2 INNEHÅLLSFÖRTECKNING

FÖRLAGETS FÖRORD 1

1 Författarens förord 2

2 Innehållsförteckning 3

3 Ämnesplan och kursplan 11

Ämnesplan – Programmering 11

Ämnets syfte 11

Undervisningen i ämnet programmering ska ge eleverna förutsättningar att utveckla följande: 12

Kurser i ämnet 12

Kursplan – Programmering 1 13

Centralt innehåll 13

Kunskapskrav 13

4 Inledning 17

4.1 Att använda detta läromedel 17

4.1.1 Till eleven 17

4.1.2 Till lärare 17

4.1.3 Självstudier 18

4.1.4 Vi, du och jag 18

4.2 Kapitelöversikt 18

4.3 Steg i inlärning av programmering 23

5 Bakgrund 25

5.1 Vad är programmering? 26

5.1.1 Sekventiell, objektorienterad och händelsebaserad programmering 26

5.1.2 Algoritmer 26

5.2 Datorer och programkörning 27

5.2.1 Operativsystemets roll 28

5.2.2 Plattformsoberoende 28

5.3 Programmeringens historik 28

5.3.1 Charles Babagge, Ada Lovelace och den analytiska maskinen 29

5.3.2 Turingkomplett 30

5.3.3 Elektriska datorer ... 30
5.3.4 Assembler ... 31
5.3.5 C och C++ ... 31

6 Kom igång med C++ ... 33
6.1 Att välja utvecklingsmiljö ... 34
6.2 Skriv ditt första program (Hello World)! ... 34
6.2.1 Binärkoden efter kompilering ... 35
6.3 Rader i C++ ... 36
6.4 Vad innebär den kod vi just har skrivit? ... 36
6.5 Vad händer vid kompilering? ... 37
6.5.1 Kompilering till binära filer ... 37
6.5.2 Skriptspråk ... 37
6.6 Pre-processorn ... 37
6.7 Kommentarer ... 38

7 Grunderna i programmering ... 39
7.1 Variabler ... 40
7.1.1 Vad en variabel är för något ... 40
7.1.2 Att deklarera (skapa) en variabel ... 40
7.1.3 Tilldelningsoperatorn = ... 41
7.1.4 Att använda en variabel ... 43
7.1.5 Variabeltyper ... 43
7.1.6 Namngivning av variabler ... 47
7.2 Operatorer ... 47
7.2.1 Jämförelseoperatorer ... 48
7.2.2 Logiska operatorer ... 48
7.2.3 Räkneoperatorer ... 48
7.3 Villkorssatsen if ... 49
7.3.1 Jämförelseoperatorn lika med == ... 49
7.3.2 if ... 49
7.3.3 else ... 50
7.3.4 else if ... 51
7.3.5 If-satser med mindre än-operatorn < ... 52
7.3.6 If-satser med och-operatorn && ... 53
7.3.7 If-satser med eller-operatorn || ... 53
7.3.8 Nästlade if-satser ... 54

7.4 Switch-satsen 55
7.5 Loopar 56
7.5.1 While-loopen 56
7.5.2 do while 60
7.6 Kodblock 61
7.6.1 Kodblock utan klammerparentes 62
7.6.2 Variablers livslängd 62
7.7 Kodformatering 63

8 Pseudokod och aktivitetsdiagram 65
8.1 Pseudokod 66
8.1.1 Inga standarder för pseudokod 66
8.1.2 Att låna färdiga lösningar 66
8.2 Aktivitetsdiagram 67
8.2.2 Tre exempel på aktivitetsdiagram 67

9 Att hantera fel i c++ 71
9.1 Att felsöka källkod 72
9.1.1 Vanliga felmeddelanden 72
9.2 Fel under körtid 73
9.2.1 Att hitta fel med debuggers 73
9.2.2 Undantag i C++ 74

10 Att finna hjälp 75
10.1 Att dela upp problemet i mindre delar 76
10.2 cplusplus.com 76
10.2.1 Att navigera på cplusplus.com 76
10.2.2 Att söka på cplusplus.com 77
10.3 Andra källor – artiklar och bloggar 77
10.4 Att ställa frågan i ett diskussionsforum 77

11 Introduktion till pekare 79
11.1 Vad en pekare är för något 80
11.2 Att skapa pekare 81
11.3 Pekare ligger som heltal i minnet 82

