Guía Práctica 11: Tipos de datos abstractos (TDA)

Ejercicio 1: Implementar el **TDA contador**. Su interfaz debe permitir crear variables del tipo e implementar las operaciones para incrementar (++) y decrementar (--) su valor. Se propone la siguiente interfaz pública:

```
contador crear (int valor);
void incrementar (contador & c, int pasos) ; //incrementa c en ´pasos'
void decrementar (contador & c, int pasos) ; //decrementa c en ´pasos'
int toInteger (contador c);
```

- Copiar y probar el siguiente programa que hace uso del nuevo TDA:

```
#include <iostream>
#include <stdio.h>
#include "contador.h"
using namespace std;

int main (void) {
    contador c1;
    c1=crear( 10);
    cout << "El programa comienza en: " <<endl;
    for (c1; toInteger(c1) > 0; decrementar(c1,1)){
       cout << toInteger(c1) << " segundos"<<endl;
    }
    return 0;
}
```

Ejercicio 2:

a) Implementar los siguientes **TDAs** y operaciones que permitan cargar sus datos y mostrarlos por pantalla, como así también aquellas que permitan calcular su área y perímetro.

- **Círculo:**
  - Archivo de interfaz
    - Definir la estructura *Circulo*
    - *Circulo crear_circulo(...);*
    - *float perimetro(Circulo c);*
    - *float area(Circulo c);*
- **Cuadrado**
  - Archivo de interfaz
    - Definir la estructura *Cuadrado*
    - *Cuadrado crear_cuadrado(...);*
    - *float perimetro(Cuadrado c);*
    - *float area(Cuadrado c);*
- **Rectángulo**
  - Archivo de interfaz

        - Definir la estructura *Rectangulo*
        - *Rectangulo crear(...);*
        - *float perimetro(Rectangulo r);*
        - *float area(Rectangulo r);*
- **Triángulo**
    - Archivo de interfaz
        - Definir la estructura *Triangulo*
        - *Triangulo crear(...);*
        - *float perimetro(Triangulo r);*
        - *float area(Triangulo r);*

b) Implementar un programa cliente que muestre un menú con las distintas formas geométricas y que al acceder a la elegida se presente un submenú con la posibilidad de crear, mostrar el perímetro y el área de la figura elegida. *Nota: para poder mostrar perímetro y área, la figura debe estar creada.*

Ejercicio 3: Definir la representación necesaria para manipular un **TDA Hora**. Este nuevo tipo de datos deberá almacenar la hora, los minutos y los segundos en forma separada y en notación de 24 horas.

a) Implementar el **TDA Hora** y sus operaciones asociadas:

- *Hora ponerAHora (int hora, int min, int seg)* permite configurar el reloj de acuerdo a valores dados en cualquier momento.
- *void mostrar24(Hora h):* imprime la hora en notación 24 horas: hh:mm:ss.
- *void mostrar12(Hora h):* imprime la hora en notación 12 hs : hh:mm:ss a.m ó hh:mm:ss p.m según corresponda.
- *void adelantar (Hora & h, int minutos):* permite adelantar la hora una cantidad determinada de minutos.
- *void atrasar (Hora &h, int minutos):* permite atrasar la hora una cantidad determinada de minutos.
- *int diferenciaEnMinutos (Hora h1, Hora h2)*: devuelve la cantidad de minutos entre 2 horas.
- *int diferenciaEnSegundos(Hora h1, Hora h2)*: devuelve la cantidad de segundos entre 2 horas.

b) Escribir un programa, que dada una lista de eventos (por cada evento se almacena el nombre del mismo y su duración esperada en minutos) y la hora de inicio del primer evento, mostrar por pantalla la hora de comienzo y fin de cada uno de los eventos.

Nota: La lista de eventos se almacena en un arreglo estático de hasta 10 elementos.

Ejercicio 4: Para manejar las cuentas bancarias se decide definir un TDA denominado ***Cuenta***, que permita almacenar y administrar la cuenta de un cliente bancario. Una cuenta tiene almacenada la siguiente información: nro, saldo, los 10 últimos movimientos (para cada uno: Fecha de la operación, tipo de transacción: ‘d’-depósito, ‘e’-extracción y el monto correspondiente). Las operaciones requeridas para el ***TDA Cuenta*** son:

- *Inicializar* (tiene como parámetro una cuenta a la que hay que setearle los valores iniciales)
- *Depositar* (tiene como parámetros el monto –real- y la cuenta).
- *Extraer* (tiene como parámetros el monto –real- y la cuenta).
- *ConsultarSaldo* (tiene como parámetro la cuenta).
- *ConsultarUltimosMovimientos* (tiene como parámetro la cuenta): imprime un listado con los últimos 10 movimientos de la cuenta. Para cada transacción registrada se debe mostrar la siguiente información: *Fecha de la operación- Tipo de transacción -Monto de la transacción*

Nota 1: Para la operación Extraer, se debe verificar la disponibilidad de saldo en la cuenta.
Nota 2: Luego de cada operación sobre el TDA, se debe visualizar el saldo actualizado de la cuenta.
Nota 3: Considerar estructuras y funciones auxiliares, privadas al TDA.

