PROGRAMACIÓN ORIENTADA A OBJETOS

GRADO EN INGENIERÍA INFORMÁTICA
SEGUNDO CURSO

DEPARTAMENTO DE INFORMÁTICA Y ANÁLISIS NUMÉRICO
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD DE CÓRDOBA

CURSO ACADÉMICO: 2012 – 2013

- **Número de práctica: 4 (versión 2)**

- **Objetivo**
  - Definición de la clase **Contorno**, que tendrá dos versiones
    a) Codificación con vectores
    b) Codificación con listas de puntos
  - Se utilizará la clase Punto2D

- **Definición de la clase Contorno**

  - **Descripción**
    - Un **contorno** es un tipo abstracto de datos que:
      - Es una línea compuesta por puntos: $\{p_1, p_2, \ldots, p_n\}$
      - Los puntos siguen una serie de forma que son vecinos o contiguos, es decir, la distancia euclídea entre dos puntos consecutivos en la serie debe ser menor o igual que $\sqrt{2}$
        - $\forall i \;\; d(p_i, p_{i+1}) \leq \sqrt{2}$
      - La línea de puntos es
        - cerrada: el último punto es vecino del primero
          - $d(p_n, p_1) \leq \sqrt{2}$
        - simple: no se cruza sobre sí misma (no tiene bucles)
          - $\forall i, j \; [(p_i = p_j) \rightarrow (i = j)]$

  - **Objetivo**
    - Comprobar que un tipo abstracto de datos es independiente de su implementación y codificación
    - Se deben hacer dos versiones de la clase **Contorno**
      a) Con vectores
      b) Con listas de puntos (estructura dinámica de memoria)
    - **Observación**
      - Ambas versiones deberán permitir la ejecución del programa denominado **practica_4.cpp** y utilizar el fichero de entrada

alicates.txt (fichero con los puntos del contorno de unos alicates).

- Ambas versiones crearán el fichero "contorno.hpp" y "contorno.cpp" con las declaraciones y funciones correspondientes.

IMPLEMENTACIÓN CON VECTOR

- **Objetivos**
  - Usar un atributo que es un vector de objetos de otra clase
  - Sobrecarga del operador []
  - Uso de la clase **string**

- **Atributos privados de Contorno**
  - nombreContorno: atributo de tipo string
  - numeroPuntos: atributo de tipo int
  - vectorPuntos: atributo que es un puntero a Punto2D

- **Funciones o métodos públicos**
  - Constructor
    - No recibe parámetros
    - Crea un objeto del tipo Contorno con los valores
      - nombreContorno: "" (cadena vacía)
      - numeroPuntos : 0
      - vectorPuntos: NULL
  - Constructor de copia
    - Recibe como parámetro un Contorno "c" pasado por referencia constante y crea una copia.
  - Destructor
    - Libera, si es necesario, la memoria ocupada por el Contorno

  - Métodos de **acceso** para los nuevos atributos propios
    - **getVectorPuntos**: devuelve el atributo vectorPuntos
    - **getNombreContorno**: devuelve el atributo "nombreContorno"
    - **getNumeroPuntos:** devuelve el atributo numeroPuntos
    - **Aviso**: estos métodos deben ser **const**
    - **getPunto**:
      - Recibe como parámetro "indice" de tipo entero
      - Devuelve una **referencia** al Punto2D que ocupa el lugar "indice".
    - **Sobrecarga del operador []**:
      - Recibe como parámetro "indice" de tipo entero
      - Devuelve una **referencia** al Punto2D que ocupa el lugar "indice".
      - Prototipo:
        Punto2D &operator[](int indice);

  - Métodos de **modificación** para los atributos
    - **setVectorPuntos**:
      - Recibe a "n" de tipo entero (valor por defecto: 0)
      - Recibe a "v" de tipo puntero a Punto2D. Es el vector de Puntos2D. (valor por defecto: NULL)

- Libera, si es necesario, la memoria que actualmente ocupa el vectorPuntos.
- Si v!=NULL reserva la nueva memoria necesaria y asigna los valores del vector recibido al atributo vectorPuntos.

▪ **setNombreContorno**:
- Recibe a "v" de tipo string y se lo asigna a la componente "nombre" del ContornoGrafico2D
- El valor por defecto es la cadena vacía

▪ **Sobrecarga del operador de asignación "="**
- Asigna al Contorno actual el Contorno pasado como parámetro
- Parámetro: un Contorno pasado como referencia constante
- Resultado: referencia al Contorno actual
- Prototipo
  inline Contorno & operator=(const Contorno &c);

◦ Funciones de lectura o escritura
▪ **escribirContornoGrafico2D:**
- Escribe por pantalla los valores de los Punto2D del contorno de la forma "(x, y)".
- Observación: se debe utilizar el "extractor" (operador <<) de la clase Punto2D.