12 Funktioner 83

12.1 Vad en funktion är för något 84

12.1.1 Exempel på ett anrop av en funktion 84

12.1.2 När ska vi skapa en funktion och hur ska den kodas på bästa sätt? 85

12.2 Indata och utdata 85

12.2.1 Fyra typer av funktioner, gällande in- och utdata 86

12.2.2 Regler för in- och utdata 86

12.2.3 Argument (parameter) och argumentlista 86

12.3 Returnering 86

12.4 Funktionsdefinition och funktionsanrop 86

12.4.1 Funktionsprototyp 87

12.5 Exempelfunktioner 88

12.5.1 Exempel på en funktion med indata – Hello() 88

12.5.2 Exempel på en funktion med endast utdata 89

12.5.3 Exempel på en funktion med både in- och utdata, CelsiusToFahrenheit() 90

12.5.4 Exempel på en funktion med flera indata, Pythagoras() 91

12.6 Variablers livslängd i funktioner 93

12.7 Referenser och pekare med metoder 94

12.8 Funktioner med utdatan bool 96

12.9 main()-funktionen 97

12.9.1 Utdata i main() 98

12.9.2 Indata i main() 98

12.10 Överlagrade funktioner 99

12.11 Filer för att strukturera funktioner 101

13 Vektorer 103

13.1 Vad en vektor är för något 104

13.1.1 indexering av vektorer 104

13.1.2 Förenklat sätt att skapa vektorer 105

13.1.3 Multidimensionella vektorer 105

13.2 For-loopen 106

13.2.1 Variabeln i 107

13.2.2 Att använda for med vektorer 107

13.2.3 Att loopa igenom en multidimensionell vektor 108

13.3 Dynamiska vektorer 109

13.4 Vektorer som argument i funktioner 110

14 Introduktion till objektorientering 113

14.1 Vad en klass är för något 114

14.2 Att använda en klass 114

14.2.1 Att skapa en klass 114

14.2.2 Klassmedlemmar 114

14.2.3 Att skapa ett objekt av en klass 115

14.2.4 Att använda ett objekt 115

14.3 Vikten av att särskilja klass och objekt (eller: ät kakan, inte receptet!) 115

14.3.1 Namngivning 116

14.4 Metoder som klassmedlemmar 116

14.5 Klassmedlemmars synlighet 117

14.5.2 Att använda public och private 118

14.6 Att dela upp metoder i prototyp och definition 118

14.7 Konstruktor 119

14.7.1 Överlagring av konstruktorer 119

14.8 Statiska metoder i klasser 120

14.9 Klasser och objekt med pekare 121

15 Sökning, sortering och rekursion 123

15.1 Sökalgoritmer 124

15.1.1 Linjär sökning 124

15.1.2 Binär sökning 127

15.2 Sorteringsalgoritmer 131

15.2.1 Bubblesort 131

15.2.2 Selection sort 135

15.3 Rekursiva algoritmer 136

15.3.1 Fibonaccitalen 136

16 Grafiska användargränssnitt 138

16.1 Händelsebaserad programmering 139

16.2 Grafiska användargränssnitt i C++ 140

16.2.1 GUI med Embarcadero C++ Builder 140

16.2.2 GUI med Microsoft Visual C++ 141

16.2.3 GUI med Win32 API 142

16.2.4 GUI med GTK+ och Glade 142

16.2.5 GUI med QT Creator 143

16.2.6 GUI med spelbiblioteket Allegro 144

17 Källor 145

Bilaga 1 – Begrepp på engelska och svenska 147

Svenska – Engelska 147

Engelska – Svenska 148

Bilaga 2 – Det binära talsystemet 149

Bilaga 3 – Komma igång med konsolprogram 151

Att komma igång med Code::Blocks i Ubuntu Linux 151

Att komma igång med Code::Blocks i Windows 154

Att komma igång Embarcadero C++ Builder XE2 157

Att komma igång med Microsoft Visual C++ Express 2010 159

Bilaga 4 – GUI med Embarcadero C++ Builder 163

Att lägga till kontroller (controls) i en form 163

Kodläge och designläge 165

Analys av den kod som ligger där från början 167

Egenskaper för former och kontroller – rutan properties 168

Former och kontroller blir till 169

Kontrollernas metoder – händelsebaserad programmering 169

Att lägga till fler events 169

Exempel på program – en kalkylator 170

Bilaga 5 – GUI med Visual C++ 179

.NET och CLI 179

Visual Studio .NET 179

Just In Time-kompilering 179

C++ med CLI 180

Att skapa ett Windowsprogram i Visual C++ 180

Kodläge och designläge 182

Att lägga till kontroller (controls) i en form 186

Egenskaper för former och kontroller – rutan properties 187

Former och kontroller blir till 189

Kontrollernas metoder – händelsebaserad programmering 189

Att lägga till fler events *189*

Argumentlista för kontrollernas metoder *190*

Exempel på program – en kalkylator 191

18 Index- och sökregister 201

6 KOM IGÅNG MED C++

I detta kapitel kommer vi att skapa ett program som skriver ut text på skärmen, samt lite ytligt titta på vad ett sånt program innebär. Vidare kommer vi att gå igenom vad som händer när ett program går från att vara källkod till att bli ett faktiskt fungerande program, som man kan köra.

