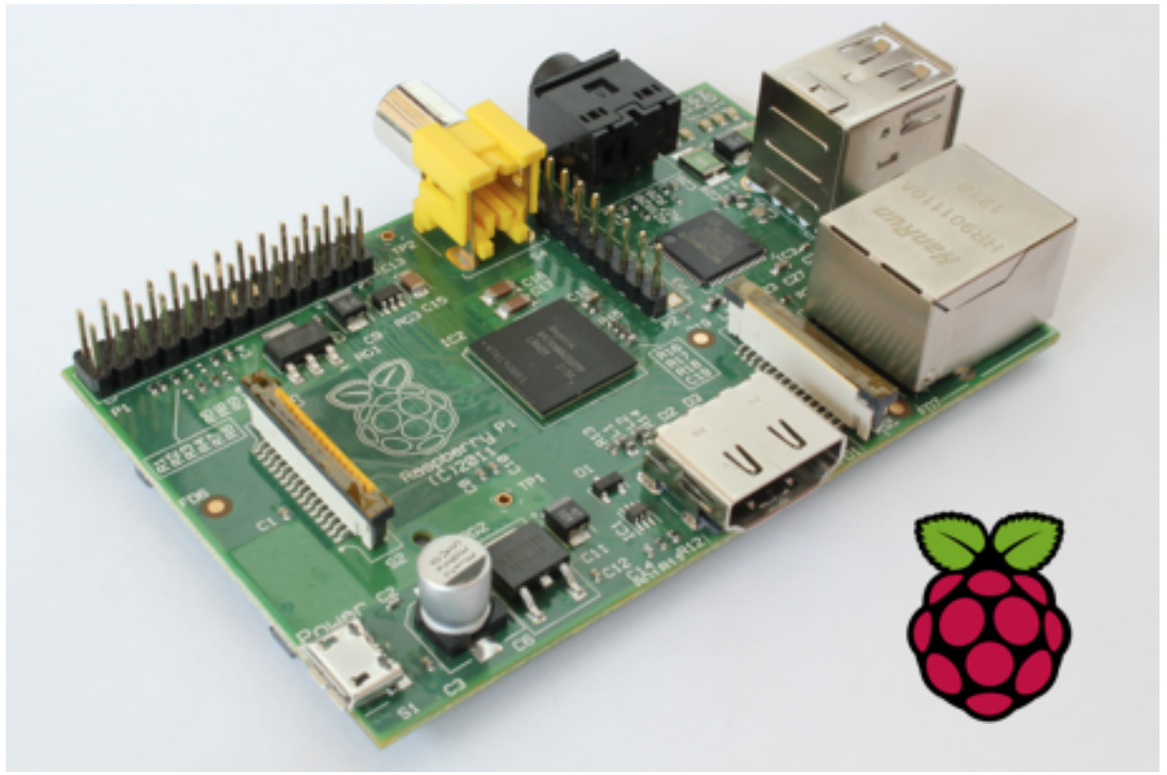
Lær Kidsa Koding

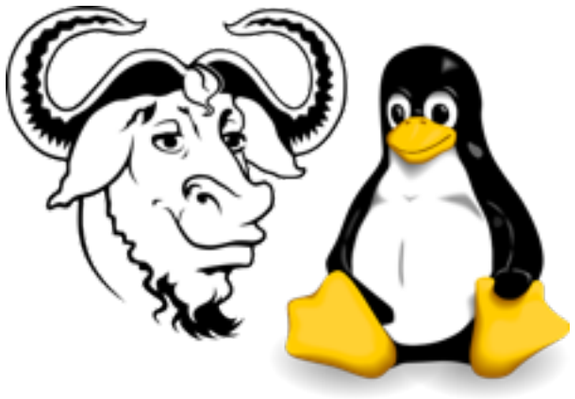
Olve Maudal, Cisco Systems



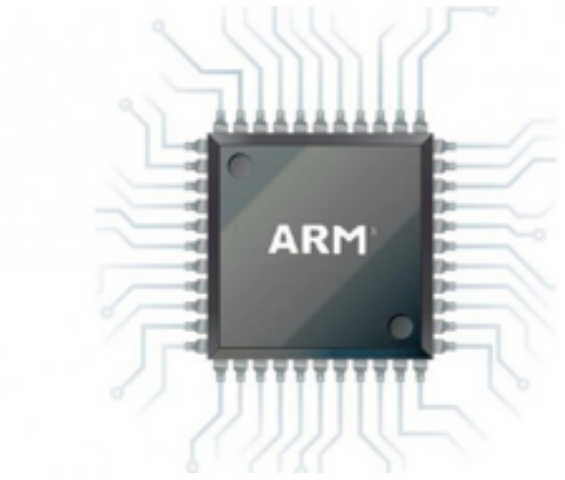
2-timers minikurs i
GNU/Linux, C, C++ og assembler
aldersgruppe 10-16 år

Sommerskole, Bleiker
Fredag 27. juni 2014

I dag skal vi lære litt om:



GNU/Linux



Bli kjent med Raspberry Pi og GNU/Linux (obligatorisk)

1) Start maskinen

2) Logg inn på maskinen (bruker: `pi`, passord: `raspberry`)

3) Se på alle filene i nåværende katalog

```
pwd
ls
```

4) Skift ned til underkatalogen python_games, og list alle filene som slutter på `.py`

```
cd python_games
pwd
ls *.py
```

5) Start spillet Wormy

```
python wormy.py
```

6) Se på kildekoden, prøv å forstå noe av det som står der

```
nano -v wormy.py
```

6) Gå tilbake til din hjemmekatalog

```
cd
pwd
ls -al
```

7) Start the grafiske brukergrensesnittet

```
startx
```

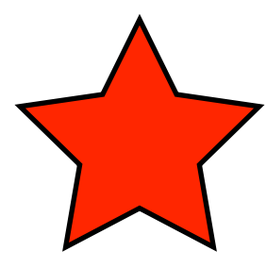
8) Start LXTerminal og skriv inn

```
ls
exit
```

9) Avslutt det grafiske brukergrensesnittet (trykk Ctrl-Alt-Backspace samtidig)

10) Restarte maskinen

```
sudo reboot
```



Hello (C)

Logg inn på maskinen (pi/raspberry).

Gå til hjemmeområdet ditt:

```
cd
```

