F01 - Introduktion, datorer och lite C

Programmeringsteknik med C och Matlab, 7,5 hp

Niclas Börlin
niclas.borlin@cs.umu.se

Datavetenskap, Umeå universitet

2023-09-28 Tor

Då kör vi igång...

Vad är ett datorsystem?

- Grov indelning:
  - Hårdvara Fysiska saker, dvs. plast, koppar, kisel, ...
  - Mjukvara Program som gör det möjligt att lösa uppgifter med en dator genom att förse datorn med listor av instruktioner

Datorns hårdvara

- De två viktigaste delarna:
  - Processorn
    - CPU (*Central Processing Unit*)
  - Minnet
    - ROM (*Read Only Memory*) — läsminne
    - RAM (*Random Access Memory*) — läs/skrivminne

Minnet (1)

- Datorns minne kan ses som en numrerad sekvens av celler
- En cells nummer = cellens adress
- Alla celler är lika stora och kan lagra ett bestämt antal bitar (*binary digit* — *bit*)
- En enskild bit kan lagra värdet 0 eller 1
- Vanligtvis grupperas bitarna i grupper om minst 8 bitar — en byte

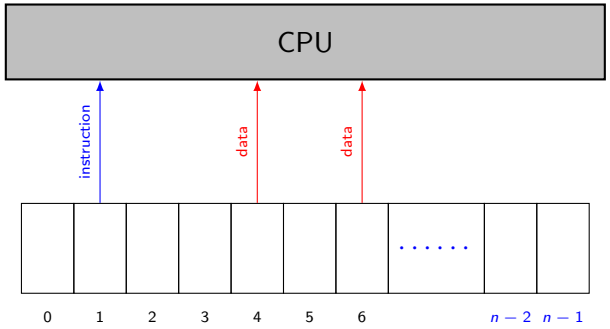
9288	?	?	?	?	?	?	?	?
9287	?	?	?	?	?	?	?	?
9286	?	?	?	?	?	?	?	?
9285	?	?	?	?	?	?	?	?
9284	0	0	0	0	0	1	1	0
9283	0	0	0	0	0	0	0	0
9282	0	0	0	0	0	0	0	0
9281	0	0	0	0	0	0	0	0
9280	0	0	1	0	1	1	0	0
9279	0	0	0	0	0	0	0	1
9278	0	1	1	0	0	0	0	1

Minnet (2)

- En cells innehåll är alltså en sekvens av bitar
- Vi kan välja att tolka denna sekvens antingen som
  - data vi behöver till en beräkning, eller som
  - en instruktion till processorn
- Alla celler innehåller information
  - Antingen meningsfull...
  - ...eller skräp

Processorn

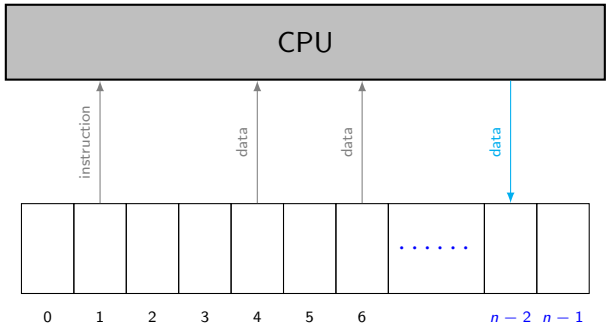
- Processorn läser instruktioner och data från minnet



- Instruktionerna säger vilka operationer den ska utföra på de data den läser
  - Ex. addition, multiplikation, etc.

Processorn (forts)

- Processorn kan sedan skriva resultatet av operationen tillbaka till minnet



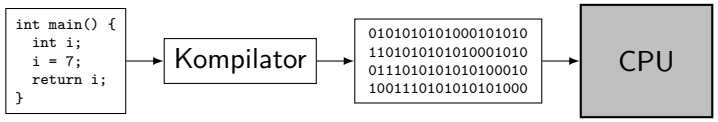
Från högnivåspråk till maskinkod (1)

- Processorn läser sekvenser av instruktioner, dvs program
- Det är i princip möjligt att skriva sekvenserna för hand, ladda in dem i datorns minne och köra programmet

Koden till *Apollo Guidance Computer* med Margaret Hamilton — *lead flight software designer*

Från högnivåspråk till maskinkod (2)

- Ett högnivåspråk låter programmeraren använda sig av algebraiska uttryck, engelska ord, etc., vilket gör programmen kortare och mer läsarvänliga
  - Uttrycket `3 + 5` är mer läsbart än `A9 05 69 03`
- Det färdiga programmet i ett högnivåspråk kan sedan köras genom en kompilator som översätter det till maskinkod, specifik för den maskinen



- Den kompilator vi kommer att använda på den här kursen heter `gcc` (*GNU Compiler Collection*)

Lite bakgrund kring C...

