

DA339A Laboration L10

Syfte

Den här laborationens syfte är att öva på att använda arrayer och iterationer som verktyg vid problemlösning och att skriva kod som innehåller arrayer. Laborationens syfte är att studenten ska öva programmering och problemlösning med arrayer och iterationer samt öva på vanliga algoritmer förknippade med arrayer. Lösningarna skall skrivas som körbbara Java-program.

Redovisning

En delmängd av uppgifterna i den här laborationen ska redovisas för godkänd laboration L10. Godkänd laboration L10 krävs för godkänt betyg G på provkod 2007 Laborationer och workshoppar del 1.

Vilka uppgifter som ska kunna redovisas är noterat efter uppgiftens rubrik. Av de uppgifter som ska kunna redovisas kommer ett slumprässigt urval att göras vid redovisningen för vad som ska redovisas då. Urvalet görs av den lärare eller assistent studenten redovisar för.

Även om inte alla uppgifter ska kunna redovisas för godkänd laboration L10 förväntas studenten för sin egen övnings skull arbeta med alla uppgifterna (enkla uppgifter som inte kräver redovisning innebär ofta en lösning på ett delproblem i de större uppgifter som ska redovisas).

Laboration 10 kan tidigast redovisas, på laboration, vid det angivna Kronox-schemat tillfället (kurstakt L10). Är man inte redo att redovisa vid detta tillfälle kan laborationer redovisas vid vilket senare labb-hjälps-tillfälle som helst som inte är i workshop-format.

Genomförande av redovisning

Laborationen redovisas muntligt och genom uppvisande av käll-kod, kompilering och exekvering av kod. Studenten ska kunna visa upp sin källkod på dator för lärare/assistent som examinerar och kunna visa hur hen kompilar och exekverar koden på dator. Studenten ska också kunna svara tillfredställande på frågor om sin kod och de lösningar hen implementerat samt kunna demonstrera förmåga att göra mindre ändringar i koden direkt, kompilera och exekvera igen (exempelvis ändra indata eller något gränsvärde). Studenten ska även kunna visa upp andra lösningar som krävs i laborationen exempelvis diagram eller sanningstabeller.

Redovisningen är en examination och är fokuserad på att göra en bedömning av den kod studenten producerat. Det finns inte tid vid redovisningen att ge utförligare kommentarer om de uppgifter som inte väljs ut att demonstreras vid redovisningen. Det finns vid denna redovisning inte heller tid att studenten beskriver sin hela tankeprocess eller förklarar hela programmet. Den lärare eller assistent som studenten redovisar för kommer att ställa frågor och be studenten demonstrera vissa saker. Diskussioner utöver detta finns det inte utrymme för vid redovisningen.

Önskar studenten återkoppling i större utsträckning på en lösning ges detta som vanlig hjälp vid laborationer.

Förberedelser för redovisning

Studenten förväntas vara förberedd när redovisningen startar och ha följande förberett innan redovisningen startar:

- Ha kommandotolk startad och redo att kompilera och exekvera de uppgifter som kan redovisas (det vill säga ha nавигerat till den katalog du har dina filer i innan din redovisningstid startar).
- Ha källkodsfiler öppnade och tillgängliga i editor för att kunna visa upp koden och kunna göra ändringar i denna vid förfrågan.

Generella krav vid redovisning

För godkänd redovisning krävs:

- Väl formaterad källkod.
- Källkod ska gå att kompilera och exekvera med ett resultat som efterfrågas för respektive uppgift via kommandotolk.
- Att uppgifterna är lösta på rimligt sätt oavsett om det är i kod eller annan form av dokumentation av lösning som efterfrågas.
- Studenten ska muntligen kunna förklara sina lösningar och beskriva varför lösningen ger det efterfrågade resultatet.
- Studenten ska kunna visa förmåga att göra mindre ändringar i koden och kompilera, exekvera den ändrade koden med efterfrågat resultat via kommandotolk

Ingen skriftlig inlämning sker i redovisningen. Studenten ska alltid vara beredd på att kunna visa legitimation vid redovisning.

Förberedelser

Den här laborationen är direkt kopplad till det innehåll som finns i föreläsning F10 och F11. För att kunna genomföra laboration L10 behöver studenten ha bekantat sig med begrepp från F10 och F11, framför allt att traversera en array och grunderna för de algoritmer som presenterades på föreläsning F10 och F11.

Laborationen förutsätter även att studenten är bekant med begrepp från tidigare föreläsningar F3-F9 så som exempelvis iteration och selektion.