Kursmål och mål i ämnesplanen som helt eller delvis behandlas i detta kapitel

Detta kapitel syftar inte mot något specifikt kursmål men finns här som en hjälp för att komma igång.

6.1 Att välja utvecklingsmiljö

C++ är ett språk som finns för de flesta operativsystem. Det finns många olika utvecklingsmiljöer i vilka man programmerar C++. En utvecklingsmiljö innehåller oftast en textredigerare som färgar koden, en kompilator och andra verktyg som hjälper programmeraren.

Valet av utvecklingsmiljö brukar ofta falla på smak och tycke. Om man jobbar på ett företag är det troligt att företaget använder sig av en viss utvecklingsmiljö. Valet kan också bero på vad man ska programmera och för vilket operativsystem.

Om man ska programmera Windowsapplikationer (alltså vanliga fönsterprogram för Windows) brukar valet ofta falla på Microsofts Visual Studio. I kapitel 16 när vi lär oss om grafiska användargränssnitt, kommer vi att använda Microsoft Visual C++. Fram till dess bör det dock fungera med de flesta utvecklingsmiljöer för C++. Om du ska utveckla grafiska program för en annan miljö än Windows, så kan du gott lära dig programmering fram till det kapitel och därefter prova dig fram själv. Eller få hjälp av lärare om skolan har valt en annan miljö.

I öppen källkods-världen skiljer man ofta på utvecklingsmiljö och kompilator. Här är en lista på intressanta utvecklingsmiljöer och kompilatorer, sök på nätet om dem:

- GCC – Gnu C Compiler (innehåller också en C++-kompilator)
- MinGW – GCC för Windows
- Code::Blocks – utvecklingsmiljö
- Eclipse – ursprungligen en utvecklingsmiljö för Java, men det finns tillägg för C++
- Embarcadero C++ Builder – utvecklingsmiljö för Windowsprogram
- Microsoft Visual Studio .NET / Visual C++ - utvecklingsmiljö för Windowsprogram (finns att ladda ner i expressversion utan kostnad)

6.2 Skriv ditt första program (Hello World)!

Det är nu dags att vi skapar vårt första program. Vårt mål är att skriva ut texten ”Hello World” på skärmen (i vårt konsolfönster). Att vi väljer att skriva ut just Hello World är av historiska anledningar. I programmeringsböcker och exempel har man i decennier valt att göra just detta program som första exempel – vi håller oss traditionen trogen!

Beroende på vad du använder dig av för utvecklingsmiljö, så skapar man C++-projekt på olika sätt. Vad vi vill göra är i alla fall att skapa ett konsolprojekt (kallas ofta console application). I bilaga 3 i denna bok så finns det guider för hur man kommer igång i med konsolprogrammering i följande utvecklingsmiljöer:

- Code::Blocks med Ubuntu Linux
- Code::Blocks med Windows
- Embarcadero C++ Builder XE2
- Microsoft Visual C++ Express 2010

När man väl har skapat ett konsolprojekt, är det vanligt att vi får en fil i vilken vi kan börja skriva kod. Filen innehåller ofta kod som utvecklingsmiljön har genererat automatiskt åt oss.

För att skapa ett program som skriver ut texten ”Hello World” på skärmen, så skriver vi följande kod:

Exempel 6.1

```
#include <iostream>
using namespace std;

int main()
{
	cout << "Hello World";
	return 0;
}
```

När vi har skrivit vårt program är det dags att starta det för en testrunda. I de flesta utvecklingsmiljöer finns en liten play-knapp som man kan trycka på. Vad som händer då är att programmet först kompileras och därefter startas. Ibland kan det dock vara så att ett konsolfönster kommer upp blixtsnabbt, men man hinner inte se vad som står. Vi kan då lägga till en rad för att användaren ska få mata in något innan programmet avslutas:

Exempel 6.2

```
#include <iostream>
using namespace std;

int main()
{
	cout << "Hello World";
	cin.get(); // För att användaren ska trycka tangent
	return 0;
}
```

Strålande! Nu har vi skapat vårt första program!

6.2.1 Binärkoden efter kompilering

Efter att man har kompilerat programmet så skapas alltså en exekverbar fil. Om man kör Windows skapas alltså en fil med filändelsen .exe. Om man kör t.ex. Linux (eller liknande system) så skapas en fil med exekverbar filrättighet.

Beroende på kompilator så hamnar dessa filer på ett bestämt ställe. Oftast ligger de någonstans i projektmappen.

6.3 Rader i C++

De flesta rader som utför en instruktion, avslutas i C++ med ett semikolon ”;”. Detta kan verka märkligt. Tanken bakom det är att det ska vara tydligt vad som är en rad och inte. Det ska också vara möjligt att ha två rader på en rad, så att säga. Alltså två instruktioner på en rad. Brukligt är dock att man har radbrytning efter semikolon.

