Panduan Mudah
Pemrograman Fortran 90/95

Imam Fachruddin

Kata Pengantar

Bermula dari "catatan tak tertulis" kami sebagai pengguna Fortran 90/95 selama beberapa tahun, kami menulis buku ini agar "catatan tak tertulis" itu menjadi tersusun lebih baik dan tak mudah hilang atau terlupakan. Dengan buku ini juga kami ingin berbagi dengan pembaca mengenai pemrograman komputer menggunakan bahasa Fortran 90/95. Meski tidak mencakup semua materi tentang Fortran 90/95, namun buku ini berisi hal-hal dasar yang kami anggap penting, perlu, dan memadai sebagai bekal untuk menyusun program komputer, yang tidak sebatas hanya program sederhana, dalam bahasa Fortran 90/95. Tentu saja itu semua menurut pertimbangan berdasarkan pengalaman kami. Kami juga berusaha untuk membuat isi buku ini mudah dipahami. Dalam hal ini, untungnya, bahasa pemrograman Fortran 90/95 itu sendiri memang relatif mudah dipelajari, bahkan secara mandiri.

Buku ini diharapkan dapat membawa banyak manfaat, apalagi mengingat betapa sedikitnya atau sulitnya menemukan buku mengenai Fortran, khususnya Fortran 90/95, di Indonesia. Padahal, bahasa Fortran, sesuai namanya yang merupakan singkatan dari *Formula Translation*, merupakan bahasa yang umum digunakan dalam bidang sains dan teknik. Dengan demikian, kemampuan menyusun program komputer dalam bahasa Fortran, khususnya Fortran 90/95, sangat diperlukan oleh mereka yang menekuni bidang sains dan teknik di Indonesia.

Depok, Desember 2016 — Imam Fachruddin

Daftar Isi

Kata Pengantar iii

1 Pendahuluan 1

2 Program Fortran 90/95 5

3 Data 11
- 3.1 Tipe Data . . . 11
- 3.2 Data Skalar dan Array . . . 14
- 3.3 Variabel dan Konstanta . . . 16

4 Operator 23

5 Percabangan 29
- 5.1 Perintah `GOTO` . . . 29
- 5.2 Pernyataan dan Konstruksi `IF` . . . 30
- 5.3 Konstruksi `SELECT CASE` . . . 33

6 Pengulangan 37
- 6.1 Konstruksi `DO` . . . 37
- 6.2 Konstruksi `DO` Bersusun . . . 38
- 6.3 Perintah `EXIT` dan `CYCLE` . . . 39
- 6.4 Konstruksi `DO` tanpa Variabel Pengulangan . . . 41

7 Subprogram 43
- 7.1 Argumen Subprogram . . . 44
- 7.2 Subrutin . . . 44
- 7.3 Fungsi . . . 46
- 7.4 Subprogram Rekursif . . . 47

8 Masukan & Luaran 49
- 8.1 Proses *I/O* Terformat dan Tak Terformat . . . 49
- 8.2 Perintah `READ` dan `WRITE` . . . 50
- 8.3 Unit . . . 50
- 8.4 Format . . . 53
  - 8.4.1 Format Bebas . . . 53

8.4.2 Format yang Ditentukan 56
8.4.2.1 Penempatan Deskriptor Edit 56
8.4.2.2 Deskriptor Edit 58

9 Topik-Topik Lanjut **65**
9.1 Data `REAL` dan `COMPLEX` Berpresisi Ganda 65
9.2 Alokasi Dinamis Variabel Array 66
9.3 Argumen *Dummy* Pilihan 68
9.4 Atribut `SAVE` untuk Variabel Lokal 70
9.5 Subprogram Eksternal 71
9.6 Modul 74
9.7 Subprogram sebagai Argumen 76

Lampiran **79**

A Program Komputer serta Cara Kompilasi dan Penautan Program **81**

B Beberapa Subprogram Intrinsik Fortran 90/95 **85**
B.1 Subrutin Intrinsik 85
B.2 Fungsi Intrinsik 87
B.2.1 Fungsi Matematis 87
B.2.2 Fungsi untuk Operasi Array 88
B.2.3 Fungsi Lain 89

C Contoh-Contoh Program Fortran 90/95 **91**
C.1 Mempertukarkan Nilai Dua Variabel 91
C.2 Mempertukarkan Dua Baris dan Dua Kolom Sebuah Matriks 92
C.3 Mencari Nilai Terkecil dan Nilai Terbesar 93
C.4 Mengurutkan Nilai 94
C.5 Menghitung Hasil Penjumlahan dan Hasil Perkalian 97
C.6 Menghitung Perkalian Matriks 98
C.7 Interpolasi Lagrange 99
C.8 Fungsi Faktorial dan Fungsi Faktorial Ganda 100
C.9 Ekspansi Binomial 101
C.10 Koefisien Clebsch-Gordan 102
C.11 Program yang Memanfaatkan Atribut `SAVE` untuk Variabel Lokal 105
C.12 Program dengan Subprogram Eksternal 106
C.13 Program dengan Modul 119
C.14 Program yang Menggunakan Subprogram sebagai Argumen 128

Daftar Pustaka **137**

Indeks **139**

Bab 1

Pendahuluan

Sebagai sedikit gambaran mengenai bahasa pemrograman Fortran, secara khusus Fortran sangat sesuai untuk aplikasi perhitungan di bidang sains dan teknik. Itulah kekuatan utama Fortran. Menurut sejarah nama Fortran itu sendiri merupakan kependekan dari *Formula Translation*. Sampai sekarang Fortran masih dipakai secara luas di bidang sains dan teknik, meskipun bukan satu-satunya, karena beberapa bahasa pemrograman lain juga telah dikembangkan orang.

Bahasa pemrograman Fortran mengalami pengembangan dari sejak kelahirannya di pertengahan abad ke-20. Fortran 66 dikembangkan dalam dekade 60, disusul Fortran 77 yang lahir dalam dekade 70, dan kemudian dalam dekade 90 Fortran 90 diluncurkan. Fortran 90 lebih modern dari Fortran 77 dan memiliki beberapa hal baru yang tidak dimiliki Fortran 77. Fortran 90 juga menjadi dasar bagi pengembangan Fortran Berkinerja Tinggi (*High Performance Fortran*, disingkat *HPF*). Hanya beberapa tahun dalam dekade yang sama setelah Fortran 90 lahir, Fortran 95 dikembangkan dan berisi tidak lain hanya revisi minor terhadap Fortran 90. Karena itulah kedua versi Fortran tersebut sering dituliskan bersama sebagai Fortran 90/95.

Sebagai sebuah panduan buku ini diharapkan dapat digunakan juga oleh mereka yang bahkan baru belajar menyusun program komputer dan memilih Fortran 90/95 sebagai bahasa pemrograman. Karena itu, kami berikan juga suatu penjelasan secara umum mengenai program komputer serta cara mengkompilasinya (*compile*) dan menautkannya (*link*) menjadi sebuah file yang bisa dijalankan (*executable file*). Penjelasan ini dapat dilihat di Lampiran A. Mereka yang sudah mengenal pemrograman komputer dalam bahasa lain dan kini ingin mengenal dan menggunakan Fortran 90/95 dapat melewatkan Lampiran A.

Bab 2 berisi penjelasan mengenai ketentuan-ketentuan umum program komputer yang ditulis dalam bahasa Fortran 90/95, disingkat program Fortran 90/95. Secara khusus bab ini hanya menunjukkan program utama sebagai salah satu unit program Fortran 90/95. Namun, penjelasan mengenainya juga meliputi hal-hal yang berlaku umum untuk unit program Fortran 90/95 yang lain, yaitu subprogram eksternal dan modul. Subprogram eksternal dan modul itu sendiri dijelaskan secara khusus dalam Bab 9 sebagai topik lanjut.

Sebuah program komputer berisi perintah-perintah kepada komputer untuk mela-

kukan suatu tindakan tertentu terhadap suatu data. Data, dengan demikian, menjadi obyek. Bab 3 membahas tentang data, tipe dan jenisnya, ekspresinya, dan sebagainya, demikian pula, mengenai wadah data, yaitu variabel dan konstanta. Dicontohkan pada bab itu, antara lain, bagaimana menyimpan data ke dalam variabel dan konstanta.

Untuk melakukan suatu tindakan terhadap data diperlukan operator. Jenis operasi operator bermacam-macam, sebagai contoh, ada operasi numerik dan ada juga operasi logika. Fortran 90/95 menyediakan sejumlah operator, disebut operator intrinsik. Penjelasan mengenai operator intrinsik tersebut disampaikan dalam Bab 4.

Sampai di sini kita masih belum masuk ke bagian isi program, tempat pembuat program memberikan perintah-perintah kepada komputer untuk melakukan suatu tugas. Ada beberapa hal dasar yang lazim terdapat dalam isi program komputer. Hal-hal tersebut dibahas dalam bab-bab selajutnya sebagai berikut:

1. Percabangan:
   Di sini komputer dihadapkan pada pilihan untuk meneruskan langkah ke mana atau menjalankan perintah yang mana. Dalam program kita harus membuat suatu kondisi, yang berdasarkan itu pilihan ditentukan. Percabangan dibahas di Bab 5.

2. Pengulangan:
   Beberapa pekerjaan perlu dilakukan secara berulang-ulang. Pengulangan bisa dilakukan sebanyak yang ditentukan atau sampai suatu kondisi terpenuhi. Bab 6 membahas mengenai pengulangan.

3. Subprogram:
   Pekerjaan yang ingin diselesaikan mungkin suatu pekerjaan yang besar. Di dalamnya ada pekerjaan-pekerjaan kecil atau subpekerjaan. Dalam hal ini, sebuah program dapat disusun agar terdiri dari beberapa subprogram, tiap-tiap subprogram mengerjakan satu subpekerjaan tertentu. Di Bab 7 subprogram dibahas.

   Fortran 90/95 juga menyediakan sejumlah subprogram intrinsik siap pakai, contoh, untuk menghitung fungsi matematis, seperti fungsi trigonometri. Kami sampaikan di Lampiran B beberapa subprogram intrinsik, yang kami anggap banyak dipakai. Daftar lengkap subprogram intrinsik beserta penjelasannya dapat ditemukan di sumber-sumber referensi Fortran 90/95.

4. Masukan & luaran (*input & output (I/O)*):
   Untuk menyelesaikan pekerjaan mungkin saja komputer membutuhkan masukan data atau parameter dari pengguna. Disamping itu, komputer tentu perlu menunjukkan hasil kerjanya, yaitu memberikan luaran, misalkan ke layar. Pembahasan mengenai masukan & luaran disampaikan di Bab 8.

Apa yang disampaikan dalam buku ini, yaitu ketentuan-ketentuan umum program Fortran 90/95, data, variabel, konstanta, operator, percabangan, pengulangan, subprogram, masukan & luaran, meski tidak secara panjang lebar diharapkan cukup untuk membantu pembaca menyusun sebuah program sungguhan, bukan hanya program mainan. Program itu bisa saja merupakan program pembantu bagi suatu program besar

atau sebuah program tunggal yang berdiri sendiri. Sebagai tambahan kami sampaikan juga beberapa topik lanjut, yang dihimpun dalam Bab 9. Topik-topik tersebut antara lain presisi ganda (*double precision*), yang lazim dituntut dari suatu perhitungan ilmiah, subprogram eksternal, dan modul.

Untuk dapat menyusun sebuah program komputer yang besar dan kompleks diperlukan latihan. Karena itu, mulailah dengan membuat program yang sederhana atau program mainan, lalu bertahap membuat program yang lebih kompleks. Beberapa contoh program Fortran 90/95 kami berikan di Lampiran C, diawali dengan yang sederhana.

Buku ini tidak membahas semua fitur Fortran 90/95, melainkan hanya yang menurut pengalaman dan perkiraan kami sebagai pengguna banyak dipakai atau dibutuhkan. Juga, jika ada beberapa cara yang dapat orang pilih dalam menuliskan suatu perintah atau suatu pernyataan Fortran 90/95, maka demi kemudahan kami hanya menunjukkan salah satu saja. Pada saatnya nanti, ketika pembaca telah mencapai suatu ketrampilan pemrograman tertentu dan buku ini dirasa tidak dapat memberikan panduan lebih lanjut, ambillah referensi lanjutan mengenai pemrograman komputer dan Fortran 90/95.

Terakhir, kami sampaikan sedikit catatan mengenai cara penulisan yang kami pakai dalam buku ini.

- Perintah, pernyataan, program, dan penggalan program Fortran 90/95 kami tuliskan menggunakan font typewriter. Contoh:

  ```
  PROGRAM fortranku
  END FUNCTION faktorial
  INTEGER
  ```

- Pada contoh-contoh perintah dan pernyataan Fortran 90/95 kami cantumkan tanda "[" dan "]" hanya untuk menunjukkan bahwa yang ada di antara keduanya harus atau dapat diisi oleh penulis program sesuai keperluan. Tanda "[" dan "]" itu sendiri tidak dicantumkan dalam program. Contoh:

  ```
  PROGRAM [nama program]
  (/ [elemen data array] /)
  IF ([syarat]) [pekerjaan]
  ```

- Dalam menuliskan nilai / bilangan riil kami tetap gunakan tanda titik "." sebagai tanda desimal, bukan tanda koma ",", meskipun nilai / bilangan riil itu dituliskan dalam teks yang bukan merupakan contoh perintah atau pernyataan Fortran 90/95.

Bab 2

Program Fortran 90/95

Sebuah program komputer yang ditulis dalam bahasa Fortran 90/95, disingkat program Fortran 90/95, dapat terdiri dari beberapa unit program, yaitu satu program utama, beberapa subprogram eksternal, dan beberapa modul, yang dapat berisi beberapa subprogram modul. Program utama, subprogram eksternal, dan subprogram modul itu sendiri masing-masing dapat berisi beberapa subprogram internal. Sebuah program bisa saja tidak memiliki subprogram eksternal dan / atau modul, tapi harus memiliki setidaknya program utama. Program sederhana dapat terdiri dari hanya program utama tanpa subprogram internal, subprogram eksternal, dan modul. Namun, program yang besar dan kompleks sebaiknya dibuat terdiri dari satu program utama dengan beberapa subprogram internal dan juga melibatkan subprogram eksternal serta modul. Pada bab ini kami sampaikan hal-hal yang umum mengenai program Fortran 90/95. Demi kemudahan kami tampilkan contoh program yang terdiri dari hanya program utama, tanpa maupun dengan subprogram internal. Penjelasan khusus mengenai subprogram eksternal dan modul kami sampaikan di Bab 9.

Struktur program utama Fortran 90/95 atau program Fortran 90/95 yang terdiri dari hanya program utama adalah sebagai berikut:

```
PROGRAM [nama program]
[bagian spesifikasi]
[isi program utama]
END PROGRAM [nama program]
```

Program utama diawali oleh pernyataan `PROGRAM` dan diakhiri oleh pernyataan `END PROGRAM`. Pernyataan `PROGRAM` dan `END PROGRAM` diikuti oleh nama program, yang diberikan pembuat program. Nama program tidak boleh berisi ruang kosong (spasi). Jika nama program terdiri dari dua atau lebih kata, maka semua kata itu dapat disambung atau saling dipisahkan oleh suatu tanda sambung, contoh:

```
PROGRAM alirankalor
...
END PROGRAM alirankalor
```

atau

```
PROGRAM aliran_kalor
...
END PROGRAM aliran_kalor
```

Nama-nama yang berasal dari bahasa Fortran 90/95 tentu sudah dikenal oleh *compiler* Fortran 90/95. Namun, nama-nama yang dibuat pembuat program, seperti nama variabel dan konstanta, tidak dikenal oleh *compiler*. Karena itu, nama-nama variabel dan konstanta harus diperkenalkan atau dideklarasikan terlebih dahulu. Hal ini dilakukan di bagian spesifikasi suatu unit / subunit program (program utama, subprogram eksternal, modul, subprogram modul, subprogram internal). Wajarlah jika bagian spesifikasi ditempatkan di awal unit / subunit program tersebut.

Bagian spesifikasi diikuti oleh isi program utama. Dalam isi program utama inilah pembuat program menuliskan perintah-perintah kepada komputer untuk menyelesaikan suatu pekerjaan. Pekerjaan itu dapat merupakan pekerjaan kecil sederhana, maupun pekerjaan besar. Jika pekerjaan itu besar, mungkin perlu dipecah menjadi beberapa subpekerjaan, tiap subpekerjaan diselesaikan oleh suatu subprogram (internal atau eksternal). Sebaliknya, jika pekerjaan itu kecil sederhana, tidak perlu dipecah menjadi beberapa subpekerjaan, tidak perlu dibuat subprogram. Penjelasan mengenai subprogram diberikan di Bab 7, yang berlaku baik untuk subprogram internal maupun subprogram eksternal.

Struktur program utama Fortran 90/95 tanpa subprogram internal seperti berikut:

```
PROGRAM [nama program]
[bagian spesifikasi]
[bagian perintah & pernyataan program utama]
END PROGRAM [nama program]
```

Adapun struktur program utama Fortran 90/95 yang memiliki subprogram internal digambarkan sebagai berikut:

```
PROGRAM [nama program]
[bagian spesifikasi]
[bagian perintah & pernyataan program utama]
CONTAINS
[subprogram internal 1]
[subprogram internal 2]
...
END PROGRAM [nama program]
```

Subprogram internal ditempatkan setelah pernyataan `CONTAINS` di dalam program utama, subprogram eksternal, maupun subprogram modul. Pada contoh di atas pernyataan

`CONTAINS` tersebut memisahkan subprogram internal dari bagian perintah & pernyataan program utama di atasnya.

Mari kita lihat satu contoh sederhana program Fortran 90/95, yang diberi nama fortranku, dan dari contoh itu kita pelajari beberapa sifat dan ketentuan umum program Fortran 90/95. Program fortranku tersebut tidak berisi subprogram internal. Untuk membantu penjelasan ditambahkan nomor baris di sisi paling kiri. Tapi, catat bahwa pencantuman nomor baris ini BUKAN bagian dari penulisan program Fortran 90/95.

```
 1 PROGRAM fortranku
 2 !last edited: August 23, 2005 by Imam
 3
 4 IMPLICIT NONE
 5 REAL :: a,b,c,d
 6
 7 !Beri nilai a dan b
 8 a=3.0
 9 b=0.7
10
11 !Hitung nilai c dan d
12    c=(a+b)*(a+b+1.0)  ! c ditentukan oleh a dan b
13  d=1.0-c+0.5*c*c      ! d bergantung pada c
14
15 a= 10.0 &
16   +b &
17       -20.0*c-d
18
19 STOP
20
21 END PROGRAM fortranku
```

Catatan:

- Pada program fortranku di atas baris ke-4 sampai 5 merupakan bagian spesifikasi dan baris ke-7 sampai 19 berisi perintah & pernyataan program utama.
- Program Fortran 90/95 tidak mengenal huruf besar atau kecil (*case insensitive*). Program dapat ditulis dengan menggunakan huruf besar saja atau huruf kecil saja atau campuran huruf besar dan kecil. Pada contoh di atas digunakan campuran huruf besar dan kecil. Nama-nama yang berasal dari Fortran 90/95 ditulis dalam huruf besar dan yang selain itu dalam huruf kecil.
- Baris kosong diperkenankan. Dengan demikian, pembuat program bebas menempatkan satu, dua, atau lebih baris kosong untuk memudahkan dirinya sendiri maupun orang lain dalam membaca dan memahami program. Pada contoh di atas terdapat baris kosong di baris ke-3, 6, 10, 14, 18, dan 20.

