

Notasi Algoritmik + Contoh ADT Sederhana

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Notasi algoritmik

Tidak mungkin mempelajari semua bahasa pemrograman → yang penting dipahami: **pola pikir komputasi** dan **paradigma pemrograman**.

Belajar memrogram ≠ Belajar bahasa pemrograman

Notasi Algoritmik: notasi standar yang digunakan untuk menuliskan teks algoritma [dalam paradigma prosedural].

Algoritma adalah solusi detail secara prosedural dari suatu persoalan dalam Notasi Algoritmik.

Program adalah program komputer dalam suatu bahasa pemrograman yang tersedia di dunia nyata.

Perlunya Notasi Algoritmik

Notasi Algoritmik → representasi cara berpikir untuk menyelesaikan persoalan dengan paradigma prosedural.

Membuat program → melihat “kosa kata” translasi notasi algoritmik ke bahasa pemrograman yang dipilih.

Dengan notasi algoritmik yang sama, bisa dibuat program dalam bahasa pemrograman yang berbeda, dengan paradigma pemrograman yang sama.

Contoh: untuk prosedural: C, Pascal, Fortran.

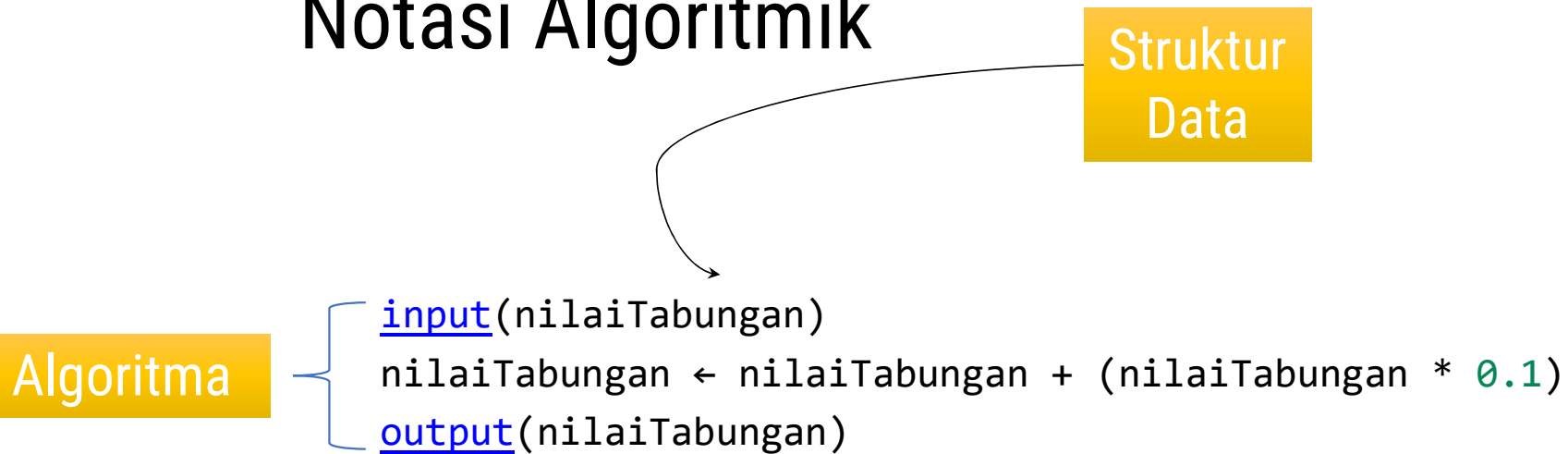
Contoh Kasus: Tabungan

Program = Algoritma + Struktur Data

Notasi Algoritmik

Struktur
Data

Algoritma



```
input(nilaiTabungan)  
nilaiTabungan ← nilaiTabungan + (nilaiTabungan * 0.1)  
output(nilaiTabungan)
```