Programmeringsspråket C

- Utvecklat 1972 av Dennis Ritchie för att användas tillsammans med operativsystemet UNIX, en föregångare till Linux, MacOS och Android
- C är ett imperativt språk
  - Bygger på stegvisa instruktioner av vad som ska göras
- C är ett högnivåspråk, men ligger "nära" maskinen
  - Programmeraren har möjlighet att direkt komma åt minne och andra enheter på datorn som döljs i andra språk
  - C översätts lätt till maskinkodsinstruktioner
- C är ett av de populäraste programmeringsspråken i verkligheten
  - Flera andra språk har utvecklats ur C, t.ex. C++, C#

Vårt första C-program

- *Hello, world* — en klassiker när man provar ett nytt programspråk
- I C ser koden ut så här:

```
#include <stdio.h>
int main(void)
{
    printf("Hello world!\n");
    return 0;
}
```


Färgkodning

- På den här kursen kommer jag att i huvudsak att presentera C-kod som är färglagd för att öka förståelsen
- Vår kod blir då:

```c
#include <stdio.h>
int main(void)
{
    printf("Hello world!\n");
    return 0;
}
```

- Olika element i språket får olika färger
- Många editorer har liknande funktioner

Strukturerad problemlösning (1)

- Ett huvudsyfte med kursen är att lära er strukturerad problemlösning
- Det innebär att du löser uppgifter i tre steg:
  1. FÖRST förstå och analysera problemet
     - Hur ser indata ut?
     - Hur ska utdata se ut?
     - Var är relationen mellan in- och utdata?
  2. SEN designa en lösning
     - lösningsidé,
     - algoritm,
     - testa lösningen på papper
  3. Och SIST...
     - Implementera lösningen i C-kod
     - Testa koden

Strukturerad problemlösning (2)

- Under kodningen...
  - Träna på olika konstruktioner fristående
  - Skriv små testprogram som testar delar av din lösning
  - Övningsuppgifterna kan vara till hjälp
- Om du kör fast i kodningen kan det vara bra att backa från tangentbordet

Övningsuppgift

- Skriv ett program som läser in ett heltal från användaren, lägger till 1 och skriver ut resultatet

Steg 1 — Förstå

- Indata: Ett heltal
- Utdata: Ett annat heltal
- Relation: Utdatat är heltalet som följer närmast efter indata

Steg 2a — Design

- Lösningsidé:
  - Vi behöver få reda på vilket tal som användaren vill lägga 1 till, sedan lägga till 1 och skriva ut resultatet

Steg 2b — Design

- Algoritm:
  1. Läs in ett heltal från användaren
  2. Lägg till 1 till talet
  3. Skriv ut det nya talet

Steg 3 – Skriv kod

f01-plus-one.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int n;
6
7      /* Read an integer from the user */
8      printf("Enter an integer >");
9      scanf("%d", &n);
10
11     /* Increment the number */
12     n = n + 1;
13
14     /* Output the new number */
15     printf("The next higher integer is: %d.\n", n);
16
17     printf("Normal exit.\n");
18     return 0;
19 }
```


f01-plus-one.c — inramning

f01-plus-one.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int n;
6
7      /* Read an integer from the user */
8      printf("Enter an integer >");
9      scanf("%d", &n);
10
11     /* Increment the number */
12     n = n + 1;
13
14     /* Output the new number */
15     printf("The next higher integer is: %d.\n", n);
16
17     printf("Normal exit.\n");
18     return 0;
19 }
```

- En c-fil måste innehålla vissa bestämda saker i början och slutet (rad 1–4, 17–19)
- Vi kommer att förklarar detaljerna senare
- Inledningsvis kan ni kopiera de raderna rakt av
- Vi ska i stället fokusera på koden däremellan

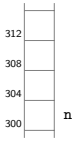
Variabler

Variabler och minnet (1)

