F02 - Läsbarhet och val

Programmeringsteknik med C och Matlab, 7,5 hp

Niclas Börlin
niclas.borlin@cs.umu.se

Datavetenskap, Umeå universitet

2023-09-29 Fre

Läsbarhet

Ett C-program: `plus_one.c`

- Betrakta detta — korrekta — C-program:

```
#include <stdio.h>
int main(void){int n;printf("Enter an integer >");scanf("%d",
&n);n=n+1;printf("The next higher integer is: %d.\n"
, n);printf("Normal exit.\n");return 0;}
```


Ett C-program: `plus_one.c`

- Tydligare med vettiga radbrytningar, indentering, kommentarer och blankrader

```
#include <stdio.h>

int main(void)
{
    int n;

    /* Read an integer from the user */
    printf("Enter an integer >");
    scanf("%d", &n);

    /* Increment the number */
    n = n + 1;

    /* Output the new number */
    printf("The next higher integer is: %d.\n", n);

    printf("Normal exit.\n");
    return 0;
}
```

Ett C-program: `plus_one.c`

- Ännu tydligare med färg... (och radnummer är bra om man ska prata om koden)

```c
#include <stdio.h>

int main(void)
{
    int n;

    /* Read an integer from the user */
    printf("Enter an integer >");
    scanf("%d", &n);

    /* Increment the number */
    n = n + 1;

    /* Output the new number */
    printf("The next higher integer is: %d.\n", n);

    printf("Normal exit.\n");
    return 0;
}
```


Läsbarhet

- Indentering
- Kommentarer
- Val av variabelnamn
- Var inte rädd för ”luft” i koden!
- Du skriver för dig själv, inte bara för kompilatorn!

Indentering (1)

- Jämför denna kod...

```c
if (day == 1) { // Monday
if (hour <= 7) {
printf("Double espresso?\n");
} else {
printf("Enjoy work!\n");
}
} else if (day <= 5) { // Friday==5
printf("Have a nice workday!\n");
} else {
printf("Yohoo, weekend!\n");
}
```

- Varje öppningsmåsvinge { ska ha en motsvarande stängningsmåsvinge }...svårt att hitta...

Indentering (2)

- ...med denna

```c
if (day == 1) { // Monday
    if (hour <= 7) {
        printf("Double espresso?\n");
    } else {
        printf("Enjoy work!\n");
   }
} else if (day <= 5) { // Friday==5
    printf("Have a nice workday!\n");
} else {
    printf("Yohoo, weekend!\n");
}
```

- Även om ni inte (än) förstår vad koden gör så lyfter indenteringen fram kodens struktur

Programstruktur

- C-kompilatorn kräver inte så mycket
- För vår skull gör vi
  - indentering,
  - ny rad,
  - mellanrum,
  - blankrader,
  - bra namn på variabler och funktioner,
  - kommentarer
- Viktigt — vi kommer vara petiga!
- I större program: Kodstandard
  - `isValid` eller `is_valid`?
    - eller `isvalid`?
    - eller bara `valid`?
  - `read_variable` eller `variable_read`?
  - korksmurf eller smurfskruv?

Blank

Kontrollstrukturer

Kontrollstruktur

- En kontrollstruktur är en kombination av individuella instruktioner som ses som en logisk enhet med en väg in och en väg ut
- Språket C har tre typer
  - Sekvens — Block, sammansatt sats (inom måsvingar)
    - Redan sett
  - Selektion — Välj en av flera vägar
    - Idag
  - Repetition —– Upprepa en sats eller ett block
    - Senare

Selektion

Selektion

- Vi behöver kunna uttrycka olika vägar i koden
  - Konstruktioner i C: `if`, `if-else`, `switch`
- Koden behöver kunna välja mellan vägarna
  - Vi behöver villkor

Villkor — Boolska test

- Döpta efter den engelske matematikern George Boole (1815–1864)
- Är uttryck som returnerar ett boolskt värde, dvs antingen sant (`true`) eller falskt (`false`)
  - Kallas också för *logiska* värden
- Använder sig oftast av villkorsoperatorer

