Parcial Algoritmos y Programación II - 7541 Curso Calvo (1º instancia) 26 de mayo 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;

  G = 'B';
  D = & G;
  E = D;
  (*E) = 'C';
  cout << (*D) << G << (*E) << endl;

  H = 64;
  A = new int[2];
  (*A) = H;

  B = & A;
  C = A + 1;
  F = A;
  F[1] = 65;
  cout << (**B) << (*F) << (*C) << endl;

  E = (char*) (F + 1);
  G = 'D';
  (*C) = 66;
  cout << (*D) << (*E)  << F[1] << endl;

  D = (char*) (*B);
  A[1] = 67;
  G = 'X';
  cout << (*E) << (*D) << (*C) << endl;

  // liberar la memoria

  return 0;
}
```

2. Implementar para la clase ListaSimplementeEnlazada el método

```
/* pre : la lista tiene al menos tantos elementos como posicionElemento.
 * post: remueve el elemento localizado en la posición posicionElemento de la lista y lo devuelve.
 */
T remover(unsigned int posicionElemento);
```

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

```
class Mensajero {

  public:

    /* post: procesa 'mensajesPendientes' para contabilizar aquellos Mensajes que tienen como uno de sus destinatarios
     *       a una Cuenta en cuya lista de remitentesBloqueados está el remitente del Mensaje.
     */
    unsigned int contarMensajesBloqueados(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 **Ascensor**. Un Ascensor se debe crear recibiendo como parámetro la cantidad de pisos por los que se mueve (sin considerar la planta baja). Debe proveer operaciones para:

- devolver el número de piso en el que se encuentra, considerando 0 como la planta baja.
- llamar desde un piso: debe moverlo y devolver la cantidad de pisos que el ascensor se movió para llegar al piso indicado.
- devolver la cantidad total de pisos que el ascensor subió y bajó.
- devolver la cantidad de veces que fue a un piso.

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