

F03 - iteration, loopar

Programmeringsteknik med C och Matlab, 7,5 hp

Niclas Börlin
niclas.borlin@cs.umu.se

Datavetenskap, Umeå universitet

2023-10-02 Mån

- ▶ Fönstret för att anmäla sig till tentamen har öppnat
- ▶ Är öppet till och med 10 okt
- ▶ Instruktioner finns på <https://www.umu.se/student/mina-studier/tentamen/digital-salstentamen/>

Iteration

- ▶ För att ett programmeringsspråk ska vara **användbart** måste man kunna uttrycka
 - ▶ **sekvenser** av satser
 - ▶ uttrycka **val**
 - ▶ **upprepa** satser (iterationer)
- ▶ Språket C innehåller följande iterationssatser:
 - ▶ **for**
 - ▶ **while**
 - ▶ **do-while**

Blank

while

- ▶ **while**-satsen upprepar något **så länge** ett test är **sant**
 - ▶ Det motsvarar ungefär:
 - ▶ *Så länge det finns disk kvar i diskmaskinen, plocka ut en sak*
 - ▶ *While there are dishes left in the dishwasher, pick one item*

while-satsen, syntax

- ▶ C-syntaxen för en **while**-sats är

```
while ( TEST )  
    STATEMENT
```

eller hellre

```
while ( TEST ) {  
    STATEMENT1  
    STATEMENT2  
    ...  
}
```

- ▶ Så länge **TEST** är sant utförs **satsen/blocket** efter slutparantesen
- ▶ **TEST** beräknas **innan** satsen/blocket körs **första gången**

while-exempel

- ▶ Vad gör följande exempel?

```
1  #include <stdio.h>  
2  
3  int main(void)  
4  {  
5      int i = 0;  
6  
7      while (i < 3) {  
8          printf("%d ", i);  
9          i = i + 1;  
10     }  
11  
12     printf("\nNormal exit.\n");  
13     return 0;  
14 }
```

Simulering while-exempel

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int i = 0;
6
7     while (i < 3) {
8         printf("%d ", i);
9         i = i + 1;
10    }
11
12    printf("\nNormal exit.\n");
13    return 0;
14 }
```

0123 i

```
0 1 2
Normal exit.
```

Hur många varv? (1)

- ▶ Studera följande kodsnuitt:

```
1 i = 0;
2
3 while (i < n) {
4     printf("%d ", i);
5     i = i + 1;
6 }
```

- ▶ Antag att **n == 5**
 - ▶ Hur många gånger kommer **satsen** `printf(...)` på rad 4 att utföras?
 - ▶ 5
 - ▶ Hur många gånger kommer **uppräknings** `i = i + 1` på rad 5 att utföras?
 - ▶ 5
 - ▶ Hur många gånger kommer **testet** `i < n` på rad 3 att utföras?
 - ▶ 6

Hur många varv? (2)

- ▶ Samma kodsnuitt:

```
1 i = 0;
2
3 while (i < n) {
4     printf("%d ", i);
5     i = i + 1;
6 }
```

- ▶ Vad händer om **n == 0**?
 - ▶ Testet körs **1** gång (och returnerar **false**)
 - ▶ Loopen körs **0** gånger
- ▶ Vad gäller i det **generella** fallet?
 - ▶ Hur många gånger körs respektive del, som **funktion av n**?
 - ▶ Testet körs **n + 1** gånger (n ggr **true**, en gång **false**)
 - ▶ Loopen körs **n** gånger

Uppräkning

- ▶ Operationen `i = i + 1` i slutet av **while**-loopen kallas för en **uppräkning**
- ▶ Det ligger vanligen **sist** i loopen
- ▶ Uppräkningen `i = i + 1` är så vanlig i C att den fått sin **egen** operator `i++`
- ▶ På samma sätt finns motsvarande operator `i--` för **nedräkning**
- ▶ Några vanliga uppräkningsoperatorer är:

Operator	Gör samma som
<code>i++</code>	<code>i = i + 1</code>
<code>i--</code>	<code>i = i - 1</code>
<code>i += 2</code>	<code>i = i + 2</code>
<code>i -= 2</code>	<code>i = i - 2</code>

- ▶ Vad är problemet med följande kod?