Det finns också kod som inte har något semikolon. Detta är kod som innehåller annan kod (t.ex. klasser, metoder, villkorssatser, loopar etc.) Vi kommer att gå igenom vad allt detta betyder.

6.4 Vad innebär den kod vi just har skrivit?

Låt oss böja med att titta på koden för vårt Hello World-program från Exempel 6.2. Under den här bokens gång kommer vi att gå djupare in på många av de detaljer som krävs för att förstå denna kod. Jag tror dock att det kan kännas tryggare att programmera om man har en viss hum om vad det hela handlar om.

Det första vi ser är en rad där det står `#include`. Det är ett sätt för C++ att lägga in annan kod som vi vill använda oss av. I exemplet använder vi oss av två funktioner. En för att skriva ut text (`cout`) och en för att låta användaren trycka en tangent (`cin.get()`). Funktionaliteten för dessa ligger i `iostream` som faktiskt länkar till några andra C++-filer. Någon annan har i de filerna skrivit C++-kod för det vi behöver. Det betyder att vi behöver inte bry oss om hur man får ut text på skärmen, vi kan bara använda `cout`!

Under raden med `#include` så finner vi en rad där det står `using namespace std;`. Namespace är ett sätt att organisera kod och påminner lite om klasser (vilket vi ska gå igenom i kapitel 14). `cout` ligger sorterad under namespacet `std`. Om vi inte skulle skriva `using namespace std;` så skulle vi istället behöva skriva `std::cout << ”Hello World”;` respektive `std::cin.get();`. Det ser ju en aning bökigare ut och tar mer plats (speciellt när man gör större program). I denna bok kommer vi inte närmre att arbeta med namespace, då det ligger utom ramarna för Programmering 1.

Efter namespace så finner vi raden `int main()` är vad som kallas för `main()`-funktionen. `main()`-funktionen är det som först laddas när man startar ett program och det är här vår kod börjar. Funktioner är ett sätt att strukturera kod, vilket vi kommer gå närmre in på i kapitel 12, till dess ska all kod vi skriver ligga i `main()`-metoden.

Du ser också att koden är indelad i klammerparenteser `{` och `}`. Det som ligger inom klammerparenteserna kallas kodblock. Ett kodblock kan innehålla flera kodblock. De ligger i en hierarkisk ordning där det som ligger innanför något är en del av det som ligger utanför. Mer om detta i kapitel 7.6.

Inne i själva `main()`-metoden så finner vi instruktioner för vårt program. Skriv ut text och låt användaren trycka en tangent. Det är inte så mycket, men lagom för ett första program.

Den sista raden i `main()`-funktionen, `return 0;` är ett sätt för `main()` att kommunicera med operativsystemet. Vi skickar ut siffran 0 och om operativsystemet vill så kan det göra något med detta. Man kan skicka ut andra siffror också, som kan t.ex. indikera olika fel som har uppstått under programmet. Vi kommer att lära oss mer om funktioner i kapitel 12.

6.5 Vad händer vid kompilering?

I kapitel 5.2 så pratade vi om att en processor endast förstår binär kod. Exakt hur den binära koden ska se ut, skiljer sig ofta från processor till processor. C++ är ett människospråk och koden som vi skriver måste därför översättas till binär kod. Denna översättning kallas kompilering.

6.5.1 Kompilering till binära filer

Vad som händer när vi kompilerar Hello World, är att vår källkods-fil, samt annan källkod (från biblioteket, som iostream och tillhörande kod) översätts till binär kod och blir en exekverbar fil. Vi kan via en filhanterare leta reda på vår exe-fil och den går att starta precis som vanligt.

Figur 6.1 – Kompilering från C++-kod till binärkod.

6.5.2 Skriptspråk

Alla programmeringsspråk kompileras inte till binärkod, åtminstone inte på detta sätt. Dessa kallas ofta skriptspråk och används i webbläsare, spel, mobiltelefoner men också som vanliga program. Ett vanligt och känt skriptspråk är JavaScript (som ej ska förväxlas med Java). Dessa språk kompileras inte utan är hela tiden möjliga att läsa för en människa. Endast, direkt då de ska användas så översätts de till binärkod (oftast rad för rad, samtidigt som det körs). Att översätta till binärkod är en krävande process, vilket innebär att för-kompilerade program är snabbare än program skrivna med skriptspråk. En fördel med skriptspråk är att de i princip kan köras på vilken dator som helst, med vilket operativsystem som helst – så länge det finns ett program som översätter skriptet.

