

Pengantar Abstract Data Type

Tim Pengajar IF1210

Sekolah Teknik Elektro dan Informatika

Algoritma, Struktur Data, dan *Abstract Data Type* (ADT)

Syarat

- Dasar pemrograman (imperatif ~ prosedural):
 - Variabel, assignment
 - Array
 - Control flow:
 - Percabangan
 - Pengulangan
 - Fungsi & Prosedur

Algoritma

Definisi umum: suatu prosedur, **langkah demi langkah**, untuk *menyelesaikan suatu masalah* atau *mencapai sebuah tujuan*.

Definisi khusus: suatu rangkaian **terhingga** dari instruksi-instruksi yang **terdefinisi** dan dapat diimplementasikan pada komputer untuk *menyelesaikan himpunan permasalahan spesifik* yang **computable**.

- **Terhingga**: langkah-langkah harus berhenti *at some point*.
- **Terdefinisi**: memenuhi semua klausa dari suatu definisi.
- **Computable**: ada himpunan masalah yang belum diketahui apakah dapat diselesaikan dengan komputer atau tidak → bukan lingkup mata kuliah ini.

Contoh: algoritma Euclid untuk mencari FPB.

Struktur Data

Adalah bagaimana data **diorganisasikan** dan **dikelola** agar dapat digunakan secara **efektif** dan **efisien**.

Contoh: merepresentasikan waktu (*time*) pukul 01:02:03

Cara I	Cara II
3 byte: h=01, m=02, s=03 0x01 0x02 0x03 <ul style="list-style-type: none"> • Menghitung selisih waktu lebih rumit • Mencetak waktu dalam format “HH:MM:SS” lebih mudah 	2 byte: detik sejak tengah malam = 3723 0x0e 0x8b <ul style="list-style-type: none"> • Menghitung selisih waktu lebih mudah • Mencetak waktu dalam format “HH:MM:SS” lebih rumit

Dari contoh tersebut juga terlihat bahwa **struktur data yang berbeda** membutuhkan **algoritma yang berbeda** meskipun merepresentasikan **permasalahan yang sama**.

“Algoritma + Struktur Data = Program”

(Niklaus Wirth)

Abstract Data Type (ADT)

ADT adalah pemodelan suatu tipe data yang didefinisikan perilakunya berdasarkan:

data yang terkandung di dalamnya,

himpunan nilai yang mungkin dimiliki oleh data tersebut, serta

operasi yang dapat diterapkan terhadap data tersebut.

Implementasi dari suatu ADT mencakup **struktur data** untuk data yang didefinisikan oleh ADT dan **algoritma** untuk setiap operasi terhadap data tersebut.

ADT, struktur data, dan algoritma adalah salah satu mekanisme modularitas (\rightarrow *reusability*) dalam paradigma prosedural.

Modularitas ADT, struktur data, dan algoritma

Menghitung selisih antara dua instan waktu, tanpa ADT. Bayangkan jika operasi ini perlu dilakukan berulang kali di program yang Anda buat:

```
{ waktu #1, 13:45:00 }
h1 ← 13
m1 ← 45
s1 ← 0
{ selisih waktu #2 dengan waktu #1 }
ss1 ← h1*60*60 + m1*60 + s1 ; ss2 ← h2*60*60 + m2*60 + s2
d_ss ← ss2-ss1
```

Dengan ADT:

```
{ waktu #1, 13:45:00 }
CreateTime(t1, 13,45,0)
d_t ← difference(t1,t2)

{ waktu #2, 14:30:00 }
; CreateTime(t2, 14,30,0)
```

Modularitas tanpa ADT?

Arguably, beberapa hal dapat dimodularkan dengan fungsi/prosedur, tanpa ADT:

```

{ waktu #1, 13:45:00 }
h1 ← 13
m1 ← 45
s1 ← 0
{ selisih waktu #2 dengan waktu #1 }
ss1 ← hhmmssToSeconds(h1,m1,s1) ; ss2 ← hhmmssToSeconds(h2,m2,s2)
d_ss ← ss2-ss1
{ atau: }
d_ss ← difference(h2,m2,s2,h1,m1,s1)
  
```

Namun mekanisme ini tidak mencegah pemrogram merangkai waktu secara salah, misalnya: `output(h1 + ":" + m2 + ":" + s1)` yang mencetak "13:30:00", yang secara logik tidak pernah relevan pada program di atas.