Villkorsoperatorer

- De vanligaste villkorsoperatorerna:

`<`	mindre än
`>`	större än
`<=`	mindre än eller lika med
`>=`	större än eller lika med
`==`	likhet
`!=`	olikhet

- Observera att operatorn "likhet" i C skrivs med ett dubbelt likhetstecken (==)
  - En enkelt likhetstecken (=) i C betyder tilldelning
- Om `m` och `n` är variabler av samma typ, så returnerar uttrycket `n > m` värdet `true` om värdet i `n` är större än värdet i `m`, annars `false`

if-satsen

- Den enklaste konstruktionen för att uttrycka ett val i C är `if`-satsen:

```
if (EXPRESSION) {
    STATEMENT
}
```

  - Om uttrycket i grönt är sant så körs den blå sammansatta satsen, annars körs ingenting

- Exempel:

```
if (hour <= 12) {
    printf("Good morning!\n");
}
```

  - Om villkoret `hour <= 12` är uppfyllt (`true`) så utförs satsen

```
printf("Good morning!\n");
```

  - Annars händer ingenting

if-satsen och beslut

- Studera `if`-satsen:

```
1 if (hour <= 12) {
2     printf("Good morning!\n");
3 }
```

- När testet på rad 1 körts tas ett beslut:
  - Om testet är `true` så:
    1. kör vi den blå *then*-satsen
  - Om testet är `false` så:
    1. hoppar vi över den blå *then*-satsen
- I bägge fallen fortsätter sedan exekveringen efter `if`-satsen, dvs. på rad 4 och framåt

if-satsen (2)

- Allt mellan måsvingarna är en sammansatt sats, så om vi vill ha mer än en sats utförd:

```
if (EXPRESSION) {
    STATEMENT1
    STATEMENT2
    ...
}
```

- Exempel

```
if (hour <= 12) {
    printf("Good morning!\n");
    printf("Do you want some coffee?\n");
}
```


Varning!

- Kompilatorn kräver inte måsvingar!
- Följande är en giltig `if`-sats:

```
if (EXPRESSION)
    STATEMENT
```

- Det betyder att vi kan skriva

```
if (hour <= 12)
    printf("Good morning!\n");
```

vilket verkar bra tills vi inser att

```
if (hour <= 12)
    printf("Good morning!\n");
    printf("Do you want some coffee?\n");
```

tolkas av kompilatorn som

```
if (hour <= 12) {
    printf("Good morning!\n");
}
printf("Do you want some coffee?\n");
```

Slutsats:

- Använd alltid måsvingar!
  - Ökar läsbarheten
  - Minskar risk för fel

`if-else`-satsen

- Vill vi göra det ena eller det andra kan vi använda `if-else`-satsen:

```
if (EXPRESSION) {
    THEN-EXPRESSION
} else {
    ELSE-EXPRESSION
}
```

  - Om uttrycket i grönt är sant så körs de blå satserna, annars körs de röda
- Exempel:

```
if (hour <= 12) {
    printf("Good morning!\n");
} else {
    printf("Good afternoon!\n");
}
```


`if-else`-satsen och beslut

- Studera `if-else`-satsen:

```
1 if (hour <= 12) {
2     printf("Good morning!\n");
3 } else {
4     printf("Good afternoon!\n");
5 }
```

- När testet på rad 1 körts tas två beslut:
  - Om testet är `true` så:
    1. kör vi den blå *then*-satsen
    2. hoppar vi över den röda *else*-satsen
  - Om testet är `false` så:
    1. hoppar vi över den blå *then*-satsen
    2. kör vi den röda *else*-satsen
- I bägge fallen fortsätter sedan exekveringen efter `if-else`-satsen, dvs. på rad 6 och framåt