Lag en ny programfil med en teksteditor:

```
nano hello.c
```

og så skriver du inn programmet som du ser til høyre.

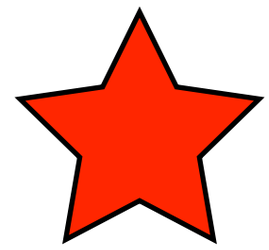
```c
#include <stdio.h>

int main(void)
{
    printf("Hello!\n");
    return 0;
}
```

Når du er ferdig med å skrive inn programmet kompilerer du det med en C kompilator:

```
gcc -o hello hello.c
```

og så kjører du programmet slik:

```
./hello
```



Hi (C++)

Logg inn på maskinen (pi/raspberry).

Gå til hjemmeområdet ditt:

```
cd
```

Lag en ny programfil med en teksteditor:

```
nano hi.cpp
```

og så skriver du inn programmet som du ser til høyre.

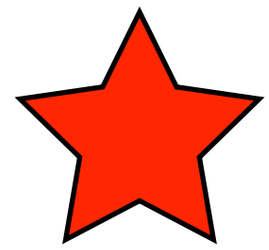
```
#include <iostream>

int main()
{
    std::cout << "Hi!" << std::endl;
    return 0;
}
```

Når du er ferdig med å skrive inn programmet kompilerer du det med en C++ kompilator:

```
g++ -o hi hi.cpp
```

og så kjører du programmet slik:

```
./hi
```



Deep Thought (Assembler)

I Douglas Adams fantastiske bok "The Hitchhiker's Guide to the Galaxy", er det en datamaskin, Deep Thought, som regner ut det ultimate svaret på "Life, the universe, and everything". Det tok 7.5 millioner år å regne ut svaret, og det var et tall. Problemet var bare at ingen visste hva spørsmålet var...

