Föreläsning 1

OH: Övergripande information

Programmering: att instruera en maskin att utföra en uppgift, kräver olika språk:

* maskinspråk = ettor och nollor, kan bara en maskin förstå.
* programmeringsspråk = språk som en människa kan förstå. (Efter kurs i programmering förstås!)

Det finns olika programmeringsspråk, det som denna kurs behandlar kallas *C*. Man skriver ett program (uppsättning instruktioner) i *C* och översätter det till maskinspråk genom en procedur som kallas *kompilering*.

Man brukar också kalla det program man skriver för *källkod* (eller *C*-kod) och det som kommer ut ur kompilatorn för *maskinkod* eller *objektkod*. (Egentligen finns det något som heter "länkning", *eng. linking*, men vi tar inte allting på en gång.)

En kompilator kan normalt sett endast översätta ett språk till en sorts dator, om man byter programmeringsspråk eller dator måste man normalt sett även byta kompilator och för att kunna köra sina program måste de program man skrivit (källkoden) kompileras om för den nya plattformen (datorn). Det finns dock *Java* där detta inte riktigt gäller. (Läs gärna om det själva.)

En kompilator är mycket dum och förstår ingenting av det som programmerats om det inte programmerats precis rätt. Det betyder att ni i början kommer att få gneta med en hel del småfel som verkar löjliga. Men ge inte upp. Det går över.

Då ska vi börja gå igenom programspråket *C*!

Ett *C*-program kan se ut ungefär så här:

```
#include <stdio.h>

main()
{
  printf("Hello World!\n");
}
```

Det enda som händer när man kompilerar och kör detta program är att vi får en utskrift:

```
Hello World!
```

Låt oss se på detaljer i programmet, det första som ofta kommer att finnas är ett så kallat ”inkluderingsdirektiv”, det måste man ha för att kunna göra in- och utmatning. Vidare finns sedan det som ska göras i en main-slinga och det som ska göras är en utskrift. Utskriften sker via anropet till `printf()` som ni kommer att använda väldigt mycket. Det finns i `stdio.h`.

Ordet `main()` markerar programmets början, det är här som det finns som ska hända först, saker och ting händer på kommando, om vi skriver ett annat program som ser ut så här:

```
#include <stdio.h>

main()
{
  printf("Hello World!\n");
  printf("Hello Moon! ");
  printf("Hello Sun!\n");
}
```

Så får vi tre utskrifter, på två olika rader, det ser ut så här:

```
Hello World!
Hello Moon! Hello Sun!
```

Lägg märke till att de två sista utskrifterna hamnar på samma rad, det beror på att vi i inte bytt rad efter utskriften av `Hello Moon!`, man byter rad med escapesekvensen **`\n`** som förekommer sist i den första och den sista utskriften.

När vi studerar program så kommer vi att tala om att saker "händer" och peka på rader i program och säga att "här händer det här och det här" och vi kan följa förloppet i ett *C*-program genom att studera texten i programmet. (”Programmet” kan också kallas ”programkoden” eller ”*C*-koden”.)

Vi tittar på ett annat exempel:

```
#include <stdio.h>

main()
{
  int i;

  i=20;
  printf("The value of i is: %d.\n",i);
}
```

När vi kör detta program får vi utskriften:

```
The value of i is 20.
```

Här introduceras något som vi kallar en *variabel*. En variabel är ett förvaringsutrymme med ett namn som kan innehålla ett värde. Variabelns namn kallas rätt och slätt *variabelnamnet*. Variabelnamnet här är `i`. När vi skriver `i=20;` så tilldelas variabeln `i` värdet 20. Värdet 20 läggs in i förvaringsutrymmet. Sedan skrivs värdet på `i` ut. Det kan nu vara lämpligt att peka ut vissa detaljer.

Vi ser att det står `int i;` överst. Det betyder att variabeln `i` *deklareras*. Vi måste göra deklarationer för alla variabler i ett program innan vi kan börja använda dem. I deklarationen finns först variabelns typ, här är det `int` som är en förkortning för *integer* = engelska för heltal. Sedan kommer variabelnamnet, `i`, och sedan avslutas deklarationen med ett semikolon ("`;`").

Efter deklarationen lämnas en blank rad, det är bra för läslighetens skull att separera deklarationerna från de övriga raderna i ett *C*-program. När vi skriver `i=20;` så läggs värdet 20 in i variabeln `i`. Det kallas en tilldelning (*assignment*) och har till följd att variabeln `i` härmed förvarar värdet 20 i sig. Vi säger kortare att "`i` har värdet 20". Sedan kommer utskriften med `printf()`. Utskriften görs med en så kallad *funktion* och `printf()` är en funktion. Lägg märke till att det som står innanför parenteserna i `printf()` anger det som ska skrivas ut, vi vill skriva ut att `i` har värdet 20 och en förklarande text om det. Det bakas ihop till en så kallad *formatsträng* och sedan anges den variabel som ska skrivas ut. Textföljden `"The value of i is: %d.\n"` kallas alltså *formatsträng* och den anger hur utskriften ska se ut. Den kan tolkas så här:

Skriv först "`The value of i is: `" och skriv sedan ut ett heltalsvärde (**`%d`**), en punkt (**`.`**) och byt sedan rad (**`\n`**). Heltalsvärdet symboliseras av symbolen `%d` och radbytet av `\n`. Teckenföljden **`%d`** brukar kallas en *omvandlingsspecifikation* och anger vad som ska skrivas ut, `%d` står för heltal. Varje omvandlingsspecifikation måste svara mot ett värde senare efter formatsträngen och vi ser att det motsvaras i programmet av att `i` förekommer efter formatsträngen. Betydelsen av hela raden `printf("The value of i is: %d.\n",i);` är alltså att värdet på `i` skrivs ut tillsammans med en förklarande text och resultatet blir alltså

```
The value of i is 20.
```

