Parcial Algoritmos y Programación II - 7541 Curso Calvo (2º instancia) 15 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 = 66;
  G = 'C';
  A = &H;
  D = &G;
  B = &A;
  cout << *A <<  *D << **B << endl;
  C = new int[3];
  F = C + 2;
  C[1] = H;
  (*A) = 70;
  (*C) = H;
  (*F) = 68;
  cout << C[0] << C[1] << C[2] << endl;
  (**B) = C[0] + 1;
  A = C + 1;
  (**B) = H;
  C[0] = *A;
  (*F) = A[0] + 1;
  cout << C[0] << C[1] << C[2] << endl;
  E = (char*) A;
  C[1] = (**B) - 5;
  cout << (*D) << (*E) << G << endl;
  // liberar la memoria
  return 0;
}
```

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

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

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

```
class Mensajero {

  public:

    /* post: procesa 'mensajesPendientes' y busca aquellos Mensajes cuya Cuenta remitente tiene entre sus
     *       remitentes bloqueados a uno de sus destinatarios, por lo tanto, esos Mensajes no van a poder ser respondidos.
     *       Devuelve una nueva lista con los Mensajes encontrados.
     */
    Lista<Mensaje*>* buscarMensajesQueNoPuedenSerRespondidos(Lista<Mensaje*>* mensajesPendientes);
};
```

```
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 **Catapulta**. Una Catapulta posee un contrapeso que le permite lanzar múltiples proyectiles. Debe proveer operaciones para:

- Crear la catapulta a partir del peso (en kilogramos) de su contrapeso.
- Indicar el peso (en kilogramos) disponible para lanzar un proyectil.
- Cargar: agrega un proyectil, indicando su peso (en kilogramos).
- Descargar: remueve todos los proyectiles antes cargados.
- Disparar: lanza todos los proyectiles cargados, quedando descargada y devolviendo la distancia a la que llegarán dichos proyectiles
  Devolver la cantidad de proyectiles lanzados.
- Devolver la distancia máxima a la que lanzó un proyectil.

Nota: Una Catapulta puede lanzar un grupo de proyectiles cuyo peso acumulado tiene que ser menor a 10 veces el peso de su contrapeso. La distancia (en metros) del disparo se calcula como: (10 * PesoDelContrapeso – PesoDeLosProyectilesCargados)^2 * K.

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