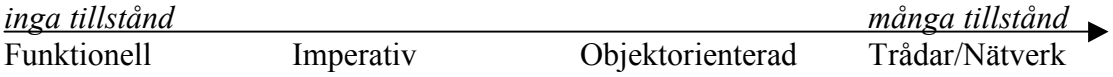
Alexander Baltatzis

C

C är ett imperativt språk. De olika paradigm skiljer sig med mer eller mindre tillstånd.



Datorns minne

För att lära sig programmera i C måste man ha en uppfattning om hur en dator fungerar på låg nivå. En dators minne kan man se som en stor vektor. Ett program består dels av olika maskinkodsinstruktioner i en del av minnet. Dels av data som lagras i en annan del av minnet. Data kan allokeras statiskt eller dynamiskt. Det brukar implementeras med en stack och en heap. Ofta används ena änden av dataminnet till en stack och andra änden till en heap. Exempel:

På stacken allokeras lokala variabler i varje funktionsanrop. Ofta läggs också annat data på stacken som t.ex. returadressen vid funktionsanrop. Data som läggs på stacken städas upp automatiskt.

stack

C är ett litet språk med få typer, flera av dem återfinns i java. T.ex. finns heltal *int*. Det är skillnad på heltal i java och C. T.ex. är det inte säkert hur stor en *int* är det beror på vilken dator man kör på. En annan skillnad är att man kan välja att deklarera den *unsigned* vilket betyder att heltalet inte kan anta negativa värden.
I tidigare versioner av C var man tvungen att deklarera lokala variabler i början av funktionen.

Det finns inga strängar i C utan man använder sig av teckenvektorer. Dessa vektorer kan man allokera statiskt med fix storlek eller dynamiskt. Dynamiskt minne allokeras med *malloc* (står för allocate memory) och *size_of*. Det finns ingen skräphanterare i C utan minne måste frigöras av programmeraren med *free*.

Man når en vektors element med en pekare till elementets minnesadress. Det är väldigt viktigt att man tänker sig för med pekare så att man inte råkar skriva i fel del av minnet.

Motsvarigheten till javas klasser är *struct*. En struct innehåller enbart data men inga funktionsdeklarationer, däremot kan man ha pekare till funktioner.

Skapa C program

Ett C-program skapas i olika steg. Först sker *preprocessor* instruktioner. Dessa opererar enbart på källkodstexten och används t.ex. för att inkludera standard-bibliotekets API. Efter preprocessorn sker *kompilering* och *assemblering*. Då bildas en objekt modul med ändelsen '.o'. Objektfilerna inndehåller maskinkod. I sista skedet länkas de olika objektmodulerna ihop till ett körbart program.

Preprocessorn

Preprocessorinstruktioner börjar med tecknet '#'. Instruktionerna opererar enbart på texten. Med #define kan man definera konstanter och makron. T.ex.

```
#define PI 3.14
#define PRINTLN(x) printf(x"\n");
```

Man kan inkludera header-filer som innehåller funktionsdeklarationer (jämför med javas *import*) för att få tillgång till färdiga definitioner. Preprocessorn klistrar in texten. För att skydda sig mot att inkluderade filer inkluderar samma fil många gånger används följande konstruktion ofta kallad *include-guard.*

```
#ifndef REDAN_INKLUDERAT_PROGRAM
#define REDAN_INKLUDERAT_PROGRAM

#include "local_fil.h"      /* leta lokalt  */
#include <system_fil.h>     /* leta i systemet */

kod...

#endif
```

Länkning

I objektfilerna finns ofta inte all maskinkod som behövs utan en hel del anrop som inte är definierade. Signaturerna för dessa anrop stod i de inkluderade header-filerna. Vid länkningen letar efter definitionerna till dessa filer och hittas de inte får man länkningsfel. Det är jobbigt att felsöka länkningsfel, felmeddelandet kan vara kryptiskt och man har inget att debugga.

Manualer

För att ta reda på mer om de anrop man kan göra finns ett hjälpprogram som heter *man.* Hjälp om printf får man genom att skriva *man printf* i ett terminalfönster (eller använda google). Hjälptexten förutsätter oftast att den som läser är väl förtrogen med C och unix.

Makefile

För att bygga de olika modulerna används ett *make*- program som tar som indata en fil som heter *Makefile,* make kan känna av vilka filer som behöver kompileras om genom att jämföra tidsstämplarna på objektfilen och källkodsfilerna. En makefile beskriver beroenden mellan olika mål (targets) och instruktioner som ska utföras.

```
FLAGS = -W -g

all: nod.o program.o
     gcc $(FLAGS) -o utfil.exe nod.o program.o

program.o: program.c
     gcc $(FLAGS) -c program.c

nod.o:    nod.c nod.h
     gcc $(FLAGS) -c nod.c
```

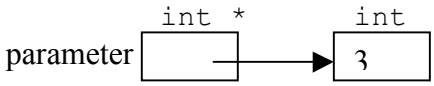
Språket C

Kan man java så känns syntaxen igen.

Parametrar

I C liksom i java finns enbart *värdeparametrar*. Det innebär att skickade parametrar kopieras till funktionen. Ändrar man på parameterns värde I funktionen påverkas inte Vill man ändra på ett parametervärde kan som i Java kapsla in parametern i en struct och skicka structen som ett värde. Då kan man ändra structens fält. Detta är inte så vanligt i C utan istället kan man skicka adressen till parametern man vill ändra på.

```
fscanf(stdin, "%f", &lengd);
```



Vill man ändra på en pekares värde kan man på samma sätt skicka adressen till pekaren. I funktionens parameterlista tar man då emot en pekare till en pekare *int ** p*

I C++ och pascal finns referensparametrar, då kopieras inte värdet utan kan ändringarna ändrar också anropsparametern. Java har inte referensparametrar!

Man kan även skicka funktionspekare d.v.s adresser till funktioner. Funktionspekare kan deklareras och användas så här:

```
void (*function_pointer) (Post * p);  // Declare
function_pointer = &writePost;        // Initialize
function_pointer(&p);                 // Call function
```

Exempel

```
#include <stdlib.h>
#include <stdio.h>

struct post {
   char name[29];
   float bmi;
   struct post * next;
};
typedef struct post Post;

void writePost(Post * p) {
   printf("Namn: %s\nbmi: %f\n", p->name, p->bmi);
}

int vikt;              /* global variabel  */
static float lengd;    /* filglobal variabel */
```

```
int main(int argc, char * argv[]) {
   Post * p  = (Post *) malloc(sizeof(Post));
   p -> next = (Post *) malloc(sizeof(Post));

   printf("Vad heter du? ");
   fscanf(stdin, "%s", p->name);

   printf("Hur lång är du (m)? ");
   fscanf(stdin, "%f", &lengd);

   printf("Vad väger du (kg)? ");
   fscanf(stdin, "%d", &vikt);

   p -> bmi = vikt / (lengd * lengd);
   writePost(p);
   return 0;
}
```

Vanligt fel

Ett vanligt fel är att råka göra en tilldelning i if-satser. Det är fullt legalt att skriva:

```
a = 1;
if (a = 5) printf("a = %d", a);
```

Övrigt

Det finns olika versioner av C

K-R C	1973	Inga parameterlistor
Ansi C	1989	Parameterlistor
C-99	1999	En hel del från C++, //-kommentarer m.m.

Det finns speciella nyckelord t.ex.

`static` – kan tillverka modulglobala variabler (synliga enbart i filen)
`register` – använd ett register (snabbt minne nära processorn)
`volatile` – variabeln kan ändras utanför programmet (t.ex. av en interrupt)