- Baris-baris program tidak harus bersifat rata kiri atau rata kanan atau rata kiri-kanan. Di baris ke-12, 16, dan 17 kita lihat pernyataan ditulis sedikit bergeser ke kanan.

- Di baris ke-2, 7, 11, 12, dan 13 terdapat kalimat atau keterangan, yang diawali oleh tanda seru "!". Ini disebut komentar. Komentar diabaikan saat program dikompilasi. Namun, komentar perlu diberikan untuk memudahkan orang memahami program. Komentar dapat ditulis dalam satu baris sendiri, seperti pada contoh di atas baris ke-2, 7, 11. Jika komentar ditulis dalam beberapa baris, maka tiap baris harus diawali oleh tanda "!". Komentar dapat juga ditempatkan di baris yang sama dengan baris suatu perintah atau pernyataan dalam program, contoh, baris ke-12 dan 13 di atas. Ingat bahwa semua yang berada di sebelah kanan tanda "!" adalah bagian dari komentar. Karena itu, jangan tempatkan perintah atau pernyataan di sebelah kanan komentar, karena akan dianggap sebagai bagian dari komentar, sehingga akan diabaikan ketika program dikompilasi.

- Huruf awal nama variabel dan konstanta dapat menentukan tipe data yang disimpan dalam variabel dan konstanta itu. Karena itu, dalam memberi nama variabel dan konstanta pembuat program harus cermat dan hati-hati agar huruf awalnya sesuai dengan tipe data yang akan disimpan dalam variabel dan konstanta tersebut. Bagi sebagian orang ini sedikit mengurangi keleluasaan dalam memberi nama variabel dan konstanta. Untungnya, ada cara untuk "terbebas" dari aturan tersebut. Caranya adalah dengan memberi pernyataan `IMPLICIT NONE` di awal bagian spesifikasi, seperti dapat dilihat di baris ke-4 pada program di atas. Dengan demikian, pembuat program dapat memberi nama variabel dan konstanta secara leluasa, tentu saja asalkan berbeda dari nama yang berasal dari Fortran 90/95.

- Pernyataan dalam program dapat dituliskan lebih dari satu baris, maksimum 40 baris, contoh, baris ke-15 sampai 17 pada program di atas. Tanda "&" di akhir baris menyatakan bahwa pernyataan di baris itu bersambung ke baris berikutnya, contoh, baris ke-15 dan 16 pada program di atas. Satu baris pernyataan dapat berisi maksimum 132 karakter, termasuk spasi, karena spasi juga merupakan karakter.

- Eksekusi program (*program execution*) adalah proses berjalannya suatu program atau proses dijalankannya perintah-perintah dalam suatu program oleh komputer. Untuk menghentikan eksekusi program di suatu baris tertentu digunakan perintah `STOP` di baris itu. Perintah `STOP` bisa diberikan di program utama maupun subprogram. Begitu menemui perintah `STOP` komputer berhenti menjalankan program. Pada contoh program di atas perintah `STOP` diberikan di baris ke-19 sebagai perintah terakhir program. Sebetulnya, dalam hal ini tanpa perintah `STOP` pun eksekusi program akan berhenti, karena memang sudah sampai di akhir program. Namun, pemberian perintah `STOP` di akhir program diperkenankan, disamping untuk menegaskan bahwa di situ program berakhir.

Kini kita lihat contoh program utama yang memiliki subprogram internal. Contoh program berikut, sesungguhnya, adalah program yang sama dengan yang sebelumnya di atas, namun ditulis sedemikian sehingga berisi subprogram internal.

```
PROGRAM fortranku

IMPLICIT NONE
REAL :: a,b,c,d


a=3.0
b=0.7
CALL subfortranku(a,b,c)
d=funcfortranku(c)
a= 10.0+b-20.0*c-d

STOP

CONTAINS

SUBROUTINE subfortranku(x,y,z)

IMPLICIT NONE
REAL, INTENT(IN) :: x,y
REAL, INTENT(OUT) :: z
REAL :: s

s=x+y
z=s*(s+1.0)

RETURN

END SUBROUTINE subfortranku

FUNCTION funcfortranku(x) RESULT(z)

IMPLICIT NONE
REAL,INTENT(IN) :: x
REAL :: z

z=1.0-x+0.5*x*x

RETURN

END FUNCTION funcfortranku
```

```
END PROGRAM fortranku
```

Catatan:

- Program di atas memiliki dua subprogram internal, keduanya ditempatkan setelah pernyataan `CONTAINS`. Subprogram pertama berupa sebuah subrutin, yang diberi nama subfortranku (diawali pernyataan `SUBROUTINE subfortranku` dan diakhiri pernyataan `END SUBROUTINE subfortranku`). Subprogram kedua berupa sebuah fungsi, yang diberi nama funcfortranku (diawali pernyataan `FUNCTION funcfortranku` dan diakhiri pernyataan `END FUNCTION funcfortranku`). Penjelasan mengenai subrutin dan fungsi diberikan di Bab 7.

- Perhatikan bahwa perintah `STOP` sebagai perintah terakhir program diberikan sebelum pernyataan `CONTAINS`, bukan sebelum pernyataan `END PROGRAM`.

- Variabel dan konstanta yang dideklarasikan di suatu subprogram berlaku secara lokal hanya di subprogram tersebut. Variabel x, y, z, dan s di subrutin subfortranku merupakan variabel lokal, demikian pula variabel x, z di fungsi funcfortranku. Variabel x dan z di subrutin subfortranku tidak akan tertukar dengan variabel x dan z di fungsi funcfortranku, karena subrutin subfortranku dan fungsi funcfortranku masing-masing merupakan subunit program utama yang saling terpisah.

- Variabel dan konstanta yang dideklarasikan di bagian spesifikasi program utama berlaku secara global di seluruh bagian program, bukan hanya di program utama dan subprogram internalnya, melainkan juga di unit / subunit lain yang terlibat. Variabel a, b, c, d merupakan variabel global pada program di atas. Namun, apabila, misalkan, variabel a tersebut dideklarasikan juga di suatu subprogram internal maupun subprogram eksternal, maka di dalam subprogram itu berlaku deklarasi lokal untuk variabel a, yaitu variabel a bersifat sesuai deklarasinya di dalam subprogram itu, bukan seperti yang dideklarasikan di bagian spesifikasi program utama.

- Perintah `RETURN` di mana pun ditemui di dalam sebuah subprogram membuat eksekusi program keluar dari subprogram itu dan kembali ke program utama atau subprogram yang memanggil subprogram itu.

Nama variabel dan konstanta tidak harus pendek berupa satu, dua, tiga huruf, namun diperkenankan panjang. Bahkan, sebaiknya variabel dan konstanta diberi nama yang jelas maknanya, walaupun agak panjang. Itu akan sangat membantu orang, termasuk si pembuat program itu sendiri, untuk membaca dan memahami program tersebut.

Bab 3

Data

Sebuah program komputer tidak lain adalah sekumpulan perintah kepada komputer untuk melakukan suatu tindakan terhadap data: menyimpan data ke variabel, menggabungkan beberapa data, mengganti data, membaca data, memindahkan data, menulis data ke layar, dan lain sebagainya. Data tersebut disimpan dalam wadah, yang tidak lain adalah variabel dan konstanta.

3.1 Tipe Data

Fortran 90/95 menyediakan lima tipe data, disebut tipe data intrinsik. Jika diperlukan, pembuat program juga dapat membuat sendiri suatu tipe data baru. Tipe data ini disebut tipe data turunan, karena dibuat dari kombinasi beberapa tipe data intrinsik. Namun, lima tipe data intrinsik yang disediakan Fortran 90/95 sesungguhnya cukup untuk memenuhi banyak sekali keperluan. Di sini hanya akan disampaikan lima tipe data intrinsik tersebut.

Tipe data intrinsik dalam Fortran 90/95 yaitu `REAL`, `INTEGER`, `COMPLEX`, `CHARACTER`, dan `LOGICAL`. Penjelasan masing-masing tipe data tersebut sebagai berikut:

- `REAL` adalah tipe data untuk nilai bilangan riil, bilangan umum. Data `REAL` meliputi nilai baik bilangan bulat maupun yang mengandung bilangan pecahan desimal ($< |1.0|$). Contoh data `REAL` dan ekspresinya yaitu:

  ```
  -129.234
  -0.00321
  0.0
  1.09
  34.287
  ```

  Data `REAL` juga dapat diekspresikan dalam bentuk eksponensial, contoh, nilai 1.6×10^{-19} ditulis sebagai:

  ```
  1.6E-19
  ```

Bilangan 0 setelah tanda desimal boleh tidak dicantumkan, namun tanda desimal "." harus dicantumkan, contoh:

```
2.
3.E-8
```

- `INTEGER` adalah tipe data untuk nilai khusus bilangan bulat. Contoh data `INTEGER` yaitu:

```
-7892213
-300
0
10
123456
```

Perhatikan bahwa ekspresi data `INTEGER` tidak memiliki tanda desimal "." dan apalagi bilangan pecahan desimal. Sebagai contoh, ekspresi data berikut:

```
100.
100.0
```

tidak menunjukkan bahwa data itu bertipe `INTEGER`, meskipun nilai bilangan 100. dan 100.0 itu bulat.

- `COMPLEX` adalah tipe data untuk nilai khusus bilangan kompleks. Bilangan kompleks memiliki dua komponen, yaitu komponen riil dan komponen imajiner. Baik nilai komponen riil maupun imajiner kedua-duanya bertipe data `REAL`. Jadi, sebuah data `COMPLEX` merupakan sepasang data `REAL`. Contoh data `COMPLEX` yaitu:

```
(2.0,4.5)
(-2.1,0.3)
(0.0,0.0)
(3.0,0.0)
(0.0,1.0)
```

Bilangan di sebelah kiri dan kanan tanda koma "," masing-masing adalah komponen riil dan komponen imajiner. Kedua komponen harus ditempatkan di antara sepasang tanda kurung.

- `CHARACTER` adalah tipe data untuk ekspresi huruf, bilangan, simbol; singkatnya ekspresi karakter. Data `CHARACTER` diberikan di antara atau dibatasi oleh sepasang tanda petik tunggal atau sepasang tanda petik ganda, tapi bukan campuran tanda petik tunggal dan ganda. Contoh data `CHARACTER` yaitu:

```
"a"
"Matahari bersinar terang."
'QED'
"Alamat"
'2 + 3 = 5'
```

Jika suatu data `CHARACTER` mengandung tanda petik tunggal, maka sebagai pembatas dapat digunakan sepasang tanda petik ganda, contoh:

```
"ini adalah huruf 'a' kecil"
"tanda ' disebut tanda petik tunggal"
```

Sebaliknya, jika suatu data `CHARACTER` mengandung tanda petik ganda, maka sebagai pembatas dapat digunakan sepasang tanda petik tunggal, contoh:

```
'ini adalah huruf "A" besar'
'tanda " disebut tanda petik ganda'
```

Jumlah karakter di antara pasangan tanda petik tunggal atau tanda petik ganda menyatakan panjang data `CHARACTER` tersebut. Contoh, panjang data `CHARACTER` 'berlibur ke pantai' adalah 18. Perhatikan bahwa spasi juga merupakan karakter, sehingga turut dihitung. Sampai di sini ada sedikit masalah untuk menuliskan spasi dalam buku ini sebagai data `CHARACTER`, yaitu apabila terdapat beberapa spasi secara berurutan, maka secara visual jumlahnya agak sulit diketahui. Karena itu, mulai dari sini dalam menuliskan suatu data `CHARACTER` kami gunakan lambang "~" hanya untuk menunjukkan bahwa di situ ada satu spasi. Contoh:

```
'berlibur~ke~pantai'
'S~E~L~A~M~A~T~~~U~L~A~N~G~~~T~A~H~U~N'
```

- `LOGICAL` adalah tipe data untuk nilai logika suatu hal / pernyataan. Nilai itu hanya dua, yaitu benar dan salah. Namun, tidak berarti bahwa data `LOGICAL` hanya ada dua, karena nilai logika benar dan salah dapat dinyatakan dalam berbagai bentuk. Contoh data `LOGICAL` yaitu:

```
.TRUE.
2 < 3
2 - 1 > 0
.FALSE.
10 : 2 < 5
```

Pada contoh di atas tiga data pertama bernilai logika benar dan dua sisanya salah.

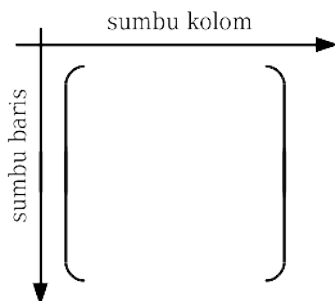
3.2 Data Skalar dan Array

Ada data tunggal, yaitu terdiri dari hanya satu data. Data seperti ini disebut data skalar. Ada pula data yang berisi sekelompok data yang tipenya sama. Data seperti ini disebut data array. Satu contoh data array yaitu data yang berisi bilangan bulat dari 0 sampai 10. Sebuah matriks juga merupakan data array, yang berisi sekelompok data bertipe sama, yang disusun dalam 2 dimensi, yaitu dalam baris dan kolom.

Data-data yang terkandung dalam sebuah data array disebut elemen data array dan disusun secara berurutan dalam ruang berdimensi tertentu. Jumlah dimensi tidak dibatasi maksimum tiga, melainkan dapat lebih dari tiga. Jumlah dimensi menunjukkan rank data array tersebut, contoh, sebuah matriks merupakan data array rank 2. Secara umum, dengan demikian, data skalar dan data array merupakan tensor. Data skalar adalah tensor rank 0, sedangkan data array merupakan tensor rank lebih dari 0.

Mari kita ambil sebuah matriks berukuran 2×3 sebagai satu contoh data array. Matriks itu memiliki 2 baris dan 3 kolom, sehingga jumlah elemennya adalah $2 \times 3 = 6$. Matriks tersebut merupakan contoh data array:

- rank 2 atau berdimensi 2, sumbu dimensi pertama adalah sumbu baris dan yang kedua adalah sumbu kolom,

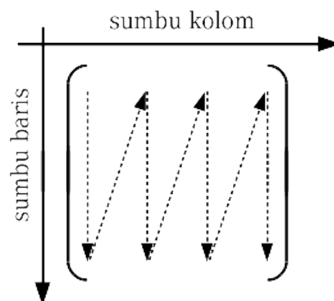


- memiliki *shape* atau bentuk (2,3),
- berukuran 6.

Berikutnya, sangat perlu mengetahui seperti apa elemen data array disimpan secara berurutan menurut Fortran 90/95, karena ini menyangkut kecepatan eksekusi program. Jika elemen data array dibaca atau diberikan secara berurutan, maka program berjalan lebih cepat daripada jika tidak secara berurutan. Sebagai contoh, perhatikan matriks 2×3 berikut:

$$\begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix}.$$

Urutan elemen matriks di atas menurut Fortran 90/95 adalah a, d, b, e, c, f. Dengan kata lain, elemen data diurutkan terlebih dahulu sepanjang sumbu dimensi pertama (baris), kemudian sepanjang sumbu dimensi kedua (kolom), seperti digambarkan sebagai berikut:



Untuk data array 3 dimensi urutan elemen-elemennya mula-mula sepanjang sumbu dimensi pertama, lalu sepanjang sumbu kedua, terakhir sepanjang sumbu ketiga.

Data array rank 1 dapat mudah dibuat dengan menggunakan satu perintah pembentuk array, yaitu `(/ [elemen data array] /)`. Sebagai contoh, data array yang berisi bilangan bulat dari 0 sampai 10 dapat dibuat dengan perintah:

```
(/ 0,1,2,3,4,5,6,7,8,9,10 /)
```

Data array rank lebih dari 1 tidak dapat dibuat dengan perintah di atas, namun bisa dengan menggunakan salah satu fungsi intrinsik Fortran 90/95, yaitu `RESHAPE`, yang dapat mengubah bentuk data array, termasuk tentu saja mengubahnya dari rank 1 menjadi rank lebih dari 1. Namun, `RESHAPE` tidak dibahas di sini. Cara lain adalah dengan menentukan elemen data array satu per satu atau per bagian. Namun, cara ini hanya dapat digunakan untuk mengisikan data array ke variabel, bukan konstanta (variabel dan konstanta dijelaskan pada subbab setelah ini).

Kini perhatikan bahwa data array yang berisi bilangan bulat dari 0 sampai 10 dapat juga dibentuk dengan cara berikut:

```
(/ (i, i=0,10) /)
```

Pada perintah di atas, i merupakan variabel bertipe `INTEGER`. Perintah itu menyatakan bahwa elemen data array sama dengan bilangan bulat i, dengan i = 0 sampai 10. Cara-cara berikut juga membentuk data array yang sama:

```
(/ (i, i=0,5), 6,7,8,9,10 /)
(/ 0,1,2,3,4, (i, i=5,10) /)
(/ (i, i=0,5), (i, i=6,10) /)
```

Satu contoh lagi, data array rank 1 yang berisi bilangan riil -0.8, -0.4, 0.0, 0.4, 0.8, 1.2 dapat dibentuk dengan cara, antara lain:

```
(/ -0.8,-0.4,0.0,0.4,0.8,1.2 /)
(/ (0.4*i, i=-2,3) /)
```

Pernyataan `[variabel]=[nilai1],[nilai2]` pada contoh-contoh perintah pembentuk array di atas menyatakan bahwa variabel, yang dalam hal ini harus bertipe `INTEGER`, memiliki nilai yang berjalan dari nilai1 sampai nilai2, dengan langkah sebesar 1. Jika besar langkah yang dikehendaki bukan 1, maka nilai langkah diberikan secara eksplisit, yaitu `[variabel]=[nilai1],[nilai2],[langkah]`. Sebagai contoh:

- `i=-2,6,2` berarti i berjalan dari -2 sampai 6 dengan langkah 2, yaitu i = -2, 0, 2, 4, 6,
- `i=12,-9,-3` berarti i berjalan dari 12 sampai -9 dengan langkah -3, yaitu i = 12, 9, 6, 3, 0, -3, -6, -9.

3.3 Variabel dan Konstanta

Variabel dan konstanta dibuat untuk menyimpan data. Data yang disimpan dalam variabel dapat diubah, sedangkan yang disimpan dalam konstanta tetap. Tipe variabel dan konstanta harus sesuai dengan tipe data yang disimpannya, begitu juga jenisnya, skalar atau array. Ada satu lagi jenis variabel, yaitu pointer. Jenis pointer tidak dimiliki oleh konstanta. Variabel pointer tidak dibahas di sini.

Sebelum digunakan, variabel dan konstanta harus dideklarasikan dulu di awal program. Kita akan selalu menggunakan pernyataan `IMPLICIT NONE` dalam deklarasi variabel dan konstanta. Dengan demikian, kita lebih bebas dalam memberi nama variabel dan konstanta. Kalaupun dalam contoh-contoh di sini pernyataan `IMPLICIT NONE` tidak terlihat, dianggap bahwa pernyataan itu sudah diberikan di awal bagian spesifikasi.