6.6 Pre-processorn

I C++ finns det något som kallas för preprocessorn. Preprocessorn letar efter instruktioner i koden, som den ska utföra innan koden kompileras till binärkod. Sådan kod anges med tecknet # innan. Som t.ex. i `#include`. Vad preprocessorn gör, när den stöter på en `#include` är att den letar reda på andra filer (som i sin tur kan ha egna `#include` o.s.v.) och lägger till dem för kompilering.

6.7 Kommentarer

Som programmerare är det otroligt viktigt att man kommenterar sin kod. Kommentarer är text i koden som inte tolkas av kompilatorn. Vi kan alltså i en kommentar skriva precis vad vi vill, även på svenska.

Att kommentera är en konst i sig och det är inte alltid självklart när och för vad man ska kommentera. Efter en stund, när man lärt sig att programmera, så känns den kod man skriver just nu ganska självklar. Varför ska man då kommentera den? Men efter att man har kommit vidare, och sedan ska hitta tillbaks i koden – så kan det vara svårt att veta vad man har gjort och var. Många programmerare slarvar med att kommentera och lika många programmerare har slitit sitt hår, då de inte hittat i sin egen kod.

Det är onödigt att skriva en kommentar för varje rad, men en tumregel är att skriva en kommentar för varje stycke kod, som på något vis hänger ihop. För att lära sig är det dock bättre att skriva för många kommentarer än att skriva för få.

Kommentarer anges med hjälp av två snedstreck (engelska slash) `//`. Allt som kommer efter dessa tecken är kommentarer:

Exempel 6.3

```
cout << "Hello World"; // Skriv ut text på skärmen
```

7 GRUNDERNA I PROGRAMMERING

Detta kapitel är bokens största kapitel och kanske det viktigaste. Vi kommer här att gå igenom grunderna för sekventiell programmering. Det vi går igenom kommer att ge dig en viktig grundinsikt i hur man kodar i C++.

Faktum är att det du efter att ha läst detta kapitel, kan skapa program som kan göra vilka matematiska beräkningar som helst. Med andra ord kan man säga att det du lär dig i detta kapitel, är vad som behövs för att ett programmeringsspråk ska vara turingkomplett (se kapitel 5.3)!

Det låter kanske konstigt, men i princip alla efterföljande kapitel (funktioner och objektorientering, t.ex.) är till för att göra livet enklare för programmeraren.

Det vi i huvudsak kommer att gå igenom i detta kapitel är:

- variabler
- villkorssatser
- loopar

Vi kommer först att gå igenom objekt och variabler, som används för att lagra data i minnet. Därefter kommer vi att gå igenom villkorssatser, de används för att ge olika utfall, t.ex. beroende på vad användaren har matat in. Därefter går vi igenom loopar, ett sätt att köra samma kodstycke om och om igen.

Vidare går vi igenom något som kallas operatorer och tittar på hur kodblock fungerar i C++. Var lugn, allt kommer att förklaras i detalj!

Kursmål och mål i ämnesplanen som helt eller delvis behandlas i detta kapitel

Kursmål

- Programmeringsspråkets eller -språkens grundläggande datatyper samt fördefinierade strukturer, regler och syntax.
- Kontrollstrukturer, till exempel sekvens, selektion och iteration [...]
- Variablers och konstanters synlighet och livslängd.

Mål i ämnesplanen

- Förståelse av och färdigheter i att använda datalogiska begrepp och algoritmer.

7.1 Variabler

När man programmerar i C++ är det i princip omöjligt att inte använda sig av något som kallas variabler. Variabler används för att lagra data i minnet.

7.1.1 Vad en variabel är för något

Du känner kanske redan till begreppet variabler? Det återkommer bland annat i matematiken. Då talar man ofta om okända variabler som X. Variabler i C++ används lite annorlunda.

En variabel ses kanske enklast som en låda, med en etikett på. På etiketten står det ett namn och i lådan ligger det något.

Figur 7.1 – Variabler kan ses som lådor med etikett och innehåll.

I C++ är det också så att alla lådor är av olika typer. Man lägger t.ex. ner bananer i bananlådor och äpplen i äppellådor. Att försöka lägga en banan i en låda för äpplen går inte. Kompilatorn skulle säga ifrån.

I C++ finns det många olika typer av variabler. I denna bok kommer vi främst att arbeta med heltal och textsträngar av typen char (egentligen char-arrayer, som vi ska se sen) men det finns också något som heter objekt som man skapar av klasser. Objekt kan hantera mer avancerade saker som bilder, hålla reda på vilken knapp på spelkonsolen som nyss blev nedtryckt och mycket annat. Vi kommer att gå djupare in på objekt i kapitel 14.

7.1.2 Att deklarera (skapa) en variabel

När man deklarerar (skapar) en variabel anger man först vilken typ variabeln ska vara av, därefter namnet. Om man t.ex. vill göra en heltalsvariabel med namnet `nr` så skriver man:

Exempel 7.1

```
int nr;
```

Ordet `int` är en förkortning av *integer* (engelska för ordet heltal).

