Föreläsning 6

Kapitel 5

5.1 `switch`-satsen

Vi ser på ett par exempel ur boken:

```
int a;
srand(time(0));
a=rand()%6+1;
if(a==1)
  printf("Hej Du glade\n");
else if(a==2)
  printf("God dag\n");
else if(a==3)
  printf("Är Du här igen!\n");
else if(a==4)
  printf("Har inte Du slutat\n");
else if(a==5)
  printf("Kul att Du kunde komma\n");
else if(a==6)
  printf("En sista gång hoppas jag\n");
```

Lite tjatigt eller hur? För sådana här situationer har man skapat `switch`-satsen. Ovanstående skulle kunnat skrivas om så här: (`a==6` rättat?)

```
int a;
srand(time(0));
a=rand()%6+1;
switch (a){
  case 1: printf("Hej Du glade\n"); break;
  case 2: printf("God dag\n"); break;
  case 3: printf("Är Du här igen!\n"); break;
  case 4: printf("Har inte Du slutat\n"); break;
  case 5: printf("Kul att Du kunde komma\n"); break;
  case 6: printf("En sista gång hoppas jag\n");
}
```

Mycket enklare! Notera hur det blir olika fall, (eng. *case* = *fall*).

Allmänt har en `switch`-sats följande utseende:

```
switch(heltalsuttryck)
{
     case konstant heltalsuttryck1: sats
     case konstant heltalsuttryck2: sats
     case ...
     [default: sats]

}
```

Boken skriver "`villkor`" istället för "`heltalsuttryck`". (*Heltalsuttryck* är bättre tycker jag.)

Vad betyder `break`? Vi kan förstå det utifrån en helhetsförståelse av `switch`-satsen. Så här fungerar en `switch`-sats:

1. Man har en heltalsvariabel man vill undersöka värdet av. (Ovan undersökte vi variabeln `a`.) För att starta en switchsats skriver man `switch(heltalsvariabeln)`, ovan skrev vi `switch(a)`.
2. Alla `case`-satser gås nu igenom. Varje `case`-sats har ett tillhörande konstant heltalsvärde. Värdet i variabeln man testar på (ovan var det variabeln `a`) jämförs med alla `case`-satsers konstanter och så fort en överenstämmelse nås så utförs den sats som finns i den `case`-sats där en överenstämmelse först uppkom. Till exempel, om `a`, ovan hade värdet 4 så kördes den `case`-sats som ser ut så här:
   ```
   case 4: printf("Har inte Du slutat\n"); break;
   ```
   Det som händer då är att vi får utskriften "`Har inte Du slutat\n`".
3. Sedan kommer vi till `break`. Saken är den att om inte `break` står där så kommer samtliga efterföljande `case`-satser att utföras ända tills ett `break` hittas.

Det är ovanligt att man vill åstadkomma effekten att alla `case`-satser körs så fort en `case`-sats ska köras, men det är viktigt att känna till att möjligheten finns. Skriv in och testa följande exempel för att första detta:

```
#include <stdio.h>
#include <conio.h>

int main(void){

  int tal=4, a=0;

  switch (tal)
  {
    case 1: a=a+1;
    case 2: a=a+2;
    case 3: a=a+3;
    case 4: a=a+4;
    case 5: a=a+5;
    case 6: a=a+6;
  }

  printf("a blir %d\n",a);
  getch();
}
```

(Diskutera hur pogrammet kör. Poängtera att först måste en `case`-sats heltalsuttryck vara lika med `tal`.)

