Parcial Algoritmos y Programación II - 7541 Curso Calvo (3º instancia) 30 de junio de 2015

Padrón: **Apellido y Nombre:**

Punteros: APROBADO - DESAPROBADO

1) Indicar la salida por pantalla y escribir las sentencias necesarias para liberar correctamente la memoria.

```
int main(){
  int *A, *C, *F;
  int **B;
  char *D, *E;
  char G;
  int H;

  H = 65;
  G = 'D';
  A = &H;
  D = &G;
  B = &A;
  cout << (**B) <<  (*D) << (*A) << endl;

  C = new int[4];
  for (int i = 0; i < 4; i++) {
          C[i] = H + i;
  }

  F = C + 2;
  (*A) = 69;
  C[1] = H;
  (*F) = 67;
  (*C) = H;
  cout << C[0] << C[1] << C[2] << C[3] << endl;

  (**B) = C[0] + 2;
  A = C + 1;
  (**B) = H;
  C[0] = *A;
  (*F) = A[0] + 1;
  cout << C[2] << C[1] << C[0] << endl;

  E = (char*) A;
  C[1] = (**B) - 6;
  cout << (*D) << (*E) << G << endl;

  // liberar la memoria
  return 0;
}
```

2. Implementar para la clase Lista con una estructura **doblemente enlazada** el siguiente método, indicando pre y post condiciones:

```
void agregar(T elemento, unsigned int posicionElemento);
```

3. Implementar el método `'agregarDestinatarios'` de la clase `'Mensajero'` a partir de las siguientes especificaciones:

```
class Mensajero {
  public:
    /* post: procesa 'mensajesPendientes' y a cada Mensaje le agrega como destinatario cada una de las Cuentas de la lista
     *       'destinatariosAdicionales' siempre y cuando el remitente del Mensaje no está en la lista de remitentes
     *       bloqueados de la Cuenta a ser agregada.
     */
    void agregarDestinatarios(Lista<Mensaje*>* mensajesPendientes, Lista<Cuenta*>* destinatoriosAdicionales);
};
```

```
class Mensaje {

  public:

    /* post: Mensaje con el contenido indicado y sin Destinatarios.
     */
    Mensaje(Cuenta* remitente, string contenido);

    /* post: elimina todos los Destinatarios asociadas.
     */
    ~Mensaje();

    /* post: devuelve el contenido del Mensaje.
     */
    string obtenerContenido();

    /* post: devuelve la Cuenta que envía el Mensaje.
     */
    Cuenta* obtenerRemitente();

    /* post: devuelve todas las Cuentas a las que debe enviar el Mensaje.
     */
    Lista<Cuenta*>* obtenerDestinatarios();
};
```

```
class Cuenta {

  public:

    /* post: Cuenta con el nombre indicado.
     */
    Cuenta(string nombre);

    /* post: identificador de la cuenta.
     */
    string obtenerNombre();

    /* post: devuelve aquellas Cuentas de las
     *       que no se desean recibir Mensajes.
     */
    Lista<Cuenta*>* obtenerRemitentesBloqueados();

};
```

4. Diseñar la especificación e implementar el TDA **Puntuación**. Una Puntuación permite contabilizar los puntos con los que diferentes usuarios valoran un elemento. Debe proveer operaciones para:

- Crear la Puntuación indicando el rango de puntajes admitidos, es decir el mínimo y máximo admitidos para puntuar. Por ejemplo: 0..100, 1..5, 100..500, etc.
- Devolver los puntaje mínimo admitido.
- Devolver los puntaje máximo admitido.
- Puntuar (recibiendo el puntaje y nombre del usuario): contabiliza el puntaje y registra el usuario que lo hizo.
- Devolver el promedio de puntos obtenidos.
- Devolver la cantidad de usuarios que puntuaron.
- Frecuencia de un puntaje: devuelve la cantidad de veces que se recibió el puntaje indicado.

Nota: Una Puntuación no debe permitirle a un mismo usuario puntuar más de una vez.

Los alumnos que tienen aprobado el parcialito de punteros no deben realizar el ejercicio 1.
Para aprobar es necesario tener al menos el 60% de cada uno de los ejercicios correctos y completos.
Para cada método escribir pre y post condición, si recibe argumentos y cuáles, y si retorna un dato y cuál. De faltar ésto, se considerará el código incompleto.
Duración del examen : 3 horas