Pengenalan rekursi

Tim Olimpiade Komputer Indonesia

Pengenalan Rekursi

- Rekursi adalah keadaan yang mana sebuah fungsi menyelesaikan sebuah permasalahan dengan cara memanggil diri sendiri secara berulang kali.
- Jika masalah sudah cukup kecil, maka fungsi rekursi dapat langsung menghasilkan jawaban.
- Jika masalah terlalu besar, maka fungsi akan memanggil diri sendiri dengan cakupan masalah yang lebih kecil.

Mengapa Perlu Ada Rekursi

- Banyak permasalahan yang lebih mudah diselesaikan dan pendek kodenya jika menggunakan pendekatan rekursif.
- Pada dasarnya, strategi iteratif (misalnya dengan *for loop*) dan rekursif sama-sama melakukan sesuatu yang berulang-ulang.
- Namun, terkadang solusi iteratif untuk suatu masalah sangat sulit untuk dipikirkan dan memerlukan teknik khusus.
- Dengan solusi rekursif, mungkin saja lebih mudah untuk melihat dan merancang alur penyelesaiannya.

Strategi Rekursif

Terdapat dua hal yang perlu dipikirkan ketika menggunakan strategi rekursif:

- *Base case*
  Apa kasus paling sederhana dari permasalahan ini?
- *Recurrence relation*
  Bagaimana hubungan rekursif dari persoalan ini dengan persoalan serupa yang lebih kecil?

Contoh Soal: Faktorial

Deskripsi:

- Pak Dengklek baru mempelajari konsep matematika baru, yaitu faktorial.
- Operasi faktorial pada N, atau ditulis dengan notasi $N!$, adalah operasi mengalikan bilangan dari 1 sampai dengan N.
- Contoh: Jika $N = 4$, maka $4! = 1 \times 2 \times 3 \times 4 = 24$
- Diberikan N, bantu Pak Dengklek mencari hasil $N!$

Contoh Soal: Faktorial (lanj.)

Format masukan:

- Sebuah baris berisi sebuah bilangan N

Format keluaran:

- Sebuah baris berisi hasil $N!$

Batasan:

- $1 \leq N \leq 10$

Solusi

- Ide 1:
  - Cukup gunakan *for loop* biasa
  - Solusi ini bekerja secara iteratif.
- Ide 2: Rekursi

Contoh Solusi Iteratif

Implementasi solusi secara iteratif cukup sederhana:

```
int faktorial(int x) {
  int jawaban = 1;
  for (int i = 2; i <= x; i++) {
    jawaban *= i;
  }
  return jawaban;
}
```

Solusi Rekursif

Base Case

- Pada batasan soal, nilai N berkisar antara 1 sampai dengan 10.
- Dari batasan tersebut, kasus terkecilnya adalah $N = 1$.
- Jadi $N = 1$ adalah *base case*, dan memang jelas diketahui bahwa $1! = 1$.

Solusi Rekursif (lanj.)

Recurrence Relation

- Bagaimana jika $N > 1$?
- Untuk mencari $N!$, kita bisa mencari $(N-1)!$ dan mengalikannya dengan N.
- Jadi persoalan mencari $N!$ bisa diselesaikan dengan mudah jika diketahui $(N-1)!$.
- Dengan observasi ini, kita mengetahui hubungan rekursif dari $N!$.

Contoh Solusi: faktorial_rekursif.cpp

Berikut implementasi pencarian faktorial secara rekursif:

```
int faktorial(int x) {
  if (x == 1) {
    return 1;
  } else {
    return x * faktorial(x-1);
  }
}
```

Contoh Solusi: faktorial_rekursif.cpp (lanj.)

Pemanggilan pada program utama bisa dilakukan seperti memanggil fungsi biasa:

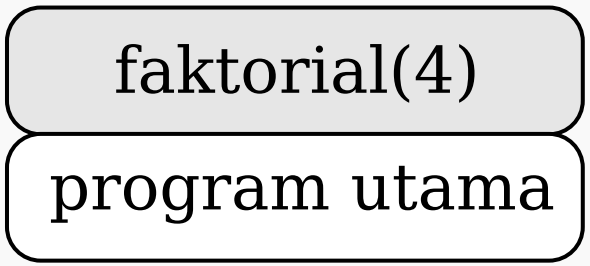
```
...

int main() {
    printf("4! = %d\n", faktorial(4));
}
```

Contoh Eksekusi Fungsi

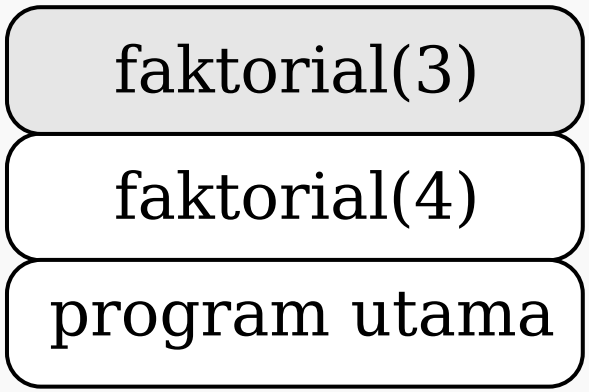
- Pada awalnya, program utama dijalankan. Misalkan hendak dicari nilai 4!.