En `switch`-sats kan ha en avdelning som heter `default`. Det engelska ordet "*default*" betyder "*förvald*" och om en `default`-sats finns i en `switch`-sats så utförs den om ingen av `case`-satserna utförts. Vi tar ett exempel:

```
int weekday;
printf("Mata in veckodagnummer 1-7: "); scanf("%d", &weekday);
switch(weekday)
{
  case 1: printf("Monday\n"); break;
  case 2: printf("Tuesday\n"); break;
  case 3: printf("Wednesday\n"); break;
  case 4: printf("Thursday\n"); break;
  case 5: printf("Friday\n"); break;
  default: printf("Weekend! Party! Groovy!\n");
}
```

Rutinen frågar efter veckodagsnumret och skriver ut olika veckodagsnummer och skriver ut olika responser baserat på vad som skrivs in, 1 ger `Monday`, 2 ger `Tuesday` osv, men om ingen av dagarna ovan valts har användaren troligen matat in 6 eller 7 som båda då ger utskriften "`Weekend!...`". I ett sådant här fall är det en bra vana att förse en `switch`-sats med en `default`-del, då är man säker på att man utför något i respons till varje inmatat heltal. Observera dock att ovanstående rutin ger responsen `Weekend` till 1000 eller -4 eller vad som helst som inte är talen 1, 2, 3, 4 eller 5!

5.2 `break`

Vi har sett satsen `break` ovan och sett att den avslutar de olika fallen i en `switch`-sats. Det finns dock flera användningar av `break` som vi bara förklara lite kort.

Satsen `break` får bara finnas inuti en sammansatt sats som styrs av en `switch`-sats, en `for`-sats, en `while`-sats eller en `do`-sats. När `break` körs så avbryts den sammansatta satsen, om den styrs av en `switch`-sats, så hoppar vi ut ur `switch`-satsen, om den styrs av en `for`-sats så avlutas den och likadant om det är en `while`-sats eller `do`-sats så avlutas de också. Det är inte troligt att ni kommer att använda det så mycket så vi kommer inte att uppehålla oss kring detta.

5.3 `continue`

En lite märklig sats som vi lämnar åt självstudier. Ni kan hoppa över den om ni vill.

5.4 `goto`

Denna sats har potientialen att ställa till det riktigt! Den är såpass farlig att jag förbjuder dess användande i laborationerna. Om man inte vet *precis* vad man gör kan något som kallas "spaghettiprogrammering" uppstå. Då blir det *mycket* svårt att förstå programmet och svårt att utveckla det. Använd inte `goto` i början av en programmeringskarriär, och OM ni börja använda det, var mycket försiktiga. Vi hoppar över den i den här kursen. Författaren har en annan hållning och på den praktiska tentamen så kommer ni inte få underkänt om ni använder `goto`, MEN jag kommer som sagt inte att acceptera `goto` i laborationerna.

5.5 Reserverade ord i *C*. Läs själva.

5.6 Satser i *C*. Ger en översikt av vad vi gått igenom. Läs själva.

5.7 Sammansatta tilldelningar. Läs själva.

5.8 Typomvandling

När *C* gör en beräkning, till exempel `a = b+c;` eller `a = b*c;` så har operanderna (`b` och `c`) typer, det är kanske `int` eller `float`, och resultatet är en annan typ. (Sedan vad som lagras i `a` beror på vilken typ `a` har.) Vi ska se på några exempel (exempel 10)

```
int a;
short int b=1234,c=100,d;
float e,f,g,h,i;
a=b*c;          // b och c är short int så b*c blir int.
d=b*c;          // b och c är short int så b*c blir int. d?
e=b/c;          // b och c är short int så b*c blir int. e?
f=(float)b/c;   // (float)b är float så (float)b/c blir float.f?
g=1.0*5/2;      // 1.0*5 är float så 1.0*5/2 blir float. (2.5)
h=5/2*1.0;      // ...
i=(float) 5/2;  // ...
```

Det är viktigt att känna till sådana här omvandlingar, det bästa är att testa sig fram till en förståelse av dessa regler. Skriv en del testprogram och experimentera er fram.

5.9 Kommentarer

Med tecknen `//` och `/**/` kan man skriva in förklarande text i sina *C*-program, man startar en kommentar med `/*` och avslutar den med `*/`. Man kan också skriva `//` och då blir hela resten av den rad man är på en kommentar. Kommentarer tas bort av kompilatorn och är endast ett sätt för programmeraren att förklara sitt program.