▪ **leerContornoDeFichero**
- Recibe como parámetro a nombreFichero de tipo string
- Lee desde el fichero indicado las coordenadas de los puntos
- El fichero debe tener el siguiente formato
  - La primera línea del fichero debe contener el nombre del contorno
  - La segunda línea debe contener el número de puntos del contorno
  - A partir de la tercera línea los puntos deben aparecer en cada línea de la forma x y

▪ **grabarContornoEnFichero:**
- Recibe como parámetro a nombreFichero de tipo string
- Escribe en el fichero indicado las coordenadas de los puntos
  - La primera línea del fichero debe contener el nombre del contorno
  - La segunda línea debe contener el número de puntos del contorno
  - Los puntos deben aparecer en cada línea de la forma x y

◦ Funciones auxiliares
▪ **calcularCentroide**
- Calcula el centroide del Contorno
  - abscisa x del centroide = media aritmética de las abscisas de los puntos del Contorno
  - ordenada x del centroide = media aritmética de las ordenadas de los puntos del Contorno
- Devuelve un Punto2D con las coordenadas del centroide.

▪ **nuevoPrimerPunto**
- Recibe como parámetro a "indice" de tipo entero y "rota" el Contorno

de forma que el primer punto sea el señalado por “indice”.

- **Observación**:
  - Las únicas funciones que podrán acceder directamente a los datos privados de Contorno serán las funciones de acceso y de modificación.
  - Las demás funciones podrán acceder a los datos privados a través de las funciones de acceso y modificación
  - Las versiones del constructor y las funciones de consulta y modificación se codificarán “**inline**”.

IMPLEMENTACIÓN CON LISTA

- **Objetivos**
  - Definición de la clase **ListaPunto2D.** Un TAD lista con cursor, en este caso de objetos de tipo Punto2D.
  - Igual que la implementación con vector, pero ahora usando la lista anterior para los puntos del Contorno.

- **Atributos privados de ContornoGrafico2D**
  - nombreContorno: atributo de tipo string
  - numeroPuntos: atributo de tipo int
  - listaPunto2D: puntero a ListaPunto2D

- **Funciones o métodos públicos**
  - Constructor
    - No recibe parámetros
    - Crea un objeto del tipo Contorno con los valores
      - nombreContorno: “” (cadena vacía)
      - numeroPuntos : 0
      - listaPunto2D vacía.
  - Constructor de copia
    - Recibe como parámetro un Contorno “c” pasado por referencia constante y crea una copia.
  - Destructor
    - Libera, si es necesario, la memoria ocupada por el Contorno

  - Métodos de **acceso** para los nuevos atributos propios
    - **getListaPuntos**: devuelve el puntero listaPunto2D
    - **getNombreContorno**: devuelve el atributo nombreContorno
    - **getNumeroPuntos:** devuelve el atributo numeroPuntos
    - **Aviso**: estos métodos son **const**
    - **getPunto**:
      - Recibe como parámetro “indice” de tipo entero
      - Devuelve **referencia** al Punto2D que ocupa lugar señalado por “indice”
    - **Sobrecarga del operador []**:

- Recibe como parámetro “indice” de tipo entero
- Devuelve **referencia** al Punto2D que ocupa lugar señalado por “indice”
- Prototipo
  Punto2D & operator[](int indice);

◦ Métodos de **modificación** para los atributos propios
- **setListaPuntos**:
  - Recibe a “lista” de tipo puntero a ListaPunto2D.
  - Libera, si es necesario, la memoria que actualmente ocupa la lista de puntos.
  - Si es necesario, reserva la nueva memoria necesaria para listaPunto2D y le asigna uno a uno los puntos recibidos en “lista”.
- **setNombreContorno**:
  - Recibe a “v” de tipo string y se lo asigna al nombre del Contorno
  - El valor por defecto es la cadena vacía
- **setNumeroPuntos**:
  - Recibe a “v” de tipo entero y se lo asigna a numeroPuntos del Contorno
  - El valor por defecto es 0

- **Sobrecarga del operador de asignación “=”**
  - Asigna al Contorno actual valores del Contorno pasada como parámetro
  - Parámetro: un Contorno pasado como referencia constante
  - Resultado: referencia al Contorno actual
  - Prototipo
    inline Contorno & operator=(const Contorno &c);