Berikut ini adalah contoh deklarasi variabel skalar:

```
INTEGER :: i,j
REAL :: massa, kelajuan
COMPLEX :: z
LOGICAL :: bs
CHARACTER(9) :: nip, nik
```

Catatan:

- Deklarasi diawali oleh tipe variabel sesuai tipe data.
- Pada contoh di atas, i dan j adalah variabel skalar bertipe `INTEGER`, massa dan kelajuan bertipe `REAL`, z bertipe `COMPLEX`, bs bertipe `LOGICAL`, nip dan nik bertipe `CHARACTER`.
- Variabel `CHARACTER` harus diberi keterangan yang jelas, seberapa panjang data `CHARACTER` yang akan disimpannya. Pada contoh di atas, variabel nip dan nik masing-masing dapat menyimpan data `CHARACTER` yang panjangnya maksimum 9 karakter.

Contoh deklarasi variabel array adalah sebagai berikut:

```
INTEGER, DIMENSION(3,5) :: gigi, gaga
REAL, DIMENSION(10) :: panjang, lebar
COMPLEX, DIMENSION(3,5,4) :: realita
LOGICAL, DIMENSION(1,2,1,4) :: flag0
CHARACTER(80), DIMENSION(60) :: halaman
```

Catatan:

- Deklarasi variabel array sama dengan deklarasi variabel skalar, namun berisi tambahan atribut `DIMENSION`. Atribut `DIMENSION` menentukan rank atau jumlah dimensi, bentuk, dan ukuran variabel array itu. Pada contoh di atas, variabel realita berdimensi 3, bentuknya (3,5,4), ukurannya 3 x 5 x 4 = 60.

Contoh deklarasi konstanta skalar dapat dilihat sebagai berikut:

```
INTEGER, PARAMETER :: kilo=1000, mega=1000*kilo
REAL, PARAMETER :: gravitasi=9.8, kecil=1.0e-4
COMPLEX, PARAMETER :: bili=(0.0,1.0)
LOGICAL, PARAMETER :: flag1=.TRUE., flag2=.FALSE.
CHARACTER(*), PARAMETER :: alamat='Jakarta,~Indonesia'
```

Catatan:

- Deklarasi diawali oleh tipe konstanta sesuai tipe data.
- Deklarasi berisi atribut `PARAMETER`.
- Pada contoh di atas, kilo dan mega adalah konstanta skalar bertipe `INTEGER`, gravitasi dan kecil bertipe `REAL`, bili bertipe `COMPLEX`, flag1 dan flag2 bertipe `LOGICAL`, alamat bertipe `CHARACTER`.
- Nilai konstanta langsung diberikan saat dideklarasikan.
- Panjang konstanta `CHARACTER` ditentukan dengan sendirinya oleh data yang disimpannya. Simbol bintang "*" dicantumkan sebagai ganti bilangan. Dengan demikian, panjang konstanta alamat di atas adalah 18. (Ingat bahwa dalam buku ini lambang "~" dicantumkan hanya untuk menunjukkan bahwa di situ ada satu spasi.)
- Program Fortran 90/95 dibaca per baris dari atas ke bawah, dalam satu baris dari kiri ke kanan. Deklarasi konstanta mega di atas menggunakan konstanta kilo. Namun, ini tidak menimbulkan masalah, karena kilo sudah dideklarasikan sebelum mega.

Konstanta array rank 1 dideklarasikan dengan bantuan perintah pembentuk array, contohnya sebagai berikut:

```
INTEGER, DIMENSION(5), PARAMETER :: bulat=(/ 6,8,10,12,14 /)
REAL, DIMENSION(10), PARAMETER :: nilai=(/ (0.5*i, i=1,10) /)
COMPLEX, DIMENSION(2), PARAMETER :: z=(/ (0.0,0.0),(0.0,0.1) /)
LOGICAL, DIMENSION(2), PARAMETER :: logika=(/ .TRUE.,.FALSE. /)
CHARACTER(*), DIMENSION(2), PARAMETER :: nama=(/ "Adi~","Sita" /)
```

Catatan:

- Deklarasi konstanta array sama dengan deklarasi konstanta skalar, namun berisi tambahan atribut `DIMENSION` untuk menyatakan rank atau jumlah dimensi (dalam contoh ini 1), bentuk, dan ukuran konstanta array itu. Pada contoh di atas, konstanta bulat berdimensi 1, bentuknya (5), ukurannya 5.

- Untuk bisa mendeklarasikan konstanta nilai seperti di atas, variabel `INTEGER` i harus sudah dideklarasikan sebelumnya.

- Semua elemen data array `CHARACTER` harus sama panjangnya. Pada contoh di atas, elemen data konstanta nama panjangnya 4.

Kini, misalkan ada satu variabel atau konstanta array x, yang dideklarasikan dengan atribut `DIMENSION(5)`. Elemen-elemen variabel atau konstanta x itu diberi nomor atau indeks yang dimulai dari 1, yaitu x(1), x(2), x(3), x(4), x(5). Bisa saja elemen-elemen x diberi indeks yang tidak dimulai dari 1. Untuk itu, batas bawah dan batas atas indeks secara eksplisit diberikan dalam atribut `DIMENSION`. Sebagai contoh, untuk memberikan indeks pada x sebagai x(-1), x(0), x(1), x(2), x(3), digunakan atribut `DIMENSION(-1:3)`.

Pernyataan `-1:3` pada contoh atribut `DIMENSION` di atas menyatakan sebaran nilai bilangan bulat secara berurutan dari -1 sampai 3, yaitu -1, 0, 1, 2, 3. Pernyataan `[nilai1]:[nilai2]` berlaku untuk nilai1 dan nilai2 bertipe `INTEGER` dan dapat ditemukan juga pada beberapa perintah, pernyataan, atribut lain Fortran 90/95, tidak hanya pada atribut `DIMENSION`. Keadaan yang mungkin untuk pernyataan `[nilai1]:[nilai2]` adalah sebagai berikut:

- Jika nilai1 $<$ nilai2, maka itu menyatakan sebaran nilai bilangan bulat dari nilai1 sampai nilai2 secara berurutan.

- Jika nilai1 $=$ nilai2, maka hanya ada satu nilai bilangan bulat, yaitu nilai1.

- Jika nilai1 $>$ nilai2, maka tidak ada suatu nilai yang berlaku dan hal tersebut, bergantung pada perintah / pernyataan / atribut yang memakainya, bisa merupakan sesuatu yang diperkenankan atau suatu kesalahan yang menyebabkan program tidak dapat dikompilasi atau eksekusi program berhenti.

- Jika pada suatu perintah / pernyataan / atribut nilai1 tidak diberikan, maka nilai1 adalah nilai bilangan bulat terkecil yang mungkin untuk perintah / pernyataan / atribut tersebut; jika nilai2 tidak diberikan, maka nilai2 adalah nilai bilangan bulat terbesar yang mungkin; jika keduanya nilai1 dan nilai2 tidak diberikan, maka itu menyatakan sebaran nilai bilangan bulat secara berurutan dari yang terkecil sampai yang terbesar, yang mungkin untuk perintah / pernyataan / atribut tersebut.

Setelah dideklarasikan, variabel dan konstanta dapat digunakan. Kita lihat berikut ini contoh-contoh pemberian nilai / data untuk variabel, yang berlaku juga untuk konstanta, kecuali jika disebutkan tidak demikian. Untuk membantu kali ini dicantumkan nomor baris disisi paling kiri, yang tentu saja bukan bagian dari penulisan program Fortran 90/95.

```
   INTEGER :: i,j
   REAL :: r,s,t
   COMPLEX :: c
   CHARACTER(4) :: kode
   CHARACTER(8) :: nama
   LOGICAL :: bs
   INTEGER, DIMENSION(5) :: u
   INTEGER, DIMENSION(6:10) :: v

 1 r=11./4.
 2 s=11/4
 3 t=11/4.
 4
 5 i=-11./4.
 6 j=3./(2.+3.)
 7
 8 c=(2.0,3.0)
 9 c=CMPLX(2.0,3.0)
10
11 kode="AB"
12 kode="ABCDEFG"
13
14 nama="ABI&
15      &MANYU"
16
17 nama(1:4)="ABIM"
18 nama(5:5)="A"
19 nama(6:8)='NYU'
20
21 kode=nama(2:5)
22
23 bs=.FALSE.
24
```

```
25 u=(/(10*i,i=1,5)/)
26 v=u
27
28 u(3:4)=u(2:3)
29 v(8)=v(7)
30 v(9)=v(8)
```

Catatan:

- Pernyataan di baris ke-1 memberikan data `REAL` 11./4. (= 2.75) untuk variabel `REAL` r, maka variabel r berisi nilai bilangan riil 2.75.

- Di baris ke-2 data 11 dan 4 kedua-duanya tanpa tanda desimal, yang berarti keduanya merupakan data `INTEGER`, sehingga 11/4 menghasilkan data `INTEGER` 2 (nilai 0.75 terbuang). Data `INTEGER` 2 itu disimpan dalam variabel `REAL` s sebagai nilai bilangan riil 2.0.

- Di baris ke-3 operasi 11/4., yang melibatkan data `INTEGER` 11 dan data `REAL` 4., menghasilkan data `REAL` 2.75 (lihat Tabel 4.3 di Bab 4), sehingga variabel `REAL` t berisi nilai bilangan riil 2.75.

- Di baris ke-5 data `REAL` -11./4. (= -2.75) diberikan untuk variabel `INTEGER` i. Akibatnya, variabel i hanya menyimpan nilai bilangan bulat -2. Begitu pula variabel j di baris ke-6 menyimpan nilai bilangan bulat 0.

- Pernyataan di baris ke-8 memberikan data `COMPLEX` dengan komponen riil 2.0 dan komponen imajiner 3.0 untuk variabel `COMPLEX` c. Hal yang sama dapat dilakukan menggunakan fungsi intrinsik `CMPLX`, seperti dilakukan di baris ke-9.

- Di baris ke-11 data `CHARACTER` "AB", yang panjangnya hanya 2, diberikan untuk variabel `CHARACTER` kode, yang panjangnya 4. Data yang disimpan dalam variabel kode adalah "AB˜˜". Jadi, data "AB" menempati dua posisi pertama dari empat posisi yang tersedia.

- Di baris ke-12 data `CHARACTER` "ABCDEFG", yang panjangnya 7, diberikan untuk variabel `CHARACTER` kode, yang panjangnya 4. Data yang disimpan dalam variabel kode adalah "ABCD", yaitu 4 karakter pertama dari data "ABCDEFG".

- Baris ke-14 sampai 15 merupakan contoh pemberian data `CHARACTER` yang bersambung ke baris berikutnya. Variabel `CHARACTER` nama menyimpan data "ABIMANYU". Tanda "&" diberikan di akhir baris sebelumnya dan di awal baris sesudahnya. Tidak ada spasi antara tanda "&" dan data `CHARACTER` yang ingin disimpan, kecuali jika memang spasi itu diinginkan sebagai bagian dari data.

- Sebuah data `CHARACTER` terdiri dari satu atau lebih karakter yang disusun dengan urutan tertentu. Tiap karakter merupakan elemen data dengan indeks tertentu. Data `CHARACTER`, dengan demikian, pada dasarnya merupakan data array. Oleh

karena itu, variabel `CHARACTER` juga dapat diberi data secara sebagian demi sebagian. Pernyataan di baris ke-17 sampai 19 menunjukkan contoh pemberian data "ABIMANYU" untuk variabel nama secara sebagian demi sebagian. Cara ini, namun, tidak berlaku untuk konstanta.

- Data yang disimpan dalam variabel dan konstanta `CHARACTER` dapat dibaca secara sebagian. Sebagai contoh, pernyataan di baris ke-21 membaca elemen data ke-2 sampai 5, yang disimpan dalam variabel nama, dan diberikan untuk variabel kode, sehingga variabel kode berisi "BIMA".

- Di baris ke-23 variabel `LOGICAL` bs diberi nilai `.FALSE.`. Perhatikan bahwa variabel `LOGICAL` ketika dideklarasikan sudah memiliki nilai *default*, yaitu `.TRUE.`. Jadi, sebelum sampai di baris ke-23 variabel bs berisi nilai `.TRUE.`.

- Pernyataan di baris ke-25 memberikan urutan data `INTEGER` 10, 20, 30, 40, 50 untuk variabel array `INTEGER` u, sehingga elemen-elemen variabel u menyimpan data sebagai berikut: u(1) = 10, u(2) = 20, u(3) = 30, u(4) = 40, u(5) = 50.

- Di baris ke-26 data yang tersimpan di variabel u disalin ke variabel v, dengan urutan elemen data yang sama, sehingga variabel v berisi v(6) = 10, v(7) = 20, v(8) = 30, v(9) = 40, v(10) = 50.

- Satu pernyataan tunggal di baris ke-28 menyalin data dari u(2) dan u(3) masing-masing ke u(3) dan u(4). Hasilnya, u(3) = 20 dan u(4) = 30.

- Pernyataan di baris ke-29 sampai 30 sepintas melakukan hal yang sama untuk variabel v, namun sesungguhnya tidak demikian. Pernyataan di baris ke-29 menyalin data dari v(7) ke v(8), sehingga v(8) = 20. Pernyataan di baris ke-30 menyalin data dari v(8) ke v(9), padahal v(8) baru saja diubah di baris ke-29, sehingga v(9) = 20.

Bab 4

Operator

Sebuah operator melakukan suatu tindakan pada data, data menjadi operand bagi operator tersebut. Ada operator yang bekerja pada satu operand, disebut operator monadik (*monadic*), dan ada yang bekerja pada dua operand, disebut operator diadik (*dyadic*). Operator minus "-" pada -c merupakan contoh operator monadik. Di sini operator minus bekerja pada satu operand c, yaitu menegatifkan c. Namun, operator minus dapat juga menjadi operator diadik, contoh, pada a-b. Di sini operator minus bekerja pada dua operand a dan b, yaitu mengurangkan b dari a. Contoh lain operator diadik adalah operator perkalian "∗", seperti pada a∗b.

Ada 4 jenis tindakan atau operasi operator, yaitu operasi numerik, operasi karakter, operasi relasi, dan operasi logika. Fortran 90/95 menyediakan sejumlah operator, disebut operator intrinsik, untuk keempat jenis operasi tersebut. Disamping itu, sesuai kebutuhan pembuat program juga dapat mendefinisikan suatu operator baru, disebut operator buatan. Di sini hanya dibahas operator intrinsik Fortran 90/95, yang sudah cukup dapat memenuhi banyak kebutuhan. Daftar lengkap operator intrinsik Fortran 90/95 diberikan pada Tabel 4.1.

Kolom pertama Tabel 4.1 menunjukkan hirarki operator. Hirarki operator menentukan urutan pengerjaan operasi, yaitu operasi operator yang hirarkinya lebih tinggi dikerjakan lebih dulu dari operasi operator yang hirarkinya lebih rendah. Contoh, operasi-operasi berikut:

```
a**b*c
d*e**f
r+s/t
```

dikerjakan oleh komputer sebagai:

```
(a**b)*c
d*(e**f)
r+(s/t)
```

yaitu bahwa operasi di antara tanda kurung '(' dan ')' dikerjakan lebih dulu.

Tabel 4.1: Operator Intrinsik Fortran 90/95

hirarki	jenis operasi	operator	operasi	sifat operator
tertinggi	numerik	**	pangkat	diadik
∧		*, /	kali, bagi	
		+, -	tambah, kurang	
			positif, negatif	monadik
	karakter	//	disambung dengan	diadik
	relasi	.EQ. (==),	sama dengan,	diadik
		.NE. (/=),	tidak sama dengan,	
		.LT. (<),	kurang dari,	
		.LE. (<=),	kurang dari atau sama dengan,	
		.GT. (>),	lebih dari,	
		.GE. (>=)	lebih dari atau sama dengan	
	logika	.NOT.	tidak	monadik
		.AND.	dan	diadik
		.OR.	atau	
∨		.EQV.,	ekivalen (xnor),	
terendah		.NEQV.	tidak ekivalen (xor)	

Keterangan: ∗ Operator .EQ. (==), .NE. (/=), .LT. (<), .LE. (<=), .GT. (>), dan .GE. (>=) memiliki hirarki yang sama.
∗ Operator .EQV. dan .NEQV. memiliki hirarki yang sama.

Terlepas dari hirarki operator, tanda kurung '(' dan ')' memang dapat selalu digunakan untuk menentukan operasi yang harus dikerjakan lebih dulu. Pada operasi terakhir di atas, jika diinginkan r+s dikerjakan lebih dulu dan setelahnya dibagi t, maka operasi itu dituliskan sebagai:

```
(r+s)/t
```

Jadi, operasi di antara tanda kurung '(' dan ')' memiliki hirarki yang lebih tinggi dari yang lain, sehingga dikerjakan lebih dulu.

Untuk masing-masing jenis operasi numerik, karakter, relasi, dan logika berlaku tipe data tertentu, baik untuk operand maupun hasil operasi, sebagaimana ditunjukkan oleh Tabel 4.2. Perhatikan bahwa operasi relasi memberikan hasil bertipe data `LOGICAL`, sedangkan tipe data operandnya bukan `LOGICAL`. Operator relasi yang berlaku untuk operand bertipe data `COMPLEX` hanya `.EQ.` (==) dan `.NE.` (/=). Operator relasi yang berlaku untuk operand bertipe data `CHARACTER` hanya `.EQ.` (==).

Tabel 4.3 menunjukkan secara lebih detil tipe data operand dan hasil operasi untuk operasi numerik. Anggap tipe data `COMPLEX` lebih kuat dari tipe data `REAL` dan tipe

Tabel 4.2: Tipe Data Operand dan Hasil Operasi

jenis operasi	operator	tipe data operand	tipe data hasil
numerik	**, *, /, +, -	INTEGER, REAL, COMPLEX	INTEGER, REAL, COMPLEX
karakter	//	CHARACTER	CHARACTER
relasi	.EQ. (==), .NE. (/=), .LT. (<), .LE. (<=), .GT. (>), .GE. (>=)	INTEGER, REAL	LOGICAL
		COMPLEX (hanya .EQ. (==) dan .NE. (/=))	
		CHARACTER (hanya .EQ. (==))	
logika	.NOT., .AND., .OR., .EQV., .NEQV.	LOGICAL	LOGICAL

data REAL lebih kuat dari tipe data INTEGER. Hasil operasi numerik memiliki tipe data yang sama dengan tipe data yang lebih kuat di antara tipe data operandnya.

Tabel 4.3: Tipe Data Untuk Operasi Numerik

a(**,*,/,+,-)b		
tipe a	tipe b	tipe hasil
I	I	I
I	R	R
I	C	C
R	I	R
R	R	R
R	C	C
C	I	C
C	R	C
C	C	C

I	:	INTEGER
R	:	REAL
C	:	COMPLEX

Kini kita lihat contoh-contoh operasi dan hasilnya. Ambillah beberapa variabel skalar yang dideklarasikan sebagai berikut:

```
INTEGER :: i,j,k
REAL :: r,s
COMPLEX :: c,d
CHARACTER(4) :: nama1,nama2
CHARACTER(8) :: nama
LOGICAL :: bs,cek,flag
```

Contoh 1:

```
j=2
k=3
s=2.0
d=(0.0,1.0)

i=(-k+2)*j
r=(s**(k-2)-3.0)*0.5
c=(1.5,-0.5)*j-2.0*d
```

Operasi di atas menghasilkan i = -2, r = -0.5, dan c = (3.0,-3.0).

