Abstract Data Type (ADT)

ADT adalah definisi **TYPE** dan sekumpulan **PRIMITIF** (operasi dasar) terhadap TYPE tersebut. Selain itu, dalam sebuah ADT yang lengkap, disertakan pula definisi invarian dari TYPE dan aksioma yang berlaku. ADT merupakan definisi **statik**. Definisi type dari sebuah ADT dapat mengandung sebuah definisi ADT lain. Misalnya:

- ADT Waktu yang terdiri dari ADT JAM dan ADT DATE
- GARIS yang terdiri dari dua buah POINT
- SEGI4 yang terdiri dari pasangan dua buah POINT (Top, Left) dan (Bottom,Right)

Type diterjemahkan menjadi type terdefinisi dalam bahasa yang bersangkutan, misalnya menjadi `record` dalam bahasa Ada/Pascal atau `struct` dalam bahasa C.

Primitif, dalam konteks prosedural, diterjemahkan menjadi fungsi atau prosedur. Primitif dikelompokkan menjadi :

- Konstruktor/Kreator, pembentuk **nilai type**. Semua objek (variabel) bertype tsb harus melalui konstruktor. Biasanya namanya diawali Make.
- Selektor, untuk mengakses komponen type (biasanya namanya diawali dengan Get)
- Prosedur pengubah nilai komponen (biasanya namanya diawali Get)
- Validator komponen type, yang dipakai untuk mentest apakah dapat membentuk type sesuai dengan batasan
- Destruktor/Dealokator, yaitu untuk "menghancurkan" nilai objek (sekaligus memori penyimpannya)
- Baca/Tulis, untuk interface dengan input/output device
- Operator relational, terhadap type tsb untuk mendefinisikan lebih besar, lebih kecil, sama dengan, dsb
- Aritmatika terhadap type tsb, karena biasanya aritmatika dalam bahasa pemrograman hanya terdefinisi untuk bilangan numerik
- Konversi dari type tersebut ke type dasar dan sebaliknya

ADT biasanya diimplementasi menjadi dua buah modul, yaitu:

- Definisi/Spesifikasi Type dan primitif .
  - Spesifikasi type sesuai dengan bahasa yang bersangkutan.
  - Spesifikasi dari primitif sesuai dengan kaidah dalam konteks prosedural, yaitu:
    - Fungsi : nama, domain, range dan prekondisi jika ada
    - Prosedur : Initial State, Final State dan Proses yang dilakukan
- Body/realisasi dari primitif, berupa kode program dalam bahasa yang bersangkutan. Realisasi fungsi dan prosedur harus sedapat mungkin memanfaatkan selektor dan konstruktor.

Supaya ADT dapat di-test secara tuntas, maka dalam kuliah ini setiap kali membuat sebuah ADT, harus disertai dengan sebuah program utama yang dibuat khusus untuk men-test ADT tsb, yang minimal mengandung pemakaian (*call*) terhadap setiap fungsi dan prosedur dengan mencakup semua kasus parameter. Program utama yang khusus dibuat untuk test ini disebut sebagai ***driver***. Urutan pemanggilan harus diatur

sehingga fungsi/prosedur yang memakai fungsi/prosedur lain harus sudah ditest dan direalisasi terlebih dulu.

Realisasi ADT dalam beberapa bahasa.

BAHASA	**SPESIFIKASI**	**BODY**
Pascal (hanya dalam turbo pascal)	Satu Unit interface	Implementation
C	File *header* dengan ekstensi `.h`	File kode dengan ekstensi `.c`
Ada	Paket dengan ekstensi `.ads`	Paket body dengan ekstensi `.adb`

Dalam modul ADT tidak terkandung definisi variabel. Modul ADT biasanya dimanfaatkan oleh modul lain, yang akan mendeklarasikan variabel bertype ADT tsb dalam modulnya. Jadi ADT bertindak sebagai ***Supplier*** (penyedia type dan primitif), sedangkan modul pengguna berperan sebagai ***Client*** (pengguna) dari ADT tsb. Biasanya ada sebuah pengguna yang khusus yang disebut sebagai ***main program*** (program utama) yang memanfaatkan langsung type tsb

Contoh dari ADT diberikan dalam bahasa Algoritmik di diktat ini, yaitu :

