque; pero si usa un bloque, entonces el fichero será el argumento del bloque, y se cerrará automáticamente cuando termine el bloque.

**REQUIRE:** lee una única vez el fichero especificado.

**LOAD:** lee el fichero indicado tantas veces como aparezca la instrucción.

**FREEZE:** ayuda a que objeto no sea modificado, convirtiéndose en una constante, si se intenta modificar provocara un error.

**SÍMBOLOS:** estos están precedidos por dos puntos ": action". Se pueden usar como índices.

**SÍMBOLOS VS STRING:** los símbolos son más útiles ya que puede dar referencia que se trata del mismo objeto por lo cual son más eficientes que los string ahorrando tiempo y memoria.

**ASHESH:** conocidos como arrays asociativos, mapas o diccionarios, son parecidos a los arrays en que son una colección de referencias a objetos. Pueden anexar cualquier tipo de objetos ya sean strings, expresiones, etc.