Ringkasan: ADT

Membuat suatu ADT mencakup:

1) Spesifikasi:

- **Definisi** tipe data
- **Himpunan nilai** yang mungkin
- Daftar **operasi-operasi** (primitif maupun penunjang)
 - **Prekondisi** yang harus terpenuhi sebelum operasi dimulai
 - **State** setelah operasi selesai

2) Implementasi:

- **Struktur data** konkret
- **Implementasi algoritma** setiap operasi
- **“Test case”** untuk setiap operasi pada setiap rentang kemungkinan nilai

Jenis Struktur Data & ADT Umum

Jenis struktur data umum

record atau *tuple*

- Agregasi beberapa nilai dengan jumlah yang tetap.
- Setiap nilai dapat berbeda tipe.
- Elemennya diakses dari namanya.

array

- Sejumlah elemen yang diletakkan di memori secara kontigu.
- Setiap elemen bertipe data sama.
- Elemennya diakses melalui indeks.

node-based atau *linked* (struktur berkait)

- Sejumlah elemen yang saling terhubung melalui *pointer*.
- Setiap elemen bertipe data sama.
- Elemen diacu oleh elemen sebelumnya.

Record/tuple

Struktur data yang digunakan dalam ADT Time alt-1 adalah contoh ***tuple***.

- Merupakan agregasi dari 3 nilai: **jam**, **menit**, dan **detik**.
- Masing-masing nilai dapat berasal dari tipe logik yang berbeda (**jam** hanya 0-23, sedangkan **menit** dan **detik** 0-59).
- Diakses berdasarkan namanya (misal: **t.hours**).

Dituliskan dengan tanda ‘<’ dan ‘>’, contoh:

Time dapat dituliskan sebagai **tuple** **<hours, minutes, seconds>**

Array

Array harus dialokasikan di memori dengan **ukuran yang sudah ditentukan**.

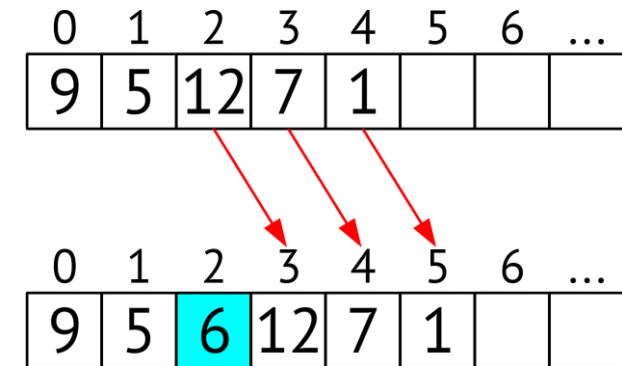
Misal: *array* 10 elemen bertipe integer → dialokasikan memori sebesar 10×4 byte.

Jika perlu mengubah ukuran *array*:

1. buat *array* baru dengan ukuran yang dikehendaki,
2. salin isi *array* lama ke *array* baru,
3. dealokasi *array* lama.

Akses ke setiap elemen *array* dilakukan melalui **indeks** (biasanya dimulai dari 0).

Menambah/menghapus elemen di tengah *array* mengakibatkan elemen-elemen setelah posisi tersebut harus **digeser satu per satu**.



Struktur berkait

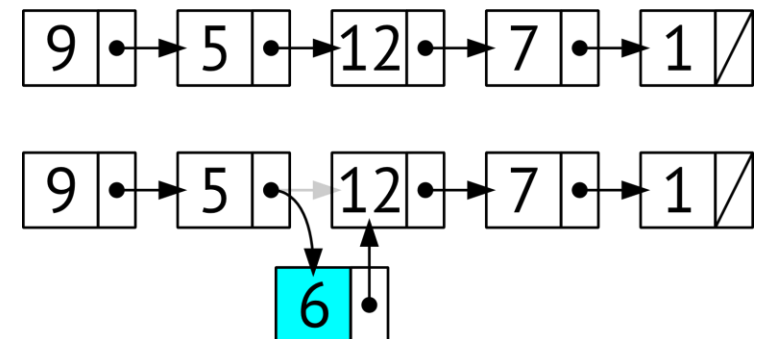
Sebuah *node* adalah ***tuple*** dengan dua elemen:

1. **Nilai** yang hendak disimpan dalam *node*,
2. **Pointer** ke *node* berikutnya.

Di memori, lokasi (alamat) *node* satu dan *node* berikutnya tidak harus bersebelahan.

Akses ke suatu *node* harus **melalui *node-node* sebelumnya**.

Menambah/menghapus elemen di tengah struktur berbasis *node* cukup dengan ***“rerouting”*** pointer-pointer.



Pakai yang mana?

