INF-0103: Rekursjon

Einar Holsbø, UiT – Norges arktiske universitet

H25

Oppvarming (1): hva husker vi fra stack/heap/call stack?

Oppvarming (2): kunne vi gjort noe annet enn call stack for funksjonskall?

Oversikt

1. The call stack gjør at en funksjon kan kalle seg selv (rekursjon)
2. Dette er en programmeringsteknikk som er ganske effektiv i noen tilfeller
3. Den kan også være ganske ineffektiv

En rekursiv funksjon er en som kaller seg selv

Fakultetsfunksjonen $n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$

En rekursiv funksjon er en som kaller seg selv

Fakultetsfunksjonen $n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$

Eller: $n! = n \times (n-1)!$, med $1! = 1$ (viktig).

En rekursiv funksjon er en som kaller seg selv

Fakultetsfunksjonen $n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$

Eller: $n! = n \times (n-1)!$, med $1! = 1$ (viktig).

Eller: $f(n) = n \times f(n-1)$, med $f(1) = 1$ (viktig).

En rekursiv funksjon er en som kaller seg selv

Fakultetsfunksjonen $n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$

Eller: $n! = n \times (n-1)!$, med $1! = 1$ (viktig).

Eller: $f(n) = n \times f(n-1)$, med $f(1) = 1$ (viktig).

```
int factorial(n) {
 // hva putter vi her?
}
```

Call stack for factorial(4)

```
int factorial(n) {
  if (n == 1) return 1;
  return n*factorial(n-1);
}
```

```
factorial(4)

----- STACK -------
|                  |
|                  |
|                  |
|                  |
|                  |<-- toppen av stack
--------------------
```

Call stack for factorial(4)

```
int factorial(n) {
  if (n == 1) return 1;
  return n*factorial(n-1);
}
```

```
factorial(4)

----- STACK -------
|                  |
|                  |
|                  |
|                  |<-- toppen av stack
|  factorial(4)    |
--------------------
```

Call stack for factorial(4)

```
int factorial(n) {
  if (n == 1) return 1;
  return n*factorial(n-1);
}
```

```
factorial(3)

----- STACK -------
|                  |
|                  |
|                  |
|                  |<-- toppen av stack
|  factorial(4)    |
--------------------
```

Call stack for factorial(4)

```
int factorial(n) {
  if (n == 1) return 1;
  return n*factorial(n-1);
}
```

```
factorial(3)

----- STACK -------
|                  |
|                  |
|                  |<-- toppen av stack
|  factorial(3)    |
|  factorial(4)    |
--------------------
```

Call stack for factorial(4)

```
int factorial(n) {
  if (n == 1) return 1;
  return n*factorial(n-1);
}
```

```
factorial(2)

----- STACK -------
|                  |
|                  |
|                  |<-- toppen av stack
|  factorial(3)    |
|  factorial(4)    |
--------------------
```

Call stack for factorial(4)

```
int factorial(n) {
  if (n == 1) return 1;
  return n*factorial(n-1);
}
```

```
factorial(2)

----- STACK -------
|                  |
|                  |<-- toppen av stack
|  factorial(2)    |
|  factorial(3)    |
|  factorial(4)    |
--------------------
```

Call stack for factorial(4)

```
int factorial(n) {
  if (n == 1) return 1;
  return n*factorial(n-1);
}
```

```
factorial(1)

----- STACK -------
|                  |
|                  |<-- toppen av stack
|  factorial(2)    |
|  factorial(3)    |
|  factorial(4)    |
--------------------
```

Call stack for factorial(4)

```
int factorial(n) {
  if (n == 1) return 1;
  return n*factorial(n-1);
}
```

```
factorial(1)

----- STACK -------
|                 |<-- toppen av stack
|  factorial(1)   |
|  factorial(2)   |
|  factorial(3)   |
|  factorial(4)   |
-------------------
```

Call stack for factorial(4)

```
int factorial(n) {
  if (n == 1) return 1;          //// NB!!!!!!!!!!!!
  return n*factorial(n-1);
}
```

```
factorial(1) returnerer 1

----- STACK -------
|                 |<-- toppen av stack
|  factorial(1)   |
|  factorial(2)   |
|  factorial(3)   |
|  factorial(4)   |
-------------------
```

Call stack for factorial(4)

```
int factorial(n) {
  if (n == 1) return 1;          //// NB!!!!!!!!!!!!
  return n*factorial(n-1);
}
```

```
factorial(2) returnerer 2*1 = 2

----- STACK -------
|                  |
|                  |<-- toppen av stack
|  factorial(2)    |
|  factorial(3)    |
|  factorial(4)    |
--------------------
```

Call stack for factorial(4)

```
int factorial(n) {
  if (n == 1) return 1;          //// NB!!!!!!!!!!!!
  return n*factorial(n-1);
}
```

```
factorial(3) returnerer 3*2 = 6
```

```
----- STACK -------
|                  |
|                  |
|                  |<-- toppen av stack
|  factorial(3)    |
|  factorial(4)    |
--------------------
```

Call stack for factorial(4)

```
int factorial(n) {
  if (n == 1) return 1;          //// NB!!!!!!!!!!!!
  return n*factorial(n-1);
}
```

```
factorial(4) returnerer 4*6 = 24

----- STACK -------
|                  |
|                  |
|                  |
|                  |<-- toppen av stack
|  factorial(4)    |
--------------------
```

Call stack for factorial(4)

```
int factorial(n) {
  if (n == 1) return 1;          //// NB!!!!!!!!!!!!
  return n*factorial(n-1);
}
```

```
factorial(4) returnerer 4*6 = 24

----- STACK -------
|                  |
|                  |
|                  |
|                  |
|                  |<-- toppen av stack
--------------------
```

Q: i en einARM64 maskin har hver funksjon en fast plass i minnet til sin “stack frame”. Kan vi ha rekursjon på en slik maskin?

Q: i en einARM64 maskin har hver funksjon en fast plass i minnet til sin “stack frame”. Kan vi ha rekursjon på en slik maskin?

A: i utgangspunktet nei

Aktivitet: Fibonacci-rekka https://bit.ly/3BivCao

- $F_n = F_{n-1} + F_{n-2}$
- $F_1 = F_2 = 1$
- 1 1 2 3 5 8 13 ...

Aktivitet: Fibonacci-rekka https://bit.ly/3BivCao

- $F_n = F_{n-1} + F_{n-2}$
- $F_1 = F_2 = 1$
- 1 1 2 3 5 8 13 ...

- **Oppgaven:** lag en rekursiv funksjon `int fib(int n) { ... }` som regner ut det `n`-te Fibonacci-tallet. Hva er F_{20}?
- Spm:
  1. hvorfor er dette så ineffektivt?
  2. hvor stor `n` fungerer dette for?
  3. hvor mange funksjonskall gjør vi?

"Extra credit:" Søk i sortert array med rekursjon:
https://tinyurl.com/binarysearch2023