Kode Program Bahasa C

```
input(nilaiTabungan)  
nilaiTabungan ← nilaiTabungan + (nilaiTabungan * 0.1)  
output(nilaiTabungan)
```



scanf akan membaca
dari hasil ketik di
keyboard

→ `scanf("%f", &nilaiTabungan);`

`nilaiTabungan += nilaiTabungan * 0.1;`

printf akan menulis hasil
di layar komputer

→ `printf("%f\n", nilaiTabungan);`

Kode Program Bahasa Pascal

```
input(nilaiTabungan)  
nilaiTabungan ← nilaiTabungan + (nilaiTabungan * 0.1)  
output(nilaiTabungan)
```



readln akan membaca
dari hasil ketik di
keyboard → `readln(nilaiTabungan);`

`nilaiTabungan := nilaiTabungan +
 nilaiTabungan * 0.1;`
writeln akan menulis
hasil di layar komputer → `writeln(nilaiTabungan);`

Kode Program Bahasa Python

```
input(nilaiTabungan)
nilaiTabungan ← nilaiTabungan + (nilaiTabungan * 0.1)
output(nilaiTabungan)
```



input akan membaca
dari hasil ketik di
keyboard

—————→ `nilaiTabungan = float(input())`

`nilaiTabungan += nilaiTabungan * 0.1`

print akan menulis hasil
di layar komputer

—————→ `print(nilaiTabungan)`

Struktur program notasi algoritmik dalam contoh

Program Penjumlahan

{ Spesifikasi Program: menghitung $a + b$ }

KAMUS

{ Deklarasi variabel }

a, b: integer

ALGORITMA

input(a)

input(b)

$a \leftarrow a + b$

output(a)

Contoh spesifikasi ADT dengan notasi algoritmik

Struktur data

```
{ Modul ADT Time }  
type Time: < hours: integer[0..23],  
              minutes: integer[0..59],  
              seconds: integer[0..59] >
```

Possible values

```
{ 0 ≤ hours ≤ 23 }  
{ 0 ≤ minutes ≤ 59 }  
{ 0 ≤ seconds ≤ 59 }
```

Deklarasi operasi

```
{ Konstruktor: membentuk Time dari komponen-komponennya: h sebagai hours, m sebagai  
  minutes, dan s sebagai seconds. }  
procedure CreateTime(output t: Time, input h: integer[0..23],  
                    input m: integer[0..59], input s: integer[0..59])  
{ Mendapatkan komponen hours dari T }  
function getHours(T: Time) → integer[0..23]  
{ Mendapatkan komponen minutes dari T }  
function getMinutes(T: Time) → integer[0..59]  
{ Mendapatkan komponen seconds dari T }  
function getSeconds(T: Time) → integer[0..59]  
{ Selisih antara dua Time, dalam satuan detik }  
function difference(start: Time, end: Time) → integer
```

Contoh realisasi operasi dalam notasi algoritmik

Signature operasi

Prasvarat operasi

Ekspektasi hasil operasi pada setiap rentang kemungkinan masukan (*test case*)

`(start, end: Time) → integer`

`time start dan end, dengan syarat $start \leq end$. }`

{ EXPECT

`CreateTime(start, 1,2,3); CreateTime(end, 2,3,4)`

`⇒ difference(start,end) = 3661`

`CreateTime(start, 2,3,4); CreateTime(end, 2,3,4)`

`⇒ difference(start,end) = 0`

`CreateTime(start, 2,3,4); CreateTime(end, 1,2,3)`

Body operasi

`⇒ difference(start,end) = tak terdefinisi }`

KAMUS LOKAL

`startSec, endSec: integer`

ALGORITMA

`startSec ← getHours(start)*60*60 + getMinutes(start)*60 + getSeconds(start)`

`endSec ← getHours(end)*60*60 + getMinutes(end)*60 + getSeconds(end)`

`→ endSec-startSec`

*Spesifikasi lengkap notasi algoritmik terdapat pada dokumen terpisah.

