Adressoperatorn &

I programmet **Pointer** gör deklarationen av variabeln **vanligVar** som **double** att en minnescell allokeras av storleken 8 bytes för lagring av ett **double**-värde. Därmed är minnescellens adress skapad och i princip känd för programmet. Tack vare *adressoperatorn* är denna adress även tillgänglig för programmet. Adressoperatorn symboliseras med tecknet & och tillämpas på en variabel genom att skriva:

&variabelnamn eller **& variabelnamn**

Även här saknar mellanslaget betydelse. Avgörande är att tecknet **&** skrivs *framför* variabelnamnet. Då returnerar adressoperatorn adressen till denna variabel. Variabeln behöver inte ens vara tilldelat ett värde. Det räcker att den är deklarerad. Även om vi i **Pointer** inte hade tilldelat variabeln **vanligVar** värdet **15.6** hade vi kunnat skriva ut dess adress med **cout << &vanligVar;** direkt efter deklarationen. Variabeln som adressoperatorn tillämpas på behöver inte vara en vanlig variabel. Den kan även vara en pekarvariabel. Då får man pekarvariabelns adress. Vi har i **Pointer** tillämpat adressoperatorn både på en vanlig variabel och en pekarvariabel. Körresultatet visar att pekarvariabelns *adress* är en annan adress än pekarvariabelns *värde*. Adressoperatorn kan även tillämpas på namngivna konstanter, däremot inte på icke-namngivna konstanter.

Vi återkommer till den mest intressanta observationen i körresultatet av **Pointer**, nämligen att pekarvariabeln **pekarVar**:s *värde* som är en adress, är identiskt med den vanliga variabeln **vanligVar**:s *adress* vilket beror på tilldelningssatsen:

```
pekarVar = &vanligVar;
```

som gör att **pekarVar** pekar på **vanligVar**. Vi kan rent formellt göra denna tilldelning då adressoperatorn returnerar adressen till en **double** och pekarvariabeln **pekarVar** är deklarerad som en pekare-till-**double**. Datatyperna på båda sidor överensstämmer alltså. Hade, om detta inte varit fallet, automatisk typkonvertering tillämpats? Svaret är nej. Observera att alla regler för *automatisk typkonvertering* endast gäller för *enkla* datatyper. Pekare är inga enkla datatyper utan sammansatta eller härledda då de är baserade på andra datatyper. Kom ihåg att med pekare allid menas pekare-till någon datatyp. Så, försöker man att tilldela **pekarVar** en annan datatyp än pekare-till-**double** som t.ex.: **pekarVar = 2.5;** blir det kompileringsfel med felmeddelandet att ett **double**-värde (enkel datatyp) inte kan konverteras till en pekare-till-**double** (sammansatt).

Med tilldelningssatsen ovan har vi gjort något man typiskt brukar göra med pekare. Vi har tagit en befintlig minnesadress, nämligen **&vanligVar**, och skrivit den i pekarvariabeln **pekarVar**:s minnescell. Vi har kopplat ihop den vanliga variabeln med pekarvariabeln. Först nu förtjänar pekaren sitt namn då den pekar på ett annat allokerat minnesutrymme, den vanliga variabelns minnesutrymme. Gör man inte det är pekaren inte väl definierad. Kom ihåg att först när en variabel är väl definierad får den användas. Först när en pekarvariabel pekar på ett minnesutrymme får den användas. Den typiska ”användningen” av en pekarvariabel är att med hjälp av

den komma åt det minnesutrymme som den pekar på. Detta har vi inte gjort i **Pointer** men kommer att göra i nästa programexempel.

*Värdeoperatorn **

Vi kallar den så med tanke på att den returnerar *värdet* analogt till adressoperatorn som returnerar adressen. *Värdeoperatorn* ***** (asterisken i en annan betydelse än hittills) ger oss möjligheten att komma åt vanliga variablers *värden* via deras adresser efter att ha kopplat ihop den vanliga variabeln med pekarvariabeln. Andra beteckningar som förekommer i litteraturen är ***-operatorn*, *indirektoperatorn* eller på engelska *dereference operator*.

Nästa programexempel **Value** på nästa sidan introducerar värdeoperatorn genom att visa hur man kommer åt *värdet* till en vanlig variabel med hjälp av en pekare som pekar på den. Samtidigt demonstreras sambandet mellan adress- och värdeoperatorn. I programmet **Value** har till skillnad från **Pointer** variabeln **vanligVar** inte tilldelats ett värde direkt via sitt namn utan indirekt via sin adress dvs en pekare som pekar på den. Dessutom har **pekarVar**:s deklarations- och tilldelningssats slagits ihop:

```
double *pekarVar = &vanligVar;
```

där i deklarationsdelen (till vänster om tilldelningstecknet) asterisken ***** använts som symbol för pekarvariabel. Precis som hos vanliga variabler kan man initiera pekarvariabler redan vid deklarationen. Gör man det till en vana kan man minska risken för oinitierade skräpadresser.

Helt ny här är tilldelningssatsen ***pekarVar = 15.6;** där asterisken ***** inte längre är kännetecknet för pekarvariabel. Satsen tilldelar istället den variabel som **pekarVar** pekar på, nämligen **vanligVar**, värdet **15.6**. Detta beror på att asterisken ***** inte har samma betydelse i programmets alla satser. Vi är redan vana att använda samma tecken för olika saker. Symbolen för multiplikation är samma som för pekare vid deklaration, och nu kommer en användning till för asterisken. Tillgången till tecken är ju begränsad, så det finns helt enkelt inte ett unikt tecken till varje tänkbar operation. Sammanhanget måste avgöra den rätta tolkningen.