Contoh 2:

```
i=i*2
s=2.0**s
```

Semua variabel di sebelah kanan tanda sama dengan "=" berisi nilai yang lama dan variabel di sebelah kiri berisi nilai yang baru. Dengan demikian, suatu variabel dapat diperbarui nilainya dengan operasi yang melibatkan variabel itu sendiri. Sesuai nilai variabel i dan s pada Contoh 1, operasi di atas menghasilkan i = -4 dan s = 4.0.

Contoh 3:

```
nama1='SENO'
nama2="PATI"

nama=nama1//nama2
```

Operasi di atas menghasilkan nama = "SENOPATI".

Contoh 4:

```
nama2=nama(1:3)//nama(6:6)
nama=nama(5:8)//nama(1:4)
```

Sesuai isi variabel nama pada Contoh 3, operasi di atas menghasilkan nama2 = 'SENA' dan nama = "PATISENO".

Contoh 5:

```
bs=10>2 .NEQV. 2*5>2
cek=nama == 'SENOPATI'
flag=d .EQ. c
```

Sesuai nilai variabel d dan c pada Contoh 1 serta nama pada Contoh 4, operasi di atas menghasilkan bs = `.FALSE.`, cek = `.FALSE.`, dan flag = `.FALSE.`.

Contoh 6:

```
flag= .NOT. cek
bs=flag .AND. bs
cek= cek .OR. .NOT. bs
```

Sesuai nilai variabel bs dan cek pada Contoh 5, operasi di atas menghasilkan flag = `.TRUE.`, bs = `.FALSE.`, dan cek = `.TRUE.`.

Selanjutnya kita lihat contoh-contoh operasi dengan variabel array, yang dideklarasikan sebagai berikut:

```
REAL :: s
REAL, DIMENSION(5) :: u,v
REAL, DIMENSION(10,20) :: p,q,r
```

Contoh 1:

```
p=q*r-q/r
u=(v+1.0)*(v-2.0)
```

Operasi yang melibatkan data array dilakukan untuk tiap elemen data array yang bersesuaian. Operasi di baris pertama di atas dikerjakan seperti p(i,j) = q(i,j)∗r(i,j)-q(i,j)/r(i,j) untuk i = 1, ..., 10 dan j = 1, ..., 20, operasi di baris kedua seperti u(k) = (v(k)+1.0)∗(v(k)-2.0) untuk k = 1, ..., 5.

Contoh 2:

```
q=s
```

Operasi di atas dikerjakan seperti q(i,j) = s untuk i = 1, ..., 10 dan j = 1, ..., 20. Dengan demikian, semua elemen variabel array q diberi nilai skalar yang sama s.

Contoh 3:

```
u=p(3:7,11)/v
q(2,:)=r(10,:)
```

Data yang tersimpan dalam variabel p merupakan data array 2 dimensi berbentuk (10,20). Namun, pernyataan `p(3:7,11)` dalam operasi di baris pertama di atas memberikan data array 1 dimensi dengan jumlah elemen 5, sehingga dapat dioperasikan dengan variabel v dan hasilnya disimpan dalam variabel u. Operasi di baris tersebut dikerjakan seperti u(k) = p(k+2,11)/v(k) untuk k = 1, ..., 5. Operasi di baris kedua dikerjakan seperti q(2,j) = r(10,j) untuk j = 1, ..., 20, yang berarti baris ke-10 matriks r disalin ke baris ke-2 matriks q.

Bab 5

Percabangan

Sebuah program komputer dijalankan baris demi baris, langkah demi langkah. Pada suatu titik tertentu eksekusi program itu mungkin dihadapkan pada dua atau lebih pilihan langkah selanjutnya. Di sini eksekusi program menemui titik percabangan.

Untuk tiap pilihan langkah selanjutnya berlaku suatu syarat yang harus dipenuhi. Langkah mana yang dijalankan, itu bergantung pada syarat mana yang dipenuhi dalam eksekusi program tersebut di titik percabangan itu. Syarat untuk masing-masing langkah selanjutnya, dengan demikian, harus unik, saling berbeda satu dari yang lain.

Satu perintah yang erat pemakaiannya dengan percabangan adalah `GOTO`. Perintah ini membuat eksekusi program melompat ke baris tertentu di dalam program. Karena itu, perintah `GOTO` dapat menjadi satu dari beberapa pilihan langkah selanjutnya di suatu titik percabangan, yaitu apabila suatu syarat tertentu dipenuhi, maka eksekusi program melompat ke suatu baris yang ditentukan oleh perintah `GOTO`.

5.1 Perintah `GOTO`

Perintah `GOTO` dipakai dalam sebuah program untuk membuat eksekusi program itu melompat ke baris tertentu. Bentuk perintah `GOTO` adalah

```
GOTO [label]
```

Label merupakan sederetan angka, contoh, 100, yang harus dicantumkan di awal baris program yang menjadi target perintah `GOTO`. Label harus berisi minimal satu angka bukan 0. Panjang label maksimum 5 angka dan angka 0 di depan (*leading zeros*) tidak berarti, dengan demikian, label 00110 sama dengan label 110. Berikut ini contoh perintah `GOTO`:

```
    x=2.0
    GOTO 110
100 x=x*x     ! baris ini dilompati, tidak dikerjakan
    z=100.0*x ! baris ini dilompati, tidak dikerjakan
110 z=0.1*x
    y=5.0+z
```

Label harus bersifat unik, supaya jelas baris program mana yang menjadi target perintah GOTO. Perlu kita ketahui di sini bahwa label berlaku hanya di bagian perintah & pernyataan suatu subprogram atau program utama tempat label itu dicantumkan. Sebagai contoh, ambillah sebuah program utama yang berisi beberapa subprogram internal. Label yang sama, misalkan 10010, dapat diberikan di dalam semua subprogram internal itu dan juga di bagian perintah & pernyataan program utama tanpa menimbulkan ketidakpastian, karena masing-masing label itu dalam hal ini bersifat unik. Label 10010 yang ada di dalam satu subprogram internal tidak akan tertukar dengan yang ada di dalam subprogram internal yang lain maupun dengan yang ada di bagian perintah & pernyataan program utama. Sebaliknya, dua atau lebih label yang sama tidak boleh berada di bagian perintah & pernyataan suatu subprogram atau program utama, karena label-label itu tidak bersifat unik, sehingga tidak dapat menjadi target perintah GOTO. Sifat label seperti tersebut di atas, dengan demikian, membuat perintah GOTO bersifat lokal. Perintah GOTO tidak dapat diberikan untuk membuat eksekusi program melompat keluar dari unit / subunit program tempat perintah GOTO itu berada.

5.2 Pernyataan dan Konstruksi IF

Jika suatu percabangan hanya memiliki dua pilihan langkah selanjutnya, maka pernyataan IF dapat dipakai. Bentuk pernyataan IF sebagai berikut:

```
IF ([syarat]) [pekerjaan]
```

Pernyataan itu mengatakan bahwa jika syarat dipenuhi, maka komputer harus melakukan pekerjaan yang dicantumkan di sebelah kanan, sebaliknya jika syarat tidak dipenuhi, maka pekerjaan itu tidak dilakukan. Pada pernyataan IF tersebut (dan juga pada konstruksi IF di bawah) syarat merupakan sebuah ekspresi skalar bertipe LOGICAL, contoh, $a+b > 0$. Syarat dipenuhi apabila bernilai .TRUE. dan sebaliknya tidak dipenuhi apabila bernilai .FALSE..

Contoh pernyataan IF sebagai berikut:

```
    r=0.0
    IF (i >= 0) GOTO 100
    r=r+3.0
100 r=r**2
```

Misalkan pada program di atas variabel i diberi nilai 3. Ini membuat syarat i >= 0 dipenuhi dan perintah GOTO dikerjakan, sehingga nilai akhir r adalah 0.0. Jika misalkan variabel i diberi nilai -1, maka syarat i >= 0 tidak dipenuhi, perintah GOTO tidak dijalankan, dan nilai akhir r adalah 9.0.

Pernyataan `IF` dapat dipakai apabila hanya terdapat satu pekerjaan yang diinginkan. Jika ada beberapa pekerjaan yang harus dilakukan, maka tidak digunakan pernyataan `IF`, melainkan konstruksi `IF` seperti di bawah ini:

```
IF ([syarat]) THEN
   [pekerjaan]
END IF
```

Di antara baris pernyataan `IF` dan `END IF` itu dapat dituliskan satu atau lebih pekerjaan. Contoh konstruksi `IF` sebagai berikut:

```
r=0.0
IF (i < 0) THEN
 r=r+3.0
END IF
r=r**2
```

Perhatikan bahwa program di atas memberikan hasil yang sama dengan contoh program sebelumnya. Ini sekaligus menunjukkan bahwa kita dapat membuat program dalam cara-cara yang berbeda.

Pada pernyataan dan konstruksi `IF` di atas tidak ada pekerjaan yang harus dilakukan komputer, apabila syarat tidak dipenuhi. Apabila ada pekerjaan yang harus dilakukan komputer ketika syarat tidak dipenuhi, maka konstruksi `IF` dikembangkan dengan menyisipkan pernyataan `ELSE`, sehingga struktur konstruksi `IF` itu menjadi seperti di bawah ini:

```
IF ([syarat]) THEN
   [pekerjaan 1]
ELSE
   [pekerjaan 2]
END IF
```

Dengan konstruksi `IF` tersebut komputer akan melakukan pekerjaan 1, apabila syarat dipenuhi, atau pekerjaan 2, apabila syarat tidak dipenuhi. Contoh:

```
IF (i >= 0) THEN
 r=0.0
ELSE
 r=3.0
END IF
r=r**2
```

Program di atas juga memberikan hasil yang sama dengan dua contoh program sebelumnya.

Konstruksi `IF` dengan struktur yang paling umum dapat dipakai untuk percabangan yang bersambung, yaitu apabila syarat tidak dipenuhi, maka eksekusi program dihadapkan lagi pada satu titik percabangan, demikian seterusnya. Struktur umum konstruksi `IF` tersebut berisi pernyataan `ELSE IF` sebagai berikut:

```
IF ([syarat 1]) THEN
   [pekerjaan 1]
ELSE IF ([syarat 2]) THEN
   [pekerjaan 2]
ELSE IF ([syarat 3]) THEN
   [pekerjaan 3]
...
ELSE
   [pekerjaan lain]
END IF
```

Keterangan:

- Struktur umum konstruksi `IF` dapat berisi satu atau lebih pernyataan `ELSE IF`.
- Syarat 1, syarat 2, syarat 3, dan seterusnya harus berbeda satu dari yang lain, tidak boleh saling beririsan.
- Pernyataan `ELSE` diperlukan hanya apabila ada pekerjaan yang harus dilakukan komputer ketika semua syarat dalam rangkaian pernyataan `IF` dan `ELSE IF` di atasnya tidak dipenuhi.

Contoh konstruksi `IF` dengan struktur yang paling umum sebagai berikut:

```
IF (i > 0) THEN
 r=0.0
ELSE IF (i < 0) THEN
 r=3.0
ELSE
 r=1.0
END IF
r=r**2
```

Sesuai konstruksi `IF` di atas jika, misalkan, nilai i sama dengan 2, maka nilai akhir r adalah 0.0; jika nilai i sama dengan -3, maka nilai akhir r adalah 9.0; jika nilai i sama dengan 0, maka nilai akhir r adalah 1.0.

5.3 Konstruksi SELECT CASE

Pernyataan dan konstruksi IF dipakai untuk percabangan yang memiliki hanya dua pilihan langkah selanjutnya. Untuk percabangan yang memiliki beberapa, dua atau lebih, pilihan langkah selanjutnya Fortran 90/95 menyediakan konstruksi SELECT CASE.

Konstruksi SELECT CASE memiliki struktur sebagai berikut:

```
SELECT CASE ([variabel selektor])
 CASE ([nilai 1])
      [pekerjaan 1]
 CASE ([nilai 2])
      [pekerjaan 2]
 CASE ([nilai 3])
      [pekerjaan 3]
 ...
 CASE DEFAULT
      [pekerjaan lain]
END SELECT
```

Keterangan:

- Di antara baris pernyataan SELECT CASE dan END SELECT terdapat beberapa pernyataan CASE.
- Variabel selektor merupakan variabel skalar bertipe INTEGER, CHARACTER, atau LOGICAL.
- Nilai 1, nilai 2, nilai 3, dan seterusnya dapat berupa hanya satu nilai atau suatu kumpulan nilai, yang tipenya sesuai dengan tipe variabel selektor.
- Nilai 1, nilai 2, nilai 3, dan seterusnya harus berbeda satu dari yang lain, tidak boleh saling beririsan.
- Jika nilai variabel selektor termasuk dalam nilai 1, maka komputer melakukan pekerjaan 1; jika termasuk dalam nilai 2, maka komputer melakukan pekerjaan 2; demikian selanjutnya.
- Pernyataan CASE DEFAULT diperlukan hanya apabila ada pekerjaan yang harus dikerjakan komputer ketika nilai variabel selektor tidak termasuk dalam semua nilai 1, nilai 2, nilai 3, dan seterusnya.

Berikut ini contoh konstruksi SELECT CASE:

```
SELECT CASE (j)
 CASE (0,3,-7,10)
  kode='merah'
 CASE (-5:-1,11:15)
```

```
  kode='hijau'
 CASE (:-10,20:)
  kode='biru'
 CASE DEFAULT
  kode='hitam'
END SELECT
```

Catatan:

- Pada contoh di atas anggaplah variabel selektor j bertipe `INTEGER` dan variabel kode bertipe `CHARACTER`.
- Konstruksi `SELECT CASE` tersebut memberikan data `CHARACTER` tertentu untuk variabel kode, berdasarkan nilai j.
- Jika j sama dengan salah satu dari nilai-nilai 0, 3, -7, dan 10, maka kode diberi data 'merah'.
- Jika j sama dengan salah satu dari nilai-nilai bulat dari -5 sampai -1 dan dari 11 sampai 15, maka kode diberi data 'hijau'.
- Jika j sama dengan salah satu dari nilai-nilai bulat dari nilai `INTEGER` terkecil sampai dengan -10 dan dari 20 sampai dengan nilai `INTEGER` terbesar, maka kode diberi data 'biru'.
- Jika j tidak termasuk dalam semua nilai di atas, maka kode diberi data 'hitam'.

Contoh lain, yaitu konstruksi `SELECT CASE` tanpa pernyataan `CASE DEFAULT`, ditunjukkan sebagai berikut:

```
SELECT CASE (kode)
 CASE ('merah','hijau','biru')
  x=x+2.0
 CASE ('hitam')
  x=x-2.0
END SELECT
```

Di sini variabel selektor kode bertipe `CHARACTER` dan variabel x `REAL`. Jika kode tidak berisi salah satu dari 'merah', 'hijau', 'biru', atau 'hitam', maka komputer tidak mengubah nilai variabel x.

Contoh konstruksi `SELECT CASE` dengan variabel selektor bertipe `LOGICAL` diberikan sebagai berikut:

```
SELECT CASE (cerita)
 CASE (.TRUE.)
  laporan='Ini kisah nyata.'
 CASE DEFAULT
  laporan='Ini bukan kisah nyata.'
END SELECT
```

Jika cerita bernilai `.TRUE.`, maka variabel `CHARACTER` laporan berisi data 'Ini kisah nyata.'. Sebaliknya, jika cerita bernilai `.FALSE.`, maka variabel laporan berisi data 'Ini bukan kisah nyata.'.

Bab 6

Pengulangan

Seringkali suatu hasil diperoleh dengan melakukan suatu pekerjaan secara berulang-ulang. Contoh, untuk mendapatkan nilai $10! = 10 \times 9 \times 8 \times ... \times 2 \times 1$ yang kita lakukan adalah mengulang-ulang pekerjaan ini: mengalikan suatu bilangan bulat dengan bilangan bulat berikutnya yang lebih kecil. Dalam bab ini kita bahas perintah untuk membuat komputer melakukan suatu pekerjaan secara berulang-ulang.

6.1 Konstruksi `DO`

Untuk mengulang-ulang suatu pekerjaan kita gunakan konstruksi `DO` sebagai berikut:

```
DO [variabel pengulangan]=[nilai1],[nilai2],[langkah]
   [pekerjaan]
END DO
```

Konstruksi tersebut mengatakan kurang lebih begini: lakukan pekerjaan yang diberikan secara berulang-ulang untuk nilai variabel pengulangan sama dengan nilai1 sampai nilai2, dengan perubahan nilai variabel pengulangan sebesar langkah. Jadi, konstruksi `DO` seperti membentuk sebuah lingkaran atau *loop*, karena itu sering disebut juga sebagai lingkaran (*loop*) `DO`. Keterangan mengenai konstruksi `DO` sebagai berikut:

- Konstruksi `DO` diawali oleh pernyataan `DO` dan diakhiri oleh pernyataan `END DO`.
- Variabel pengulangan merupakan variabel `INTEGER` skalar.
- Data nilai1, nilai2, dan langkah bertipe `INTEGER`, baik positif maupun negatif, yang ekspresinya dapat berupa bilangan, contoh, 0, 10, 2, maupun kombinasi bilangan, konstanta, variabel `INTEGER`, contoh, j+k, j*k, 2*(k-j) (anggaplah j dan k variabel atau konstanta `INTEGER`).
- Mungkin saja nilai2 yang diberikan tidak sama dengan nilai1 + (n × langkah), dengan n = bilangan bulat = 0, 1, 2, ... Apabila, contoh, nilai2 yang diberikan berada di antara nilai1 + (9 × langkah) dan nilai1 + (10 × langkah), maka nilai terakhir variabel pengulangan yang mungkin adalah bukan nilai2, melainkan

nilai1 + (9 × langkah). Jadi, nilai terakhir variabel pengulangan yang mungkin tidak melampaui nilai2.

- Jika langkah tidak diberikan seperti pada konstruksi DO berikut:

```
DO [variabel pengulangan]=[nilai1],[nilai2]
   [pekerjaan]
END DO
```

  maka dengan sendirinya langkah ditentukan sama dengan 1.

Berikut ini adalah contoh konstruksi DO, yang dalam hal ini menghitung nilai $x = 1 + 3 + 5 + 7$:

```
x=0.0
DO i=1,8,2
 x=x+i
END DO
```

Catatan:

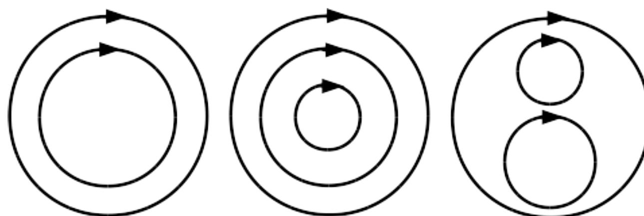
- Nilai terakhir variabel i dalam konstruksi DO di atas adalah 7, bukan 8.
- Sebelum eksekusi program memasuki konstruksi DO nilai x = 0.0.
- Setelah eksekusi program masuk ke konstruksi DO tiap kali nilai x diubah menjadi x = 1.0 saat i = 1, x = 4.0 saat i = 3, x = 9.0 saat i = 5, dan terakhir x = 16.0 saat i = 7.