```
1  int i = 0;
2
3  while (i < 5) {
4      printf("%d ", i);
5  }
```

- ▶ i uppdateras aldrig, så loopen kommer aldrig att **terminera**!
- ▶ Kom ihåg att **uppdatera** loop-variabeln!

Blank

- ▶ Det går att avbryta en **while**-loop (och **for** och **do-while**) "inifrån" loopen med **break**

```
1  while (i < n) {
2      ...
3      if (user_pressed_quit_button) {
4          break;
5      }
6      ...
7      i++;
8  }
```

- ▶ Det gör det möjligt att använda "oändliga" loopar:

```
1  while (true) {
2      ...
3      if (user_pressed_quit_button) {
4          break;
5      }
6      ...
7      if (computation_is_done) {
8          break;
9      }
10     ...
11 }
```

Blank

for

En typisk while-loop

- ▶ Studera följande kodsnuitt:

```
1 i = 0;  
2  
3 while (i < n) {  
4     printf("%d ", i);  
5     i++;  
6 }
```

- ▶ Vad händer?

- ▶ Initieringen `i = 0` körs **först**
- ▶ **Sen** körs testet `i < n`
 - ▶ Om testet är **true** så körs **innehållet** i **loopen**
 - ▶ Det **sista** som körs inuti loopen är **uppräknningen** `i++`

- ▶ Vi kan skriva en sån här loop på ett **kompaktare** sätt: **for**

for-satsen

- ▶ C-syntaxen för en **for**-loop ser ut så här:

```
for ( [INIT] ; [TEST] ; [UPDATE] )  
    STATEMENT
```

eller hellre

```
for ( [INIT] ; [TEST] ; [UPDATE] ) {  
    STATEMENT1  
    STATEMENT2  
    ...  
}
```

- ▶ Koden `[INIT]` utförs **en** gång när **for**-satsen **startas**
- ▶ Varje varv **påbörjas** med att `[TEST]` evalueras
 - ▶ Om `[TEST]` returnerar **true** kommer den **blå** koden i huvudsatsen att köras
 - ▶ **Sist** i varje varv körs koden i `[UPDATE]`

while kontra for

- ▶ C-koden:

```
1 i = 0;  
2  
3 while (i < n) {  
4     printf("%d ", i);  
5     i++;  
6 }
```

är semantiskt **identisk** med följande C-kod:

```
1 for ( i = 0 ; i < n ; i++ ) {  
2     printf("%d ", i);  
3 }
```

Simulering av for-loop

- ▶ Ordningen för delarna i **for**-loopen är densamma som för **while**-loopen:
 - ▶ Init-test-(loop-update-test)-(loop-update-test)-...

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int i;
6
7     for (i = 0; i < 3; i++) {
8         printf("%d ", i);
9     }
10
11     printf("\nNormal exit.\n");
12     return 0;
13 }
```

0123 i

```
0 1 2
Normal exit.
```

Behövs for-satsen?

- ▶ Nej! (inte i C)
- ▶ En vanlig presentation är att den ungefär motsvarar den språkliga konstruktionen
 - ▶ för varje tal $i=0, 1, \dots, 5$
- ▶ I praktiken är **for**-satsen lämplig om vi vet hur **många gånger** vi vill upprepa något