7.5 Loopar

En loop, även kallad slinga, är ett kodstycke som upprepas (om och om igen). Oftast upprepas kodstycket så länge ett visst villkor är sant, precis som i en `if`-sats. Ibland kan man vilja upprepa något om och om igen, tills användaren stänger programmet. Ibland vill man upprepa något ett visst antal gånger, om man t.ex. vill gå igenom en lista med femtio saker i så loopar man ett kodstycke femtio gånger.

I C++ finns det flera typer av loopar med olika syften. Faktum är dock att man egentligen behöver kunna endast en loop och det är `while`-loopen. Alla de andra looparna är till för att förenkla för programmeraren, men man skulle lika gärna kunna använda `while`. Vi kommer att gå igenom andra loopar längre fram i boken. Vi börjar med `while`-loopen.

7.5.1 While-loopen

Som sagt är `while`-loopen den enda loop du egentligen behöver kunna. Man kan beskriva `while`-loopen i pseudokod så här:

MEDAN någonting SÅ
Gör detta
SLUT MEDAN

Vi kan t.ex. be någon ange temperatur. Så länge temperaturen är mindre än 100, så kör vi ett varv i loopen. För varje varv i loopen, så ökar vi temperaturen med en grad och skriver ut den. När temperaturen är 100 så går vi vidare i programmet och skriver ut ett meddelande:

MEDAN temperatur är mindre än 100 SÅ
Plussa på temperaturen med ett och skriv ut den
Skriv ut ”vattnet kokar”
SLUT MEDAN

I C++ blir det:

Exempel 7.29

```
cout << "Ange temperatur: ";
int temperature;
cin >> temperature;
while (temperature < 100)
{
    temperature++; // ++ är en räkneoperator som ökar värdet med 1
    cout << "Temperaturen är nu " << temperature << endl;
}
cout << "Vattnet kokar!";
```

Observera att vi la in en `endl` i utskriften av temperatur. `endl` står för *new line*, och ger oss en radbrytning.

Om vi anger den nuvarande temperaturen som 92 får vi följande resultat:

```
Ange temperatur: 92
Temperaturen är nu 92
Temperaturen är nu 93
Temperaturen är nu 94
Temperaturen är nu 95
Temperaturen är nu 96
Temperaturen är nu 97
Temperaturen är nu 98
Temperaturen är nu 99
Temperaturen är nu 100
Vattnet kokar!
```

Om vi istället anger temperaturen 100 (eller mer) så kommer kodstycket i loopen aldrig att köras. Programmet hoppar då direkt till ”Vattnet kokar!”-meddelandet och vi får följande resultat:

```
Ange temperatur:100
Vattnet kokar!
```

Gissa talet!

Låt oss tillverka ett enkelt litet spel. Spelaren ska få gissa på ett tal mellan 1 och 100, ett tal som vi programmerare redan bestämt innan. Talet är 42. Så länge spelaren gissar fel så ska den få fortsätta att gissa. När spelaren har gissat rätt skriver vi ut ett grattis-meddelande. Vi tar det först i pseudokod:

Skriv ut ”Gissa ett tal mellan 1 och 100”
Mata in tal
MEDAN talet inte är 42 SÅ
  Skriv ut ”Fel. Gissa igen”
  Mata in tal
SLUT MEDAN
Skriv ut ”Grattis! Du gissade rätt!”

Kodat i C++ blir det:

Exempel 7.30

```
cout << "Gissa ett tal mellan 1 och 100:";
int tal;
cin >> tal;
while (tal != 42)
{
    cout << "Fel! Gissa igen: ";
    cin >> tal;
}
cout << "Grattis! Du gissade rätt!";
```

Här har min kompis försökt spela spelet:

```
Gissa ett tal mellan 1 och 100:
50
Fel! Gissa igen:
25
Fel! Gissa igen:
37
Fel! Gissa igen:
44
Fel! Gissa igen:
41
Fel! Gissa igen:
42
Grattis! Du gissade rätt!
```

Det här spelet är inte så roligt att spela mer än en gång. I övningshäftet finns en uppgift som gör spelet lite roligare. Försök gärna med den uppgiften!

Evighetsloop och break

Ibland kan något gå snett och det uppstår en loop som loopar oändligt länge. Det kan leda till att programmet kraschar eller beter sig felaktigt. Ett sätt att skapa en sådan loop är att skriva följande kodstycke:

Exempel 7.31

```
while(true)
{
    cout << "loop...";
}
```

En `while`-loop fortsätter så länge det som står i parenteserna efter ordet `while` är sant. I detta fall skriver vi `true`, och `true` är ju sant och alltid sant.