- ADT JAM. Contoh tsb sekaligus memberikan gambaran mengapa bahasa Algoritmik masih dibutuhkan, karena bersifat "umum" dan tidak tergantung pada pernik-pernik dalam bahasa pemrograman, sehingga dapat dipakai sebagai bahasa dalam membuat spesifikasi umum. Perhatikanlah catatan implementasi dalam bahasa C dan Bahasa Ada yang disertakan pada akhir definisi ADT
- ADT POINT. Contoh ini diberikan karena merupakan ADT yang sangat penting yang dipakai dalam bidang Informatika. Contoh ini bahkan dapat dikatakan "standard" dalam pemrograman, dan akan selalu dipakai sampai dengan pemrograman berorientasi Objek
- ADT GARIS. Contoh ini memberikan gambaran sebuah ADT yang memanfaatkan ADT yang pernah dibuat (yaitu ADT POINT), dan sebuah ADT yang mempunyai konstraint/invariant (persyaratan tertentu).
- ADT SEGIEMPAT dengan posisi sejajar sumbu X dan Y, didefinisikan sebagai dua buah POINT (TopLeft dan BottomRight)

Dari contoh-contoh tersebut, diharapkan mahasiswa dapat membangun ADT lain, minimal ADT yang diberikan definisinya dalam latihan.
Pada kehidupan nyata, pembuatan ADT selalu dimulai dari definisi "objek" yang berkomponen tersebut. Maka sangat penting, pada fase pertama pendefinisian ADT, diberikan definisi yang jelas. Lihat contoh pada latihan.
Semua contoh ADT pada bagian ini diberikan dalam bahasa Algoritmik, untuk diimplementasikan dalam bahasa C dan Ada, terutama karena contoh implementasi (sebagian) sudah diberikan dalam Diktat bahasa pemrograman yang bersangkutan.

Standard kuliah IF222 untuk implementasi ADT

1. Setiap ADT harus dibuat menjadi spesifikasi, body dan driver
2. Prosedur untuk melakukan implementasi kode program ADT (cara ini juga berlaku untuk bahasa Ada) :
   - Ketik header file secara lengkap, dan kompilasilah supaya bebas kesalahan
   - Copy header file menjadi body file
   - Lakukan editing secara "cepat", sehingga spesifikasi menjadi body:
     - Buang tanda ";" di akhir spesifikasi
     - Tambahkan body { } di setiap fungsi/prosedur
     - Khusus untuk fungsi, Beri return value sesuai type fungsi
   - Kodelah instruksi yang menjadi **body** fungsi/prosedur per kelompok (disarankan untuk melakukan "incremental implementation") sesuai dengan urutan pemanggilan.
   - Segera setelah sekelompok fungsi/prosedur selesai, tambahkan di driver dan lakukan test ulang.

   Setiap saat, banyaknya fungsi/prosedur yang sudah dikode bodynya harus seimbang.

Standard kuliah IF222 untuk implementasi ADT dalam bahasa C

- Dalam bahasa C, Modul spesifikasi. Realisasi dan driver harus disimpan dalam sebuah direktori sesuai dengan nama ADT. Jadi sourcode lengkap sebuah ADT dalam bahasa C adalah sebuah direktori dan isinya adalah minimal *header file, body file* dan *driver* . *Driver* harus disimpan dan dipakai mentest ulang jika ADT dibawa dari satu platform ke paltform lain
- Perhatikanlah **syarat realisasi** file header dalam bahasa C dan paket dalam bahasa Ada sesuai dengan yang ada pada Contoh program kecil Bahasa C dan Bahasa Ada.
- Bahasa C yang digunakan untuk mengkode harus bahasa C yang standard, misalnya ANSI-C

