MODUL PRA-PRAKTIKUM
IF2110/ALGORITMA DAN STRUKTUR DATA

NAMA :
NIM :
KELAS :
SEM/TAHUN :

DAFTAR ISI

CHECKLIST IMPLEMENTASI

Isilah kolom Status dengan:

- Angka 0 jika belum dikerjakan.
- Angka 1 jika sudah dikerjakan, namun belum selesai atau masih salah.
- Angka 2 jika sudah dikerjakan dan sudah benar.

No.	Nama ADT	Nama File	Status	Diisi oleh Asisten	
				Tgl. Periksa	Paraf
1.	Jam				
2.	Point				
3.	Garis				
4.	Tanggal				
5.	Tabel Kontigu – Representasi Eksplisit Statik				
6.	Tabel Kontigu – Representasi Eksplisit Dinamik				
7.	Tabel Kontigu – Representasi Implisit Statik				
8.	Tabel Kontigu – Representasi Implisit Dinamik				
9.	Matriks				
10.	Mesin Kata – Model Akuisisi Versi 1				
11.	Mesin Kata – Model Akuisisi Versi 2				
12.	Stack – Representasi Tabel Kontigu – Alokasi Statik				
13.	Stack – Representasi Tabel Kontigu – Alokasi Dinamik				
14.	Queue – Rep. Tabel Kontigu – Alternatif I				
15.	Queue – Rep. Tabel Kontigu – Alternatif II				
16.	Queue – Rep. Tabel Kontigu – Alternatif III				

No.	Nama ADT	Nama File	Status	Diisi oleh Asisten	
				Tgl. Periksa	Paraf
17.	List Berkait Linier – Rep. Fisik dengan Pointer – First Eksplisit				
18.	List Berkait Linier – Rep. Fisik dengan Pointer – First Implisit				
19.	List Berkait Linier – Rep. Fisik dengan Tabel Berkait				
20.	List Berkait Linier – Rep. Fisik dengan Tabel Kontigu				
21.	List First-Last dengan Dummy pada Last				
22.	List Sirkuler				
23.	List dengan Double Pointer				
24.	Stack – Rep. dengan List Linier				
25.	Queue – Rep. dengan List Linier				
26.	Priority Queue				
27.	List Rekursif				
28.	Pohon Biner				
29.	SK-1 : Polinom – Rep. dengan Tabel Kontigu				
30.	SK-1 : Polinom – Rep. Berkait dengan Pointer				
31.	SK-2 : Kemunculan Huruf dan Posisi pada Pita Karakter – Rep. dengan Matriks				
32.	SK-2 : Kemunculan Huruf dan Posisi pada Pita Karakter – Rep. dengan List Kontigu dan List Berkait				
33.	SK-3 : Pengelolaan Memori – Rep. Memori dengan Tabel Kontigu				
34.	SK-3 : Pengelolaan Memori – Rep. Zone Kosong dengan List Berkait				
35.	SK-4 : MultiList – Alternatif I				

No.	Nama ADT	Nama File	Status	Diisi oleh Asisten	
				Tgl. Periksa	Paraf
36.	SK-4 : MultiList – Alternatif II				
37.					
38.					
39.					
40.					

P-01. ADT JAM

1. Buatlah ADT JAM sesuai dengan definisi dan spesifikasi di bawah ini dalam Bahasa C dengan pembagian modul sebagaimana dijelaskan dalam kuliah. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**). Gunakan *macro* untuk mendefinisikan selektor (baik selektor *get* maupun *set*).
2. Buatlah juga driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

```
{ *** Definisi TYPE JAM <HH:MM:SS> *** }
type JAM : < HH : integer,  { integer [0..23] }
             MM : integer,  { integer [0..59] }
             SS : integer   { integer [0..59] } >

{ ************************************************************* }
{ DEFINISI PRIMITIF                                             }
{ ************************************************************* }
{ KELOMPOK VALIDASI TERHADAP TYPE                               }
{ ************************************************************* }
function IsJAMValid (H, M, S : integer) → boolean
{ Mengirim true  jika H,M,S dapat membentuk J yang valid }
{ dipakai untuk mentest SEBELUM membentuk sebuah Jam }

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

{ *** Operasi terhadap komponen : selektor Get dan Set *** }
{ *** Selektor *** }
function GetHour (J : JAM) → integer
{ Mengirimkan bagian jam (HH) dari JAM }
function GetMinute (J : JAM) → integer
{ Mengirimkan bagian menit (MM) dari JAM }
function GetSecond (J : JAM) → integer
{ Mengirimkan bagian detik (SS) dari JAM }
{ *** Pengubah nilai Komponen *** }
procedure  SetHH (input/output J : JAM, input newHH : integer)
{ Mengubah nilai komponen jam (HH) dari J dengan newHH }
procedure  SetMM (input/output J : JAM, input newMM : integer)
{ Mengubah nilai komponen menit (MM) dari J dengan newMM }
procedure  SetSS (input/output J : JAM, input newSS : integer)
{ Mengubah nilai komponen detik (SS) dari J dengan newSS }

{ ************************************************************* }
{ KELOMPOK BACA/TULIS                                           }
{ ************************************************************* }
procedure BacaJam (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. Tidak mungkin menghasilkan J yang tidak valid. }
procedure TulisJam (input J : JAM)
{ I.S. : J sembarang }
{ F.S. :   Nilai J ditulis dg format HH:MM:SS }
{ Proses : menulis nilai setiap komponen J ke layar }

{ ************************************************************* }
{ KELOMPOK KONVERSI TERHADAP TYPE                               }
{ ************************************************************* }
function JamToDetik (J : JAM) → integer
{ Diberikan sebuah JAM, mengkonversi menjadi jumlah Detik dari pukul 0:0:0 }
{ Rumus : detik = 3600*HH + 60*MM + SS  }
{ nilai maksimum = 3600*23+59*60+59 }
{ 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 }
{ Catatan: Jika N ≥ 86400, maka harus dikonversi dulu menjadi jumlah detik yang
  mewakili jumlah detik yang mungkin dalam 1 hari, yaitu dengan rumus: N1 = N mod
  86400, baru N1 dikonversi menjadi JAM }

{ ************************************************************* }
{ 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 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 }
{ *** Kelompok Operator Aritmetika *** }
function Durasi (Jaw : JAM , Jakh : JAM) → integer
{ Mengirim JAkh-JAw dlm Detik, dengan kalkulasi }
{ Jika JAw > JAkhir, maka JAkhir adalah 1 hari setelah JAw }
```

P-02. ADT POINT

1. Buatlah ADT POINT sesuai dengan definisi dan spesifikasi di bawah ini dalam Bahasa C dengan pembagian modul sebagaimana dijelaskan dalam kuliah. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**). Gunakan *macro* untuk mendefinisikan selektor (baik selektor *get* maupun *set*).
   Dalam implementasi dengan bahasa C, nama fungsi untuk operator aritmatika dan operator relasional (seperti "<") tidak dimungkinkan. Buatlah nama yang sesuai untuk operator-operator tersebut.
   Beberapa primitf di bawah belum mempunyai spesifikasi yang jelas. Untuk itu, sebelum implementasi, buatlah dulu spesifikasi yang lebih jelas.
2. Buatlah juga driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

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

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

{ *** Operasi terhadap komponen : selektor Get dan Set *** }
{ *** Selektor POINT *** }
function  GetAbsis (P : POINT) → real
{ Mengirimkan komponen absis dari P }
function  GetOrdinat (P:POINT) → real
{ Mengirimkan komponen ordinat dari P POINT }
{ *** Set nilai komponen *** }
procedure SetAbsis (input/output P : POINT, input newX : real)
{ Mengubah nilai komponen absis dari P }
procedure SetOrdinat (input/output P:POINT, input newY : real)
{ Mengubah nilai komponen ordinat dari P }

{ *** KELOMPOK Interaksi dengan I/O device, BACA/TULIS  *** }
procedure BacaPOINT (output P : POINT)
{ Membaca nilai absis dan ordinat dari keyboard dan membentuk POINT P berdasarkan
dari nilai absis dan ordinat tersebut }
{ I.S. Sembarang }
{ F.S. P terdefinisi }
procedure TulisPOINT (input P : POINT)
{ Nilai P ditulis ke layar dengan format "(X,Y)" }
{ I.S. P terdefinisi }
{ F.S. P tertulis di layer dengan format "(X,Y)" }

{ *** Kelompok operasi relasional terhadap POINT *** }
function EQ (P1, P2 : POINT) → boolean
{ Mengirimkan true jika P1 = P2 : absis dan ordinatnya sama }
function NEQ (P1, P2 : POINT) → boolean
{ Mengirimkan true jika P1 tidak sama dengan P2 }

{ *** 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 }
{ Prekondisi : 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 (P : POINT; deltaX, deltaY : real) → POINT
{ Mengirim salinan P yang absisnya adalah Absis(P) + deltaX dan ordinatnya adalah
Ordinat(P) + deltaY }
function MirrorOf (P : POINT; SbX : boolean) → POINT
{ Menghasilkan salinan P yang dicerminkan terhadap salah satu sumbu }
{ Jika SbX bernilai true, maka dicerminkan terhadap sumbu X }
{ Jika SbX bernilai false, maka dicerminkan terhadap 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.
Tuliskan spec fungsi yang lebih tepat. }
procedure Geser (input/output P : POINT, input deltaX, deltaY : real)
{ I.S. P terdefinisi }
{ F.S. P digeser, absisnya sebesar deltaX dan ordinatnya sebesar deltaY }
procedure GeserKeSbX (input/output P : POINT)
{ I.S. P terdefinisi }
{ F.S. P berada pada sumbu X dengan absis sama dengan absis semula. }
{ Proses : P digeser ke sumbu X. }
{ Contoh : Jika koordinat semula (9,9), maka menjadi (9,0) }
procedure GeserKeSbY (input/output P : POINT)
{ I.S. P terdefinisi}
{ F.S. P berada pada sumbu Y dengan ordinat yang sama dengan semula. }
{ Proses : P digeser ke sumbu Y. }
{ Contoh : Jika koordinat semula (9,9), maka menjadi (0,9) }
procedure Mirror (input/output P : POINT, input SbX : boolean)
{ I.S. P terdefinisi }
{ F.S. P dicerminkan tergantung nilai SbX atau SbY }
{ Jika SbX true maka dicerminkan terhadap sumbu X }
{ Jika SbX false maka dicerminkan terhadap sumbu Y }
procedure Putar (input/output P : POINT, input Sudut : real)
{ I.S. P terdefinisi }
{ F.S. P digeser sebesar Sudut derajat dengan sumbu titik (0,0) }
```

P-03. ADT GARIS

ADT GARIS adalah contoh ADT yang memanfaatkan ADT lain, dalam hal ini ADT POINT. Sebuah GARIS dibentuk oleh dua buah POINT yang menyatakan titik awal dan titik akhir GARIS.

1. Buatlah ADT GARIS sesuai dengan definisi dan spesifikasi di bawah ini dalam Bahasa C dengan pembagian modul sebagaimana dijelaskan dalam kuliah. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**). Gunakan ADT POINT dari modul P-02. Gunakan *macro* untuk mendefinisikan selektor (baik selektor *get* maupun *set*).
2. Buatlah juga driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

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

{ ************************************************************* }
{ 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 (G : GARIS) → POINT
{ Mengirimkan komponen Titik Awal dari L GARIS }
function  GetPAkh (G : GARIS) → POINT
{ Mengirimkan komponen Titik Akhir dari L GARIS }
{ *** Set nilai komponen *** }
procedure  SetPAw (input/output G : GARIS, input newPAw : POINT)
{ Mengubah 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. L terdefinisi sebagai GARIS yang valid }
{ Proses : mengulangi membaca dua buah nilai P1 dan P2 sehingga dapat membentuk
GARIS yang valid. GARIS disebut valid jika titik awal tidak sama dengan titik
akhirnya. }
procedure TulisGARIS (input L : GARIS)
{ I.S. L terdefinisi }
{ F.S. L tertulis di layar dengan format ((x,y), (x,y)) }

{ ************************************************************* }
{ KELOMPOK OPERASI TERHADAP TYPE                                }
{ ************************************************************* }
{ *** 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 L1 sama dengan titik awal dari
L2 dan titik akhir L1 sama 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 (L : GARIS) → boolean
{ Menghasilkan true jika L terletak pada sumbu Y }
function Kuadran (L : GARIS)  → integer
{ Menghasilkan kuadran dari L (di mana PAw dan PAkh berada) }
{ Prekondisi : L  tidak terletak di salah satu sumbu, dan }
{ PAw serta PAkh selalu terletak pada kuadran yang sama }

{ *** 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 : real) → GARIS
{ Menghasilkan salinan L yang titik awal dan titik akhirnya digeser sejauh DX dan
DY }
function MirrorOf (L : GARIS; SbX : boolean) → GARIS
{ Menghasilkan salinan L  yang tergantung nilai SbX dan SbY }
{ Jika SbX bernilai true, maka dicerminkan terhadap sumbu X }
{ Jika SbX bernilai false, maka dicerminkan terhadap sumbu Y }
function Panjang (L : GARIS) → real
{ Menghitung panjang garis L : berikan rumusnya }
function Arah (L : GARIS) → real
{ Menghitung arah dari garis L yaitu sudut yang dibentuk dengan sumbu X positif }
function SudutGaris (L, L1 : GARIS) → real
{ Menghasilkan sudut perpotongan antara L  dengan L1 }
{ Prekondisi : L tidak sejajar dengan L1 dan }
{              L tidak berimpit dengan L1 }
procedure Geser (input/output L : GARIS, input deltaX, deltaY : real)
{ I.S. L terdefinisi }
{ F.S. L digeser sebesar deltaX dan ordinatnya sebesar deltaY }
{ Proses : 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 }
procedure Putar (input/output L : GARIS, input Sudut : real)
{ I.S. L terdefinisi }
{ F.S. L diputar sebesar Sudut derajat : PAw tetap dan PAkh berubah }
```

P-04. ADT TANGGAL

1. Buatlah ADT TANGGAL sesuai dengan definisi dan spesifikasi di bawah ini dalam Bahasa C dengan pembagian modul sebagaimana dijelaskan dalam kuliah. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**). Gunakan *macro* untuk mendefinisikan selektor (baik selektor *get* maupun *set*).
2. Buatlah juga driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

```
{ *** DEFINISI  DAN SPESIFIKASI TYPE TANGGAL *** }
type TANGGAL : < DD : integer [1..31],  { hari  }
                 MM : integer [1..12],  { bulan }
                 YY : integer > 0       { tahun } >

{ *** DEFINISI DAN SPESIFIKASI SELEKTOR *** }
{ *** SELEKTOR GET *** }
function Day (T : TANGGAL) → integer [1..31]
{ Memberikan hari DD dari T yang terdiri dari <DD, MM, YY> }
function Month (T : TANGGAL) → integer [1..12]
{ Memberikan bulan MM dari T yang terdiri dari <DD, MM, YY> }
function Year (T : TANGGAL) → integer > 0
{ Memberikan tahun YY dari T yang terdiri dari <DD, MM, YY> }
{ *** SELEKTOR SET *** }
procedure SetDay (input/output T : TANGGAL, input newDay : integer [1..31])
{ Mengubah nilai komponen DD dari T }
procedure SetMonth (input/output T : TANGGAL, input newMonth : integer [1..12])
{ Mengubah nilai komponen MM dari T }
procedure SetYear (input/output T : TANGGAL, input newYear : integer > 0)
{ Mengubah nilai komponen YY dari T }

{ *** VALIDASI TERHADAP TYPE  *** }
function IsTanggalValid (D, M, Y : integer) → boolean
{ Mengirim true  jika D, M, Y dapat membentuk TANGGAL yang valid }
{ dipakai untuk mengetes SEBELUM membentuk sebuah TANGGAL }
{ Perhatikan bahwa setiap bulan memiliki jumlah hari yang berbeda, dan
  khusus bulan Februari bisa 28 atau 29, tergantung apakah tahunnya kabisat atau
  bukan }

{ *** DEFINISI DAN SPESIFIKASI KONSTRUKTOR *** }
function MakeTANGGAL (h : integer, b : integer, t : integer) → TANGGAL
{ Membentuk  TANGGAL dari tanggal h, bulan b, dan tahun t. }
{ Prekondisi : h, b, t dapat membentuk TANGGAL yang valid. }

{ *** DEFINISI DAN SPESIFIKASI OPERATOR BACA/TULIS *** }
procedure BacaTANGGAL (output T : TANGGAL)
{ I.S. Sembarang }
{ F.S. T terdefinisi dan merupakan TANGGAL yang valid }
{ Proses : Mengulangi membaca komponen DD, MM, YY sehingga membentuk T yang valid.
           Tidak mungkin menghasilkan T yang tidak valid. }
procedure TulisTANGGAL (input T : TANGGAL)
{ I.S. T terdefinisi }
{ F.S. Nilai T ditulis ke layar dengan format DD/MM/YY }
{ Proses : Menulis nilai setiap komponen T ke layar }

{ *** DEFINISI DAN SPESIFIKASI OPERATOR/FUNGSI LAIN TERHADAP TANGGAL *** }
function NextDay (D : TANGGAL) → TANGGAL
{ Menghitung TANGGAL yang merupakan keesokan hari dari TANGGAL D yang diberikan}
function PrevDay (D : TANGGAL) → TANGGAL
{ Menghitung TANGGAL yang merupakan 1 hari sebelum TANGGAL D yang diberikan }
function NextNday (D : TANGGAL, N : integer) → TANGGAL
{ Menghasilkan TANGGAL yang merupakan N hari sesudah TANGGAL D yang diberikan }
function PrevNDay (D : TANGGAL, N : integer) → TANGGAL
{ Menghasilkan TANGGAL N hari sebelum T }
```

```
function HariKe (D : TANGGAL) → integer [0..366]
{ Menghitung jumlah hari terhadap 1 Januari pada tahun yang bersangkutan, dengan
memperhitungkan apakah tahun tersebut adalah tahun kabisat }

{ *** DEFINISI DAN SPESIFIKASI PREDIKAT *** }
function IsEqual? (D1, D2 : TANGGAL) → boolean
{ Menghasilkan true jika D1=D2 yaitu jika Day(D1)=Day(D2) dan Month(D1)=Month(D2)
dan Year(D1)=Year(D2). }
function IsBefore? (D1, D2 : TANGGAL) → boolean
{ Menghasilkan true jika D1 adalah tanggal sebelum D2 }
function IsAfter? (D1, D2 : TANGGAL) → boolean
{ Menghasilkan true jika D1 adalah tanggal sesudah D2 }
function IsKabisat? (Y : integer > 0) → boolean
{ Menghasilkan true jika tahun Y adalah tahun kabisat yaitu habis dibagi 4 tetapi
tidak habis dibagi 100, atau habis dibagi 400 }
```