För laboration L10 förutsätts studenten även behärska tidigare genomgångens innehåll på kursen.

Det finns en fil *Test_parseInt.java* som förklarar hur man konverterar tal till strängar och strängar till tal. Bekanta dig med denna då du kommer att ha nytta av det i laborationen. Java-filen finns att ladda ner på kurssidan.

Du behöver inte ladda ner några andra filer för att lösa uppgifterna. Du använder de program och lösningar du har jobbat med sedan innan i tidigare laborationer. Där finns i vissa fall mindre stycken med kod i uppgifterna som du ska använda i din lösning (oftast någon form av testdata). Se upp om du kopierar in denna text med klipp/klistra till din kod - det kommer ibland in extra tecken när man gör detta som inte syns men som ställer till det i kompileringen eller exekveringen.

Uppgifter

Lösningarna till uppgifterna ska skrivas i Java-kod och köras i kommandotolk vid redovisning. Du förväntas lösa uppgifter **utan** ”genvägar” som try-catch, hjälpklasser som Arrays eller ArrayList eller liknande då problemlösning är en del av det som examineras.

Uppgift 1 Linjär sökning

Utgår från din lösning till L9 Uppgift 1 och skriv ett Java-program som talar om ifall ett sökt heltalet finns i en array eller inte. Programmet ska avbryta att leta efter det sökta heltalet om det hittar en matchning. Om det sökta talet inte finns ska detta meddelas till användaren, annars ska programmet tala om att ”talet X hittades i arrayen” där X ersätts av det tal som söktes.

Indata i form av det sökta talet kan deklarerats i början av programmet och behöver inte matas in av användaren (men du får gärna göra en lösning som läser in indata från användaren om du vill). Kodraden nedan ska användas för den array vi söker talet i.

```
int[] testArray = {32, 27, 64, 18, 95, 14, 90, 70, 60, 37};
```

Uppgift 2a Hitta största

Utgå från din lösning till L9 Uppgift 2a och skriv ett Java-program som hittar det största värdet i en array med heltalet och skriver ut värdet samt positionen av det största talet i arrayen. Gör följande varianter av programmet:

- om det största värdet finns på mer än en position så ska det med lägst position hittas och sparas när loopen kört klart.
- om det största värdet finns på mer än en position så ska det med störst position hittas och sparas när loopen kört klart.

Du kan använda samma array som till Uppgift 1 ovan för att testa ditt program, men modifiera denna för att testa om det finns två lika stora och största tal i arrayen.

Uppgift 2b Hitta minsta

Utgå från dina lösningar i 2a och förändra dessa så att du skapar ett nytt Java-program som söker det minsta värdet istället.

Uppgift 3a Datautskrift

I en tabell som representeras av en array finns textsträngar med data strukturerad enligt mönstret:

- kolumn 0: förnamn
- kolumn 1: efternamn
- kolumn 2: födelsedatum
- kolumn 3: poäng
- kolumn 4: betyg

Varje rad representerar en students resultat.

Utgå från din lösning till L9 Uppgift 3 och skriv ett Java-program som går igenom arrayen och skriver ut efternamn samt poäng på alla med ett godkänt eller väl godkänt betyg.

Koden nedan ska användas för en array med testdata i ditt program.

```
String[][] students={{ "Adam", "Ason", "661122", "35", "U" },
{ "Beata", "Bson", "770111", "38", "G" }, { "Calle", "Ceson", "880222", "23", "U" },
{ "Dorotea", "Deson", "990311", "44", "VG" }, { "Eivar", "Eson", "550423", "40", "G" }};
```

Uppgift 3b Datautskrift

Betygsgränsen har ändrats så tabellen som användes i Uppgift 3a måste uppdateras. De nya betygsnivåerna är 25 för G och 40 för VG. Skriv ett program som går igenom arrayen och ändrar betyg efter de nya gränserna. Skriv ut förnamn och efternamn samt poäng och betyg för alla resultat för att kontrollera att ändringen blir rätt.

Du ska alltså inte ändra initieringen av arrayen som är testdata ovan till att matcha de nya gränserna. Denna array med testdata ska fortfarande användas när programmet startar. Ditt program ska göra ändringarna i arrayen genom att kontrollera poängen i kolumn 3 och uppdatera kolumn 4 med korrekt betyg efter de nya gränserna för G och VG.

