Peer Review – Kodgranskningschecklista för Inlämningsuppgift 1

Syfte: Ge och få konstruktiv återkoppling på kodstruktur, namngivning, logik och läsbarhet.
Instruktion:

- Få tillgång till koden från din partner. Försök köra programmet på din egen dator genom att klona ner deras repo. Läs igenom den andres kod noggrant.
- Kryssa i checklistan punkt för punkt. Behöver inte vara lång feedback, men konstruktiv. Ta hjälp av AI för att förstå delar av checklistan bättre.
- Skriv korta kommentarer där det finns förbättringspotential.
- Tänk på att ge **konstruktiv** feedback. Syftet är att hjälpa varandra utvecklas som programmerare.
- Fyll i checklistan direkt i dokumentet
- Skicka feedback-dokumentet till din partner.
  Ladda sedan upp er ifyllda recension på Learnpoint tillsammans med din egen inlämning, eller skicka den till läraren på Teams.
- Diskutera gärna muntligt med kodens författare efteråt.
- **DEADLINE:** Måndag 23.59. För att *få* en recension måste du *ge* en recension.

Sektion 1: Allmänt intryck

☐ Koden kompilerar och körs utan fel eller varningar
☐ Programmet gör det som uppgiften beskriver
☐ README eller kommentar i början av `main.cpp` förklarar programmets syfte
☐ Det går att följa programmets flöde utan att läsa all kod i detalj

Kommentarer:
.....
.....
.....

Sektion 2: Namngivning och tydlighet

☐ Variabler har beskrivande namn (t.ex. `temperature` istället för `t`)
☐ Funktionsnamn och metoder beskriver vad de gör (t.ex. `calculateAverage()`)
☐ Klassnamn och structs använder versal initial, d.v.s. stor första bokstav (`Sensor`, `Measurement`)
☐ Namnkonventionen är konsekvent i hela projektet
☐ Ingen otydlig förkortning eller blandning av språk (svengelska)

Kommentarer:
- Variabelnamn som name, unit, minValue, maxValue är beskrivande.
- Klassnamnet Sensor börjar med en stor bokstav
- Inga svenska uttryck används. Getter-metoder har tydliga namn.

Sektion 3: Struktur och uppdelning

☐ Koden är uppdelad i `.h`- och `.cpp`-filer på ett logiskt sätt
☐ Huvudprogrammet (`main.cpp`) är lättöverskådligt och fokuserar på programflödet
☐ Funktioner/metoder gör **en sak var**
☐ Det finns inga upprepade kodstycken som borde ha flyttats till en funktion
☐ Ingen funktion är orimligt lång eller svår att läsa

Kommentarer:
- Filstruktur:

- **.hpp filer -** (Sensor.hpp,Storage.hpp,Utils.hpp & resource.hpp)
- **.cpp filer -** (Sensor.cpp, Storage.cpp och Utils.cpp)

Huvudprogrammet (main.cpp) är inte lättöverskådligt och fokuserar på programflödet.

Sektion 4: Funktioner, metoder och logik

☐ Funktioner/metoder har rimliga och tydliga parametrar
☐ Returvärden används konsekvent och korrekt
☐ Kontrollstrukturer (`if`, `switch`, `for`, `while`) är tydligt indenterade
☐ Logiken är lätt att följa utan onödiga specialfall
☐ Felhantering finns (t.ex. kontroller av inmatning, tomma listor, nollvärden)

Kommentarer:
- Saknas: read()-metoden.
.....
.....

Sektion 5: Objektorienterad design (om tillämpligt)

☐ Klassen/klasserna har tydligt ansvar (Single Responsibility Principle)
☐ Data (variabler) är inkapslade (`private` eller `protected`)
☐ Metoder som används utifrån är `public` och relevanta
☐ Inga onödiga beroenden mellan klasser
☐ Klasser/structs används på ett meningsfullt sätt – inte bara som "datasäckar"

Kommentarer:
- Det finns en klass(Sensor) och datamedlemmar är private.
.....
.....

Sektion 6: Kodstil och läsbarhet

☐ Indentering och mellanrum är konsekvent
☐ Koden följer en enhetlig stil (t.ex. `{` på ny rad eller samma rad överallt)
☐ Kommentarer används sparsamt och förklarar *varför* snarare än *vad*
☐ Ingen gammal, utkommenterad kod kvar
☐ Inga magiska tal (t.ex. `3.14159` utan namnkonstant)

Kommentarer:
- Kommentarer är förvirrande — de refererar till andra klasser som inte finns här.
.....
.....

Sektion 7: Datahantering och STL

☐ Vektorer eller andra STL-containers används där det passar
☐ Loopar och algoritmer (`std::sort`, `std::accumulate` etc.) används tydligt
☐ Felaktig inmatning hanteras utan att programmet kraschar
☐ Beräkningar (t.ex. statistik) är korrekta och lätta att följa
☐ Utskrifter till användaren är tydliga och användarvänliga

Kommentarer:
- STL används indirekt via std::string.
.....
.....

Sektion 8: Dokumentation och kommentarer

☐ Kommentarer finns i rimlig mängd och beskriver programmets delar
☐ Klass- och funktionskommentarer förklarar syftet tydligt
☐ README eller inledande beskrivning finns och är uppdaterad
☐ Förklaringar till eventuella extra funktioner finns
☐ Eventuella antaganden eller begränsningar nämns

Kommentarer:
- Saknas dokumentation för read() som nämns men inte finns.
- README finns inte
.....

Sektion 9: Samlad återkoppling

Styrkor i koden:
- Bra namngivning

.....
.....

Förslag på förbättringar:
- Lägg till deklarationen för `read()`-metoden

.....
.....

Helhetsbedömning:
☐ Mycket tydlig och välstrukturerad kod
☐ God struktur, men vissa delar kan förtydligas
☐ Vissa designval behöver ses över
☐ Koden fungerar men är svår att följa
☐ Ej fungerande / kräver större omarbetning

- Ej fungerande / kräver större omarbetning.

Granskare

Granskad av: __Faid_____
Datum: ___10/11/2025_____
Kodens ägare: _____Love_____
