TIF1101 – Dasar-Dasar Pemrograman

HO 12 - Fungsi Rekursif

Opim Salim Sitompul

Department of Information Technology
Universitas Sumatera Utara

Outline

Pendahuluan

- Fungsi rekursif adalah suatu fungsi yang untuk melaksanakan suatu tugas tertentu, kembali memanggil dirinya sendiri.
  - Contoh: untuk menghitung berapakah 7 pangkat 5, sebuah fungsi akan memanggil kembali dirinya untuk menghitung berapakah 7 pangkat 4, untuk menghitung berapakah 7 pangkat 4, fungsi kembali menghitung berapakah 7 pangkat 3, dst., hingga dipenuhi syarat berhenti.
  - Dalam contoh ini adalah setelah menghitung 7 pangkat 1 atau tujuh pangkat 0 yang diketahui hasilnya adalah 1.

Pendahuluan

- Dalam hal-hal tertentu, rekursi lebih disenangi daripada bentuk iteratif yang menggunakan struktur pengulangan, karena fungsi rekursif menggunakan variabel yang lebih sedikit dibandingkan fungsi iteratif.
- Fungsi-fungsi rekursif menangani variabel dan argumennya melalui tumpukan (*stack*).

Pendahuluan

- Akan tetapi penumpukan variabel ini walaupun tidak terlihat oleh pemakai memerlukan waktu dan memori yang lebih banyak.
- Karena itu, dalam hal efisiensi pemakaian waktu dan memori, pemrogram lebih cenderung untuk menggunakan fungsi-fungsi iteratif.

Syarat-syarat Rekursi

- Agar sebuah fungsi rekursif tidak melakukan pemanggilan secara terus-menerus, maka harus diberikan beberapa syarat yang menentukan kapan pemanggilan rekursif itu dihentikan:
  - Harus ada kasus penghentian (termination case)
  - Tiap pemanggilan harus lebih sederhana daripada pemanggilan sebelumnya.
  - Terdapat sebuah "pemicu" yang memulai pemanggilan rekursif.

Syarat-syarat Rekursi

```
1  /* Nama file: rsuka.c
2     Program rekursif sederhana */
3
4  #include <stdio.h>
5
6  void hitung(int); /* prototipe fungsi */
7
8  int main()
9  {
10   hitung(1);   /* pemicu */
11
12   return 0;
13 }
```

Syarat-syarat Rekursi

```
14 void hitung(int n)
15 {
16   printf("%d Saya suka rekursif!\n",n);
17
18   if (n < 5)  /* termination case */
19     hitung(n+1);  /* pemanggilan rekursif */
20
21   printf("%d Rekursif saya suka!\n",n);
22 }
```

Proses Rekursi

- Untuk memahami bagaimana proses yang dilakukan oleh kompiler pada waktu mengeksekusi fungsi rekursi:
  1. Semua variabel otomatis yang terdapat di dalam fungsi rekursif memiliki lokasi memori sendiri pada setiap pemanggilan.
  2. Pernyataan-pernyataan yang terdapat sebelum pemanggilan rekursif akan dieksekusi dengan urutan yang sama.
  3. Pernyataan-pernyataan yang terdapat setelah pemanggilan rekursif akan dieksekusi dengan urutan yang terbalik terhadap pemanggilan.
  4. Kode program untuk fungsi rekursif di dalam memori tidak duplikat, walaupun fungsi tersebut dipanggil beberapa kali.

Proses Rekursi

- Pada contoh program rsuka.c. Variabel n akan terbentuk sebanyak 5 kali sesuai dgn jumlah pemanggilan.
- Pernyataan "Saya suka rekursif!" ditampilkan sesuai dgn urutan pemanggilan, dari 1 - 5, pernyataan "Rekursif saya suka!", ditampilkan dgn urutan terbalik thdp pemanggilan, dari 5-1.

