INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

CARRERA:

INGENIERÍA EN SISTEMAS COMPUTACIONALES

Materia:

Paradigmas de programación

Profesor: García Floriano Andrés

Alumno:

Escobar Rodriguez Alfonso 3CV1

Fecha de entrega: 05 de marzo del 2024.

Objetivo.

Identificar las ventajas, desventajas de usar distintos tipos de lenguaje para una misma tarea, además de marcar si existe alguna diferencia en los tiempos de ejecución

Introducción.

Python es un lenguaje de programación orientado a objetos de alto nivel con semántica dinámica. Proporciona estructuras de datos integradas convenientes para la creación de scripts. Python también funciona bien como lenguaje adhesivo para combinar componentes de software. También es útil para el desarrollo de acción rápida (RAD).

C es un lenguaje de programación procedimental de propósito general con gran popularidad por su simplicidad y flexibilidad. Los programadores utilizan ampliamente el lenguaje para desarrollar sistemas operativos, aplicaciones y otro software complejo, lo que significa que transforma el código fuente del programa en un lenguaje legible por máquina. Después de la compilación, vincula archivos de objetos y crea un solo archivo ejecutable.

Las diferencias clave entre los lenguajes de programación C y Python son:

- C es un lenguaje de programación estructural, mientras que Python es un lenguaje de programación orientado a objetos.
- La sintaxis de Python es más fácil de entender que la de C.
- C es un lenguaje compilado y Python es un lenguaje interpretado.

- Python es un lenguaje de programación de propósito general, mientras que C se usa principalmente para aplicaciones relacionadas con hardware y código de bajo nivel.

Instrucciones.

Elabora un programa, en al menos dos lenguajes diferentes, que realicen lo siguiente:

1. Tenga una función que genere un arreglo de números enteros aleatorios.

a) El tamaño del arreglo será del al menos 1000 elementos.

2. Tenga una función para imprimir el contenido de los arreglos.
3. Proporciona al menos una función de búsqueda secuencial. Ejemplifica la búsqueda para el arreglo del paso 1.
4. Tenga una función que ordene el arreglo generado en el paso 1.
5. Ejemplifica la búsqueda con el arreglo resultante del paso 4.

6. Calcula el tiempo de ejecución de elementos 1 a 5.

Desarrollo

Lenguaje C:

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

// Función para generar arreglo
int *generar_arreglo(int n, int min, int max) {
    int *arr = (int *)malloc(n * sizeof(int));
    if (arr == NULL) {
        printf("Error: No se pudo asignar memoria para el arreglo\n");
        exit(1);
    }
    for (int i = 0; i < n; i++) {
        arr[i] = (rand() % (max - min + 1)) + min;
    }
    return arr;
}
```

Generación de arreglo: La función generar_arreglo(int n, int min, int max) crea un arreglo de tamaño n con números aleatorios enteros en el rango de min a max. Utiliza la función malloc para asignar memoria dinámica al arreglo.

```
// Función para ordenar un arreglo
void ordenar(int *arreglo, int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arreglo[j] > arreglo[j + 1]) {
                int temp = arreglo[j];
                arreglo[j] = arreglo[j + 1];
                arreglo[j + 1] = temp;
            }
        }

    }
}
```

Ordenamiento de arreglo: La función ordenar(int *arreglo, int n) ordena un arreglo dado utilizando el algoritmo de ordenamiento de burbuja. Este algoritmo compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto.

```
// Función de búsqueda secuencial
int busqueda_secuencial(int *arr, int n, int numero) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == numero) {
            return i;
        }
    }
    return -1;
}
```

Búsqueda secuencial: La función busqueda_secuencial(int *arr, int n, int numero) realiza una búsqueda secuencial en un arreglo dado arr para encontrar el numero dado. Si el número se encuentra en el arreglo, la función devuelve la posición donde se encuentra. Si el número no está en el arreglo, devuelve -1.

```
void calcular_tiempo_ejecucion() {
    clock_t start_time = clock();

    srand(time(NULL));

    int *arreglo_original = generar_arreglo(10, 1, 100);
    int *arreglo_ordenado = (int *)malloc(10 * sizeof(int));
    if (arreglo_ordenado == NULL) {
        printf("Error: No se pudo asignar memoria para el arreglo\n");
        exit(1);
    }
    for (int i = 0; i < 10; i++) {
        arreglo_ordenado[i] = arreglo_original[i];
    }
    ordenar(arreglo_ordenado, 10);

    // Imprimir los arreglos
    printf("Arreglo original:");
    for (int i = 0; i < 10; i++) {
        printf(" %d", arreglo_original[i]);
    }
    printf("\nArreglo ordenado:");
    for (int i = 0; i < 10; i++) {
        printf(" %d", arreglo_ordenado[i]);
    }
    printf("\n");

    int numero;
    printf("Ingrese el número a buscar: ");
    scanf("%d", &numero);

    int position_original = busqueda_secuencial(arreglo_original, 10, numero);
    int position_ordenado = busqueda_secuencial(arreglo_ordenado, 10, numero);

    // Imprimir los resultados de la búsqueda secuencial
    if (position_original != -1) {
        printf("Para la lista original: El numero %d se encuentra en la posición
%d\n", numero, position_original + 1);
    } else {
```

