Programmera i **C**

Varför programmera i **C** när det finns språk som Simula och Pascal??

- **C** är ett språk på relativt "låg" nivå vilket gör det möjligt att konstruera effektiva kompilatorer, samt att komma nära konstruktionen utan att behöva skriva maskinkod.
- **C** är i det närmaste industristandard. I och med detta är **C** ett språk som varje teknolog bör vara bekant med.

Detta kapitel är tänkt att introducera den oinvigde programmeraren i **C**. Beskrivningen är *mycket* kortfattad och *absolut* inte tillräcklig för att kunna skriva program i **C**. För att skriva fungerande program rekommenderas boken *The C Programming Language* av Kerninghan & Ritchie.

C är ett svagt typat språk till skillnad från Pascal och Simula. Stark typning innebär att funktioner, procedurer och operatorer definierade för en viss datatyp endast kan användas på data av samma typ. I **C** finns en möjlighet att definiera exempelvis en "pekare till vad som helst" och använda denna pekare till heltal *och* pekare till flyttal. Detta betyder att **C** program har hög flexibilitet men svårfunna programmeringsfel.

Ett enkelt **C**-program kan se ut enligt nedan;

```
#include ‘‘constants.h’’
#define ANTAL 10

char tecken;            /* Globala variabel */

main()                  /* Huvudprogram */
{
  int x,y,sum           /* Lokala variabler */

  x=y=0;
     .                  /* Satser */
     .
  sum=rakna(x,y);       /* Funktionsanrop */
     .
     .                  /* Satser */
     .
}                       /* Slut på main */

int rakna(int k, int r)   /* Funktionen rakna */
{
  int res;
     .
     .                 /* Satser */
  return(res);         /* returnera beräknat värde */
}
```

Kommentarer till programmet:

- **include:** Till **C** finns ett "bibliotek" med diverse standardfunktioner eller funktioner programmeraren själv har skapat, så kallade *include-filer*. Kommando till förprocessorn.
- **define:** Definierar ett *macron*. I exemplet betyder detta att ANTAL ersätts med 10 i programmet. Kommando till förprocessorn.
- **Kommentarer:** Allt mellan **/*** och ***/** betraktas som kommentarer.
- **globala variabler:** Variabler definierade utanför main betraktas som globala. Dessa gäller överallt.
- **main:** Själva huvudprogrammet.
- **lokala variabler:** Variabler definierade i en funktion betraktas som lokala. Dessa gäller endast i funktionen.
- **Tilldelning: =** är tecknet för att tilldela en variabel något värde.
- **Avsluta satser: ;** är tecknet som avslutar en sats.
- **Sammansatta satser: {** och **}** används för att knyta samman flera satser till ett block. Jämför Pascal's begin och end.
- **Funktionsanrop:** En funktion anropas med funktionsnamnet och eventuella argument.
- **Funktioner: int** talar om vad för datatyp funktionen returnerar. Variablerna **k**,**r** är lokala och av samma typ som **x**,**y**. **res** är returnerat värde och av samma typ som **sum**.

0.0.1 Typer, Uttryck och Operatorer

Det finns ett fåtal datatyper i **C**.

- **char** Tecken, oftast 8 bitar.
- **int** Heltal, storleken beror på maskinen.
- **float** Flyttal
- **double** Dubbel flyttal.

Till dessa kan knytas ett antal giltiga "förtyper"

- **short** halverar antal bitar till integer.
- **long** fördubblar antal bitar till integer.
- **signed** Med teckenbit.
- **unsigned** Utan teckenbit.

Exempel: Om inget anges till kompilatorn (Kap. **??**) gäller för **-68**.

- **char** = **unsigned short int**: 8 bitar utan tecken.
- **int** = **signed int**: 16 bitar med tecken.
- **unsigned int** : 16 bitar utan tecken.
- **long int** = **signed long int** : 32 bitar med tecken.
- **unsigned long int** : 32 bitar utan tecken.
- **Pekare** Alltid 32 bitar utan tecken.

