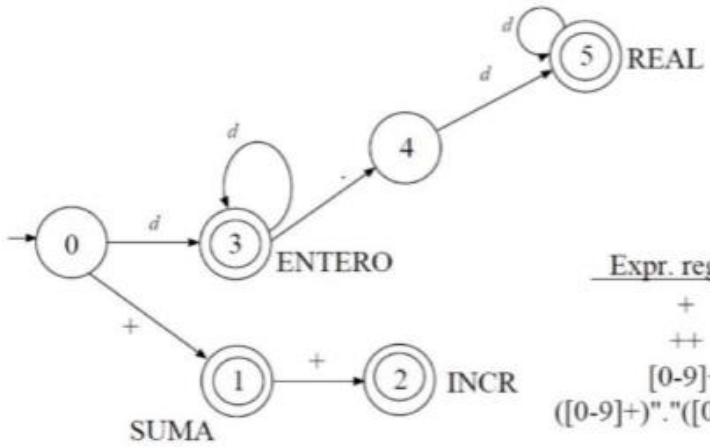
Parcial 1 – Lenguajes de Programación

Realizado por: Mariana Ruge Vargas

1. Con el siguiente diagrama de estados de un AFD, implemente un programa en AWK que acepte expresiones regulares.



Expr. regular	Token
+	SUMA
++	INCR
[0-9]+	ENTERO
([0-9]+)"."([0-9]+)	REAL

- Versión del AWK

```
(base) mariana@mariana-asusvivobook ~/Escritorio/Universidad/Desarrollo p1 awk -W version
mawk 1.3.4 20200120
Copyright 2008-2019,2020, Thomas E. Dickey
Copyright 1991-1996,2014, Michael D. Brennan
```

Se analizan la suma, el incremento, el número real y el número entero.

```
// put your code # Definición de patrones y tokens
BEGIN {
    # Tokens y patrones para distintos tipos de tokens
    token_suma = "\\+"              # Token para el símbolo de suma
    token_incremento = "\\+\\+"     # Token para el operador de incremento
    token_entero = "^[0-9]+"        # Expresión regular para números enteros
    token_real = "^[0-9]+\\.[0-9]+" # Expresión regular para números reales
}

{
    # Eliminar espacios en blanco al inicio de la línea
    gsub(/^[ \t]+/, "", $0)

    while (length($0) > 0) {
        # Eliminar espacios en blanco al inicio de la línea
        gsub(/^[ \t]+/, "", $0)

        # Procesar cada patrón en el orden correcto
        if (match($0, token_incremento)) {
            print "INCREMENTO"           # Imprimir token para incremento
            $0 = substr($0, RSTART + RLENGTH) # Eliminar el token procesado de la línea
        } else if (match($0, token_suma)) {
            print "SUMA"                 # Imprimir token para suma
            $0 = substr($0, RSTART + RLENGTH) # Eliminar el token procesado de la línea
        } else if (match($0, token_real)) {
            print "NUMERO_REAL"          # Imprimir token para número real
            $0 = substr($0, RSTART + RLENGTH) # Eliminar el token procesado de la línea
        } else if (match($0, token_entero)) {
            print "NUMERO_ENTERO"        # Imprimir token para número entero
            $0 = substr($0, RSTART + RLENGTH) # Eliminar el token procesado de la línea
        } else {
            # Si no coincide con ningún patrón, eliminar el primer carácter
            $0 = substr($0, 1)
        }
    }
}
```

Ejecución

```
(base) ✘ mariana@mariana-asusvivobook ~/Escritorio/Universidad/Desarrollo p1/punto 1 awk -f Automata.awk prueba.txt
SUMA
INCR
ENTERO
REAL
REAL
^C
```