Armar un programa (cliente) que haga uso de este TDA de cuenta que permita gestionar un menú para poder administrar hasta 100 cuentas bancarias, habilitando las opciones de crear (inicializar), depositar, extraer, consultar saldo y consultar los últimos movimientos.

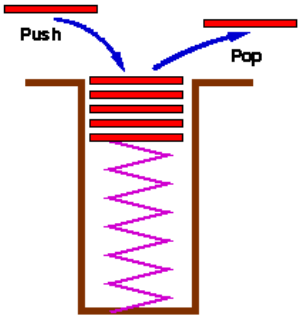
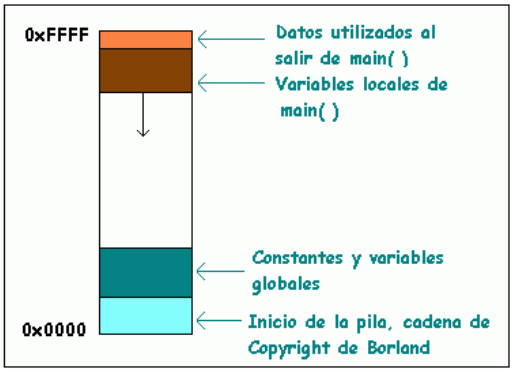
Ejercicio 5: según el calendario Gregoriano instaurado en el año 1582 se considera que los años divisibles por 4 son bisiestos excepto aquellos divisibles por 100 que no lo sean por 400.

a) Implementar el ***TDA Fecha*** y sus operaciones asociadas:
   - *Fecha cargarFecha(int dia,int mes,int anio):* permite asignar valores a una fecha.
   - *int dia (Fecha f)*: retorna el día de la fecha.
   - *int mes (Fecha f)*: retorna el mes de la fecha.
   - *int anio (Fecha f)*: retorna el año de la fecha.
   - *int esValida (Fecha f):* retorna 1si la fecha es válida y 0 si no lo es.
   - *int diferenciaFecha(Fecha f1, Fecha f2)*: retorna el número de días desde la fecha f2 a la fecha f1.
   - *int diaSemanaFecha(Fecha f):* retorna el día de la semana (0 equivale a lunes, 6 a domingo).
   - *void mostrarFechaEuropea (Fecha f):* imprime fecha en formato dd/mm/yyyy.
   - *void mostrarFechaAmericana (Fecha f):* imprime fecha en formato mm/dd//yyyy.

b) Escribir un programa que, haciendo uso del TDA Fecha, calcule el tiempo necesario para que una cantidad depositada en un banco consiga una determinada rentabilidad, ofreciendo como resultado la fecha en que se puede retirar el dinero original más la rentabilidad deseada. Suponer que el banco ofrece una tasa de interés del 0.75% a los 30 días.

c) Escribir otro programa que permita ingresar un mes y un año y que en base a estos 2 números se impriman todos los días de ese mes en formato Europeo.

Ejercicio 6: Implementar **TDA Pila:** una pila (stack en inglés) es una lista en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés Last In First Out, último en entrar, primero en salir) que permite almacenar y recuperar datos.
Para el manejo de los datos se cuenta con dos operaciones básicas: **apilar** (*push*), que coloca un objeto en la pila, y su operación inversa, **retirar** (o desapilar, *pop*), que retira el último elemento apilado.