Konstanter

- En heltalskonstant skrives:
  - Decimalt 65
  - Oktalt Inleds med **0** Exempelvis: **0**101
  - Hexadecimalt Inleds med **0x** Exempelvis: **0x**41
- Flyttal anges med decimalpunkt. Exempelvis: 65.5
- En teckenkonstant skrives med tecknet inom ' '. Exempelvis: 'A'
- En teckensträng skrives inom " " . Exempelvis: "Hej". Varje teckensträng avslutas med NULL (0x00).

Matematiska operatorer

- + Addition
- **-** Subtraktion
- ***** Multiplikation
- / Heltalsdivision
- % Modulo division

Jämförelseoperatorer

- > Större än
- >= Större än eller lika med
- < Mindre än
- <= Mindre än eller lika med
- == Logiska operatorn LIKA MED
- **!=** Logiska operatorn ICKE LIKA MED
- || Logiska operatorn ELLER
- **&&** Logiska operatorn OCH

Bitoperatorer

- & OCH
- | ELLER
- ^ Exklusiv ELLER
- << Vänsterskift
- >> Högerskift
- ~ Komplement

0.0.2 Funktioner

Funktioner i **C** bryter ner och förenklar stora program. **C** program består oftast av många små funktioner (eventuellt i olika filer, *det finns länkare*), för att hanteringen av programmet skall bli enklare.

En funktion skapas genom att skriva den enligt prototypen; resultattyp **funktionsnamn** (parameterlista)

{

satser

}

Resultattyp är vilken datatyp funktionen returnerar. Returneras inget används **void** och skrives inget blir resultatet **int**.

Funktionsnamnet är upp till programmeraren.

Parameterlistan innehåller vilka parametrar funktionen anropas med. Man måste ange typen för varje parameter. Om funktionen saknar parametrar används typen **void**.

Exempel: Funktionen kalle anropas med int k, char c och returnerar inget.

```
void kalle(int k, char c)
{
  int  n;     /* En lokal variabel, gäller endast i funktionen */
  sats;
  sats;
  return;
}
```

Anropet av funktionen kalle;

```
      .
  kalle(2,'A');
      .
```

Exempel: En liknande funktion som kalle men pelle returnerar en int.

```
int pelle(int k, char c)
{
  int  n;     /* En lokal variabel, gäller endast i funktionen */
  sats;
  sats;
  return(n);
}
```

Anropet av funktionen pelle;

```
      .
  r=pelle(2,'A'); /* r och pelle måsta vara av samma typ (int) */
      .
```

0.0.3 Flödeskontroll

Flödeskontroll används i **C** för att beskriva alternativa vägar som programmet kan ta. Allmänt gäller att alla satser kan göras sammansatta med **{** (jmf. Pascal's begin) och **}** (jmf. Pascal's end).

if-satsen Det vanligaste sättet att åstadkomma alternativa vägar i ett program är med **if**-satsen.

if (uttryck)
    sats

Om uttryck är sant (inte noll) utföres sats(er).

Exempel:

```
      .
  if ( teck == 'A')    /* Om teck's innehåll är lika med ett A */
   {
     k = 2;            /* Tilldela k värdet 2 */
     b = teck;         /* Flytta teck's innehåll till b */
   }
      .
```

En annan version av **if** har formen

if (uttryck)
    sats_1
else
    sats_2

Om uttryck är sant (inte noll) utföres sats_1 annars sats_2.

Exempel:

```
      .
if ( teck == 'A')    /* Om teck är lika med ett A */
 {
   k = 2;            /* Tilldela k värdet 2 */
   b = teck;         /* Flytta teck till b */
 }
else                 /* Om teck inte är lika med ett A */
    teck = 'B';      /* Tilldela teck tecknet B */
      .
```

while-, do-while-sats Ett sätt att åstadkomma repetition är med **while**-satsen. Den allmänna formen är;
while (uttryck)
    sats

Så länge uttryck är sant, utföres sats. Tänk på att om uttryck är falskt från början utföres inte sats.

Exempel: Så länge **int** n är mindre än eller lika med 10, räkna ner **int** k.

```
n=0;
while (n <= 10) {
  k--;
  n++;
}
```

Jämfört med;

do {
    sats
}while (uttryck**);**

vilken alltid utföres minst en gång och testas därefter.

