CS1103

Programación Orientado a Objetos II

Práctica Calificada #1 (PC1)

Sección 101

2021 - 1

Profesor: Rubén Rivas:

---

5 puntos: clases, punteros y sobrecarga de operadores

1. Desarrollar la clase **pagination_t** que simule el paginado del contenido de números enteros. La clase contará con un constructor que cuente con los siguientes parámetros:
   - Parámetro **ítems** que permita ingresar una lista de enteros (utilizar el tipo **std::initializer_list**).
   - El atributo **page_size** con valor por defecto 10, que defina la cantidad de ítems (números enteros) que se mostrara al usar el **operador <<**.

   La clase deberá ser implementada utilizando arreglos dinámicos (punteros). Los métodos que deberán ser implementados serán los siguientes:

   - void set_page_size(int value);
   - void next_page()
   - void previous_page()
   - void first_page()
   - void last_page()
   - void goto_page(int page)

   Adicionalmente se deberá implementar la sobrecarga del **operador <<** de modo que al ejecutarla de solo mostrar la información de la página actual el número de ítems.

   Ejemplo:

```
// se ubica en la primera página y define paginamiento a 10
pagination_t pg1 = {1, 2, 3, 4, 5, 6, 5};
// cambia paginamiento de 10 a 3
pg1.set_page_size(3);
// avanza 2 paginas
pg1.next_page();
pg1.next_page(); // si la página es la última no se sigue avanzando
// muestra el resultado
cout << pg1 << endl;        // 5
pg1.previous_page();
pg1.previous_page(); // si es la primera no sigue retrocediendo
cout << pg1 << endl;        // 4 5 6
pg1.first_page();
cout << pg1 << endl;        // 1 2 3
pg1.last_page();
cout << pg1 << endl;        // 5
pagination_t pg2 = pg1;     // pg2 se ubica en la página 1
cout << pg2 << endl;        // 1 2 3
```

5 puntos: template de funciones

2. Escribir la función de template **generate_triangle** que reciba 2 parámetros:
   - **cnt**, un contenedor genérico y
   - **triangle_base**, que es un número entero que represente el tamaño de la base de un triángulo.

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 contenedores de tamaño **triangle_base.**

En caso la cantidad de valores del contenedor ingresado en el parámetro sea mayor al necesario para llegar hasta formar la base, esos números serán descartados.

En caso la cantidad de valores del contenedor ingresado en el parámetro sea menor al necesario para llegar hasta formar la base, el resto será llenado con ceros.

Ejemplo #1:

```
vector<int> vec1 = {1, 2, 3, 4, 5, 6, 7, 8};
auto t1 = generate_triangle(vec1, 4);
for (auto row: t1) {
   for(auto value: row)
         cout << value << " ";
   cout << endl;
}
/*
 Se imprimirá:
 1
 2 3
 5 6 7
 8 0 0 0
*/
```

Ejemplo #2:

```
deque<double> deq1 = {1, 2, 3, 4, 5.5, 6, 7, 8, 9, 10, 11, 12};
auto t1 = generate_triangle(deq1, 4);
for (auto row: t1) {
   for(auto value: row)
         cout << value << " ";
   cout << endl;
}
/*
 Se imprimirá:
 1
 2 3
 5.5 6 7
 8 9 10 11
*/
```

5 puntos: template de funciones y librería estándar

3. Crear un template de funciones denominado **contain_same_values** que reciba 2 contenedores de igual o diferente tipo del mismo tamaño, con valores no necesariamente ordenados de la misma forma y verificar que ambos contienen los mismos valores.

Ejemplo #1:

```
deque<int> deq1 = {1, 2, 3, 4};
vector<int> vec1 = {1, 4, 3, 2};
auto result = contain_same_values(deq1, vec1);
if (result == true)
   cout << "contienen lo mismo\n"; // Este es el resultado correcto
else
   cout << "NO contienen lo mismo\n";
```

Ejemplo #2:

```
list<char> lst1 = { 'a ', ' b', 'c', 'c', 'd'};
deque<char> deq2 = { 'a ', ' d', 'c', 'b ', 'b'};
if (contain_same_values(lst1, deq2) == true)
   cout << "contienen lo mismo\n";
else
   cout << "NO contienen lo mismo\n"; // Este es el resultado correcto
```

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