2.

```lex
%{
#include <stdio.h>
#include <stdlib.h>

int lambda_found = 0;
int print_found = 0;
%}

%%

"lambda"         { lambda_found = 1; }  // Encontró la palabra clave "lambda"
"print"          { print_found = 1; }   // Encontró la palabra clave "print"
[a-zA-Z_][a-zA-Z0-9_]*   { /* Identificador */ }
[0-9]+           { /* Número */ }
"*"              { /* Operador de potencia */ }
"="              { /* Operador de asignación */ }
":"              { /* Dos puntos */ }
"("              { /* Paréntesis izquierdo */ }
")"              { /* Paréntesis derecho */ }
[ \t\n]+         { /* Ignorar espacios, tabulaciones y saltos de línea */ }
.                { /* Ignorar todo lo demás */ }

%%

int main(int argc, char **argv) {
    if (argc < 2) {
        fprintf(stderr, "Uso: %s <archivo>\n", argv[0]);  // Mensaje de uso del programa
        return 1;
    }

    // Abrir el archivo
    FILE *file = fopen(argv[1], "r");
    if (!file) {
        fprintf(stderr, "Error al abrir el archivo %s\n", argv[1]);  // Mensaje de error al abrir el archivo
        return 1;
    }

    // Establecer el archivo de entrada para Lex
    yyin = file;

    // Analizar el archivo de entrada
    yylex();

    // Verificar si se encontraron expresiones lambda y declaraciones print
    if (lambda_found && print_found) {
        printf("ACEPTA\n");  // Si se encontraron ambas, imprimir "ACEPTA"
    } else {
        printf("NO ACEPTA\n");  // Si no se encontraron ambas, imprimir "NO ACEPTA"
    }

    fclose(file);
    return 0;
}
```

Ejecución

```
(base) mariana@mariana-asusvivobook ~/Escritorio/Universidad/Desarrollo p1/Punto 2 ./lambda_checker prueba3.txt
ACEPTA
```

3. Escriba un programa...

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LONGITUD_LINEA 1024

// Función para contar las ocurrencias de la palabra clave en una línea
int contar_ocurrencias(const char *linea, const char *clave) {
    int conteo = 0;
    const char *ptr = linea;

    // Busca la primera ocurrencia de la palabra clave en la línea
    while ((ptr = strstr(ptr, clave)) != NULL) {
        conteo++;
        // Avanza el puntero después de la palabra clave encontrada
        ptr += strlen(clave);
    }

    return conteo;
}

int main(int argc, char *argv[]) {
    // Verifica que se hayan pasado exactamente dos argumentos
    if (argc != 3) {
        fprintf(stderr, "Uso: %s <archivo> <palabra_clave>\n", argv[0]);
        return 1;
    }

    // Asigna los nombres de archivo y palabra clave a variables
    const char *nombre_archivo = argv[1];
    const char *palabra_clave = argv[2];

    // Abre el archivo para lectura
    FILE *archivo = fopen(nombre_archivo, "r");

    // Verifica si el archivo se abrió correctamente
    if (archivo == NULL) {
        perror("Error al abrir el archivo");
        return 1;
    }

    char linea[MAX_LONGITUD_LINEA];
    int conteo_total = 0;

    // Lee el archivo línea por línea
    while (fgets(linea, sizeof(linea), archivo) != NULL) {
        // Suma las ocurrencias encontradas en cada línea
        conteo_total += contar_ocurrencias(linea, palabra_clave);
    }

    // Cierra el archivo
    fclose(archivo);

    // Muestra el resultado
    printf("La palabra '%s' se repite %d veces en el texto.\n", palabra_clave, conteo_total);