ADT JAM dalam bahasa Algoritmik

```
{ Definisi TYPE JAM <HH:MM:SS>     }
    TYPE Hour : integer [0..23]
    TYPE Minute : integer [0..59]
    TYPE Second : integer [0..59]

    TYPE JAM : < HH: Hour,  { Hour [0..23] }
                 MM: Minute,  { Minute [0..59] }
                 SS:Second  { Second [0..59] }
             >
{*************************************************************}
{ DEFINISI PRIMITIF                                           }
{*************************************************************}
{ KELOMPOK VALIDASI TERHADAP TYPE                              }
{*************************************************************}
function IsJValid (H,M,S:integer)→ boolean
{ Mengirim true  jika H,M,S dapat membentuk J yang valid      }

{Konstruktor: Membentuk sebuah JAM dari komponen-komponennya }
function MakeJam (HH:integer, MM:integer, SS:integer) → JAM
{ Membentuk sebuah JAM dari komponen-komponennya yang valid }
{ Pre cond : HH,MM,SS valid untuk membentuk JAM }

(** Operasi terhadap komponen : selekstor Get dan Set **)
{** Selektor **}
function Hour(J: JAM) → integer
{ Mengirimkan bagian HH (Hour) dari JAM }
function Minute(J: JAM) → integer
{ Mengirimkan bagian Menit (MM) dari JAM }
function Second (J: JAM) → integer
{ Mengirimkan bagian Second(SS) dari JAM }
{** Pengubah nilai Komponen **}
procedure  SetHH(Input/Output J: JAM, Input newHH : integer)
{Menngubah nilai komponen HH dari J}
procedure  SetMM(Input/Output J: JAM, Input newMM : integer)
{Menngubah nilai komponen MM dari J}
procedure  SetSS(Input/Output J: JAM, Input newSS : integer)
{Menngubah nilai komponen SS dari J}


{** Destruktor ***}
{* tidak perlu. Akan dijelaskan kemudian *}

{*************************************************************}
{ KELOMPOK BACA/TULIS                                          }
{*************************************************************}
procedure BacaJam (Input/Output J: JAM)
{ I.S. : J tidak terdefinisi  }
{ F.S. : J terdefinisi dan merupakan jam yang valid           }
{ Proses : mengulangi membaca komponen H,M,S sehingga membentuk J
yang valid                    }
{*************************************************************}
procedure TulisJam (Input J: JAM)
{ I.S. : J sembarang                                        }
{ F.S. :   Nilai J ditulis dg format HH:MM:SS               }
{ Proses : menulis nilai ke layar         }
{*************************************************************}
```

```
{ KELOMPOK KONVERSI TERHADAP TYPE                                    }
{*********************************************************************}
function JamToDetik (J: JAM) → integer
{ Diberikan sebuah JAM, mengkonversi menjadi Detik                   }
{ Rumus : detik = 3600*hour+menit*60 + detik                         }
{ nilai maksimum = 3600*23+59*60+59*60 }
{ hati-hati dengan representasi integer pada bahasa implementasi  }
{ kebanyakan sistem mengimplementasi integer,                     }
{bernilai maksimum kurang dari nilai maksimum hasil konversi }
{*********************************************************************}
function DetikToJam (N:integer) → JAM
{ Mengirim  konversi detik ke JAM                                    }
{ pada beberapa bahasa, representasi integer tidak cukup untuk }
{ menampung N }
{*****************************************************}
{ KELOMPOK OPERASI TERHADAP TYPE                      }
{*****************************************************}
{** Kelompok Operator Relational }
function JEQ(J1: JAM, J2: JAM) → boolean
{ Mengirimkan true jika J1=J2, false jika tidak }
function JNEQ(J1: JAM, J2: JAM) → boolean
{ Mengirimkan true jika J1 tidak sama dengan J2 }
function JLT(J1: JAM, J2: JAM) → boolean
{ Mengirimkan true jika J1<J2 , false jika tidak }
function JGT(J1: JAM, J2: JAM) → boolean
{ Mengirimkan true jika J1>J2, false jika tidak}
{******** Operator aritmatika JAM ****************************}
function JPlus(J1: JAM, J2: JAM) → JAM
{ Menghasilkan J1+J2, dalam bentuk JAM }
function JMinus(J1: JAM, J2: JAM) → JAM
{ Menghasilkan J1-J2, dalam bentuk JAM }
{ Precond : J1<=J2 }
function NextDetik (J: JAM) → JAM
{ Mengirim 1 detik setelah J dalam bentuk JAM          }
function NextNDetik (J: JAM, N: integer) → JAM
{ Mengirim N detik setelah J dalam bentuk JAM          }
function PrevDetik (J: JAM) → JAM
{ Mengirim 1 detik sebelum J dalam bentuk JAM          }
function PrevNDetik (J: JAM, N: integer) → JAM
{ Mengirim N detik sebelum J dalam bentuk JAM         }
function Durasi (Jaw:JAM , JAkh: JAM) → integer
{ Mengirim JAkh -JAw dlm Detik, dengan kalkulasi }
{ Hasilnya negatif jika Jaw > JAkhir }
{*************************************************}
```