Pekerjaan yang sama juga dapat diselesaikan dengan konstruksi DO seperti berikut, yang kali ini nilai variabel pengulangan i bukan bertambah melainkan berkurang:

```
x=0.0
DO i=7,1,-2
 x=x+i
END DO
```

6.2 Konstruksi DO Bersusun

Konstruksi DO dapat dibuat bersusun, yaitu ada konstruksi DO di dalam konstruksi DO yang lain. Konstruksi-konstruksi DO tersebut tidak boleh berpotongan. Beberapa contoh konstruksi DO bersusun diilustrasikan oleh gambar berikut:



Pada konstruksi DO bersusun yang pertama kali diselesaikan adalah konstruksi paling dalam dan yang terakhir adalah konstruksi paling luar. Kita ingat kembali, sebagaimana disampaikan di Subbab 3.2, bagaimana elemen data array diurutkan menurut Fortran 90/95. Karena itu, jika proses di dalam suatu konstruksi DO bersusun melibatkan data array, sebaiknya usahakan agar konstruksi DO paling dalam berkaitan dengan sumbu dimensi pertama dan konstruksi DO paling luar berkaitan dengan sumbu dimensi terakhir. Berikut ini contoh konstruksi DO bersusun yang baik:

```
DO j=1,30
 DO i=1,20
  a(i,j)=x(i)*y(j)
 END DO
END DO
```

sedangkan yang di bawah ini contoh yang kurang baik:

```
DO i=1,20
 DO j=1,30
  a(i,j)=x(i)*y(j)
 END DO
END DO
```

Eksekusi program untuk konstruksi DO bersusun yang terakhir di atas lebih lambat dari yang untuk konstruksi DO bersusun sebelumnya, karena variabel a diubah lebih dulu sepanjang sumbu dimensi kedua (diwakili oleh variabel j) dan kemudian sepanjang sumbu dimensi pertama (diwakili oleh variabel i).

6.3 Perintah EXIT dan CYCLE

Bayangkan suatu keadaan sebagai berikut. Ada suatu pekerjaan yang harus dilakukan berulang-ulang sebanyak 10 kali untuk memberikan suatu hasil. Namun, apabila belum sampai 10 kali dilakukan ternyata sudah diperoleh hasil yang diinginkan, maka pengulangan harus dihentikan. Pada keadaan seperti itu perintah EXIT dapat digunakan.

Perintah EXIT membuat eksekusi program segera keluar dari konstruksi DO tanpa menyelesaikan sisa pekerjaan dan sisa pengulangan dalam konstruksi DO tersebut. Dengan kata lain, eksekusi program langsung melompat ke baris program setelah baris yang berisi pernyataan END DO. Contoh penggunaan perintah EXIT sebagai berikut:

```
x=0.0
DO i=1,8,2
 x=x+i
 IF (x > 3.0) EXIT
 y=x**2
END DO
```

Catatan:

- Saat i = 1, maka x = 1.0 dan perintah `EXIT` tidak dilaksanakan, karena syarat pernyataan `IF`, yaitu x > 3.0, tidak dipenuhi. Lalu, nilai y dihitung, diperoleh y = 1.0, dan konstruksi `DO` berlanjut ke putaran berikutnya untuk nilai i = 3.
- Saat i = 3, maka x = 4.0 dan ini berarti syarat pernyataan `IF` dipenuhi, sehingga perintah `EXIT` dijalankan dan konstruksi `DO` tidak dilanjutkan. Nilai y tidak dihitung lagi, sehingga nilai terakhir y adalah 1.0. Nilai terakhir x adalah 4.0.

Kini bayangkan suatu keadaan yang berbeda. Andaikan pekerjaan di dalam suatu konstruksi `DO` terdiri dari dua bagian. Bagian pertama harus terus dikerjakan berulang-ulang sampai konstruksi `DO` itu selesai dijalankan, sedangkan bagian kedua dikerjakan berulang-ulang hanya sampai memberikan suatu hasil tertentu. Jadi, sementara konstruksi `DO` terus berlanjut sampai selesai dan mengerjakan bagian pertama, bagian kedua berhenti dikerjakan setelah didapatkan suatu hasil. Untuk keadaan seperti ini diperlukan perintah `CYCLE`.

Perintah `CYCLE` membawa eksekusi program melompat ke baris yang berisi pernyataan `END DO`. Jadi, eksekusi program tidak keluar dari konstruksi `DO`, tapi langsung melompat ke putaran berikutnya. Dengan demikian, semua pekerjaan yang ada di antara perintah `CYCLE` dan pernyataan `END DO` tidak dilakukan. Mari kita ambil contoh konstruksi `DO` di atas yang berisi perintah `EXIT`, namun kita ganti perintah `EXIT` dengan perintah `CYCLE` untuk melihat perbedaannya sebagai berikut:

```
x=0.0
DO i=1,8,2
 x=x+i
 IF (x > 3.0) CYCLE
 y=x**2
END DO
```

Catatan:

- Saat i = 1, maka x = 1.0 dan perintah `CYCLE` tidak dilaksanakan, karena syarat pernyataan `IF`, yaitu x > 3.0, tidak dipenuhi. Lalu, nilai y dihitung, diperoleh y = 1.0, dan konstruksi `DO` dilanjutkan ke putaran berikutnya untuk nilai i = 3.
- Saat i = 3, maka x = 4.0 dan berarti syarat pernyataan `IF` dipenuhi, sehingga perintah `CYCLE` dijalankan. Hal yang sama terjadi juga ketika nilai i = 5 dan 7. Jadi, sejak nilai i = 3 nilai y tidak dihitung lagi, sehingga nilai terakhir y adalah 1.0. Sementara, nilai x terus dihitung, sehingga nilai terakhir x adalah 16.0.

6.4 Konstruksi DO tanpa Variabel Pengulangan

Konstruksi DO juga bisa dibuat tanpa variabel pengulangan. Ini berarti jumlah pengulangan tidak ditentukan dan pengulangan pada dasarnya dapat berlangsung terus menerus tanpa batas. Konstruksi DO seperti ini digunakan untuk melakukan secara berulang-ulang suatu pekerjaan, yang jumlah pengulangannya tidak dapat ditentukan, namun, diketahui suatu kondisi yang harus dipenuhi oleh pekerjaan itu. Dengan demikian, apabila kondisi tersebut sudah dipenuhi, maka pengulangan dihentikan. Untuk menghentikan pengulangan digunakan perintah EXIT.

Struktur konstruksi DO dengan jumlah pengulangan yang tidak ditentukan seperti digambarkan di atas kurang lebih sebagai berikut:

```
DO
   [pekerjaan]
   IF ([kondisi]) EXIT
END DO
```

Jadi, jika kondisi dipenuhi, maka eksekusi program keluar dari konstruksi DO itu. Sebagai contoh kita ambil kembali konstruksi DO di subbab sebelum ini dan lakukan sedikit perubahan sebagai berikut:

```
i=-1
x=0.0
DO
 i=i+2
 x=x+i
 IF (x > 3.0) EXIT
END DO
```

Pada putaran ke-2 konstruksi DO di atas berhenti dengan hasil $i = 3$ dan $x = 4.0$.

Bab 7

Subprogram

Sebuah program komputer bisa saja terdiri dari hanya program utama tanpa subprogram internal. Namun, sebaiknya sebuah program, khususnya program yang besar, disusun sedemikian sehingga terdiri dari beberapa subprogram, tiap subprogram menjalankan satu subpekerjaan tertentu. Keuntungan menyusun sebuah program yang terdiri dari beberapa subprogram antara lain:

- Penulisan program menjadi lebih mudah. Pembuat program dapat berkonsentrasi pada satu subpekerjaan dan untuk itu menulis satu subprogram.
- Pencarian kesalahan lebih mudah dan aman dilakukan. Pembuat program hanya perlu memeriksa dan memperbaiki subprogram yang mengandung kesalahan. Bagian lain dari program itu, yang tidak mengandung kesalahan, tidak terusik meski secara tak sengaja.
- Program lebih ringkas. Jika ada sebagian pekerjaan yang dilakukan beberapa kali di tempat-tempat berbeda dalam sebuah program, maka sepenggal program untuk sebagian pekerjaan itu tidak perlu dituliskan beberapa kali, melainkan cukup sekali sebagai satu subprogram. Hal ini juga dapat mengurangi jumlah kesalahan yang mungkin dibuat dalam menulis sebuah program.

Subprogram dapat disertakan dalam sebuah program sebagai subprogram internal, subprogram eksternal, maupun subprogram modul. Subprogram modul yaitu subprogram yang berada di dalam sebuah modul. Subprogram internal adalah subprogram yang berada di dalam program utama, subprogram eksternal, atau subprogram modul, yaitu setelah pernyataan `CONTAINS`. Modul dan subprogram eksternal ditempatkan di luar program utama, bisa di dalam file yang sama maupun terpisah. Yang disampaikan dalam bab ini berlaku sama untuk subprogram-subprogram tersebut: subprogram internal, subprogram eksternal, dan subprogram modul. Dalam Subbab 9.5 diberikan penjelasan tambahan berkenaan dengan subprogram eksternal, khususnya mengenai blok `INTERFACE`. Penjelasan tentang modul, termasuk subprogram modul, diberikan dalam Subbab 9.6.

Subprogram dipanggil (dijalankan) oleh program utama atau oleh subprogram lain. Pada pemanggilan subprogram terjadi perpindahan data dari pemanggil ke subprogram

dan sebaliknya dari subprogram ke pemanggil. Perpindahan data tersebut berlangsung melalui argumen-argumen subprogram itu. Ada juga subprogram yang dapat dipanggil oleh dirinya sendiri, selain oleh program utama dan subprogram lain. Subprogram seperti ini disebut subprogram rekursif dan dibahas di Subbab 7.4.

Bentuk subprogram adalah subrutin dan fungsi. Biasanya, subrutin dibuat untuk menyelesaikan suatu pekerjaan yang relatif besar atau kompleks serta memberikan beberapa luaran. Sementara, fungsi untuk pekerjaan yang ringkas dan memberikan hanya satu luaran, contoh, menghitung nilai suatu fungsi matematis. Dalam Fortran 90/95 juga telah tersedia banyak subprogram intrinsik, baik berupa subrutin maupun fungsi, sebagian diberikan di Lampiran B.

7.1 Argumen Subprogram

Pada pemanggilan subprogram terjadi perpindahan data antara pemanggil dan subprogram melalui argumen-argumen subprogram itu. Ada argumen aktual, yang berada di sisi pemanggil, dan argumen *dummy*, yang berada di sisi subprogram. Argumen *dummy* berupa variabel yang dideklarasikan di dalam subprogram. Dengan demikian, argumen *dummy* termasuk variabel lokal subprogram itu. Argumen aktual dapat berupa variabel, konstanta, atau ekspresi data, seperti -0.1E-04. Jika suatu argumen aktual harus menerima data dari argumen *dummy*, maka argumen aktual itu harus berupa variabel. Argumen *dummy* juga dapat bersifat pilihan (*optional*), yaitu boleh tidak dihubungkan dengan suatu argumen aktual. Argumen *dummy* pilihan dibahas di Subbab 9.3.

Data dipindahkan antara argumen aktual dan argumen *dummy*. Dengan demikian, argumen aktual dan argumen *dummy* harus sama tipenya (`REAL`, `INTEGER`, `COMPLEX`, `CHARACTER`, atau `LOGICAL`) dan jenisnya (skalar atau array). Untuk argumen bertipe `REAL` dan `COMPLEX` harus sama juga presisinya. Jika argumen itu berupa array, maka harus sama rank dan ukurannya, namun boleh berbeda indeks elemennya. Sebagai contoh, data dapat dipindahkan antara argumen aktual array a, yang elemen-elemennya a(1), a(2), a(3), dan argumen *dummy* array x, yang elemen-elemennya x(-1), x(0), x(1). Nama argumen aktual boleh sama dengan atau berbeda dari nama argumen *dummy* yang dihubungkan dengan argumen aktual itu.

Saat subprogram dipanggil, argumen *dummy* yang mana dihubungkan dengan argumen aktual yang mana, itu ditentukan berdasarkan posisi argumen itu dalam urutan argumen. Argumen *dummy* pertama dihubungkan dengan argumen aktual pertama, yang kedua dengan yang kedua, demikian seterusnya. Selain berdasarkan posisi, ada juga cara lain yang dijelaskan di Subbab 9.3. Cara lain tersebut bermanfaat, khususnya, untuk argumen *dummy* pilihan.

7.2 Subrutin

Struktur subrutin serupa dengan struktur program utama Fortran 90/95, yaitu:

```
SUBROUTINE [nama subrutin] ([argumen])
[bagian spesifikasi]
[isi subrutin]
END SUBROUTINE [nama subrutin]
```

Seperti nama program, nama subrutin juga tidak boleh mengandung spasi. Subrutin dapat memiliki argumen (argumen *dummy*), yang dituliskan setelah nama subrutin, dalam tanda kurung. Namun, sebuah subrutin tidak harus memiliki argumen; itu bergantung pada pekerjaan yang harus diselesaikannya.

Berikut ini contoh subrutin, yang diberi nama subfortranku:

```
SUBROUTINE subfortranku(x,y,z)

IMPLICIT NONE
REAL, INTENT(IN) :: x,y
REAL, INTENT(OUT) :: z
REAL :: s

s=x+y
z=s*(s+1.0)

RETURN

END SUBROUTINE subfortranku
```

Catatan:

- Subrutin subfortranku di atas memiliki argumen *dummy*, yaitu x, y, dan z.
- Variabel s bukan argumen, melainkan variabel lokal biasa.
- Argumen *dummy* x dan y untuk menerima masukan, karena dideklarasikan dengan atribut `INTENT(IN)`.
- Argumen *dummy* z untuk menyimpan luaran, karena dideklarasikan dengan atribut `INTENT(OUT)`.
- Atribut `INTENT` dapat diberi 1 dari 3 spesifikasi berikut:
  - `IN`, yaitu untuk argumen *dummy* yang menerima masukan. Nilai argumen *dummy* yang dideklarasikan dengan atribut `INTENT(IN)` ditentukan di luar subrutin melalui argumen aktual dan nilai tersebut tidak boleh diubah di dalam subrutin.
  - `OUT`, yaitu untuk argumen *dummy* yang menyimpan luaran. Nilai argumen *dummy* yang dideklarasikan dengan atribut `INTENT(OUT)` ditentukan di dalam subrutin, sebagai hasil subrutin tersebut.

  - `INOUT`, yaitu untuk argumen *dummy* yang menerima masukan dan juga menyimpan luaran. Argumen *dummy* yang dideklarasikan dengan atribut `INTENT(INOUT)` menerima masukan saat awal subrutin itu dijalankan. Di dalam subrutin nilainya dapat diubah. Ketika subrutin selesai dijalankan argumen itu menyimpan luaran. Nilai argumen *dummy* itu sebelum dan sesudah subrutin dijalankan dapat berbeda.

- Perintah `RETURN` di akhir subrutin atau di mana pun ditemui dalam subrutin itu membuat eksekusi program keluar dari subrutin itu, kembali ke pemanggil, tepat ke baris program setelah baris yang berisi perintah pemanggilan subrutin itu.

Sebuah subrutin dipanggil dengan perintah `CALL`. Sebagai contoh:

```
CALL subfortranku(a,b,c)
```

Pada pemanggilan itu a, b, c merupakan argumen aktual. Sesuai urutan argumen, nilai a dan b masing-masing disalin ke argumen *dummy* x dan y. Terakhir, nilai argumen *dummy* z disalin ke c.

7.3 Fungsi

Struktur fungsi sebagai berikut:

```
FUNCTION [nama fungsi] ([argumen]) RESULT([variabel hasil])
[bagian spesifikasi]
[isi fungsi]
END FUNCTION [nama fungsi]
```

Berbeda dari subrutin, yang boleh tidak memiliki argumen, sebuah fungsi harus memiliki argumen (argumen *dummy*). Argumen *dummy* tersebut hanya untuk menerima masukan bagi fungsi itu, sedangkan luaran fungsi tidak perlu diberikan ke pemanggil melalui sebuah argumen, karena bagaimanapun sebuah fungsi hanya memberikan satu luaran. Argumen dituliskan setelah nama fungsi, dalam tanda kurung. Nama fungsi tidak boleh mengandung spasi.

Berikut ini contoh fungsi, yang diberi nama funcfortranku:

```
FUNCTION funcfortranku(x) RESULT(z)

IMPLICIT NONE
REAL,INTENT(IN) :: x
REAL :: z

z=1.0-x+0.5*x*x
```

```
RETURN

END FUNCTION funcfortranku
```

Catatan:

- Fungsi funcfortranku di atas memiliki satu argumen *dummy*, yaitu x, untuk menerima masukan dan, karena itu, dideklarasikan dengan atribut `INTENT(IN)`.
- Variabel z menyimpan hasil fungsi tersebut. Karena bukan argumen, melainkan variabel lokal biasa, z tidak dideklarasikan dengan atribut `INTENT`.
- Perintah `RETURN` di akhir fungsi atau di mana pun ditemui dalam fungsi itu membuat eksekusi program keluar dari fungsi itu, kembali ke pemanggil, ke posisi tempat fungsi itu dipanggil, sambil membawa luaran.

Sebuah fungsi dipanggil tanpa perintah `CALL`. Pemanggilan sebuah fungsi dapat menyisip di suatu baris program sebagai bagian dari pernyataan di baris tersebut. Contoh:

```
d=a*funcfortranku(c)+b
```

Di situ c merupakan argumen aktual. Nilai c disalin ke argumen *dummy* x dan terakhir fungsi itu memberikan hasil ke pemanggil di posisi pemanggilan.

7.4 Subprogram Rekursif

Subprogram yang disampaikan sebelum ini, baik subrutin di Subbab 7.2 maupun fungsi di Subbab 7.3, hanya dapat dipanggil dari luar subprogram itu. Tapi, ada juga subprogram yang bisa dipanggil oleh subprogram itu sendiri, dari dalam subprogram tersebut. Subprogram seperti itu disebut subprogram rekursif.

Struktur subprogram rekursif sama dengan struktur subprogram biasa, namun diberi awalan `RECURSIVE` pada pernyataan `SUBROUTINE` atau `FUNCTION` sebagai berikut:

```
RECURSIVE SUBROUTINE [nama subrutin] ([argumen])
[bagian spesifikasi]
[isi subrutin]
END SUBROUTINE [nama subrutin]

RECURSIVE FUNCTION [nama fungsi] ([argumen]) RESULT([variabel hasil])
[bagian spesifikasi]
[isi fungsi]
END FUNCTION [nama fungsi]
```

Subprogram rekursif dipanggil dengan cara yang sama seperti memanggil subprogram biasa. Subprogram rekursif juga dapat memiliki argumen *dummy* pilihan.

