|  |
| --- |
| Guide |

Frekvenstabell över tärningskast med C#

Introduktionsuppgift

|  |
| --- |
| *Författare:* Mats Loock  *Kurs:* Inledande programmering med C#  *Kurskod:*1DV402 |

**Upphovsrätt för detta verk**

Detta verk är framtaget i anslutning till kursen Inledande programmering med C# vid Linnéuniversitetet.

**Du får använda detta verk så här:**

Allt innehåll i verket Frekvenstabell över tärningskast med C# av Mats Loock, förutom Linnéuniversitetets logotyp, symbol och kopparstick, är licensierad under:

|  |  |
| --- | --- |
| Creative Commons-licens | Creative Commons Erkännande-IckeKommersiell-DelaLika 2.5 Sverige licens. <http://creativecommons.org/licenses/by-nc-sa/2.5/se/> |

**Det betyder att du i icke-kommersiella syften får:**

* kopiera hela eller delar av innehållet
* sprida hela eller delar av innehållet
* visa hela eller delar av innehållet offentligt och digitalt
* konvertera innehållet till annat format
* du får även göra om innehållet

Om du förändrar innehållet så ta inte med Linnéuniversitetets logotyp, symbol och/eller kopparstick i din nya version!

Vid all användning måste du ange källan: ”Linnéuniversitetet – Inledande programmering med C#” och en länk till <https://coursepress.lnu.se/kurs/inledande-programmering-med-csharp> och till Creative Common-licensen här ovan.

# Innehåll

Du ska följa ”steg för steg”-instruktionen i denna introduktionsuppgift och skapa en konsolapplikation med C# och Visual Studio 2012. Applikationen ska slumpa i det slutna intervallet mellan 100 till 1000 tärningskast och presentera en frekvenstabell över förekomsten av ettor, tvåor, osv.