Ett tips är att du behöver använda metoder för att omvandla strängarna i arrayen till heltal som du kan jämföra betygsgränserna mot. Se exempel i filen *Test_parseInt.java*.

Uppgift 4 Mätvärden

Utgå från din lösning till L9 Uppgift 4 och skriv ett Java-program som tar fram medelvärdet av temperaturen under dygnet samt det största och minsta uppmätta värdet och skriver ut dessa. Testa ditt program med koden för indata nedan. Indatan kan deklareras i början av programmet och behöver inte matas in av användaren.

```
double[] testArray = {3.3, 2.7, 6.4, 1.8, 9.5, 1.4, 9.0, 7.0, 6.5, 3.7};
```

Försök göra din lösning så att du endast behöver en loop för att ta fram alla de tre sakerna: medelvärde, största tal och minsta tal (det vill säga använd inte tre separata loopar för detta).

Utmaning: Ändra testdatan så att vissa tal är negativa. Fungerar ditt program fortfarande?

Uppgift 5 Kalibrera sensorer

Utgår från din lösning till L9 Uppgift 5 och skriv ett Java-program som jämför värdena i de två arrayerna och talar om ifall arrayerna är lika eller inte. Om arrayerna inte är lika så vill man veta hur många fall som skiljer sig och hur stor den största skillnaden var. Testdata för ditt program finns att kopiera in nedan:

```
double[] testArray1 = {3.3, 2.7, 6.4, 1.8, 9.5, 1.4, 9.0, 7.0, 6.5, 3.7};
double[] testArray2 = {5.6, 4.7, 2.8, 3.7, 5.8, 2.7, 6.4, 1.8, 9.5, 10.2};
```

Uppgift 6 Redundant data REDOVISAS

Utgå från din lösning till L9 uppgift 6a eller 6b (du kan välja den lite enklare 6a eller den lite knepigare 6b) och skriv ett Java-program som går igenom arrayen med rådata och tar bort den data som har värdet 0. Den data som är relevant ska lagras mer kompakt (det vill säga ta bort 0 ur datan) i en ny array (6a) eller göra allt i samma array (6b). Lösningen ska fungera oavsett storlek på den array som är indata. Skriv ut både indata och utdata i kommandotolken för att se resultatet av ditt program.

Indatan kan deklaras i början av programmet och behöver inte matas in. Exempel att använda som indata:

```
int[] array = {12,13,14,13,0,0,15,15,0,13};  
int[] array = {0,12,13,14,13,0,0,15,15,0,13,0,0,15,34};
```

Uppgift 7 Spara det största REDOVISAS

Utgå från din lösning till L9 Uppgift 7 och skriv ett Java-program som implementerar jämförelsen av arrayerna. För varje position i den nya arrayen ska programmet skriva ut vilket värde som sparas där och om det kommer från array 1 eller array 2 i indata (görs när värden sparas i den nya arrayen). Programmet ska fungera oavsett storlek på arrayerna som är indata. Programmet ska ge ett felmeddelande till användaren om man försöker köra det med två arrayer som inte har samma antal rader och kolumner.

Testdata för två lika stora arrayer:

```
int[][] array1 = {{1,2,3},{11, 12, 13},{6, 3, 23}};  
int[][] array2 = {{1,1,4},{1, 2, 3},{21, 7, 23}};
```

Testdata med två olika stora arrayer (byt ut en array ovan, tänk på att arrayerna inte ska ha samma namn):

```
int[][] array1 = {{1,4},{2, 3},{6, 7, 2}};
```

Skapa egen utökad testdata som där arrayerna har en annan storlek än ovan och testa med detta.

Uppgift 8 Selection Sort REDOVISAS

Skriv ett Java-program för Selection Sort. Skriv ut både indata (arrayen innan den sorterats) och utdata (arrayen efter att den sorterats). Lösningen ska fungera för olika storlekar på arrayer som är indata.

Tänk på att lösa ett problem i taget och bygga upp din lösning efter hand. Ett tips är att skriva ut hela arrayen i varje iteration så du ser hur algoritmen fungerar.

Testa ditt program genom att använda indata nedan:

```
int[] array = {32, 27, 64, 18, 95, 14, 90, 70, 60, 37};
```

Utmanship (behöver inte redovisas): Selection Sort, Insertion Sort och Bubble Sort är snarlika. Använd din kod till Selection Sort och förändra den för att skapa kod för Insertion Sort och Bubble Sort.