Det är bra att kommentera sina program, de laborationer ni redovisar ska vara *väl* kommenterade. Hur mycket ska man skriva som kommentarer? Det får ni utveckla en känsla för efter hand. Skriv hellre för mycket än för litet. En kommentar ska vara av förklarande karaktär och bekriva mer abstrakt syftet med den programkod man skriver. Det är meningslöst att i kommentarform bara beskriva vad som sker programmeringstekniskt, alltså, skriv inte så här:

```
int a; //Variablen a deklareras
a=10; //a tilldelas 10
```

Det tillför inget nytt. Kommentarerna blir överflödiga. Bättre kommentarer skulle vara

```
int max; //Variablen max innehåller max antal rundor i spelet
max=10; //Som förval ges max värdet 10.
```

Här har vi även varit mer tydliga och valt ett bättre variabelnamn, ”`max`” är ett bättre namn än bara ”`a`”. Med bra variabelnamn som beskriver det de innehåller och utförliga förklarande kommentarer och bra indentering i ett program så uppnår man mycket hög grad av läslighet. Det ska ni absolut sträva efter.

5.10 Förprocessorn

Vi ser på ett exempel från boken:

```
#include <stdio.h>
#include <conio.h>
#define MOMS 25
#define RABATT 10
int main(void) {
  int antal=10;
  float tpris;
  const float pris=3.25;
  tpris=antal*pris*(1+MOMS/100.0)*(1-RABATT/100.0);
  printf("%6.2f kr\n",tpris);
  getch();
}
```

Det finns tre typer av datalagring i detta, dels genom variablerna `antal` och `tpris`, men dels också genom en konstant `pris` samt två symboliska konstanter, `MOMS` och `RABATT`. Variabler har vi sett tidigare, de utgör en lagringsplats för data som kan variera under programmets körning, vi tilldelar variabeln `tpris` ett nytt värde som vi ser ovan. Konstanter som `pris` ovan har dock samma värde under hela programmets körning. Symboliska konstanter är liknande konstanter, skillnaden är att innan kompilering så ersätts texten `MOMS` överallt i programkoden med 25 och texten `RABATT` ersätts med texten 10. Skillnanden är inte så stor att vi behöver tala som mycket om den i den här kursen. Ni kan bara veta att symboliska konstanter skapas med så kallade förprocessordirektiv, (som inleds med `#`) precis som vissa bibliotek inkluderas, som `stdio.h` och `conio.h`.

Kapitel 6.

6.1 Indicerade variabler

Det finns många namn på detta avsnitt, *indexerade variabler*, *arrayer*, *fält*, *vektor*, *lista*, *matris* etc. Vi kommer att tala om det som *indexerade variabler* eller *arrayer*, boken använder ett *c* i stavningen, men det är ingen skillnad.

En *array* eller *indexerad variabel* är en variabel som kan innehålla flera exemplar av data av samma typ. Dessa enskilda exemplar indexeras med ett heltal. Indexerade variabler deklareras som vanliga variabler men med ett antal angivet inom hakparentes. Ex: `int a[10]` ger plats för 10 heltal. Detta skapar i själva verket 10 skilda variabler som man kommer åt genom `a[0]`, `a[1]`, ..., `a[9]`. Observera att index löper från 0 till 9, det blir 10 lagringsplatser. Man börjar alltid på 0 i *C* och det är en god vana att följa den konventionen.