|  |  |
| --- | --- |
|  | Starta Visual Studio 2012.    Figur 1. |
|  | Du ska skapa ett projekt för en konsolapplikation med C#, välj därför File ► New ► Project....    Figur 2. |
|  | Dialogrutan New Project visas.    Figur 3.   1. Markera Visual C# under Installed, Templates. 2. Kontrollera så att .NET Framework 4.5 visas i den nedrullningsbara listrutan. 3. Markera Console Application. 4. Vid Name skriver du in projektets namn (RollFrequencyTable). 5. Ange vid Location en lämplig katalog där projektet ska sparas (t.ex. C:\Projects). |
|  | Visual Studio skapar ett nytt konsolprojekt som placerats i en egen ”*solution*” med samma namn som projektet. Det nya projektet innehåller bland annat filen Program.cs.  Filer är organiserade i projekt och ett projekt kan innehålla flera filer som tillsammans utgör applikationen.  Projekt är organiserade i ”*solutions*”. En ”*solution*” kan innehålla flera projekt. Fönstret Solution Explorer innehåller projekt och filer.    Figur 4. |
|  | Filen Program.cs innehåller klassen Program med metoden Main och därmed är minimikraven uppfyllda för att du nu ska kunna köra applikationen.  För att köra applikationen välj menykommandot Debug ► Start Without Debugging, eller tryck ner tangentbordkombinationen Ctrl + F5.    Figur 5.  **Vad händer med källkoden i Program.cs?**  Källkoden i Program.cs kompileras till IL-kod (”*Intermediate Language*”) som placeras i en ny fil, Program.exe, en ”*assembly*”. Vid exekvering av applikationen ansvarar en virtuell maskin kallad CLR (”*Common Language Runtime*”), en central del av dotnetramverket (*.NET Framework*), för att IL-koden i ”*assemblyn*” ”*just-in-time*”-kompileras till maskinkod som sedan exekveras. |
|  | Applikationen körs och ett tomt konsolfönster, förutom texten Press any key to continue som Visual Studio lagt dit, visas. Stäng konsolfönstret genom att trycka på vilken tangent som helst på tangentbordet.    Figur 6. |
|  | Applikationen ska simulera tärningskast. Ett tal mellan 1 och 6 kan representera ett tärningskast. Resultatet av ett tärningskast måste sparas i en variabel vars värde sedan presenteras i konsolfönstret.   1. Börja med att radera satsen på rad 5 using System.Threading.Tasks; för att radnummer i anvisningar och bilder ska stämma! 2. Deklarera en variabel, med namnet roll, av typen int och initiera den till värdet 4 (*stasen på rad 12 i Figur 7*). 3. Presentera variabelns värde (*satsen på rad 13 i Figur 7*).     Figur 7. Deklaration och initiering av variabeln roll vars värde skrivs ut. |
|  | Testa applikationen genom att välj menykommandot Debug ► Start Without Debugging, eller tryck ner tangentbordkombinationen Ctrl + F5. (Gjorda förändringar sparas automatiskt i samband med att ett projekt exekveras).    Figur 8. Värdet 4 skrivs ut eftersom variabeln roll har det värdet. |
|  | Resultatet av ett tärningskast slumpas inte på något vis av applikationen utan är vad som kallas ”hårkodat” till värdet 4. För att slumpa ett värde i det slutna intervallet mellan 1 och 6 måste ett objekt av typen Random användas.   1. Istället för att tilldela variabeln roll värdet 4 ska variabeln tilldelas värdet 0 (*satsen på rad 12 i Figur 9*). 2. Skapa och initiera en referensvariabel, med namnet die (som betyder tärning på engelska), av typen Random att referera till ett objekt av typen Random (*satsen på rad 13 i Figur 9*). 3. Tilldela roll värdet som metoden Next() returnerar (*satsen på* *rad 15 i Figur 9*).     Figur 9. Metodanropet die.Next(1,7) returnerar ett tal i det slutna intervallet mellan 1 och 6. |
|  | Testa applikationen flera gånger och konstatera värden mellan 1 och 6 skrivs ut i konsolfönstret.    Figur 10. |
|  | Applikationen kan nu simulera ett tärningskast. Kravet är att mellan 100 och 1000 tärningskast ska kunna simuleras. Hur ska applikationen göra för att simulera t.ex. 273 tärningskast?  Det kan enklast göras genom att låta en ”for”-sats omsluta raderna 15 och 17 i Figur 9. Satserna på raderna 17 och 19 i Figur 11 kommer att upprepas 273 gånger.    Figur 11. En ”for”-sats som används för att upprepa de omslutna satserna 273 gånger. |
|  | Antalet tärningskast är nu ”hårdkodat” till 273. Användaren av applikationen ska kunna ange ett heltal i det slutna intervallet mellan 100 och 1000.   1. Deklarera och initiera variabeln count till 0 (*satsen på rad 12 i Figur 12*). 2. Skriv ut en ledtext (*satsen på rad 16 i Figur 12*). 3. Läs in heltalet användaren matat in och lagra värdet i count (*satsen på rad 17 i Figur 12*). 4. Ersätt 273 med count i ”for”-satsens villkorsuttryck (*uttrycket på rad 19 i Figur 12*).     Figur 12. Variabeln count tilldelas Inläst heltal av typen int. |
|  | Hur många gånger har du hittills sparat de ändringar du gjort? Inte någon gång? Huuu! Hög tid att klicka på knappen Save All  som sparar alla öppna filer som har osparade ändringar. Ta för vana att spara ofta, t.ex. efter varje sats. Ctrl + S, som sparar filen som har fokus, ska sitta i ryggmärgen. |
|  | Efter att ett antal förändringar är gjorda är det lämpligt att testa applikationen igen.    Figur 13. Resultatet av varje tärningskast skriv ut och applikationen fungerar, så långt den är skriven, som förväntat. |
|  | Applikationen kan nu simulera av användaren angivet antal tärningskast. Men hur ska en frekvenstabell över tärningskasten kunna skrivas ut? För att kunna skriva ut en frekvenstabell måste applikationen räkna antalet framslumpade ettor, tvåor, o.s.v. En träning har sex sidor varför en variabel med plats för sex värden behövs, en så kallad array.   1. Ersätt satsen som deklarerar variabeln roll med en sats som skapar referensvariabeln frequencyTable, av typen int[], och tilldela den en referens till en array med sex element (*satsen på rad 13 i Figur 14*). 2. En array har 0-baserat index, d.v.s. det första elementet har index 0, varför tal i det slutna intervallet mellan 0 och 5 nu måste slumpas fram. Satsen som tilldelar roll ett slumpat värde ska ersättas med en sats där det framslumpade värdet leder till att motsvarande elements värde i arrayen ökas med 1 (*satsen på rad 21 i Figur 14*). 3. En ”foreach”-loop är lämplig att använda för att skriva ut elementens värden i arrayen (*satserna på rad 24-27 i Figur 14*).     Figur 14. |