Uppgift 9 Spelplan REDOVISAS

Utgår från din lösning för L9 Uppgift 12 och skriv ett Java-program som får en två-dimensionell array som input.

Indatan är en array som representerar en spelplan där figurer kan stå på olika positioner. I arrayen är det lagrat olika heltalsvärdet som avgör hur farligt det är att stå på en viss position. Programmet ska fungera oavsett storlek på arrayen som är indata.

Utdatan ska vara en array med booleska värden (true/false) som anger om man överlever till nästa runda på en viss position. True representerar att figuren skulle överleva på positionen och false att den inte skulle överleva.

Regler för att överleva/dö:

- Om positionen en figur står på har värdet 3 dör figuren direkt.
- Om det sammanlagda värdet av grannarna till positionen (utan positionens egna värde) är 15 eller mer dör figuren direkt.
- Om positionen figuren står på har värdet 2 eller mindre och grannarnas sammanlagda värde är mindre än 15 lever figuren vidare.

Grannar som saknas i kanterna räknas som värdet 0.

Programmet ska skriva ut arrayen som är indata i början av programmet och sedan skriva ut den nya arrayen med true/false i slutet av programmet.

Indatan kan deklarerats i början av programmet och behöver inte matas in. Skapa din egen array för indata.

För de som ligger före: Lösningen får inte använda try-catch för att lösa problemet med att trilla ur arrayen när man undersöker grannar.

Exempel:

Array som ges som input: Resultat i ny array:

1	1	2	3	3
2	1	1	2	3
3	2	2	1	2
3	3	3	3	3

T	T	T	F	F
T	T	T	F	F
F	F	F	F	T
F	F	F	F	F

Ledning: Om du behöver hjälp med att lösa offset för grannar och kontroll för att inte trilla ur arrayen – titta på kodfilen *NbrTrueNeighbours.java* från L9.

Uppgift 10a Tandläkarklinik

Utgå från L4 Uppgift 7 och skriv kod för din lösning (du har även delar av den här lösningen implementerad via tidigare laborationer). Du kan utgå från koden i filen *DentistReception.java* från L5 och din lösning där eller från din egen lösning i senare laborationer.

Istället för att använda konstanter när du ska beräkna kostnaden och för att göra det enklare att ändra och lägga till nya behandlingar skall du använda en array för att lägga in behandlingar och motsvarande kostnad. Arrayen med de olika behandlingarna och kostnaderna ska skapas enligt mönstret:

- Kolumn 0: Namnet på behandlingen
- Kolumn 1: Kostnaden för behandlingen

```
String [][] behandling = {{"Kontroll", "600"}, {"..."}};
```

Lägg in följande behandlingar i din array:

- Kontroll 600kr
- Lagning hål 1500kr
- Rengöring 300kr

Använd datan i arrayen för att skriva ut behandlingen och kostanden till användaren när denna ska göra sitt val.

Du ska även hämta informationen om kostanden från arrayen när kostnaden för den enskilda behandlingen användaren valde ska läggas till den totala kostnaden. När användaren gör sitt val i menyn låter du valet motsvara index för raden för behandlingen i arrayen och hämtar kostnaden från motsvarande rad och kolumn 1 i den raden (utom när man väljer -1 för att man är klar). Glöm inte att omvandla strängen från arrayen till integer med `parseInt()` när du ska öka den totala kostnaden.

Uppgift 10b REDOVISAS

Utöka ditt Java-program från Uppgift 10a ovan så att det också skapar ett kvitto för de utförda behandlingarna när användaren matat in alla behandlingar som utförts.

Börja med att skapa en tom array med 10 element. Arrayen behöver inte ha mer än 10 element eftersom det statliga högkostnadsskyddet alltid går in efter 10 behandlingar oavsett pris (du kan avbryta loopen för inmatning av behandlingar vid 10 behandlingar). Programmet får inte krascha om man anger mer än 10 behandlingar. Spara ner behandlingarna i arrayen allt eftersom de matas in av användaren (man lagar exempelvis flera hål som ska lagas). Skriv ut kvittot med priset både före och efter rabatten. Titta på metoden `String.format()` om du vill skapa en snyggt formaterad utskrift (inte nödvändigt för godkänt).

Exempel på utskrift av kvitto:

Kvitto	
Rengöring	300kr
Kontroll	600kr
Lagning hål	1500kr
Lagning hål	1500kr
Lagning hål	1500kr
Rengöring	300kr
<hr/>	
Kostnad	5400kr
Rabatt	540kr
Summa att betala	4950kr