Event logger rapport.

Event: En struct som simulerar ett senor event och lagrar relvant data. skapas som som pekare via event producer och.

EventQueue: En struct som innehåller en array som håller eventpekare som enqueueas via event producer, vid misslyckade and en enqueue så förstör producern eventet för att förhidra memoryleaks, event consumer dequeuear sedan eventet och nollställer index efter dequeue för att förhindra att pekaren lever kvar där. sedan tar Observer över.

Observer/Alarm: observer tar emot event pekaren och meddelar alarmSet och EventLog, alarmSet fungerar som ett SET (mängd), Observer kollar varje event innan det läggs i loggen och om eventes value överstiger ett satt värde lagar sensorns ID i ett set. Efter det så kallas logAppend för att lägga in pekaren i loggen, om det skulle misslyckas förstörs pekaren för att eliminera minnesläckor.

EventLog: Event loggen är bygg med en länkad lista, som håller en eventpekare, det är också loggen som i slutet sköter all minneshantering. Min utgångs tanka var att varje event bara skulle skapas en gång och sedan förstöras i slutet för att hålla kopiering till ett minimum detta leder till mer komplex minneshantering och jag vet inte om det är något man gör i prduktion, då det hade varit lättare att skapa en kopia av eventet och lägga in i loggen. Jag valde en länkad lista då man inte behöver sköta storleken på den på samma sätt som en array.

Utils: här finns alla hjälpfunktioner som programmet behöver.

Sortering: jag valde mergesort och insertionsort, då de ska fungera bra på länkade listor men valde istället att spara ned loggen till en array innan sortering så loggen behålls orörd. Inte lika minnes effektivt som att sortera inplace men kan ibland vara bra att ha orginalet orört. Jag valde mergesort för att det är en stabil sortering algoritm och den är effektiv med tids komplexitet på O (n log n)

Insertionsort valde jag för att den är relativt enkel att implementera och stabil men har en sämre tidskomplextiet är mergesort då den generellt ligger på O (n^2) om listan inte är nästan sorterad.

Print och findID har båda linjär tidskomplexitet O (n) då de båda går igenom hela loggen.

ProgramManager: sköter programmet huvudfunktioner som tick + n skapar n antal element, help listar alla tillgängliga kommandon, sort + funtion sorterar loggen, print skriver ut orginal loggen, find + id skriver ut alla event med ett specifikt ID, alarm visar alla ID som för tillfället är triggade, och quit som freear allt minne och avslutar programmet.

Program flöde: eventet skapas av eventproducer och läggs i even queue därefter dequeues eventet av event consumer som lämnar över till Obserer som notifierar AlarmSet och kallar logAppend där loggen blir den slutgiltiga ägaren av objektet. För att hindra minnesläckor så har jag implementerart att eventet förstörs om någon funktion misslyckas