Kedjade `if`-satser

Kedjade `if`-satser

- Vi kan skriva flera `if`-satser i en kedja efter varandra:

```
1  if (hour <= 12) {
2      printf("Good morning!\n");
3  } else if (hour <= 18) {
4      printf("Good afternoon!\n");
5  } else if (hour <= 22) {
6      printf("Good evening!\n");
7  } else {
8      printf("Good night!\n");
9  }
```

- Det första testet som blir `true` avgör vilken `printf`-sats som körs
- `if`-satsen har en väg in (rad 1) och en väg ut (rad 10)

Kedjade `if`-satser, beslut 1

- När testet på rad 1 körts tar vi två beslut:
  - Om testet är `true` så:
    1. kör vi den blå *then*-satsen
    2. hoppar vi över den röda *else*-satsen
  - Om testet är `false` så:
    1. hoppar vi över den blå *then*-satsen
    2. kör vi den röda *else*-satsen

```
1  if (hour <= 12) {
2      printf("Good morning!\n");
3  } else if (hour <= 18) {
4      printf("Good afternoon!\n");
5  } else if (hour <= 22) {
6      printf("Good evening!\n");
7  } else {
8      printf("Good night!\n");
9  }
```

- I bägge fallen fortsätter vi sedan exekveringen efter `if`-satsen, dvs. på rad 10 och framåt

Kedjade `if`-satser, beslut 2

- Om det första testet var `false` så kör vi nedanstående sats:

```
1  if (hour <= 12) {
2      printf("Good morning!\n");
3  } else if (hour <= 18) {
4      printf("Good afternoon!\n");
5  } else if (hour <= 22) {
6      printf("Good evening!\n");
7  } else {
8      printf("Good night!\n");
9  }
```

- Då detta också är en `if-else`-sats så har vi motsvarande beslut att ta:
  - Köra den blå *then*-satsen och hoppa över den röda *else*-satsen

  eller
  - hoppa över den blå *then*-satsen och köra den röda *else*-satsen
- I bägge fallen fortsätter vi sedan exekveringen efter `if`-satsen, vilket fortfarande är på rad 10 och framåt

Kedjade `if`-satser, beslut 3

- Om också det andra testet var `false` så kör vi nedanstående sats:

```
1  if (hour <= 12) {
2      printf("Good morning!\n");
3  } else if (hour <= 18) {
4      printf("Good afternoon!\n");
5  } else if (hour <= 22) {
6      printf("Good evening!\n");
7  } else {
8      printf("Good night!\n");
9  }
```

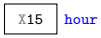
- Då detta också är en `if-else`-sats så har vi motsvarande beslut att ta:
  - Köra den blå *then*-satsen och hoppa över den röda *else*-satsen

  eller
  - hoppa över den blå *then*-satsen och köra den röda *else*-satsen
- I bägge fallen fortsätter vi sedan exekveringen efter `if`-satsen, vilket fortfarande är på rad 10 och framåt

Simulering `hour = 15`

```c
#include <stdio.h>

int main(void)
{
    int hour = 15;
    if (hour < 12) {
        printf("Good morning!\n");
    } else if (hour < 18) {
        printf("Good afternoon!\n");
    } else {
        printf("Good evening!\n");
    }
    printf("Normal exit.\n");
    return 0;
}
```




```
Good afternoon!
Normal exit.
```


Blank

Nästlade `if`-satser

Nästlade `if`-satser (1)

- Det finns inget som hindrar att satsen inuti en `if`-sats är en till `if`-sats:

```c
1  if (day == 1) { // Monday
2      if (hour <= 7) {
3          printf("Double espresso?\n");
4      } else {
5          printf("Enjoy work!\n");
6      }
7  } else if (day <= 5) { // Friday==5
8      printf("Have a nice workday!\n");
9  } else {
10     printf("Yohoo, weekend!\n");
11 }
```

- Det kallas för nästlade `if`-satser
- `if`-satsen har en väg in (rad 1) och en väg ut (rad 12)

Nästlade if-satser (2)