Her skal vi skrive et lite program som også regner ut det ultimate svaret. Den bruker riktignok mye kortere tid en Douglas Adams maskin. Men problemet er fortsatt at vi ikke vet hva spørsmålet er.

Skriv inn koden akkurat slik den står, prøv å reflekter over hva hver linje betyr.

```
nano deepthought.s
```

Programmet er skrevet i assembler. Den kan lett oversettes til maskinkode med en såkalt assembler og deretter kan man lage en kjørbar fil med en linker:

```
as -o deepthought.o deepthought.s
ld -o deepthought deepthought.o
```

Hvis du skrev inn programmet nøyaktig som vist her så har du fått en såkalt eksekverbar objektfil som maskinen kan kjøre direkte, og så kan vi sjekke returverdien fra programmet for å finne ut hva svaret er:

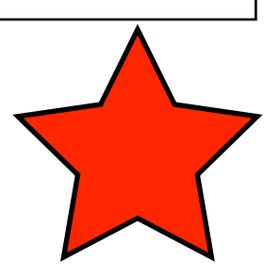
```
./deepthought
echo $?
```

Oppgaver:

a) Skriv inn og kjør programmet. Hva er det ultimate svaret?

b) Prøv å forstå hva programmet gjør. Assembleroperasjonen for å multiplisere heter MUL og den tar tre operander. Klarer du å endre programmet slik at den bare multipliserer 6 og 7, og deretter returnere det riktige resultatet?

```
.globl _start

_start:
mov r1, $6
mov r2, $7
mov r3, $0

loop:
cmp r2, $0
beq exit
add r3, r3, r1
sub r2, r2, $1
b loop

exit:
mov r0, r3
mov r7, $1
svc $0
```



Name (C++)

Logg inn på maskinen (pi/raspberry).

Gå til hjemmeområdet ditt:

```
cd
```

Lag en kopi av den gamle filen, og editer kopien med en teksteditor:

```
cp hi.cpp name.cpp
nano name.cpp
```

og så skriver du inn programmet som du ser nedenfor.

```
#include <iostream>

int main()
{
    std::string name;
    std::cout << "What is your name? " << std::flush;
    std::cin >> name;
    int i = 0;
    while (i < 10) {
        std::cout << "Hello " << name << "!" << std::endl;
        i = i + 1;
    }
    return 0;
}
```

Kompiler og kjør:

```
g++ -o name name.cpp
./name
```

Primtall

Et primtall er et tall større enn 1 som bare er delbart med seg selv og 1. De ti første primtallene er:

```
2, 3, 5, 7, 11, 13, 17, 19, 23, 29
```

Vi skal lage et program som regner ut primtall. Bruk en teksteditor for å skrive inn programmet:

```
nano primtall.c
```

Programmet er skrevet i programmeringsspråket C og før det kan kjøres så må det konverteres til maskinkode av en såkalt kompilator:

```
gcc -O2 -o primtall primtall.c
```

gcc er navnet på programmet som kompilerer koden. "-O2" betyr at kompilatoren kan bruke litt ekstra tid på å optimalisere koden, "-o primtall" betyr at output-filen skal hete "primtall", og så oppgir vi navnet på filen som skal kompileres.

