Beslutsfattande

- Hittills har våra program alltid körts en rad i taget, från första raden till sista
- För att skriva realistiska program behövs mekanismer för att få datorn att fatta **beslut** utgående från olika **villkor**
- Gör det möjligt att köra endast delar av program (=*flödeskontroll*), eller köra valda delar av programmet upprepade gånger (=*iteration*)
- För att åstadkomma detta krävs *logiska uttryck* och *villkor* – ***boolesk logik***

Logiska uttryck

- Logiska uttryck har antingen värdet *sant* eller *falskt*
  - "I dag skiner solen" =>
    sant eller falskt, beroende på vädret
  - "1+2 är större än 5" => alltid falskt
  - "a-3 är mindre än 0" =>
    sant om a är mindre än 3
- Datorprogrammering går till stor del ut på att analysera logiska uttryck och beroende på resultatet utföra olika instruktioner
- "Om uttrycket är sant, gör si. Om inte, gör så."

Relationsoperatorer

- Kan använda *relationsoperatorer* liknande dem som används i vanlig algebra för att konstruera logiska uttryck:

`==`	ekvivalent med, “lika med” (Obs! Två likhetstecken!)
`!=`	icke ekvivalent med, “olika”
`<`	mindre än
`>`	större än
`<=`	mindre än eller lika med
`>=`	större än eller lika med

Konstanta logiska uttryck

```
10 < 23   (sant)
10 == 23  (falskt)
10 != 23  (sant)
10 >= 10  (sant)
```

- Värdet på ovanstående uttryck är **konstanta**
- I ett program innehåller de logiska uttrycken vanligen en variabel
  => uttryckets värde kan **variera** under programkörningens gång

Övning: Variabla logiska uttryck

- För vilka värden på **a** och **b** är följande uttryck sanna respektive falska?:

```
a<21

a*4 == 20

121 != a

a > b

2*b <= a
```

Beslutsfattande och programflöde:

- Med hjälp av logiska uttryck och **`if`**-satser kan valda delar av ett program köras:

```
#include <stdio.h>
int main(void)
{
    int age = 0;
    printf("Välkommen till Trafi\n");
    printf("Hur gammal är du? ");
    scanf("%d", &age);
    if (age < 18)
    {
        printf("Inget körkort för dig!\n");
    }

    return 0;
}
```

Inget semikolon här!

Kodblock som körs endast om villkoret är **sant**

Beslutsfattande och programflöde:

- **`if`**-satsen kan utökas med en **`else`**-del för ett alternativt programflöde:

```
if (age < 18)
{
    printf("Inget körkort för dig!\n");
}
else // means that age >= 18
{
    printf("Ok, tuta och kör!\n");
}
```

Kodblock som körs endast om villkoret är **falskt**

Beslutsfattande och programflöde:

- Det kan finnas flera **`else`**-delar:

```
if (age < 18)
{
    printf("Inget körkort för dig!\n");
}
else if (age > 69)
{
    printf("Läkarkontroll krävs!\n");
}
else // means that age is between 18 and 69
{
    printf("Ok, tuta och kör!\n");
}
```

Nästlade if-satser

- Kan ha **`if`**-satser inne i andra **`if`**-satser:

Nästlad if-sats

```
if (age > 69)
{
    int last_checkup;
    printf("Hur många år sedan senaste kontroll?");
    scanf(" %d", &last_checkup);
    if (last_checkup > 4)
    {
        printf("Läkarkontroll krävs!\n");
    }
    else
    {
        printf("Ok, återkom om %d år.\n",
                              5-last_checkup);
    }
}
```

Iteration: While-loopen

- En **`while`**-loop består av ett kodblock som upprepas *så länge ett logiskt villkor är sant*

Kodblock som skall upprepas så länge a<10

```
#include <stdio.h>

int main(void)
{
    int a = 0;
    while (a < 10)
    {
        printf("Loop!\n");
        a++; /* Öka värdet på a */
    }
    return 0;
}
```

Inget semikolon!

