CS1103

Programación Orientado a Objetos II

Práctica Calificada #1 (PC1)

Sección 201

2021 - 1

Profesor: Rubén Rivas:

---

5 puntos: clases, punteros y sobrecarga de operadores

1. Desarrollar la clase **generate_vector_t** que permita fabricar vectores de un tamaño determinado con el contenido de una lista de números enteros. La clase contará con un constructor que cuente con el siguiente parámetro:
   - Parámetro **ítems** que permita ingresar una lista de enteros (utilizar el tipo **std::initializer_list**).

   Además, la clase debe contar con un atributo adicional a los que requiera:

   - El atributo **block_size** con valor por defecto 6, que define la cantidad máxima de ítems (números enteros) que se generara utilizando el operadpr ().

   La clase deberá ser implementada utilizando **arreglos dinámicos (punteros).**

   Los métodos que deberán ser implementados serán los siguientes:

   - **void set_block_size(int value),** para actualizar el atributo b**lock_size.**
   - **void next_block (),** ubica la posición de recorrido al siguiente bloque.
   - **void next_block (),** ubica la posición de recorrido al anterior bloque.
   - **void first_block(),** se ubica en el primer bloque.
   - **void last_block(),** se ubica el último bloque.

   Adicionalmente se deberá implementar la sobrecarga del **operador ()** de modo que al ejecutarla genere un vector con la cantidad máxima de **block_size** valores del bloque actual.

Ejemplo:

```
// se ubica en el primer bloque y define paginamiento a 6
generate_vector_t gv1 = {1, 2, 3, 4, 5, 6, 5};
// cambia el tamaño del bloque de 6 a 3
gv1.set_bock_size(3);
// avanza 2 bloques
gv1.next_block();
gv1.next_block(); // si la página es la última no se sigue avanzando
// muestra el resultado
for (const auto& item: gv1())
     cout << item << " ";    // 5
 cout << endl;
gv1.previous_block();
for (const auto& item: gv1())
     cout << item << " ";    // 4 5 6
 cout << endl;
gv1.first_block();
gv1.previous_block(); // si es la primera no sigue retrocediendo
for (const auto& item: gv1())
     cout << item << " ";    // 1 2 3
 cout << endl;
gv1.last_block();
for (const auto& item: gv1())
     cout << item << " ";    // 5
 cout << endl;
generate_vector_t gv2 = gv1;// gv2 se ubica en la página 1
for (const auto& item: gv2())
     cout << item << " ";    // 1 2 3
 cout << endl;
for (const auto& item: gv1())
     cout << item << " ";    // 5
 cout << endl;
```

5 puntos: template de funciones

2. Escribir la función de template **generate_fibonacci_shape** que reciba 2 parámetros:
   - **cnt**, un contenedor genérico y
   - **n**, que es la cantidad de números fibonacci a partir de 1 que deben generar.

La función debe retornar un vector de contenedores que serán del mismo tipo del primer parámetro, los contenedores deben contener los valores ingresados en el primer parámetro de modo que cada contenedor almacenado en el vector incluya un número creciente de esos valores agrupados en contenedores de tamaño 1 hasta alcanzar el tamaño del **n**-enésimo número fibonacci.

En caso la cantidad de valores del contenedor ingresado en el parámetro sea mayor al necesario para llegar al tamaño del n-enésimo contenedor, esos números serán descartados.

En caso la cantidad de valores del contenedor ingresado en el parámetro sea menor al tamaño del n-enésimo contenedor, el resto será llenado con ceros.

Ejemplo #1:

```
list<int> lst1 = {1, 2, 3, 4, 5};
auto f1 = generate_fibonacci_shape(lst1, 4);
for (auto row: f1) {
   for(auto value: row)
         cout << value << " ";
   cout << endl;
}
/*
 Se imprimirá:
 1             // Contenedor list<int> de tamaño 1
 2             // Contenedor list<int> de tamaño 1
 3 4           // Contenedor list<int> de tamaño 2
 5 0 0         // Contenedor list<int> de tamaño 3
*/
```

Ejemplo #2:

```
deque<double> deq1 = {1, 2, 3, 4, 5.5, 6, 7, 8, 9, 10, 11, 12, 13};
auto f1 = generate_fibonacci_shape(deq1, 4);
for (auto row: f1) {
   for(auto value: row)
         cout << value << " ";
   cout << endl;
}
/*
 Se imprimirá:
 1             // Contenedor deque <int> de tamaño 1
 2             // Contenedor deque <int> de tamaño 1
 3 4           // Contenedor deque <int> de tamaño 2
 5.5 6 7       // Contenedor deque <int> de tamaño 3
 8 9 10 11 12  // Contenedor deque <int> de tamaño 5
*/
```

5 puntos: template de funciones y librería estándar

3. Crear un template de funciones denominado **move_to_end** que reciba como parámetros:
   - **cnt**, un contenedor genérico y
   - **value**, un valor del mismo tipo de los ítems almacenados en el contenedor,

El template de función deberá mover un contenedor del tipo seleccionado con todos los valores del contenedor **cnt** que coinciden con el valor **value** ubicados al final del contenedor.

Ejemplo #1:

```
deque<int> deq1 = {1, 5, 3, 4, 5, 8, 5, 10, 11};
auto result = move_to_end(deq1, 5);
for (const auto& item: result)
   cout << item << " ";
cout << '\n';
// El resultado es: 1 3 4 8 10 11 5 5 5
```

Ejemplo #2:

```
list<char> lst1 = { 'a', 'b', 'c', 'c', 'd', 'e', 'r' };
for (const auto& item: move_to_end(lst1, 'c'))
   cout << item << " ";
cout << '\n';
// El resultado es: a b d e r c c
```

5 puntos: complejidad algorítmica

4. Basado en la respuesta anterior, determine y sustente exactamente el **BigO** de **tiempo y espacio** del algoritmo que usted ha desarrollado.

Barranco, 1 de junio del 2021