

# Komplext beslutsfattande

```
// Print OK if age is between 18 and 69
if (age >= 18)
{
    if (age <= 69) // Nested loop :(
    {
        printf("OK!\n");
    }
}

// Print NOT OK if age is less than 18 or more than 69
if (age < 18)
{
    printf("NOT OK!\n");
}
if (age > 69) // Duplicated code :(
{
    printf("NOT OK!\n");
}
```

# Logiska operatorer

- Ett logiskt uttryck kan bestå av flera *deluttryck* som kopplas samman med hjälp av *logiska operatorer*:
- **del1 AND del2** är sant om båda deluttrycken är sanna
  - Med C-syntax: **del1 && del2**
- **del1 OR del2** är sant om del1 **eller** del2 (eller bägge villkoren) är sanna
  - Med C-syntax: **del1 || del2**
- **NOT del1** är sant om del1 **inte** är sant
  - Med C-syntax: **!del1**

# Logiska operatorer, exempel

```
if (age >= 18 && age <= 69) // AND
{
    printf("OK!\n");
}
```

```
if (age < 18 || age > 69) // OR
{
    printf("NOT OK!\n");
}
```

```
if (!(age < 18 || age > 69)) // NOT, OR
{
    printf("OK..I think?");
}
```

# Övning: Logiska operatorer

- Vad innebär följande uttryck?  
(antag att *a* och *b* är heltalsvariabler)  
`( a > 50 ) && ( a < 100 )`  
`( b % 2 == 0 ) || ( b == 1 )`
- Formulera villkor för följande situationer med hjälp av relationsoperatorer och logiska operatorer:
  - Variabeln **age** skall vara lika med 30 eller 40
  - Variabeln **value1** skall vara större än en fjärdedel men mindre än **value2**
  - Variabeln **a** skall vara mindre än **b** som i sin tur skall vara större än eller lika med **c**.

# Mera valmöjligheter: **switch**

- Alternativt sätt att styra programflödet
- Avsett för situationer där alternativet är en lång **if..else** konstruktion
- **expression** kan anta olika värden
- om **value1** => utför case 1
- om **value2** => utför case 2
- annars, utför **default**

```
switch (expression)
{
    case value1:
        program statements
        break;
    case value2:
        program statements
        break;
    ...
    default:
        program statements
        break;
}
```

# switch, exempel

```
int i = 0;
printf("Enter month (1-12)");
scanf("%d", &i);
switch (i)
{
    case 1:
        printf("January\n");
        break;
    case 2:
        printf("February\n");
        break;
    // and so on...
    case 12:
        printf("December\n");
        break;
    default:
        printf("Illegal month!");
        break;
}
```

```
int noOfDays = 0;
switch (i) // assume i is an integer value
{
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        noOfDays = 31;
        break;
    case 2:
        noOfDays = 28; // Ignoring leap year
        break;
    case 4:
    case 6:
    case 9:
    case 11:
        noOfDays = 30;
        break;
    default:
        printf("Illegal month!\n");
}
```

**Inget break after case  
=> 'fallthrough'**

**Inget break after case  
=> 'fallthrough'**

# Iteration: Do-loopen

- En variant av **while**-loopen – kodblocket körs alltid minst en gång, även om det logiska villkoret är falskt

Kodblock som körs en gång trots att villkoret inte uppfylls

```
#include <stdio.h>

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



# Iteration: Do-loopen

```
#include <stdio.h>

int main(void)
{
    char input; // initial value irrelevant!
    printf("Welcome to the merry-go-round\n\n");
    do // at least one round!
    {
        printf("I'm getting dizzy!\n");
        printf("Once more (y/n)? ");
        scanf(" %c", &input);
    }
    while (input != 'n');
}
```

# Iteration: **for**-loopen

- Används ofta då man på förhand vet hur många iterationer som skall utföras
- Mer kompakt syntax än **while**-loopen, men en **for**-loop kan alltid uttryckas som en **while**-loop
- Allmän syntax:  
**for** (*init\_expression; loop\_condition; loop\_expression*)  
{  
    program statements  
}

# Iteration: **for**-loopen

- Exempel: Skriv ut "Hej!" tio gånger:

Initialisering:  
Utförs **en gång**

Loopen upprepas  
så länge detta  
villkor är sant

Utförs **efter varje iteration**

Kodblock som  
utförs medan  
villkoret är sant

```
int a;  
for ( a = 0; a < 10; a++ )  
{  
    printf("Hej!\n");  
}
```

# Variabeldeklaration & initialisering

- ANSI C99 tillåter deklaration av en variabel inne i for-loopens initialiseringsdel  
=> Variabeln giltig endast inne i for-loopen
- **Äldre versioner av gcc (pre-5.x)** kräver flaggan **std=c99** för att nedanstående kod skall kunna kompileras

```
for ( int a = 0; a < 10; a++ )  
{  
    printf("Hej!\n");  
}  
  
// use 'gcc test.c -std=c99' if needed
```

# Variationer på **for**-loopar

- Ett fält i en for-loop kan innehålla flera instruktioner som separeras med komma
- Ett fält kan lämnas helt tomt

```
/* Example: One statement in the 3rd field */  
for ( int a = 0, b = 200; a < 10; a++, b = b - 4 )  
...  
  
/* Example: First field is empty */  
int c = 50;  
for ( ; c > 100; c = c + 10 )  
...
```

# Nästlade for-loopar

- For-loopens kompakta syntax gör den lämplig för nästlade loopar

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