6.2 Till vad används indexerade variabler?

Vi tittar på ett exempel: (Tyvärr är numreringen inte från 0, men exemplet är bra.)

```
#include <stdio.h>
#include <conio.h>
int main(void)
{
  int s2,s3,s4,s5,s6,s7,s8;
  int s9,s10,s11,s12,k,kast;

  s2=s3=s4=s5=s6=s7=0;
  s8=s9=s10=s11=s12=0;

  srand(time(0));

  for(k=1;k<=10000;k++){
    kast=rand()%6+1+rand()%6+1;
    if (kast==2) s2++;
    if (kast==3) s3++;
    if (kast==4) s4++;
    if (kast==5) s5++;
    if (kast==6) s6++;
    if (kast==7) s7++;
    if (kast==8) s8++;
    if (kast==9) s9++;
    if (kast==10) s10++;
    if (kast==11) s11++;
    if (kast==12) s12++;
  }

  printf("Antal 2: %d\n",s2);
  printf("Antal 3: %d\n",s3);
  printf("Antal 4: %d\n",s4);
  printf("Antal 5: %d\n",s5);
  printf("Antal 6: %d\n",s6);
  printf("Antal 7: %d\n",s7);
  printf("Antal 8: %d\n",s8);
  printf("Antal 9: %d\n",s9);
  printf("Antal 10: %d\n",s10);
  printf("Antal 11: %d\n",s11);
  printf("Antal 12: %d\n",s12);
  getch();
}
```

Vad gör detta program? Jo, det slår två sexsidiga tärningar 10000 gånger och lagrar antal 2:or i `s2`, antal 3:or i `s3` osv upp till antal 12:or i `s12`. Sedan skrivs de 11 variablerna ut. Ganska klumpigt, men det bli mycket bättre när man använder en array med 11 platser. Vi ser på nästa exempel som är betydligt elegantare:

```
#include <stdio.h>
#include <conio.h>
int main(void)
{
  int s[13],kast,k;
  srand(time(0));

  for(k=2;k<=12;k++)
    s[k]=0;

  for(k=1;k<=10000;k++)
  {
    kast=rand()%6+rand()%6+2;
    s[kast]++;
  }

  for(k=2;k<=12;k++)
    printf("Antal %2d: %d\n",k,s[k]);

  getch();
}
```

Här använder vi en array som deklareras med int `s[13]`. Den ger oss då 13 platser att lagra heltal på som numreras från 0 till och med 12. Vi använder bara de 11 översta som numreras med 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 och 12 som är de olika utfallen av två tärningar som kastas. (Gå igenom steg för steg hur programmet fungerar.)

6.3 Sortering

Att sortera data är en mycket viktig aktivitet inom databehandling. Det är till och med så att jag vill höja dess status till en standardiserad problemdomän och säga på förhand att på den praktiska tentamen kommer en uppgift att komma som är ett rent sorteringstekniskt uppdrag. Om vi har en array med tal som har blandade storlekar kanske vi vill sortera innehållet så att det minsta talet kommer först och det största sist och alla mellanliggande förekommer i storleksordning.

Osorterat: `34 78 12 90 4`

Sorterat: `4 12 34 78 90`

För att sortera lämpar sig arrayer alldeles ypperligt. Det finns flera metoder, det hela liknar att stortera en korthand när man spelar kort: Först hittar man det minsta kortet och placerar det först, sedan tar man det näst minsta och placerar det på andra platsen och så vidare. Om vi skulle sortera en pokerhand med låt oss säga bara en massa hjärterkort med valörerna 11 5 3 9 4 skulle det kanske gå till så här:

11 5 3 9 4, finn minsta, det är 3, placera den först, det ger oss
3 11 5 9 4, finn minsta bland de fyra högraste korten, det är 4, placera den näst först,
3 4 11 5 9, finn minsta bland de tre högraste korten, det är 5, placera den på sin plats,
3 4 5 11 9, finn minsta bland de två högraste korten, det är 9, placera den på sin plats.

Det är ungefär så här det går till när man sorterar i en dator också, skillnaden är att man byter plats på den minsta man hittar istället för att flytta upp alla ett steg, så här:

Sortera arrayen med talen `11 5 3 9 4`:

Steg 1: Finn minsta i hela arrayen, det är 3, byt plats på 3 och talet på första platsen. (11 och 3 byter plats). Det ger: `3 5 11 9 4`.

