Loops / forgreninger (switch)

Læringsmål	Du kan: • **1Pf1**: anvende centrale metoder til at specificere og konstruere algoritmer [...] • **1Pf2**: anvende centrale faciliteter i programmeringssproget til realisering af algoritmer […] • **1Pf3**: anvende et i professionen udbredt, integreret udviklingsværktøj, herunder versionsstyringssystem […] til at designe og konstruere praksisnære applikationer […] • **1Pk3**: i en struktureret sammenhæng tilegne sig ny viden, færdigheder og kompetencer inden for programmeringssprog, udviklingsværktøjer, programmeringsteknikker og programdesign
Forventede produkter	• Implementering af programmeringsøvelserne
Din forberedelse	Som minimum til opgavens løsning, da læs tekst eller gennemse videoer markeret med **fed tekst** (eller start med dem). Se videoerne og læs teksterne i den rækkefølge, der er angivet forneden. Videoer og tekster uden et link findes i Planer i ItsLearning. Programmering: • The char type - repetition • **The C# Switch Statement (video: 9:39)** • **Switch Statement in C#** • C# Switch Examples • **C# While Loop (video: 7:26)** • **C# Do While Loop (video: 6:18)** • **Working with Code Files, Projects, and Solutions (video: 11:35)**

Du skal i denne opgave se videre på styring af programflowet med switch-sætningen, endnu en forgrenings-sætning (en: conditional), og med de to løkketyper (en: loops) while og do-while.

Øvelse 1: Programmering

De følgende øvelser er en lang række af primært færdighedsopgaver, som dækker stoffet til i dag samt repetition af tidligere emner. Emnerne er blandede, så du kommer godt rundt i stoffet, også selvom du ikke når dem alle. Du kan altid vende tilbage og lave de øvelser, du ikke har nået, når du har tid.

- Lav så mange øvelser, du kan nå, med dagens makker.
- Du skal anvende **parprogrammering**, så husk at skift rolle (driver / navigator) ofte, f.eks. ved hver øvelse. Det er forbudt, at flere programmerer samtidigt.

Øvelse 1.1: Terminologi

Del teamet op i to mindre grupper, og brug **Ordet rundt** til at reflektere over de tre sætningstyper ”switch”, ”while” og ”do-while”. Sørg for, at alle får mulighed for at tale.

Tidsramme: 10 minutter

Øvelse 1.2: Forgrening med Switch

Switch er den anden forgreningssætning (en: conditional statement) ud af to, hvor den første er if-else. Switch og if-else kan det samme, men afhængigt af hvad man vil opnå (øget læsbarhed, færre linjer, mm.), har de to syntakser fordele og ulemper. Det lærer du efterhånden ved at bruge dem.

- Tilpas din kode fra Menu-øvelsen i den tidligere opgave med brug af switch-sætningen i stedet for if-else-sætningen. Programmet udskriver en simpel menu samt en besked afhængig af menuvalg.

Øvelse 1.3: Debugging igen

Det kan ikke siges for ofte: Har man en fejl i sin kode, som er svær at finde, kan man med fordel debugge (”afluse”) koden i stedet for at køre hele koden på en gang. At debugge betyder, at man kører programmet trin for trin, dvs. sætning for sætning. Man kan derved følge programmets flow nøjere, overvåge hvordan hver variabel skifter værdi, og se, hvor fejlen opstår i koden. **Debugging er en meget stærk teknik, som du anbefales at anvende hele tiden (essentiel færdighed), så du hurtigst muligt får det på rygraden.** Udover at finde fejl, så kan debugging altid hjælpe en med at forstå kode hurtigere og måske få ideer til at optimere, specielt hvis en anden har skrevet koden.

I denne øvelse skal du prøve at gå trinvist igennem jeres kode med funktionstasten **F10 (Step Over)**.

Udfør følgende:

- Debug din kode fra forrige øvelse ved at starte med at trykke F10 (Step Over) for at starte debugging. Programmet går i gang og stopper på der første mulige trin i koden; i denne opgave er det starten af Main()-metoden.
- Klik nu på Locals-fanen for at kunne løbende følge med i ændringer af variable mm.
- Med F10 gå igennem koden trin for trin.

Du kan på ethvert tidspunkt inspicere alle de variable, som er tilgængelige, hvor koden er stoppet, bl.a. også ved at sætte musepilen over et variabelnavn i koden.