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int i;
6
7     for (i = 0; i < 3; i++) {
8         printf("%d ", i);
9     }
10
11     printf("\nNormal exit.\n");
12     return 0;
13 }
```

- ▶ Koden blir **lättare att läsa** (om man förstår hur **for**-loopen fungerar)

for-satsen - loop-variabel (1)

- ▶ I **for**-satsen använder vi oftast en **loop-variabel** för att ha kontroll på antalet varv
- ▶ Om variabeln bara ska användas i loopen kan vi **flytta in** deklarationen inuti **for**-satsen:

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     for (int i = 0; i < 3; i++) {
6         printf("%d ", i);
7     }
8
9     printf("\nNormal exit.\n");
10     return 0;
11 }
```

- ▶ Fördelen med denna konstruktion är
 - ▶ Loop-konstruktionen blir **ännu kompaktare**
 - ▶ Loop-variabeln är bara synlig **inuti** loopen
 - ▶ Loop-variabeln **skräpar inte ner** resten av koden

for-satsen - loop-variabel (2)

- ▶ Om vi försöker använda loop-variabeln **efter** loopen kommer vi att få ett **kompileringsfel**

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     for (int i = 0; i < 3; i++) {
6         printf("%d ", i);
7     }
8
9     printf("%d ", i);
10
11     printf("\nNormal exit.\n");
12     return 0;
13 }
```

for-satsen – nedräkning

- ▶ Det går utmärkt att skriva en **for**-loop som **räknar ner**
- ▶ Vad skriver följande kod ut?

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      for (int i = 4; i > 0; i--) {
6          printf("%d ", i);
7      }
8      return 0;
9  }
10
```

- ▶ Utskrift: 4 3 2 1

for-satsen — val av tester

- ▶ Det är viktigt att ha koll på vilket test man använder:
 - ▶ < eller <= vid uppräkning?
 - ▶ > eller >= vid nedräkning?
- ▶ Vilket är det **första** och **sista** värdet i kommer att ha **inuti** loopen och hur många **varv** kommer att köras?

```
for (i = 0; i < 12; i++) {
    ...
}
```

- ▶ 0, 1, ..., 11 (12 varv)

```
for (i = 12; i > 0; i--) {
    ...
}
```

- ▶ 12, ..., 1 (12 varv)

```
for (i = 0; i <= 12; i++) {
    ...
}
```

- ▶ 0, 1, ..., 12 (13 varv)

```
for (i = 12; i >= 0; i--) {
    ...
}
```

- ▶ 12, ..., 0 (13 varv)

for-satsen - block

- ▶ Den sammansatta loop-satsen kan förstås innehålla **flera** satser:

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      for (int i = 0, sum = 0; i < 5; i++) {
6          sum += i;
7          printf("%d ", sum);
8      }
9
10     printf("\nNormal exit.\n");
11     return 0;
12 }
```

- ▶ Notera att vi kan deklarera och initiera **flera** variabler i **for**-satsen

Simulering

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      for (int i = 0, sum = 0; i < 5; i++) {
6          sum += i;
7          printf("%d ", sum);
8      }
9
10     printf("\nNormal exit.\n");
11     return 0;
12 }
```

```
013610 sum
012345 i
```

```
0 1 3 6 10
Normal exit.
```

for-satsen – uppdatering

- Uppdateringen av loop-variabeln kan vara **vad som helst**

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     printf("Feet Meters\n");
6     printf("-----\n");
7
8     for (int feet = 20 ; feet > 0 ; feet -= 5 ) {
9         double meters = 0.3048 * feet;
10        printf("%4d %6.2f\n", feet, meters);
11    }
12
13    printf("\nNormal exit.\n");
14    return 0;
15 }
```

- Utskrift:

```
Feet Meters
-----
20    6.10
15    4.57
10    3.05
5     1.52

Normal exit.
```

for-satsen – slutvärde

- Sista värdet för loop-variabeln behöver inte vara nära "stopp"-värdet

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     for (int i = 0; i < 20; i+=8) {
6         printf("%d ", i);
7     }
8
9     printf("\nNormal exit.\n");
10    return 0;
11 }
```