P-05. ADT TABEL KONTIGU

Bagian 1. Representasi Eksplisit – Statik

1. Buatlah ADT Tabel Kontigu dengan representasi **eksplisit** dan alokasi memori **statik** sesuai dengan definisi dan spesifikasi di bawah ini dalam Bahasa C dengan pembagian modul sebagaimana dijelaskan dalam kuliah. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
2. Buatlah juga driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

```
{ MODUL TABEL INTEGER }
{ Berisi definisi dan semua primitif pemrosesan tabel integer }
{ Penempatan elemen selalu rapat kiri }
{ Versi I : dengan banyaknya elemen didefinisikan secara eksplisit, memori tabel
            statik }

{  Kamus Umum }

constant IdxMax : integer = 100
constant IdxMin : integer = 1
constant IdxUndef : integer = -999 { indeks tak terdefinisi}

{ Definisi elemen dan koleksi objek }
type IdxType : integer  { type indeks }
type ElType  : integer  { type elemen tabel }
type TabInt  : < TI : array [IdxMin..IdxMax] of ElType,
                 { memori tempat penyimpan elemen (container) }
                 Neff : integer [IdxMin..IdxMax] { banyaknya elemen efektif }
               >
{ Indeks yang digunakan [IdxMin..IdxMax] }
{ Jika T adalah TabInt, cara deklarasi dan akses: }
{ Deklarasi : T : TabInt }
{ Maka cara akses:
  T.Neff  untuk mengetahui banyaknya elemen
  T.TI    untuk mengakses seluruh nilai elemen tabel
  T.TI[i] untuk mengakses elemen ke-i }
{ Definisi :
  Tabel kosong: T.Neff = 0
  Definisi elemen pertama : T.TI[i] dengan i=1
  Definisi elemen terakhir yang terdefinisi: T.TI[i] dengan i=T.Neff }

{ ********** KONSTRUKTOR ********** }
{ Konstruktor : create tabel kosong  }
procedure MakeEmpty (output T : TabInt)
{ I.S. sembarang }
{ F.S. Terbentuk tabel T kosong dengan kapasitas IdxMax-IdxMin+1 }

{ ********** SELEKTOR ********** }
{ *** Banyaknya elemen *** }
function NbElmt (T : TabInt) → integer
{ Mengirimkan banyaknya elemen efektif tabel }
{ Mengirimkan nol jika tabel kosong }
{ *** Daya tampung container *** }
function MaxNbEl (T : TabInt) → integer
{ Mengirimkan maksimum elemen yang dapat ditampung oleh tabel }
{ *** Selektor INDEKS *** }
function GetFirstIdx (T : TabInt) → IdxType
{ Prekondisi : Tabel T tidak kosong }
{ Mengirimkan indeks elemen pertama }
function GetLastIdx (T : TabInt) → IdxType
{ Prekondisi : Tabel T tidak kosong }
{ Mengirimkan indeks elemen terakhir }
{ *** Menghasilkan sebuah elemen *** }
```

```
function GetElmt (T : TabInt, i : IdxType) → ElType
{ Prekondisi : Tabel tidak kosong, i antara FirstIdx(T)..LastIdx(T) }
{ Mengirimkan elemen tabel yang ke-i }

{ *** Selektor SET : Mengubah nilai TABEL dan elemen tabel *** }
{ Untuk type private/limited private pada bahasa tertentu }
procedure SetTab (input Tin : TabInt, output Tout : TabInt)
{ I.S. Tin terdefinisi, sembarang }
{ F.S. Tout berisi salinan Tin }
{ Assignment THsl ← Tin }
procedure SetEl (input/output T : TabInt, input i : IdxType, input v : ElType)
{ I.S. T terdefinisi, sembarang }
{ F.S. Elemen T yang ke-i bernilai v }
{ Mengeset nilai elemen tabel yang ke-i sehingga bernilai v }
procedure SetNeff (input/output T : TabInt, input N : IdxType)
{ I.S. T terdefinisi, sembarang }
{ F.S. Nilai indeks efektif T bernilai N }
{ Mengeset nilai indeks elemen efektif sehingga bernilai N }

{ ********** Test Indeks yang valid ********** }
function IsIdxValid (T : TabInt, i : IdxType) → boolean
{ Prekondisi : i sembarang }
{ Mengirimkan true jika i adalah indeks yang valid utk ukuran tabel }
{ yaitu antara indeks yang terdefinisi utk container}
function IsIdxEff (T : TabInt, i : IdxType) → boolean
{ Prekondisi : i sembarang}
{ Mengirimkan true jika i adalah indeks yang terdefinisi utk tabel }
{ yaitu antara FirstIdx(T)..LastIdx(T) }

{ ********** TEST KOSONG/PENUH ********** }
{ *** Test tabel kosong *** }
function IsEmpty (T : TabInt) → boolean
{ Mengirimkan true jika tabel T kosong, mengirimkan false jika tidak }
{ *** Test tabel penuh *** }
function IsFull (T : TabInt) → boolean
{ Mengirimkan true jika tabel T penuh, mengirimkan false jika tidak }

{ ********** BACA dan TULIS dengan INPUT/OUTPUT device ********** }
{ *** Mendefinisikan isi tabel dari pembacaan *** }
procedure BacaIsi (output T : TabInt)
{ I.S. sembarang }
{ F.S. tabel T terdefinisi }
{ Proses : membaca banyaknya elemen T dan mengisi nilainya }
procedure TulisIsi (input T : TabInt)
{ Proses : Menuliskan isi tabel dengan traversal }
{ I.S. T boleh kosong }
{ F.S. Jika T tidak kosong : indeks dan elemen tabel ditulis berderet ke bawah }
{      Jika T kosong : Hanya menulis "Tabel kosong" }
procedure TulisIsiTab (input T : TabInt)
{ Proses : Menuliskan isi tabel dengan traversal, tabel ditulis di antara kurung
siku; antara dua elemen dipisahkan dengan separator "koma" }
{ I.S. T boleh kosong }
{ F.S. Jika T tidak kosong: [e1, e2, ... ,en] }
{ Contoh : jika ada tiga elemen bernilai 1, 20, 30 : [1, 20, 30] }
{ Jika tabel kosong : menulis [] }

{ ********** OPERATOR ARITMATIKA ********** }
{ *** Aritmatika tabel : Penjumlahan, pengurangan, perkalian, ... *** }
function PlusTab (T1, T2 : TabInt) → TabInt
{ Prekondisi : T1 dan T2 berukuran sama dan tidak kosong }
{ Mengirimkan  T1 + T2 }
function MinusTab (T1, T2 : TabInt) → TabInt
{ Prekondisi : T1 dan T2 berukuran sama dan tidak kosong }
{ Mengirimkan T1 - T2 }
```

```
function KaliTab (T1, T2 : TabInt) → TabInt
{ Prekondisi : T1 dan T2 berukuran sama dan tidak kosong }
{ Mengirimkan T1 * T2 dengan definisi setiap elemen dengan indeks yang sama
dikalikan }
function KaliKons (Tin : TabInt, c : ElType) → TabInt
{ Prekondisi : Tin tidak kosong }
{ Mengirimkan tabel dengan setiap elemen Tin dikalikan c }

{ ********** OPERATOR RELASIONAL ********** }
{ *** Operasi pembandingan tabel : < =, > *** }
function IsEQ (T1 : TabInt, T2 : TabInt) → boolean
{ Mengirimkan true jika T1 sama dengan T2 yaitu jika ukuran T1 = T2 dan semua
elemennya sama }
function IsLess (T1 : TabInt, T2 : TabInt) → boolean
{ Mengirimkan true jika T1 < T2, }
{ yaitu : sesuai dg analogi 'Ali' < Badu'; maka [0, 1] < [2, 3] }

{ ********** SEARCHING ********** }
{ ***  Perhatian : Tabel boleh kosong!! *** }
function Search1 (T : TabInt, X : ElType) → IdxType
{ Search apakah ada elemen tabel T yang bernilai X }
{ Jika ada, menghasilkan indeks i terkecil, dengan elemen ke-i = X }
{ Jika tidak ada, mengirimkan IdxUndef }
{ Menghasilkan indeks tak terdefinisi (IdxUndef) jika tabel T kosong }
{ Memakai skema search TANPA boolean }
function Search2 (T: TabInt, X : ElType) → IdxType
{ Search apakah ada elemen tabel T yang bernilai X }
{ Jika ada, menghasilkan indeks i terkecil, dengan elemen ke-i = X }
{ Jika tidak ada, mengirimkan IdxUndef }
{ Menghasilkan indeks tak terdefinisi (IdxUndef) jika tabel T kosong }
{ Memakai skema search DENGAN boolean Found }
function SearchB (T : TabInt, X : ElType) → boolean
{ Search apakah ada elemen tabel T yang bernilai X }
{ Jika ada, menghasilkan true, jika tidak ada menghasilkan false }
{ Menghasilkan indeks tak terdefinisi (IdxUndef) jika tabel T kosong }
{ Memakai Skema search DENGAN boolean }
function SearchSentinel (T : TabInt, X : ElType) → integer
{ Search apakah ada elemen tabel T yang bernilai X }
{ Jika ada, menghasilkan true, jika tidak ada menghasilkan false }
{ dengan metoda sequential search dengan sentinel }
{ Untuk sentinel, manfaatkan indeks ke-0 dalam definisi array dalam Bahasa C yang
tidak dipakai dalam definisi tabel }
{ Menghasilkan indeks tak terdefinisi (IdxUndef) jika tabel T kosong }

{ ********** NILAI EKSTREM ********** }
function ValMax (T : TabInt) → ElType
{ Prekondisi : Tabel T tidak kosong }
{ Mengirimkan nilai maksimum tabel }
function ValMin (T : TabInt) → ElType
{ Prekondisi : Tabel T tidak kosong }
{ Mengirimkan nilai minimum tabel }
{ *** Mengirimkan indeks elemen bernilai ekstrem *** }
function IdxMaxTab (T : TabInt) → IdxType
{ Prekondisi : Tabel T tidak kosong }
{ Mengirimkan indeks i dengan elemen ke-i adalah nilai maksimum pada tabel }
function IdxMinTab (T : TabInt) → IdxType
{ Prekondisi : Tabel tidak kosong }
{ Mengirimkan indeks i }
{ dengan elemen ke-i nilai minimum pada tabel }

{ ********** OPERASI LAIN ********** }
procedure CopyTab (input Tin : TabInt, output Tout : TabInt)
{ I.S. sembarang }
{ F.S. Tout berisi salinan dari Tin (elemen dan ukuran identik) }
{ Proses : Menyalin isi Tin ke Tout }
```

```
function InverseTab (T : TabInt) → TabInt
{ Menghasilkan tabel dengan urutan tempat yang terbalik, yaitu : }
{ elemen pertama menjadi terakhir, }
{ elemen kedua menjadi elemen sebelum terakhir, dst.. }
{ Tabel kosong menghasilkan tabel kosong }
function IsSimetris (T : TabInt) → boolean
{ Menghasilkan true jika tabel simetrik }
{ Tabel disebut simetrik jika: }
{      elemen pertama = elemen terakhir, }
{      elemen kedua = elemen sebelum terakhir, dan seterusnya }
{ Tabel kosong adalah tabel simetris }

{ ********** SORTING ********** }
procedure MaxSortDesc (input/output T : TabInt)
{ I.S. T boleh kosong }
{ F.S. T elemennya terurut menurun dengan Maximum Sort }
{ Proses : mengurutkan T sehingga elemennya menurun/mengecil }
{          tanpa menggunakan tabel kerja }
procedure InsSortAsc (input/output T : TabInt)
{ I.S. T boleh kosong }
{ F.S. T elemennya terurut menaik dengan Insertion Sort }
{ Proses : mengurutkan T sehingga elemennya menaik/membesar }
{          tanpa menggunakan tabel kerja }

{ ********** MENAMBAH ELEMEN ********** }
{ *** Menambahkan elemen terakhir *** }
procedure AddAsLastEl (input/output T : TabInt, input X : ElType)
{ Menambahkan X sebagai elemen terakhir tabel }
{ I.S. Tabel boleh kosong, tetapi tidak penuh }
{ F.S. X adalah elemen terakhir T yang baru }
{ Proses : Menambahkan sebagai elemen ke-i yang baru }
procedure AddEli (input/output T : TabInt, input X : ElType, input i : IdxType)
{ Menambahkan X sebagai elemen ke-i tabel tanpa mengganggu kontiguitas terhadap
elemen yang sudah ada }
{ I.S. Tabel tidak kosong dan tidak penuh }
{      i adalah indeks yang valid. }
{ F.S. X adalah elemen ke-i T yang baru }
{ Proses : Geser elemen ke-i+1 s.d. terakhir }
{          Isi elemen ke-i dengan X }

{ ********** MENGHAPUS ELEMEN ********** }
procedure DelLastEl (input/output T : TabInt, output X : ElType)
{ Proses : Menghapus elemen terakhir tabel }
{ I.S. Tabel tidak kosong }
{ F.S. X adalah nilai elemen terakhir T sebelum penghapusan, }
{      Banyaknya elemen tabel berkurang satu }
{      Tabel T mungkin menjadi kosong }
procedure DelEli(input/output T : TabInt, input i : IdxType, output X : ElType)
{ Proses : Menghapus elemen ke-i tabel tanpa mengganggu kontiguitas }
{ I.S. Tabel tidak kosong, i adalah indeks efektif yang valid }
{ F.S. Elemen T berkurang satu }
{      Banyaknya elemen tabel berkurang satu }
{      Tabel T mungkin menjadi kosong }
{ Proses : Geser elemen ke-i+1 s.d. elemen terakhir }
{          Kurangi elemen efektif tabel }

{ ********** TABEL DGN ELEMEN UNIK (SETIAP ELEMEN HANYA MUNCUL 1 KALI) ********** }

procedure AddElUnik (input/output T : TabInt, input X : ElType)
{ Menambahkan X sebagai elemen terakhir tabel, pada tabel dengan elemen unik}
{ I.S. Tabel boleh kosong, tetapi tidak penuh }
{      dan semua elemennya bernilai unik, tidak terurut}
{ F.S. Jika tabel belum penuh, menambahkan X sbg elemen terakhir T, jika belum ada
elemen yang bernilai X. Jika sudah ada elemen tabel yang bernilai X maka I.S. =
F.S. dan dituliskan pesan "nilai sudah ada" }
{ Proses : Cek keunikan dengan sequential search dengan sentinel}
{          Kemudian tambahkan elemen jika belum ada }
```

```
{ ********** TABEL DGN ELEMEN TERURUT MEMBESAR ********** }

function SearchUrut (T : TabInt, X : ElType) → IdxType
{ Prekondisi: Tabel boleh kosong. Jika tidak kosong, elemen terurut membesar. }
{ mengirimkan indeks di mana harga X dengan indeks terkecil diketemukan }
{ mengirimkan IdxUndef jika tidak ada elemen tabel bernilai X }
{ Menghasilkan indeks tak terdefinisi (IdxUndef) jika tabel kosong }

function Max (T : TabInt) → ElType
{ Prekondisi : Tabel tidak kosong, elemen terurut membesar }
{ Mengirimkan nilai maksimum pada tabel}
function Min (T : TabInt) → ElType
{ Prekondisi : Tabel tidak kosong, elemen terurut membesar }
{ Mengirimkan nilai minimum pada tabel}
function MaxMin (T : TabInt) → <ElType, ElType>
{ Prekondisi : Tabel tidak kosong, elemen terurut membesar }
{ Mengirimkan nilai maksimum dan minimum pada tabel }

procedure Add1Urut (input/output T : TabInt, input X : ElType)
{ Menambahkan X tanpa mengganggu keterurutan nilai dalam tabel }
{ Nilai dalam tabel tidak harus unik. }
{ I.S. Tabel boleh kosong, boleh penuh. }
{      Jika tabel isi, elemennya terurut membesar. }
{ F.S. Jika tabel belum penuh, menambahkan X. }
{      Jika tabel penuh, maka tabel tetap. }
{ Proses : Search tempat yang tepat sambil geser }
{          Insert X pada tempat yang tepat tersebut tanpa mengganggu keterurutan }

procedure Del1Urut (input/output T : TabInt, input X : ElType)
{ Menghapus X yang pertama kali (pada indeks terkecil) yang ditemukan }
{ I.S. Tabel tidak kosong }
{ F.S. Jika ada elemen tabel bernilai X , }
{      maka banyaknya elemen tabel berkurang satu. }
{      Jika tidak ada yang bernilai X, tabel tetap. }
{      Setelah penghapusan, elemen tabel tetap kontigu! }
{ Proses : Search indeks ke-i dengan elemen ke-i=X. }
{          Delete jika ada. }
```