Øvelse 1.4: Læs enkelte karakterer i en tekststreng

En tekststreng (repræsenteret ved datatypen **string**) er i realiteten en ordnet samling af karakterer, hvor hver karakter er af typen **char**. Man kan tilgå hvert enkelt element i en string-samling ved at benytte notationen:

- <variabelnavn> ”[” + <index> + ”]”, hvor <index> er et heltal fra 0 og opefter.

Index angiver den position i tekststrengen, man ønsker at læse. Bemærk at første element har index 0.

Eksempelvis hvis man ønsker at læse den fjerde karakter ’d’ i nedenstående string-variabel `str`:

```
string str = "abcdefg";
```

skriver man:

```
char ch = str[3]; // 'd'
```

Bemærk, man kan kun læse en karakter fra en tekststreng med [ ]-notationen og aldrig sætte den. Det gøres på andre måder.

Implementér følgende:

- Indlæs en vilkårlig tekststreng (på mindst seks karakterer)
- Læs og udskriv specifikt det andet, fjerde og sjette element (dvs. karakter) i tekststrengen

Debug koden med F10 (Step Over).
Når alle fejl er fundet og rettet, kør koden med F5.

Øvelse 1.5: While-løkke: Gennemløb alle karakterer i en tekststreng

Udskriv alle karakterer i en tekststreng kun ved hjælp af en while-løkke:

- Indlæs en vilkårlig tekststreng
- For hver enkel karakter i tekststrengen:
  - Udskriv karakteren på formen ”<index>: ’<karakter>’”

Eksempel:

- Tekststrengen ”hej” bliver til de 3 linjer:
  0: ’h’
  1: ’e’
  2: ’j’

Gode overvejelser til dig, før du begynder at kode:

- Tekststrengen kan jo have vilkårlig længde, da du ikke ved, hvad brugeren vil indtaste. Det skal du tage højde for i koden. Det kan bl.a. være godt at finde ud, hvor lang den indtastede tekststreng er, inden man udskriver hver enkel karakter. While-løkken styres jo af en betingelse, der afgør om løkkens kodeblok skal udføres eller ej. Overvej først algoritmisk, hvordan du styrer det rigtige antal aktiveringer af kodeblokken, før du begynder at kode.

Debug koden med F10 (Step Over).
Når alle fejl er fundet og rettet, kør koden med F5.

Øvelse 1.6: Do-while-løkke: Gennemløb alle karakterer i en tekststreng

Implementér nu forrige øvelse 1.5, men denne gang udelukkende ved hjælp af en do-while-løkke.

Overvej forskellen, og om der kan opstå problemer for visse tekststrenge.

Øvelse 1.7: Heltalsdivision

Du har allerede tidligere (beregning af hældningskvotient) oplevet, at division med to heltal ikke helt går, som du måske først forventede det. Forhåbentlig har du opdaget, at:

- Dividerer man to heltal (int-datatypen) med hinanden i C#, bliver resultatet uundgåeligt også et heltal, dvs. datatypen int.

Derfor kan 3/2 ikke blive til 1,5 men kun til heltalskvotienten 1, da resultatet skal være et heltal.

Så 2 går op én gang i 3, men hvordan finder man så rest-delen?

- Åben google, og søg efter: ”C# remainder operator”.

Benyt den fundne operator til at implementere følgende:

- Indlæs første heltal
- Konvertér og gem heltalsværdien i int-variablen a
- Indlæs andet heltal
- Konvertér og gem heltalsværdien i int-variablen b
- Beregn og gem (i passende variabel) heltalskvotienten ved heltalsdivisionen a/b
- Beregn og gem (i passende variabel) rest-delen ved heltalsdivisionen a/b
- Udskriv den fundne heltalskvotient og rest-del

Debug koden med F10 (Step Over).
Når alle fejl er fundet og rettet, kør koden med F5.

Øvelse 1.8: Hold programmet kørende, indtil brugeren afslutter det

Stort set alle øvelser indtil nu har fulgt mønstret:

1. Indlæs data
2. Beregn eller behandl data
3. Udskriv resultat

Efter disse tre trin afsluttes programmet, hvilket betyder, at du skal starte programmet igen for at prøve med nogle nye data. Det er ret irriterende, hvis du gerne vil prøve med flere data, så det må du gøre noget ved.