Hvis du skrev inn programmet nøyaktig som vist her så har du fått en såkalt eksekverbar objektfil som maskinen kan kjøre direkte.

```
./primtall
```

Oppgaver:

a) Skriv inn og kjør programmet. Hvor mange primtall er det mellom 1000 og 1100?

b) Bruk en stoppeklokke eller bruk kommandoen time for å måle tiden det tar å kjøre programmet, feks:

```
time ./primtall
```

Prøv å endre programmet til å regne ut større og større primtall intill du ser at det begynner å gå ganske sakte. Hvor stort må tallene være før det tar over et sekund å sjekke om det er et primtall? (trykk Ctrl-C for å avbryte et programm som kjører alt for sakte)

c) Et tall n som ikke er delelig med noen av tallene fra 2 til kvadratroten av n er et primtall. Klarer du å bruke den kunnskapen til å lage programmet mye raskere? (hint: predikatet i <= sqrt(n) kan også skrives i*i <= n)

d) Søk på nettet eller spør deg litt rundt. Hva er ansett som den raskeste måten å generere primtall på?

```c
#include <stdio.h>
#include <stdbool.h>

bool is_prime(int n)
{
    if (n < 2)
        return false;
    int i = 2;
    while (i < n) {
        if (n % i == 0)
            return false;
        i++;
    }
    return true;
}

int main(void)
{
    int from = 1000;
    int to = from + 100;
    int n = from;
    while (n < to) {
        if (is_prime(n))
            printf("%d\n", n);
        n += 1;
    }
}
```

Calc, del 1

Alle større programmer består av flere filer som oversettes til maskinspråk hver for seg (slik at vi får en .o objektfil) og så lenkes disse objektfilene sammen med en såkalt linker slik at vi får et kjørbart program. Her får du også se et enkelt eksempel på objekt-orientert programmering i C++. Skriv inn filene under, og så kjører du disse kommandoene:

```
g++ -c my_calculator.cpp
g++ -c calc1.cpp
g++ -o calc1 my_calculator.o calc1.o
./calc1
```

Hva er svaret?

my_calculator.hpp

```
#include <stack>

class my_calculator {
public:
    void push(double value);
    double pop();
    double top() const;
    void add();
    void sub();
private:
    std::stack<double> stack;
};
```

calc1.cpp

```
#include "my_calculator.hpp"
#include <iostream>

int main()
{
    my_calculator calc;

    calc.push(35);
    calc.push(9);
    calc.add();
    calc.push(2);
    calc.sub();
    std::cout << calc.top() << std::endl;
}
```

my_calculator.cpp

```
#include "my_calculator.hpp"
#include <stdexcept>

void my_calculator::push(double value)
{
    stack.push(value);
}

double my_calculator::top() const
{
    if (stack.empty())
        throw std::runtime_error("stack error");
    return stack.top();
}

double my_calculator::pop()
{
    double value = top();
    stack.pop();
    return value;
}

void my_calculator::add()
{
    push( pop() + pop() );
}

void my_calculator::sub()
{
    double op1 = pop();
    double op2 = pop();
    push( op2 - op1 );
}
```

Calc, del 2

Nå skal vi lage et program med et brukegrensesnitt ved å gjenbruke modulen som vi laget i del 1.

Skriv inn filen til høyre og link med objektfilen my_calculator.o:

```
g++ -c my_calculator.cpp
g++ -c calc2.cpp
g++ -o calc2 my_calculator.o calc2.o
./calc2
```

Prøv å forbedre programmet slik at det også håndterer multiplikasjon og divisjon. Da blir du nok nødt til å endre alle tre filene.

calc2.cpp

```
#include "my_calculator.hpp"

#include <iostream>
#include <sstream>
#include <stdexcept>
#include <stdlib.h>

double to_double(const std::string & str)
{
    std::istringstream ss(str);
    double d;
    ss >> d;
    if ( !(ss >> std::ws).eof() )
        throw std::invalid_argument("huh?");
    return d;
}

int main()
{
    my_calculator calc;
    std::string input;
    while (std::cin >> input) {
        try {
            if (input == "q")
                exit(0);
            if (input == "p")
                std::cout << calc.top() << std::endl;
            else if (input == "+")
                calc.add();
            else if (input == "-")
                calc.sub();
            else
                calc.push(to_double(input));
        } catch(std::exception & ex) {
            std::cout << ex.what() << std::endl;
        }
    }
    return 0;
}
```