Bagian 2. Representasi Eksplisit – Dinamik

1. Buatlah ADT Tabel Kontigu dengan representasi **eksplisit** dan alokasi memori **dinamik** sesuai dengan definisi dan spesifikasi di bawah ini dalam Bahasa C dengan pembagian modul sebagaimana dijelaskan dalam kuliah. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
   Spesifikasi primitif yang dibuat sama seperti pada modul P-05. Bagian 1. Perhatikan perbedaan pada struktur data dan konstruktor. Perhatikan pula jika diperlukan penambahan/pengurangan primitif terkait perubahan struktur data.
2. Buatlah juga driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

```
{  MODUL TABEL INTEGER }
{  berisi definisi dan semua primitif pemrosesan tabel integer }
{  penempatan elemen selalu rapat kiri }
{  Versi II : dengan banyaknya elemen didefinisikan secara eksplisit, memori tabel
              dinamik }

{  Kamus Umum }

constant IdxMax : integer = 100
constant IdxMin : integer = 1
constant IdxUndef : integer = -999 { indeks tak terdefinisi}

{ Definisi elemen dan koleksi objek }
type IdxType : integer  { type indeks }
type ElType  : integer  { type elemen tabel }
type TabInt  : < TI : ..., { memori tempat penyimpan elemen (container) }
                           { cara definisi tergantung bahasa pemrograman }
                 Neff : integer [IdxMin..Size], { banyaknya elemen efektif }
                 Size : integer   { ukuran tabel }
               >
{ Jika T adalah TabInt, cara deklarasi dan akses: }
{ Deklarasi T : TabInt }
  Maka cara akses : }
  T.Neff untuk mengetahui banyaknya elemen
  T.TI untuk mengakses seluruh nilai elemen tabel
  T.TI[i] untuk mengakses elemen ke-i
  T.Size untuk mengakses ukuran tabel }
{ Definisi :
  Tabel kosong: T.Neff = 0
  Definisi elemen pertama : T.TI[i] dengan i=1
  Definisi elemen terakhir yang terdefinisi: T.TI[i] dengan i=T.Neff }

{ ********** KONSTRUKTOR ********** }
{ Konstruktor : create tabel kosong dengan ukuran N }
procedure MakeEmpty (input N : integer, output T : TabInt)
{ I.S. sembarang }
{ F.S. Terbentuk tabel T kosong dengan kapasitas N }
```

Bagian 3. Representasi Implisit – Statik

1. Buatlah ADT Tabel Kontigu dengan representasi **implisit** dan alokasi memori **statik** sesuai dengan definisi dan spesifikasi di bawah ini dalam Bahasa C dengan pembagian modul sebagaimana dijelaskan dalam kuliah. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
   Spesifikasi primitif yang dibuat sama seperti pada modul P-05. Bagian 1. Perhatikan perbedaan pada struktur data dan konstruktor. Perhatikan pula jika diperlukan penambahan/pengurangan primitif terkait perubahan struktur data.
2. Buatlah juga driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

```
{  MODUL TABEL INTEGER }
{  berisi definisi dan semua primitif pemrosesan tabel integer }
{  penempatan elemen selalu rapat kiri }
{  Versi III : dengan banyaknya elemen didefinisikan secara implisit, memori tabel
               statik }

{  Kamus Umum }

constant IdxMax : integer = 100
constant IdxMin : integer = 1
constant IdxUndef : integer = -999  { indeks tak terdefinisi }
constant ValUndef : integer = -9999 { nilai tidak terdefinisi }

{ Definisi elemen dan koleksi objek }
type IdxType : integer  { type indeks }
type ElType  : integer  { type elemen tabel }
type TabInt  : < TI : array [IdxMin..IdxMax] of ElType,
                      { memori tempat penyimpan elemen (container) }
               >
{ Jika T adalah TabInt, cara deklarasi dan akses: }
{ Deklarasi T : TabInt }
{ Maka cara akses :
  T.TI untuk mengakses seluruh nilai elemen tabel
  T.TI[i] untuk mengakses elemen ke-i }
{ Definisi :
  Tabel kosong: semua elemen tabel berisi ValUndef
  Definisi elemen pertama : T.TI[i] dengan i=1 }

{ ********** KONSTRUKTOR ********** }
{ Konstruktor : create tabel kosong  }
procedure MakeEmpty (output T : TabInt)
{ I.S. sembarang }
{ F.S. Terbentuk tabel T kosong dengan semua elemen bernilai ValUndef }
```

Bagian 4. Representasi Implisit – Dinamik

1. Buatlah ADT Tabel Kontigu dengan representasi **implisit** dan alokasi memori **dinamik** sesuai dengan definisi dan spesifikasi di bawah ini dalam Bahasa C dengan pembagian modul sebagaimana dijelaskan dalam kuliah. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
   Spesifikasi primitif yang dibuat sama seperti pada modul P-05. Bagian 1. Perhatikan perbedaan pada struktur data dan konstruktor. Perhatikan pula jika diperlukan penambahan/pengurangan primitif terkait perubahan struktur data.
2. Buatlah juga driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

```
{  MODUL TABEL INTEGER }
{  berisi definisi dan semua primitif pemrosesan tabel integer }
{  penempatan elemen selalu rapat kiri }
{  Versi III : dengan banyaknya elemen didefinisikan secara implisit, memori tabel
               dinamik }

{  Kamus Umum }

constant IdxMax : integer = 100
constant IdxMin : integer = 1
constant IdxUndef : integer = -999  { indeks tak terdefinisi }
constant ValUndef : integer = -9999 { nilai tidak terdefinisi }

{ Definisi elemen dan koleksi objek }
type IdxType : integer  { type indeks }
type ElType  : integer  { type elemen tabel }
type TabInt  : < TI : ..., { memori tempat penyimpan elemen (container),
                             cara definisi tergantung pada bahasa }
                 Size : integer   { ukuran tabel }
               >
{ Jika T adalah TabInt, cara deklarasi dan akses: }
{ Deklarasi T : TabInt }
{ Maka cara akses : }
  T.Size untuk mengakses ukuran tabel
  T.TI untuk mengakses seluruh nilai elemen tabel
  T.TI[i] untuk mengakses elemen ke-i }
{ Definisi :
  Tabel kosong: semua elemen tabel berisi ValUndef
  Definisi elemen pertama : T.TI[i] dengan i=1 }

{ ********** KONSTRUKTOR ********** }
{ Konstruktor: create tabel kosong  }
procedure MakeEmpty (input N : integer, output T : TabInt)
{ I.S. sembarang }
{ F.S. Terbentuk tabel T kosong dengan kapasitas maksimum N dan semua elemen
       bernilai ValUndef }
```

P-06. ADT MATRIKS

1. Buatlah ADT MATRIKS sesuai dengan definisi dan spesifikasi di bawah ini dalam Bahasa C dengan pembagian modul sebagaimana dijelaskan dalam kuliah. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**). Dalam implementasi dengan bahasa C, nama fungsi untuk operator aritmatika dan operator relasional (seperti "+") tidak dimungkinkan. Buatlah nama yang sesuai untuk operator-operator tersebut.
2. Buatlah juga driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

```
{ ********** Definisi TYPE MATRIKS dengan indeks integer ********** }
{ Ukuran minimum dan maksimum baris dan kolom }
type indeks : integer { indeks baris, kolom }
constant BrsMin : indeks = 1
constant BrsMax : indeks = 100
constant KolMin : indeks = 1
constant KolMax : indeks = 100
type el_type : integer
type MATRIKS :
      < Mem : matrix(BrsMin..BrsMax, KolMin..KolMax) of el_type,
        NBrsEff : integer,  { banyaknya/ukuran baris yg terdefinisi }
        NKolEff : integer   { banyaknya/ukuran kolom yg terdefinisi }
      >
{ NBrsEff ≥ 1 dan NKolEff ≥ 1 }
{ Memori matriks yang dipakai selalu di "ujung kiri atas" }

{ ********** DEFINISI PROTOTIPE PRIMITIF ********** }
{ *** Konstruktor membentuk MATRIKS *** }
procedure MakeMATRIKS (input NB, NK : integer, output M : MATRIKS)
{ Membentuk sebuah MATRIKS "kosong" berukuran NB x NK di "ujung kiri" memori }
{ I.S. NB dan NK adalah valid untuk memori matriks yang dibuat }
{ F.S. Matriks M sesuai dengan definisi di atas terbentuk }

{ *** Selektor "DUNIA MATRIKS" *** }
function GetIdxBrsMin → indeks
{ Mengirimkan indeks baris minimum matriks apapun }
function GetIdxKolMin → indeks
{ Mengirimkan indeks kolom minimum matriks apapun }
function GetIdxBrsMax → indeks
{ Mengirimkan indeks baris maksimum matriks apapun }
function GetIdxKolMax → indeks
{ Mengirimkan indeks kolom maksimum matriks apapun }
function IsIdxValid (i, j : integer) → boolean
{ Mengirimkan true jika i, j adalah indeks yang valid }

{ *** Untuk sebuah matriks M yang terdefinisi: *** }
function FirstIdxBrs (M : MATRIKS) → indeks
{ Mengirimkan indeks baris terkecil M }
function FirstIdxKol (M : MATRIKS) → indeks
{ Mengirimkan indeks kolom terkecil M }
function LastIdxBrs (M : MATRIKS) → indeks
{ Mengirimkan indeks baris terbesar M }
function LastIdxKol (M : MATRIKS) → indeks
{ Mengirimkan indeks kolom terbesar M }
function GetNBrsEff (M : MATRIKS) → integer
{ Mengirimkan banyaknya baris efektif M }
function GetNKolEff (M : MATRIKS) →  integer
{ Mengirimkan banyaknya kolom efektif M }
function IsIdxEff (M : MATRIKS, i, j : indeks) → boolean
{ Mengirimkan true jika i, j adalah indeks efektif bagi M }
function GetElmt (M : MATRIKS, i, j : indeks) → el_type
{ Mengirimkan elemen M dg nomor baris i dan nomor kolom j }
```

```
function GetElmtDiagonal (M : MATRIKS, i : indeks) → el_type
{ Mengirimkan elemen M(i,i) }

{ *** Operasi mengubah nilai elemen matriks: Set / Assign *** }
procedure SetBrsEff (input/output M : MATRIKS, input NB : integer)
{ I.S. M sudah terdefinisi }
{ F.S. Nilai M.BrsEff diisi dengan NB,  }
procedure SetKolEff (input/output M : MATRIKS, input NK : integer)
{ I.S. M sudah terdefinisi }
{ F.S. Nilai M.NKolEff diisi dengan NK }
procedure SetEl (input/output M : MATRIKS, input i, j : integer,
                 input X : el_type)
{ I.S. M sudah terdefinisi }
{ F.S. M(i,j) bernilai X }
{ Proses: Mengisi M(i,j) dengan X }

{ ********** Assignment  MATRIKS ********** }
procedure CopyMATRIKS (input MIn : MATRIKS, output MHsl : MATRIKS)
{ Melakukan assignment MHsl ← Min }

{ ********** KELOMPOK BACA/TULIS ********** }
procedure BacaMATRIKS (output M : MATRIKS, input NB, NK : integer)
{ I.S. IsIdxValid(NB,NK) }
{ F.S. M terdefinisi nilai elemen efektifnya, dan berukuran NB x NK }
{ Melakukan MakeMatriks(M,NB,NK) dan mengisi nilai efektifnya }
{ dari pembacaan dengan traversal per baris }
procedure TulisMATRIKS (input M : MATRIKS)
{ I.S. M terdefinisi }
{ F.S. Sama dengan I.S, dan nilai M(i,j) ditulis ke layar}
{ Menulis nilai setiap indeks dan elemen M ke layar dengan traversal per baris }

{ ********** KELOMPOK OPERASI ARITMATIKA TERHADAP TYPE ********** }
function "+" (M1, M2 : MATRIKS) → MATRIKS
{ Prekondisi : M1  berukuran sama dengan M2 }
{ Mengirim hasil penjumlahan matriks: M1 + M2 }
function "-" (M1, M2 : MATRIKS) → MATRIKS
{ Prekondisi : M berukuran sama dengan M }
{ Mengirim hasil pengurangan matriks: salinan M1 - M2 }
function "*" (M1, M2 : MATRIKS) → MATRIKS
{ Prekondisi : Ukuran kolom efektif M1 = ukuran baris efektif M2 }
{ Mengirim hasil perkalian matriks: salinan M1 * M2 }
function "*" (M : MATRIKS, X : integer) → MATRIKS
{ Mengirim hasil perkalian setiap elemen M dengan X }
procedure "*" (input/output M : MATRIKS, input K : integer)
{ Mengalikan setiap elemen M dengan K }

{ ********** KELOMPOK OPERASI RELASIONAL TERHADAP MATRIKS ********** }
function "=" (M1, M2 : MATRIKS) → boolean
{ Mengirimkan true jika M1 = M2, }
{ yaitu NBElmt(M1) = NBElmt(M2) dan }
{ untuk setiap i,j yang merupakan indeks baris dan kolom }
{ M1(i,j) = M2(i,j) }
function NEQ (M1, M2 : MATRIKS) → boolean
{ Mengirimkan true jika not strongEQ(M1,M2) }
function EQSize (M1, M2 : MATRIKS) →  boolean
{ Mengirimkan true jika ukuran efektif matriks M1 sama dengan ukuran efektif M2 }
{ yaitu GetBrsEff(M1) = GetNBrsEff (M2) dan GetNKolEff (M1) = GetNKolEff (M2) }
function "<" (M1,M2 : MATRIKS) → boolean
{ Mengirimkan true jika ukuran efektif M1 < ukuran efektif M2 }

{ ********** Operasi lain ********** }
function NBElmt (M : Matriks) → integer
{ Mengirimkan banyaknya elemen M }
```

```
{ ********** KELOMPOK TEST TERHADAP MATRIKS ********** }
function IsBujurSangkar (M : Matriks) → boolean
{ Mengirimkan true jika M adalah matriks dg ukuran baris dan kolom sama }
function IsSymetri (M : Matriks) → boolean
{ Mengirimkan true jika M adalah matriks simetri : IsBujurSangkar(M) dan untuk
setiap elemen M, M(i,j)=M(j,i) }
function IsSatuan (M : Matriks) → boolean
{ Mengirimkan true jika M adalah matriks satuan: IsBujurSangkar(M) dan setiap
elemen diagonal M bernilai 1 dan elemen yang bukan diagonal bernilai 0 }
function IsSparse (M : Matriks) → boolean
{ Mengirimkan true jika M adalah matriks sparse: mariks "jarang" dengan definisi:
hanya maksimal 5% dari memori matriks yang efektif bukan bernilai 0 }
function Invers1 (M : Matriks) → MATRIKS
{ Menghasilkan salinan M dg setiap elemen "di-invers", yaitu dinegasikan }
function Invers (M : Matriks) → MATRIKS
{ Menghasilkan salinan M dg setiap elemen "di-invers", yaitu di-invers sesuai
dengan aturan inversi matriks }
function Determinan (M : Matriks) → real
{ Prekondisi: IsBujurSangkar(M) }
{ Menghitung nilai determinan M }
procedure Invers1(input/output M : MATRIKS)
{ I.S. M terdefinisi }
{ F.S. M di-invers, yaitu setiap elemennya dinegasikan }
procedure Invers(input/output M : MATRIKS)
{ I.S. M terdefinisi }
{ F.S. M "di-invers", yaitu diproses sesuai dengan aturan invers matriks }
procedure Transpose (input/output M : MATRIKS)
{ I.S. M terdefinisi dan IsBujursangkar(M) }
{ F.S. M "di-transpose", yaitu setiap elemen M(i,j) ditukar nilainya dengan elemen
M(j,i) }
```