- Värden som programmet behöver arbeta med lagras i variabler
- En variabels värde kan förändras under exekveringen av ett program
- En variabel kan ses som en låda i minnet med en speciell position (adress), ett namn samt en storlek (typ)
- I illustrationen till höger har `n` fått adressen 300 och storleken 4 bytes
  - Den ligger alltså på adress 300, 301, 302 och 303
- När nuvarande värde av variabeln `n` behövs så görs en avläsning från minnesplatserna 300–303
- När ett värde lagras i `n` skrivs detta värde till minnesplatserna 300–303

Variabler och minnet (2)

- Den exakta adressen (här: 300) och storleken (här: 4) för en variabel avgörs av kompilatorn och är oftast ointressant för programmeraren



Variabelnamn

- Variabelnamn får inte se ut hur som helst
- Godkända tecken:
  - Stora och små bokstäver, "a"–"z", "A"–"Z"
  - Siffror, "0"–"9"
  - understrykningstecknet, "_" (*underscore*)
- Ogiltiga tecken:
  - Allt annat, t.ex. mellanslag, ääö, plustecknet, etc.
- Men namnet...
  - ...får inte börja med en siffra
  - ...får inte vara ett av C:s reserverade ord (`return`, `void`, etc.)
  - ...bör inte vara definierat i något av C:s standardbibliotek (t.ex. `printf`)
- Namn i C är s.k. *skiftlägeskänsliga* (*case sensitive*)
  - Det betyder att små och stora bokstäver motsvarar olika variabelnamn
  - Variabelnamnen `n1` och `N1` är olika

Variabeldeklarationer

- Innan vi använder variabler i vår C-kod måste vi deklarera dem
  - Vid deklarationen anger vi namnet och vilken typ av data variabeln ska innehålla
- God metodik:
  - Använd variabelnamn som hjälper dig och andra att komma ihåg vad variabeln används till
    - Jämför `first_free_position` med `index` med `p`
    - Jämför `month_name` med `name` med `m`
  - Om korta variabelnamn kan användas utan att förståelsen försämras, gör det!
    - `pos` kan vara bättre än `position`
    - `i` kan vara bättre än `index`
  - Sammanhanget avgör om `s`, `sec`, `seconds` eller `seconds_until_launch` är bästa variabelnamnet

Datatyper

- Beskriver en viss typ av data, t.ex. heltal, tecken, eller flyttal
  - Ett heltal kan lagra ett heltal av en viss storlek
  - Ett flyttal kan lagra s.k. decimaltal
- I C finns ett antal inbyggda datatyper, t.ex. `int`, `char`, `bool`, `float`, `double`
  - Det går att skapa egna — senare!
- Källkoden behöver beskriva vilken typ ett visst data har för att kompilatorn ska veta...
  - hur mycket minnesutrymme vår variabel behöver
  - vilka operationer som kan användas för att manipulera variablerna
- Till exempel:
  - en `int` kräver 4 bytes på de flesta datorsystem
  - operationerna plus (+) och minus (-) är definierade för en `int`, men inte indexering (`[]`)

f01-plus-one.c — variabeldeklaration

f01-plus-one.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int n;
6
7      printf("Enter an integer >");
8      scanf("%d", &n);
9
10     /* Increment the number */
11     n = n + 1;
12
13     /* Output the new number */
14     printf("The next higher integer is: %d.\n", n);
15
16     printf("Normal exit.\n");
17     return 0;
18 }
19
```

- Rad 5 innehåller deklarationen av en variabel med namn `n` av typen `int`

Satser

- Instruktionerna i C är indelade i satser (*statements*)
- Det finns enkla och sammansatta satser
- En enkel sats avslutas alltid med semikolon `;`, t.ex.

```
printf("Hello, world!\n");
```

- En sammansatt sats skrivs mellan måsvingar `{}` (*braces*, *curly braces*)
  - Innehållet i en sammansatt sats är en sekvens av enkla och/eller sammansatta satser, t.ex.