Sammanfattning: I **while** testas först uttrycket och sats(er) utföres om sant, medan i **do-while** testas uttrycket efter varje utförd sats(er). Lägg märke till var ett semikolon avslutar satsen.

for-satsen Den allmänna formen för **for**-satsen är;

for (uttryck_1**;**uttryck_2**;**uttryck_3**)**

for är ekvivalent med följande **while**-sats
uttryck_1
while (uttryck_2) {
    sats
    uttryck_3;}

uttryck_1 utföres en gång före första varvet. uttryck_2 utföres först en gång på varje varv och är det uttryck som styr repetitionens fortsättning. uttryck_3 utföres sist på varje varv. Oftast används **for** när det finns behov av att räkna upp/ner på varje varv.

Exempel: Skriv ut k's värde från 0 till 99.

```
for (k=0; k < 100; k++)
      printf(`` K-antal=%x\n'',k);
```

switch-satsen **switch** är en naturlig sats att använda i flervalssituationer i stället för **if**. Den allmänna formen är;

switch (uttryck) **{**
case konst **:** sats
case konst **:** sats
.
.
case konst **:** sats
default : sats
}

Beroende på utfallet av variablen uttryck (jämfört med konst), utföres de olika satserna. Satsen efter **default** utföres om inga av **case**-satserna blir uppfyllda.

Exempel: c är av typen char. Testa vilket tecken det är och addera till ett värde till int k beroende på tecknet.

```
switch (c) {
  case 'A' : k=k+1; break;          /* Om c == A öka k med ett */
  case 'B' : k=k+2; break;          /* Om c == B öka k med två */
  case 'C' : k=k+3; putc(c); break;/* Om c == C öka k med tre, skriv c */
  default  : k=0; break;            /* Om c != (A | B | C) sätt k = 0 */
}
```

Exekveringen av **break** medför att **switch**-satsen avbrytes. Finns det inget **break** fortsätter exekveringen med nästa sats vilket inte var meningen i detta fallet.

break, continue Ibland är det nödvändigt (jämför **switch**) att avbryta exekveringen av en slinga utan att fullfölja. **break**-satsen förorsakar att den innersta loopen (**for, while, do** eller **switch**) avbryts omedelbart. **continue** är mycket likt **break**. Skillnaden är att **continue** avbryter bara pågående iteration i **for**, **while** och **do**.

0.0.4 Pekare, fält och poster

En pekare i **C** är referenser (adresser) till objekt (minnesutrymme). Pekarvariabler deklareras enligt:

int *ip; En pekare som pekar på en integer.
char *ip; En pekare som pekar på ett tecken. Fält är ett sätt att lagra ett antal värden av samma typ under samma namn. Ett fält deklareras enligt:

int iflt[10]; Ett heltalsfält med tio element.
char buf[100]; Ett teckenfält med 100 element.

OBS! Fälten's index börjar på noll. **OBS!** Poster i **C** kallas **struct**. På samma sätt som i fält vill man lagra variaber under samma namn, men i **struct** kan dessa vara av olika typer. Exempel:

```
struct bok {
    char titel[50], författare[100];
    int antal_sidor,pris;
    float vikt;
};
```

All hantering av pekare, fält och poster hänvisas till **R&K** vilket förtydligar användbarheten och finessen med dessa.

0.0.5 Fallgropar

Det finns ett antal "vanliga" fel i **C**.

- Tilldelning skrivs med **ett** likhetstecken.
- Likhetstest skrivs med **två** likhetstecken.
- Skilj mellan **logiska** operationerna && och || och motsvarande **bitoperationer** & och |.
- Första elementet i ett fält börjar på **noll**.
- Blanda inte ihop pekarens värde (pek) och värdet av pekaren (*pek).
- Teckensträng ("hello") avslutas med NULL, ej att förväxla med teckenkonstanten ('A').
- Samma **typer** på bägge sidor av tilldelningstecknet??
- **case** i **switch**-satsen avslutas inte med ett **break**.