PRÁCTICA 4

IMPORTANTE: para cada función pedida en los TADs analizar, diseñar y programar todos los casos:

- **Si la operación se puede llevar a cabo, determinar que retorna la función y el estado de las distintas variables y / o parámetros.**
- **Casos en que la operación no se puede llevar a cabo: a modo de ejemplo, realizar la operación `Top` en una pila vacía. Determinar precondiciones y postcondiciones, decidir qué realiza la función en estos casos, programar y documentar. Las funciones deben funcionar correctamente para todos los casos.**
- **Analizar y justificar los parametros de las funciones. Detectar para cada parámetro si está pasado por valor o por referencia y justificar el por qué.**

1. PILAS

1) Implementar el TAD Pila utilizando arreglos. Una posible implementación del tipo sería:

```
typedef struct
{
  int indice;                    // tope de la pila
  Tipo_Dato pila[TamMax];        // pila
  int max;                       // máximo de la pila
}Pila_T;
```

El TAD debe disponer de las siguientes funciones:

1. `int Crear_Pila(Pila_T *pila);`
   // crea la pila pasada como parámetro: con 0 elementos y asignando el valor correcto al resto de los campos. O esta función puede no recibir parámetros y devolver un dato de tipo Pila_T
2. `int Q_Push(PilaA_T *pila, Tipo_Dato x );`
   // Añade x a la pila, si la misma no está llena.
3. `Tipo_Dato Q_Pop(Pila_T *pila);`
   // Saca un elemento de la pila si la pila no está vacía. La función retorna dicho dato.
4. `int Pila_Llena(Pila_T pila);`
   // Retorna True si la pila está llena. False caso contrario.
5. `int Pila_Vacia(Pila_T pila);`
   // retorna True si la pila está vacía. False caso contrario.

6. `Tipo_Dato Q_Top(Pila_T pila);`
   // Retorna el dato que está en el tope de la pila, pero sin sacarlo (no modifica la pila)

7. `int Longitud_Pila(Pila_T pila);`
   // Retorna la longitud de la pila.

8. `int Vaciar_Pila(Pila_T *pila);`
   // Vacia la pila pasada como parámetro.

2) Implementar el TAD Pila utilizando punteros. Se piden las mismas operaciones para que en el punto 1. Si alguna operación no tiene sentido justificar. Una posible implementación del tipo sería:

```
struct Nodo
{
   Tipo_Dato dato;
   struct Nodo *sig;
}

typedef struct
{
   int dim;   // cantidad de datos de la pila
   struct Nodo *pila;  // puntero a la pila
}Pila_T;
```

3) Usando una pila de caracteres, indicar si una palabra es un palíndromo (que se lee igual en ambos sentidos). Ayuda: leer la palabra caracter a caracter, armando a la vez un string y una pila. Luego, utilizar la pila y el string para comparar los caracteres. Puede utilizar la función `getchar()` para leer los caracteres hasta el fin de línea.

4) Usando el TAD pila, crear una pila de enteros y agregar datos a la pila. Luego, imprimir los datos de la pila. Luego de imprimir los datos, la pila original no debe verse modificada (utilizar estructuras auxiliares, como otra pila).

5) ¿Cuál es la salida del siguiente fragmento de código asumiendo que la pila es una pila de enteros?

```
int x = 4, y;

Crear_Pila(&P);

Q_Push(&P,x);

printf ("\%d ",Q_Top(P)) ;

y = Q_Pop(&P) ;
Q_Push(&P,32);
Q_Push(&P,Pop(&P));

do {
  printf (" \%d",Q\Pop(&P));
} while(!Pila_Vacia(P));
```

6) Obtener una secuencia de 10 elementos reales, guardarlos en un array y ponerlos en una pila. Imprimir la secuencia original y, a continuación, imprimir la pila extrayendo los elementos.

7) El tipo de dato pila se puede utilizar para determinar si una expresión está "balanceada". Por ejemplo, tomando una expresión algebraica como: $7+[5*(3+8)]/(7+9)$ se puede determinar que la expresión está balanceada: cada símbolo de apertura "{", "[" o "(" tiene su símbolo de cierre "}", "]" o ")". Escribir un programa que dada una secuencia de caracteres, determine si la secuencia está balanceada o no. Puede utiizar la función `getchar()` para leer los caracteres hasta el fin de linea.