Nu tittar vi på ett mer avancerat exempel ur *C-programming, A Modern Approach* (förra årets bok):

```
/* dweight.c (Chapter 2, page 20) */
/* Computes the dimensional weight of a 12" x 10" x 8" box */

#include <stdio.h>

int main(void)
{
  int height, length, width, volume, weight;

  height = 8;
  length = 12;
  width = 10;
  volume = height * length * width;
  weight = (volume + 165) / 166;

  printf("Dimensions: %dx%dx%d\n", length, width, height);
  printf("Volume (cubic inches): %d\n", volume);
  printf("Dimensional weight (pounds): %d\n", weight);

  return 0;
}
```

Detta program ger utskriften

```
Dimensions: 12x10x8
Volume (cubic inches): 960
Dimensional weight (pounds): 6
```

Vi har här deklarerat flera variabler, `height`, `length`, `width`, `volume` och `weight` och tilldelat dem flera olika värden och kombinerat dem i olika beräkningar. Beräkningarna ser ni direkt efter tilldelningarna som ger variablerna värdena `8`, `12` respektive `10` och de beräkningarna innebär att vi beräknar volym och vikt av något slags paket.

När man skriver `height * length * width` så utför programmet den motsvarande matematiska operationen som innebär att de tre värdena i variablerna `height`, `length` och `width` multipliceras med varandra. Värdet läggs sedan in i variabeln `volume` genom att man skriver `volume =` det förra uttrycket `height * length * width`.

Det viktiga är inte tillämpningen utan att förstå lite om hur man skriver ett *C*-program. Vi observerar att flera saker händer efter varandra, först tilldelningarna, sedan beräkningarna och sist utskrifterna. Dessa utskrifter innehåller mer komplicerade formatsträngar än den utskrift vi tidigare såg, men principen är den samma: en förtydligande text tillsammans med omvandlingsspecifikationer för de värden som ska skrivas ut. För varje omvandlingsspecifikation måste motsvarande värde anges efter formatsträngen.

När något händer i *C* brukar man kalla det en *sats* (*statement*) när man gör plats för något kallas det en *deklaration* (*declaration*), både satser och deklarationer ska avslutas med semikolon "`;`".

Vi tittar på programmet ovan, vilka rader är deklarationer och vilka är satser?

Vi tittar på ett nytt exempel från boken:

```
/* dweight2.c (Chapter 2, page 23) */
/* Computes the dimensional weight of a
   box from input provided by the user */

#include <stdio.h>

int main(void)
{
  int height, length, width, volume, weight;

  printf("Enter height of box: ");
  scanf("%d", &height);
  printf("Enter length of box: ");
  scanf("%d", &length);
  printf("Enter width of box: ");
  scanf("%d", &width);
  volume = height * length * width;
  weight = (volume + 165) / 166;

  printf("Volume (cubic inches): %d\n", volume);
  printf("Dimensional weight (pounds): %d\n", weight);

  return 0;
}
```

Detta program accepterar värden från användaren och körningen ser ut så här:

```
Enter height of box: 8
Enter length of box: 12
Enter width of box: 10
Volume (cubic inches): 960
Dimensional weight (pounds): 6
```

Här har vi fetmarkerat det som användaren matar in. Inmatning sker med en funktion som heter `scanf()`. Den fungerar som `printf()` fast tvärtom: data åker in i datorn istället för ut. Det är lite svårt att förklara exakt hur den fungerar nu, men det har i alla fall också en formatsträng.

Både `printf()` och `scanf()` har ett *f* i sina namn för att de tar en formatsträng, namnet för `printf()` är egentligen "*Print Formatted Output*" och namnet för `scanf()` är egentligen "*Scan Formatted Input*", men dessa namn är förstås för otympliga att hålla på med. De förkortas således `printf()` respektive `scanf()`.

Programmeringsmiljö

När vi programmerar i *C* i denna kurs ska vi använda en miljö som heter Code Blocks. Vi kommer att programmera främst under *Windows 7* och *Windows XP*. Möjligtvis tittar vi på hur det går till i Linux också. Den praktiska tentamen kommer troligtvis att köras under *Windows XP* så ni måste träna ordentligt på att programmera i *Windows XP*. Inga av övningarna kommer att hållas i skolans datasalar men ni har tillgång till datasalarna och rekommenderas att gå in där på andra tider än de schemalagda. Ni kan även använda andra plattformar som *Linux*, *Mac OS X*, osv. Men tyvärr kommer det inte att finnas mycket support på dessa plattformar. Men speciellt när det gäller *Linux* så brukar folk vara ganska självständiga.

OBS: Samtliga laborationer kommer att redovisas på skolans stationära datorer (med *Windows XP* på), det betyder att ni måste flytta era program till skolans datorer i god tid före redovisningen. Detta är av tre skäl:

1. Vi harbokat skolans datasalar för redovisningar.
2. Ni ska träna på att arbeta i *Windows XP* för att det är bra att kunna det.
3. Ni ska träna på att arbeta i *Windows XP* för den praktiska tentamen går i *Windows XP*.

Det vi ska göra under resten av dagen är att ni får installera *Code Blocks* på er maskin (om ni inte redan gjort det) och börja med de övningsuppgifter som är rekommenderade. Installationsanvisningarna finns på kursens webbsida (några har redan gjort det i introduktionskursen), de rekommenderade övningarna finns angivna i planeringen (som också finns på kursens webbsida) och de rekommenderade övningarna finns i kurslitteraturen.

Så det är bara att sätta igång. Jag finns här och hjälper er på alla sätt jag kan.