Catatan implementasi:

- Dalam implementasi dengan bahasa C, di mana representasi integer ada bermacam-macam, maka anda harus berhati-hati misalnya:
  - dalam menentukan *range* dari fungsi Durasi, karena nilai detik dalam dua puluh empat jam melebihi representasi `int`. Fungsi durasi harus mempunyai range dalam bentuk `Long int` dan semua operasi dalam body fungsi harus di-casting.
  - Representasi Hour, Minute dan Ssecond terpaksa harus dalam bentuk `int` (atau `byte`) dan tidak mungkin dibatasi dengan angka yang persis. Itulah sebabnya fungsi validitas terhadap type diperlukan

- Fungsi selektor Get dan Set dapat digantikan dengan macro berparameter. Misalnya untuk Selektor terhadap Hour(P) , Minute(J) dan Second(J) dituliskan sebagai

```
#define Hour(J) (J).HH
#define Minute(J) (J).MM
#define Second(J) (J).SS
```

- Dalam implementasi dengan bahasa Ada, fungsi Valid dapat dihilangkan, karena dalam pembentukan sebuah JAM dapat memanfaatkan `sub type` untuk HH, MM dan SS serta memanfaatkan exception dalam menangani apakah 3 buah nilai H,M dan S dapat membentuk sebuah JAM yang valid.