Beberapa jenis data dapat distrukturkan dengan lebih dari satu cara. Contoh:

- Time bisa saja distrukturkan dalam **array** berukuran 3, sehingga bagian jam diakses dengan $t[0]$, menit dengan $t[1]$, dan detik dengan $t[2]$.
→ Konsekuensi: ketiga nilai tersebut kini bertipe sama (e.g., integer).
- Sebuah antrian (elemen yang baru datang harus “berdiri” di belakang, dan elemen yang boleh “dilayani” hanya elemen yang paling depan) dapat distrukturkan dalam bentuk **array** maupun **node-based**.

Pemilihan struktur data berakibat pada *trade-off* **efisiensi algoritma** pada berbagai operasi.

Diperlukan **analisis algoritma** untuk memilih struktur yang lebih efisien.

ADT umum yang akan dibahas

Algoritma dan Pemrograman 1

List/sequence

Matriks

Stack, queue

Set, Multiset

Map

Algoritma dan Pemrograman 2

Tree, binary tree, binary search tree

Graph

ADT dalam Notasi Algoritmik

Contoh spesifikasi ADT dengan notasi algoritmik

```
{ Modul ADT Time }
```

```
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 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 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 }
```

{ 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 spesifikasi ADT dengan notasi algoritmik

```
{ Modul ADT Time }
```

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

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

Contoh realisasi operasi dalam notasi algoritmik

Signature operasi

```
function difference(start: Time, end: Time) → integer
```

```
{ Menghasilkan selisih antara dua 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)
```

```
    ⇒ 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
```