```
{
    printf("Hello, ");
    printf("world!\n");
}
```

  - En sammansatt sats ses som en enhet, dvs hela sekvensen av satser inom `{` och `}` kommer att utföras

f01-plus-one.c — satser

f01-plus-one.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int n;
6
7      /* Read an integer from the user */
8      printf("Enter an integer >");
9      scanf("%d", &n);
10
11     /* Increment the number */
12     n = n + 1;
13
14     /* Output the new number */
15     printf("The next higher integer is: %d.\n", n);
16
17     printf("Normal exit.\n");
18     return 0;
19 }
```

- Rad 8 innehåller en enkel sats
- Raderna 4–19 innehåller en sammansatt sats

Kommentarer (1)

- Kommentarer är text som finns för att göra koden lättare att förstå för människor
- Kompilatorn ignorerar kommentarer
- Bra kommentarer gör det möjligt att få en känsla för vad koden gör utan att förstå koden

```
/* Read an integer from the user */
...
/* Increment the number */
...
/* Output the new number */
```

- Om koden innehåller fel kan bra kommentarer ge ledtrådar till vad programmeraren tänkt sig att koden ska göra

Kommentarer (2)

- En kommentar kan skrivas in på två sätt:
- En kommentar som skrivs mellan `/*` och `*/` kan sträcka sig över flera rader

```
/*
  This comment
  extends through
  many lines.
*/
```

- En kommentar som påbörjas med tecknen `//` sträcker sig till slutet av raden

```
// Read an integer from the user
...
// Increment the number
...
// Output the new number
```

```
// This is how you write multi-line
// comments as multiple single-line
// comments.
```


Funktionsanrop

- En funktion i C är kod — en grupp av satser — som är sammansatt för att lösa ett specifikt problem
- Ett stort antal funktioner "följer med" språket C i dess s.k. standardbibliotek
- Ett funktionsanrop i C skrivs som funktionsnamnet följt av parametrar inom paranteser
  - Parametrarna separeras av kommatecken: `,`
  - En viktig funktion heter `printf` (*print*, *formatted*) som skriver ut text till användaren
  - En annan viktig funktion heter `scanf` (*scan*, *formatted*) som läser in svar från användaren
- Bägge funktionerna tar parametrar som är strängar
  - En sträng är en sekvens av tecken och skrivs inom citationstecken: `"Hello, world"`

f01-plus-one.c — printf

f01-plus-one.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int n;
6
7      /* Read an integer from the user */
8      printf("Enter an integer >");
9      scanf("%d", &n);
10
11
12
13
14
15
16
17
18     return 0;
19 }
```

- Rad 8 anropar funktionen `printf` med en parameter — strängen `"Enter an integer >"`
- Resultatet är att texten `Enter an integer >` skrivs ut

f01-plus-one.c — scanf (1)

f01-plus-one.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int n;
6
7      /* Read an integer from the user */
8      printf("Enter an integer >");
9      scanf("%d", &n);
10
11
12
13
14     /* Output the new number */
15     printf("The next higher integer is: %d.\n", n);
16
17     printf("Normal exit.\n");
18     return 0;
19 }
```

- Rad 9 anropar funktionen `scanf` med två parametrar — strängen `"%d"` och . . . nånting mer

f01-plus-one.c — scanf (2)

- Den första parametern i anropet till `scanf`

f01-plus-one.c
```
9       scanf("%d", &n);
```

  innehåller en sträng med formatkoden `%d`
  - `scanf` läser in text som användaren skriver
  - Formatkoden säger att `scanf` ska
    1. Försöka tolka texten som ett heltalsvärde (*d=decimal integer*)
- Den andra parametern innehåller adressen för variabeln `n`
  - Den säger till `scanf` att
    2. Lagra heltalsvärdet i minnet på den adressen
- Och-tecknet (`&`) före variabelnamnet ger minnesadressen för variabeln
  - Koden `&n` går att tolka som "adressen för variabeln `n`"
  - Notera att värdet på adressen döljs för programmeraren

f01-plus-one.c — tilldelning (1)

f01-plus-one.c
```
1   #include <stdio.h>
2
3   int main(void)
4   {
5       int n;
6
7       /* Read an integer from the user */
8       printf("Enter an integer >");
9       scanf("%d", &n);
10
11      /* Increment the number */
12      n = n + 1;
13
14      printf("Exit \n");
15      return 0;
16  }
```

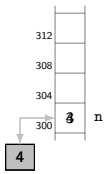
- Satsen på rad 12 innehåller tre operationer:
  1. Läs värdet i variabeln `n` från minnet till ett register i CPU:n
  2. Addera 1 till värdet (i CPU:n)
  3. Skriv det nya värdet från registret tillbaka till variabeln `n`
     - Detta kallas för en tilldelning

f01-plus-one.c — tilldelning (2)

- Vi tar det där igen...