ADT POINT dalam Bahasa Algoritmik

```
{Definisi ABSTRACT DATA TYPE POINT }
TYPE POINT  : < X: integer,  { absis  }
                Y: integer   { ordinat} >

{ DEFINISI PROTOTIP PRIMITIF      }
{** Konstruktor membentuk POINT **}
function MakePOINT (X:integer; Y:integer)→ POINT
{ Membentuk sebuah POINT dari komponen-komponennya }

(** Operasi terhadap komponen : selekstor Get dan Set **)
{** Selektor POINT **}
function  GetAbsis(P:POINT) → integer
{Mengirimkan komponen Absis dari P}
function  GetOrdinat (P:POINT) → integer
{ Mengirimkan komponen Ordinat dari P POINT}
(** Set nilai komponen **)
procedure  SetAbsis(Input/Output P:POINT, Input newX : integer)
{Menngubah nilai komponen Absis dari P}
procedure  SetOrdinat (Input/Output P:POINT, Input newY : integer)
{ Mengubah nilai komponen Ordinat dari P }


{** Destruktor/Dealokator: tidak perlu **}

{*** KELOMPOK Interaksi dengan I/O device, BACA/TULIS  ***}
procedure BacaPOINT (Output P: POINT)
{ Makepoint(x,y,P) membentuk P dari x dan y yang dibaca  }
procedure TulisPOINT(Input P:POINT)
{ Nilai P ditulis ke layar dg format "(X,Y}"  }

{ KELOMPOK OPERASI ARITMATIKA TERHADAP TYPE  }
function "+" (P1, P2: POINT) →  POINT
{ Menghasilkan POINT yang bernilai P1+P2 }
{ Melakukan operasi penjumlahan vektor}
function "-" (P1, P2: POINT) →  POINT
{ Menghasilkan POINT bernilai P1-P2 }
function "." (P1,P2: POINT) → POINT
{ Operasi perkalian P1.P2, Melakukan operasi dot product}
function x (P1,P2: POINT) → POINT
{ Operasi perkalian P1xP2, Melakukan operasi cross product}

{** Kelompok operasi relasional terhadap POINT }
function EQ(P1,P2: POINT) → boolean
{Mengirimkan true jika P1 = P2 }
function NEQ(P1,P2: POINT) → boolean
{Mengirimkan true jika P1 tidak sama dengan P2 }
function "<"(P1,P2 : POINT) → boolean
{Mengirimkan true jika P1 < P2. Definisi lebih kecil: lebih "kiri-
bawah" dalam bidang kartesian }
function ">"(P1,P2: POINT) → boolean
{Mengirimkan true jika P1 > P2. Definisi lebih besar: lebih "kanan-
atas" dalam bidang kartesian }

{ ** Kelompok menentukan di mana P berada}
```

```
function IsOrigin (P:POINT) → boolean
{ Menghasilkan true jika P adalah titik origin }
function IsOnSbX (P:POINT)→ boolean
{ Menghasilkan true jika P terletak Pada sumbu X}
function IsOnSbY (P:POINT)→ boolean
{ Menghasilkan true jika P terletak pada sumbu Y}
function Kuadran(P:POINT) → integer
{ Menghasilkan kuadran dari P: 1,2,3, atau 4}
{ Precondition : P bukan Titik Origin, }
{             dan P tidak terletak di salah satu sumbu}

{ ** KELOMPOK OPERASI LAIN TERHADAP TYPE           }
function NextX (P: POINT) → POINT
{ Mengirim salinan P dengan absis ditambah satu    }
function NextY (P: POINT) → POINT
{ Mengirim salinan P dengan ordinat ditambah satu}
function PlusDelta ( deltaX,DeltaY: integer) → POINT
{ Mengirim salinan P yang absisnya = Absis(P)+DeltaX dan  }
{ Ordinatnya adalah Ordinat(P)+ DeltaY}
function MirrorOf (P: POINT, SbX,SbY:boolean) → POINT
{ Menghasilkan salinan P yang dicerminkan}
{ tergantung nilai SbX dan SBY}
{ Jika SbX bernilai true, maka dicerminkan thd Sumbu X}
{ Jika SbY bernilai true, maka dicerminkan thd Sumbu Y}
function Jarak0 (P: POINT) → real
{ Menghitung jarak P ke (0,0) }
function Panjang (P1,P2:POINT) → real
{ Menghitung panjang garis yang dibentuk P1 dan P2}
{ Perhatikanlah bahwa di sini spec fungsi "kurang" baik}
{ sebab menyangkut ADT Garis!!! }
procedure Geser (Input/Output P: POINT, Input Deltax,deltaY: integer)
{I.S. P terdefinisi}
{F.S. P digeser sebesar DeltaX dan ordinatnya sebesar Delta Y}
procedure GeserKeSbX (Input/Output P: POINT)
{I.S. P terdefinisi}
{F.S. P di Sumbu X dg absis sama dg absis semula.
{ Proses :tergeser Ke Sumbu X. }
{ Contoh: Jika koordinat semula(9,9) menjadi (9,0) }
procedure GeserKeSbY(Input/Output P: POINT)
{I.S. P terdefinisi}
{F.S. P di Sumbu Y dengan absis yang sama dengan semula
{ P digeser Ke Sumbu Y. }
{ Contoh: Jika koordinat semula(9,9) menjadi (0,9) }
procedure Mirror (Input/Output P: POINT, Input SbX,SbY:boolean)
{I.S. P terdefinisi}
{F.S.  P dicerminkan tergantung nilai SbX atau SBY}
{ Jika SbX true maka dicerminkan terhadap Sumbu X }
{ Jika SbY true maka dicerminkan terhadap Sumbu Y}
procedure Putar (Input/Output P: POINT, Input Sudut: Real);
{I.S. P terdefinisi}
{F.S.  P diputar sebesar (Sudut) derajat }
```

Catatan implementasi:

- Dalam implementasi dengan bahasa C, nama fungsi seperti "<" tidak dimungkinkan. Dalam hal ini, harus disesuaikan

