Programmering og bitmanipulering.

For å kunne programmere i C på en mikrokontroller trenger vi en C-kompilator som den aktuelle kontrolleren, for eksempel for Atmel AVR bruker vi avr-gcc som kommer sammen med Atmel Studio 6. Da kan vi inkludere en "header"-fil som gir oss muligheten til å skrive og lese fra registre og manipulere (endre på) bit:

```
#include <avr/io.h>
```

Registre kan sees på som en variabel, det vil si en datalokasjon det går an å skrive tallverdier til og lese tallverdier fra. Alt annet vi skriver er standard c-kode. Se også kompendiet til Knut Olsen-Solberg om bitvise operatorer. Setting av bit i registre kan nå gjøres ved å skrive en verdi med 1-ere på de riktige plassene:

```
DDRB = 0b0000 1001  // Setter PB3 og PB0 som utganger.
```

0b er C-kode for binært tall. Tilsvarende er 0x for heksadesimale tall. Som standard oppfatter c alle tall som desimale og vi kan godt bruke det hvis vi vil, men binære og hexadesimale er som oftest lettere å lese når vi opererer på bit i registre.

Bit operators:

& bitvis og (AND). Vil sette en bit i resultatet hvis den eksisterer i begge operandene og slette hvis ikke. Eksempel:

```
  0b10001011
& 0b01101010
= 0b00001010
```

Brukes i mikroprosessorprogrammering til å slette bit, eller for isolere bit før tester (if / while).

| bitvis eller (OR). Vil sette en bit i resultatet hvis den eksisterer i den ene eller begge operandene. Eksempel:

```
  0b10001011
| 0b01101010
= 0b11101011
```

Brukes i mikroprosessorprogrammering til å sette bit.

^ eksklusiv eller (XOR). Vil sette en bit i resultatet hvis den er satt i enten den ene eller andre operanden, men ikke i begge. Eksempel:

```
  0b10001011
^ 0b01101010
= 0b11100001
```

Brukes i mikroprosessorprogrammering til å toggle bit (for eksempel lav – høy – lav – osv.).

~ ens komplement. Hvert bit blir invertert.

```
 ~0b10001011
= 0b01110100
```

Brukes i mikroprosessorprogrammering for å lage inverse bitmasker for sletting av bit.

<< venstreskift. Tallet før << blir skiftet det antall bit til venstre som er angitt etter <<. Er ekvivalent med multiplikasjon med en faktor av 2.

```
  5 << 3
= 0b00000101 << 3
= 0b00101000
= 40
= 5 x 2^3
```

Mye brukt for å lage bitmasker i mikroprosessorteknikk.

>> høyreskift. Tilsvarende som vensteskift. Mindre brukt i mikroprosessorprogrammering. Ekvivalent med deling på en faktor av 2.

C har en kortform som blir mye brukt:
```
x += 2; // som er det samme som x = x +2;
```
Den kan også brukes sammen med andre operatorer, for eksempel &= |= ^=.

Anvendelse i mikroprosessorprogrammering.
Lage bitmaske med en bit satt. Eksempel:
```
  1<<DDB2
= 1<<2
= 0b00000100
```
Resultatet er en bitmaske (en tallverdi) med bit DDB2 satt.

Lage bitmaske med flere enn en bit satt, for eksempel:
```
  (1<<DDB2) | (1<<DDB0)
= (1<<2) | (1<<0)
= 0b00000100 | 0b00000001
= 0b00000101
```
Resultatet er en bitmaske med bit DDB2 og DDB0 satt. Den kan utvides til flere bit. Hvis alle bit skal settes gir blir det oftest mer oversiktlig med bare tallverdien 0xFF.

Sette register lik en bitmaske, for eksempel:
```
DDRB = (1<<DDB2) | (1<<DDB0);
DDRB = 0b00000101;
```
Dette er det samme, men øverste linje viser hvilke bit som settes. Med forståelse av porter så leser man øverste linje som at PB2 og PB0 blir utganger, mens resten blir innganger.

Sette bit i et register, for eksempel:
```
DDRB |= (1<<DDB3);
DDRB = DDRB | 0b00001000;
```
Begge linjene gjør det samme; Leser DDRB registeret, setter DDB3 bitet, før resultat skrives tilbake til DDRB. Resultatet er at PB3 blir utgang uten at andre pinner er endret. Man kan også sette flere bit samtidig.

Slette bit i register, for eksempel:
```
DDRB &= ~(1<<DDB2);
DDRB = DDRB & 0b11111011
```
Begge linjene gjør det samme; Leser DDRB registeret, sletter bit DDB2, før resultatet skrives tilbake til DDRB registeret. Resultatet er at PB2 blir en inngang uten at andre pinner er endret. Man kan også slette flere bit samtidig.

Toggle (veksle) bit i register, for eksempel:
```
PORTB ^= (1<<PORTB0);
PORTB = PORTB ^ 0b00000001;
```
Begge linjene gjør det samme; Leser PORTB registeret, toggler (veksle, dvs. endre fra 0 til 1 eller motsatt) PORTB0 bitet, før resultatet skrives tilbake til PORTB med resultat at PB0 (forutsatt at den er utgang) toggles.

Bitsjekk:

Hvis man skal sjekke en pinne må man lese pinneregisteret og sjekke bitet for den aktuelle pinnen, for eksempel for PA2 pinnen:

```
if ( PINA & (1<<PINA2) ) {
      // gjør ett eller annet hvis PA2 er høy.
}
```

Alternativer som feiler:

```
if (PINA2){} // if(2) vil alltid slå til
if (PINA & PINA2){} // sjekker PINA1 i stedet for PINA2.
if( PINA == (1<<PINA2) ) {} // slår til hvis PINA2 er høy og SAMTIDIG ALLE andre pinner på port A er lave, ellers feiler det.
```