f01-plus-one.c
```
12      n = n + 1;
```

1. Läs det nuvarande värdet i variabeln `n` från minnet till ett register i CPU:n
2. Addera 1 till värdet (i CPU:n)
3. Skriv det nya värdet från registret tillbaka till minnet för variabeln `n`



Tilldelning

- En tilldelningssats i C skrivs som `<variabel> = <uttryck>`
  - `variabel` är ett variabelnamn, t.ex. `n`
  - Tilldelningsoperatorn i C är likamedtecknet `=`
  - `<uttryck>` kan vara vilket giltigt uttryck i C som helst
- Notera att `<uttryck>` evalueras (beräknas) först, sedan utförs tilldelningen
  - Det gör det möjligt att skriva uttryck av slaget
    - `n = n + 1;`
- Det finns olika sorters uttryck, aritmetiska och logiska

Aritmetiska uttryck

- Aritmetiska uttryck byggs upp av variabler, konstanter och aritmetiska operatorer
- Bland operatorerna finns de binära operatorerna +, -, *, /, %
  - Binära operatorer tar två operander, t.ex. `4 + 3`
  - Observera att division, /, mellan två heltal betyder heltalsdivision, dvs. division följt av avrundning mot noll
  - Operatorn % kallas modulo-operatorn och ger resten vid heltalsdivision av två tal
- Studera koden

```
int i = 7, j = 3;
printf("i - j = %d",  i - j); // Outputs i - j = 4
printf("i + j = %d",  i + j); // Outputs i + j = 10
printf("i * j = %d",  i * j); // Outputs i * j = 21
printf("i / j = %d",  i / j); // Outputs i / j = 2
printf("i %% j = %d", i % j); // Outputs i % j = 1
```

  - Notera att formatkoden %d säger att `printf` ska tolka och skriva ut parametervärdet som ett heltal
- Om man använder flera operatorer i samma uttryck, t.ex.

```
int m = i + j * k;
```

finns det prioritetsregler för det (mer senare)

f01-plus-one.c — utskrift

f01-plus-one.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int n;
6
7      /* Read an integer from the user */
8      printf("Enter an integer >");
9      scanf("%d", &n);
10
11     /* Increment the number */
12     n = n + 1;
13
14     /* Output the new number */
15     printf("The next higher integer is: %d.\n", n);
16
17     return 0;
18 }
19
```

- Rad 15 skriver ut det nya värdet på `n`

f01-plus-one.c — printf (igen)

- Anropet till `printf` har två parametrar

```
printf("The next higher integer is: %d.\n", n);
```

- Funktionen `printf` kan ta ett variabelt antal parametrar
  - Den första parametern är en formatsträng
  - Formatsträngen innehåller formatkoden %d som anger att ett heltal ska skrivas ut
  - Parameter två, tre, etc., hör ihop med formatkoderna i formatsträngen
- Formatsträngen innehåller också en styrkod (*escape sequence*) \n som står för radbrytning (*newline*)
- Om värdet som lagras i variabeln `n` är 4 kommer `printf` att skriva ut följande:

```
The next higher integer is: 4.
```

följt av en radbrytning

Formatkoder till printf

- De vanligaste formatkoder till `printf` är dessa:
  - %c ett tecken
  - %d ett heltal
  - %f ett flyttal (”decimaltal”), 28.350000
  - %e ett flyttal i exponentialform, 2.835000e+01
  - %g som %e eller %f beroende på vilken utskrift som blir kortast
  - %s en textsträng
  - %% tecknet %
- Varje formatkod hör ihop med en parameter utom %%

Formatkoder, utrymme

- Man kan även styra hur mycket utrymme (hur många tecken) en parameter ska få vid utskrift:
  - `%4d` totalt 4 tecken, högerjusterat, utfyllt med mellanslag
  - `%04d` samma, men utfyllt med nollor
  - `%4.2f` totalt 4 tecken med 2 decimaler
- Utskriften

```
printf("%4d\n%04d\n%4.2f\n", 23, 23, 3.14159265);
```

ger

```
  23
0023
3.14
```


Styrkoder till `printf`

- Det finns många styrkoder till `printf` med lång historik...

- Ni behöver bara ha koll på `\n`

Lite om heltal och flyttal

- Operatorerna +, −, *, / är definierade för både heltal (`int`, m.fl.) och flyttal (`float`, `double`)
- Om båda operanderna är heltal blir resultatet ett heltal, annars ett flyttal
- Det går att tvinga fram en typkonvertering (*type cast*) genom att skriva typen inom paranteser före uttrycket `(double)i/j`
- Studera koden