Operasi primitif dan penunjang

Operasi primitif

Beberapa operasi sangat bergantung pada struktur data yang digunakan.

Operasi-operasi tersebut adalah operasi primitif.

Pada contoh ADT Time sebelumnya, `CreateTime`, `getHours`, `getMinutes`, dan `getSeconds` akan memiliki implementasi yang berbeda jika Time menggunakan representasi detik saja.

Sementara itu, operasi selisih tidak harus bergantung pada struktur data karena implementasinya dapat memanfaatkan primitif yang sudah ada.

(Tidak mempertimbangkan efisiensi algoritma.)

Contoh: ADT Time alt-1 dan CreateTime

```
{ Modul ADT Time alt-1 }
```

```
type Time: < hours: integer[0..23],    { 0 ≤ hours ≤ 23 }  
             minutes: integer[0..59],    { 0 ≤ minutes ≤ 59 }  
             seconds: integer[0..59] > { 0 ≤ seconds ≤ 59 }
```

```
{ Konstruktor: membentuk Time t dari komponen-komponennya: h sebagai hours, m sebagai  
  minutes, dan s sebagai seconds. }
```

```
procedure CreateTime(output t: Time, input h: integer[0..23],  
                    input m: integer[0..59], input s: integer[0..59])
```

KAMUS

-

ALGORITMA

```
t.hours ← h  
t.minutes ← m  
t.seconds ← s
```

Contoh: ADT Time alt-2 dan CreateTime

```
{ Modul ADT Time alt-2 }
```

```
type Time: < seconds: integer[0..86399] > {  $0 \leq \text{seconds} \leq 86400$  }
```

```
{ Konstruktor: membentuk Time t dari komponen-komponennya: h sebagai hours, m sebagai minutes, dan s sebagai seconds. }
```

```
procedure CreateTime(output t: Time, input h: integer[0..23],  
                    input m: integer[0..59], input s: integer[0..59])
```

KAMUS

-

ALGORITMA

```
t.seconds  $\leftarrow$  h*60*60 + m*60 + s
```

Contoh: implementasi getHours(t)

```
{ alt-1 }  
{ Mendapatkan bagian hours dari t }  
function getHours(t: Time) → integer[0..23]
```

KAMUS

-

ALGORITMA

→ t.hours

```
{ alt-2 }  
{ Mendapatkan bagian hours dari t }  
function getHours(t: Time) → integer[0..23]
```

KAMUS

-

ALGORITMA

→ t.seconds div (60*60)

Contoh: selisih dua waktu

function difference(start: Time, end: Time) → integer
{ Menghasilkan selisih antara dua Time start dan end, dengan syarat $\text{start} \leq \text{end}$. }

{ **EXPECT**

CreateTime(start, 1,2,3); CreateTime(end, 2,3,4)

⇒ difference(start,end) = 3661

CreateTime(start, 2,3,4); CreateTime(end, 2,3,4)

⇒ difference(start,end) = 0

CreateTime(start, 2,3,4); CreateTime(end, 1,2,3)

⇒ difference(start,end) = tak terdefinisi }

KAMUS LOKAL

startSec, endSec: integer

ALGORITMA

startSec ← getHours(start)*60*60 + getMinutes(start)*60 + getSeconds(start)

endSec ← getHours(end)*60*60 + getMinutes(end)*60 + getSeconds(end)

→ endSec-startSec

Latihan

Ambil sebuah program yang pernah Anda tulis pada kuliah **Dasar Pemrograman** bagian pemrograman prosedural, yang minimal memiliki:

Percabangan

Perulangan

Prosedur/fungsi

Tulis kembali program tersebut dalam **Notasi Algoritmik**.

Gunakan diktat kuliah sebagai referensi Notasi Algoritmik.

Tujuan: mengenali dan membiasakan diri menulis program dalam Notasi Algoritmik.