|  | Testa applikationen och konstatera att något som kan vara en frekvenstabell skrivs ut i konsolfönstret.    Figur 15.  Applikationen uppfyller nu de löst ställda kraven och en frekvenstabell över ett antal tärningskast skrivs ut. Men hur är det med kvalitéten på användargränssnittet?   1. Användaren uppmanas att mata in ett heltal i det slutna intervallet mellan 100 och 1000. Men vad händer om användaren matar in något som inte kan tolkas som ett heltal? Vad händer om användaren matar in t.ex. 13? Ska det gå? 2. Presentationen av frekvenstabellen har en del övrigt att önska. Vad betyder 129? Vad betyder 163?   Allvarligaste problemet är första punkten ovan eftersom om användaren matar in något som inte kan tolkas som ett heltal så kraschar applikationen.    Figur 16. Applikationen kraschar då användaren matar in strängen tvåhundra istället för heltalet 200. |
|  | Eventuella fel i samband med inläsning av antalet tärningskast som ska göras måste hanteras.  Satsen som sköter inläsningen måste ersättas av ”do-while”-satsen, som får omsluta satsen som skriver ut ledtexten, på raderna 16-19 i Figur 17.    Figur 17. |
|  | Testa applikationen och konstatera att applikationen inte längre kraschar. Det går fortfarande att mata in ett tal som inte är i det slutna intervallet mellan 1 och 100!    Figur 18. Resultat då ett värde mindre än 100 matats in. |
|  | I samband med inläsningen måste validering ske av värdet så att det ligger i det slutna intervallet mellan 100 och 1000.  Villkorsuttrycket i ”do-while”-satsen måste kompletteras så att det undersöker om det inlästa värdet är större eller lika med 100 och om det är mindre eller lika med 1000 (*uttrycket på raderna 19-21 i Figur 19*).    Figur 19. |
|  | Vid test av applikationen ska det nu vara omöjligt att mata in något annat än ett heltal i det slutna intervallet mellan 100 och 1000.    Figur 20.  Punkt 16 i) är nu åtgärdad. Återstår att åtgärda gör punkt 16 ii), d.v.s. att låta applikationen presentera frekvenstabellen på ett något bättre sätt så att t.ex. tärningssidan framgår:  Ettor: 107 Tvåor: 114 … Sexor: 113 |
|  | Applikationen måste känna till vad tärningssidorna kallas. Lämpligt är då att låta en array innehålla de strängar som beskriver tärningens sidor. Då frekvenstabellen sedan presenteras kan respektive sträng skrivas ut tillsammans med resultatet.[[1]](#footnote-1)   1. Lägg till en sats som skapar referensvariabeln facets, av typen string[], och initiera den med en array innehållande de sex strängar som var och en beskriver tärningens sidor (*satsen på rad 14 i Figur 21*). 2. Ersätt ”foreach”-satsen med en ”for”-sats som skriver ut varje tärningssida med tillhörande värde i frekvenstabellen (*satserna på rad 29-32 i Figur 21*).     Figur 21. |
|  | Testa applikationen på nytt. Nu ska det tydligare framgå hur många ettor, tvåor, o.s.v. som har slumpats fram. Lägg märke till att tabellen är något svår att läsa då en-, tio- och hundratal inte står under varandra.    Figur 22. Frekvenstabellen är dåligt formaterad vilket gör den svår att läsa. |
|  | Värden (strängar, heltal, etc.) som skrivs ut med hjälp av en formatspecificerare kan justeras på olika sätt. Den förändrade satsen på rad 31 i Figur 23 ser till att tärningssidan vänsterjusteras med en fältbredd på sex tecken, och resultatet högerjusteras med en fältbredd på fyra tecken.    Figur 23. |
|  | En test av applikationen visar att den nu beträffande användargränssnittet är acceptabel, men har kanske en hel del övrigt att önska, bl.a. saknas felmeddelande vid felaktig inmatning.    Figur 24. Frekvenstabellen bättre formaterad vilket underlättar läsningen av den.  Applikationen uppfyller nu det löst ställda kraven bättre beträffande användargränssnittet. Men hur är det med kvalitéten beträffande koden?   1. All kod är skriven i en och samma metod. Ofta vinner även mindre applikationer på att omstruktureras i flera metoder. Det blir då lättar att förstå och därmed underhålla koden. 2. Avsaknaden av kommentarer är total vilket försvårar förståelse av koden. |
|  | Placera koden som har hand om inläsningen av antal tärningskast som ska simuleras i en separat metod med namnet ReadNumberOfRolls() enligt satserna på raderna 30-42 i Figur 25. Metoden anropas enligt satsen på rad 17 i Figur 25 och värdet metoden returnerar lagras i den lokala variabeln count.    Figur 25. |
|  | Även skapandet av frekvenstabellen och presentationen av den kan omstruktureras i två separata metoder enligt Figur 26.  Trots att koden fortfarande inte innehåller några kommentarer är den lättare att förstå då metodernas namn har valts med omsorg och beskriver väl vad respektive metod har till uppgift.    Figur 26. |
|  | Då användaren råkar mata in något som inte kan tolkas som ett heltal i det slutna intervallet mellan 100 och 1000 visas inget felmeddelande. Metoden ReadNumberOfRolls() måste skrivas om så att så sker (*satserna på raderna 39-48 och 50-58 i Figur 27*).    Figur 27. |
|  | Testa applikationen genom att mata in några felaktiga värden.    Figur 28. Felmeddelande visas då användaren matar in något som inte kan tolkas som ett heltal i det slutna intervallet mellan 100 och 1000. |
|  | Presentationen av frekvenstabellen behöver åtgärdas så den liknar en tabell (*satserna på raderna 92-97 och 100-101 i Figur 29*).    Figur 29. |
|  | Testa applikationen och verifiera att frekvenstabellen nu påminner mer om en tabell än tidigare.    Figur 30. Frekvenstabellen presenterad på mer tilltalande sätt. |
|  | Kod ska alltid dokumenteras. Figur 31 visar hur källkoden blir enklare att förstår när den kompletterats med kommentarer.        Figur 31. |

1. Det är inte optimalt med en lösning som kräver synkronisering av två olika arrayer. En bättre lösning är att använda en associativ array (”*dictionary*”), men den konstruktionen får vänta till den avslutande delen av kursen. [↑](#footnote-ref-1)