```
int i = 3, j = 2;
double m = 3, n = 2;
double k = i / j;
printf("i / j = %f", k); // Outputs i / j = 1.000000
printf("m / n = %f", m/n); // Outputs m / n = 1.500000
printf("i / 2.0 = %f", i / 2.0);  // Outputs i / 2.0 = 1.500000
printf("(double)i / j = %f", (double)i / j); // Outputs (double)i / j = 1.500000
```


`f01-plus-one.c` — avslut (1)

f01-plus-one.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int n;
6
7      /* Read one integer from the user */
8      printf("Enter one integer: ");
9      scanf("%d", &n);
10
11     /* Print the next higher integer */
12     n = n + 1;
13
14     /* Print the result */
15     printf("The next higher integer is: %d.\n", n);
16
17     printf("Normal exit.\n");
18     return 0;
19 }
```

- Utskriften på rad 17 är det sista som skrivs ut från programmet
- Utskriften är en signal till användaren att programmet avslutats på normalt sätt
- Om utskriften saknas så är det ett tecken på att nåt ovanligt hänt
- Ta för vana att alltid avsluta ditt program med den utskriften!

f01-plus-one.c — avslut (2)

f01-plus-one.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int n;
6
7
8
9
10
11
12
13
14
15
16
17     printf("Normal exit.\n");
18     return 0;
19 }
```

- Kommandot `return 0` på rad 18 avslutar `main`-funktionen, och därmed programmet
- Programmet kommer att returnera värdet 0 till operativsystemet
- Det är praxis att låta `main` returnera 0 om programmet avslutas utan fel

Kompilera

- Antag att vi sparat vårt program i filen `plus-one.c`
- Vi kan då kompilera koden med följande kommando:

```
gcc -Wall -std=c99 -o plus-one plus-one.c
```

  - `gcc` är namnet på kompilatorn
  - `-Wall` säger till kompilatorn att varna för allt som verkar suspekt (*Warn all*)
  - `-std=c99` säger vilken C-standard som ska användas
  - `-o plus-one` specificerar namnet på den resulterande exekverbara filen
    - Den exekverbara filen kommer alltså att heta `plus-one` (`plus-one.exe` under windows)
  - `plus-one.c` är namnet på källkodsfilen som ska kompileras
    - ändelsen `.c` talar om för kompilatorn att det är ett C-program som vi vill kompilera
  - OBS! Notera att källkodsfilen heter `plus-one.c` (med `.c`-ändelse) medan den körbara programfilen kommer att heta `plus-one` (utan ändelse)!

Exekvera

- Om kompilering gick bra kan ni köra (exekvera) ert program
- Med exemplet från förra bilden, skriv

  `./plus-one`
  - Under windows kan ni behöva skriva

    `./plus-one.exe`

    eller

    `.\plus-one.exe`

Debugger

- Vill ni köra ert program i en debugger så ska ni lägga till flaggan `-g`

```
gcc -Wall -std=c99 -o plus-one -g plus-one.c
```

- Om ni använder debuggern `nemiver`, starta debuggern med kommandot

```
nemiver ./plus-one
```

Syntax och semantik

- Begreppen syntax och semantik förekommer när man pratar om att programmera
- Syntax
  - Hur skriver man koden?
  - I C så ska varje enkel sats avslutas med semikolon, variabelnamn se ut på ett visst sätt, osv.
- Semantik
  - Vad betyder koden?
  - T.ex. vad är skillnaden mellan `a/b` om `b` är heltal eller flyttal

Att göra fel

- Tre huvudtyper av fel
  - Syntaktiska — källkoden är felskriven och kompilerar ej
    - T.ex. ett bortglömt semikolon
  - Run-Time — uppstår, kan eventuellt upptäckas vid exekvering
    - T.ex. att programmet kraschar när man matar in `abc` i stället för ett heltal
  - Logiska fel — programmet kraschar ej men gör fel saker
    - T.ex. programmet beräknar priset utan moms men skulle beräkna priset med moms
- Alla gör fel när de skriver kod!
  - Det är en naturlig del av processen!
  - Lär dig förstå felmeddelandena!
  - Läs alltid det första felmeddelandet!
- Man tränar upp sin förmåga att förstå och fixa fel
- Kom ihåg: Kompilatorn är din vän!