Vad man behöver veta om C

Digitala system

Datatyper skiljer sig mellan olika datorer

Detta gäller i kursen Digitala system:

Några datatyper	Antal byte	Talområde
`unsigned char`	1	0 – 255
`signed char`	1	-128 – 127
`unsigned int`	2	0 – 65535
`signed int`	2	-32768 – 32767
`unsigned long int`	4	0 – 4294967295
`signed long int`	4	-2147483648 – 2147483647
`float`	4	±1,18 E-38 – 3,39 E+38

Om man utelämnar ”unsigned” eller ”signed” så väljer kompilatorn själv det ena.

Under vårens JAVA-kurs gäller annat. Hur många bitar som används för olika typer beror på vilken dator man använder.

.

Exempel

```
char Tal, Max, Min;
unsigned int Adress;
```

```
const char Tabell [][3] = { { 23, 30, 64 } ,
                            { 12, 31, 16 } ,
                            { 42, 54, 86 } ,
                            { 29, 32, 64 } };
```

```
const char String [] = "ABC";
```

innebär samma som:

```
const char String [] = {0x41, 'B', 0x43, 0};
```

Olika skrivsätt för samma sak:

```
char a = 65;          ← decimalt
char a = 0x41;        ← hexadecimalt
char a = 081;         ← oktalt (används knappt)
char a = 0b01000001;  ← binärt
char a = 'A';         ← motsvarande ASCII-kod
```

Aritmetiska operationer

+	Addition
-	Subtraktion
*****	Multiplikation
/	Division `13/5` ger värdet 2 ifall man använder heltal
%	Modulodivision: `13%5` ger värdet 3 vid heltal

```
Tiotal = Tal / 10;
Ental = Tal % 10;
```

```
unsigned int b=10000;
float a;
char c;

a = b*100; ger 16960  (felaktigt resultat)

a = (float)b*100; ger 1000000

   (detta kallas typomvandling, casting)

c = b; ger c =16 (varför?)
```

Bitoperationer är viktiga i

Digitala system

&	Bitvist OCH
|	Bitvist ELLER
^	Bitvist exklusivt ELLER
~	Bitvis invertering
<<	vänsterskift
>>	högerskift

```
a = ~a;
```

Var försiktig med operatorn ~!

Den gör alltid om talet till interger!
Bitvis invertering av ett 8-bitarstal gör man istället exempelvis med:

```
a = a ^0xFF
```

```
a = a<<1;
```

Tilldelningssatser

Tilldelning:	**=**	
Ökning	**++**	`antal = antal+1: antal++`
Minskning	**--**	`antal = antal-1: antal--`

```
PORTB |= 0x80;

char antal = 5;
A = antal++;   ger A = 5
A = ++antal;   ger A = 6


PORTB = Tabell [2][0];
```

Jämförelser och logiska operatorer

Jämförelseoperatorer	**<**	Mindre än
	>	Större än
	<=	Mindre än eller lika med
	>=	Större än eller lika med
	==	Lika med
	!=	Inte lika med

Logiska operatorer	**&&**	OCH
	||	ELLER
	!	ICKE

Programkontroll: if, while, for, switch/case

if

```
if (villkorsuttryck){
    satser;
}
else if (uttryck){
    satser;
}
else {
    satser;
}

if (a==b&&c==d){
    satser;
}
else if (uttryck)
{
    satser;
} else
{
    satser;
}
```

while

```
while (uttryck) {
    satser;
}
```

```
do
{
    satser;
}
while (uttryck);
```

for

```
for (startvärde;villkor;sats)
{
    satser;
}
```

```
a = 5;
while (a != 0) a--;
```

är exakt samma sak som

```
for (a = 5; a != 0; a--);
```

Alternativ till konstruktionen if - else:

switch/case

```
switch (month)
{
    case 1:         if (month==1){
        ....;           ....;
        ....;           ....;
        break;      } else
    case 2:         if (month==2){
        ....;           ....;
        ....;           ....;
        break;      } else
 ............       ............
    default:        } else {
        ....;               ....;
        ....;               ....;
        break;      }
}
```

Funktioner, funktionsvärden och parametrar

```
int main(void){
    satser;
}
```

```
char Count (char byte){
    char antal;
    // Satser som räknar ut hur många ettor
det finns i byte.
    return antal;
}

PORTA = Count(PORTC & 0b00000111);
```

Avbrottsfunktioner

```
ISR (SIG_OVERFLOW1){
    ....;
    ....;
}
```

Pekare och adresser till variabler

```
char count(char *p)
{
Ta emot pekare till char
(deklaration av pekarvariabel)
   *p = *p+1;
Här är * en operator.
Betyder: det värde som p pekar på.
   return *p;
Returnera det som p pekar på.
}


A = count(&PORTB)


Write_String (&String[0]);

Write_String (String);
```