Du skal fremover anvende følgende korrigerede mønster:

1. Så længe brugeren ønsker at indtaste data, da:
    a. Indlæs data
    b. Beregn eller behandl data
    c. Udskriv resultat

Overvej, hvilken type af programflow der kan realisere dette mønster?

Udfør følgende:

- Opdatér koden i øvelse (den med menuen), så det bruger det nye mønster, og man bliver i programmet, indtil en passende slutbetingelse er opfyldt. Find selv på en slutbetingelse, som

brugeren kan vælge til at afslutte programmet, dvs. udtrykker at brugeren IKKE ønsker at indtaste mere data, men afslutte.
- Det kan i øvrigt være fornuftigt at 'rense' konsolvinduet for indhold, før brugeren indlæser nye data. Indtast "Console." (husk punktum) i editoren og brug IntelliSense til at finde en passende metode til netop dette.

Brug det nye mønster i de følgende øvelser, men vær opmærksom på, at en slutbetingelse nok skal angives på forskellige måder afhængig af programmets funktion.

Øvelse 1.9: Udskriv hver anden karakter i en tekststreng

Tag udgangspunkt i din kode fra øvelse 1.5, og udskriv denne gang udelukkende hver anden karakter i tekststrengen. Hint: remainder-operatoren kan måske bruges ☺

Debug koden med F10 (Step Over).
Når alle fejl er fundet og rettet, kør koden med F5.

Øvelse 1.10: Tekststrenge med cifre

Udskriv alle forekomster af cifre, dvs. karaktererne '0' til '9', du finder i en tekststreng.

- Indlæs en vilkårlig tekststreng (indsæt cifre hist og pist)
- For hver enkel karakter, som er et ciffer i tekststrengen:
  - Udskriv ciffer-karakteren på formen "<index>: '<karakter>'"

Debug koden med F10 (Step Over).
Når alle fejl er fundet og rettet, kør koden med F5.

Øvelse 1.11: Tekststrenge med cifre og operatorer

Udskriv alle forekomster af cifre, dvs. karaktererne '0' til '9' samt operatorerne '+' og '-', du finder i en tekststreng.

- Indlæs en vilkårlig tekststreng (indsæt cifre hist og pist)
- For hver enkel karakter i tekststrengen:
  - Hvis karakter er et ciffer:
    - Udskriv ciffer-karakter på formen "<index>: '<karakter>' (ciffer)"
  - Hvis karakter er en operator '+' eller '-'
    - Udskriv operator-karakter formen "<index>: '<karakter>' (operator)"
  - Ellers udskriv karakteren på formen "<index>: '<karakter>' (ukendt)"

Overvej, hvilken forgreningssætning (en: conditional) egner sig bedst her: Evt. prøv begge, du kender.

Debug koden med F10 (Step Over).
Når alle fejl er fundet og rettet, kør koden med F5.

Øvelse 1.12: Skan og udregn simple regneudtryk (lidt sværere)

Med udgangspunkt i koden fra forrige øvelse 1.11, skal brugeren nu kunne angive et regneudtryk med cifrene '0' til '9' samt operatorerne '+' og '-'. Koden skal skanne tekststrengen med regneudtrykket, finde cifre og operatorer og beregne værdien af regneudtrykket, der angives. F.eks.:

- Tekststrengen "4-5+9" skannes og udregnes til 8
- Tekststrengen "7+4-3+6" skannes og udregnes til 14

Dette sætter selvfølgelig nogle krav til et gyldigt regneudtryk:

- Et udtryk skal starte og slutte med et ciffer
- Et ciffer skal efterfølges af en operator med mindre, at det er det sidste ciffer i udtrykket

- En operator skal følges af et ciffer
- Alle karakterer, der ikke er et ciffer, '+' eller '-', er ugyldige.
  Sørg for at meddele brugeren, at der er angivet et ugyldigt udtryk, hvis en ugyldig karakter forekommer i udtrykket.

Disse krav sætter de fleste emner, vi har set på, i spil, både i C# og i Computational Thinking.

Udbyg koden fra øvelse 1.11.
Debug koden med F10 (Step Over).
Når alle fejl er fundet og rettet, kør koden med F5.

Øvelse 1.13: Multiplikation og division (endnu sværere)

Hvis du bliver færdig med øvelse 2.12, inden tiden er gået, da implementer også operatorerne '*' (multiplikation) og '/' (division) i regneudtrykket.