Contoh Eksekusi Fungsi (lanj.)



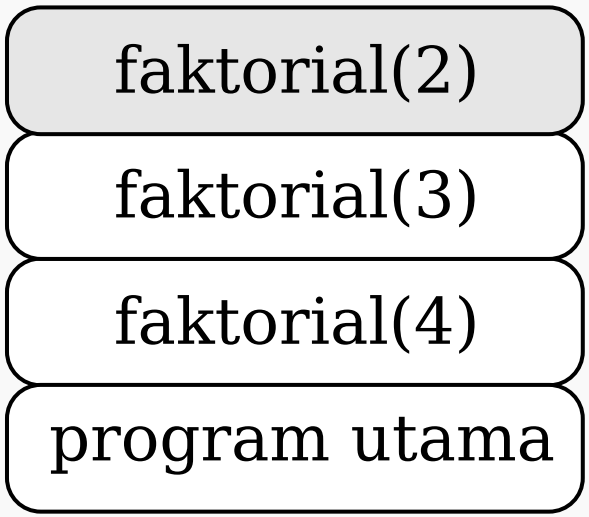
- Dari program utama, dipanggil fungsi **faktorial(4)**.
- Pada saat ini, status program utama adalah "tidak aktif", dan akan "aktif" kembali setelah fungsi **faktorial(4)** selesai.

Contoh Eksekusi Fungsi (lanj.)



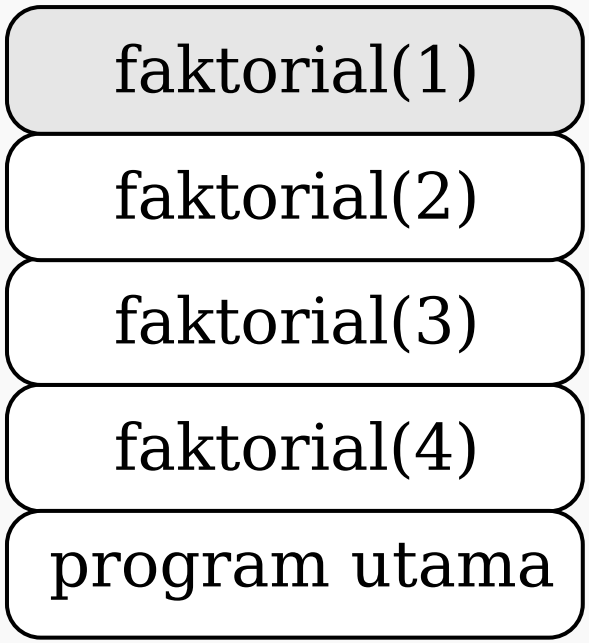
- **faktorial(4)** mengeksekusi "`return x * faktorial(x-1)`", yang pada kasus ini **x** = 4.
- Akibatnya, dipanggil fungsi **faktorial(3)**.
- Kini yang "aktif" adalah **faktorial(3)**.

Contoh Eksekusi Fungsi (lanj.)



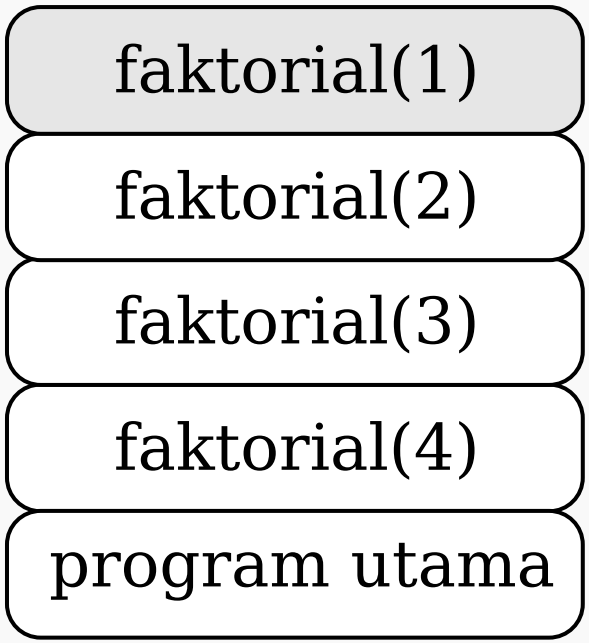
- Hal serupa terjadi untuk mencari nilai **faktorial(3)**.

Contoh Eksekusi Fungsi (lanj.)



