Slutuppgifter

Slutuppgiftsperiod på våren: Under en längre period på våren kommer vi att fokusera enbart på slutuppgifterna. Denna tid är avsatt för att du ska kunna arbeta koncentrerat och strukturerat med din slutuppgift. Du får möjlighet att planera, koda och förbättra ditt program. Det är viktigt att du utnyttjar denna period för att visa kontinuerliga framsteg och följa den plan du lagt upp.

Viktiga punkter att tänka på:

1. **Arbetsprocess och kontinuitet**: Först och främst, kom ihåg att detta **inte** är en inlämningsuppgift där du bara lämnar in ett färdigt resultat. Det är viktigt att jag kan följa din arbetsprocess **under lektionstid**. Om du bara lämnar in uppgiften vid slutet av kursen utan att visa kontinuerligt arbete, kommer uppgiften **inte att bedömas**. Endast kontinuerligt arbete räknas, inte en inlämning i sista stund.
2. **Inte bara kod**: Det räcker inte att bara göra uppgiften. Du måste också vara beredd att:
    - Förklara **hur** du har tänkt under arbetet.
    - Svara på frågor om **genomförandet**.
    - Diskutera hur programmet kan **utvecklas vidare**.
3. **Bedömning av svårighetsgrad**: Varje uppgift har en bedömning av svårighetsgraden. Välj uppgifter som motsvarar ditt betygsmål. **Hårt arbete** med flera uppgifter på medelnivå kan ge högre omdöme än att bara göra en svår uppgift halvdant.
4. **Komplexitet och tidsåtgång**: Vissa uppgifter går inte att färdigställa helt under den tid du har. Det är okej. Ju längre du kommer och ju närmare du är en färdig lösning, desto bättre blir ditt resultat.
5. **Planering är nyckeln**: Innan du börjar programmera, **ska du göra en planering** över hur du tänker lösa problemet. Det är viktigt att du inte försöker lösa alla problem på en gång, speciellt om uppgiften är mer komplicerad. Bryt ned problemen i mindre delar.
6. **Välj rätt svårighetsgrad**: Självinsikt är viktigt. Att börja med en uppgift på avancerad nivå, för att sikta på ett högre betyg, är inte en bra strategi om du inte känner att du har rätt kunskapsnivå. Det är bättre att börja med en enklare uppgift, göra klart den, och sedan gå vidare till svårare uppgifter.
7. **Egen kodning**:
    - Du **får inte ladda ner färdigskriven kod** eller använda kod som någon annan har skrivit åt dig.

- Jag kommer att **ställa frågor** om din kod och eventuellt be dig göra ändringar medan jag sitter bredvid. Om du inte har skrivit koden själv kommer det att bli tydligt vid den punkten.
- Det är viktigt att du **själv** skriver all kod. Om du tycker att uppgiften är för svår, börja med en enklare uppgift. Om även de enklare uppgifterna är svåra, bör du gå tillbaka och öva på tidigare kursmoment för att stärka dina grundkunskaper.

8. **Snygghet är inte nödvändigt, men kan vara givande:** Det är inte ett krav att skapa ett grafiskt användargränssnitt (GUI) för ditt program, men det kan vara ett roligt sätt att göra programmet mer tilltalande och användarvänligt. Grundregeln är att du först och främst ska lösa uppgiften i ett textbaserat gränssnitt.
9. **Kommentera din kod (ett krav)**: Kommentering av koden kommer att vara en del av bedömningen. Särskilt när era program blir längre, är det viktigt att vänja sig vid att kommentera koden regelbundet. Kommentarer hjälper både dig själv och andra att förstå hur ditt program fungerar, särskilt vid mer komplicerade delar av koden. **Viktiga delar av ditt program** ska ha tydliga kommentarer som beskriver vad de gör. Detta underlättar både din egen genomgång och bedömningen, och gör det lättare att följa din tankeprocess.
10. **README-fil:** Till en av dina slutuppgifter, ska du redovisa en README-fil på svenska eller engelska som kortfattat beskriver projektet och ger en överblick över dess funktion och användning. Följ strukturen nedan när du skriver din README.

    1. **Sammanfattning (Summary)**

       Namnge ditt projekt och beskriv det kortfattat i 2-3 meningar. Vad gör ditt program? Håll beskrivningen enkel och lättförståelig.