    return 0;
}
```

```
(base) ✘ mariana@mariana-asusvivobook ~/Escritorio/Universidad/Desarrollo p1 cd 'punto3'
(base)   mariana@mariana-asusvivobook ~/Escritorio/Universidad/Desarrollo p1/punto3 gcc index.c
(base)   mariana@mariana-asusvivobook ~/Escritorio/Universidad/Desarrollo p1/punto3 ./a.out listado.txt gato
La palabra 'gato' se repite 1 veces en el texto.
```

4. Realice la comparación del rendimiento de un lenguaje de programación compilado y un lenguaje de programación interpretado.

-Se realizará la comparación entre C y Python.

1. La versión del compilador de C usada es **gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0.**

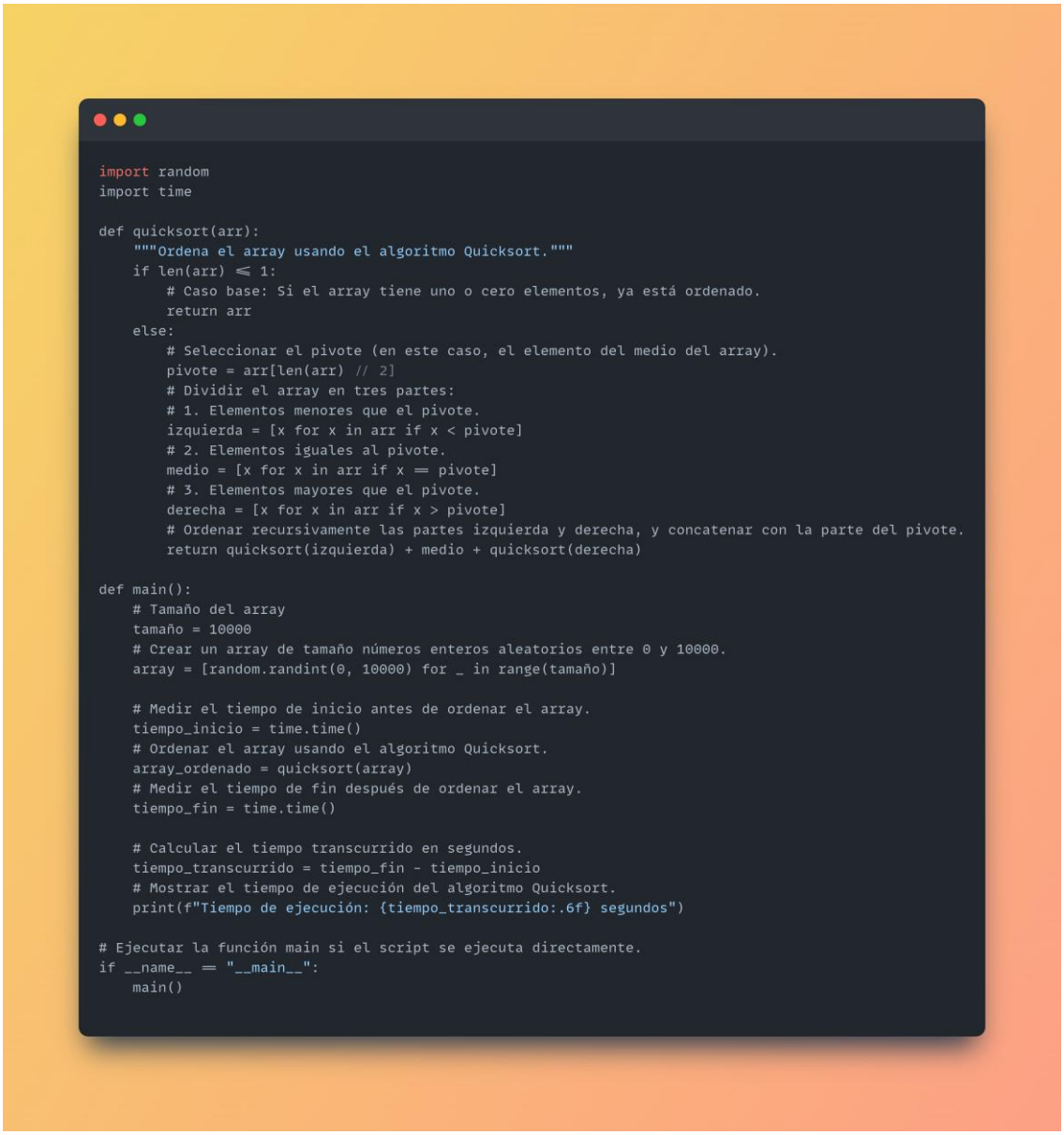
2. La versión del interprete de **Python usada es: 3.11.7.**

Para esta prueba se ejecutó el algoritmo Quicksort en C (que es un lenguaje compilado) y en Python (que es un lenguaje interpretado). Y se midió el tiempo, de ejecución en ambos. Donde podemos evidenciar que C es más rápido a la hora de ordenar por medio de este algoritmo, con un mejor rendimiento. Esto es dado que el compilador genera un archivo traducido a lenguaje máquina, y cada sentencia se analiza una sola vez y no cada vez que se ejecuta. Por otra parte, el interprete solo tomas las líneas fuente y lo traduce a máquina sin un archivo, por lo tanto debe ser traducido cada vez que se ejecuta.

Código C

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Función para intercambiar dos elementos
void intercambiar(int *a, int *b) {
    //Variable temporal
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Función para particionar el array
int particionar(int arr[], int bajo, int alto) {
    int pivote = arr[alto]; // Seleccionar el pivote (último elemento en este caso)
    int i = bajo - 1; // Índice del elemento menor

    //Ciclo for
    for (int j = bajo; j < alto; j++) {
        if (arr[j] < pivote) {
            i++;
            intercambiar(&arr[i], &arr[j]);
        }
    }
    intercambiar(&arr[i + 1], &arr[alto]);
    return i + 1;
}

// Función de Quicksort
void quicksort(int arr[], int bajo, int alto) {
    //Límite superior y límite inferior
    if (bajo < alto) {
        //particiones del array
        int pi = particionar(arr, bajo, alto); // Particionar el array
        quicksort(arr, bajo, pi - 1); // Ordenar la parte izquierda
        quicksort(arr, pi + 1, alto); // Ordenar la parte derecha
    }
}

// Función para imprimir el array
void imprimirArray(int arr[], int tamaño) {
    //Recorrer el array
    for (int i = 0; i < tamaño; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    // Tamaño del array
    int tamaño = 10000;
    //malloc → Memoria dinámica
    int *array = (int *)malloc(tamaño * sizeof(int));
    //El array está vació
    if (array == NULL) {
        printf("Error de asignación de memoria\n");
        return 1;
    }

    // Inicializar el array con valores aleatorios
    srand(time(0));
    //Generar los valores
    for (int i = 0; i < tamaño; i++) {
        array[i] = rand() % 10000;
    }

    // Medir el tiempo de inicio antes de ordenar el array
    clock_t inicio = clock();
    // Ordenar el array usando el algoritmo Quicksort
    quicksort(array, 0, tamaño - 1);
    // Medir el tiempo de fin después de ordenar el array
    clock_t fin = clock();

    // Calcular y mostrar el tiempo de ejecución
    double tiempo_transcurrido = ((double)(fin - inicio)) / CLOCKS_PER_SEC;
    printf("Tiempo de ejecución: %f segundos\n", tiempo_transcurrido);

    free(array);
    return 0;
}
```