- Den första `if`-satsen har dessa delar:

```
1  if (day == 1) {
2      if (hour <= 7) {
3          printf("Double espresso?\n");
4      } else {
5          printf("Enjoy work!\n");
6      }
7  } else if (day <= 5) {
8      printf("Have a nice workday!\n");
9  } else {
10     printf("Yohoo, weekend!\n");
11 }
```

- Om testet blir `false` så tar vi beslutet:
  1. Hoppa över den blå satsen
  2. Kör den röda satsen

Nästlade if-satser (3)

- . . . och hamnar här:

```
1  if (day == 1) {
2      if (hour <= 7) {
3          printf("Double espresso?\n");
4      } else {
5          printf("Enjoy work!\n");
6      }
7  } else if (day <= 5) {
8      printf("Have a nice workday!\n");
9  } else {
10     printf("Yohoo, weekend!\n");
11 }
```

- Den sats vi tagit beslut om att hoppa över har blivit gråad
- Vi har en ny `if`-sats på rad 7–11 att ta ställning till
- När vi är klar med den `if`-satsen fortsätter vi på rad 12

Nästlade if-satser (4)

- Så här såg den första `if`-satsen ut:

```
1  if (day == 1) {
2      if (hour <= 7) {
3          printf("Double espresso?\n");
4      } else {
5          printf("Enjoy work!\n");
6      }
7  } else if (day <= 5) {
8      printf("Have a nice workday!\n");
9  } else {
10     printf("Yohoo, weekend!\n");
11 }
```

- Om testet blir `true` så tar vi beslutet:
  1. Kör den blå satsen
  2. Hoppa över den röda satsen

Nästlade if-satser (5)

- . . . och hamnar här:

```
1  if (day == 1) {
2      if (hour <= 7) {
3          printf("Double espresso?\n");
4      } else {
5          printf("Enjoy work!\n");
6      }
7  } else if (day <= 5) {
8      printf("Have a nice workday!\n");
9  } else {
10     printf("Yohoo, weekend!\n");
11 }
```

- Den sats vi tagit beslut om att hoppa över har blivit gråad
- Vi har en ny `if`-sats på rad 2–6 att ta ställning till
- När vi är klar med den `if`-satsen fortsätter vi på rad 12, precis som i det andra fallet

Simulering `day = 4, hour = 9`

```c
#include <stdio.h>

int main(void)
{
















}
```

X4	day
X9	hour

```
Have a nice workday!
Normal exit.
```


Simulering `day = 1, hour = 9`

```c
#include <stdio.h>

int main(void)
{
















}
```

X1	day
X9	hour

```
Enjoy work!
Normal exit.
```


Validering av data

- En vanlig tillämpning av val är validering av indata

```c
printf("How many people are going to the cinema? ");
scanf("%d", &n);
if (n <= 0) {
    printf("Bad number.\n");
} else if (n > 5) {
    printf("Please contact the group reservation for more than 5 tickets.\n");
} else {
    printf("So, what movie do you want to see?\n");
    ...
}
```


Kombinationer av Boolska test

- Ibland vill vi uttrycka kombinationer av Boolska uttryck
- Det åstadkommer vi med hjälp av de Boolska operatorerna OCH (`&&`), ELLER (`||`) och ICKE (`!`)

```c
if (month == 12 && day == 24) {
   printf("Christmas eve!\n");
}
if (month == 6 || month == 7) {
   printf("Summer!\n");
}
if ( ! (temperature >= 0) ) {
   printf("It's freezing!\n");
}
```

- Notera dubbeltecknen för OCH och ELLER!
  - Det finns andra operatorer (`&` och `|`) som fungerar annorlunda

Sanningstabell

- Operatorn `&&` (OCH) är sann om bägge operanderna är sanna
- Operatorn `||` (ELLER) är sann om någon av operanderna (eller bägge) är sanna

p	q	p && q	p || q
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

- Operatorn `!` (ICKE) är sann om operanden är falsk

p	!p
true	false
false	true

Sammansatta boolska uttryck

- Vad gäller om vi har flera boolska uttryck?

```
if (a == c || b <= a || 2 != b) {
    ...
}
```

- ...eller om vi kombinerar olika operatorer?

```
int a = 1;
int b = 2;
int c = a;
if (a == c || b <= a && 2 != b) {
    printf("Is this printed...\n");
}
if ((a == c || b <= a) && 2 != b ) {
    printf("...or this?\n");
}
```

  - Vilken evalueras först, `&&` eller `||`?

