INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

INGENIERÍA EN SISTEMAS COMPUTACIONALES

PRACTICA 1

UNIDAD DE APRENDIZAJE:

ANALISIS Y DISEÑO DE ALGORITMOS

PROFESOR:

ANDRES GARCIA FLORIANO

ALUMNOS:

LOPEZ MARTINEZ JONATHAN

NAVA MONTIEL ERASMO

SILVA VÁZQUEZ DIEGO

GRUPO: 3CV5

CODIGO EN C

```c
void multiply_primary_algorithm(const char *num1, const char *num2, char *result) {
    int len1 = strlen(num1);
    int len2 = strlen(num2);
    int *a = (int *)malloc(len1 * sizeof(int));
    int *b = (int *)malloc(len2 * sizeof(int));
    int *res = (int *)calloc(len1 + len2, sizeof(int));
    int i,j;

    for ( i = 0; i < len1; i++) a[i] = num1[len1 - 1 - i] - '0';
    for ( j = 0; j < len2; j++) b[j] = num2[len2 - 1 - j] - '0';

    int k, p;
    for ( k = 0; k < len1; k++) {
        int carry = 0;
        for ( p = 0; p < len2; p++) {
            int temp = a[k] * b[p] + res[k + p] + carry;
            carry = temp / 10;
            res[k + p] = temp % 10;
        }
        res[k + len2] = carry;
    }

    int idx = len1 + len2 - 1;
    while (idx > 0 && res[idx] == 0) idx--;

    int pos = 0;
    for (; idx >= 0; idx--) result[pos++] = res[idx] + '0';
    result[pos] = '\0';

    free(a);
    free(b);
    free(res);
}
```

```c
int main() {

    char num1[256];
    char num2[256];

    printf("Introduce el primer número: ");
    scanf("%s", num1);
    printf("Introduce el segundo número: ");
    scanf("%s", num2);
    char result[100] = {0};

    multiply_primary_algorithm(num1, num2, result);
    printf("%s * %s = %s\n", num1, num2, result);

    return 0;
}
```

Para este código en C se hizo lo siguiente:

1. Función main. Esta función se encarga de la interacción con el usuario:

- **Declara arreglos de caracteres** (char num1[256], num2[256]) para almacenar los números que el usuario escribirá.
- **Pide al usuario** que introduzca los dos números usando printf.
- **Lee los números** desde la consola y los guarda en los arreglos usando scanf.
- **Llama a la función** multiply_primary_algorithm para que realice el cálculo.
- **Imprime el resultado** final en la pantalla.

2. Función multiply_primary_algorithm (El cerebro de la operación)

Aquí ocurre la magia, siguiendo los mismos pasos que la versión de Python:

- **Preparación**:
    - Obtiene la longitud de los dos números con strlen.
    - **Asigna memoria dinámicamente** con malloc y calloc para crear arreglos de enteros (a, b, res) que contendrán los dígitos.
    - Convierte los números (que son char) a enteros (int) y los guarda en los arreglos en orden inverso. Lo hace con la operación num1[len1 - 1 - i] - '0', un truco común en C para convertir un dígito de carácter a su valor numérico.
- **Multiplicación dígito a dígito**:
    - Usa dos bucles for anidados para multiplicar cada dígito del primer número por cada dígito del segundo, exactamente igual que en el algoritmo de papel y lápiz.
    - Gestiona el "acarreo" (carry) para pasarlo a la siguiente columna.
- **Limpieza y formato del resultado**:
    - Encuentra la posición del primer dígito para ignorar los ceros a la izquierda.
    - Convierte el resultado (que está en un arreglo de enteros) de nuevo a una cadena de caracteres, usando la operación inversa res[idx] + '0'.
    - Añade el **carácter nulo \0** al final de la cadena de resultado. Esto es crucial en C para indicar que la cadena ha terminado.
- **Liberación de memoria**:
    - Al final, usa free(a), free(b) y free(res) para liberar la memoria que se solicitó al principio.

CODIGO EN PYTHON

```python
def multiply_primary_algorithm(num1, num2):
    # Convertir cadenas a listas de enteros en orden inverso
    a = list(map(int, num1))[::-1]
    b = list(map(int, num2))[::-1]

    # Inicializar arreglo de resultado con ceros
    result = [0] * (len(a) + len(b))

    # Multiplicar dígito por dígito
    for i in range(len(a)):
        carry = 0
        for j in range(len(b)):
            temp = a[i] * b[j] + result[i + j] + carry
            carry = temp // 10
            result[i + j] = temp % 10
        result[i + len(b)] = carry

    # Eliminar ceros a la izquierda
    while len(result) > 1 and result[-1] == 0:
        result.pop()

    # Convertir a cadena en orden correcto
    return ''.join(map(str, result[::-1]))

# Pruebas con números de n dígitos (n potencia de 2)
test_cases = [
    ("12", "13"),      # n=2
    ("1234", "5678"),  # n=4
    ("12345678", "87654321")  # n=8
]

for num1, num2 in test_cases:
    print(f"{num1} * {num2} = {multiply_primary_algorithm(num1, num2)}")
```

El código en Python se explica de la siguiente forma:

La función multiply_primary_algorithm:

1. **Preparación de los números**:
    - Convierte las cadenas de texto (num1, num2) en listas de dígitos enteros.
    - Invierte el orden de estas listas. Por ejemplo, "123" se convierte en [3, 2, 1]. Esto facilita los cálculos, ya que se empieza por el dígito menos significativo (el de la derecha).
2. **Inicialización del resultado**:
    - Crea una lista llamada result llena de ceros. Su tamaño es la suma de las longitudes de los dos números, que es el tamaño máximo que podría tener el resultado.
3. **Multiplicación dígito a dígito (El núcleo del algoritmo)**:
    - Utiliza dos bucles anidados para multiplicar cada dígito del primer número por cada dígito del segundo.
    - Suma este producto al valor que ya está en la posición correspondiente del result, más cualquier "acarreo" (carry) de la operación anterior.
    - El nuevo dígito para la posición actual se calcula con el módulo 10 (temp % 10).
    - El nuevo acarreo para la siguiente posición se calcula con la división entera por 10 (temp // 10).
4. **Limpieza del resultado**:
    - Elimina los ceros no significativos que puedan haber quedado a la izquierda del resultado. Por ejemplo, si el resultado es 0156, lo convierte en 156.
5. **Formato final**:
    - Vuelve a invertir la lista result para que los dígitos estén en el orden correcto.
    - Convierte la lista de enteros de nuevo en una sola cadena de texto.

LINK DE GITHUB

GitHub - Lil-Mitowsker/An-lisis-Y-Dise-o-de-Algoritmos