```
        printf("Para la lista original: El numero %d no se encuentra en la lista\n",
numero);
    }
    if (position_ordenado != -1) {
        printf("Para la lista ordenada: El numero %d se encuentra en la posición
%d\n", numero, position_ordenado + 1);
    } else {
        printf("Para la lista ordenada: El numero %d no se encuentra en la lista\n",
numero);
    }

    clock_t end_time = clock();
    double execution_time = (double)(end_time - start_time) / CLOCKS_PER_SEC;
    printf("Tiempo de ejecución del programa: %f segundos\n", execution_time);
}
```

Cálculo de tiempo de ejecución: La función calcular_tiempo_ejecucion() mide el tiempo de ejecución del programa utilizando la función clock() de la biblioteca time.h. Genera dos arreglos, uno aleatorio y otro ordenado, luego solicita al usuario un número para buscar en ambos arreglos, realiza la búsqueda secuencial en cada uno de los arreglos y muestra los resultados. Finalmente, imprime el tiempo total de ejecución del programa.

```
int main() {
    calcular_tiempo_ejecucion();
    return 0;
}
```

Python:

```
import random
import time

# Función para generar arreglo
def generar_arreglo(n, min, max):
    return [random.randint(min, max) for _ in range(n)]
```

Generación de arreglo: La función generar_arreglo(n, min, max) crea un arreglo de longitud n con números aleatorios enteros en el rango de min a max.

```
# Función para ordenar un arreglo
def ordenar(arreglo):
    return sorted(arreglo)
```

Ordenamiento de arreglo: La función ordenar(arreglo) ordena un arreglo dado utilizando el método sorted() de Python, que devuelve una nueva lista ordenada.

```
# Función de búsqueda secuencial
def busqueda_secuencial(arr, numero):
    for i in range(len(arr)):
        if arr[i] == numero:
            return "El numero {} se encuentra en la posición {}".format(numero, i+1)
    return "El numero: {} no se encuentra en la lista".format(numero)
```

Búsqueda secuencial: La función busqueda_secuencial(arr, numero) realiza una búsqueda secuencial en un arreglo dado arr para encontrar el numero dado. Si el número se encuentra en el arreglo, la función devuelve un mensaje indicando la posición donde se encuentra el número. Si el número no está en el arreglo, devuelve un mensaje indicando que no se encontró.

```
def calcular_tiempo_ejecucion():
    start_time = time.time()

    arreglo_original = generar_arreglo(10, 1, 100)
    arreglo_ordenado = ordenar(arreglo_original)

    # Función para imprimir los arreglos
    print("Arreglo original:", arreglo_original)
    print("Arreglo ordenado:", arreglo_ordenado)

    numero = int(input("Ingrese el número a buscar: "))
    position_original = busqueda_secuencial(arreglo_original, numero)
    position_ordenado = busqueda_secuencial(arreglo_ordenado, numero)

    # Imprimir los resultados de la busqueda secuencial
    print("Para la lista original:\n", position_original)
    print("Para la lista ordenada:\n", position_ordenado)

    end_time = time.time()
    execution_time = end_time - start_time
    print(f"Tiempo de ejecución del programa: {execution_time} segundos")


calcular_tiempo_ejecucion()
```

Cálculo de tiempo de ejecución: La función calcular_tiempo_ejecucion() mide el tiempo de ejecución del programa. Primero, genera un arreglo aleatorio, lo ordena, luego solicita al usuario un número para buscar en ambos arreglos (el original y el ordenado), realiza la búsqueda secuencial en cada uno de los arreglos y muestra los resultados. Finalmente, imprime el tiempo total de ejecución del programa.

Conclusiones

Python es un lenguaje de programación interpretado de alto nivel, mientras que C es un lenguaje de programación compilado de nivel medio. Esto significa que Python es más fácil de aprender y usar, pero puede ser más lento en términos de velocidad de ejecución en comparación con C.

Sintaxis: La sintaxis de Python es más limpia y legible en comparación con C. Por ejemplo, no hay necesidad de declarar tipos de variables en Python, mientras que en C es necesario. Además, Python usa sangrías para definir bloques de código, mientras que C utiliza llaves {}.

Gestión de memoria: En Python, la gestión de memoria es automática y administrada por el recolector de basura, lo que hace que sea más fácil de programar y menos propenso a errores de memoria. En C, es necesario gestionar manualmente la memoria usando funciones como malloc y free, lo que puede llevar a errores si no se manejan correctamente.

Longitud del código: En general, el código en Python tiende a ser más corto y conciso en comparación con el código en C debido a su sintaxis más expresiva y a la gestión automática de la memoria. Esto hace que Python sea más rápido de escribir y más fácil de mantener.

Complejidad: La complejidad de ambos códigos puede variar dependiendo de la perspectiva. Python a menudo se considera más simple y fácil de entender debido a su sintaxis clara y a la gestión automática de la memoria. Sin embargo, en términos de complejidad de tiempo y espacio, la eficiencia de Python puede ser menor que la de C para aplicaciones que requieren un rendimiento extremadamente alto.

Ventajas y desventajas: Python es más rápido para desarrollar y más fácil de mantener debido a su sintaxis simple y gestión automática de la memoria, lo que lo hace ideal para prototipos rápidos y aplicaciones donde el tiempo de desarrollo es crítico. Sin embargo, puede ser más lento en términos de rendimiento en comparación con C, especialmente para aplicaciones intensivas en CPU. C, por otro lado, ofrece un mayor control sobre la memoria y un mejor rendimiento, lo que lo hace más adecuado para aplicaciones de bajo nivel y sistemas embebidos donde la eficiencia es crucial.