◦ Funciones de lectura o escritura
- **escribirContorno:**
  - Escribe por pantalla los valores de los Punto2D del contorno de la forma “(x, y)”.
  - Observación: se debe utilizar el “extractor” de la clase Punto2D.
- **leerContornoDeFichero**
  - Recibe como parámetro a nombreFichero de tipo string
  - Lee desde el fichero indicado por nombreFichero las coordenadas de los puntos de un Contorno.
  - El fichero debe tener el siguiente formato
    - La primera línea del fichero debe contener el nombre del contorno
    - La segunda línea debe contener el número de puntos del contorno
    - Los puntos deben aparecer en cada línea de la forma x y
- **grabarContornoEnFichero:**
  - Recibe como parámetro a nombreFichero de tipo string
  - Escribe en el fichero indicado por nombreFichero las coordenadas de los puntos de un Contorno
    - La primera línea del fichero debe contener el nombre del contorno
    - La segunda línea debe contener el número de puntos del contorno
    - Los puntos deben aparecer en cada línea de la forma x y

- Funciones auxiliares
  - **calcularCentroide**
    - Calcula el centroide del Contorno
      - $x_{centroide} = \frac{\sum_{i=1}^{n} x_i}{n}$
      - $y_{centroide} = \frac{\sum_{i=1}^{n} y_i}{n}$

        donde "n" es el número de puntos del contorno
    - Devuelve un Punto2D con las coordenadas del centroide.
  - **nuevoPrimerPunto**
    - Recibe como parámetro a "indice" de tipo entero y "rota" el Contorno de forma que el primer punto sea el señalado por "indice".

- Observación:
  - Las únicas funciones que podrán acceder directamente a los datos privados de Contorno serán las funciones de acceso y de modificación.
  - Las demás funciones podrán acceder a los datos privados a través de las funciones de acceso y modificación
  - Las versiones del constructor y las funciones de consulta y modificación se codificarán "**inline**".

Definición de la clase ListaPunto2D

- **Atributos privados de ListaPunto2D**
  - _n: numero entero (no negativo) que representa el número de puntos de la lista
  - _cabeza: atributo del tipo puntero a NodoLista
  - _cola: atributo del tipo puntero a NodoLista
  - _cursor:
  - Observación: el tipo NodoLista es una estructura compuesta por
    - punto del tipo Punto2D
    - siguiente del tipo puntero a NodoLista

- **Funciones o métodos públicos**
  - Constructor: **ListaPunto2D**
    - Versión única: **constructor sin argumentos**
      - No recibe parámetros
      - Crea un objeto del tipo ListaPunto2D con los valores
        - _n = 0;
        - _cabeza = NULL;
        - _cola = NULL;
        - _cursor = NULL;

- **Destructor** de la clase ListaPunto2D
  - Libera la memoria ocupada por la ListaPunto2D actual

- Métodos de **acceso** para los nuevos atributos propios
  - **vacia**:
    - Función de tipo bool que comprueba si la ListaPunto2D está vacía (true) o no (false).
  - **longitud**
    - Devuelve el valor del atributo "_n" de la ListaPunto2D.
    - Función inline
  - **get**
    - Devuelve una referencia al punto actual apuntado por el atributo _cursor
    - Función inline

- Métodos de **modificación** para los atributos propios
  - **liberarMemoria**:
    - Libera la memoria ocupada por la ListaPunto2D actual
    - La lista toma el valor NULL
  - **set**
    - Recibe a "p" de tipo Punto2D (pasado como referencia constante) y se lo asigna al punto actual apuntado por _cursor
    - Función de tipo bool: devuelve true, si la operación se ha hecho con éxito; false, en caso contrario
  - **insertarAlFinal**:
    - Recibe a "p" de tipo Punto2D (pasado como referencia constante) y lo inserta al final de la lista.
    - Los atributos _cola y _cursor pasan a apuntar al nuevo nodo.
    - Función de tipo bool: devuelve true, si la operación se ha hecho con éxito; false, en caso contrario
  - **borrar**
    - Borra el punto actual apuntado por _cursor, si la lista no está vacía
    - El _cursor pasa a apuntar al siguiente punto, pero si el elemento borrador era el último entonces pasa a apuntar al primero.
    - Función de tipo bool: devuelve true, si la operación se ha hecho con éxito; false, en caso contrario
  - **reset**
    - Colocar el _cursor apuntado a la _cabeza
    - Función de tipo bool: devuelve true, si la operación se ha hecho con éxito; false, en caso contrario
  - **siguiente**
    - Colocar el _cursor apuntado al siguiente punto.
    - Si el _cursor estaba al final entonces pasa a apuntar al primero.
    - Función de tipo bool: devuelve true, si la operación se ha hecho con éxito; false, en caso contrario

- Funciones de escritura
  - **verLista:**
    - Escribe por pantalla los valores de los Punto2D de la lista de la forma "(x, y)".
    - Observación: se debe utilizar la función get de esta clase y el "extractor" de la clase Punto2D.

- **Observación**
  - Se podrá comprobar que la clase **ListaPunto2D** ha sido codificada correctamente ejecutando el programa de ejemplo denominado "**principalLista.cpp**"