Kodblock och variabler

- En variabel kan användas **inne i det kodblock där den är definierad samt i alla underliggande nästlade kodblock**
- **Inte tillåtet** att definiera en ny variabel med samma namn inom samma kodblock
- **Tillåtet** att definiera en ny variabel med samma namn i ett inre, nästlat kodblock

  => denna variabel gäller endast i det inre kodblocket

  => en “yttre” variabel med samma namn “skrivs över”

  => flera variabler med samma namn gör programmet svårförståeligt och ökar risken för buggar

Kodblock och variabler, exempel

- Vad ger följande program för utskrift?

```
#include <stdio.h>

int main(void)
{
     int num = 10;
     while ( num >= 10 )
     {
          int num = 20;
          printf("num is now %d\n", num);
          num--;
     }
}
```

Kodblock utan 'curly braces'

- Om ett kodblock innehåller endast en instruktion kan vågparenteserna utelämnas
- Förekommer i kursboken, men rekommenderas inte!

```
int a=200;

while (a > 100)
    printf("%d\n", a);
    a = a - 10;

if (a<100)
    printf("Case one!\n");
else
    printf("Case two!\n");

// Utskrift?
```

Nästlade loopar

- En loop kan innehålla en annan loop
- Vad kommer nedanstående program att skriva ut?

```
int a=1;
while ( a <= 10 )
{
     int b=0;
     while ( b <= 5 )
     {
        printf("a:s värde är nu %d\n", a);
        printf("b:s värde är nu %d\n", b);
        b++;
     }
a++;
}
```

Datatyper för booleska värden

- Kan lagra resultatet från ett logiskt uttryck i en boolesk variabel
- Traditionellt använder C heltalsvärden för att representera sant eller falskt: 0 för *false*,1 för *true*
- Fr.o.m. C99 stöder C även en boolesk datatyp **`_Bool`**
  - Giltiga värden är 0 eller 1
- Kan inkludera headern **`stdbool.h`**

  => Kan använda datatypen **`bool`** och värdena **`true`** och **`false`** i dina program
  => Kompilatorn omvandlar **`true`** till 1 och **`false`** till 0

Booleska variabler, exempel

```
#include <stdio.h>
#include <stdbool.h>
int main(void)
{
    bool go_on = true;
    printf("Welcome to the merry-go-round\n\n");
    while (go_on == true)
    {
        printf("I'm getting dizzy!\n");
        printf("Once more (y/n)? ");
        char input = 0;
        scanf(" %c", &input);
        if (input == 'n')
        {
            go_on = false;
        }
    }
}
```

Booleska variabler, exempel

```
#include <stdio.h>
#include <stdbool.h>
int main(void)
{
    bool go_on = true;
    printf("Welcome to the merry-go-round\n\n");
    while (go_on) // Can skip the '== true' part
    {
        printf("I'm getting dizzy!\n");
        printf("Once more (y/n)? ");
        char input = 0;
        scanf(" %c", &input);
        go_on = (input == 'y'); // Can assign the value of
                                // the logical expression
                                // directly to the variable
    }
}
```

Booleska variabler, exempel

```
#include <stdio.h>
// Note: No stdbool required in this case!
int main(void)
{
    int go_on = 1; // Use integer value directly
    printf("Welcome to the merry-go-round\n\n");
    while (go_on)
    {
        printf("I'm getting dizzy!\n");
        printf("Once more (y/n)? ");
        char input = 0;
        scanf(" %c", &input);
        if (input == 'n')
        {
            go_on = 0;
        }
    }
}
```

Booleska variabler, exempel

```
#include <stdio.h>
// Note: No boolean variable needed at all!
int main(void)
{
    char input = 'y';
    printf("Welcome to the merry-go-round\n\n");
    while (input != 'n')
    {
        printf("I'm getting dizzy!\n");
        printf("Once more (y/n)? ");
        scanf(" %c", &input);
    }
}
```