    2. **Bakgrund (Background)**

       Vilket problem kommer din idé att lösa? Hur vanligt eller frekvent är detta problem? Vad är din personliga motivation för att välja detta projekt? Varför tycker du att detta ämne eller problem är viktigt eller intressant att lösa med programmering?

    3. **Nyckelaspekter (Key aspects) (frivilligt)**

       Om du vill, kan du beskriva några viktiga funktioner eller tekniska detaljer som du tycker är särskilt värda att lyfta fram. Här kan du till exempel nämna något intressant om hur programmet är uppbyggt eller hur du löste ett specifikt problem.

    4. **Hur används programmet (How is it used?)**

       Tänk på detta som en enkel användarmanual för ditt program. Beskriv hur man kör ditt program och hur det fungerar. Om det är relevant, kan du ge exempel på vad som händer när programmet

körs och vad användaren kan förvänta sig för resultat. Om det finns speciella kommandon eller inmatningar användaren måste göra, förklara dessa här.

5. **Utmaningar (Challenges)**

   Vilka delar av problemet löser ditt projekt inte? Det är viktigt att förstå att alla tekniska lösningar har sina begränsningar.

6. **Vad härnäst (What next?)**

   Hur kan ditt projekt utvecklas vidare? Vilka funktioner skulle du kunna lägga till för att förbättra programmet? Finns det möjligheter att göra det mer användarvänligt, effektivt eller spännande? Reflektera över hur projektet kan växa och bli något ännu bättre.

En lämplig textlängd för **README-filer** är att de inte överstiger **1-2 sidor** i text, eller cirka **300-500 ord**.

Exempel på README för ett Sten-sax-påse-spel:

1. Sammanfattning
Sten-sax-påse är ett textbaserat spel där spelaren möter datorn i ett klassiskt matchande-spel. Spelaren väljer mellan sten, sax eller påse, och datorn gör ett slumpmässigt val. Programmet jämför valen och avgör vem som vinner.

2. Bakgrund
Sten-sax-påse är ett enkelt och roligt spel som alla känner till. Genom att skapa detta spel kan jag träna på att använda villkorsstyrd logik och slumpmässiga val i programmering. Det är ett bra exempel på hur man kan arbeta med inmatning från användaren och jämföra olika val för att avgöra ett resultat.

3. Nyckelaspekter *(frivilligt)*
En viktig aspekt av programmet är hur datorns val genereras slumpmässigt med hjälp av en slumptalsgenerator.

4. Hur används programmet?
För att spela spelet, starta programmet och välj mellan sten, sax eller påse genom att skriva in ditt val när du blir ombedd. Datorn gör samtidigt sitt val, och programmet visar resultatet av omgången. Spelet kan spelas flera gånger tills användaren väljer att avsluta.

Exempel på körning:

```
Välj: sten, sax eller påse: sten

Datorn valde: sax

Du vann! Sten slår sax.

Vill du spela igen? (j/n): j
```

5. Utmaningar
Programmet hanterar endast ett enstaka spel åt gången. Det saknar avancerade funktioner som poängräkning eller möjlighet att spela mot en annan människa. Dessutom finns ingen felhantering om användaren skriver in ogiltiga värden.

6. Vad härnäst (What next?)
I framtiden kan spelet utvecklas genom att hålla räkningen på hur många gånger varje spelare har vunnit. En annan förbättring kan vara att implementera en meny där spelaren kan välja mellan olika spellägen, till exempel att spela mot en mänsklig motståndare.

Inlämning och redovisning:

- **Muntlig redovisning:** Du kommer att redovisa din lösning muntligt och förklara hur du har arbetat med uppgiften. Du bör vara beredd på att svara på frågor om koden.
- **Inlämning av kod:**
    - **Alternativ 1:** Lämna in din kod som ett kodblock i ett Google-dokument. Använd formatet ***Byggsten-kodblock (C++)*** för att tydligt visa din kod.
    - **Alternativ 2:** Länka till ditt program om du har en **GitHub-repository,** så att koden kan granskas direkt i din repository.
    - **Koden lämnas in någon lektion innan den muntliga redovisningen med ett meddelande som talar om att inlämning av koden skett.**