CS2013 - Programación III

Práctica Calificada #2 (PC2)

2023 - 1

Profesor: Rubén Rivas

Librería Estándar – 6 puntos

Diseñar y desarrollar el template de función **same_values** que en un tiempo lineal O(n) permita identifica si 2 contenedores del mismo tamaño contienen los mismos elemento no importando que sus valores se encuentre en diferente orden. Ejemplo:

```
container_1 = { 1, 4, 5, 19, 20, 11, 22, 1 };

container_2 = { 1, 1, 4, 5, 11, 19, 20, 22 };
```

Casos de uso

```
// 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;
cout << same_values(vec1, vec2) << endl;
// Caso de uso #2
deque<int> deq1(n);
list<int> lis1(n);
for(auto& item: deq1)
cin >> item;
for(auto& item: lis1)
   cin >> item;
cout << same_values(deq1, lis1) << endl;
```

Complejidad Algorítmica – 6 puntos

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

```
typedef std::complex<double> Complex;
typedef std::vector<Complex> ComplexVector;


void function (ComplexVector& x) {
    int N = x.size();
    if (N <= 1) {
        return;
    }

    ComplexVector even, odd;
    for (int i = 0; i < N; i += 2) {
        even.push_back(x[i]);
        odd.push_back(x[i + 1]);
    }

    function (even);
    function (odd);

    double theta = 2 * M_PI / N;
    Complex w(1);
    Complex wn(cos(theta), sin(theta));

    for (int k = 0; k < N / 2; ++k) {
        Complex t = w * odd[k];
        Complex u = even[k];
        x[k] = u + t;
        x[k + N / 2] = u - t;
        w *= wn;
    }
}
```

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:

Imaginemos que estamos desarrollando un sistema para crear y gestionar documentos en una empresa. Queremos diseñar una solución flexible que permita crear diferentes tipos de documentos y manipular su contenido de manera eficiente.

Nuestra solución se basa en la idea de tener una clase central que se encargue de la gestión de documentos. Esta clase tiene la capacidad de crear diferentes tipos de documentos según las necesidades del sistema. Para lograr esto, se utiliza un enfoque que nos permite centralizar la creación de documentos en una única ubicación.

Además, queremos separar la forma en que interactuamos con los documentos de su implementación subyacente. Para lograrlo, definimos una interfaz común que especifica las operaciones básicas que se pueden realizar en un documento, como abrir, guardar e imprimir. Cada tipo de documento concreto implementa esta interfaz, lo que nos permite tratar todos los documentos de manera uniforme sin importar su tipo específico.

Además, para permitir la manipulación de documentos más complejos, utilizamos una estructura de composición que nos permite tratar tanto documentos individuales como documentos compuestos de manera uniforme. Esto nos permite realizar operaciones recursivas en la estructura de documentos y facilita la manipulación de jerarquías de documentos.

Finalmente, para garantizar que nuestra solución tenga una única instancia de la clase de gestión de documentos en todo el sistema, utilizamos un enfoque que nos permite tener una única instancia compartida. Esto evita la duplicación de recursos y asegura que todos los componentes del sistema utilicen la misma instancia del gestor de documentos.

Programación Concurrente – 6 puntos

Escribir la función template **find_max_double_product** que debe retornar una lista de pares cuyo producto de los 2 valores sea el valor maximo obtenido, la función deberá retornar un vector de pares con todos los valores que cumplan la condición.

Caso de uso #1

```
vector<int> vec = generate_container<int, 10000>(-100, 100);
auto result = find_max_triple_product(begin(vec), end(vec));
cout << result << endl;
```

Caso de uso #2

```
list<long int> lst = generate_container<long int, 10000>(-100, 100);
auto result = find_max_triple_product(begin(lst), end(lst));
cout << result << endl;
```