P-07. MESIN KATA

Bagian 1. Model Akuisisi Versi-1

1. Buatlah modul mesin karakter berdasarkan diktat ”Contoh Program Kecil dalam Bahasa C” (mesinkar.h dan mesinkar.c atau mesinkar1.h dan mesinkar1.c).
2. Dengan memanfaatkan mesin karakter tersebut, buatlah mesin kata dengan spesifikasi yang tercantum di bawah ini. Mesin kata dibuat dengan memanfaatkan pola pada studi kasus hitung panjang rata-rata kata pada pita karakter **versi 1** (menggunakan boolean EndKata sebagai penanda akhir proses akuisisi) yang dapat dilihat pada Diktat Dasar Pemrograman Bagian Pemrograman Prosedural. Pada tugas ini, proses hitung panjang rata-rata kata diubah dengan mengakuisisi dan menyimpan kata dalam sebuah *state* CKata melalui prosedur SalinKata (seperti pada studi kasus ”Hitung While”).
3. Buatlah sebuah driver untuk memeriksa apakah semua primitif mesin kata telah berjalan dengan baik. Driver dapat dibuat berdasarkan kasus ”Hitung While”.

```
{ ***** Mesin lain yang dipakai ***** }
use MESINKAR

{ ***** Konstanta ***** }
constant MARK  : character = '.'
constant BLANK : character = ' '
constant NMax  : integer = 50 { jumlah maksimum karakter suatu kata }

{ ***** Definisi Type Kata ***** }
type Kata : < TabKata : array [1..NMax] of character,
              Length : integer >
{ TabKata adalah tempat penampung/container kata,
  Length menyatakan panjangnya kata }

{ ***** Definisi State Mesin Kata ***** }
EndKata : boolean { penanda akhir akuisisi kata }
CKata : Kata      { kata yang sudah diakuisisi dan akan diproses }

{ ***** Primitif-Primitif Mesin Kata ***** }
procedure Ignore_Blank
{ Mengabaikan satu atau beberapa BLANK }
{ I.S. : CC sembarang }
{ F.S. : CC ≠ BLANK atau CC = MARK }
procedure STARTKATA
{ I.S. : CC sembarang }
{ F.S. : EndKata = true, dan CC = Mark; }
{        atau EndKata = false, CKata adalah kata yang sudah diakuisisi,
         CC karakter pertama sesudah karakter terakhir kata }
procedure ADVKATA
{ I.S. : EndKata = false; CC adalah karakter sesudah karakter terakhir
         dari kata yg sudah diakuisisi }
{ F.S. : Jika CC = MARK, maka EndKata = true
         atau EndKata = false, CKata adalah kata terakhir yang sudah diakuisisi;
         CC karakter pertama sesudah karakter terakhir kata }
procedure SalinKata
{ Mengakuisisi kata, menyimpan dalam CKata }
{ I.S. : CC adalah karakter pertama dari kata }
{ F.S. : CKata berisi kata yang sudah diakuisisi, jika karakternya melebihi
         NMax, sisa "kata" dibuang; CC = BLANK atau CC = MARK; CC adalah
         karakter sesudah karakter terakhir yang diakuisisi }

{ ***** Operasi Lain ***** }
function IsKataSama (K1, K2 : Kata) → boolean
{ Mengembalikan true jika K1 = K2; dua kata dikatakan sama jika panjangnya sama dan
urutan karakter yang menyusun kata juga sama }
```

Bagian 2. Model Akuisisi Versi-2

1. Buatlah modul mesin karakter berdasarkan diktat ”Contoh Program Kecil dalam Bahasa C” (mesinkar.h dan mesinkar.c atau mesinkar1.h dan mesinkar1.c).
2. Dengan memanfaatkan mesin karakter tersebut, buatlah mesin kata dengan spesifikasi yang tercantum di bawah ini. Mesin kata dibuat dengan memanfaatkan pola pada studi kasus hitung panjang rata-rata kata pada pita karakter **versi 2** (menggunakan *state* LKata yang menyatakan panjang kata sehingga proses akuisisi kata berakhir jika menemukan LKata = 0) yang dapat dilihat pada Diktat Dasar Pemrograman Bagian Pemrograman Prosedural. Pada tugas ini, proses hitung panjang rata-rata kata diubah dengan mengakuisisi dan menyimpan kata dalam sebuah *state* CKata melalui prosedur SalinKata (seperti pada studi kasus ”Hitung While”).
3. Buatlah sebuah driver untuk memeriksa apakah semua primitif mesin kata telah berjalan dengan baik. Driver dapat dibuat berdasarkan kasus ”Hitung While”.

```
{ ***** Mesin lain yang dipakai ***** }
use MESINKAR

{ ***** Konstanta ***** }
constant MARK  : character = '.'
constant BLANK : character = ' '
constant NMax  : integer = 50 { jumlah maksimum karakter suatu kata }

{ ***** Definisi Type Kata ***** }
type Kata : < TabKata : array [1..NMax] of character,
              Length : integer >
{ TabKata adalah tempat penampung/container kata,
  Length menyatakan panjangnya kata }

{ ***** Definisi State Mesin Kata ***** }
CKata : Kata    { kata yang sudah diakuisisi }

{ ***** Primitif-Primitif Mesin Kata ***** }
procedure Ignore_Blank
{ Mengabaikan satu atau beberapa BLANK }
{ I.S. : CC sembarang }
{ F.S. : CC ≠ BLANK atau CC = MARK }
procedure STARTKATA
{ I.S. : CC sembarang }
{ F.S. : CKata.Length = 0, dan CC = Mark; }
{        atau CKata.Length ≠ 0, CKata adalah kata yang sudah diakuisisi,
         CC karakter pertama sesudah karakter terakhir kata }
procedure ADVKATA
{ I.S. : CKata.Length ≠ 0; CC adalah karakter sesudah karakter terakhir dari kata
         yang sudah diakuisisi }
{ F.S. : Jika CC = MARK, maka CKata.Length = 0;
         atau CKata.Length ≠ 0, CKata adalah kata terakhir yang sudah diakuisisi;
         CC karakter pertama sesudah karakter terakhir kata }
procedure SalinKata
{ Mengakuisisi kata, menyimpan dalam CKata.Length }
{ I.S. : CC adalah karakter pertama dari kata }
{ F.S. : CKata.Length berisi kata yang sudah diakuisisi, jika karakternya melebihi
         NMax, sisa "kata" dibuang; CC = BLANK atau CC = MARK; CC adalah
         karakter sesudah karakter terakhir yang diakuisisi }

{ ***** Operasi Lain ***** }
function IsKataSama (K1, K2 : Kata) → boolean
{ Mengembalikan true jika K1 = K2; dua kata dikatakan sama jika panjangnya sama dan
urutan karakter yang menyusun kata juga sama }
```

Bagian 3. Model Akuisisi Versi-3

1. Buatlah modul mesin karakter berdasarkan diktat ”Contoh Program Kecil dalam Bahasa C” (mesinkar.h dan mesinkar.c atau mesinkar1.h dan mesinkar1.c).
2. Dengan memanfaatkan mesin karakter tersebut, buatlah mesin kata dengan spesifikasi yang tercantum di bawah ini. Mesin kata dibuat dengan memanfaatkan pola pada studi kasus hitung panjang rata-rata kata pada pita karakter **versi 3** (model akuisisi tanpa mark) yang dapat dilihat pada Diktat Dasar Pemrograman Bagian Pemrograman Prosedural. Pada tugas ini, proses hitung panjang rata-rata kata diubah dengan mengakuisisi dan menyimpan kata dalam sebuah *state* CKata melalui prosedur SalinKata (seperti pada studi kasus ”Hitung While”).
3. Buatlah sebuah driver untuk memeriksa apakah semua primitif mesin kata telah berjalan dengan baik. Driver dapat dibuat berdasarkan kasus ”Hitung While”.

```
{ ***** Mesin lain yang dipakai ***** }
use MESINKAR

{ ***** Konstanta ***** }
constant MARK  : character = '.'
constant BLANK : character = ' '
constant NMax  : integer = 50 { jumlah maksimum karakter suatu kata }

{ ***** Definisi Type Kata ***** }
type Kata : < TabKata : array [1..NMax] of character,
              Length : integer >
{ TabKata adalah tempat penampung/container kata,
  Length menyatakan panjangnya kata }

{ ***** Definisi State Mesin Kata ***** }
CKata : Kata    { kata yang sudah diakuisisi }

{ ***** Primitif-Primitif Mesin Kata ***** }
procedure Ignore_Blank
{ Mengabaikan satu atau beberapa BLANK }
{ I.S. : CC sembarang }
{ F.S. : CC ≠ BLANK atau CC = MARK }
procedure INITAKSES
{ Mengabaikan satu atau beberapa BLANK pada awal pita }
{ I.S. : CC sembarang }
{ F.S. : CC = MARK; atau CC = karakter pertama dari kata yang akan diakuisisi }
procedure ADVKATA
{ I.S. : CC adalah karakter pertama yang akan diakusisi }
{ F.S. : CKata adalah kata terakhir yang sudah diakuisisi,
         CC adalah karakter pertama dari kata yang berikutnya, mungkin MARK. }
procedure SalinKata
{ Mengakuisisi kata, menyimpan dalam CKata }
{ I.S. : CC adalah karakter pertama dari kata }
{ F.S. : CKata berisi kata yang sudah diakuisisi, jika karakternya melebihi
         NMax, sisa "kata" dibuang; CC = BLANK atau CC = MARK; CC adalah
         karakter sesudah karakter terakhir yang diakuisisi }

{ ***** Operasi Lain ***** }
function IsKataSama (K1, K2 : Kata) → boolean
{ Mengembalikan true jika K1 = K2; dua kata dikatakan sama jika panjangnya sama dan
urutan karakter yang menyusun kata juga sama }
```

P-08. ADT STACK

Bagian 1. Representasi Tabel Kontigu dengan Alokasi Memori Statik

1. Buatlah **ADT Stack** yang diimplementasikan dengan tabel kontigu dengan **alokasi memori statik** dalam bahasa C berdasarkan atau pada spesifikasi di bawah ini dalam Bahasa C dengan pembagian modul sebagaimana dijelaskan dalam kuliah. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
2. Buatlah driver untuk memeriksa apakah seluruh primitif yang didefinisikan telah berjalan dengan baik.

```
{ MODUL STACK }
{ Deklarasi stack yang diimplementasi dengan tabel kontigu alokasi statik }
{ dan ukuran sama }
{ TOP adalah alamat elemen puncak }

constant Nil   : integer = 0   { Nil adalah stack dengan elemen kosong }
constant MaxEl : integer = 10

type infotype : integer
type address  : integer   { indeks tabel }

{ Contoh deklarasi variabel bertype stack dengan ciri TOP : }
{ Versi I : dengan menyimpan tabel dan alamat top secara eksplisit}
type Stack : < T : array [1..MaxEl] of infotype, { tabel penyimpan elemen }
               TOP : address { alamat TOP : elemen puncak } >
{ Definisi stack S kosong : S.TOP = Nil }
{ Elemen yang dipakai menyimpan nilai Stack T[1]..T[MaxEl] }
{ Jika S adalah Stack maka akses elemen : }
{ S.T[(S.TOP)] untuk mengakses elemen TOP }
{ S.TOP adalah alamat elemen TOP }
{ Definisi akses dengan Selektor : Isilah dengan selektor yang tepat }

{ ********* Prototype ********* }

{ *** Konstruktor/Kreator *** }
procedure CreateEmpty (output S : Stack)
{ I.S. Sembarang }
{ F.S. Membuat sebuah stack S yang kosong berkapasitas MaxEl }
{ jadi indeksnya antara 1..MaxEl }
{ Ciri stack kosong : TOP bernilai Nil }

{ ********* Predikat Untuk test keadaan KOLEKSI ********* }
function IsEmpty (S : Stack) → boolean
{ Mengirim true jika Stack kosong: lihat definisi di atas }
function IsFull (S : Stack) → boolean
{ Mengirim true jika tabel penampung nilai elemen stack penuh }

{ ********** Operator Dasar Stack ********* }
procedure Push (input/output S : Stack, input X : infotype)
{ Menambahkan X sebagai elemen Stack S. }
{ I.S. S mungkin kosong, tabel penampung elemen stack TIDAK penuh }
{ F.S. X menjadi TOP yang baru, TOP bertambah 1 }
procedure Pop (input/output S : Stack, input X : infotype);
{ Menghapus X dari Stack S. }
{ I.S. S  tidak kosong }
{ F.S. X adalah nilai elemen TOP yang lama, TOP berkurang 1 }
```

Bagian 2. Representasi Tabel Kontigu dengan Alokasi Memori Dinamik

1. Buatlah **ADT Stack** yang diimplementasikan dengan tabel kontigu dengan **alokasi memori dinamik** dalam bahasa C berdasarkan atau pada spesifikasi di bawah ini dalam Bahasa C dengan pembagian modul sebagaimana dijelaskan dalam kuliah. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
2. Buatlah driver untuk memeriksa apakah seluruh primitif yang didefinisikan telah berjalan dengan baik.

```
{ MODUL STACK }
{ Deklarasi stack yang diimplementasi dengan tabel kontigu secara dinamik }
{ dan ukuran sama }
{ TOP adalah alamat elemen puncak }

constant Nil   : integer = 0   { Nil adalah stack dengan elemen kosong }

type infotype : integer
type address  : integer   { indeks tabel }

{ Contoh deklarasi variabel bertype stack dengan ciri TOP : }
{ Versi I : dengan menyimpan tabel dan alamat top secara eksplisit}
type Stack : < T : ...,        { tabel penyimpan elemen, tergantung bahasa }
               TOP : address,  { alamat TOP : elemen puncak }
               Size : integer  { ukuran stack }  >
{ Definisi stack S kosong : S.TOP = Nil }
{ Elemen yang dipakai menyimpan nilai Stack T[1]..T[Size] }
{ Jika S adalah Stack maka akses elemen : }
{ S.T[(S.TOP)] untuk mengakses elemen TOP }
{ S.TOP adalah alamat elemen TOP }
{ S.Size adalah ukuran stack }
{ Definisi akses dengan Selektor : Isilah dengan selektor yang tepat }

{ ********* Prototype ********* }

{ *** Konstruktor/Kreator *** }
procedure CreateEmpty (output S : Stack, input Size : integer)
{ I.S. Sembarang }
{ F.S. Membuat sebuah stack S yang kosong berkapasitas Size }
{ jadi indeksnya antara 1..MaxEl }
{ Ciri stack kosong : TOP bernilai Nil }

{ *** Destruktor *** }
procedure Destruct (input/output S : Stack)
{ Destruktor: dealokasi seluruh tabel memori sekaligus }

{ ********* Predikat Untuk test keadaan KOLEKSI ********* }
function IsEmpty (S : Stack) → boolean
{ Mengirim true jika Stack kosong: lihat definisi di atas }
function IsFull (S : Stack) → boolean
{ Mengirim true jika tabel penampung nilai elemen stack penuh }

{ ********** Operator Dasar Stack ********* }
procedure Push (input/output S : Stack, input X : infotype)
{ Menambahkan X sebagai elemen Stack S. }
{ I.S. S mungkin kosong, tabel penampung elemen stack TIDAK penuh }
{ F.S. X menjadi TOP yang baru,TOP bertambah 1 }
procedure Pop (input/output S : Stack, input X : infotype);
{ Menghapus X dari Stack S. }
{ I.S. S  tidak kosong }
{ F.S. X adalah nilai elemen TOP yang lama, TOP berkurang 1 }
```

P-09. ADT QUEUE

Bagian 1. Representasi Tabel Kontigu Alokasi Dinamik - Alternatif 1

1. Buatlah **ADT Queue** sesuai spesifikasi di bawah ini yang diimplementasikan dengan tabel kontigu dengan **alokasi dinamik** dalam bahasa C dengan representasi alternatif I yaitu dengan TAIL adalah indeks elemen terakhir dan HEAD selalu diset sama dengan 1 jika queue tidak kosong (lihat kembali diktat "Struktur Data"). Jika queue kosong, HEAD dan TAIL menunjuk indeks 0. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
2. Buatlah driver untuk memeriksa apakah seluruh primitif yang didefinisikan telah berjalan dengan baik.

```
{ Modul ADT Queue – Alternatif I }
{ *** Deklarasi Queue yang diimplementasi dengan tabel kontigu *** }
{ *** HEAD dan TAIL adalah alamat elemen pertama dan terakhir  *** }
{ *** Queue mampu menampung MaxEl buah elemen *** }

{ *** Konstanta *** }
constant Nil : integer = 0

{ *** Definisi elemen dan address *** }
type infotype : integer
type address  : integer   { indeks tabel }

{ *** Definisi Type Queue *** }
type Queue : < T     : ...,     { tabel penyimpan elemen, tergantung bahasa }
               HEAD  : address, { alamat penghapusan }
               TAIL  : address, { alamat penambahan  }
               MaxEl : integer  { maksimum banyaknya elemen queue }
             >
{ Definisi Queue kosong: Head = Nil; TAIL = Nil. }
{ Catatan implementasi: T[0] tidak pernah dipakai }

{ Definisi akses dengan Selektor : Isilah dengan selektor yang tepat }

{ *** Predikat Pemeriksaan Kondisi Queue *** }
function IsEmpty (Q : Queue) → boolean
{ Mengirim true jika Q kosong }
function IsFull (Q : Queue) → boolean
{ Mengirim true jika tabel penampung elemen Q sudah penuh yaitu mengandung MaxEl
elemen }
function NBElmt (Q : Queue) → integer
{ Mengirimkan banyaknya elemen queue. Mengirimkan 0 jika Q kosong. }

{ *** Konstruktor *** }
procedure CreateEmpty (output Q : Queue, input Max : integer > 0)
{ I.S. Max terdefinisi }
{ F.S. Sebuah Q kosong terbentuk dan salah satu kondisi sbb :       }
{      Jika alokasi berhasil, tabel memori dialokasi berukuran Max  }
{      atau : jika alokasi gagal, Q kosong dg Maksimum elemen=0     }
{ Proses : Melakukan alokasi memori dan membuat sebuah Q kosong     }

{ *** Destruktor *** }
procedure DeAlokasi (input/output Q : Queue)
{ Proses : Mengembalikan memori Q }
{ I.S. Q pernah dialokasi }
{ F.S. Q menjadi tidak terdefinisi lagi, MaxEl(Q) diset 0 }
```

```
{ *** Operator-Operator Dasar Queue *** }
procedure Add (input/output Q : Queue, input X : infotype)
{ Proses : Menambahkan  X pada Q dengan aturan FIFO }
{ I.S. Q mungkin kosong, tabel penampung elemen Q TIDAK penuh }
{ F.S. X menjadi TAIL yang baru, TAIL "maju" }
procedure Del (input/output Q : Queue, output X : infotype)
{ Proses: Menghapus elemen pertama pada Q dengan aturan FIFO }
{ I.S. Q tidak kosong }
{ F.S. X = nilai elemen HEAD pada I.S.,
       Jika Queue masih isi : HEAD diset tetap = 1, elemen-elemen setelah HEAD yang
       lama digeser ke "kiri", TAIL = TAIL - 1;
       Jika Queue menjadi kosong, HEAD = TAIL = Nil. }
```