- Utskrift:

```
0 8 16

Normal exit.
```

for-satsen – icke-heltal som loop-variabel

- Loop-variabeln behöver inte vara ett heltal

```
1 for (double s = 1 ; s < 10 ; s *= 1.7) {
2     printf("%.5f\n", s);
3 }
```

- Utskrift:

```
1.00000
1.70000
2.89000
4.91300
8.35210
```

for-satsen – varning för flyttal

- Se upp med flyttal i villkor (ska 1.0 skrivas ut?):

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     for (double x = 0.0 ; x < 1.0 ; x += 0.1) {
6         printf("%.1f\n", x);
7     }
8
9     printf("\nNormal exit.\n");
10    return 0;
11 }
```

- Utskrift:

```
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

Normal exit.
```


- ▶ Det är oftast vettigt att ha med en **utskrift** av loop-variabeln vid debugging!

do-while

do-while-satsen

- ▶ Ibland är det önskvärt att utföra något **innan** villkoret beräknas
- ▶ Ett typfall är att kontrollera **indata** från användaren
- ▶ Då kan **do-while**-satsen vara användbar!
- ▶ C-syntaxen för en **do-while**-loop ser ut så här:

```
do {  
    STATEMENT1  
    STATEMENT2  
    ...  
} while ( TEST ) ;
```

- ▶ Först körs den **blå sammansatta satsen**
- ▶ Sen evalueras **TEST**
 - ▶ Om testet är **true** så körs loopen ett varv till
- ▶ Notera det avslutande semikolonet!

do-while-satsen — exempel

► Exempel:

```
1 int n;
2 do {
3     printf("Enter an integer in the range 0-100: ");
4     scanf("%d", &n);
5 } while ( ! (n >= 0 && n <= 100) );
6 printf("n = %d\n", n);
```

► Exempelkörning:

```
Enter an integer in the range 0-100: -5
Enter an integer in the range 0-100: 345
Enter an integer in the range 0-100: 34
n = 34
```

Ackumulera en summa — do-while

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int n, sum = 0; // Initialize sum
6
7     printf("Enter a positive integer to accumulate.\n");
8     printf("Enter -1 to quit.\n\n");
9
10    do {
11        printf("Enter number: ");
12        scanf("%d", &n);
13        if (n > 0) {
14            sum += n;
15        }
16    } while (n > 0);
17
18    printf("\nThe sum is %d.\n", sum);
19
20    printf("Normal exit.\n");
21    return 0;
22 }
```

Ackumulera en summa — while

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int n, sum;
6
7     printf("Enter a positive integer to accumulate.\n");
8     printf("Enter -1 to quit.\n\n");
9
10    sum = 0; // Initialize sum
11    n = 1; // To enter the loop
12    while (n > 0) {
13        printf("Enter number: ");
14        scanf("%d", &n);
15        if (n > 0) {
16            sum += n;
17        }
18    }
19    printf("\nThe sum is %d.\n", sum);
20
21    printf("Normal exit.\n");
22    return 0;
23 }
```

Ackumulera en summa — for

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int sum = 0; // Initialize sum
6
7     printf("Enter a positive integer to accumulate.\n");
8     printf("Enter -1 to quit.\n\n");
9
10    for (int n = 1; n > 0; ) {
11        printf("Enter number: ");
12        scanf("%d", &n);
13        if (n > 0) {
14            sum += n;
15        }
16    }
17    printf("\nThe sum is %d.\n", sum);
18
19    printf("Normal exit.\n");
20    return 0;
21 }
```

► Notera den **tomma** satsen i [UPDATE]-delen av **for**-satsen på rad 10

Vanliga fel (1)

- Glömma måsvingar {}

```
for (int i = 1, sum = 0 ; i <= 5 ; i++)  
    int j = i*i;  
    sum += j;  
printf("sum=%d\n",sum);
```

