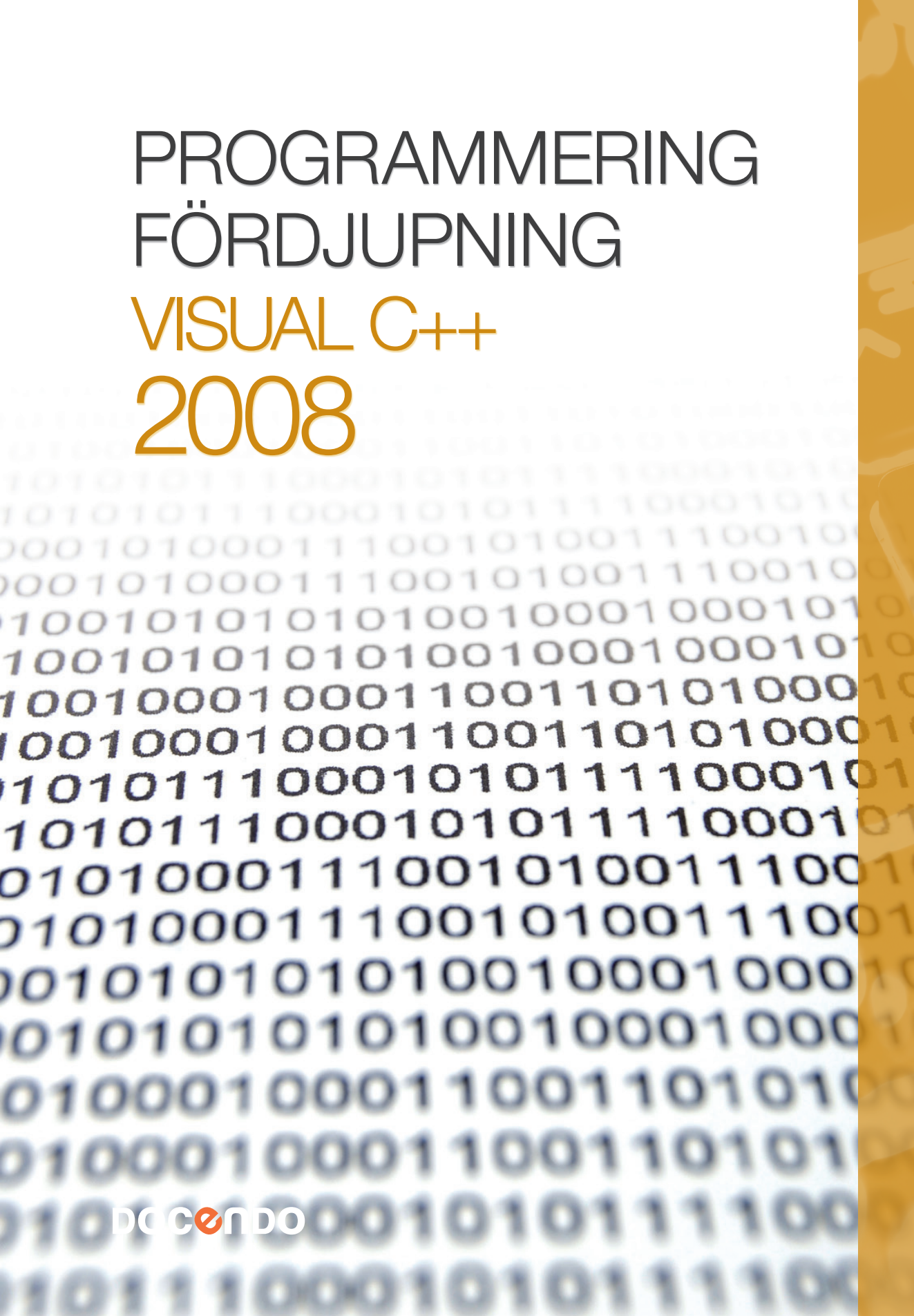
PROGRAMMERING FÖRDJUPNING

VISUAL C++ 2008



DOCENDO

Innehållsförteckning

3 Klasser

I detta kapitel

Du kommer nu att få kunskap om en del av det som kallas objektorientering. Objektorientering i dess egentliga mening kommer du inte att komma i kontakt med förrän i senare kurser (Programmering C). Förhoppningen är att om du redan nu bekantar dig med några av begreppen kommer du lättare att förstå fördelarna med objektorientering.

Att kapitlet är mycket kort speglar att det endast rör sig om en introduktion som du sedan ska använda på enklast möjliga sätt för att göra intressanta implementationer.

Kort om objektorientering

När man skriver objektorienterad programkod i C++ så bör man ha i åtanke vilka objekt som ska finnas i det färdiga systemet och hur de ska interagera. I klassisk C-programmering tenderar man mer att tänka på hur man kan lösa ett visst problem. Det är enklare för vår mänskliga hjärna att tänka i banor av hur ett system ser ut snarare än att lösa avancerade problem. Därför är det också enklare att skriva programkod med färre fel om man skriver den objektorienterad – konstigt nog, kan tyckas, tar det dock lite längre tid att lära sig hur man på bästa sätt bygger programkoden objektorienterat.

> Icke objektorienterad programmering kallas normalt procedurell programmering (för den klass av språk som C++ tillhör).

Programkod skriven i ett objektorienterat språk är ofta bättre uppdelad och ”modulariserad” än procedurell kod. Modulariserad innebär att hela applikationen består av flera små moduler, eller komponenter, som kapslar in sin funktionalitet och data. Kod skriven på detta vis ger högre kvalitet, färre brister och är dessutom lättare att återanvända och vidareutveckla.