ADT GARIS Dalam Bahasa Algoritmik

```
{ Contoh ADT yang memanfaatkan ADT Lain}
{ Definisi : GARIS dibentuk oleh dua buah POINT }

{ *** ADT LAIN YANG DIPAKAI*****}
   USE POINT
{ ********************** Definisi TYPE **********************}
TYPE  GARIS : <  PAw : POINT,  { Titik Awal }
                 PAkh : POINT { Titik Akhir}  >

{ ********************** Definisi METHOD **********************}
{ DEFINISI PRIMITIF                  }
{** Konstruktor membentuk GARIS ***}
Procedure MakeGARIS (Input P1,P2:POINT, Output L:GARIS)
{I.S. P1 dan P2 terdefinisi }
{F.S. L terdefinisi dengan L.PAw= P1 dan L.Pakh=P2 }
{ Membentuk sebuah L dari komponen-komponennya }

{** Selektor GARIS **}
function  GetPAw → POINT
{Mengirimkan komponen Titik pertama  dari L  GARIS}
function  GetPAkh → POINT
{Mengirimkan komponen Titik kedua  dari L  GARIS}
(** Set nilai komponen **)
procedure  SetPAw(Input/Output G:GARIS, Input newPAw : POINT)
{Menngubah nilai komponen PAw dari G}
procedure  SetPAkh (Input/Output G:GARIS, Input newPAkh : POINT)
{ Mengubah nilai komponen PAkh dari G }

{ KELOMPOK Interaksi dengan I/O device, BACA/TULIS **}
procedure BacaGARIS (Output L: GARIS)
{I.S. sembarang}
{F.S. mengulangi membaca dua buah nilai P1 dan P2 sehingga }
{ dapat membentuk GARIS yang valid }
{ MakeGARIS(P1,P2) dari P1 dan P2 yang dibaca     }
procedure TulisGARIS ( Input L: GARIS)
{ Nilai L  ditulis ke layar dengan format ((x,y) , (x,y) )}

{** Kelompok operasi relasional terhadap GARIS**}
function EQ(L1,L2: GARIS) → boolean;
{Mengirimkan true jika L1  = L2 }
{ L1  dikatakan sama dengan L2 }
{ jika Titik Awal dari L =Titik awal dari L dan }
{ Titik akhir L1  = Titik akhir dari L2}
function NEQ(L1,L2: GARIS) → boolean; }
{Mengirimkan true jika L  tidak sama dengan L }
{ Negasi dari fungsi EQ}

{ ** Kelompok menentukan di mana L  berada}
function IsOnSbX (L:GARIS) → boolean;
{ Menghasilkan true jika L  terletak Pada sumbu X}
function IsOnSbY return boolean;
{ Menghasilkan true jika L  terletak pada sumbu Y}
function Kuadran(L:GARIS)  → integer;
{ Menghasilkan kuadran dari L  (dimana PAw dan PAkh berada) }
{ Precondition : L  tidak terletak di salah satu sumbu}
```

```
{ Kelompok predikat lain}
function IsTegakLurus (L, L1: GARIS) → boolean
{ Menghasilkan true jika L  tegak lurus terhadap L1}
function IsSejajar (L, L1: GARIS) → boolean
{ Menghasilkan true jika L  "sejajar" terhadap L1}

{ *** Kelompok operasi lain ****}
function HslGeser (L: GARIS, DX,DY:integer) → GARIS
{ Menghasilkan salinan L  yang titik awal dan akhirnya }
{ digeser sejauh DX dan DY}
function MirrorOf (L: GARIS, SbX,SbY:boolean) → GARIS;
{ Menghasilkan salinan L  yang dicerminkan}
{ tergantung nilai SbX dan SBY}
{ Jika SbX bernilai true, maka dicerminkan thd Sumbu X}
{ Jika SbY bernilai true, maka dicerminkan thd Sumbu Y}

function Panjang (L:GARIS) → real
{ Menghitung panjang garis L}
function Arah (L:GARIS) → real
{ Menghitung arah dari garis L }
{ yaitu sudut yang dibentuk dengan Sumbu X+}
function SudutGaris (L, L1:GARIS) → real
{ Menghasilkan sudut perpotongan antara L  dengan L1}
{ Precondition : L  tidak sejajar dengan L1 dan}
{                L  tidak berimpit dengan L1}

procedure Geser ( Input/Output L:GARIS,Input Deltax,deltaY: integer)
{I.S. L terdefinisi }
{F.S. L  digeser sebesar DeltaX dan ordinatnya sebesar Delta Y}
{ PAw dan PAkh digeser! }
procedure Mirror (Input/Output L:GARIS Input SbX,SbY:boolean);
{I.S. L terdefinisi }
{F.S. L  dicerminkan tergantung nilai SbX atau SBY}
{ Jika SbX true maka dicerminkan terhadap Sumbu X }
{ Jika SbY true maka dicerminkan terhadap Sumbu Y}
{ L  dicerminkan tergantung nilai SbX atau SBY}
procedure Putar (Input/Output L:GARIS, Input Sudut: real)
{I.S. L terdefinisi }
{F.S. L  diputar sebesar (Sudut) derajat : PAw dan PAkh diputar}

{ ******* CONSTRAINT/INVARIANT DARI ADT  **********************}
{   NEQ(Pakh,PAw) {dengan NEQ adalah fungsi relational utk POINT}
```