Bagian 2. Representasi Tabel Kontigu Alokasi Dinamik - Alternatif II

1. Buatlah **ADT Queue** yang diimplementasikan dengan tabel kontigu dengan **alokasi dinamik** dalam bahasa C dengan representasi alternatif II yaitu dengan HEAD dan TAIL yang "bergerak". Setelah TAIL mencapai indeks terakhir, jika terjadi penambahan elemen baru dan tabel belum penuh, semua elemen dimundurkan kembali sampai HEAD = 1 (lihat kembali diktat "Struktur Data"). Jika queue kosong, HEAD dan TAIL menunjuk indeks 0. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
   Definisi konstanta, struktur data queue, primitif-primitif yang terkait dengan pemeriksaan kondisi queue, konstruktor, destruktor, dan selektor sama dengan P-09. Bagian 1.
2. Buatlah driver untuk memeriksa apakah seluruh primitif yang didefinisikan telah berjalan dengan baik.

```
{ Modul ADT Queue – Alternatif II }

{ Definisi konstanta, struktur data queue, primitif-primitif yang terkait dengan
pemeriksaan kondisi queue, konstruktor, destruktor, dan selektor sama dengan P-09.
Bagian 1. }

{ *** Operator-Operator Dasar Queue *** }
procedure Add (input/output Q : Queue, input X : infotype)
{ Proses : Menambahkan  X pada Q dengan aturan FIFO }
{ I.S. Q mungkin kosong, tabel penampung elemen Q TIDAK penuh }
{ F.S. X menjadi TAIL yang baru, TAIL "maju". }
{       Jika TAIL = MaxEl, semua elemen digeser mundur terlebih dahulu sampai
        HEAD = 1, baru X ditambahkan pada TAIL }
procedure Del (input/output Q : Queue, output X : infotype);
{ Proses : Menghapus elemen pertama pada Q dengan aturan FIFO }
{ I.S. Q tidak kosong }
{ F.S. X = nilai elemen HEAD pada I.S.,
        Jika Queue masih isi : HEAD "maju".
        Jika Queue menjadi kosong, HEAD = TAIL = Nil. }
```

Bagian 3. Representasi Tabel Kontigu Alokasi Dinamik - Alternatif III

1. Buatlah **ADT Queue** yang diimplementasikan dengan tabel kontigu dengan **alokasi dinamik** dalam bahasa C dengan representasi alternatif III yaitu dengan HEAD dan TAIL yang "berputar"mengelilingi indeks tabel (lihat kembali diktat "Struktur Data"). Jika queue kosong, HEAD dan TAIL menunjuk indeks 0. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
   Definisi konstanta, struktur data queue, primitif-primitif yang terkait dengan pemeriksaan kondisi queue, konstruktor, destruktor, dan selektor sama dengan P-09. Bagian 1.
2. Buatlah driver untuk memeriksa apakah seluruh primitif yang didefinisikan telah berjalan dengan baik.

```
{ Modul ADT Queue – Alternatif III }

{ Definisi konstanta, struktur data queue, primitif-primitif yang terkait dengan
pemeriksaan kondisi queue, konstruktor, destruktor, dan selektor sama dengan P-09.
Bagian 1. }

{ *** Operator-Operator Dasar Queue *** }
procedure Add (input/output Q : Queue, input X : infotype)
{ Proses : Menambahkan  X pada Q dengan aturan FIFO }
{ I.S. Q mungkin kosong, tabel penampung elemen Q TIDAK penuh }
{ F.S. X menjadi TAIL yang baru, TAIL "maju". }
{       Jika TAIL baru = MaxEl + 1, maka TAIL diset = 1. }
procedure Del (input/output Q : Queue, output X : infotype);
{ Proses : Menghapus elemen pertama pada Q dengan aturan FIFO }
{ I.S. Q tidak kosong }
{ F.S. X = nilai elemen HEAD pada I.S.,
       Jika Queue masih isi : HEAD "maju".
       Jika HEAD baru menjadi MaxEl + 1, maka HEAD diset = 1;
       Jika Queue menjadi kosong, HEAD = TAIL = Nil. }
```

P-10. ADT LIST BERKAIT LINIER

Bagian 1. Representasi Fisik Pointer – Type List dengan First Eksplisit

1. Buatlah ADT List Berkait Linier dengan representasi fisik pointer dalam bahasa C sesuai spesifikasi di bawah ini. Untuk type List, digunakan type list dengan elemen First yang dinyatakan secara eksplisit (lihat kembali diktat "Struktur Data"). Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
2. Buatlah driver untuk memeriksa apakah semua primitif yang didefinisikan telah berjalan dengan baik.

```
{ ***************** MODUL LIST BERKAIT ***************** }
{ List direpresentasi dengan pointer, First dinyatakan secara eksplisit. }

{ Konstanta }
constant Nil : ... { address tidak terdefinisi }

type infotype : integer
type ElmtList : < Info : infotype, Next : address >
type address : pointer to ElmtList
type List : < First : address >

{ Jika L : List dan P : address (address untuk traversal), maka penulisan : }
{      First(L) menjadi L.First
       Next(P)  menjadi P↑.Next
       Info(P)  menjadi P↑.Info   }

{ Definisikan selektor yang tepat. }

{ PROTOTYPE }
{ ***************** TEST LIST KOSONG ***************** }
function IsListEmpty (L : List) → boolean
{ Mengirim true jika list kosong }

{ ***************** PEMBUATAN LIST KOSONG ***************** }
procedure CreateList (output L : List)
{ I.S. sembarang             }
{ F.S. Terbentuk list kosong }

{ ***************** Manajemen Memori ***************** }
function Alokasi (X : infotype) → address
{ Mengirimkan address hasil alokasi sebuah elemen }
{ Jika alokasi berhasil, maka address tidak Nil, dan misalnya menghasilkan P, maka
  Info(P) = X, Next(P) = Nil }
{ Jika alokasi gagal, mengirimkan Nil }
procedure Dealokasi (input/output P : address)
{ I.S. P terdefinisi }
{ F.S. P dikembalikan ke sistem }
{ Melakukan dealokasi/pengembalian address P }

{ ***************** PENCARIAN SEBUAH ELEMEN LIST ***************** }
function Search (L : List, X : infotype) → address
{ Mencari apakah ada elemen list dengan Info(P) = X }
{ Jika ada, mengirimkan address elemen tersebut }
{ Jika tidak ada, mengirimkan Nil }
function FSearch (L : List, P : address) → boolean
{ Mencari apakah ada elemen list yang beralamat P }
{ Mengirimkan true jika ada, false jika tidak ada }
function SearchPrec (L : List, X : infotype) → address
{ Mengirimkan address elemen sebelum elemen yang nilainya = X }
{ Mencari apakah ada elemen list dengan Info(P) = X }
{ Jika ada, mengirimkan address Prec, dengan Next(Prec) = P dan Info(P) = X }
```

```
{ Jika tidak ada, mengirimkan Nil }
{ Jika P adalah elemen pertama, maka mengirimkan Nil }
{ Search dengan spesifikasi seperti ini menghindari traversal ulang jika setelah
  Search akan dilakukan operasi lain }

{ ***************** PRIMITIF BERDASARKAN NILAI ***************** }

{ *** PENAMBAHAN ELEMEN *** }
procedure InsVFirst (input/output L : List, input X : infotype)
{ I.S. L mungkin kosong }
{ F.S. X ditambahkan sebagai elemen pertama L }
{ Proses : Melakukan alokasi sebuah elemen dan menambahkan elemen pertama dengan
           nilai X jika alokasi berhasil.
           Jika alokasi gagal: I.S.= F.S. }
procedure InsVLast (input/output L : List, input X : infotype)
{ I.S. L mungkin kosong }
{ F.S. X ditambahkan sebagai elemen terakhir L }
{ Proses : Melakukan alokasi sebuah elemen dan menambahkan elemen list di akhir :
           elemen terakhir yang baru bernilai X jika alokasi berhasil.
           Jika alokasi gagal: I.S.= F.S. }

{ *** PENGHAPUSAN ELEMEN *** }
procedure DelVFirst (input/output L : List, input X : infotype)
{ I.S. List L tidak kosong }
{ F.S. Elemen pertama list dihapus : nilai info disimpan pada X }
{      dan alamat elemen pertama di-dealokasi }
procedure DelVLast (input/output L : List, output X : infotype)
{ I.S. list tidak kosong }
{ F.S. Elemen terakhir list dihapus : nilai info disimpan pada X }
{      dan alamat elemen terakhir di-dealokasi }

{ ***************** PRIMITIF BERDASARKAN ALAMAT ***************** }
{ *** PENAMBAHAN ELEMEN BERDASARKAN ALAMAT *** }
procedure InsertFirst (input/output L : List, input P : address)
{ I.S. Sembarang, P sudah dialokasi  }
{ F.S. Menambahkan elemen ber-address P sebagai elemen pertama }
procedure InsertAfter (input/output L : List, input P : address, input Prec :
address)
{ I.S. Prec pastilah elemen list dan bukan elemen terakhir, }
{      P sudah dialokasi  }
{ F.S. Insert P sebagai elemen sesudah elemen beralamat Prec }
procedure InsertLast (input/output L : List, input P : address P)
{ I.S. Sembarang, P sudah dialokasi }
{ F.S. P ditambahkan sebagai elemen terakhir yang baru }

{ *** PENGHAPUSAN SEBUAH ELEMEN *** }
procedure DeleteFirst (input/output L : List, output P : address)
{ I.S. List tidak kosong }
{ F.S. P adalah alamat elemen pertama list sebelum penghapusan }
{      Elemen list berkurang satu (mungkin menjadi kosong) }
{ First element yang baru adalah suksesor elemen pertama yang lama }
procedure DeleteP (input/output L : List, input X : infotype)
{ I.S. Sembarang }
{ F.S. Jika ada elemen list beraddress P, dengan Info(P) = X }
{ Maka P dihapus dari list dan di-dealokasi }
{ Jika tidak ada elemen list dengan Info(P) = X, maka list tetap }
{ List mungkin menjadi kosong karena penghapusan }
procedure DeleteLast (input/output L : List, input P : address)
{ I.S. List tidak kosong }
{ F.S. P adalah alamat elemen terakhir list sebelum penghapusan  }
{      Elemen list berkurang satu (mungkin menjadi kosong) }
{ Last element baru adalah predesesor elemen pertama yang lama, jika ada }
procedure DeleteAfter (input/output L : List, output Pdel : address, input Prec :
address)
{ I.S. List tidak kosong. Prec adalah anggota list L. }
{ F.S. Menghapus Next(Prec) : Pdel adalah alamat elemen list yang dihapus }
```

```
{ ***************** PROSES SEMUA ELEMEN LIST ***************** }
procedure PrintInfo (input L : List)
{ I.S. List mungkin kosong }
{ F.S. Jika list tidak kosong, }
{      Semua info yg disimpan pada elemen list diprint }
{      Jika list kosong, hanya menuliskan "list kosong" }
function NbElmt (L : List) → integer
{ Mengirimkan banyaknya elemen list; mengirimkan 0 jika list kosong }

{ *** Prekondisi untuk Max/Min/rata-rata : List tidak kosong *** }
function Max (L : List) → infotype
{ Mengirimkan nilai Info(P) yang maksimum }
function AdrMax (L : List) → address
{ Mengirimkan address P, dengan Info(P) yang bernilai maksimum }
function Min (L : List) → infotype
{ Mengirimkan nilai Info(P) yang minimum }
function AdrMin (L : List) → address
{ Mengirimkan address P, dengan Info(P) yang bernilai minimum }
function Average (L : List) → real
{ Mengirimkan nilai rata-rata Info(P) }

{ ***************** PROSES TERHADAP LIST ***************** }

procedure DeleteAll (input/output L : List)
{ Delete semua elemen list dan alamat elemen di-dealokasi }
{ I.S. : L terdefinisi, boleh kosong }
{ F.S. : Jika L tidak kosong, semua elemen list di-delete dan didealokasi }
procedure InversList (input/output L : List)
{ I.S. L terdefinisi, boleh kosong }
{ F.S. Elemen list L dibalik : }
{      Elemen terakhir menjadi elemen pertama, dan seterusnya. }
{      Membalik elemen list, tanpa melakukan alokasi/dealokasi. }
function FInversList (L : List) → List
{ Mengirimkan list baru, hasil invers dari L dengan menyalin semua elemn list. }
{ Alokasi mungkin gagal. Jika alokasi gagal, hasilnya list kosong dan semua elemen
  yang terlanjur di-alokasi, harus didealokasi. }
procedure CopyList (input/output L1 : List, output L2 : List)
{ I.S. L1 terdefinisi, L2 sembarang. F.S. L2 = L1 }
{ L1 dan L2 "menunjuk" kepada list yang sama. }
{ Tidak ada alokasi/dealokasi elemen baru. }
function FCopyList (L : List) → List
{ Mengirimkan list yang merupakan salinan L dengan melakukan alokasi elemen baru. }
{ Jika ada alokasi gagal, hasilnya list kosong dan semua elemen yang terlanjur di-
  alokasi, harus didealokasi. }
procedure CpAlokList (input Lin : List, input/output Lout : List)
{ I.S. Lout sembarang, Lin terdefinisi. }
{ F.S. Jika semua alokasi berhasil, maka Lout berisi hasil copy Lin }
{      Jika ada alokasi yang gagal, maka Lout = Nil dan semua elemen yang terlanjur
       dialokasi, didealokasi }
procedure Concat (input L1, L2 : List; input/output L3 : List)
{ I.S. L1 dan L2 terdefinisi, boleh kosong. }
{ F.S. L1 dan L2 tetap, L3 adalah hasil konkatenasi L1 dan L2 }
{ Jika semua alokasi berhasil, maka L3 adalah hasil konkatenasi L1 dan L2. }
{ Jika ada alokasi yang gagal, semua elemen yang sudah dialokasi harus didealokasi
  dan L3 = Nil. }
{ Konkatenasi dua buah list : L1 dan L2 menghasilkan L3 yang "baru". }
{ Elemen L3 adalah hasil alokasi elemen yang "baru". }
{ Jika ada alokasi yang gagal, maka L3 harus bernilai Nil dan semua elemen yang
  pernah dialokasi harus didealokasi. }
procedure Concat1 (input/output L1, L2 : List; output L3 : List)
{ I.S. L1 dan L2 sembarang }
{ F.S. L1 dan L2 kosong, L3 adalah hasil konkatenasi L1 dan L2 }
{ Konkatenasi dua buah list : L1 dan L2  menghasilkan L3 yang baru (dengan elemen
  list L1 dan L2) dan L1 serta L2 menjadi list kosong. }
{ Tidak ada alokasi/dealokasi pada prosedur ini }
```

```
procedure PecahList (output L1, L2 : List, input L : List)
{ I.S. L mungkin kosong }
{ F.S. Berdasarkan L, dibentuk dua buah list L1 dan L2 }
{ L tidak berubah : untuk membentuk L1 dan L2 harus alokasi. }
{ L1 berisi separuh elemen L dan L2 berisi sisa elemen L. }
{ Jika elemen L ganjil, maka separuh adalah NbElmt(L) div 2. }
```

Bagian 2. Representasi Fisik Pointer – Type List dengan First Implisit

1. Buatlah ADT List Berkait Linier dengan representasi fisik pointer dalam bahasa C sesuai spesifikasi di bawah ini. Untuk type List, digunakan type list dengan elemen First yang dinyatakan secara implisit (lihat kembali diktat "Struktur Data"). Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
   Primitif yang harus dibuat sama dengan modul P-10. Bagian 1.
2. Buatlah driver untuk memeriksa apakah semua primitif yang didefinisikan telah berjalan dengan baik.

```
{ ***************** MODUL LIST BERKAIT ***************** }
{ List direpresentasi dengan pointer, First dinyatakan secara implisit. }

{ Konstanta }
constant Nil : ... { address tidak terdefinisi }

type infotype : integer
type ElmtList : < Info : infotype, Next : address >
type address : pointer to ElmtList
type List : address

{ Jika L : List dan P : address (address untuk traversal), maka penulisan : }
{      First(L) menjadi L
       Next(P)  menjadi P↑.Next
       Info(P)  menjadi P↑.Info   }

{ Definisikan selektor yang tepat. }

{ Primitif-primitif yang harus dibuat sama seperti P-10. Bagian 1. }
```

