Actividad 3.1

Lista contigua

Algoritmos y Estructuras de Datos

Tema 3: listas

1º Grado en Ingeniería en Desarrollo de Contenidos Digitales
© Profesor Dr. Carlos Grima Izquierdo (www.carlosgrima.com)
U-tad (www.u-tad.com), curso 2014/15

Programar la clase "Lista" que será una lista contigua de objetos de la clase "Elemento". Además será redimensionable automáticamente según se van añadiendo o quitando elementos.

Como hemos mencionado, los elementos de la lista serán objetos de la clase "Elemento", que hay que programar:

- Esta vez, la clase Elemento simplemente tendrá un float (privado). Pero eso la lista no lo sabrá. Sólo lo sabrá el main(). De este modo, en el futuro podremos usar la lista para contener cualquier tipo de dato (int, Persona, float, etc.) sólo tocando la clase Elemento y no la clase Lista (máxima cohesión, mínimo acoplamiento).
- Tendrá un constructor, al que se pasará el float por parámetro.
- Tendrá métodos set() y get() para modificar y ver el float
- Tendrá sobrecargados los operadores "==", "<", ">", "<=" y ">=" para comparar dos objetos de tipo Elemento. Todos ellos devolverán true o false.
- Tendrá sobrecargado el operador "<<" para poder imprimir por pantalla un elemento con cout. Recuerda no obstante que esto no es adecuado porque viola el principio de la separación entre modelo e interfaz.
- Tendrá la sobrecarga del operador de asignación "=".
- Tendrá también un constructor de copia.

La clase Lista tendrá las siguientes operaciones públicas:

- Un constructor para crear una lista vacía y con capacidad 0.
- Constructor de copia.
- Un destructor para liberar su memoria.
- Un método para devolver el elemento de una determinada posición (empezando en 0).
- Un método para modificar el elemento de una determinada posición
- Un método para insertar un elemento antes de la posición indicada como argumento. Por lo tanto, si la posición es 0, se inserta al principio de la lista. Si la posición es n, se inserta al final.
    - Si, antes de insertar, se ve que la capacidad de la lista ya está llena, hay que ampliar dicha capacidad en "INCREMENTO" posiciones más (INCREMENTO es una constante #define que se pondrá en el .h de la clase). Pondremos que

INCREMENTO sea 2 (aunque esto no sea muy realista), con el fin de probar mejor el programa.

- Un método para eliminar un elemento de la posición indicada como argumento. Una vez eliminado, la lista se adapta para que no haya huecos vacíos en ella.
    - Si, después de eliminar, se ve que en la lista sobran 2*INCREMENTO posiciones, la capacidad de la lista se reduce en INCREMENTO.
- Un método para devolvernos el tamaño actual de la lista.
- Un método para devolvernos la capacidad actual de la lista.
- Sobrecarga de "=" para asignar dos listas.
- Sobrecarga de "+" para concatenar dos listas. Devolverá el resultado de concatenar la lista actual con la que se pasa como argumento.
- Sobrecarga de "<<" para imprimir por pantalla una lista. Imprimirá el tamaño, la capacidad y la propia lista de los elementos, separados por comas entre ellos. Atención: recuerda no obstante que este método viola el principio de separación entre interfaz y modelo. Para imprimir cada elemento de la lista utilizaremos "<<", el cual hemos sobrecargado en Elemento.
- Sobrecarga de "[]" para acceder a un elemento de la lista (empezando por 0), como si estuviéramos en un vector normal de C++.
- Método "buscar" que, ante un Elemento pasado como parámetro, nos devuelve su posición en la lista (empezando por 0), o bien -1 si no lo ha encontrado. Este método utilizará "==" de Elemento para comparar un Elemento con otro.
- Método para ordenar la lista de mayor a menor. Utiliza los operadores de comparación de Elemento. Usa el método de ordenación que quieras. Si utilizas qsort(), el método "compare" usará los operadores de comparación de "Elemento".

Algunas consideraciones:

- Recuerda que, a partir de ahora, en los comentarios de cada método o función siempre deberás poner la complejidad temporal y espacial en el peor caso.
- Utiliza realloc() para reasignar memoria (ampliarla o disminuirla). Para ampliarla, si a continuación no hay memoria libre entonces busca un hueco suficiente de memoria libre, lo reserva, copia lo antiguo a lo nuevo, y libera lo viejo. Para disminuirla, simplemente notifica al sistema operativo la liberación de las posiciones que sobran. Todo esto es relevante para calcular la complejidad espacial de los métodos que usen realloc().
- Utiliza memcpy o memmove (elige cuál en cada momento) para insertar o eliminar.
- Se deberán añadir los atributos y métodos privados que se consideren oportunos. Se valorará una buena división de las tareas siguiendo el principio de la máxima cohesión y el mínimo acoplamiento.
- Ojo con cómo se pasan los parámetros y se devuelven los resultados en la sobrecarga de operadores y en los constructores de copia. En la mayoría de los casos hay que hacerlo por referencia, no por valor.

Para probar la lista, el main deberá hacer lo siguiente:

1. Crear una lista vacía. Imprimirla.
2. Rellenar con los números naturales de 0 a 11, en orden. Imprimir la lista cada vez que se inserta uno.
3. Imprimir el elemento 0 de la lista.
4. Imprimir el elemento 11 de la lista.

5. Cambiar el elemento 4 por 50. Imprimir la lista.
6. Insertar 100 en la posición 0. Imprimir la lista.
7. Preguntar a la lista cuántos elementos tiene, e insertar 200 en la última posición. Imprimir la lista.
8. Insertar 300 en la posición 4. Imprimir la lista.
9. Borrar el elemento 4. Imprimir la lista.
10. Borrar el elemento 4. Imprimir la lista.
11. Preguntar a la lista cuántos elementos tiene y borrar el último. Imprimir la lista.
12. Borrar el elemento 4. Imprimir la lista.
13. Borrar el elemento 0. Imprimir la lista.
14. Crear dos listas vacías (“a” y “b”)
15. Rellenar la primera lista con los números naturales de 0 a 5, en orden, y la segunda con 6 a 10. Imprimir ambas listas.
16. Imprimir los elementos 0, 3 y 5 de “a”. Utilizar para ello el operador “[]”.
17. Buscar el 5 en “a”. Imprimir la posición en donde se ha encontrado.
18. Buscar el 10 en “a”. Imprimir la posición en donde se ha encontrado.
19. Crear una tercera lista (“c”), inicializándola a “b”. Imprimir “c”.
20. Crear una cuarta lista (“d”). A continuación, asignarle la concatenación de “a” y “b”, todo en la misma instrucción (d = a+b). Imprimir d.
21. Ordenar “d”. Imprimir “d”.