Tema 3 resumen/mapa conceptual. Arrays Dinámicos.

Alejandro Ruiz López
PROGRAMACIÓN CON ESTRUCTURAS LINEALES

Clase Dynarray. Semántica de copia.

Inicialización vs asignación

La diferencia entre inicialización y asignación es que la primera crea un objeto a partir de otro ya existente, mientras que la segunda asigna un objeto a otro ya existente, eliminando su valor original.

Ejemplo, definimos plantilla: `Complex<double> x {1.0,2.0};`

Ejemplo, inicialización `y{x};`

Ejemplo, asignación `y=x;`

- El compilador define implícitamente un constructor copia y un operador de asignación copia para permitir estas operaciones
- La sentencia "Complex y = x;" implica una inicialización, no una asignación. Y es una copia de x.
- Se elimina su valor original.

Ejemplo, un constructor copia, que permite inicializar un objeto copiando la representación de otro dado:

```
template<typename T> inline
Complex<T>::Complex(Complex<T> const&);
```

Un operador de asignación copia, que permite copiar la representación de un objeto dado en otro ya existente. El valor retornado es una referencia al propio objeto con el fin de permitir la concatenación de asignaciones del tipo x = y = z.

```
template<typename T> inline
Complex<T>& Complex<T>::operator=(Complex<T> const&);
```

Estructura de la clase Dynarray<>

La clase Dynarray<> es una plantilla que encapsula matrices dinámicas unidimensionales con elementos de tipo genérico T.

Esta clase sigue la técnica RAII, por lo que cuenta con un constructor que se encarga de asignar la memoria dinámica y un destructor que la libera al concluir la duración de almacenamiento del objeto

Además, en su interfaz pública se incluyen múltiples declaraciones “using” que proporcionan un listado coherente de tipos compatibles con la biblioteca estándar del lenguaje (alias que facilitan la referencia a los tipos de datos).

Implementación de la clase Dynarray<>

Constructores y destructor

La clase Dynarray<> es una plantilla que permite crear matrices dinámicas de cualquier tipo de dato.
El constructor de la clase Dynarray<> inicializa una matriz de tamaño especificado y asegura que no se emitan excepciones durante su creación.
Para su implementación, se utilizan dos constructores:

- Uno por defecto
- Otro que permite especificar el tamaño y el valor de los elementos de la matriz.

Además, se define un destructor encargado de liberar la memoria asignada por el constructor. El destructor de la clase Dynarray<> libra la memoria asignada para los elementos de la matriz, evitando fugas de memoria.

Semántica de copia

En este caso, el constructor copia y el operador de asignación copia generados automáticamente por el compilador no son adecuados, ya que simplemente copian la representación del objeto original, lo que puede llevar a problemas de memoria.
Por lo tanto, es responsabilidad del programador definir correctamente estas operaciones.

Problemas con la copia automática: La copia automática generada por el compilador resulta inadecuada, ya que realiza una copia byte a byte, lo que puede llevar a referencias compartidas a la misma memoria.

Constructor de copia: El constructor de copia de Dynarray<> asigna memoria dinámica suficiente y copia cada elemento de la matriz original a la nueva matriz.

Operador de asignación de copia: El operador de asignación de copia utiliza la técnica COPY-AND-SWAP para garantizar que la representación original del objeto se mantengan en caso del excepciones durante la copia.

Resto de la interfaz pública

Manejo de excepciones:

- Garantía fuerte ante excepciones: La implementación de la clase Dynarray<> proporciona una garantía ante excepciones, asegurando que la memoria se maneje correctamente incluso si ocurre un error durante la copia.
- Uso de bloques TRY-CATCH: Se utilizan bloques “try-catch” en el constructor y el operador de asignación y libera recursos adecuadamente.

Interfaz pública:

- Funciones Miembro: La clase Dynarray<> incluye funciones miembro que permiten iterar a través de la matriz y acceder a sus elementos, tanto con como sin verificación de rango.
- Inclusión de ficheros de cabecera: Se requiere la inclusión de varios ficheros de cabecera estándar para asegurar una correcta compilación y funcionamiento de la clase Dynarray<>.

Funciones begin() y end(). Bucles for basados en rango

Permiten iterar desde el primer al último elemento de la matriz.
rango

- `begin()` retorna un iterador que apunta al primer elemento de la matriz
- `end()` lo retorna hipotético elemento de tipo value_type posterior a la última entrada de la matriz.
- `cbegin()` y `cend()` para objetos no constantes.
- Ayudan a recorrer las sentencias for.

RESUMEN VECTOR

- Representa array de elementos alojado en el free store.
- Tamaño aumenta o disminuye a conveniencia.
- Cada elemento va una entrada continua a otra en memoria al ser añadidos´

Partes:

Parámetros:

`V_ , begin()` puntero señala al inicio del vector
`space_ , end()` puntero que apunta al hipotético elemento posterior al último dato almacenado.
`last_` puntero centinela que apunta al siguiente espacio fuera del vector.

Métodos:

`size()` elementos del vector
`capacity()` capacidad del vector reservada en memoria
`empty()` booleano retorna `false` si `size() = 0`

Método “push_back” `void vector<T>::push_back(T const& val)`

Si hay una entrada libre donde introducir un elemento `space() != last_`
el método introducirá en el siguiente hueco del vector el elemento.
Si no hay hueco `space() == last_` tenemos dos casos:

- 1 Primera inserción:
  - Se reserva espacio para el doble del elemento a introducir, en previsión de otra posible inserción.
- 2 Vector agotado:
  - Se reserva un nuevo espacio en el free store con el doble de tamaño del anterior.
  - Se copia el contenido del array original a este espacio.
  - Si se finaliza con éxito, se destruye el original.
  - Hay una nueva referencia.