Proses Rekursi

- Pada proses rekursi terdapat sebuah konsep yang disebut pemrosesan akhir (*post processing*), yakni pemrosesan yang dilakukan setelah kembali dari pemanggilan.
- Hasil tiap-tiap pemanggilan tergantung pada hasil pemanggilan berikutnya, sehingga setiap kali kembali dari sebuah pemanggilan, akan dilakukan pemrosesan terhadap hasil yang diperoleh untuk menentukan hasil pemanggilan sebelumnya.

Proses Rekursi

```
1  /* Nama file:  rpangkat.c
2     Menghitung perpangkatan bilangan bulat */
3  #include <stdio.h>
4  long pangkat(int, int);
5
6  int main()
7  {
8    int x, n, y;
9
10   printf("Berikan bil. pokok dan eksponen: ");
11   scanf("%d %d", &x, &n);
12   y = pangkat(x, n);  /* trigger */
13   printf("%d pangkat %d = %ld\n", x, n, y);
14
15   return 0;
16 }
```

Proses Rekursi

```
17 long pangkat(int m, int n)
18 {
19   long p=1;
20
21   if (n > 0)
22     p = m * pangkat(m, n-1);
23
24   return p;
25 }
```

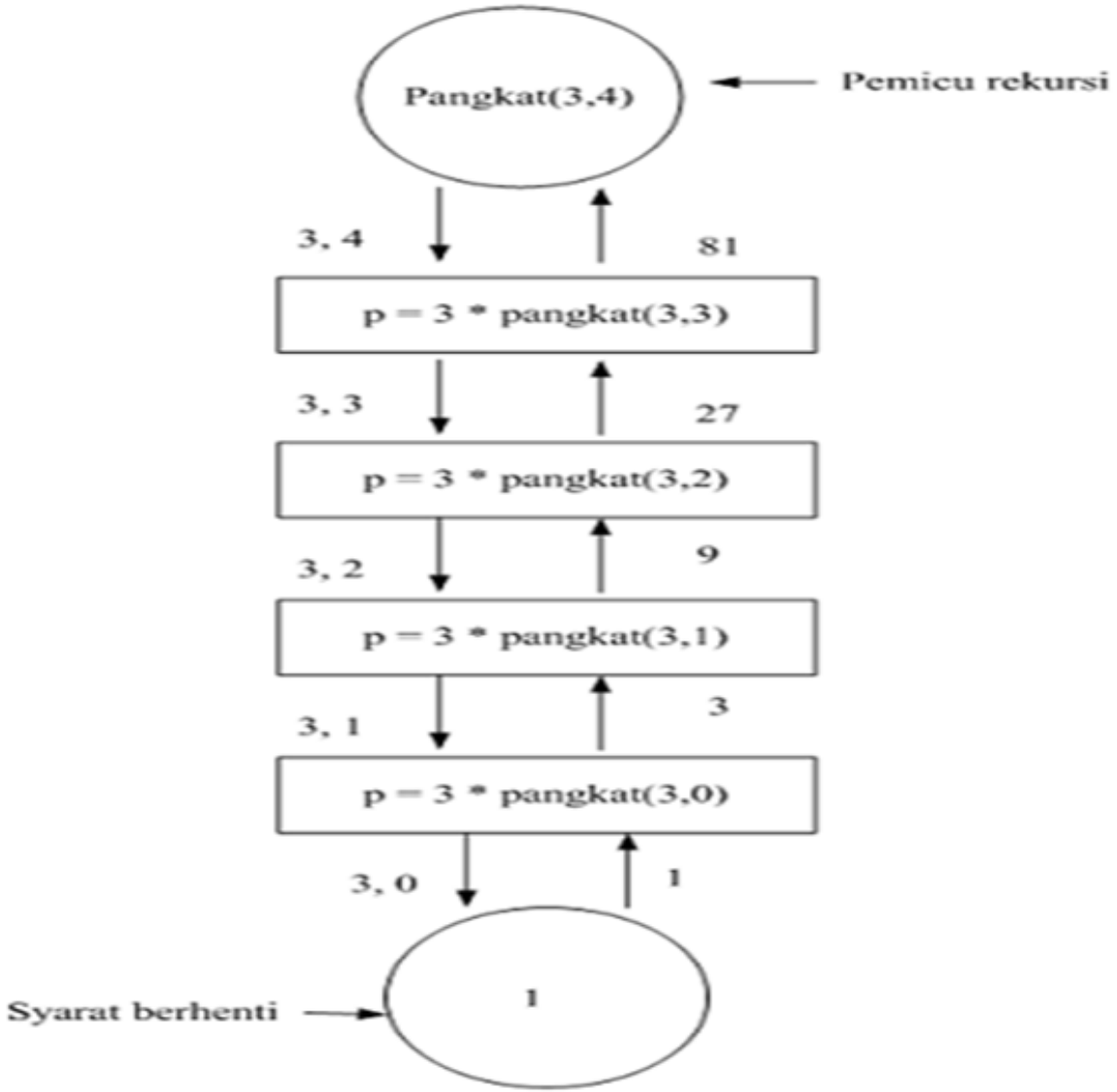
Proses Rekursi

- Fungsi main() memanggil fungsi pangkat() dengan mengirim argumen 3 dan 4.
- Untuk menghitung 3^4 terlebih dahulu dihitung 3^3, untuk menghitung 3^3 terlebih dahulu dihitung 3^2, untuk menghitung 3^2 telebih dahulu dihitung 3^1, untuk menghitung 3^1 terlebih dahulu dihitung 3^0.
- Setelah ini, pemanggilan rekursi dihentikan karena telah mencapai kasus penghentian dan sudah dapat ditentukan bahwa $3^0 = 1$.

Proses Rekursi

- Hasil perpangkatan dikembalikan untuk menghitung $3^1 = 3 \times 3^0$, hasilnya dikembalikan untuk menghitung $3^2 = 3 \times 3^1 = 9$, hasilnya dikembalikan untuk menghitung $3^3 = 3 \times 3^2 = 27$, hasilnya dihitung untuk menghitung $3^4 = 3 \times 3^3 = 81$.