Steg 2: Finn minsta talet i de fyra högraste talen, det är 4, byt plats på 4:an och talet på andra platsen. (5 och 4 byter plats.) Det ger: `3 4 11 9 5`.

Steg 3: Finns minsta talet i de tre högraste talen, det är 5, byt plats på 5:an och talet på 3:e plats. (11 och 5 byter plats.) Det ger: `3 4 5 11 9`.

På samma sätt hamnar också 9:an och 11:an på rätt plats och ger oss den sorterade arrayen `3 4 5 9 11`.

Ett programexempel baserat på detta sätt att sortera kan se ut så här:

```
#include <stdio.h>
#include <conio.h>
int main(void){
  int j,k,m,tmp;
  int v[7]={0,45,34,23,51,32,19};

  for(k=1;k<=5;k++)
  {

    for(m=k+1;m<=6;m++)
      if(v[k]>v[m])
      {
        tmp=v[k];
        v[k]=v[m];
        v[m]=tmp;
      }

    for(j=1;j<=6;j++)          //Förklara detta först
      printf("%d ",v[j]);
    printf("\n");

  }

  getch();
}
```

Dess utskrift ser ut så här:

```
19 45 34 51 32 23
19 23 45 51 34 32
19 23 32 51 45 34
19 23 32 34 51 45
19 23 32 34 45 51
```

Programmet ovan är baserat på exempel 8 i boken men med mer utskrifter.

(Gå igenom steg för steg hur sorteringen fungerar.)

6.4 Matriser

Indexerade variabler med två index kan kallas matriser, eller två dimensionella vektorer. Det är enkelt att deklarera dessa i *C*, man lägger bara på ett extra index:

```
int a[10][8];
```

Denna deklaration skapar en variabel a med 80 lagringsplatser numrerade från `a[0][0]`, `a[1][0]`, `a[2][0]`, ..., `a[9][6]`, `a[9][7]`.

Med en matris är det lämpligt att använda en dubbelloop, att gå igenom ovanstående matris kan alltså göras så här:

```
int i, j;
for(i=0;i<10;i++)
  for(j=0;j<8;j++)
    a[i][j] = 0;
```

På detta sätt kan alla element i matrisen nollställas.

Man kan ha flera än två index i en tabell, följande exempel illustrerar en tabell med 1000000 platser på vardera 8 byte, alltså en 8 MBytestabell:

```
double c[100][100][100];
```

6.5 Tidtagning: Läs själva, inte så viktigt.

Kommentar om denna föreläsning och detta avsnitt

Innehållet här är förmodligen ett slags nyckel: om ni studerar detta kapitel noggrannt och gör många av dess övningsuppgifter kommer ni troligen att klara kursen bättre. Det innehåller mer komplicerad problemlösning och författaren har förmodligen varit medveten om detta eftersom han gjort hela 21 övningsuppgifter. Gör samtliga dessa övningsuppgifter!

Vi har dessutom öppnat upp en till av de standardmässiga problemdomänerna som hör till kursen: sortering och sökning. Det kommer en särskild föreläsning senare för att fördjupa och förtydliga det här med sökning och sortering. Kursen innehåller fyra problemdomäner:

1. Allmän problem lösning, kan vara vad som helst inom problemlösning med generella problem som inte har någon speciell tillhörighet. Huvuddelen av tentamen kommer att vara uppgifter inom den här problemdomänen.
2. Grafisk representation av matematiska/fysikaliska modeller (har vi redan undersökt.)
3. Sökning, sortering, representation.
4. Stora program

Jag arbetar för att den praktiska tentamen ska få följande utformning:

- Problemdomän 1, allmän problemlösning: 5 uppgifter.
- Problemdomän 2, grafisk representation: 1 uppgift.
- Problemdomän 3, sökning, sortering och representation: 1 uppgift.
- Problemdomän 4, stora program: 1 uppgift.

(Jag vet inte om jag kommer att lyckas, men vi får se. Jag ger er besked om detta i god tid.)