Dessutom kan programmet **Value** användas – om man vill – för en kraschtest. Tar man bort kopplingen mellan den vanliga och pekarvariabeln, dvs deklarerar bara **pekarVar** utan att tilldela den adressen till **vanligVar**, så kan man väl kompilera koden. Men exekveringen leder till minneskrasch därför att det uppstår en pekare med en oinitierad skräpadress som pekar på ett minnesutrymme som med största sannolikhet är upptaget av ett annat program i datorns RAM.

Inversa operatorer

Inversa operatorer finns fler än man tror: Addition och subtraktion, multiplikation och division, att potentiera och dra roten, i C++: ökningsoperatorn **++** och minskningsoperatorn **--** , den logiska negationen **!** , **new** och **delete** (sid 285) osv.

Tillämpar man en operator på en operand och sedan den inversa operatorn på resultatet av denna operation får man tillbaka den ursprungliga operanden. Därför säger man: Inversa operatorer tar ut varandra, de neutraliserar varandra. T.ex. tillämpar man operationen +3 på operanden 4 och sedan den inversa operationen –3 på resultatet, får man tillbaka operanden 4. Dvs: (4 + 3) – 3 = 4. Ett annat enkelt exempel är (6 x 5) / 5 = 6. Den logiska negationen är ett exempel på en operator som är sin *egen invers*: **`!(!v) = v`** där **`v`** kan vara ett godtyckligt villkor eller en godtycklig utsaga. Man kan alltså förstå den dubbla negationen även med hjälp av den inversa operatorn. Vad har allt detta att göra med adresser och pekare? Jo, även värdeoperatorn kan man förstå utifrån adressoperatorn som vi redan känner till, plus konceptet om den inversa operatorn.

Vad gör adressoperatorn? Den tar en variabel och returnerar dess adress. Följaktligen gör värdeoperatorn som dess invers det motsatta: den tar en adress och returnerar dess variabel. Men vad är ”dess variabel”? Jo, det är inget annat än den variabel pekaren pekar på. Lite mer noggrant måste vi alltså formulera: Värdeoperatorn tar en pekare som pekar på en variabel och returnerar denna variabel. Själva variabeln som returneras behöver endast vara deklarerad. Antingen är den redan tilldelad ett värde, då returnerar värdeoperatorn detta värde. Eller så är variabeln inte tilldelad än, då kan man med värdeoperatorn tilldela den ett värde. Detta gjorde vi i programmet **`Value`** med satsen **`*pekarVar = 15.6;`** där **`*`** är värdeoperatorn. När vi skriver denna sats pekar pekarvariabeln **`pekarVar`** redan på den vanliga variabeln **`vanligVar`** genom tilldelningen **`pekarVar = &vanligVar;`** vilket gör att **`vanligVar`** i resten av programmet blir utbytbar mot **`*pekarVar`**. De refererar båda till samma minnescell och kan användas vare sig för att läsa eller skriva data. Med **`*pekarVar = 15.6;`** skrivs värdet **`15.6`** i minnescellen. Med **`cout << vanligVar;`** läses det från minnescellen. Beviset är att vi får **`15.6`** utskrivet med denna **`cout`**-sats fast vi tilldelat variabeln med **`*pekarVar`**. Detta visas i utskriften till programmet **`Value`** som visades tidigare.

Vi återkommer till det tidigare utlovade kraschtestet. Kommenterar man i **`Value`** bort kopplingen **`pekarVar = &vanligVar;`** mellan den vanliga variabeln och pekarvariabeln kan man fortfarande kompilera, men det blir exekveringsfel följt av minneskrasch. Testet kan lätt göras genom att ändra den andra satsen i **`main()`** till:

```
double *pekarVar;          // = &vanligVar;
```

Kopplingen som bryts ovan är ju inget annat än pekarvariabeln **`pekarVar`**:s tilldelningssats. Tar man bort den blir pekaren oinitierad. Den är deklarerad, men pekar på ingenting, närmare bestämt på inget minnesutrymme allokerat av programmet. Men som vi vet finns det ju rent fysiskt inga tomma minnesceller. I den deklarerade pekarvariabeln **`pekarVar`**:s minnescell står från tidigare användning något slumpmässigt odefinierat skräpvärde – en skräpadress. Med värdeoperatorn **`*pekarVar`** tillämpad på pekarvariabeln försöker vi nu komma åt värdet som **`pekarVar`** pekar på. Pga skräpadressen pekar **`pekarVar`** på ett minnesutrymme som med största sannolikhet är upptaget av ett annat program i datorn. Försöker man att från

ett program komma åt det minnesutrymme, blir det exekveringsfel och programmet kraschar pga minneskonflikt.

Ett annat lärorikt test är att i **`Value`** byta ut satsen **`*pekarVar = 15.6;`** mot:

```
*((double *) 0x12FF78) = 15.6;
```

Programmet kommer att fungera precis som förut, förutsatt att adressen som använts i denna sats är samma adress som **`pekarVar`**:s värde. Observera att sådana adresser blir olika när man kör programmet på annan dator eller vid olika tillfällen på samma dator. Det bästa är att ta adressen vid en aktuell körning från pekarvariabelns värde (se körresultatet ovan). Observera att den första asterisken är värdeoperatorn medan den andra tillhör den explicita typkonvertering som omvandlar den råa adressen till en adress-till-**`double`**. Testet visar att värdeoperatorn som vanligen skrivs framför en pekarvariabel, även fungerar när man skriver den framför en konstant adress. I så fall måste den konverteras till rätt datatyp och peka på ett allokerat minnesutrymme, i det här fallet variabeln **`vanligVar`** som även på det här sättet kan få värdet **`15.6`**.