Bagian 3. Representasi Fisik dengan Tabel Berkait

1. Buatlah ADT List Berkait Linier dengan representasi fisik tabel berkait dalam bahasa C sesuai spesifikasi di bawah ini (lihat kembali diktat "Struktur Data"). Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
   Primitif-primitif yang harus dibuat sama dengan modul P-10. Bagian 1.
2. Buatlah driver untuk memeriksa apakah semua primitif yang didefinisikan telah berjalan dengan baik.

```
{ ***************** MODUL LIST BERKAIT ***************** }
{ List direpresentasi secara berkait dengan tabel }

{ *** Konstanta *** }
constant IndexMin : integer = 1
constant IndexMax : integer = 100
constant Nil : integer = 0
{ Nil : address tak terdefinisi, di luar [IndexMin..IndekMax] }

{ *** Definisi List Berkait *** }
type infotype : integer
type ElmtList : < Info : infotype, Next : address >
type address  : integer [IndexMin..IndexMax, Nil]

type List : < First : address >

{ *** TABEL MEMORI LIST, GLOBAL *** }
TabElmt : array [IndexMin..IndexMax] of ElmtList
FirstAvail : address { alamat pertama list siap pakai }
{ Maka penulisan First(L) menjadi L.First
                 Next(P) menjadi TabElmt(P).Next
                 Info(P) menjadi TabElmt(P).Info }

{ Definisikan selektor yang tepat. }

{ *** Primitif Pemrosesan Tabel Berkait *** }
function MemFull
{ Mengirim true jika memori list sudah "habis" : FirstAvail = Nil }
procedure InitTab
{ Inisialisasi tabel yang akan dipakai sebagai memori list }
{ I.S. Sembarang. }
{ F.S. TabElmt[IndexMin..IndexMax] siap dipakai sebagai elemen list berkait,
       Elemen pertama yang available adalah FirstAvail = IndexMin,
       Next(i)=i+1 untuk i = [IndexMin..IndexMax-1], Next(IndexMax) = Nil }
procedure AllocTab (output P : address)
{ Mengambil sebuah elemen siap pakai P  pada awal list FirstAvail }
{ I.S. FirstAvail mungkin kosong. }
{ F.S. Jika FirstAvail tidak Nil, P adalah FirstAvail dan FirstAvail yang baru
       adalah  Next(FirstAvail) }
{      Jika FirstAvail = Nil, P = Nil,
       tulis pesan "Tidak tersedia lagi elemen siap pakai" }
procedure DeAllocTab (input P : address)
{ Mengembalikan sebuah elemen P pada awal list FirstAvail }
{ I.S. FirstAvail mungkin kosong. P tidak Nil. }
{ F.S. FirstAvail = P }

{ Primitif-primitif yang harus dibuat sama seperti P-10. Bagian 1. }
```

Bagian 4. Representasi Fisik dengan Tabel Kontigu

1. Buatlah ADT List Berkait Linier dengan representasi fisik tabel kontigu dalam bahasa C sesuai spesifikasi di bawah ini (lihat kembali diktat "Struktur Data"). Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
   Primitif-primitif yang harus dibuat sama dengan modul P-10. Bagian 1.
2. Buatlah driver untuk memeriksa apakah semua primitif yang didefinisikan telah berjalan dengan baik.

```
{ ***************** MODUL LIST BERKAIT ***************** }
{ List direpresentasi secara kontigu dengan tabel }

{ *** Konstanta *** }
constant IndexMin : integer  = 1
constant IndexMax : integer = 100
constant Nil : integer = 0

{ *** Definisi Type List *** }
type infotype : integer
type ElmtList : < Info : infotype >
                  { tidak perlu mengandung Next karena dapat dikalkulasi }
type address : integer [IndexMin..IndexMax, Nil]
type List : < TabMem : array [IndexMin..IndexMax] of ElmtList,
              N : address >
              { N alamat elemen terakhir.
                Karena field NEXT tidak ada secara eksplisit, maka
                satu-satunya jalan untuk mengenali elemen terakhir
                adalah dengan addressnya }

{ Jika L : List dan P : address (address untuk traversal)
  Maka First(L)..Last(L) adalah indeks efektif elemen tabel anggota list,
  Next(P) P ← P + 1, Next(P) tidak terdefinisi untuk P = N, }
  Info(P) menjadi L.TabMem[P].Info }

{ Definisikan selektor yang tepat. }

{ Primitif-primitif yang harus dibuat sama seperti P-10. Bagian 1. }
```

P-11. VARIASI LIST LINIER

Bagian 1. ADT List First-Last dengan Dummy pada Last

1. Buatlah ADT List yang memiliki penunjuk First dan Last, dengan elemen *dummy* pada *last element*, dengan dengan representasi fisik pointer, dalam bahasa C sesuai spesifikasi di bawah ini. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
   Elemen dummy di-create di awal, dan elemen ini selalu menjadi elemen dummy.
2. Buatlah driver untuk memeriksa apakah semua primitif yang didefinisikan telah berjalan dengan baik.

```
{ ADT list dummy last, berkait dengan representasi fisik pointer }
{ Dengan 2 penunjuk, first dan last, dummy selalu menunjuk last }
{ Representasi address dengan pointer }
{ infotype adalah integer }
{ Dummy dialokasi pada saat create list, dan selalu menjadi elemen dummy }

{ *** Konstanta *** }
constant Nil : ...

{ *** Definisi Type List *** }
type infotype : integer
type address  : pointer to ElmtList
type ElmtList : < info : infotype,
                  next : address >

{ Definisi list : }
{ List kosong : First(L) = dummy@ dan Last(L) = dummy@ }
{ Setiap elemen dengan address P dapat diacu Info(P), Next(P) }
{ Elemen dummy terletak pada last }

type List : < First : address,
              Last : address   >

{ Definisikan selektor yang tepat }

{ ***************** PRIMITIF-PRIMITIF LIST ***************** }

{ *** TEST LIST KOSONG *** }
function IsListEmpty (L : List) → boolean
{ Mengirim true jika list kosong: First(L) = dummy@ dan Last(L) = dummy@ }

{ *** PEMBUATAN LIST KOSONG *** }
procedure CreateList (output L : List)
{ I.S. Sembarang }
{ F.S. Terbentuk list L kosong, dengan satu elemen dummy }

{ *** Manajemen Memori *** }
function Alokasi (X : infotype) → address
{ Mengirimkan address hasil alokasi sebuah elemen }
{ Jika alokasi berhasil, maka address tidak Nil, dan misalnya menghasilkan P, maka
  Info(P) = X, Next(P) = Nil }
{ Jika alokasi gagal, mengirimkan Nil }
procedure Dealokasi (input/output P : address)
{ I.S. P terdefinisi }
{ F.S. P dikembalikan ke sistem }
{ Melakukan dealokasi/pengembalian address P }

{ *** PENCARIAN SEBUAH ELEMEN LIST *** }
function FSearch (L : List, P : address) → boolean
{ Mencari apakah ada elemen list yang beralamat P }
{ Mengirimkan true jika ada, false jika tidak ada }
```

```
function Search (L : List, X : infotype) → address
{ Mencari apakah ada elemen list dengan Info(P) = X }
{ Jika ada, mengirimkan address elemen tersebut }
{ Jika tidak ada, mengirimkan Nil }

{ ***************** PRIMITIF BERDASARKAN NILAI *****************}
{ *** PENAMBAHAN ELEMEN *** }
procedure InsVFirst (input/output L : List, input X : infotype)
{ I.S. L mungkin kosong }
{ F.S. Melakukan alokasi sebuah elemen dan menambahkan elemen pertama dengan nilai
       X jika alokasi berhasil }
procedure InsVLast (input/output L : List, input X : infotype)
{ I.S. L mungkin kosong }
{ F.S. Melakukan alokasi sebuah elemen dan menambahkan elemen list di sebelum
       elemen akhir (elemen sebelum elemen dummy) bernilai X
       jika alokasi berhasil. }
{      Jika alokasi gagal: I.S. = F.S. }

{ *** PENGHAPUSAN ELEMEN *** }
procedure DelVFirst (input/output L : List, output X : infotype)
{ I.S. List L tidak kosong  }
{ F.S. Elemen pertama list dihapus : nilai info disimpan pada X }
{      dan alamat elemen pertama didealokasi }
procedure DelVLast (input/output L : List, output X : infotype)
{ I.S. List tidak kosong }
{ F.S. Elemen sebelum dummy dihapus : nilai info disimpan pada X }
{      dan alamat elemen terakhir sebelum dummy di-dealokasi }

{ ***************** PRIMITIF BERDASARKAN ALAMAT ***************** }
{ *** PENAMBAHAN ELEMEN BERDASARKAN ALAMAT *** }
procedure InsertFirst (input/output L : List, input P : address)
{ I.S. Sembarang, P sudah dialokasi  }
{ F.S. Menambahkan elemen ber-address P sebagai elemen pertama }
procedure InsertAfter (input/output L : List, input P : address, input Prec :
address)
{ I.S. Prec pastilah elemen list dan bukan elemen terakhir, }
{      P sudah dialokasi  }
{ F.S. Insert P sebagai elemen sesudah elemen beralamat Prec }
procedure InsertLast (input/output L : List, input P : address)
{ I.S. Sembarang, P sudah dialokasi  }
{ F.S. P ditambahkan sebagai elemen terakhir yang baru, }
{      yaitu menjadi elemen sebelum dummy }

{ *** PENGHAPUSAN SEBUAH ELEMEN *** }
procedure DelFirst (input/output L : List, output P : address)
{ I.S. List tidak kosong }
{ F.S. P adalah alamat elemen pertama list sebelum penghapusan }
{      Elemen list berkurang satu (mungkin menjadi kosong) }
{ First element yg baru adalah suksesor elemen pertama yang lama }
procedure DelLast (input/output L : List, output P : address)
{ I.S. List tidak kosong }
{ F.S. P adalah alamat elemen terakhir sebelum dummy pada list sebelum
       penghapusan  }
{      Elemen list berkurang satu (mungkin menjadi kosong) }
procedure DelAfter (input/output L : List, output Pdel : address, input Prec :
address)
{ I.S. List tidak kosong. Prec adalah anggota list. }
{ F.S. Menghapus Next(Prec) : Pdel adalah alamat elemen list yang dihapus  }

{ ***************** PROSES SEMUA ELEMEN LIST ***************** }
procedure PrintInfo (input L : List)
{ I.S. List mungkin kosong }
{ F.S. Jika list tidak kosong, }
{ Semua info yg disimpan pada elemen list (kecuali dummy) diprint }
{ Jika list kosong, hanya menuliskan "list kosong" }
```

Bagian 2. ADT List Sirkuler

1. Buatlah ADT List Sirkuler yaitu list dengan next dari elemen terakhir menunjuk pada first list. List ini memiliki penunjuk tunggal yaitu First, diimplementasikan dengan dengan representasi fisik pointer, dalam bahasa C yang diberikan spesifikasi struktur datanya di bawah ini. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
   Primitif yang dibuat sama dengan modul P11. Bagian 1. Namun demikian, perhatikan perubahan-perubahan yang harus dilakukan pada bagian spesifikasi primitif untuk disesuaikan dengan struktur data yang harus dibuat.
2. Buatlah driver untuk memeriksa apakah semua primitif yang didefinisikan telah berjalan dengan baik.

```
{ ADT list sirkuler, berkait dengan representasi fisik pointer }
{ Representasi address dengan pointer }
{ infotype adalah integer }

{ *** Konstanta *** }
constant Nil : ...

{ *** Definisi Type List *** }
type infotype : integer
type address  : pointer to ElmtList
type ElmtList : < info : infotype,
                  next : address >

{ Definisi list : }
{ List kosong : First(L) = Nil }
{ Setiap elemen dengan address P dapat diacu Info(P), Next(P) }
{ Elemen terakhir list : jika addressnya Last, maka Next(Last) = First }

type List : < First : address >

{ Definisikan selektor yang tepat }

{ Primitif-primitif yang harus dibuat sesuaikan dengan primitif-primitif pada P-11.
Bagian 1. }
```

Bagian 3. ADT List dengan Double Pointer

1. Buatlah ADT List Double Pointer yaitu list dengan next dari elemen terakhir menunjuk pada first list. List ini memiliki penunjuk tunggal yaitu First, diimplementasikan dengan dengan representasi fisik pointer, dalam bahasa C yang diberikan spesifikasi struktur datanya di bawah ini. Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
   Primitif yang dibuat sama dengan modul P11. Bagian 1. Namun demikian, perhatikan perubahan-perubahan yang harus dilakukan pada bagian spesifikasi primitif untuk disesuaikan dengan struktur data yang harus dibuat.
2. Buatlah driver untuk memeriksa apakah semua primitif yang didefinisikan telah berjalan dengan baik.

```
{ ADT list dengan double pointer: prev dan next }
{ Dengan 2 penunjuk list: first dan last }
{ Representasi address dengan pointer }
{ infotype adalah integer }

{ *** Konstanta *** }
constant Nil : ...

{ *** Definisi Type List *** }
type infotype : integer
type address  : pointer to ElmtList
type ElmtList : < info : infotype,
                  prev : address,
                  next : address, >

{ Definisi list : }
{ List kosong : First(L) = Nil, Last(L) = Nil }
{ Setiap elemen dengan address P dapat diacu Info(P), Prev(P), Next(P) }

type List : < First : address,
              Last  : address >

{ Definisikan selektor yang tepat }

{ Primitif-primitif yang harus dibuat sesuaikan dengan primitif-primitif pada P-11.
Bagian 1. }
```

P-12. ADT STACK – REPRESENTASI DENGAN LIST LINIER

1. Buatlah ADT Stack yang direpresentasikan dengan list linier dengan representasi fisik pointer dalam bahasa C sesuai dengan spesifikasi yang diberikan di bawah ini.
2. Buatlah driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

```
{ MODUL STACK }
{ Deklarasi stack yang diimplementasi dengan list linier }
{ dengan representasi fisik pointer }
{ TOP adalah alamat elemen puncak }

constant Nil : ...   { Nil adalah stack dengan elemen kosong }

{ *** Definisi Type Stack *** }
type infotype : integer
type address  : pointer to ElmtStack
type ElmtStack : < info : infotype,
                   next : address   >
type Stack : < TOP : address >

{ TOP(S) = Nil adalah stack dengan elemen kosong }
{ Definisi stack dengan representasi berkait : }
{ Jika S adalah Stack maka akses elemen : }
   { InfoTop(S) untuk mengakses elemen TOP }
   { TOP(S) adalah alamat elemen TOP }
   { Info(P) untuk mengakses elemen info dengan alamat P }
   { Next(P) untuk mengakses elemen next dengan alamat P }

{ ********* Prototype ********* }

{ *** Konstruktor/Kreator *** }
procedure CreateEmpty (output S : Stack)
{ I.S. Sembarang }
{ F.S. Membuat sebuah stack S yang kosong berkapasitas MaxEl }
{ jadi indeksnya antara 1..MaxEl }
{ Ciri stack kosong : TOP bernilai Nil }

{ *** Prototype manajemen memori *** }
procedure Alokasi (output P : address, input X : infotype)
{ I.S. P Sembarang, X terdefinisi }
{ F.S. Alamat P dialokasi, jika berhasil maka Info(P) = X dan Next(P) = Nil }
{      P = Nil jika alokasi gagal }
procedure Dealokasi (input/output P : address)
{ I.S. P adalah hasil alokasi, P <> Nil }
{ F.S. Alamat P didealokasi, dikembalikan ke sistem }

{ ********* Predikat Untuk test keadaan KOLEKSI ********* }
function IsEmpty (S : Stack) → boolean
{ Mengirim true jika Stack kosong : lihat definisi di atas }
function IsFull (S : Stack) → boolean
{ Mengirim true jika tabel penampung nilai elemen stack penuh }

{ ********** Operator Dasar Stack ********* }
procedure Push (input/output S : Stack, input X : infotype)
{ Menambahkan X sebagai elemen Stack S. }
{ I.S. S mungkin kosong, tabel penampung elemen stack TIDAK penuh }
{ F.S. X menjadi TOP yang baru, jika alokasi elemen baru berhasil. }
{      Jika alokasi gagal, S tetap. }
procedure Pop (input/output S : Stack, input X : infotype);
{ Menghapus X dari Stack S. }
{ I.S. S tidak kosong }
{ F.S. X adalah nilai elemen TOP yang lama, elemen top yang lama didealokasi,
       TOP(S) = Next(TOP(S)). }
```