Detta kan verka dumt och är det också om vi lämnar det så här. Men vi kan faktiskt använda detta till vettiga saker. T.ex. så kan man skapa en meny:

Exempel 7.32

```
// Skapa en loop som körs tills användaren trycker Q:
while (true)
{
    // Skriv ut menyn:
    cout << "Välj:\n";
    cout << "[S]pela spelet\n";
    cout << "[H]ighscore\n";
    cout << "[Q]uit\n";

    char menySelection; // Användarens val
    cin >> menySelection;

    if (menySelection == 'S' || menySelection == 's')
    {
        cout << "=== VÄLKOMMEN TILL SPELET! ===\n";
        // lägg in kod för spelet här
    }
    else if (menySelection == 'H' || menySelection == 'h')
    {
        cout << "=== HIGHSCORE ===\n";
        // lägg in kod för highscore här
    }
    else if (menySelection == 'Q' || menySelection == 'q')
    {
        break; // avbryter while-loopen
    }
    else
    {
        cout << "Ogiltligt val!\n";
    }
}
```

Vad vi gjorde var att skapa en meny som tack vare en `while`-loop, körs om och om igen, tills att användaren trycker på Q eller q. Vad som händer då är att loopen avbryts med ordet `break`. En `break` avslutar alltid ”aktuell” loop (om man har en loop inuti en annan loop är det alltså den innersta som avbryts).

Som du ser är menyn i princip färdig. Nu behöver vi bara skapa ett spel och en highscore-lista (det finns uppgifter i övningshäftet för just detta). Hade det funnits ett spel så hade spelet startats när man tryckte på S eller s. När man spelat klart hade man åter igen hamnat i menyn. (Man kan också se det som att man, då man startat själva spelet fortfarande är i menyn – man är ju faktiskt fortfarande i `while`-loopen.)

14 INTRODUKTION TILL OBJEKTORIENTERING

Det som gör att ett språk är objektorienterat, är att man arbetar med klasser och objekt av klasser. Vi har redan arbetat med vissa klassobjekt utan att gå in på det. T.ex. `cout` och `cin` är två objekt.

Ett objektorienterat programmeringssätt finns för att förenkla livet i programmerare. Precis som metoder så är det ett sätt att strukturera upp koden. När man skapar objekt av klasser, så påminner de till viss del om variabler.

Detta kapitel heter *introduktion* till objektorientering, just för att kursen *Programmering 1* behandlar grunderna i objektorienterad programmering. Vi kommer inte fördjupa oss i sådant som arv, polymorfism och andra avancerade aspekter av klasser. Det hör till kursen *Programmering 2*.

Kursmål och mål i ämnesplanen som helt eller delvis behandlas i detta kapitel

Kursmål

- Grunderna för klasser, objekt, egenskaper och metoder**.**
- Variablers och konstanters synlighet […].

Mål i ämnesplanen

- Kunskaper om objektorienterad programmering i teori och praktik.

14.1 Vad en klass är för något

En klass används för att klumpa ihop saker som hör ihop. Exempel där klasser är bra att använda, kan vara när man vill lagra information om personer. Det är smidigt att personernas namn, adress, telefonnummer etc. ligger samlade på ett och samma ställe. Eller kanske vill vi ha fiender i ett spel och vi vill veta deras position och hur mycket hälsa de har. Ett annat exempel är när man vill skapa en highscore-lista i ett spel. Gör man en klass av det kan man simpelt manipulera listan med metoder som lägger till, skriver ut, sparar på fil etc.

Det vi klumpar ihop är (en eller) flera objekt tillsammans med (en eller) flera funktioner. Funktionerna i klassen används ofta till att göra något med de medlemmar som finns i klassen. Av klassen skapar man sedan ett nytt objekt. När man skapar funktioner i klasser så kallas de metoder.

14.2 Att använda en klass

Låt oss ta ett exempel. Säg att vi vill arbeta med hundar. Det finns vissa saker vi vill veta om hunden. Vi vill veta dess namn, ålder och vilken ras den är. Detta kan vi klumpa ihop i en klass. Vi börjar med att skapa en klass.

14.2.1 Att skapa en klass

När vi konstruerar vår klass kan vi göra det ovanför vår `main()`-metod eller i en egen .cpp-fil med tillhörande .h-fil. Här är ett exempel på en klass som hanterar data om en hund:

Exempel 14.1

```
class Dog
{
public:
    char* name;
    int age;
    char* race;
};
```

14.2.2 Klassmedlemmar

Klassen i exemplet ovan har tre objekt i sig, `name`, `age` och `race`. De brukar kallas klassmedlemmar. Om man lägger in metoder i klasserna kallas dessa också för klassmedlemmar. Objekt i metoder (som endast finns inne i en viss metod, men inte utanför) räknas inte som klassmedlemmar.