Contoh realisasi operasi dalam notasi algoritmik

Prasyarat operasi

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

Contoh realisasi operasi dalam notasi algoritmik

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

`difference(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)
    ⇒ 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
```

Contoh realisasi operasi dalam notasi algoritmik

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)

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

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.)

Kelompok operator primitif (1/2)

Primitif dalam konteks prosedural diterjemahkan menjadi fungsi dan prosedur dan dikelompokkan menjadi:

- **Konstruktor/kreator:** pembentuk nilai type
 - Semua objek (variabel) bertipe tersebut harus melalui konstruktor. Biasanya fungsi/prosedur dengan nama diawali dengan kata “Create”, “Make”
- **Selektor:** untuk mengakses komponen type
 - Biasanya fungsi dengan nama diawali dengan kata “Get”
- Prosedur **pengubah** komponen type
 - Biasanya prosedur dengan nama diawali dengan kata “Set”
- **Validator** komponen type: mengetes apakah nilai-nilai pembentuk type sesuai dengan batasan
- **Destruktor/dealokator:** untuk “menghancurkan” nilai objek (sekaligus memori penyimpanannya)

Kelompok operator primitif (2/2)

- **Baca/Tulis:** interface dengan input/output device
- **Operator relational** terhadap type untuk mendefinisikan perbandingan dua nilai: lebih besar, lebih kecil, sama dengan, dsb.
- **Operator aritmatika** terhadap type: penjumlahan, pengurangan, dsb.
- **Konversi** type ke type dasar dan sebaliknya

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 ≤ seconds ≤ 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 ← 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

Contoh ADT dalam Notasi Algoritmik

- Diktat Struktur Data
 - ADT JAM (Hlm. 10-12)
 - ADT POINT (Hlm. 12-14)
 - ADT GARIS (Hlm. 14-16) → contoh ADT yang menggunakan ADT lain
- Draf Diktat tersedia di Edunex

Latihan Soal 1

- Ambil ADT POINT yang tersedia di Diktat Struktur Data Hlm. 12-14
- Salin spesifikasi ADT POINT dan lakukan koreksi (jika perlu) dan penyesuaian nama (jika perlu) terhadap operator-operator yang ada.
- Tuliskan implementasi operator-operator yang disediakan.

Modularitas Program dalam Bahasa C

Tujuan

- Mahasiswa memahami kegunaan modul program
- Mahasiswa memahami konsep reusability dalam pembuatan program
- Mahasiswa memahami pembuatan program C dengan beberapa modul program (dalam beberapa file)
- Mahasiswa dapat mengimplementasikan program dengan memakai modul program dalam bahasa C

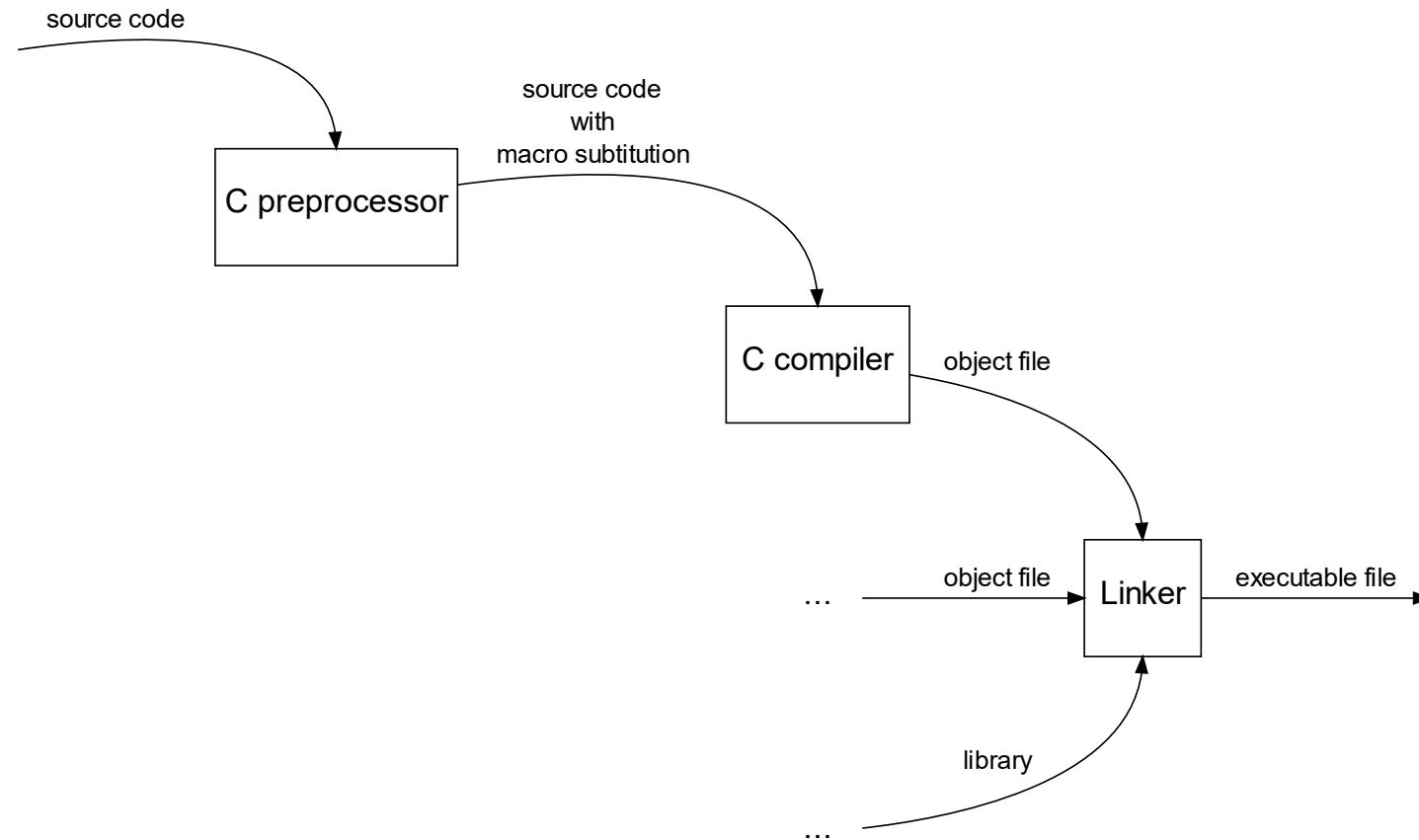
Modularitas Program

- Sebuah program yang “utuh”, seringkali terdiri dari beberapa modul program.
- Modul program dapat mewakili:
 - Sekumpulan **rutin** (prosedur & fungsi) sejenis
 - **ADT** (Abstract Data Type): definisi type dan primitifnya
 - **Mesin** : definisi *state variable* dari mesin dan primitifnya

Pembuatan program

- Program terdiri dari :
 - Satu program utama (*main program*)
 - Beberapa modul yang lain
- Program yang dibagi-bagi menjadi beberapa file seharusnya dapat dikompilasi terpisah.
Setiap modul membentuk sebuah *object code*.
- Pembuatan sebuah *executable code* dilakukan dgn melakukan *linking* terhadap sejumlah *object code* yang sudah dikompilasi.
 - Penghematan waktu dan duplikasi usaha (*reusability*).

Pemrosesan kode sumber dalam Bahasa C



Modul program dalam C (1/3)

Program utuh terdiri dari 4 kelompok file

1. File *header* dengan nama `xxx.h` di folder `src/`

- Untuk setiap type dan primitifnya, ada sebuah file *header*.
- Contoh: untuk ADT Time dan ADT Date ada 2 buah file header, yaitu `time.h` dan `date.h`
- Fungsi selektor (`get*`, `set*`) dapat digantikan dengan macro berparameter. Misalnya untuk selektor `HOURS(t)`, `MINUTES(t)`, dan `SECONDS(t)` dituliskan sebagai:
 - `#define HOURS(t) (t).hours`
 - `#define MINUTES(t) (t).minutes`
 - `#define SECONDS(t) (t).seconds`

Modul program dalam C (2/3)

2. File body dengan nama `xxx.c` di folder `src/`

- Berisi realisasi dari prototype yang didefinisikan dalam file *header*.
- Akan ada sebuah `xxx.c` untuk setiap `xxx.h`
- Contoh : untuk file header `Time.h` dan `Date.h` akan ada file body `Time.c` dan `Date.c`

Modul program dalam C (3/3)

3. File main (driver) di folder `src/`

- Berisi program utama dan prosedur/fungsi lain yang hanya dibutuhkan oleh main
- Misalnya diberi nama `main.c`

4. File unit test di folder `tests/`

- Berisi beberapa *test case* untuk setiap prosedur/fungsi yang ada di header
- Misalnya, menggunakan library check, diberi nama `check_time.c`
- Program Utuh akan terdiri dari sebuah `main.c`, sebuah `xxx.h`, `xxx.c`, dan `check_xxx.c`

File Header