P-13. ADT QUEUE – REPRESENTASI DENGAN LIST LINIER

1. Buatlah ADT Queue yang direpresentasikan dengan list linier dengan representasi fisik pointer dalam bahasa C sesuai dengan spesifikasi yang diberikan di bawah ini.
2. Buatlah driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

```
{ Modul Queue }
{ Direpresentasikan dengan list linier, secara fisik dengan pointer }
{ Queue direpresentasikan sebagai list dengan first dan last }

constant Nil : ... { terdefinisi }

{ *** Definisi Type Queue *** }
type infotype  : integer
type address   : pointer to ElmtQueue
type ElmtQueue : < info : infotype,
                   next : address   >

{ Type Queue dengan ciri HEAD dan TAIL : }
type Queue : < HEAD : address,  { alamat penghapusan }
               TAIL : address   { alamat penambahan  } >

{ *** Prototype manajemen memori *** }
procedure Alokasi (output P : address, input X : infotype)
{ I.S. P sembarang, X terdefinisi }
{ F.S. Alamat P dialokasi, jika berhasil maka Info(P) = X dan Next(P) = Nil }
{      P = Nil jika alokasi gagal }
procedure Dealokasi (input/output P : address)
{ I.S. P adalah hasil alokasi, P <> Nil }
{ F.S. Alamat P didealokasi, dikembalikan ke sistem }

{ *** Predikat Pemeriksaan Kondisi Queue *** }
function IsEmpty (Q : Queue) → boolean
{ Mengirim true jika Q kosong: HEAD(Q) = Nil and TAIL(Q) = Nil }
function NBElmt (Q : Queue) → integer
{ Mengirimkan banyaknya elemen queue. Mengirimkan 0 jika Q kosong. }

{ *** Konstruktor *** }
procedure CreateEmpty (input/output Q : Queue)
{ I.S. sembarang }
{ F.S. Sebuah Q kosong terbentuk }

{ *** Primitif Add/Delete *** }
procedure Add (input/output Q : Queue, input X : infotype)
{ Proses : Mengalokasi X dan menambahkan X pada bagian TAIL dari Q jika alokasi
           berhasil; jika alokasi gagal Q tetap }
{ I.S. Q mungkin kosong }
{ F.S. X menjadi TAIL, TAIL "maju" }
procedure Del (input/output Q : Queue, output X : infotype)
{ Proses : Menghapus X pada bagian HEAD dari Q dan mendealokasi elemen HEAD }
{ I.S. Q tidak mungkin kosong }
{ F.S. X = nilai elemen HEAD pd I.S., HEAD "mundur" }
```

P-14. ADT PRIORITY QUEUE

1. Buatlah ADT Priority Queue yang direpresentasikan dengan list linier dengan representasi fisik pointer dalam bahasa C sesuai dengan spesifikasi yang diberikan di bawah ini.
2. Buatlah driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

```
{ ADT priority queue berkait dengan representasi fisik pointer  }
{ Representasi address dengan pointer }
{ infotype adalah integer }
{ Prioritas berdasarkan nilai elemen prio yang menunjukkan prioritas }
{ Queue terurut mengecil berdasarkan prioritas }

constant Nil : ... { terdefinisi }

{ *** Definisi Type Queue *** }
type infotype : integer
type address  : pointer to ElmtQueue
type ElmtQ    : < info : infotype,
                  prio : integer,
                  next : address   >
type Queue : < HEAD : address >

{ Definisi priority queue : }
{ Queue kosong : Head(Q) = Nil }
{ Setiap elemen dengan address P dapat diacu info(P), Next(P) }
{ Elemen terakhir list : jika addressnya Last, maka Next(Last)=Nil }

{ *** Prototype manajemen memori *** }
procedure Alokasi (output P : address, input X : infotype)
{ I.S. P sembarang, X terdefinisi }
{ F.S. Alamat P dialokasi, jika berhasil maka Info(P) = X dan Next(P) = Nil }
{      P = Nil jika alokasi gagal }
procedure Dealokasi (input/output P : address)
{ I.S. P adalah hasil alokasi, P <> Nil }
{ F.S. Alamat P didealokasi, dikembalikan ke sistem }

{ *** Predikat Pemeriksaan Kondisi Queue *** }
function IsEmpty (Q : Queue) → boolean
{ Mengirim true jika Q kosong: HEAD(Q) = Nil dan TAIL(Q) = Nil }
function NBElmt (Q : Queue) → integer
{ Mengirimkan banyaknya elemen queue. Mengirimkan 0 jika Q kosong. }

{ *** Konstruktor *** }
procedure CreateEmpty (output Q : Queue)
{ I.S. sembarang }
{ F.S. Sebuah Q kosong terbentuk }

{ *** Primitif Add/Delete *** }
procedure Add (input/output Q : Queue, input X : infotype, input Pr : integer)
{ Proses : Mengalokasi X dan menambahkan X pada bagian TAIL dari Q jika alokasi
           berhasil dengan memperhatikan prioritas; jika alokasi gagal Q tetap }
{ I.S. Q mungkin kosong, X terdefinisi }
{ F.S. X menjadi elemen Q sesuai prioritas Pr,
       Q tetap terurut mengecil sesuai prioritas }
procedure Del (input/output Q : Queue, input X : infotype, input Pr : integer)
{ Proses : Menghapus X pada bagian HEAD dari Q dan mendealokasi elemen HEAD,
           X berisi elemen dengan prioritas tertinggi }
{ I.S. Q tidak kosong }
{ F.S. X = nilai elemen HEAD dan Pr = nilai elemen prioritas HEAD pd I.S.,
       HEAD "maju" }
```

P-15. LIST REKURSIF

1. Buatlah ADT List Rekursif yang direpresentasikan dengan list linier dengan representasi fisik pointer dalam bahasa C sesuai dengan spesifikasi yang diberikan di bawah ini.
2. Buatlah driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

```
KAMUS
{ List rekursif direpresentasi dengan pointer }
constant Nil  : ... { terdefinisi }

type infotype : integer { terdefinisi }
type address  : pointer to ElmtList
type ElmtList : < info : infotype,
                  next : address >
type List : address
{ Deklarasi  nama untuk variabel kerja }
{      L : List }
{      P : address  }
{ Maka penulisan First(L) menjadi L }
{                P↑.info menjadi Info(P); P↑.next menjadi Next(P) }

{ *** Manajemen Memori *** }
function Alokasi (X : infotype) → address
{ Mengirimkan address hasil alokasi sebuah elemen }
{ Jika alokasi berhasil, maka address tidak Nil, dan misalnya menghasilkan P,
  maka Info(P) = X, Next(P) = Nil }
{ Jika alokasi gagal, mengirimkan Nil }
procedure Dealokasi (input/output P : address)
{ I.S. P terdefinisi }
{ F.S. P dikembalikan ke sistem }
{ Melakukan dealokasi/pengembalian address P }

{ *** Primitif-primitif yang harus direalisasikan *** }

{ *** Selektor *** }
function FirstElmt (L : List) → infotype
{ Mengirimkan elemen pertama sebuah list L yang tidak kosong }
function Tail (L : List) → List
{ Mengirimkan list L yang tidak kosong tanpa elemen pertamanya }

{ *** Konstruktor *** }
function Konso (e : infotype, L : List) → List
{ Mengirimkan list L dengan tambahan e sebagai elemen pertamanya }
{ e harus dialokasi terlebih dahulu }
{ Jika alokasi e gagal, mengirimkan L }
function Kons• (L : List, e : infotype) → List
{ Mengirimkan list L dengan tambahan e sebagai elemen terakhir }
{ e harus dialokasi terlebih dahulu }
{ Jika alokasi e gagal, mengirimkan L }

{ *** Predikat *** }
function IsListEmpty (L : List) → boolean
{ Mengirimkan true jika list kosong, false jika tidak kosong }
{ Mungkin yang dikirimkan adalah sebuah list kosong }

{ *** Operasi Lain *** }
function Copy (L : List) → List
{ Mengirimkan salinan list L (menjadi list baru) }
{ Jika ada alokasi gagal, mengirimkan L }
function Concat (L1, L2 : List) → List
{ Mengirimkan salinan hasil konkatenasi list L1 dan L2 (menjadi list baru) }
{ Jika ada alokasi gagal, menghasilkan Nil }
```

```
procedure  Printlist (input L : List)
{ I.S. L terdefinisi. }
{ F.S. Setiap elemen list dicetak. }
function NBElmtList (L : List) → integer
{ Mengirimkan banyaknya elemen list L, Nol jika L kosong }
function Search (L : List, X : infotype) → boolean
{ Mengirim true jika X adalah anggota list, false jika tidak }
```

P-16. ADT POHON BINER

1. Buatlah ADT Pohon Biner dalam Bahasa C sesuai spesifikasi di bawah ini. Untuk ADT list yang diperlukan dalam beberapa primitif, gunakan ADT List Rekursif pada modul P-15. Jika terjadi konflik penamaan, misalnya nama type address, maka lakukan perubahan yang diperlukan.
2. Buatlah driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

```
{ ADT Pohon Biner }

{ Modul lain yang digunakan : }
USE LIST LINIER

{ *** Definisi Type Pohon Biner *** }
type node : < Info : infotype,
              Left : address,
              Right : address >
type BinTree : address

{ *** Definisi Type List of Node *** }
type ListOfNode : List

{ *** PROTOTYPE *** }

{ *** Selektor *** }
function Akar (P : BinTree) → infotype
{ Mengirimkan nilai Akar pohon biner P }
function Left (P : BinTree) → BinTree
{ Mengirimkan Anak Kiri pohon biner P }
function Right (P : BinTree) → BinTree
{ Mengirimkan Anak Kanan pohon biner P }

{ *** Konstruktor *** }
function Tree (Akar : infotype, L : BinTree, R : BinTree) → BinTree
{ Menghasilkan sebuah pohon biner dari A, L, dan R, jika alokasi berhasil }
{ Menghasilkan pohon kosong (Nil) jika alokasi gagal }
procedure MakeTree (input Akar : infotype, input L : BinTree,
                    input R : BinTree, output P : BinTree)
{ I.S. Sembarang }
{ F.S. Menghasilkan sebuah pohon P }
{ Menghasilkan sebuah pohon biner P dari A, L, dan R, jika alokasi berhasil }
{ Menghasilkan pohon P yang kosong (Nil) jika alokasi gagal }
procedure BuildTree (output P : BinTree)
{ Membentuk sebuah pohon biner P dari pita karakter yang dibaca }
{ I.S. Pita berisi "konstanta" pohon dalam bentuk prefix.
       Memori pasti cukup, alokasi pasti berhasil. }
{ F.S. P dibentuk dari ekspresi dalam pita }

{ *** Predikat-Predikat Penting *** }
function IsTreeEmpty (P : BinTree) → boolean
{ Mengirimkan true jika P adalah pohon biner kosong }
function IsOneElmt (P : BinTree) → boolean
{ Mengirimkan true jika P adalah pohon biner tidak kosong dan hanya memiliki 1
elemen }
function IsUnerLeft (P : BinTree) → boolean
{ Mengirimkan true jika pohon biner tidak kosong P adalah pohon unerleft: hanya
mempunyai subpohon kiri }
function IsUnerRight (P : BinTree) → boolean
{ Mengirimkan true jika pohon biner tidak kosong P adalah pohon unerright: hanya
mempunyai subpohon kanan}
function IsBiner (P : BinTree) → boolean
{ Mengirimkan true jika pohon biner tidak kosong P adalah pohon biner: mempunyai
subpohon kiri dan subpohon kanan}
```

```
{ *** Traversal *** }
procedure PrintPreorder (input P : BinTree)
{ I.S. P terdefinisi }
{ F.S. Semua simpul P sudah dicetak secara preorder: akar, pohon kiri, dan pohon
kanan. Setiap pohon ditandai dengan tanda kurung buka dan kurung tutup (). }
procedure PrintInorder (input P : BinTree)
{ I.S. P terdefinisi }
{ F.S. Semua simpul P sudah dicetak secara inorder: pohon kiri, akar, dan pohon
kanan. Setiap pohon ditandai dengan tanda kurung buka dan kurung tutup ().  }
procedure PrintPostorder (input P : BinTree)
{ I.S. P terdefinisi }
{ F.S. Semua simpul P sudah dicetak secara postorder: pohon kiri, pohon kanan, dan
akar. Setiap pohon ditandai dengan tanda kurung buka dan kurung tutup ().  }
procedure PrintTree (input P : BinTree, input h : integer)
{ I.S. P terdefinisi, h adalah jarak indentasi }
{ F.S. Semua simpul P sudah ditulis dengan indentasi }

{ *** Searching *** }
function Search (P : BinTree, X : infotype) → boolean
{ Mengirimkan true jika ada node dari P yang bernilai X }

{ *** Fungsi-Fungsi Lain *** }
function NbElmt (P : BinTree) → integer
{ Mengirimkan banyaknya elemen (node) pohon biner P }
function NbDaun (P : BinTree) → integer
{ Mengirimkan banyaknya daun (node) pohon biner P }
function IsSkewLeft (P : BinTree) → boolean
{ Mengirimkan true jika P adalah pohon condong kiri }
function IsSkewRight (P : BinTree) → boolean
{ Mengirimkan true jika P adalah pohon condong kiri }
function Level (P : BinTree, X : infotype) → integer
{ Mengirimkan level dari node X yang merupakan salah satu simpul dari pohon biner
P. Akar(P) level-nya adalah 1. Pohon P tidak kosong. }

{ *** Operasi lain *** }
procedure AddDaunTerkiri (input/output P : BinTree, input X : infotype)
{ I.S. P boleh kosong }
{ F.S. P bertambah simpulnya, dengan X sebagai simpul daun terkiri }
procedure AddDaun (input/Output P : BinTree, input X, Y : infotype,
                   input Kiri : boolean)
{ I.S. P tidak kosong, X adalah salah satu daun Pohon Biner P }
{ F.S. P bertambah simpulnya, dengan Y sebagai anak kiri X (jika Kiri = true), atau
       sebagai anak Kanan X (jika Kiri = false) }
procedure DelDaunTerkiri (input/output P : BinTree, output X : infotype)
{ I.S. P tidak kosong }
{ F.S. P dihapus daun terkirinya, dan didealokasi, dengan X adalah info yang semula
       disimpan pada daun terkiri yang dihapus }
procedure DelDaun (input/output P : BinTree, input X : infotype)
{ I.S. P tidak kosong, X adalah salah satu daun }
{ F.S. Semua daun bernilai X dihapus dari P }
function MakeListDaun (P : BinTree) → ListOfNode
{ Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{ Jika P bukan pohon kosong : menghasilkan list yang elemennya adalah semua daun
pohon P, jika semua alokasi list berhasil. Menghasilkan list kosong jika ada
alokasi yang gagal. }
function MakeListPreorder (P : BinTree) → ListOfNode
{ Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{ Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua elemen
pohon P dengan urutan Preorder, jika semua alokasi berhasil. Menghasilkan list
kosong jika ada alokasi yang gagal. }
function MakeListLevel (P : BinTree, N : integer) → ListOfNode
{ Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{ Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua elemen
pohon P yang levelnya=N, jika semua alokasi berhasil. Menghasilkan list kosong jika
ada alokasi yang gagal. }
```

```
{ *** Balanced tree *** }
function BuildBalanceTree (n : integer) → BinTree
{ Menghasilkan sebuah balanced tree dengan n node, nilai setiap node dibaca }

{ *** Binary Search Tree *** }
function BSearch (P : BinTree, X : infotype) → boolean
{ Mengirimkan true jika ada node dari P yang bernilai X }
function InsSearch (P : BinTree, X : infotype) → BinTree
{ Menghasilkan sebuah pohon Binary Search Tree P dengan tambahan simpul X. Belum
ada simpul P yang bernilai X. }
procedure DelBtree (input/output P : BinTree, input X : infotype)
{ I.S. Pohon P tidak kosong }
{ F.S. Nilai X yang dihapus pasti ada }
{ Sebuah node dengan nilai X dihapus }
```

P-17. STUDI KASUS 1 : POLINOM

Bagian 1. Representasi dengan Tabel Kontigu

1. Buatlah studi kasus Polinom (lihat diktat Struktur Data) yang direpresentasikan dengan tabel kontigu dalam bentuk ADT Polinom sesuai spesifikasi yang diberikan di bawah ini.
2. Buatlah driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

```
{ ADT Polinom }
{ Representasi dengan Tabel Kontigu }

{ *** Konstanta *** }
constant Nmax : integer = 100 { Derajat tertinggi polinom yang diproses }

{ *** Definisi Type Polinom *** }
type polinom : < Degree : integer,
                 TabSuku : array [0..Nmax] of integer >

{ *** Primitif operasi terhadap polinom yang dibutuhkan untuk proses *** }

procedure InitPol (output P : polinom)
{ Membentuk sebuah polinom kosong }
{ I.S. : sembarang. }
{ F.S. Polinom P yang kosong dibentuk }

procedure AdjustDegree (input/output P : polinom)
{ Melakukan adjustment terhadap Degree. }
{ Diaktifkan jika akibat suatu operasi, derajat polinom hasil berubah. }
{ I.S. P terdefinisi, mungkin kosong. }
{      P.Degree belum tentu merupakan derajat polinom }
{ F.S. P terdefinisi,
       P.Degree berisi derajat polinom berdasarkan keadaan P.TabSuku }
{      Derajat hanya mungkin "turun". Search i = P.Degree terbesar, dengan
       TabSuku[i] ≠ 0 }

procedure CreatePol (output P : polinom)
{ I.S. sembarang }
{ F.S. Polinom P terdefinisi derajat dan koefisien-koefisiennya jika tidak
       "kosong" }
{ Mengisi polinom P dengan membaca dari alat masukan, pemasukan harga diakhiri
  dengan mark }

procedure TulisPol (input P : polinom)
{ I.S. P terdefinisi dan mungkin kosong }
{ F.S. Menulis polinom P ke layar dengan format menurun }

procedure AddPol (input P1, P2 : polinom, output P3 : polinom)
{ Menjumlahkan P1 + P2 dan menyimpan hasilnya di P3, P3 ≠ P1 dan P3 ≠ P2 }
{ I.S. P1, P2 terdefinisi dan mungkin kosong }
{ F.S. P3 = P1+P2, P3 adalah polinom baru }

procedure SubPol (input P1, P2 : polinom, output P3 : polinom)
{ Mengurangkan P1 - P2 dan menyimpan hasilnya di P3, P3 ≠ P1 dan P3≠ P2 }
{ I.S. P1, P2 terdefinisi dan mungkin kosong }
{ F.S. P3 = P1-P2, P3 adalah polinom baru }

procedure DerivPol (input P : polinom, output P1 : polinom)
{ Membuat turunan P dan menyimpan hasilnya di P1, P1 ≠ P }
{ I.S. P terdefinisi, mungkin kosong }
{ F.S. P1 adalah turunan P }
```