Som du ser skrev vi också `public` innan allt, det har med synlighet att göra för klassmedlemmar. Vi kommer att diskutera detta i kapitel 14.5.

14.2.3 Att skapa ett objekt av en klass

Med hjälp av klassen från exempel Exempel 14.1 kan vi skapa en hund:

Exempel 14.2

```
Dog myDog;
```

14.2.4 Att använda ett objekt

När vi nu har skapat klassen kan vi börja använda den. Vi kan sätta olika värden på variablerna:

Exempel 14.3

```
myDog.name = "Fido";
myDog.age = 3;
myDog.race = "tax";
```

När vi nu har satt väldena på variablerna, kan vi börja använda dem:

Exempel 14.4

```
cout << "Hunden heter " << myDog.name << ", är en " << myDog.race
     << " och är " << myDog.age << " år gammal";
```

14.3 Vikten av att särskilja klass och objekt (eller: ät kakan, inte receptet!)

Precis som när vi arbetar med metoder och skiljer på metoddefinition och metodanrop, så är det viktigt att skilja på klass och objekt. `int` är som sagt en klass, utifrån vilken vi skapar ett objekt.

Vi skriver aldrig så här:

Exempel 14.5

```
int = 5; // FEL!
```

Utan vi anger alltid ett namn på variabeln, t.ex. `nr` och tilldelar det ett värde:

Exempel 14.6

```
int x;
x = 5;
```

Att göra på det första sättet, `int = 5` vore som att försöka äta receptet. Klassen är receptet och objektet är kakan. Vi måste först skapa en kaka, med hjälp av receptet. Sen kan vi börja använda kakan.

På samma sätt gör man aldrig så här:

Exempel 14.7

```
Dog = "Fido"; // FEL!
```

14.3.1 Namngivning

Särskiljningen mellan klass och objekt kan man tydliggöra för sig själv, som programmerare, genom att ge klasser och objekt vettiga namn. En tumregel är att första bokstaven i klasser alltid är en versal, alltså en stor bokstav. Den första bokstaven i ett objekt är alltid en gemen, alltså en liten bokstav. Detta gäller alla objekt och klasser.

14.4 Metoder som klassmedlemmar

Vi kan som sagt använda metoder i klasser. Till skillnad från vanliga funktioner så anropas metoder genom objektet. En metod i en klass gör ofta något med olika variabler (eller objekt) som är klassmedlemmar i samma klass som metoden. Till hundexemplet kan vi t.ex. skapa en metod som låter hunden skälla. Hur den skäller beror på dess ålder. Vi tar hela koden, från början till slut, för att vara säker på att vi får ordning på var allt ska ligga:

Exempel 14.8

```
#include <iostream>
using namespace std;

// KLASS: Dog, en hund som kan skälla
class Dog
{
public:
    char* name;
    int age;
    char* race;

    // METOD: Bark. Hunden skäller
    void Bark()
    {
        if(age > 1)
            cout << name << " skäller VOFF VOFF!";
        else
            cout << name << " är bara en liten valp och vågar inte
                             skälla på dig.";
    }
};
```

//Programmet fortsätter på nästa sida

```
// FUNKTION: Main, start på programmet
int main()
{
    Dog myDog;
    myDog.age = 3;
    myDog.name = "Fido";
    myDog.race = "Golden Retriever";

    cout << "Hunden heter " << myDog.name << ", är en "
         << myDog.race << " och är " << myDog.age << " år gammal"
         << endl;
    myDog.Bark();

    cin.get();
    return 0;
}
```

Vi får följande utskrift i konsolfönstret:

```
Hunden heter Fido, är en tax och är 3 år gammal
Fido skäller VOFF! VOFF!
```

Den metod vi skapade hade ingen utdata, `void` och ingen indata. Men det går så klart att göra metoder med både in- och utdata, på samma sätt som med funktioner.

14.5 Klassmedlemmars synlighet

Hittills har vi alltid skrivit `public` framför alla klassmedlemmar. Det är nu dags att gå igenom vad det är för något. Klassmedlemmar kan ha tre olika synligheter, `public`, `private` och `protected`. Detta kallas ibland på engelska för *protection level*.

Public

Om vi skriver `public` framför klassmedlemmar (vare sig det är ett objekt eller en metod), betyder det att den går att komma åt utifrån. Med andra ord kommer vi åt klassmedlemmen via de objekt vi skapar (genom att skriva namnet på objektet, följt av en punkt och därefter namnet på klassmedlemmen). Samma sak gäller statiska medlemmar, bara det att man skriver namnet på klassen direkt.

Private

Om vi inte skriver något räknas klassmedlemmen som `private`. Det går också att skriva `private` framför – båda har samma resultat. En klassmedlem som är private går inte att komma åt utifrån. Objekt som är `private` kan endast påverkas av metoder som är klassmedlemmar i samma klass som de privata objekten.