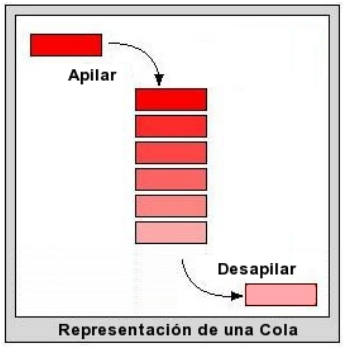
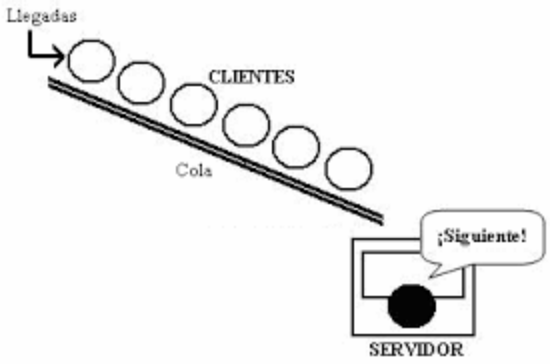
Pila: acceso LIFO	Ejemplo: pila de datos
	

En cada momento sólo se tiene acceso a la parte superior de la pila, es decir, al último objeto apilado (denominado **TOS**, *Top of Stack* en inglés). La operación **retirar** permite la obtención de este elemento, que es retirado de la pila permitiendo el acceso al siguiente (apilado con anterioridad), que pasa a ser el nuevo TOS. Por analogía con objetos cotidianos, una operación **apilar** equivaldría a colocar un plato sobre una pila de platos, y una operación **retirar** a retirarlo.

Operaciones

- *Crear:* se crea la pila vacía.
- *Apilar:* se añade un elemento a la pila (push).
- *Desapilar:* se elimina el elemento frontal de la pila (pop).
- *Cima:* devuelve el elemento que está en la cima de la pila (top o peek).
- *Vacía:* devuelve cierto si la pila está vacía o falso en caso contrario.

Ejercicio 7: Implementar **TDA Cola:** una cola es una estructura de datos, caracterizada por ser una secuencia de elementos en la que la operación de inserción push se realiza por un extremo y la operación de extracción pop por el otro. También se le llama estructura FIFO (del inglés First In First Out), debido a que el primer elemento en entrar será también el primero en salir.

Cola: acceso FIFO	Ejemplo: pedidos al Servidor
	

Operaciones

- *Crear:* se crea la cola vacía.
- *Encolar (push):* se añade un elemento a la cola. Se añade al final de esta.
- *Desencolar (sacar, salir, pop):* se elimina el elemento frontal de la cola, es decir, el primer elemento que entró.
- *Frente (consultar, front):* se devuelve el elemento frontal de la cola, es decir, el primero elemento que entró.

Ejercicio 8: Implementar el TDA Conjunto de enteros y sus operaciones asociadas:

- *Conjunto Vacio():* transforma un conjunto en el conjunto vacío.
- *Conjunto Insertar(Conjunto,int):* agrega un elemento a un conjunto.
- *Conjunto Eliminar(Conjunto,int): e*limina un elemento dado de un conjunto.
- *bool EsVacio(Conjunto):* indica si el conjunto es vacío.
- *Conjunto Union(Conjunto,Conjunto):* efectúa la unión de dos conjuntos.
- *Conjunto Interseccion(Conjunto,Conjunto):* intersección de 2 conjuntos.
- *bool Pertenece(Conjunto,int): i*ndica si un elemento pertenece a un conjunto.
- *bool EsSubconjunto(Conjunto,Conjunto):* indica si un conjunto es subconjunto de otro.
- *int Tamanio(Conjunto):* da la cardinalidad de un conjunto.
- *int Extraer(Conjunto):* retorna un elemento del conjunto y lo elimina.

Incluir operaciones o funciones para imprimir y leer hacia o desde la consola en el TDA

Ejercicio 9: Un número binario es un número compuesto por una cadena de **1**'s y **0**'s con cierto significado. Cómo sabemos, cada posición del número binario, de derecha a izquierda, representa una potencia de 2 (comenzando con la potencia 0) que se multiplica por el dígito binario correspondiente. Dependiendo de qué valor tiene en esa posición, se puede determinar el decimal correspondiente según la sumatoria:

$2^5 + 2^2 + 2^0 = 32 + 4 + 1 = 37$ (valor del binario 100101)

a) Implementar un **TDA Binario**, para representar binarios de 8 dígitos, el cual internamente debe almacenar el mismo como un número entero. El TDA debe implementar las siguientes operaciones:

- *void asignar(int *char ,Binario b):* recibe una cadena de unos y ceros y lo almacena el entero correspondiente.
- *char * valorBinario():* retorna como un arreglo de unos y ceros, el valor del binario.
- *Binario sumar(Binario b1, Binario b2):* suma dos binarios y devuelve el resultado en otro binario.
- *Binario mayor(Binario b1, Binario b2):* comprueba si un binario es > a otro.

b) Codificar un programa que haga uso del TDA Binario, en el cual se declaren 3 números binarios, uno inicializado en cero y los otros con valores. Muestre una parte del código que sume los dos últimos binarios y lo guarde en el primero.