Berikut ini contoh fungsi rekursif yang menghitung nilai fungsi faktorial:

```
RECURSIVE FUNCTION faktorial(n) RESULT(m)

IMPLICIT NONE
INTEGER, INTENT(IN) :: n
INTEGER :: m

IF (n==1 .OR. n==0) THEN
 m=1                 ! 0! = 1! = 1
ELSE IF (n>1) THEN
 m=n*faktorial(n-1)  ! m = n(n-1)(n-2)...1 = n!
ELSE
 STOP                ! program berhenti jika n negatif
END IF

RETURN

END FUNCTION faktorial
```

Setelah dipanggil dari luar, apabila nilai argumennya lebih besar dari 1, fungsi faktorial akan terus memanggil dirinya sendiri dengan nilai argumen yang makin kecil sampai argumennya bernilai 1.

Bab 8

Masukan & Luaran

Sebuah program mungkin memerlukan data atau parameter awal. Untuk itu, program tersebut harus menerima atau membaca data, yang mungkin saja disimpan dalam sebuah file atau diketikkan orang pada keyboard. Juga, sebuah program mungkin harus memberikan atau menulis data, misalkan ke layar monitor atau ke sebuah file. Ini merupakan proses masukan & luaran atau untuk selanjutnya kami singkat sebagai proses *I/O* (*input & output*).

8.1 Proses *I/O* Terformat dan Tak Terformat

Data masukan maupun data luaran, disingkat data *I/O*, dapat berupa data terformat (*formatted*) maupun data tak terformat (*unformatted*). Data *I/O* tak terformat adalah data yang masih dalam wujud asalnya sesuai sistem komputer yang digunakan. Data *I/O* terformat adalah yang sudah diubah menjadi susunan karakter, sehingga dapat ditampilkan atau dapat kita baca sebagai suatu teks atau susunan bilangan dan huruf.

Pada proses *I/O* terformat pengubahan data dari wujud asalnya menjadi data terformat bisa saja menyebabkan semacam penurunan kualitas, seperti penurunan presisi atau ketelitian suatu nilai akibat pembulatan. Namun, sepanjang hal itu dapat diterima atau tidak menimbulkan masalah, maka kita dapat memilih proses *I/O* terformat. Bahkan mungkin saja proses *I/O* yang diperlukan justru yang bersifat terformat. Misalkan sebuah program dikehendaki menghasilkan data yang dapat kita baca dan untuk ditampilkan dalam bentuk grafik. Dalam hal ini program tersebut harus menghasilkan data terformat agar dapat dibaca baik oleh kita maupun oleh software pembuat grafik.

Proses *I/O* tak terformat digunakan, contoh, apabila data yang dihasilkan oleh suatu program diperlukan oleh beberapa program lain untuk menyelesaikan beberapa perhitungan lanjutan. Agar kualitas perhitungan tetap terjaga, maka data yang dihasilkan oleh program pertama harus disimpan sebagai data tak terformat. Contoh lain, apabila data perlu dipindahkan dari satu komputer ke komputer lain yang sistemnya sama, tanpa penurunan kualitas. Data yang dipindahkan itu harus bersifat tak terformat atau dihasilkan dari proses *I/O* tak terformat.

8.2 Perintah READ dan WRITE

Untuk keperluan membaca dan menulis data Fortran 90/95 menyediakan perintah `READ` dan `WRITE`. Perintah `READ` dan `WRITE` untuk proses *I/O* terformat memiliki struktur sebagai berikut:

```
READ([unit],[format]) [daftar masukan]
WRITE([unit],[format]) [daftar luaran]
```

Keterangan:

- Unit menentukan dari / ke mana data dibaca / ditulis. Contoh, data dibaca dari keyboard atau dari file, data ditulis ke layar monitor atau file.
- Format menentukan format / ekspresi / tampilan data *I/O*, juga dapat mengontrol proses *I/O*.
- Daftar masukan adalah urutan variabel untuk menyimpan data masukan.
- Daftar luaran adalah urutan variabel dan / atau konstanta, yang nilainya ingin ditulis, serta dapat juga langsung suatu ekspresi data `CHARACTER`.

Untuk proses *I/O* tak terformat penentuan format tidak diperlukan. Dengan demikian, perintah `READ` dan `WRITE` untuk proses *I/O* tak terformat lebih sederhana strukturnya, yaitu:

```
READ([unit]) [daftar masukan]
WRITE([unit]) [daftar luaran]
```

Proses *I/O* tak terformat hanya dapat dilakukan dari / ke file, sehingga unit menentukan dari / ke file mana data dibaca / ditulis. Daftar luaran tidak dapat berisi suatu ekspresi data `CHARACTER`.

8.3 Unit

Pada perintah `READ` unit menentukan dari mana data dibaca. Pada perintah `WRITE` unit menentukan ke mana data ditulis. Di sini kami tunjukkan cara:

- membaca dari keyboard dan menulis ke layar monitor, lebih tepatnya ke jendela tempat program dijalankan,
- membaca dari & menulis ke file internal,
- membaca dari & menulis ke file eksternal,

dengan menentukan unit. Perhatikan bahwa pada proses *I/O* tak terformat hanya yang terakhir yang dapat dilakukan, yaitu membaca dari & menulis ke file eksternal.

Untuk membaca dari keyboard dan menulis ke layar monitor kita tentukan unit $= *$ sebagai berikut:

```
READ(*,[format]) [daftar masukan]
WRITE(*,[format]) [daftar luaran]
```

Jika bertemu sebuah perintah `READ` seperti di atas eksekusi program akan menunggu sampai semua data dimasukkan, sesuai format dan daftar masukan yang diberikan. Pada keyboard data dimasukkan dengan menekan tombol `ENTER`. Perintah `WRITE` di atas menulis data yang tersimpan dalam daftar luaran ke jendela pada layar monitor, tempat program dijalankan, sesuai format yang diberikan.[1]

Yang dimaksud dengan file internal pada proses *I/O* terformat tidak lain adalah sebuah variabel bertipe `CHARACTER`. Membaca data dari file internal berarti membaca isi sebuah variabel `CHARACTER`. Data yang dibaca itu kemudian disalin ke variabel-variabel yang ada dalam daftar masukan sesuai format yang diberikan. Pada proses itu data yang dibaca diubah tipenya menjadi sesuai dengan tipe variabel yang menyimpannya. Menulis data ke file internal berarti memberi sebuah variabel `CHARACTER` suatu nilai, sesuai format dan data yang terkandung dalam daftar luaran. Pada proses tersebut data yang terkandung dalam daftar luaran diubah tipenya menjadi `CHARACTER`. Perhatikan agar panjang variabel `CHARACTER` itu cukup untuk menampung data yang terkandung dalam daftar luaran, sehingga tidak ada data yang hilang. Berikut ini contoh perintah membaca dari dan menulis ke file internal yang bernama infile, yaitu sebuah variabel `CHARACTER` yang panjangnya 10:

```
CHARACTER(10) :: infile ! deklarasi variabel infile
...
READ(infile,[format]) [daftar masukan]
WRITE(infile,[format]) [daftar luaran]
```

[1]Keyboard dan layar monitor merupakan perangkat *default* yang digunakan dalam proses *I/O* terformat untuk unit $= *$. Namun, dengan *redirection* pada saat menjalankan program kita dapat mengarahkan aliran data, sehingga dalam suatu program semua perintah `READ` membaca data bukan dari keyboard, melainkan dari sebuah file; begitu pula, semua perintah `WRITE` menulis data bukan ke layar monitor, melainkan ke sebuah file. Misalkan kita memiliki sebuah *executable file*, yang bernama procoba. Berikut ini contoh eksekusi procoba serta keterangan proses *I/O*-nya untuk unit $= *$:

- procoba < input.dat
  (`READ` membaca data dari file input.dat)
- procoba > output.dat
  (`WRITE` menulis data ke file output.dat)
- procoba < input.dat > output.dat
  (`READ` membaca data dari file input.dat dan `WRITE` menulis data ke file output.dat)

Kini, kami tunjukkan cara membaca dari dan menulis ke sebuah file eksternal, yang tidak lain adalah file yang biasa kita kenal, yang disimpan di media penyimpanan, seperti harddisk komputer. Untuk itu diperlukan perintah `OPEN` dan `CLOSE`. Perintah `OPEN` membuka atau menyiapkan sebuah file untuk proses *I/O* yang ingin dilakukan. Perintah `CLOSE` menutup file tersebut setelah proses *I/O* selesai dikerjakan. Perintah `OPEN` memberi suatu nomor (nomor unit) untuk file yang dibuka. Nomor unit itu selanjutnya dipakai sebagai unit pada perintah `READ` dan `WRITE`. Berikut ini ilustrasi cara membaca dari sebuah file, yang diberi nomor unit 11, dan cara menulis ke sebuah file, yang diberi nomor unit 12:

```
OPEN(11,[argumen lainnya])
READ(11,[format]) [daftar masukan]
CLOSE(11)

OPEN(12,[argumen lainnya])
WRITE(12,[format]) [daftar luaran]
CLOSE(12)
```

Cara serupa berlaku juga untuk proses *I/O* tak terformat, namun tentu saja tanpa penentuan format pada perintah `READ` dan `WRITE`.

Perintah `OPEN` dan `CLOSE` memiliki struktur sebagai berikut:

```
OPEN([nomor unit],[argumen lainnya])
CLOSE([nomor unit])
```

Nomor unit dapat diisi bilangan 1 sampai 99. Perlu diperhatikan bahwa beberapa nomor terendah mungkin sudah dipakai oleh komputer / sistem komputer yang digunakan. Oleh karena itu, sebaiknya gunakan nomor yang lebih besar, seperti 5, 6, dan seterusnya. Argumen lainnya pada perintah `OPEN` adalah `FILE`, `STATUS`, `FORM`, dan `ACTION`, yang dijelaskan sebagai berikut:

- Argumen `FILE`, `STATUS`, `FORM`, dan `ACTION` semuanya merupakan variabel bertipe `CHARACTER`. Karena itu, jangan lupakan sepasang tanda petik tunggal atau ganda ketika memberi nilai untuk keempatnya. Contoh, `FILE="input.dat"`.
- Argumen `FILE` menyimpan nama file yang dibuka, termasuk *path*-nya, apabila file itu tidak berada di folder / direktori yang sama dengan folder / direktori *executable file*.
- Argumen `STATUS` dapat diberi nilai '`NEW`', '`OLD`', atau '`REPLACE`'. Masukkan '`NEW`' untuk membuat dan membuka file baru, yang tidak boleh sudah ada sebelumnya. Masukkan '`OLD`' untuk membuka file, yang harus sudah ada sebelumnya. Masukkan '`REPLACE`' untuk membuka dan mengganti file yang sudah ada atau untuk membuat dan membuka file baru, apabila file itu belum ada.

- Argumen FORM dapat diberi nilai 'FORMATTED' atau 'UNFORMATTED'. Masukkan 'FORMATTED' untuk proses *I/O* terformat atau 'UNFORMATTED' untuk proses *I/O* tak terformat. Nilai *default* FORM adalah 'FORMATTED'. Dengan demikian, apabila FORM tidak dicantumkan pada perintah OPEN, maka proses *I/O* yang berlangsung adalah yang bersifat terformat.
- Argumen ACTION dapat diberi nilai 'READ', 'WRITE', atau 'READWRITE'. Masukkan 'READ' apabila file dibuka untuk hanya dibaca, 'WRITE' untuk hanya ditulis, dan 'READWRITE' untuk dibaca dan ditulis.

Pada contoh berikut perintah OPEN membuat dan membuka file bernama hasil.out di folder / direktori yang sama dengan folder / direktori *executable file* untuk hanya ditulis:

```
OPEN(6,FILE='hasil.out',STATUS='NEW',ACTION='WRITE')
WRITE(6,[format]) [daftar luaran]
CLOSE(6)
```

Pada contoh itu argumen FORM tidak dicantumkan pada perintah OPEN. Dengan demikian, proses *I/O* yang dilakukan adalah yang bersifat terformat, sesuai nilai *default* FORM, yaitu 'FORMATTED'. Setelah proses menulis selesai dikerjakan dengan perintah WRITE, file hasil.out ditutup dengan perintah CLOSE. Contoh perintah OPEN untuk proses *I/O* tak terformat adalah sebagai berikut:

```
OPEN(6,FILE='output.dat',STATUS='REPLACE',FORM='UNFORMATTED',ACTION='WRITE')
WRITE(6) [daftar luaran]
CLOSE(6)
```

Untuk proses *I/O* tak terformat argumen FORM harus dicantumkan pada perintah OPEN, dengan nilai 'UNFORMATTED'.

8.4 Format

Format data *I/O* pada proses *I/O* terformat dapat dibiarkan bebas atau ditentukan. Kita sebut saja format data pada kasus pertama adalah format bebas dan pada kasus kedua format yang ditentukan.

8.4.1 Format Bebas

Jika ingin format data dibiarkan bebas, maka pada perintah READ dan WRITE kita tentukan format $= *$ sebagai berikut:

```
READ([unit],*) [daftar masukan]
WRITE([unit],*) [daftar luaran]
```

Dalam hal ini perintah `READ` membaca data masukan dengan ekspresi apa adanya dari unit yang ditentukan. Beberapa ketentuan berlaku sebagai berikut:

- Data masukan dapat dipisahkan oleh tanda koma, atau spasi, atau kombinasi tanda koma dan spasi; spasi itu bisa cukup satu atau lebih dari satu.
- Jika terdapat bilangan kompleks, maka harus dituliskan dalam sepasang tanda kurung.
- Bila terdapat tanda "/" dalam data masukan, maka proses membaca berhenti sampai tanda "/" dan semua data yang berada setelah tanda "/" tidak dibaca, kecuali, jika tanda "/" itu merupakan bagian dari suatu data `CHARACTER`.

Perintah `WRITE` menulis data yang tersimpan dalam daftar luaran ke unit yang ditentukan dengan format atau ekspresi yang bergantung pada *processor*. Dengan demikian, dua komputer dengan *processor* yang berbeda mungkin memberikan tampilan data luaran yang berbeda.

Di bawah ini diberikan contoh proses *I/O* terformat dengan format bebas. Anggaplah ada beberapa variabel yang dideklarasikan sebagai berikut:

```
IMPLICIT NONE
INTEGER :: j
REAL :: r
COMPLEX :: z
CHARACTER(4) :: g,h
LOGICAL :: b,d
```

Variabel-variabel itu diberi nilai melalui, contoh saja, keyboard dengan sebuah perintah `READ` berikut:

```
READ(*,*) j,r,z,g,h,b,d
```

Data yang dimasukkan adalah:

```
2049,37.089,(-3.0,5.1),'BAR/',AMAT,.TRUE.,/,F
```

Perhatikan bahwa data `COMPLEX` diberikan dalam sepasang tanda kurung. Pada proses itu berlangsung hal-hal berikut:

- Nilai yang disimpan dalam variabel-variabel j, r, z, g, h, b, dan d adalah j = 2049, r = 37.089, z = (-3.0,5.1), g = 'BAR/', h = 'AMAT', b = `.TRUE.` , dan d = `.TRUE.`.
- Tanda "/" yang pertama tidak membuat proses membaca berhenti, karena tanda "/" itu merupakan bagian dari data `CHARACTER` "BAR/".

- Tanda "/" yang kedua membuat proses membaca berhenti, sehingga variabel d tidak diberi nilai pada proses masukan itu dan, dengan demikian, masih menyimpan nilai `.TRUE.` sebagai nilai *default* variabel `LOGICAL`.

Hasil yang sama seperti di atas diperoleh apabila data masukan diberikan seperti dua contoh berikut (ingat kembali bahwa lambang "~" di sini menunjukkan adanya spasi):

```
~~2049~37.089~~~(-3.0~,~5.1~)~"BAR/"~AMAT~t/~.FALSE.~
~2049,~~37.089,~(-3.0,~5.1),'BAR/'~,"AMAT",~~T/,f
```

Tampak bahwa data masukan untuk variabel `LOGICAL` dapat diberikan dalam bentuk singkat `T` atau `F`, maupun dalam bentuk panjang `.TRUE.` atau `.FALSE.`. Hal ini berlaku juga untuk proses *I/O* dengan format yang ditentukan (Subbab 8.4.2). Selanjutnya, nilai variabel j, r, z, g, h, b, dan d itu ditampilkan ke layar dengan sebuah perintah `WRITE` berikut:

```
WRITE(*,*) j,r,z,g,h,b,d
```

Tampilan data luaran bergantung pada *processor*, salah satu contoh adalah seperti di bawah ini:

```
~~~~~~~~2049~~~37.0890007~~~~~(~-3.00000000~~~~,~~5.09999990~~~~)~BAR/AMAT~T~T
```

Pada komputer lain mungkin diperoleh tampilan data luaran yang berbeda. Kita lihat bahwa data `COMPLEX` ditampilkan dalam sepasang tanda kurung. Contoh lain, perintah-perintah `WRITE` berikut:

```
WRITE(*,*) 'nilai-nilai~yang~telah~dimasukkan:'
WRITE(*,*) 'j=',j,'r=',r,'z=',z
WRITE(*,*) 'g=',g,'h=',h,'b=',b,'d=',d
```

menghasilkan

```
~nilai-nilai~yang~telah~dimasukkan:
~j=~~~~~~~~2049~r=~~~37.0890007~~~~~z=~(~-3.00000000~~~~,~~5.09999990~~~~)
~g=BAR/h=AMATb=~T~d=~T
```

Contoh-contoh di atas menunjukkan bahwa perintah `WRITE` untuk format bebas selalu menambahkan spasi sebagai karakter pertama yang ditulis. Selain itu, nilai variabel (dan juga konstanta) `LOGICAL` selalu dituliskan dalam bentuk singkat, yaitu `T` atau `F`, yang berlaku juga untuk proses *I/O* dengan format yang ditentukan (Subbab 8.4.2).

8.4.2 Format yang Ditentukan

Untuk menentukan format data *I/O* digunakan deskriptor edit (*edit descriptors*) sebagai isi format pada perintah `READ` dan `WRITE`. Kami tunjukkan terlebih dahulu pada Subbab 8.4.2.1 tiga cara untuk menempatkan deskriptor edit sebagai isi format pada perintah `READ` dan `WRITE`. Deskriptor edit itu sendiri dijelaskan lebih detil pada Subbab 8.4.2.2.

8.4.2.1 Penempatan Deskriptor Edit

Format pada perintah `READ` dan `WRITE` merupakan ekspresi `CHARACTER`. Dengan demikian, deskriptor edit merupakan ekspresi `CHARACTER`. Ada tiga cara untuk menempatkan deskriptor edit sebagai isi format. Pada cara pertama deskriptor edit dituliskan langsung pada perintah `READ` dan `WRITE` dalam tanda kurung. Deskriptor edit dalam tanda kurung tersebut harus diberikan di antara sepasang tanda petik ganda atau tunggal, karena merupakan data `CHARACTER`, sebagai berikut:

```
READ([unit],"([deskriptor edit 1])") [daftar masukan]
WRITE([unit],"([deskriptor edit 2])") [daftar luaran]
```

Pada cara kedua deskriptor edit disimpan dalam sebuah konstanta atau variabel `CHARACTER`, kemudian konstanta atau variabel `CHARACTER` itu dicantumkan dalam perintah `READ` dan `WRITE`. Pada contoh berikut digunakan konstanta `CHARACTER` format1 dan format2 untuk menentukan format data *I/O*.

