Tarea 3 - Programación y algoritmos
Giovanni Gamaliel López Padilla

Problema 1

Programa que verifique si un número de entrada dado es palíndomo. Un palíndromo, es un número o string que se lee igual en un sentido que en otro (por ejemplo: Ana, 121...). Usar una función que regrese un numero invertido (al revés).
El algoritmo que se creo es el siguiente:

```
1    reverse = 0
2    number = 12345
3    while number != 0:
4        aux = number % 10
5        reverse = reverse * 10 + aux
6        number = number / 10
```

Al final la variable reverse contendra el numero escrito al reves, esto es porque la operación de la linea 4 se obtiene el número que se encuentra en la posicion de las unidades, por ende es sumada en la linea 4. Al ser number una variable de tipo entero entonces al momento de realizar la operación de la linea 5 el numero de la parte decimal es eliminada, por ende en la proxima iteración en la linea 3 se tomara el que se encontraba en la posicion de las centenas. La función que realiza el algoiritmo es llamada reverse y se encuentra en el archivo reverse.h. El programa se encuentra en la carpeta Problema_1. Para compilar se uso el siguiente comando:

```
1    gcc -Wall -Wextra -Werror -pedantic -ansi -o main.out main.c -std=c11
```

El output esperado para un número palindromo es el siguiente:

```
1
2    Escribe el numero que quieres saber si es palindromo:
3    123454321
4
5
6    El numero 123454321 si es palindromo
```

El output esperado para un número no palindromo es el siguiente:

```
1
2    Escribe el numero que quieres saber si es palindromo:
3    123456789
4
5
6    El numero 123456789 no es palindromo
```

Problema 2

Escribe un programa que implemente la estructura de datos Stack usando un arreglo de enteros. Se puede asumir que la capacidad máxima del Stack es de 10 enteros.
El stack es una estructura de datos que permite almacenar información. El modo de acceso a los datos es de tipo ultimo en entrar, primero en salir (LIFO por sus siglas en inglés). El manejo de los datos se maneja con dos operaciones llamadas *push* y *pop*, las cuales ingresan y retiran información respectivamente del stack. En la figura 1 se representa visualmente el stack junto a sus operaciones.

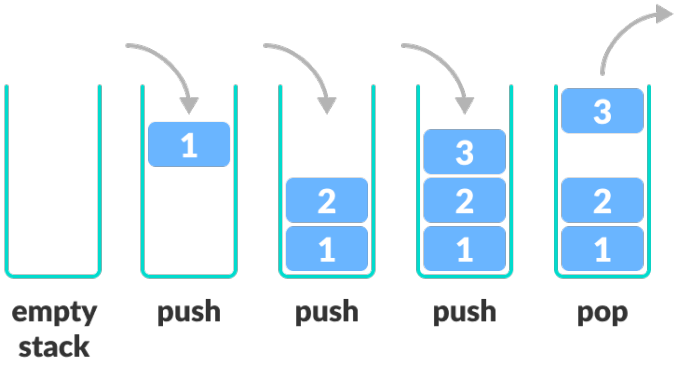

Figura 1: Representación de la estructura de datos stack. Imagen obtenida de Programiz.

El pograma realiza las siguentes acciones:

```
Create stack
Push(2)
Push(3)
Push(4)
Push(5)
Push(6)
Pop
Pop
Pop
Push(4)
Push(5)
```

El output del programa es el siguiente:

```
_____________________________
Push number 2:
2 |
_____________________________
Push number 3:
2 | 3 |
_____________________________
Push number 4:
2 | 3 | 4 |
_____________________________
Push number 5:
2 | 3 | 4 | 5 |
_____________________________
Push number 6:
2 | 3 | 4 | 5 | 6 |
_____________________________
pop:
2 | 3 | 4 | 5 |
_____________________________
pop:
2 | 3 | 4 |
_____________________________
pop:
2 | 3 |
_____________________________
Push number 4:
2 | 3 | 4 |
_____________________________
Push number 5:
2 | 3 | 4 | 5 |
```

El programa se encuentra en la carpeta Problema_2. Para compilarlo se uso el siguiente comando:

```
gcc -Wall -Wextra -Werror -pedantic -ansi -o main.out main.c -std=c11
```

Problema 3

Escribe una función que reciba un arreglo de enteros y que calcule el producto acumulado para cada entrada usando la ecuación 1. Se espera que la complejidad del algoritmo sea lineal O(n). Se puede asumir que la longitud máxima del arreglo será 50.

$$a[i] = \prod_{j \neq i} a[j] \tag{1}$$

El algoritmo que se ideo es el siguiente:

```
1  all_product=1
2  size=len(data)
3  product = []
4  for i in range(size):
5      all_product = all_product * data[i]
6  for i in range(size):
7      product.push(all_product / data[i])
```