Código Python



```python
import random
import time

def quicksort(arr):
    """Ordena el array usando el algoritmo Quicksort."""
    if len(arr) <= 1:
        # Caso base: Si el array tiene uno o cero elementos, ya está ordenado.
        return arr
    else:
        # Seleccionar el pivote (en este caso, el elemento del medio del array).
        pivote = arr[len(arr) // 2]
        # Dividir el array en tres partes:
        # 1. Elementos menores que el pivote.
        izquierda = [x for x in arr if x < pivote]
        # 2. Elementos iguales al pivote.
        medio = [x for x in arr if x == pivote]
        # 3. Elementos mayores que el pivote.
        derecha = [x for x in arr if x > pivote]
        # Ordenar recursivamente las partes izquierda y derecha, y concatenar con la parte del pivote.
        return quicksort(izquierda) + medio + quicksort(derecha)

def main():
    # Tamaño del array
    tamaño = 10000
    # Crear un array de tamaño números enteros aleatorios entre 0 y 10000.
    array = [random.randint(0, 10000) for _ in range(tamaño)]

    # Medir el tiempo de inicio antes de ordenar el array.
    tiempo_inicio = time.time()
    # Ordenar el array usando el algoritmo Quicksort.
    array_ordenado = quicksort(array)
    # Medir el tiempo de fin después de ordenar el array.
    tiempo_fin = time.time()

    # Calcular el tiempo transcurrido en segundos.
    tiempo_transcurrido = tiempo_fin - tiempo_inicio
    # Mostrar el tiempo de ejecución del algoritmo Quicksort.
    print(f"Tiempo de ejecución: {tiempo_transcurrido:.6f} segundos")

# Ejecutar la función main si el script se ejecuta directamente.
if __name__ == "__main__":
    main()
```