- Nilai tersebut kemudian dikembalikan ke fungsi main().
- Perhitungan pada waktu mengembalikan nilai setelah pemanggilan disebut pemrosesan akhir.

Proses Rekursi



Jenis-jenis Rekursi

- Berdasarkan bentuk pemanggilan yang dilakukan, fungsi rekursif dapat dibagi atas tiga kelompok:
  1. Rekursi akhir (*tail recursive*), apabila pemanggilan dilakukan pada bagian akhir fungsi.
  2. Rekursi tunggal (textitsingly recursive), apabila tiap-tiap pemanggilan paling banyak membentuk satu pemanggilan lagi.
  3. Rekursi ganda (*doubly recursive*), apabila tiap pemanggilan membentuk lebih dari satu pemanggilan.

Rekursi Akhir

- Rekursi akhir adalah bentuk yang paling efisien dan sederhana karena tidak terdapat pemrosesan akhir.
- Tanpa pemrosesan akhir, kompiler tidak perlu menyediakan stack untuk menampung variabel yang diperlukan pada waktu pemrosesan setelah kembali dari sebuah pemanggilan.
- Stack yang diperlukan hanya untuk kerangka fungsi pada setiap pemanggilan.

Rekursi Akhir

```
1  /* Namafile: rTail.c
2     Contoh tail rekursif */
3  #include <stdio.h>
4  int fTail(int);
5  int main()
6  {
7     int n, hsl;
8
9     printf("Rekurensi bilangan\n");
10    printf("Berikan sebuah blangan bulat: ");
11    scanf("%d", &n);
12    hsl = fTail(n); /* Trigger */
13
14    return 0;
15 }
```

Rekursi Akhir

```
16 int fTail(int n)
17 {
18     if (n == 0) /* Syarat berhenti */
19         n = 0;
20   else
21   {
22     printf("%d ", n); /* Print menurun */
23     return fTail(n-1) + 1; /* rekursif call */
24   }
25
26   return n;
27 }
```

