CS2013 - Programación III

Práctica Calificada #2 (PC2)

2023 - 1

Profesor: Rubén Rivas

Librería Estándar – 6 puntos

Diseñar y desarrollar el template de función **split_range** que permita dividir un contenedor en base a una lista de inicializacion (***std::initializer_list***). La lista de contenedores generados deberan de ser del mismo tipo del contenedor de original y deberan ser devueltos en un vector.

Si se tiene un container con los siguientes valores:
cnt = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

Y si se llama:
result = **split_range**(*cnt*, {4, 6, 2})

result contendria:
```
{
   {1, 2, 3, 4},
   {5, 6, 7, 8, 9, 10},
   {11, 12}
}
```

Casos de uso #1

```
int n = 0; cin >> n;
vector<int> vec(n);
for(auto& item: vec) cin >> item;
auto res = split_range(vec, {n/2, n-n/2});

// Verifica que contenedor haya sido generado en el tipo correcto
static_assert(is_same_v<decltype(res), vector<vector<int>>>,
              "Error in result type");

// Muestra resultado
for(const auto& row: res) {
    copy(begin(row), end(row), ostream_iterator<int>(cout, " "));
    cout << endl;
}
```

Casos de uso #2

```
int n = 0; cin >> n;
forward_list<int> lst(n);
for(auto& item: lst) cin >> item;
auto res = split_range(lst, {n/2, n-n/2});

// Verifica que contenedor haya sido generado en el tipo correcto
static_assert(is_same_v<decltype(res), vector<forward_list<int>>>,
              "Error in result type");

// Muestra resultado
for(const auto& row: res) {
    copy(begin(row), end(row), ostream_iterator<int>(cout, " "));
    cout << endl;
}
```

Complejidad Algorítmica – 6 puntos

Dada la siguiente función determinar la complejidad algoritmica, incluir en su respuesta el procedimiento.

```
bool nested_search(std::vector<int>& vec, int value) {

  std::sort(begin(vec), end(vec));
  int left = 0, right = size(vec) - 1;

  while (left <= right) {
    int mid = left + (right - left)/2;

    // Búsqueda binaria anidada desde left hasta mid
    auto it = begin(vec);
    auto found = std::binary_search(next(it, left), next(it, mid), value);
    if(found) return true;

    // Comprueba si el value está en la derecha o en la izquierda
    if(vec[mid] < value)
      left = mid + 1;
    else
      right = mid - 1;
  }
  return false;
}
```

Patrones – 6 puntos

Dada la siguiente descripción, identifique que patrones podrías ser usados para la implementación y justifique seleccionando el párrafo donde se podría utilizar:

Se desea desarrollar un sistema de seguimiento de flota de vehículos que tenga las siguientes características:

- *Nuestro sistema necesita crear objetos que representen diferentes tipos de vehículos (camiones, coches, motos). Dependiendo de los datos de entrada, el sistema decidirá qué tipo de objeto vehículo crear.*
- *Aseguramos que sólo exista una única instancia de la base de datos de vehículos en todo el sistema para evitar conflictos de datos. Esta única instancia proporciona un punto de acceso global a la base de datos de vehículos.*
- *Los vehículos pueden tener diferentes tipos de sistemas de seguimiento, por ejemplo, GPS o GLONASS. Nuestro diseño separa el concepto de un vehículo de su sistema de seguimiento, permitiendo que el sistema de seguimiento de un vehículo pueda cambiarse en tiempo de ejecución.*
- *Podemos tratar un grupo de vehículos (una flota) de la misma manera que tratamos a un único vehículo. Esto nos permite realizar operaciones sobre la flota entera, como calcular la ubicación media de todos los vehículos en la flota, de la misma manera que encontraríamos la ubicación de un solo vehículo.*
- *El sistema recibe notificaciones cada vez que la ubicación de un vehículo cambia. Cuando un vehículo cambia su ubicación, todos los componentes del sistema que están registrados para recibir estas actualizaciones (como una interfaz de usuario que muestra la ubicación del vehículo en un mapa, o un sistema de alertas que se dispara si un vehículo sale de una zona designada) son notificados y pueden tomar acción.*

Programación Concurrente – 6 puntos

Escribir la función template **parallel_product** que retorne la multiplicación 1x1 de los elementos de ambos contenedores en un nuevo contenedor del tipo vector.

Debe asegurarse que ambos contenedores sean del mismo tamaño, sino generar una excepción con el siguiente mensaje: "**containers must have same size**" (std::runtime_error).

Los contenedores podrian ser de diferente tipo y su recorrido será a traves de sus interadores.

Si se tiene 2 containers con los siguientes valores:
cnt1 = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
cnt2 = {10, 20, 10, 20, 10, 30, 10, 20, 10, 30, 10, 20}

Y si se llama:
*result = **parallel_product**(*
begin(cnt1), end(cnt1), begin(cnt2), end(cnt2))

result contendria:
{ 10, 40, 30, 80, 50, 180, 70, 160, 90, 300, 110, 240}

Caso de uso #1

```
int n = 0; cin >> n;
vector<int> vec1(n);
vector<int> vec2(n);
for(auto& item: vec1) cin >> item;
for(auto& item: vec2) cin >> item;
static_assert(is_same_v<decltype(vec1), vector<int>>,
                          "Error in result type");

auto result = parallel_product(begin(vec1), end(vec1),
                               begin(vec2), end(vec2));

for (const auto& item: result) cout << item << " ";
cout << endl;
```

Caso de uso #2

```
int n = 0; cin >> n;
forward_list<int> lst(n);
vector<int> vec(n);
for(auto& item: lst) cin >> item;
for(auto& item: vec) cin >> item;

auto result = parallel_product(begin(lst), end(lst),
                               begin(vec), end(vec));

for (const auto& item: result) cout << item << " ";
cout << endl;
```

Caso de uso #3

```
list<int> lst {1, 2, 3, 4, 5, 6};
vector<int> vec {10, 20, 30, 40};
try {
    auto result = parallel_product(
        begin(lst), end(lst), begin(vec), end(vec));
}
catch (exception& e) {
    cout << e.what() << endl;  // containers must have same size
}
```