Prioritetsordning (1)

- Om flera operatorer har olika prioritet utförs operatorerna med högst prioritet först
- Detta är några av de vanligaste operatorerna i C:

Prioritet	Operator	Beskrivning
Högst	()	parenteser
	!	"ICKE" (NOT)
	*, /, %	multiplikation, division, modulo
	+, -	addition, subtraktion
	<, <=, >, >=	relationsoperatorer
	==, !=	likhet, olikhet
	&&	logiskt "OCH" (AND)
	||	logiskt "ELLER" (OR)
Lägst	=	tilldelning

Prioritetsordning (2)

- Om flera operatorer har samma prioritet evalueras de flesta från vänster till höger
- Ett undantag är tilldelningsoperatorn = som evalueras från höger till vänster!
  - Det gör det möjligt att skriva `i = j = 1;`

Nästlad `if` och `&&`

- Nästlade `if`-satser motsvarar `&&`-uttryck
- Följande kod

```
if (month == 12) {
    if (day == 24) {
        printf("Christmas eve!\n");
    }
}
```

gör samma sak som

```
if (month == 12 && day == 24) {
    printf("Christmas eve!\n");
}
```

- Vilken kontruktion som är bäst beror på sammanhanget

Boolska värden i C (1)

- Från början hade C ingen egen datatyp för boolska värden
  - I stället användes `int`
    - Värdet 0 representerar falskt
    - Alla andra värden representerar sant
- Det öppnar för ett vanligt nybörjarfel:
  - Att skriva

```
if (n = 5) {
    ....
}
```

när man menar

```
if (n == 5) {
    ....
}
```

  - Uttrycket `n = 5` är en tilldelningssats, som har värdet hos den tilldelade variabeln
    - Här blir värdet alltid 5, vilket tolkas som `true`
- Lyckligtvis så varnar `gcc` för detta misstag:

```
warning: suggest parentheses around assignment used as truth value [-Wparentheses]
```


Boolska värden i C (2)

- Sedan standarden c99 finns datatypen `bool` och sanningsvärdena `true` och `false`
- Kod som använder dessa behöver skriva följande i början av källkodsfilen:

```
#include <stdbool.h>
```

- Ni ska använda datatypen `bool` om ni behöver boolska variabler

Boolska variabler

- Rätt använt kan boolska variabler göra koden tydligare:

```
1  #include <stdio.h>
2  #include <stdbool.h>
3
4  int main(void)
5  {
6      ...
7      bool is_weekday = day >= 1 && day <= 5;
8      bool is_weekend = ! is_weekday;
9      bool is_christmas = month == 12 && (month_day >= 24 && month_day <= 26);
10     bool is_new_years = (month == 12 && month_day == 31) ||
11                         (month == 1 && month_day == 1);
12     bool has_presents = money_to_spend >= cost_of_lego;
13     ...
14
15     if (is_christmas && has_presents) {
16         printf("Ho-ho-ho\n");
17     }
18
19     if (is_new_years) {
20         printf("Happy new year!\n");
21     }
22
23     if (is_weekday && hour >= 6) {
24         printf("Time to get out of bed!\n");
25     }
26
27     ...
28 }
```

Vad *kan* användas som villkor?