Rekursi Akhir

- Contoh: hailston.c
  1. Program pembangkit bilangan bulat yg dsbt "hailstones."
  2. Bertitik tolak dari sebuah bilangan yang ditentukan, dihasilkan bilangan-bilangan berikutnya.
     Misalkan n_i adalah bilangan bulat positif, untuk i = 0, 1, 2, ... didefinisikan:

$$n_{i+1} = \begin{cases} \frac{n_i}{2}, & \textit{jika } n_i \textit{ genap} \\ 3n_i + 1, & \textit{jika } n_i \textit{ ganjil} \end{cases}$$

  3. Jika bilangan yang dihasilkan dari proses rekursi sebelumnya adalah genap dikirim nilai n/2, jika ganjil dikirim 3n + 1.
  4. Proses rekursi berakhir apabila dicapai nilai n = 1.

Rekursi Akhir

```
1  /* Nama file: hailston.c
2     Bilangan "hailstones" */
3  #include <stdio.h>
4  int hailstone(int);
5
6  int main()
7  {
8     int n;
9
10    printf("\nBerikan bilangan bulat: ");
11    scanf("%d", &n);
12    printf("Hailstone dari %d=\n%d ", n,n);
13    printf("\nJumlah: %d.\n", hailstone(n));
14    return 0;
15 }
```

Rekursi Akhir

```
16 int hailstone(int n)
17 {
18   int n1;
19   static int ct = 1;
20
21   if (n == 1)
22     return 1;
23   else if ((n%2) == 0)
24   {
25     n1 = n/2;
26     printf("%d ", n1);
27     ct++;
28     hailstone(n1);
29   }
```

Rekursi Akhir

```
30    else
31    {
32      n1 = 3*n + 1;
33      printf("%d ", n1);
34      ct++;
35      hailstone(n1);
36    }
37    return ct;
38  }
```

Rekursi Tunggal

- Pada rekursi tunggal, tiap-tiap pemanggilan paling banyak menyebabkan satu kali pemanggilan lagi.
- Sekembalinya dari pemanggilan terakhir, dilakukan pemrosesan untuk menghitung hasil yang harus diberikan ke pemanggilan sebelumnya.

Rekursi Tunggal

```
1  /* Nama file: rgcd.c
2     Program untuk menentukan PPB */
3  #include <stdio.h>
4  int gcd(int, int);
5  int main()
6  {
7    int m, n;
8
9    printf("Greatest Common Divisor\n");
10   printf("Berikan dua buah bilangan bulat: ");
11   scanf("%d %d", &m, &n);
12   printf("GCD dari %d dan %d = %d\n", m, n,
        gcd(m,n));
13
14   return 0;
15 }
```

Rekursi Tunggal

```
16 int gcd(int p, int q)
17 {
18   int r , res;
19
20   r = p % q;
21   if (r == 0)
22     res = q;
23   else
24     res = gcd(q, r);
25   return res;
26 }
```

Rekursi Tunggal

- Pada program berikut diberikan pula sebuah contoh untuk menghitung hasil bagi dari dua buah bilangan positif.
- Pembagian dilakukan dengan metoda pengurangan seperti diperlihatkan berikut ini:

$$\begin{aligned} 17/5 &\rightarrow 17 - 5, \mathit{sisa} = 12 \\ &\rightarrow 12 - 5, \mathit{sisa} = 7 \\ &\rightarrow 7 - 5, \mathit{sisa} = 2 \end{aligned}$$

- Setelah sisa pembagian lebih kecil dari pembagi atau sama dengan nol, pengurangan dihentikan.
- Hasil bagi adalah banyaknya pengurangan yang dilakukan, dan sisa pembagian adalah bilangan yang diperoleh pada pengurangan terakhir.
- Pada contoh di atas, hasil bagi = 3, sisa = 2.