Bagian 2. Representasi Berkait dengan Pointer

1. Buatlah studi kasus Polinom (lihat diktat Struktur Data) yang direpresentasikan secara berkait dengan pointer dalam bentuk ADT Polinom sesuai spesifikasi yang diberikan di bawah ini.
2. Buatlah driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

```
{ ADT Polinom }
{ Representasi Berkait dengan Pointer }

constant Nil : address = ... { untuk address tidak terdefinisi }

{ Struktur data untuk representasi polinom }
type address : pointer to Suku
type Suku : < Degree : integer,
              Coefficient : integer,
              Next : address >
type polinom : address

{ *** Manajemen Memori *** }
procedure AllocSuku (output Pt : address)
{ Alokasi sebuah suku }
{ I.S. Sembarang }
{ F.S. Pt dialokasi. Jika alokasi gagal Pt = Nil. }
procedure DeAllocSuku (input/output Pt : address)
{ Dealokasi sebuah suku }
{ I.S. Pt terdefinisi }
{ F.S. Pt dikembalikan ke memori }
procedure InitListPol (output P : polinom)
{ Membuat polinom kosong P }
{ I.S. sembarang; F.S. polinom kosong P terbentuk}

{ *** Operasi Penambahan Polinom *** }
procedure InsertLast (input Pt : address, input/output P : polinom,
                      input/output Last : address)
{ I.S. P mungkin kosong, Pt adalah address dari sebuah suku yang akan "dicopy"
       nilainya }
{ F.S. <Degree, Coeff> dari Pt di-copy, dan di-insert sebagai elemen terakhir yang
       baru, yaitu Last }
{ Insert Pt sesudah elemen terakhir Polinom P }

{ *** Primitif Operasi Polinom *** }
procedure CreateListPol (output P1 : polinom)
{ I.S. Sembarang. }
{ F.S. Polinom P terdefinisi derajat dan koefisien-koefisiennya jika tidak
       "kosong" }
{ Mengisi list polinom P1 }
procedure TulisListPol (input P : polinom)
{ I.S. P terdefinisi dan mungkin kosong }
{ F.S. Menulis polinom P sesuai dengan spesifikasi }
procedure AddListPol (input P1, P2 : polinom, output P3 : polinom)
{ I.S. P1, P2 terdefinisi dan mungkin kosong }
{ F.S. P3 = P1+P2, P3 polinom baru }
{ Menjumlahkan P1 + P2 dan menyimpan hasilnya di P3, P3 ≠ P1 dan P3 ≠P2 }
procedure SubListPol (input P1, P2 : polinom, output P3 : polinom)
{ I.S. P1, P2 terdefinisi dan mungkin kosong }
{ F.S. P3 = P1-P2, P3 polinom baru }
{ Mengurangkan P1 - P2 dan menyimpan hasilnya di P3, P3 ≠ P1 dan P3 ≠ P2 }
procedure DerivListPol (input P : polinom, output P1 : polinom)
{ I.S. P terdefinisi, mungkin kosong }
{ F.S. P1 adalah turunan P, polinom baru }
{ Membuat turunan P dan menyimpan hasilnya di P1, P1 ≠ P }
```

P-18. STUDI KASUS 2 : KEMUNCULAN HURUF DAN POSISI PADA PITA KARAKTER

Bagian 1. Representasi dengan Struktur Data Matriks

Buatlah program untuk studi kasus Kemunculan Huruf dan Posisi pada Pita Karakter (lihat diktat Struktur Data) yang direpresentasikan dalam struktur data matriks berdasarkan kamus yang diberikan di bawah (lihat juga diktat "Struktur Data").

```
KAMUS
    constant Nmax : integer = 100 { maksimum isi pita yang diproses adalah
                                    100 }
    MatMuncul : matrix ['a'..'z', 1..Nmax] of boolean
    { MatMuncul(C,I) adalah kemunculan karakter C sebagai urutan ke-I
      dalam pita }
    Count : integer [1..Nmax] { jumlah kemunculan sebuah huruf }
    Posisi : integer              { posisi karakter pada pita }
    JumlahKar : integer       { banyaknya karakter pada pita }
    C : character ['a'..'z']  { indeks traversal matriks }
    i, j : integer [1..NMax]  { kemunculan karakter, juga indeks traversal matriks }
```

Bagian 2. Representasi dengan List Kontigu dan List Berkait

Buatlah program untuk studi kasus Kemunculan Huruf dan Posisi pada Pita Karakter (lihat diktat Struktur Data) dengan list huruf direpresentasikan sebagai list kontigu dan list posisi direpresentasikan sebagai list berkait dengan representasi pointer berdasarkan kamus yang diberikan di bawah (lihat juga diktat "Struktur Data").

```
KAMUS
    type address : pointer to ElmtList
    type ElmtList : < Posisi : integer [1..100], Next : address >
    HeadList : array ['a'..'z'] of address { tabel alamat elemen pertama }

    P : address
    Count : integer [1..100] { Kemunculan setiap huruf }
    JumlahKar : integer      { Jumlah karakter pada pita }
    C : character ['a'..'z'] { Untuk traversal karakter abjad }

    procedure InsertLast (input/output First, P : address)
    { Insert P sesudah elemen terakhir list First }
    { I.S. First terdefinisi, boleh kosong; P sudah dialokasi dan informasinya
           terdefinisi }
    { F.S. P menjadi elemen terakhir list First }
```

P-19. STUDI KASUS 3 : PENGELOLAAN MEMORI

Bagian 1. Representasi Memori dengan Tabel Kontigu

1. Buatlah studi kasus Pengelolaan Memori (lihat diktat Struktur Data) dengan representasi memori dengan tabel kontigu sesuai spesifikasi di bawah ini dalam bentuk ADT.
2. Buatlah juga driver untuk memeriksa apakah semua primitif yang dibuat telah berjalan dengan baik.

```
{ Modul Pengelolaan Memori }
{ Representasi memori dengan tabel kontigu }

{ *** Konstanta *** }
constant NB : integer = 100

{ *** Tabel Global Status Memori *** }
STATMEM : array [1..NB] of boolean { true jika ISI, false jika KOSONG }

{ *** Primitif-Primitif Pengelolaan Memori *** }

procedure InitMemK
{ I.S. Sembarang }
{ F.S. Semua blok memori dinyatakan KOSONG }
{ Proses : Semua status blok dengan indeks [1..NB] dijadikan KOSONG dengan
           traversal blok [1..NB].
           Proses sekuensial tanpa penanganan kasus kosong (NB ≥ 0). }

procedure AlokBlokKF (input X : integer, output IAw : integer)
{ Strategi pengalokasian adalah First Fit }
{ I.S. X adalah banyaknya blok yang diminta untuk dialokasi, yaitu dijadikan ISI }
{ F.S. IAw akan berisi indeks blok kosong pertama pada tabel Status Memori
       jika ada X blok kontigu berstatus KOSONG yang masih bisa dialokasi,
       kemudian memutakhirkan status pemakaian memori.
       IAw bernilai 0 jika tidak ada blok kontigu berukuran minimal X. }

procedure AlokBlokKB (input X : integer, output IAw : integer)
{ Strategi pengalokasian adalah Best Fit }
{ I.S. X adalah banyaknya blok yang diminta untuk dialokasi, yaitu dijadikan ISI }
{ F.S. IAw akan berisi indeks blok kosong pertama pada tabel Status Memori
       jika ada X blok kontigu berstatus KOSONG yang masih bisa dialokasi,
       kemudian memutakhirkan status pemakaian memori.
       IAw bernilai 0 jika tidak ada blok kontigu berukuran minimal X }

procedure DealokBlokK (input X, IAw : integer)
{ I.S. X adalah ukuran zone, bilangan positif dan
      IAw adalah alamat blok awal zone tersebut,
      dengan IAw ∈ [1..NB-X], blok dengan indeks IAw s.d. IAw+X-1 pasti berstatus
      ISI. }
{ F.S. Tabel status memori dengan indeks blok IAw..IAw+X-1 menjadi KOSONG }
{ Proses : Sebuah zone berukuran X dan bebawal pada blok IAw di-dealokasi
           (statusnya dijadikan KOSONG) }

procedure GarbageCollectionK
{ I.S. Sembarang }
{ F.S. Terbentuk sebuah zone kosong di "kiri" }
```

Bagian 2. Representasi Zone Kosong dengan List Berkait

1. Buatlah studi kasus Pengelolaan Memori (lihat diktat Struktur Data) dengan representasi zone kosong pada memori dengan list berkait sesuai spesifikasi di bawah ini dalam bentuk ADT.
2. Buatlah juga driver untuk memeriksa apakah semua primitif yang dibuat telah berjalan dengan baik.

```
{ Modul Pengelolaan Memori }
{ Representasi zone kosong dengan list berkait dengan pointer }

{ *** Definisi List Memori *** }
type address : pointer to ZB
type ZB : < Aw : integer,    { indeks awal zone kontigu kosong }
            Nb : integer,    { banyaknya blok zone kontigu kosong }
            Next : address >

{ *** Variabel global : Address zone kosong pertama *** }
FIRSTZB : address

{ Elemen list terurut menurut Aw }

{ Fungsi akses untuk penulisan algoritma secara lojik }
{ Jika P adalah sebuah address, maka dituliskan :
     - Next(P) adalah address elemen list sesudah elemen beralamat P
     - Aw(P) adalah nilai blok kosong pertama pada sebuah elemen list beralamat
       P yang mewakili sebuah zone kontigu
     - Nb(P) adalah ukuran blok sebuah zone kosong yang disimpan informasinya
       dalam elemen list beralamat P
     - Allocate(P) adalah prosedur utk melakukan alokasi sebuah ZB dengan alamat
       P, P tidak mungkinsama dengan Nil (alokasi selalu berhasil)
     - Deallocate(P) adalah prosedur utk men-dealokasi sebuah alamat P }

procedure InitMemB
{ I.S. Sembarang }
{ F.S. Semua blok memori dinyatakan KOSONG }
{ Proses : Create list kosong, buat elemen baru, insertFirst }
procedure AlokBlokBF (input X : integer, output IAw : integer)
{ I.S. X adalah banyaknya blok yang diminta untuk dialokasi, yaitu dijadikan ISI. }
{ F.S. IAw adalah alamat awal sebuah zone bebas dengan X buah blok kontigu kosong,
       zone bebas paling "kiri", jika ada.
       Iaw = 0 jika tidak ada zone kontigu berukuran IAw. }
{ Strategi pengalokasian adalah First Fit }
procedure AlokBlokBB (input X : integer, output IAw : integer)
{ I.S. X adalah banyaknya blok yang diminta untuk dialokasi, yaitu dijadikan ISI. }
{ F.S. IAw adalah alamat awal sebuah zone bebas dengan X buah blok kontigu kosong,
       zone bebas minimal ukurannya jika ada.
       IAw = 0 jika tidak ada zone kontigu berukuran IAw. }
{ Strategi pengalokasian adalah Best Fit, skema search dengan boolean. }
procedure DeAlokBlokB (input X, IAw : integer )
{ I.S. X adalah ukuran zone, bilangan positif dan
       IAw adalah alamat blok awal zone tersebut, dengan IAw ∈ [1..NB-X],
       Blok dengan indeks IAw s.d. IAw+X-1 pasti berstatus ISI. }
{ F.S. Tabel status memori dengan indeks blok IAw..IAw+X-1 menjadi KOSONG }
{ Proses : Sebuah zone berukuran X dan bebawal pada blok IAw didealokasi (statusnya
           dijadikan KOSONG) }
procedure GarbageCollectionB
{ I.S. Sembarang }
{ F.S. List zone kosong terdiri atas 1 elemen yang menyatakan seluruh zone kosong
       berada di "kiri" }
```

P-20. STUDI KASUS 4 : MULTILIST

Bagian 1. Alternatif I

1. Buatlah studi kasus Daftar Pegawai dan Anak (lihat Diktat Struktur Data) dengan menggunakan representasi struktur data alternatif I yaitu list Pegawai dengan elemennya mengandung list Anak. Implementasikan dalam bentuk ADT dengan spesifikasi di bawah ini. Untuk tanggal lahir anak digunakan ADT TANGGAL. Jika diperlukan, tambahkan primitif-primitif lain.
2. Buatlah juga driver untuk memeriksa setiap primitif apakah telah berjalan dengan baik.

```
{ Modul Daftar Pegawai dan Anak }
{ Representasi Alternatif I : List Pegawai mengandung List Anak }
{ Representasi list berkait dengan pointer }

{ *** ADT lain yang digunakan *** }
USE ADT_TANGGAL

{ *** Type List Pegawai dan List Anak *** }
type AdrPeg  : pointer to Pegawai
type AdrAnak : pointer to Anak
type Pegawai : < NIP       : integer,
                 Nama      : string,
                 Jabatan   : string,
                 GajiPokok : real,
                 FirstAnak : AdrAnak,
                 NextPeg   : AdrPeg   >
type Anak    : < Nama     : string,
                 TglLahir : TANGGAL,
                 NextAnak : AdrAnak  >

type ListPeg : AdrPeg

{ *** Primitif-Primitif Pemrosesan Daftar Pegawai dan Anak *** }

procedure ListPegLengkap (input FirstPeg : ListPeg)
{ I.S. FirstPeg terdefinisi, mungkin kosong }
{ F.S. List FirstPeg ditulis informasinya beserta informasi semua anaknya
       jika tidak kosong }
{ Menuliskan daftar pegawai, untuk setiap pegawai dilist anaknya jika ada. }
{ Menuliskan "List kosong, tidak ada pegawai" jika kosong. }

procedure ListTunjAnak (input FirstPeg : ListPeg)
{ I.S. FirstPeg terdefinisi, mungkin kosong }
{ F.S. Untuk setiap pegawai, setiap anak yang berumur < 18 tahun ditulis ke layar.
       Jika pegawai tidak mempunyai anak, menuliskan "Pegawai tidak mempunyai
       anak" }

function Umur (TglLahir : TANGGAL) → integer
{ Mengirimkan umur jika yang dihitung dari tanggal hari ini dari sistem
  dibandingkan dengan TglLahir yang diberikan }

procedure ListPegNonKB (input FirstPeg : ListPeg)
{ Membuat daftar pegawai yang mempunyai lebih dari 3 anak }
{ I.S. List FirstPeg terdefinisi, mungkin kosong }
{ F.S. Semua pegawai yang anaknya lebih dari 3 orang ditulis informasinya }
```

```
procedure OrTuAnak (input FirstPeg : ListPeg, input NamaAnak : string)
{ I.S. List Pegawai terdefinisi }
{ F.S. Jika ada anak yang bernama sesuai dengan NamaAnak, nama Pegawai ditulis.
       Jika tidak ada NamaAnak, tidak menuliskan apa-apa. }
{ Cari anak dengan nama NamaAnak, tuliskan nama orangtua dari anak yang namanya =
  NamaAnak }

procedure AddAnak (input/output FirstPeg : ListPeg, input NIPPeg : string,
                   input NamaAnak : string, input TglLahirAnak : TANGGAL)
{ Mendaftar seorang anak yang baru lahir, insert selalu pada awal list }
{ I.S. List Pegawai terdefinisi }
{ F.S. Jika pegawai dengan NIP = NIPeg ada, alokasi anak,
       insert seorang anak sebagai elemen pertama list anak }
```

Bagian 2. Alternatif II

1. Buatlah studi kasus Daftar Pegawai dan Anak (lihat Diktat Struktur Data) dengan menggunakan representasi struktur data alternatif II yaitu list Pegawai dan list Anak yang dikelola secara terpisah. Implementasikan dalam bentuk ADT dengan spesifikasi di bawah ini (dengan primitif yang sama dengan modul P-19. Bagian 1). Untuk tanggal lahir anak digunakan ADT TANGGAL. Jika diperlukan, tambahkan primitif-primitif lain.
2. Buatlah juga driver untuk memeriksa setiap primitif apakah telah berjalan dengan baik.

```
{ Modul Daftar Pegawai dan Anak }
{ Representasi Alternatif II : List Pegawai dan List Anak dikelola terpisah }
{ Representasi list berkait dengan pointer }

{ *** ADT lain yang digunakan *** }
USE ADT_TANGGAL

{ *** Type List Pegawai dan List Anak *** }
type AdrPeg  : pointer to Pegawai
type AdrAnak : pointer to Anak
type Pegawai : < NIP       : integer,
                 Nama      : string,
                 Jabatan   : string,
                 GajiPokok : real,
                 FirstAnak : AdrAnak,
                 NextPeg   : AdrPeg   >
type Anak    : < Nama     : string,
                 TglLahir : TANGGAL,
                 NextAnak : AdrAnak,
                 Father   : AdrPeg    >

type ListPeg : AdrPeg
type ListAnak : AdrAnak

{ *** Primitif-Primitif Pemrosesan Daftar Pegawai dan Anak *** }

{ Sama dengan modul P-19. Bagian 1. }
```