2. Colas

8) Implementar el TAD cola utilizando arreglos. Se puede utilizar la siguiente estructura:

```
#define Max 100

typedef struct
{
  int frente;
  int final;
  Tipo_Dato Cola[Max];      // arreglo con los datos de la cola
}Cola_T;
```

1. `int Crear_Cola(Cola_T *cola);`
   // crea la cola pasada como parámetro: con 0 elementos y asignando el valor correcto al resto de los campos.

2. `int S_Push(Cola_T *cola, Tipo_Dato x );`
   // Añade x a la cola, si la misma no está llena.

3. `Tipo_Dato S_Pop(Cola_T *cola);`
   // Saca un elemento de la cola si la cola no está vacía. La función retorna dicho dato.

4. `int Cola_Llena(Cola_T cola);`
   // Retorna True si la cola está llena. False caso contrario.

5. `int Cola_Vacia(Cola_T cola);`
   // retorna True si la cola está vacía. False caso contrario.

6. `Tipo_Dato S_Top(Cola_T cola);`
   // Retorna el dato que está en el tope de la cola, pero sin sacarlo (no modifica la cola)

7. `int Longitud_Pila(Cola_T cola);`
   // Retorna la longitud de la cola.

8. `int Vaciar_Cola(cola_T *cola);`
   // Vacia la cola pasada como parámetro.

9) Implementar el TAD cola utilizando punteros. Se puede utilizar la siguiente estructura:

```
struct Nodo
{
   Tipo_Dato dato;
   struct Nodo *sig;
}

typedef struct
{
  struct Nodo *frente;
  struct Nodo *final;
  int size;
}Cola_T;
```

1. `int Crear_Cola(Cola_T *cola);`
   // crea la cola pasada como parámetro: con 0 elementos y asignando el valor correcto al resto de los campos (frente = final = NULL).

2. `int S_Push(Cola_T *cola, Tipo_Dato x );`
   // Añade x a la cola.

3. `Tipo_Dato S_Pop(Cola_T *cola);`
   // Saca un elemento de la cola si la cola no está vacía. La función retorna dicho dato.

4. `int Cola_Vacia(Cola_T cola);`
   // retorna True si la cola está vacía. False caso contrario.

5. `Tipo_Dato S_Top(Cola_T cola);`
   // Retorna el dato que está en el tope de la cola, pero sin sacarlo (no modifica la cola)

6. `int Longitud_Cola(Cola_T cola);`
   // Retorna la longitud de la cola.

7. `int Vaciar_Cola(Cola_T *cola);`
   // Vacia la cola pasada como parámetro.

10) Se tiene una pila de enteros positivos. Con las operaciones básicas de pilas y colas escribir un programa para poner todos los elementos que son par de la pila en una cola. Luego, mostrar los elementos de la cola. NO acceder a la estructura, usar funciones del TAD.

11) Reescribir el ejercicio 3) para descubrir si una frase es palíndroma. Esta vez, utilizar pilas y colas de caracteres. NO acceder a la estructura, usar funciones del/los TAD/s.

12) Escribir un programa en el que se generen 100 números aleatorios entre -25 y +25 y se guarden en una cola. Una vez creada la cola, implementar una función que forme otra cola con los números negativos de la cola original. NO acceder a la estructura, usar funciones del TAD.

13) Escribir una función que tenga como argumento dos colas del mismo tipo y devuelva True si las colas son idénticas, False caso contrario. NO acceder a la estructura, usar funciones del TAD.

14) Encontrar un número capicúa leído del dispositivo estándar de entrada (usando pilas o colas o ambas). NO acceder a la estructura, usar funciones del TAD.

15) Escribir una función para crear una copia de una cola determinada. La función tendrá dos argumentos, el primero es la cola origen y el segundo la cola que va a ser la copia. Las operaciones que se han de utilizar serán únicamente las definidas para el tipo cola. En esta función no se acceden a las estructuras del tipo, solo se deben invocar funciones del TAD.

16) Se tiene una lista de enteros. Escribir una función que imprima los elementos de la lista en orden inverso. NO acceder a la estructura, usar funciones del TAD.