- Glömma semikolon i for-satsen

```
for (int i = 1, i <= n, i++)
```

- Semikolon efter slutparantesen för for och while

```
for (int i = 1, sum = 0 ; i <= 5 ; i++); {  
    sum += i*i;  
}  
printf("sum=%d\n",sum);
```

Blank

Vanliga fel (2)

- Tilldelning = i stället för likhetstest == i while-testet

```
while (i = 0) {  
    ...  
}
```

- Se upp med flyttal!

```
for (double x = 0.0; x < 1.0; x += 0.1) {  
    printf("%.1f\n", x);  
}
```

Blank

Nästlade loopar

Nästlade loopar

- Det är vanligt att man skriver en loop inuti en annan

```
1  for (int i = 0 ; i < 2 ; i++) {
2      for (int j = 0 ; j < 3 ; j++) {
3          printf("(%d,%d) ", i, j);
4      }
5      printf("\n");
6  }
```

- Det kallas att looparna är **nästlade**

Nästlade loopar, delar

- Delarna av den **yttre** **for**-satsen ser ut så här:

```
1  for ( int i = 0 ; i < 2 ; i++ ) {
2      for ( int j = 0 ; j < 3 ; j++ ) {
3          printf("(%d,%d) ", i, j);
4      }
5      printf("\n");
6  }
```

dvs. hela den inre **for**-loopen och rad 5 ligger inuti

- Delarna av den **inre** **for**-satsen ser ut så här:

```
1  for ( int i = 0 ; i < 2 ; i++ ) {
2      for ( int j = 0 ; j < 3 ; j++ ) {
3          printf("(%d,%d) ", i, j);
4      }
5      printf("\n");
6  }
```

dvs. endast **printf**-anropet på rad 3 ligger inuti

Simulering nästlad for-loop (1)

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      for (int i = 0; i < 2; i++) {
6          for (int j = 0; j < 3; j++) {
7              printf("(%d,%d) ", i, j);
8          }
9          printf("\n");
10     }
11
12     printf("\nNormal exit.\n");
13     return 0;
14 }
```

0x0123	j
0x012	i

```
(0,0) (0,1) (0,2)
(1,0) (1,1) (1,2)

Normal exit.
|
```

Initiering av inre nästlad loop

- ▶ Loop-variabeln i den yttre loopen kan användas för att styra antalet varv i den inre

```
for ( int outer = 0 ; outer < 3 ; outer++ ) {  
    for ( int inner = outer ; inner < 3 ; inner++ ) {  
        printf("*");  
    }  
    printf("\n");  
}
```

Simulering nästlad for-loop (2)

```
1  #include <stdio.h>  
2  
3  int main(void)  
4  {  
5      for (int outer = 0; outer < 3; outer++) {  
6          for (int inner = outer; inner < 3; inner++) {  
7              printf("*");  
8          }  
9          printf("\n");  
10     }  
11  
12     printf("\nNormal exit.\n");  
13     return 0;  
14 }
```

0123	inner
0123	outer

```
***  
**  
*  
  
Normal exit.  
|
```

Många loopar blir det...

- ▶ Tänk på att det totala antalet loop-varv ökar snabbt med antalet nästlade loopar

```
1  // 10 outermost loops  
2  for (int i = 0; i < 10; i++) {  
3      // 10*10=100 intermediate loops  
4      for (int j = 0; j < 10; j++){  
5          // 10*10*10=1000 innermost loops  
6          for (int k = 0; k < 10; k++) {  
7              printf("Bip: %2d %2d %2d\n", i, j, k);  
8          }  
9      }  
10 }
```

När använder man vilken loop?

- ▶ Välj den som är lättast att förklara!
- ▶ Tumregler:
 - ▶ for om man vet antalet varv
 - ▶ while annars
 - ▶ do-while om man ska köra minst en gång

Obligatorisk uppgift 1

- ▶ Uppgiften
- ▶ Strukturerad problemlösning