Rekursi Tunggal

```
1  /* Nama file: rbagi.c
2     Program menghitung hasil bagi bil. bulat */
3  #include <stdio.h>
4  #include <stdlib.h>
5  unsigned RBagi(unsigned,unsigned,unsigned *);
6  int main()
7  {
8    unsigned m, n, hb, *sisa;
9
10   printf("Pembagian\n");
11   printf("Berikan 2 bil. bulat positip: ");
12   scanf("%u %u", &m, &n);
13   sisa = (unsigned *)malloc(sizeof(unsigned));
14   *sisa = 0;
15   hb = RBagi(m, n, sisa);
```

Rekursi Tunggal

```
16    printf("%d/%d=%u %u/%u.\n",m,n,hb,*sisa,n);
17    return 0;
18  }
19
20  unsigned RBagi(unsigned yb, unsigned pb,
        unsigned *si)
21  {
22    unsigned hb = 0;
23    if(pb > yb)
24    {
25        *si = yb;
26        hb = 0;
27    }
28    else
29        hb += RBagi(yb - pb, pb, si) + 1;
30    return hb;
```

Rekursi Tunggal

- Program di atas menggunakan sebuah variabel pointer untuk menyimpan sisa pembagian.
- Untuk mengalokasikan memori tempat penyimpanan data pada variabel ini, digunakan fungsi *malloc*() yg dideklarasikan pada file judul *stdlib.h*.
- Alamat lokasi memori variabel ini kemudian dikirim ke fungsi RBagi().

Rekursi Tunggal

- Hal ini diperlukan karena program perhitungan hasil bagi tersebut perlu menggunakan sebuah fungsi yang mengembalikan dua buah nilai, yaitu hasil bagi dan sisa.
- Jadi, pernyataan **return** untuk mengembalikan hasil bagi, pointer untuk mengembalikan sisa.

Rekursi Ganda

- Bentuk rekursi yg paling rumit di antara kedua bentuk rekursi sebelumnya.
- Pada rekursi ini masing2 pemanggilan akan mengakibatkan terjadinya dua atau lebih pemanggilan lagi.
- Pemrosesan akhir juga akan dilakukan pada bentuk ini.

Rekursi Ganda

```
1  /* Nama file: rfibo.c
2     Program untuk menentukan suku fibonacci */
3  #include <stdio.h>
4  int rfibo(int);
5
6  int main()
7  {
8    unsigned n;
9    int f;
10
11   printf("Deret Fibonacci\n");
12   printf("Suku deret fibonacci ke berapa? ");
13   scanf("%d", &n);
14   printf("Suku Fibonacci ke %d adalah", n);
```

Rekursi Ganda

```
15    f = rfibo(n);
16    printf(" %d\n", f);
17    return 0;
18  }
19
20  int rfibo(int n)
21  {
22    int i, f=0;
23
24    if (n < 3)
25      return 1;
26    else
27      f = f + (rfibo(n-1) + rfibo(n-2));
28    return f;
29  }
```

Latihan Pemrograman

- Buatlah sebuah program C untuk menjumlahkan digit-digit yang terdapat dalam sebuah bilangan secara rekursif.
- Contoh:
  Jumlah digit bilangan 1708 = 16
  Jumlah digit biangan 1105 = 7

Latihan Pemrograman

```
1  /* Nama file: jlhDigit.c
2  Menghitung penjumlahan digit bilangan */
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  int jlhDigit(int);  /* prototype */
8
9  int main()
10 {
11   int bil, hsl;
12
13   printf("Penjumlahan digit bilangan\n");
14   printf("Berikan sebuah bilangan bulat: ");
15   scanf("%d", &bil);
```

Latihan Pemrograman

```
16      hsl = jlhDigit(bil); /* function call */
17
18    printf("Jumlah digit bilangan %d = %d\n",
          bil, hsl);
19
20    return 0;
21  }
22
23  int jlhDigit(int bil)
24  {
25    int sisa, jlh = 0;
26
27    sisa = bil % 10;
28    bil = bil / 10;
```

Latihan Pemrograman

```
29    if(bil == 0)
30      return sisa;
31    else
32      jlh = sisa + jlhDigit(bil);
33
34    return jlh;
35  }
```