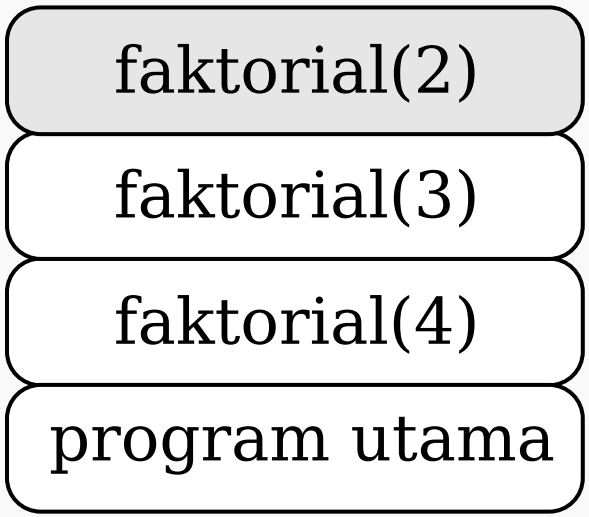
- Terjadi juga untuk mencari nilai **faktorial(2)**.

Contoh Eksekusi Fungsi (lanj.)



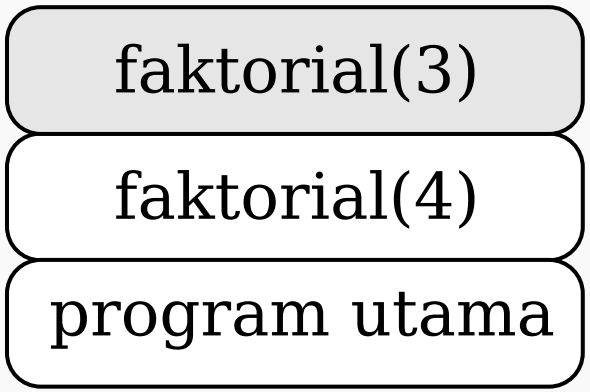
- Pada saat ini, **faktorial(1)** tidak lagi melakukan pemanggilan rekursif, berhubung ditemui *base case*.
- Sebaliknya, langsung dikembalikan nilai 1 sebagai jawaban atas **faktorial(1)**.

Contoh Eksekusi Fungsi (lanj.)



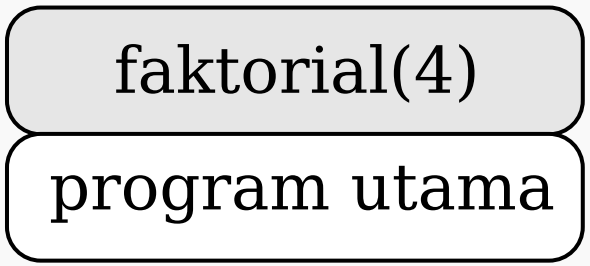
- **faktorial(1)** selesai, kini kembali ke **faktorial(2)**.
- Nilai **faktorial(2)** kini ditemukan, yaitu 2 $\times$ **faktorial(1)**.
- Fungsi **faktorial(2)** mengembalikan nilai 2 ke pemanggilnya, lalu selesai.

Contoh Eksekusi Fungsi (lanj.)



- Setelah menerima nilai kembalian **faktorial(2)**, **faktorial(3)** kembali aktif.
- Hasilnya dapat ditemukan, yaitu $3 \times$ **faktorial(2)**.
- Fungsi **faktorial(3)** mengembalikan nilai 6 ke pemanggilnya, lalu selesai.

Contoh Eksekusi Fungsi (lanj.)



- Kini **faktorial(4)** kembali aktif.
- Hasilnya dapat ditemukan, yaitu $4 \times$ **faktorial(3)**.
- Fungsi **faktorial(4)** mengembalikan nilai 24 ke pemanggilnya, lalu selesai.

Contoh Eksekusi Fungsi (lanj.)

program utama

- Program utama yang memanggil **faktorial(4)** menerima nilai kembaliannya, yaitu 24.
- Program utama kembali menjalankan perintah-perintah berikutnya.

Kompleksitas Solusi

- Baik secara iteratif maupun rekursif, kompleksitasnya adalah $O(N)$.
- Setiap pemanggilan rekursif membutuhkan alokasi memori, sehingga jika pemanggilannya semakin dalam, semakin banyak tambahan memori yang digunakan.
- Waktu untuk mengalokasikan memori juga menyebabkan solusi rekursif cenderung bekerja lebih lambat dibandingkan solusi iteratif.

Materi Selanjutnya

- Pada pembelajaran ini, rekursi yang digunakan masih sangat sederhana.
- Bahkan belum terasa bahwa solusi rekursi lebih mudah dan pendek kodenya dibandingkan solusi iteratif.
- Pembelajaran selanjutnya tentang rekursi yang lebih kompleks akan menunjukkan hal tersebut.