```
/* File : xxx.h */
/* Deskripsi : keterangan isi file header */
/* Isi : deklarasi konstanta, type dan prototype */
/* File header TIDAK BOLEH mengandung deklarasi variabel! */

#ifdef xxx_h
#define xxx_h
/* Bagian I : berisi deklarasi konstanta */

/* Bagian II : berisi deklarasi type */

/* Bagian III : berisi deklarasi prototype prosedur & fungsi */
/* yg merupakan primitif type tsb */
/* Kelompokkan fungsi dan prosedur sesuai standar di kelas */
/* Mis.: konstruktor, selektor, predikat, operator relasional*/
/* operator aritmatika, operator lain, dsb */

#endif
```

File Body

```
/* File : xxx.c */  
/* Deskripsi : keterangan isi file body */  
/* Isi : realisasi/ kode program dari semua prototype */  
/* yg didefinisikan pada xxx.h */  
/* Untuk sebuah mesin akan mengandung deklarasi variabel */  
/* state dari mesin tsb */
```

```
# include "xxx.h"
```

```
/* Realisasi kode program, sesuai urutan pada xxx.h */
```

```
/* Copy dari xxx.h, kemudian edit */
```

File Main Program

```
/* File : main_xxx.c */
/* Deskripsi: program utama & semua nama lokal thd persoalan */

#include "xxx.h"
/* include file lain yg diperlukan */
/* Bagian I : berisi kamus GLOBAL dan prototype */
/* deklarasi semua nama dan prosedur/fungsi global */

/* Bagian II : program utama */
int main() {
    /* Kamus lokal terhadap main */

    /* Algoritma */

    return 0;
}

/* Bagian III : berisi realisasi kode program yang merupakan */
/* BODY dari semua prototype yg didefinisikan pada file ini */
/* yaitu pada bagian I, dengan urutan-urutan yang sama */
/* Copy prototype, kemudian edit */
```

File Unit Test

```
/* File : check_xxx.c */
/* Deskripsi: unit test untuk modul xxx */

#include <check.h>
#include "../src/xxx.h"
/* include file lain yg diperlukan */

START_TEST(test_xxx_namafungsiyangdittest) {
    /* Bagian I: berisi test untuk sebuah fungsi dari modul xxx */
} END_TEST

Suite *xxx_suite(void) {
    /* Bagian II: mengumpulkan semua test menjadi satu test suite */
}

int main(void) {
    /* Bagian III: menjalankan unit test */
}
```

Penyimpanan Modul Program

- Untuk setiap modul xxx.h dan xxx.c dibuat unit test untuk menguji setiap fungsi/prosedur yg dibuat.
- Setiap paket yg terdiri dari xxx.h, xxx.c, unit test (misal: check_xxx.c), dan hasil test disimpan dalam satu direktori.
- **Pelajari** [Check 0.15.2: 3 Tutorial: Basic Unit Testing](#).



Latihan Soal 2

- Buatlah implementasi ADT POINT yang dibuat dalam Latihan Soal 1 ke dalam program Bahasa C, yaitu:
 - point.h
 - point.c
 - mpoint.c
 - check_point.c