Catatan implementasi:

- Dalam implementasi dengan bahasa C, "use" ADT lain menjadi "include" file header dari ADT yang dipakai
- Dalam implementasi dengan bahasa Ada, maka "use" menjadi kata "`with`" dalam konteks (paket yang dipakai), yang dapat disertai dengan `use` untuk menghilangkan pemanggilan dengan prefiks
- Constraint/invariant, tidak dapat dituliskan tetapi menjadi sebuah fungsi yang akan memeriksa validitas Paw dan Pakh, sehingga dapat membentuk sebuah GARIS (lihat contoh JAM)
- Nama fungsi yang sama untuk ADT POINT dan ADT GARIS (misalnya fungsi relasional EQ dan NEQ) akan menimbulkan dalam bahasa C, sehingga harus diganti. Nama yang sama ini tidak menimbulkan masalah dalam bahasa Ada, karena pemanggilan disertai dengan prefix nama paket, atau tanpa kata "use" dalam konteks, dengan fasilitas parameter overloading konflik nama ini terselesaikan

LATIHAN :

1. Buatlah implementasi semua ADT tsb dalam bahasa C dan bahasa Ada (setelah anda mengenal bahasa Ada), dengan memperhatikan catatan implementasi tsb.
2. Buatlah ADT JAMT dengan Representasi JAM sbb :

```
{ Definisi jam dengan am dan pm }
{*  Jam : 0:00:00 pm  s/d 11:59:59 am **}
{ Definisi TYPE JAMT <HH:MM:SS AMPM>     }
 TYPE ampm : enumerasi [am, pm]
 Type Hour : integer [0..11]
 Type Minute : integer [0..59]
 Type Second : integer [0..59]
 TYPE JAMT : < HH: Hour ,
               MM: Minute ,
               SS: Second ,}
               T : ampm>
```

   a) Realisasikanlah, seolah-olah anda memulai modul JAM dengan type ini.
   b) Realisasikanlah, dengan asumsi bahwa anda sudah mempunyai ADT JAM dengan representasi sebelumnya, maka semua operasi aritmatika dan yang lain akan dilakukan dalam type JAM, sehingga anda hanya membuat sebuah "konvertor type" dari JAMX menjadi JAM. Konvertor sebaiknya diletakkan di dalam modul JAM atau JAMX ?
3. Buatlah ADT bernama DATE, yang terdiri dari Tanggal, Bulan dan Tahun kalender.
4. Buatlah ADT bernama WAKTU, yang terdiri dari JAM dan DATE yang pernah didefinisikan sebelumnya.
5. Buatlah ADT PECAHAN untuk merepresentasi bilangan pecahan yang terdiri dari numerator dan denumerator sbb :

```
{ Definisi TYPE Pecahan     }
TYPE Pecahan
      <  b: integer  { bagian bulat }
         d: integer  { numerator, pembagi, n< d }
         n: integer  { denumerator, penyebut }
         s: integer  { [-1,1], tanda pecahan }
       >
```

   Jika pecahan bernilai nol, maka representasinya adalah : d=0, n=0 dan s=1.
   Jika b>0 maka artinya ada bagian bulat..

6. Buatlah ADT Segi4 yang mewakili Segi-empat, dengan definisi segi empat adalah:
   a) sebuah segi empat yang selalu "sejajar" terhadap sumbu X dan Y, dan terdefinisi oleh dua buah titik yaitu titik Atas-terkiri dan titik Kanan-terbawah (Top-Left dan Bottom-Right)
   b) sebuah segi empat terdefinisi oleh sebuah Titik, arah terhadap sumbu X+ dan juga panjang sisi-sisinya
7. Buatlah sebuah ADT bernama BUJURSANGKAR, yang sebenarnya adalah sebuah segi empat yang spesifik, yaitu yang sisi-sisinya sama panjang. Ada dua solusi: anda membuat type baru, atau memanfaatkan type Segi4 dan menambahkankan konstraint/invariant. Inilah gunanya mendefinisikan invarian pada sebuah ADT.