- Tänk på att vilket uttryck som helst kan användas som ”villkor”:

```c
float x = 3.2;
int year = 2023;

if (x) { // No warning, better to write x != 0
    printf("x is non-zero!\n");
}

if (year % 4) { // Better to write (year % 4 != 0)
    printf("%d is not a leap year.\n", year);
}
```


Blank

Blank

Flervägsval

Flervägsval

- Antag att vi har en heltalsvariabel `i` som ska styra programflödet
- Antag vidare att `i` kan anta 5 olika giltiga värden 1, 2, . . . , 5

```c
int i;

printf("Enter an integer in the range 1-5 > ");
scanf("%d", &i);

if (i == 1) {
    printf("Hello!\n");
} else if (i == 2) {
    printf("Howdy!\n");
} else if (i == 3) {
    printf("G'day!\n");
} else if (i == 4) {
    printf("Hi!\n");
} else if (i == 5) {
    printf("Na worries!\n");
} else {
    printf("Ooops!\n");
}
```


Simulering

```c
#include <stdio.h>

int main(void)
{
    int i;

    printf("Enter an integer in the range 1-5 > ");
    scanf("%d", &i);

    if (i == 1) {
        printf("Hello!\n");
    } else if (i == 2) {
        printf("Howdy!\n");
    } else if (i == 3) {
        printf("G'day!\n");
    } else if (i == 4) {
        printf("Hi!\n");
    } else if (i == 5) {
        printf("Na worries!\n");
    } else {
        printf("Ooops!\n");
    }

    printf("Normal exit.\n");
    return 0;
}
```

X3 i

```
Enter an integer in the range 1-5 > 3
G'day!
Normal exit.
```


`switch`-satsen

- Ett smidigare sätt är att använda en `switch`-sats

```c
int i;

printf("Enter an integer in the range 1-5 > ");
scanf("%d", &i);

switch (i) {
    case 1:
        printf("Hello!\n");
        break;
    case 2:
        printf("Howdy!\n");
        break;
    case 3:
        printf("G'day!\n");
        break;
    case 4:
        printf("Hi!\n");
        break;
    case 5:
        printf("Na worries!\n");
        break;
    default:
        printf("Ooops!\n");
        break;
}
```


Simulering

```c
#include <stdio.h>

int main(void)
{
    int i;

    printf("Enter an integer in the range 1-5 > ");
    scanf("%d", &i);

    switch (i) {
    case 1:
        printf("Hello!\n");
        break;
    case 2:
        printf("Howdy!\n");
        break;
    case 3:
        printf("G'day!\n");
        break;
    case 4:
        printf("Hi!\n");
        break;
    case 5:
        printf("Na worries!\n");
        break;
    default:
        printf("Ooops!\n");
        break;
    }

    printf("Normal exit.\n");
    return 0;
}
```

X3 i

```
Enter an integer in the range 1-5 > 3
G'day!
Normal exit.
```

break-satsen

- Varför behövs `break`?
- Programflödet hoppar till det fall (*case*) som matchar det aktuella värdet på variabeln
- Därefter utförs alla satser som följer detta `case`
- Vi måste uttryckligen tala om för kompilatorn när den ska avbryta
  - Detta görs med `break`-satsen

Simulering (break saknas)

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int i;
6
7      printf("Enter an integer in the range 1-5 > ");
8      scanf("%d", &i);
9
10     switch (i) {
11     case 1:
12         printf("Hello!\n");
13     case 2:
14         printf("Howdy!\n");
15     case 3:
16         printf("G'day!\n");
17     case 4:
18         printf("Hi!\n");
19     case 5:
20         printf("Na worries!\n");
21     default:
22         printf("Ooops!\n");
23     }
24
25     printf("Normal exit.\n");
26     return 0;
27 }
```

```
Enter an integer in the range 1-5 > 3
G'day!
Hi!
Na worries!
```

Mer om case-satsen

- `case`-variabeln kan bara vara en heltalsdatatyp (tex `int`) eller `char`
- `switch`-satsen använder sig av en sammansatt sats (måsvingar)
- Satser efter respektive `case` behöver/bör oftast inte vara inramade av måsvingar {}
- Exekvering forsätter tills ett `break` stöts på eller `switch`-satsen tar slut
- Kom ihåg att använda `default` för att fånga upp alla andra värden än de i `case`-satserna