Fördomar och nackdelar

Diskussionen kring objektorientering handlar i korthet om följande punkter:

- Objektorientering är svårare att lära sig programmera än procedurell programmering.
- Objektorienterad kod blir längre än motsvarande kod skriven i ett procedurellt språk.

- Objektorienterad kod är mer prestandakrävande än procedurell kod.
- Objektorienterad programkod är mer svårläslig än icke objektorienterad kod.

Den första punkten i föregående lista är faktiskt sann; punkt två beror på vad för slags program du utvecklar; de två sista är rent nonsens, de beror endast på programmeraren. Trots att objektorienterad programmering tar längre tid att lära sig så är nästan alla programmerare överens om att fördelarna med objektorientering klart överväger nackdelarna.

Lagring av flera typer i en

De typer du hittills använt har varit så kallade atomära typer. Nu ska du lära dig definiera en egen typ, en så kallad klass. Så här kan det se ut om du vill skapa en klass med två medlemsvariabler i:

```
class ArmeKapten
{
public:
  unsigned int IQ;
  unsigned int ViloPuls;
};
```

Klassens namn är **`ArmeKapten`** och de två medlemsvariablernas namn är **`IQ`** och **`ViloPuls`**. Medlemsvariabler är variabler i en klass, precis som globala variabler är variabler utanför alla funktioner och lokala variabler är variabler inuti funktioner.

Nyckelordet **`public:`** använder du för att visa att medlemsvariablerna ska kunna användas av alla, det vill säga publikt. När klassen är deklarerad så kan du skapa en variabel av typen **`ArmeKapten`**, precis som du tidigare kunnat skapa en variabel av typen **`int`**. När man skapar en variabel av klass-typ så kallas den normalt för *objekt* eller *instans*. Pekare och dylikt fungerar på samma sätt som för atomära typer:

```
ArmeKapten x;
ArmeKapten* p = &x;

ArmeKapten* Knasen = new ArmeKapten[5];
delete[] Knasen;
Knasen = 0;
```

Du ser dessutom att operatorerna **`new`** och **`delete[]`** fungerar precis som tidigare. Dock krävs det något nytt sätt för att komma åt medlemsvariablerna i den nya instansen.

I C++ används operatorn . . Du använder . för att hämta medlemsvariabler från ett vanligt objekt, medan operatorn -> används för att hämta medlemsvariabler från en pekare till ett objekt. Efter operatorn skriver du namnet på medlemsvariabeln. Så här:

```
ArmeKapten Fransson;
ArmeKapten* FranssonPek = &Fransson;

Fransson.IQ = 4;
FranssonPek->ViloPuls = 42;
```

Den sista raden kan skrivas om:

```
(*FranssonPek).ViloPuls = 42;
```

Ett standardexempel

Standardexemplet på en klass brukar vara något fordon med dess tillhörande attribut såsom färg, motorvolym, antal gaspedaler och maxhastighet. Här ett något annorlunda exempel:

```
class Human
{
public:
  bool IsGenderWoman;
  int EatenApples;
  char* Name;
};

class Park
{
public:
  int NumberOfTrees;
  int NumberOfFlowers;
  float WindDirection;
  float WindVelocity;
  Human People[2];
};

int main(int, char**)
{
  Park GardenOfEden;

  GardenOfEden.NumberOfTrees = 1301;
  GardenOfEden.NumberOfFlowers = 200002;
  GardenOfEden.WindDirection = 2.223f;
  GardenOfEden.WindVelocity = 36.9f;

  GardenOfEden.People[0].IsGenderWoman = true;
  GardenOfEden.People[0].EatenApples = 1;
  GardenOfEden.People[0].Name = ”Eve”;
```

```
    GardenOfEden.People[1].IsGenderWoman = false;
    GardenOfEden.People[1].EatenApples = 1;
    GardenOfEden.People[1].Name = "Adam";

    return 0;
}
```

Den första klassen heter **Human** och innehåller tre atomära variabler. Den andra klassen heter **Park** och innehåller fyra atomära variabler och en liten array med två objekt i. Dessa två objekt är av typen **Human**.

GardenOfEden är ett objekt av typen **Park**. På raderna efter deklarationen känner du igen initieringen av variabler i objektet, precis som i fallet med armékaptenen Fransson. Därtill ser du hur man kan använda klasser inuti andra klasser; detta är överkurs men kan vara bra att veta för framtiden.

Som parentes kan nämnas att man kan ha medlemsfunktioner likväl som medlemsvariabler i en klass. En medlemsfunktion i en klass kallas för en *metod*.

Övningsuppgifter

Övning 3.1 Beskriv skillnaden mellan lokal variabel, medlemsvariabel och global variabel.

Övning 3.2 Definiera en klass, **Knut**, med tre heltal och tre flyttal i. Medlemsvariablerna ska vara publika.

Övning 3.3 Deklarera ett objekt, **A**, av klassen **Knut** (från övningsuppgift 3.2). Tilldela alla medlemsvariabler i **A** var sitt värde.

Övning 3.4 Allokera ett objekt, **B**, av klassen **Knut** (från övningsuppgift 3.2). Tilldela alla medlemsvariabler i **B** var sitt värde. Frigör det allokerade minnet.

Övning 3.5 Allokera en array med 3 objekt, **C**, av klassen **Knut** (från övningsuppgift 3.2). Tilldela alla medlemsvariabler i alla objekt i **C** var sitt värde. Frigör det allokerade minnet.