En el ciclo de la linea 4 se calcula el producto de todos los números contenidos en la lista de datos. En el ciclo de la linea 6 se guardan los productos acumulados siguiendo la ecuación 1. Se esta manera se obtiene con una complejidad de O(n). El programa se encuentra en la carpeta Problema_3. El programa acepta valores de entrada especificando cuantos valores se introducira. Un ejemplo de esto es con la siguiente lista de valores: $data = \{3, 5, 3, 2\}$, el output que da el programa es el siguiente:

```
Deseas usar el programa con los datos de prueba?(Y/n): n

Escribe el tamano de los datos: 4
Escribe el numero 1 de 4: 3
Escribe el numero 2 de 4: 5
Escribe el numero 3 de 4: 3
Escribe el numero 4 de 4: 2

_____________________________________________

Numbers     Product
3           30
5           18
3           30
2           45
```

Los datos de ejemplo son creados aleatoriamente. El output de uno de ellos es el siguiente:

```
Deseas usar el programa con los datos de prueba?(Y/n): y

_____________________________________________

Numbers     Product
8           87210
17          41040
2           348840
19          36720
9           77520
15          46512
```

El comando para compilar el programa es el siguiente:

```
gcc -Wall -Wextra -Werror -pedantic -ansi -o main.out main.c -std=c11
```

Problema 4

Escribe una función que reciba dos arreglos de enteros ordenados en forma no decreciente, y combine dichos arreglos en un solo arreglo también ordenado en forma no decreciente.

Se creo un programa el cual recibe dos listas de numeros de n_1 y n_2, donde n_1 y n_2 pueden tomar cualquier valor entero positivo. Estas listas pueden estar desordenadas, ya que internamente se ordenan de menor a mayor cada lista. Este paso se realiza siguiendo el algoritmo de *quick sort* el cual se encuentra en el archivo quick_sort.h. La acción de unir a estas dos listas se encuentra el el archivo merge.h. El algoritmo que se empleo es el siguiente:

```
1     i = 0
2     j = 0
3     k = 0
4     while i<n1 or j<n2:
5         if data1[i] < data2[j]:
6             if i<n1:
7                 merge[k] = data1[i]
8                 i += 1
9             else:
10                merge[k] = data2[j]
11                j += 1
12        else:
13            if i<n1:
14                merge[k] = data2[j]
15                j += 1
16            else:
17                merge[k] = data1[i]
18                i += 1
19        k += 1
```

Las variables inicializadas en las lineas 1 al 3 son las encargadas del control en las posiciones de los datos 1, datos 2 y su union (merge). El ciclo iniciado en la linea 4 se detendra cuando las posiciones de cada arreglo hayan llegado a su maximo dando por resultado el arreglo merge con todos los datos. El if de la linea 5 decide que lista de datos será asignada, ya que se quiere un arreglo creciente. Los if de las lineas 6 y 13 realizan el control de no acceder a un valor de una lista dos veces o a una memoria invalida, ya que, puede darse el caso que un arreglo haya sido escrito totalmente en merge y su ultimo elemento sea menor a los elementos restantes del otro arreglo que aun no han sido escritos.

El programa se encuentra en la carpeta Problema_4. El programa tiene la opción de recibir valores de un usuario o crear datos de forma aleatoria. El output esperado del programa con los valores $data_1 = \{-1, 4, 6, 1\}$ y $data_2 = \{30, 1, 4, 2, 5, 7\}$ es el siguiente:

```
1     Deseas usar el programa con los datos de prueba?(Y/n): n
2     Escribe el tamano de los datos 1: 4
3     Escribe el tamano de los datos 2: 6
4
5     Datos 1:
6     Escribe el numero 1 de 4: -1
7     Escribe el numero 2 de 4: 4
8     Escribe el numero 3 de 4: 6
9     Escribe el numero 4 de 4: 1
10
11    Datos 2:
12    Escribe el numero 1 de 6: 30
13    Escribe el numero 2 de 6: 1
14    Escribe el numero 3 de 6: 4
15    Escribe el numero 4 de 6: 2
16    Escribe el numero 5 de 6: 5
17    Escribe el numero 6 de 6: 7
18
19
```

```
Numbers 1:
-1 1 4 6

Numbers 2:
1 2 4 5 7 30

Merge numbers:
-1 1 1 2 4 4 5 6 7 30
```

El output para los valores generados aleatoriamente es el siguiente:

```
Deseas usar el programa con los datos de prueba?(Y/n): y

Numbers 1:
-46 -25 -15 27 39

Numbers 2:
-50 -43 -32 8 13 38 48

Merge numbers:
-50 -46 -43 -32 -25 -15 8 13 27 38 39 48
```

El comando para compilar el programa es el siguiente:

```
gcc -Wall -Wextra -Werror -pedantic -ansi -o main.out main.c -std=c11
```