Inlämningsuppgift 1: Flersensorsystem med mätvärdesinsamling och analys

Deadline för kod: Fredag vecka 45 kl. 23:59

Individuell inlämning

Redovisning: Code review av klasskamrats kod, samma deadline som kod

---

Syfte

Den här uppgiften syftar till att du ska

- bygga ett system i C++ med flera klasser och tydlig ansvarsfördelning
- modellera ett enkelt IoT-liknande system med flera sensorer
- hantera mätdata med korrekt struktur, analys och lagring
- designa och implementera ett program med god kodkvalitet
- använda C++-funktionalitet som klasser, vektorer, filhantering och menybaserat användargränssnitt

Det här projektet är också grunden till Inlämningsuppgift 2, där du kommer vidareutveckla systemet med arv, polymorfism och mer avancerad objektorienterad design.

Projektbeskrivning

Du ska bygga ett program i C++ som

- hanterar flera sensorer av olika typer
- simulerar mätvärden från varje sensor
- lagrar mätvärden i ett strukturerat format
- erbjuder analysfunktioner per sensor
- kan läsa och skriva mätdata till fil
- ger användaren ett menygränssnitt i terminalen
- är uppdelat i tydligt separerade klasser och .h/.cpp-filer

Rekommenderad filstruktur

```
projektmapp/
├── main.cpp
├── sensor.h / sensor.cpp
├── measurement.h / measurement.cpp
├── storage.h / storage.cpp
├── utils.h / utils.cpp (vid behov)
└── README.md
```

Del A – Sensorer och simulering

1. Skapa en klass `Sensor` med:
    - Ett namn (till exempel "TempSensor 1")
    - Enhet (till exempel "°C")
    - Simuleringsintervall (min- och maxvärde)
    - En metod `read()` som returnerar ett slumpmässigt flyttal inom det givna intervallet
2. Skapa minst två olika sensorer (till exempel temperatur- och luftfuktighetssensor)
3. Kalla på `read()` för att hämta ett mätvärde

Sensorernas mätvärdes-data ska inte lagras här, utan endast returneras för vidare hantering.

Del B – Mätvärden och datastruktur

1. Skapa en `Measurement`-struktur med:
    - Sensorens namn
    - Enhet
    - Mätvärde (float eller double)
    - Tidsstämpel (till exempel `std::string`)
2. Skapa en klass `MeasurementStorage` som:
    - Innehåller en `std::vector<Measurement>`
    - Har metoden `addMeasurement(const Measurement&)`
    - Har metoden `printAll()`

3. Vid varje avläsning från sensor ska ett `Measurement` skapas och lagras via `MeasurementStorage`

Del C – Statistik och analys

Skapa funktionalitet för att analysera insamlade mätvärden. Statistik ska kunna visas per sensor. Du kan lägga denna logik i `MeasurementStorage` eller en separat klass om du vill dela upp ansvaret.

Analyser som ska kunna göras:

- Antal mätvärden
- Medelvärde
- Min- och maxvärde
- Standardavvikelse

Datautskriften ska vara tydlig och lättläst. Använd till exempel `std::setprecision` för formatering av utskrifter.

Del D – Användargränssnitt och filhantering

Skapa ett menygränssnitt där användaren kan:

1. Läsa nya mätvärden från alla sensorer
2. Visa statistik per sensor
3. Visa alla mätvärden
4. Spara alla mätvärden till fil (CSV- eller textformat)
5. Läsa in mätvärden från fil (som läggs till i befintlig lista)
6. Avsluta programmet

Exempel på filens format:

```
2025-10-28 09:15, Temperatur, 22.5, °C
2025-10-28 09:30, Luftfuktighet, 48.0, %
```

Filhanteringen ska använda `std::ofstream` och `std::ifstream`.

Programmet ska hantera felinmatning på ett robust sätt, till exempel genom att validera menyval eller ignorera felaktiga rader i fil.

Del E – Valfria fördjupningar (för betyg VG)

Välj minst två av följande utmaningar. Alla ska vara integrerade i användargränssnittet och dokumenteras i README-filen.

1. Tidsstämplar med aktuell systemtid

Använd `std::chrono` eller `std::time` för att generera tidsstämplar för varje mätning. Tidsinformationen ska användas vid utskrift av mätdata och statistik.

2. Tröskelvärdesanalys

Låt användaren ange ett kritiskt gränsvärde för varje sensor. Programmet ska:

- hålla reda på tröskelvärdet
- visa hur många mätvärden som ligger över eller under gränsen
- visa vid utskrift om ett mätvärde passerat tröskeln

3. ASCII-histogram

Visa ett enkelt diagram i textformat över mätvärden för en sensor. Exempel:

```
22.0°C | ***
23.0°C | *****
24.0°C | **
```

4. Automatisk mätning i intervall

Skapa ett läge där programmet:

- automatiskt samlar in mätvärden var X:e sekund (till exempel via `std::this_thread::sleep_for`)
- samlar data tills användaren stoppar processen
- simulerar realtidssystem

5. Sensorprofiler

Gör det möjligt att konfigurera varje sensor med en inställningsstruktur (`SensorConfig`) innehållande:

- namn
- enhet
- simuleringsintervall
- tröskelvärde

Denna inställning kan anges i kod eller laddas från fil.

6. Sökfunktion

Låt användaren filtrera mätdata genom att:

- söka på sensornamn
- söka på tidsintervall eller datum
- visa endast de mätvärden som matchar

Tekniska krav

- C++17 eller senare ska användas
- Programmet ska vara uppdelat i `.cpp`- och `.h`-filer enligt god praxis
- All logik ska kapslas i klasser – `main.cpp` ska bara styra flödet
- `new` och `delete` får inte användas i detta projekt

- Standardbiblioteket ska användas där möjligt: `vector`, `string`, `iostream`, `fstream`, `cmath`, `chrono`