SortEngine isimli bir classımız var.

Bu class'ın BubbleSort, QuickSort, SelectionSort isimli 3 tane private fonksiyonu var.

1 adet SORT\_TYPE isimli ENUM (mSortType) ve 1 adet int tipinde vector taşıyan QUEUE (mInputQueue) variable'ı var

1 adet void Process() public fonksiyonu var.

Class'ın constructor'ı parametre olarak 1 adet SORT\_TYPE tipinde ENUM ve 1 adet int tipinde vector taşıyan QUEUE alacak.

Constructor parametre olarak aldığı değerleri kendi member variable'larına atayacak.

BubbleSort, QuickSort, SelectionSort fonksiyonlarında isimlerinde belirtilen sıralama algoritmaları implement edilecek.

Process fonksiyonu içinde bir sonsuz döngü yazılacak. Bu sonsuz döngü içinde mInputQueue'da bekleyen int tipinde vector alınacak ve bu vector mSortType enum'unda belirlenen sıralama algoritması ile sıralanacak.

Queue'da eleman yoksa döngü o turu pas geçecektir. Sonsuz döngü her döndüğünde queue'dan eleman almaya çalışacaktır.

Amaç input queue'ya eklenen vectorlerin alınıp sıralanması. Process fonksiyonun bir return değeri olmayacaktır. Sıralama fonksiyonu işini bitirince sıraladığı vectoru return edecektir. Process fonksiyonu içine return edilen

bu vector kullanılmayacaktır.

main içerisinde 3 adet int tipinde vector taşıyan queue yazılacaktır. ( inputQueue1,inputQueue2,inputQueue3 ) ( Bu queue'lar main.cpp içerisinde global tanımlanabilir )

SortEngine isimli class'dan 3 tane obje yaratılacaktır. Objeler yaratılırken constructor'a parametre olarak sırasıyla (inputQueue1,Sort::BUBBLE) , (inputQueue2,Sort::QUICK), (inputQueue3,Sort::SELECTION) değerleri verilecektir.

Bu 3 objenin Process fonksiyonları thread olarak çalıştırılacaktır. Process fonksiyonları içerisinde sonsuz döngü olacağından 3ünü de çalıştırmanız thread kullanmadan mümkün değildir.

Thread kullanmadığınız durumda 1. objenin Process fonksiyonunu çalıştırdığınız anda diğer taraflara sıra gelmeyecektir.

Bu aşamaya geldiğinizde,

Elimizde her biri kendi input queue'suna sahip 3 tane thread var. Input queue'larına bir vector konulduğu anda thread bu vectoru alıp kendi yöntemi ile sıralayacaktır.

Şimdi bu queue'ları dolduracak GenerateRandomVector isimli fonksiyonu yazacağız.

Bu fonksiyon main.cpp içerisinde main function altına yazılacaktır.

GenerateRandomVector isimli fonksiyon sonsuz döngü içerisinde 1000 adet int tipinde 1-10000 arasında rastgele sayı üretip bu sayıları bir vectore dolduracaktır. Dolduruğu vectoru inputQueue1,inputQueue2 ve inputQueue3

kuyruklarına ekleyecektir. 500 ms bekleyecektir. 500 ms main.cpp içerisinde global değişken olacaktır. (int SleepTime = 500)

Özetlemek gerekirse GenerateRandomVector fonksiyonu her 500 ms'de bir sıralama threadlerinin input queue'larına veri basacaktır. Sıralama threadleri de bu veriyi alıp sıralayacaktır.

Bu aşamaya geldiğinizde 3 adet sıralama threadiniz var. Bunlar input queue'larında bekleyen vectoru alıp sıralamaya çalışıyorlar. GenerateRandomVector fonksiyonu da onlara sürekli sıralamaları için yeni vector veriyor.

Bu yapıda 3 sıralama algoritmanızda farklı hızlarda çalışıyor olacaktır.Çünkü yazdığınız sıralama algoritmaların performansları birbirinden farklı.

Yavaş çalışan threadiniz 500ms'de işini yetiştiremiyor olabilir. Yada 3üde 500 ms'den daha kısa sürede işlerini bitiriyor olabilir.

main.cpp içerisinde Observer isimli bir fonksiyon yazacağız.

Bu fonksiyon da sonsuz döngü içerisinde sürekli inputQueue1,inputQueue2 ve inputQueue3 içerisinde kaç eleman olduğuna bakacak ve 1000ms bekleyecektir.

Yani her 1000ms'de bir queue'ların size'ını kontrol edecektir.

Eğer queue'ların bir yada daha fazlasında size artışı görürse ( örneğin eleman sayısı 50'yi geçerse ) global tanımladığınız ve GenerateRandomVector isimli fonksiyonun kullandığı SleepTime değerini 50 arttıracaktır.

Queue'ların tamamında 5den daha az eleman varsa o zaman da SleepTime değerini 50 azaltacaktır.

Program belirli bir süre çalıştığında artık GenerateRandomVector isimli fonksiyonun kullandığı SleepTime verisi belirli bir değer aralığına evrilecektir.

Bu değer kodu çalıştırdığınız bilgisayarın işlemcisine göre değişiklik gösterecektir.

Yani Observer, threadlerin input queue'larını izleyerek GenerateRandomVector isimli fonksiyonun hızını belirleyecektir.

Daha iyi anlamanız adına gerçek hayattan bir örnekle açıklamak istiyorum

3 tane işçimiz var. ( Bunlar SortEngine class'ından ürettiğimiz nesneler )

İşçilere iş veren 1 tane ustabaşımız var. ( GenerateRandomVector threadimiz )

Ustabaşı belirli periyodlarda ( 500ms'de bir ) işçilere iş veriyor. Verdiği işte sayı sıralama işi.

Ustabaşı sıralanacak sayıların dosyasını ( int tipinde vector ), işçilerin masasına ( input queue ) koyuyor.

Eğer işçi verilen işi 500 ms'den daha hızlı bitirebilirse, ustabaşının yeni iş vermesini bekler.

Eğer işçi verilen işi 500 ms'den daha yavaş yaparsa, o işçinin masasında dosyalar birikmeye başlar.

Bir gözlemci işçilerin masasını kontrol eder.

Eğer bazı işçilerin masasında dosya birikmeye başlarsa, ustabaşına biraz yavaşla, işçiler yetiştiremiyor der ( SleepTime değerini arttırır )

Eğer işçilerin masasını hep boş görürse ( işçiler çok hızlı çalışıyor, ustabaşı yeni iş vermekte yavaş kalıyor ), ustabaşına biraz daha hızlı iş ver der ( SleepTime değerini azalt )

Not : Sıralama algoritmalarını implement ederken SortEngine class’ına istediğiniz fonksiyonları ekleyebilirsiniz. Size gönderdiğim cpp ve hpp dosyaları taslak olarak yol göstermesi için verilmiştir ( eksik yada hata olabilir ). Onun üzerine yazabilir ve/veya istediğiniz değişikliği yapabilirsiniz.

Thread kullanımında özgürsünüz ( boost, std::thread vs. )