```
CHARACTER(*), PARAMETER :: format1='([deskriptor edit 1])', &
                           format2='([deskriptor edit 2])'
...
READ([unit],format1) [daftar masukan]
WRITE([unit],format2) [daftar luaran]
```

Pada cara ketiga digunakan perintah `FORMAT`. Perintah `FORMAT` itu ditempatkan pada suatu baris program yang diberi label (label dijelaskan di Subbab 5.1). Deskriptor edit diberikan sebagai argumen perintah `FORMAT`, yaitu `FORMAT([deskriptor edit])`. Penentuan format pada perintah `READ` dan `WRITE` dilakukan dengan mencantumkan label baris program yang berisi perintah `FORMAT`, seperti ditunjukkan oleh contoh berikut:

```
READ([unit],110) [daftar masukan]
WRITE([unit],120) [daftar luaran]
...
110 FORMAT([deskriptor edit 1])
120 FORMAT([deskriptor edit 2])
```

Cara yang mana saja dari ketiga cara di atas dapat dipakai. Namun, apabila ada suatu format tertentu yang digunakan lebih dari satu kali, maka cara kedua dan ketiga

lebih menguntungkan dari cara pertama, karena isi format cukup ditentukan atau dituliskan satu kali, yaitu pada pemberian nilai suatu konstanta atau variabel `CHARACTER` (cara kedua) atau pada sebuah perintah `FORMAT` (cara ketiga).

Untuk sekedar memberikan gambaran, berikut ini diulangi lagi proses *I/O* untuk variabel j, r, z, g, h, b, dan d seperti di Subbab 8.4.1, namun dengan memakai deskriptor edit. Variabel-variabel itu diberi nilai dengan perintah `READ` berikut:

- Cara 1:

```
READ(*,"(I4,F6.3,2F4.1,2A4,2L7)") j,r,z,g,h,b,d
```

- Cara 2:

```
CHARACTER(*), PARAMETER :: format1='(I4,F6.3,2F4.1,2A4,2L7)'
...
READ(*,format1) j,r,z,g,h,b,d
```

- Cara 3:

```
READ(*,110) j,r,z,g,h,b,d
...
110 FORMAT(I4,F6.3,2F4.1,2A4,2L7)
```

Sesuai format "(I4,F6.3,2F4.1,2A4,2L7)" (makna deskriptor edit dijelaskan di Subbab 8.4.2.2), nilai-nilai untuk j, r, z, g, h, b, dan d dimasukkan sebagai berikut:

```
204937.089-3.0~5.1BAR/AMAT~~~~~~T~~~~~~F
```

atau

```
204937.089-3.0~5.1BAR/AMAT~.TRUE..FALSE.
```

Selanjutnya nilai variabel j, r, z, g, h, b, dan d ditampilkan ke layar dengan perintah `WRITE` berikut:

- Cara 1:

```
WRITE(*,'(1X,I4,F7.3,2F5.1,2A5,2L7)') j,r,z,g,h,b,d
```

- Cara 2:

```
CHARACTER(*), PARAMETER :: format2='(1X,I4,F7.3,2F5.1,2A5,2L7)'
...
WRITE(*,format2) j,r,z,g,h,b,d
```

- Cara 3:

```
WRITE(*,120) j,r,z,g,h,b,d
...
120 FORMAT(1X,I4,F7.3,2F5.1,2A5,2L7)
```

Hasilnya adalah

```
~2049~37.089~-3.0~~5.1~BAR/~AMAT~~~~~~T~~~~~~F
```

8.4.2.2 Deskriptor Edit

Kami berikan di sini hampir semua deskriptor edit, yaitu yang menurut kami sangat diperlukan atau banyak digunakan. Pertama, kita lihat deskriptor edit yang berfungsi untuk menentukan format data *I/O*, disebut deskriptor edit data, untuk semua tipe data `INTEGER`, `REAL`, `COMPLEX`, `LOGICAL`, dan `CHARACTER`. Deskriptor edit data tersebut ditunjukkan oleh Tabel 8.1. Ingat kembali bahwa data `COMPLEX` adalah sepasang data `REAL`, karena itu deskriptor edit untuk data `REAL` dan `COMPLEX` sama, hanya saja untuk data `COMPLEX` diperlukan dua deskriptor edit, yang boleh saling berbeda, yaitu untuk komponen riil dan komponen imajinernya.

Tabel 8.1: Deskriptor Edit Data

<table>
<tr><th>tipe data</th><th>deskriptor edit</th><th>contoh</th><th>keterangan</th></tr>
<tr><td>INTEGER</td><td>Iw</td><td>~1001</td><td rowspan="6">• w adalah panjang ekspresi, termasuk tanda +/-, titik desimal, dan eksponen.
• Tanpa w panjang ekspresi data CHARACTER mengikuti panjang data I/O
• d adalah jumlah bilangan di belakang tanda titik desimal</td></tr>
<tr><td rowspan="2">REAL & COMPLEX</td><td>F$w.d$</td><td>~~-72.367</td></tr>
<tr><td>E$w.d$</td><td>~-0.89E-10</td></tr>
<tr><td>LOGICAL</td><td>Lw</td><td>~~F</td></tr>
<tr><td rowspan="2">CHARACTER</td><td>Aw</td><td>lengk</td></tr>
<tr><td>A</td><td>lengkap</td></tr>
</table>

Penjelasan lebih detil mengenai deskriptor edit data pada Tabel 8.1 diberikan dalam contoh-contoh berikut:

- `I7` dipakai untuk satu data `INTEGER` yang panjangnya 7, contoh:

```
1250027
~~~7525
```

Perhatikan bahwa jika panjang data `INTEGER` itu kurang dari w, dalam contoh ini $w = 7$, maka disisipkan satu atau beberapa spasi di bagian awal ekspresi.

- `I4,I2` dipakai untuk dua data `INTEGER` berurutan, yang pertama panjangnya 4 dan yang kedua 2, contoh:

  ```
  ~-47~0
  -37121
  ```

  Contoh yang terakhir dalam hal ini bukan menunjukkan satu data `INTEGER` -37121, melainkan dua, yang pertama -371 dan yang kedua 21.

- `F7.3` dipakai untuk satu data `REAL`, yang panjangnya 7, dengan 3 bilangan di belakang tanda titik desimal, contoh:

  ```
  ~-0.030
  ~75.125
  ```

  Satu atau beberapa spasi disisipkan di bagian awal ekspresi, apabila panjang data kurang dari w, dalam contoh ini $w = 7$.

- `E11.3` dipakai untuk satu data `REAL` dalam bentuk eksponensial, yang panjangnya 11, dengan 3 bilangan di belakang tanda titik desimal, contoh:

  ```
  ~-0.430E-08
  ~~0.525E+12
  ```

  Satu atau beberapa spasi disisipkan di bagian awal ekspresi, apabila panjang data kurang dari w, dalam contoh ini $w = 11$.

- `E10.3,F4.1` dipakai untuk dua data `REAL` berurutan atau satu data `COMPLEX`, contoh:

  ```
  -0.231E-0210.0
  ~0.212E+03-0.2
  ```

  Contoh pertama menunjukkan dua bilangan -0.231E-02 dan 10.0, sedangkan contoh kedua 0.212E+03 dan -0.2. Tidak diperlukan sepasang tanda kurung untuk menyatakan dua bilangan tersebut sebagai satu data `COMPLEX`.

- `L8` dipakai untuk satu data `LOGICAL` yang panjangnya 8, contoh:

  ```
  ~~~~~~~F
  ~~.TRUE.
  ```

  Satu atau beberapa spasi disisipkan di bagian awal ekspresi, apabila panjang data kurang dari w, dalam contoh ini $w = 8$. Contoh kedua, yang melibatkan data `LOGICAL` dalam bentuk panjang `.TRUE.` atau `.FALSE.`, hanya berlaku untuk proses masukan. Proses luaran selalu menampilkan data `LOGICAL` dalam bentuk singkat `T` atau `F`.

- Untuk contoh-contoh `A` dan `A`w kita ambil satu variabel `CHARACTER` vchar yang dideklarasikan panjangnya sama dengan 7. Pertama, kita lihat proses luaran, anggaplah vchar = '1234567':

  - Jika vchar dituliskan dengan deskriptor edit `A`, maka data luaran adalah seluruh isi vchar, yaitu:

    ```
    1234567
    ```

  - Jika dituliskan dengan `A`w dan w lebih dari panjang vchar sebanyak n, maka data luaran adalah isi vchar, dengan tambahan n spasi di bagian kiri. Contoh, `A9` menghasilkan:

    ```
    ˜˜1234567
    ```

  - Jika dituliskan dengan `A`w dan w kurang dari panjang vchar, maka data luaran adalah w karakter pertama (paling kiri) dari isi vchar. Contoh, `A5` menghasilkan:

    ```
    12345
    ```

  Berikutnya, kita lihat proses masukan:

  - Jika data masukan untuk vchar dibaca dengan deskriptor edit `A`, maka data masukan diambil dari kiri ke kanan sesuai panjang vchar. Apabila panjang data kurang dari panjang vchar, maka ditambahkan spasi di dalam vchar. Contoh, data masukan:

    ```
    123456789
    1234
    ```

    secara berurutan menghasilkan vchar yang berisi:

    ```
    1234567
    1234˜˜˜
    ```

  - Jika dibaca dengan `A`w dan w lebih dari panjang vchar sebanyak n, maka mula-mula data masukan diambil dari kiri ke kanan sesuai w dan jika panjang data masukan kurang dari w, ditambahkan maksimal n spasi di bagian kiri serta, jika masih kurang, satu atau beberapa spasi di bagian kanan. Kemudian, data yang disimpan dalam vchar diambil dari yang paling kanan sesuai panjang vchar. Contoh, data masukan:

    ```
    1234567890
    12345678
    123456
    ```

secara berurutan menghasilkan, untuk `A9`, vchar yang berisi:

```
3456789
2345678
123456~
```

– Jika dibaca dengan `A`w dan w kurang dari panjang vchar, maka data masukan diambil dari kiri ke kanan sesuai w dan di dalam vchar ditambahkan spasi di bagian kanan. Jika panjang data masukan kurang dari w, ditambahkan juga spasi di bagian kanan. Contoh, data masukan:

```
123456
123
```

secara berurutan menghasilkan, untuk `A5`, vchar yang berisi:

```
12345~~
123~~~~
```

Selanjutnya, kita lihat beberapa deskriptor edit yang memiliki fungsi untuk mengontrol proses *I/O*, disebut deskriptor edit kontrol. Tabel 8.2 menunjukkan deskriptor edit kontrol tersebut diikuti contoh-contoh pemakaiannya pada perintah `READ` dan `WRITE`. Dalam contoh-contoh itu i, j adalah variabel `INTEGER` dan u, v variabel `CHARACTER`, yang panjangnya 9. Meskipun digunakan unit = $*$, tapi yang ditunjukkan oleh contoh-contoh itu juga berlaku sama dalam proses *I/O* untuk unit yang lain.

Tabel 8.2: Deskriptor Edit Kontrol

deskriptor edit	keterangan
n`X`	menggeser posisi pembacaan / penulisan sejauh n ke kanan
`/`	pada proses masukan berarti pindah ke awal baris berikutnya
	pada proses luaran berarti membuat baris baru
`:`	data *I/O* untuk proses *I/O* setelah deskriptor edit ini bersifat optional, yaitu boleh ada atau tidak ada

- Anggaplah perintah

```
READ(*,'(I3,2X,A)') i,u
```

menerima data masukan

```
~20~~DEWABRATA
```

Deskriptor edit `2X` menggeser posisi pembacaan sejauh 2 ke kanan setelah angka 20. Hasilnya adalah i = 20 dan u = 'DEWABRATA'. Kemudian, perintah

```
WRITE(*,'(1X,I2,5X,A)') i,u
```

menghasilkan luaran

```
~20~~~~~DEWABRATA
```

Di situ deskriptor edit `1X` menggeser posisi penulisan sejauh 1 ke kanan, memberikan satu spasi di awal luaran. Deskriptor edit `5X` menggeser posisi penulisan sejauh 5 ke kanan setelah angka 20.

- Misalkan perintah

```
READ(*,'(I3,/,2X,A)') i,u
```

mendapat dua baris data masukan

```
~20~~DEWABRATA
~~BRATADEWA
```

Deskriptor edit `/` membuat pembacaan setelah angka 20 di baris pertama dilanjutkan ke awal baris kedua. Hasilnya i = 20 dan u = 'BRATADEWA'. Lalu, perintah

```
WRITE(*,'(1X,I2,/,5X,A)') i,u
```

menghasilkan dua baris luaran

```
~20
~~~~~BRATADEWA
```

Deskriptor edit `/` pada perintah `WRITE` di atas membuat penulisan setelah angka 20 dilanjutkan ke baris baru.

- Untuk perintah

```
READ(*,'(I3,2X,A,:,I5,3X,A7)') i,u,j,v
```

tidak harus diberikan 4 data masukan untuk variabel i, u, j, dan v, melainkan minimal 2 data, yaitu satu data `INTEGER` untuk variabel i dengan deskriptor edit `I3` dan satu data `CHARACTER` untuk variabel u dengan deskriptor edit `A`. Jika diberikan data masukan

```
~20~~DEWABRATA
```

dihasilkan i = 20 dan u = 'DEWABRATA'. Hasil yang sama juga diperoleh dengan perintah

```
READ(*,'(I3,2X,A,:,I5,3X,A7)') i,u
```

Kemudian, perintah

```
WRITE(*,'(1X,I2,5X,A,:,I3,I5,A)') i,u
```

tidak harus memiliki 5 data luaran, melainkan minimal 2 data, yang terdiri dari satu data `INTEGER` dengan deskriptor edit `I2` dan satu data `CHARACTER` dengan deskriptor edit `A`. Hasil perintah `WRITE` di atas adalah

```
~20~~~~~DEWABRATA
```

Dengan deskriptor edit : ini dimungkinkan untuk membuat program komputer dengan proses *I/O* yang relatif fleksible dalam hal jumlah data *I/O* yang terlibat.

Satu atau beberapa deskriptor edit data maupun kontrol yang secara berturut-turut berulang dapat dinyatakan dengan mencantumkan jumlah pengulangan dan, jika perlu, sepasang tanda kurung untuk mengelompokkan (*grouping*) beberapa deskriptor edit. Contoh, `3I5,2(I3,2X,E10.3)` sama dengan `I5,I5,I5,I3,2X,E10.3,I3,2X,E10.3`. Di bawah ini diberikan beberapa contoh pengulangan deskriptor edit pada perintah `READ` dan `WRITE`, dengan memakai beberapa variabel yang dideklarasikan sebagai berikut:

```
IMPLICIT NONE
INTEGER :: i,j,k
INTEGER, DIMENSION(3) :: m
REAL :: r,s
COMPLEX :: z
```

Andaikan perintah

```
READ(*,'(2(I3,1X,F5.2),2F4.1)') k,r,j,s,z
READ(*,'(3(I4,/),2X,L7)') (m(i), i=1,3),b
```

menerima data masukan

```
625~-2.21~10~~0.03~3.1-0.2
7500
~750
~~75
~~.FALSE.
```

maka hasilnya adalah k = 625, r = -2.21, j = 10, s = 0.03, z = (3.1,-0.2), m(1) = 7500, m(2) = 750, m(3) = 75, dan b = .FALSE.. Kemudian, perintah

```
WRITE(*,'(1X,2(I4,1X),A,2E11.3,A)') k,j,'(',z,')'
WRITE(*,'(1X,2(F7.3,/,1X),3I5,L3)') r,s,(m(i), i=1,3),b
```

menghasilkan luaran

```
~~625~~~10~(~~0.310E+01~-0.200E+00)
~~-2.210
~~~0.030
~~7500~~750~~~75~~F
```

Untuk mengakhiri penjelasan mengenai proses *I/O* dengan format yang ditentukan, kami sampaikan hal berikut. Beberapa karakter, apabila berada di posisi pertama dalam data luaran, tidak dituliskan, melainkan berfungsi mengontrol proses luaran, contoh, mencetak ke printer. Mengingat data luaran sangat bervariasi, dapat saja terjadi suatu proses luaran berlangsung tidak seperti yang kita harapkan, dikarenakan karakter pertama data luaran secara kebetulan adalah karakter yang memiliki fungsi kontrol tertentu. Untuk menghindari hal ini dapat kita sisipkan spasi sebagai karakter pertama data luaran. Spasi tidak mempunyai fungsi kontrol dan, karena itu, tetap dituliskan sebagai spasi. Dalam contoh-contoh perintah `WRITE` di atas untuk proses luaran dengan format yang ditentukan dapat dilihat bahwa selalu kami cantumkan deskriptor edit `1X` untuk menyisipkan satu spasi sebagai karakter pertama data luaran.

Bab 9

Topik-Topik Lanjut

9.1 Data REAL dan COMPLEX Berpresisi Ganda

Fortran 90/95 banyak digunakan untuk keperluan perhitungan ilmiah, yang menuntut ketelitian atau presisi yang tinggi. Ketelitian suatu nilai dapat ditunjukkan oleh jumlah angka penting yang menyatakannya. Data REAL standar memiliki ketelitian hingga kurang lebih 7 angka penting. Ketelitian seperti itu disebut presisi tunggal. Presisi tunggal mungkin saja tidak cukup untuk suatu perhitungan ilmiah. Untuk itu digunakan presisi ganda, yaitu ketelitian hingga kurang lebih 15 angka penting.

Untuk menggunakan presisi ganda, kita bisa lakukan cara berikut:

- Pertama, kita deklarasikan sebuah konstanta INTEGER, sebut saja dpr, dengan fungsi intrinsik KIND sebagai berikut:

  ```
  INTEGER, PARAMETER :: dpr=KIND(1.0D0)
  ```

  Sebagai argumen fungsi KIND di atas data REAL 1.0D0 memberikan nilai yang sama dengan 1.0E0, yaitu 1.0, namun diterima oleh Fortran 90/95 sebagai data REAL berpresisi ganda. Dengan deklarasi seperti itu konstanta dpr menyimpan satu nilai INTEGER, yang disebut *kind type parameter*, yang dihubungkan dengan tipe data REAL berpresisi ganda. Perhatikan bahwa data REAL yang diberikan sebagai argumen fungsi KIND tidak harus bernilai 1.0, melainkan apa saja, asalkan tipenya sesuai dengan yang diinginkan, dalam hal ini REAL berpresisi ganda.

- Selanjutnya, konstanta dpr dipakai untuk mendeklarasikan variabel dan konstanta REAL berpresisi ganda, contoh:

  ```
  REAL(dpr), PARAMETER :: pi=3.141592654_dpr, kecil=1.0E-10_dpr
  REAL(dpr) :: berat, massa
  REAL(dpr), DIMENSION(3) :: kecepatan
  ```

  Perhatikan bahwa di akhir ekspresi suatu data REAL harus dicantumkan _dpr, seperti dalam deklarasi konstanta pi dan kecil di atas.

- Pencantuman _dpr tersebut tidak hanya untuk deklarasi suatu konstanta, melainkan juga untuk ekspresi data REAL berpresisi ganda di manapun dalam program, contoh:

```
massa=10.0_dpr
gaya=9.8_dpr*massa
vx=0.0_dpr
vy=3.0_dpr
vz=-1.0_dpr
energik=0.5_dpr*massa*(vx**2.0_dpr+vy**2.0_dpr+vz**2.0_dpr)
```

Kini, mengingat sebuah data COMPLEX terdiri dari sepasang data REAL, maka data REAL berpresisi ganda dapat dipakai untuk memberikan data COMPLEX berpresisi ganda. Contoh deklarasi variabel dan konstanta serta pernyataan dalam program yang melibatkan data COMPLEX berpresisi ganda dapat dilihat sebagai berikut:

```
INTEGER, PARAMETER :: dpr=KIND(1.0D0), dpc=KIND((1.0D0,1.0D0))
COMPLEX(dpc), PARAMETER :: imajiner=(0.0_dpr,1.0_dpr)
COMPLEX(dpc) :: znumber1,znumber2,znumber3

znumber1=(4.0_dpr,-7.0_dpr)
znumber2=CMPLX(1.0_dpr)
znumber3=CMPLX(1.0_dpr,-1.0_dpr)
```

Pada contoh di atas dideklarasikan satu konstanta INTEGER, yaitu dpc, yang dihubungkan dengan tipe data COMPLEX berpresisi ganda. Konstanta INTEGER dpr diperlukan juga untuk menyatakan bahwa tipe data komponen riil dan imajiner kedua-duanya, tidak boleh hanya salah satu, adalah REAL berpresisi ganda.

9.2 Alokasi Dinamis Variabel Array

Untuk tiap variabel atau konstanta yang dideklarasikan dalam sebuah program dialokasikan sejumlah memori komputer untuk menyimpan data yang menjadi isi variabel atau konstanta itu. Di Subbab 3.3 kita lihat bagaimana sebuah variabel array dideklarasikan dengan menentukan rank, bentuk, dan ukurannya. Rank, bentuk, dan ukuran variabel array itu tetap seperti yang dideklarasikan, yang berarti demikian pula jumlah memori komputer yang dialokasikan untuk variabel array itu. Namun, mungkin saja data yang disimpan dalam suatu variabel array hanya diperlukan di sebagian waktu eksekusi program, misalkan, hanya di bagian awal, sedangkan selanjutnya data itu tidak diperlukan lagi. Meskipun begitu, sejumlah memori komputer sebanyak yang diminta pada saat variabel array itu dideklarasikan tetap dipakai untuk menyimpan data tersebut. Dengan demikian, dalam hal ini terjadi pemakaian memori komputer yang sia-sia.

Fortran 90/95 menyediakan suatu cara agar kita dapat membebaskan sejumlah memori komputer apabila data yang tersimpan dalam suatu variabel array tidak lagi diperlukan. Dengan demikian, pemakaian memori komputer dapat dibuat lebih efisien. Untuk itu, variabel array tersebut harus dideklarasikan dengan hanya menentukan ranknya, sementara bentuk dan ukurannya dibiarkan bebas. Bentuk dan ukuran variabel array itu baru ditentukan kemudian di bagian perintah & pernyataan program utama atau subprogram, sekali atau beberapa kali sesuai kebutuhan. Tiap kali ditentukan bentuk dan ukuran variabel array itu bisa berbeda-beda, bergantung pada jumlah data yang harus disimpannya, namun ranknya tetap sesuai dengan yang dideklarasikan. Ini yang dimaksud dengan alokasi dinamis variabel array. Variabel array yang bersifat seperti itu tentu tidak dapat menjadi argumen *dummy* suatu subprogram, karena bentuk dan ukuran argumen *dummy* array ditentukan pada saat subprogram itu dipanggil, yaitu harus cocok dengan bentuk dan ukuran argumen aktual yang bersesuaian.

Variabel array yang alokasinya bersifat dinamis, kita sebut saja variabel array teralokasikan (*allocatable*), dideklarasikan dengan atribut `ALLOCATABLE`, contoh,

```
REAL, DIMENSION(:,:,:), ALLOCATABLE :: omega
```

Pada contoh itu omega adalah variabel array rank 3 `REAL`, yang bentuk dan ukurannya dapat ditentukankan secara dinamis kemudian pada saat diperlukan sesuai jumlah data yang harus disimpannya.

Alokasi variabel array omega dilakukan dengan perintah `ALLOCATE` seperti contoh berikut:

```
ALLOCATE(omega(10,5,20))
```

Di sini variabel array omega berbentuk (10,5,20) dan dapat menampung 1000 data `REAL`. Jika alokasi variabel array omega tersebut tidak berhasil, misalkan, karena tidak tersedia cukup memori komputer pada saat perintah itu dijalankan, maka eksekusi program berhenti.

Apabila data yang disimpannya tidak lagi diperlukan, maka variabel array omega dapat didealokasikan dengan perintah `DEALLOCATE` sebagai berikut:

```
DEALLOCATE(omega)
```

Perhatikan bahwa pada perintah `DEALLOCATE` bentuk variabel array tidak dicantumkan, cukup namanya saja. Apabila dealokasi variabel array omega itu tidak berhasil, misalkan, karena variabel itu sebelumnya tidak dialokasikan, maka eksekusi program berhenti.

Jika variabel array omega perlu dialokasikan lagi, dijalankan kembali perintah `ALLOCATE` sesuai keperluan, misalkan saja:

```
ALLOCATE(omega(5,5,5))
```

Kini variabel array omega memiliki bentuk (5,5,5) dan dapat menampung data `REAL` sebanyak 125 buah.

Status alokasi variabel array omega dapat diketahui dengan menggunakan satu fungsi intrinsik Fortran 90/95, yaitu `ALLOCATED`, sebagai berikut:

```
ALLOCATED(omega)
```

Hasilnya adalah `.TRUE.`, jika variabel array omega masih dalam keadaan teralokasikan, atau `.FALSE.`, jika variabel tersebut sedang dalam keadaan tidak teralokasikan.

Dalam Fortran 95, tapi tidak dalam Fortran 90, sebuah variabel array teralokasikan, yang sempat dialokasikan tetapi belum didealokasikan, akan secara otomatis didealokasikan ketika eksekusi program kembali dari suatu subprogram. Hal ini berlaku untuk variabel array teralokasikan yang tidak memiliki atribut `SAVE` (atribut `SAVE` dijelaskan di Subbab 9.4) dan merupakan:

- variabel lokal dan bukan argumen *dummy* subprogram tersebut atau
- variabel lokal suatu modul, yang pada saat itu hanya digunakan oleh subprogram tersebut.

Akan tetapi, sebagai suatu kebiasaan yang baik kita dapat melakukan secara eksplisit perintah `DEALLOCATE` untuk semua variabel teralokasikan tersebut di atas, yang sempat dialokasikan namun belum didealokasikan, sebelum perintah `RETURN` di suatu subprogram.

9.3 Argumen *Dummy* Pilihan

Misalkan sebuah subprogram, baik subrutin maupun fungsi, memiliki tiga argumen *dummy*, yaitu secara berurutan dummy1, dummy2, dummy3. Namun, hanya data untuk dummy1 dan dummy2 yang secara minimal diperlukan, sedangkan data untuk dummy3 bersifat tambahan, boleh tidak ada. Dengan demikian, saat subprogram itu dipanggil argumen dummy1 dan dummy2 masing-masing harus dihubungkan dengan satu argumen aktual, sedangkan argumen dummy3 tidak harus dihubungkan dengan satu argumen aktual. Dalam hal ini dummy3 merupakan argumen *dummy* pilihan (*optional*).

Sebuah subprogram dapat memiliki lebih dari satu argumen *dummy* pilihan. Bahkan sebuah subprogram dapat memiliki argumen *dummy* yang semuanya bersifat pilihan. Argumen *dummy* pilihan dideklarasikan dengan atribut `OPTIONAL`. Contoh:

```
REAL,INTENT(IN),OPTIONAL :: dummy3
```

Tidak hanya dengan atribut `INTENT(IN)`, atribut `OPTIONAL` juga dapat dikombinasikan dengan atribut `INTENT(OUT)` maupun `INTENT(INOUT)`.

Misalkan sebuah subrutin yang diberi nama suboptional memiliki lima argumen *dummy*, yaitu secara berurutan dummy1, dummy2, dummy3, dummy4, dummy5. Argumen dummy1 dan dummy2 tidak bersifat pilihan, sedangkan dummy3, dummy4, dan dummy5 bersifat pilihan (catatan: usahakan untuk menempatkan semua argumen *dummy* pilihan setelah semua argumen *dummy* bukan pilihan). Jika dummy3, dummy4, dan dummy5 tidak diperlukan, subrutin suboptional dipanggil sebagai berikut:

```
CALL suboptional(aktual1,aktual2)
```

Berdasarkan posisi argumen, dummy1 dihubungkan dengan aktual1 dan dummy2 dengan aktual2. Jika dummy3 dan dummy4 diperlukan, subrutin suboptional dipanggil sebagai berikut:

```
CALL suboptional(aktual1,aktual2,aktual3,aktual4)
```

Di sini dummy1 dihubungkan dengan aktual1, dummy2 dengan aktual2, dummy3 dengan aktual3, dan dummy4 dengan aktual4. Kini, bagaimana jika dummy4 diperlukan, sedangkan dummy3 dan dummy5 tidak dibutuhkan? Subrutin suboptional TIDAK dapat dipanggil seperti berikut ini:

```
CALL suboptional(aktual1,aktual2,,aktual4)
```

Untuk kasus seperti yang terakhir di atas argumen *dummy* dapat dihubungkan dengan argumen aktual bukan berdasarkan posisi, melainkan dengan menggunakan kata kunci. Kata kunci tersebut adalah nama argumen *dummy* itu. Untuk kasus terakhir di atas subrutin suboptional dapat dipanggil sebagai berikut:

```
CALL suboptional(aktual1,aktual2,dummy4=aktual4)
```

Cara yang menggunakan kata kunci berupa nama argumen *dummy* tersebut tidak hanya berlaku untuk argumen *dummy* pilihan, melainkan juga untuk argumen *dummy* bukan pilihan. Pemanggilan subrutin suboptional di bawah ini sama dengan yang terakhir di atas:

```
CALL suboptional(dummy1=aktual1,dummy2=aktual2,dummy4=aktual4)
```

Setelah cara dengan kata kunci digunakan, tidak berlaku lagi cara yang berdasarkan posisi argumen. Pemanggilan subrutin suboptional di atas tidak bisa dilakukan seperti berikut ini:

```
CALL suboptional(dummy1=aktual1,aktual2,dummy4=aktual4)
```

namun, dapat dilakukan sebagai berikut:

```
CALL suboptional(aktual1,dummy2=aktual2,dummy4=aktual4)
```

Dengan digunakannya kata kunci posisi argumen menjadi tidak lagi berperan. Pemanggilan subrutin suboptional di bawah ini sama dengan yang terakhir di atas:

```
CALL suboptional(aktual1,dummy4=aktual4,dummy2=aktual2)
```

Jika sebuah subprogram memiliki argumen *dummy* pilihan, maka saat subprogram itu dipanggil sebuah argumen aktual mungkin diberikan atau mungkin tidak diberikan untuk argumen *dummy* pilihan tersebut. Pada kasus yang pertama argumen *dummy* pilihan itu dianggap ada dan pada kasus yang kedua dianggap tidak ada. Untuk mengetahui apakah sebuah argumen *dummy* pilihan ada atau tidak ada tersedia satu fungsi intrinsik Fortran 90/95, yaitu `PRESENT`. Sebagai contoh, untuk mengetahui apakah argumen dummy5 ada atau tidak setelah subprogram suboptional dipanggil, dijalankan perintah berikut:

```
PRESENT(dummy5)
```

Perintah tersebut akan memberikan nilai logika `.TRUE.` jika dummy5 ada atau `.FALSE.` jika dummy5 tidak ada. Dengan fungsi `PRESENT` ini dimungkinkan untuk membuat konstruksi (atau juga pernyataan) `IF` di dalam subprogram semacam contoh berikut:

```
IF (PRESENT(dummy5)) THEN
   [pekerjaan]
END IF
```

9.4 Atribut `SAVE` untuk Variabel Lokal

Nilai yang tersimpan dalam variabel lokal sebuah subprogram terhapus ketika eksekusi program telah kembali dari subprogram tersebut setelah bertemu perintah `RETURN`. Begitu pula, nilai yang tersimpan dalam variabel lokal sebuah modul terhapus ketika eksekusi program telah kembali dari suatu subprogram, yang pada saat itu menjadi satu-satunya pengguna modul itu. Apabila diinginkan satu atau beberapa nilai tidak terhapus, melainkan tetap tersimpan dalam satu atau beberapa variabel lokal tersebut, maka variabel-variabel lokal itu harus memiliki atribut `SAVE`.

Ilustrasi sederhana berikut dapat memberikan gambaran satu contoh manfaat atribut `SAVE`. Misalkan sebuah subprogram dibuat untuk menghitung nilai z = x + y, dengan x dan y merupakan variabel lokal, namun bukan argumen *dummy*, subprogram itu. Untuk itu mula-mula subprogram tersebut menghitung nilai x dan y, kemudian nilai z. Anggap

subprogram itu dipanggil dua kali dalam suatu program. Kebetulan, pada pemanggilan kedua nilai argumen aktual subprogram itu menghasilkan nilai x yang sama seperti pada pemanggilan pertama. Dalam hal ini, sebetulnya nilai x tidak perlu dihitung lagi, melainkan dapat langsung digunakan untuk menghitung z. Hal itu dapat dilakukan apabila variabel lokal x memiliki atribut `SAVE`, sehingga nilainya pada pemanggilan pertama subprogram masih tetap tersimpan dan dapat langsung digunakan pada pemanggilan kedua.

Secara implisit atribut `SAVE` dapat diberikan pada sebuah variabel lokal dengan memberikan suatu nilai awal pada variabel itu ketika dideklarasikan di dalam suatu subprogram atau modul. Variabel lokal tersebut bisa dari jenis skalar maupun array rank 1. Jika variabel itu dari jenis array rank 1, maka pemberian nilai awal dilakukan dengan menggunakan perintah pembentuk array `(/ [elemen data array] /)` (lihat Subbab 3.2). Contoh:

```
INTEGER, DIMENSION(5) :: nn=(/ 10,20,30,40,50 /)
REAL :: xx=0.0
```

Perhatikan bahwa nn dan xx bukanlah konstanta, melainkan variabel, yang dalam hal ini sudah langsung menyimpan suatu nilai ketika dideklarasikan dan, karena itu, memiliki atribut `SAVE`.

Atribut `SAVE` juga dapat diberikan secara eksplisit pada sebuah variabel lokal ketika dideklarasikan. Selain untuk variabel dari jenis skalar dan array, pencantuman atribut `SAVE` secara eksplisit juga dapat dilakukan pada deklarasi variabel lokal dari jenis array teralokasikan. Contoh:

```
INTEGER, SAVE :: aa
REAL, DIMENSION(n,m), SAVE :: bb
CHARACTER, DIMENSION(:,:,:), ALLOCATABLE, SAVE :: cc
```

Deklarasi tersebut di atas membuat variabel aa, bb, dan cc memiliki atribut `SAVE`, sehingga masih tetap menyimpan nilainya yang terakhir, walaupun eksekusi program telah kembali dari subprogram tempat variabel itu dideklarasikan atau dari subprogram yang merupakan satu-satunya pengguna modul tempat variabel itu dideklarasikan.

9.5 Subprogram Eksternal

Dalam subbab ini kami sampaikan beberapa hal khusus mengenai subprogram eksternal. Selain dari yang disampaikan di sini, hal-hal yang dijelaskan dalam Bab 7 mengenai subprogram berlaku juga untuk subprogram eksternal.

Sebuah subprogram eksternal dapat berisi subprogram lain di dalamnya, disebut subprogram internal. Struktur subprogram eksternal secara umum sebagai sebuah subrutin atau fungsi masing-masing adalah sebagai berikut:

```
SUBROUTINE [nama subrutin] ([argumen])
[bagian spesifikasi]
[bagian perintah & pernyataan subrutin]
CONTAINS
[subprogram internal 1]
[subprogram internal 2]
...
END SUBROUTINE [nama subrutin]
```

```
FUNCTION [nama fungsi] ([argumen]) RESULT([variabel hasil])
[bagian spesifikasi]
[bagian perintah & pernyataan fungsi]
CONTAINS
[subprogram internal 1]
[subprogram internal 2]
...
END FUNCTION [nama fungsi]
```

Jika subprogram eksternal tersebut merupakan subprogram rekursif, maka awalan `RECURSIVE` ditambahkan pada pernyataan subprogram itu (`SUBROUTINE` atau `FUNCTION`), seperti disampaikan di Subbab 7.4.

Subprogram eksternal ditempatkan di luar program utama maupun modul. Subprogram itu dapat ditaruh dalam satu file bersama unit program lain (program utama, subprogram eksternal, modul) yang memakainya maupun dalam file tersendiri, terpisah dari file yang berisi unit program lain tersebut (lihat Lampiran A mengenai kompilasi dan penautan program yang terdiri dari lebih dari satu file). Jika ditaruh dalam satu file bersama unit program lain yang memakainya, subprogram eksternal dapat ditempatkan sebelum atau sesudah unit program itu. Di Sublampiran C.12, C.13, dan C.14 ditunjukkan contoh-contoh program yang menggunakan subprogram eksternal.

Pada saat suatu unit program dikompilasi, untuk mudahnya kita bisa bilang bahwa *compiler* "tidak melihat" subprogram eksternal yang dipakai di dalam unit program itu, karena subprogram eksternal tersebut berada di luar unit program itu, meskipun keduanya mungkin saja berada dalam satu file yang sama. Oleh karena itu, di dalam unit program tersebut harus diberikan informasi yang diperlukan oleh *compiler* mengenai subprogram eksternal yang dipakai. Informasi tersebut diberikan dalam blok `INTERFACE`.

Blok `INTERFACE` ditempatkan di bagian spesifikasi suatu unit / subunit program, diawali oleh pernyataan `INTERFACE` dan diakhiri oleh `END INTERFACE`, seperti digambarkan berikut ini:

```
...
IMPLICIT NONE
[deklarasi variabel dan konstanta]
INTERFACE
[isi blok interface]
```

```
END INTERFACE
...
```

Blok INTERFACE sesungguhnya dapat digunakan untuk berbagai keperluan. Di sini kami sampaikan pemakaiannya untuk menyertakan sebuah subprogram eksternal dalam suatu unit program.

Blok INTERFACE untuk suatu subprogram eksternal dinyatakan di bagian spesifikasi program utama, subprogram eksternal lain, modul, subprogram modul, atau subprogram internal (yang tidak berada di dalam subprogram eksternal itu) yang memerlukannya. Informasi mengenai subprogram eksternal yang diberikan dalam blok INTERFACE antara lain nama subprogram, argumen-argumen *dummy*-nya beserta tipe, jenis, urutannya, variabel dan konstanta yang diperlukan. Singkatnya, isi blok INTERFACE untuk suatu subprogram eksternal seperti salinan sebagian, tidak semua, bagian kepala (*header*) subprogram eksternal tersebut. Kita lihat berikut ini contoh blok INTERFACE untuk beberapa subprogram yang telah ditampilkan sebelumnya.

- Blok INTERFACE untuk subrutin subfortranku (Subbab 7.2):

```
INTERFACE
 SUBROUTINE subfortranku(x,y,z)
 IMPLICIT NONE
 REAL, INTENT(IN) :: x,y
 REAL, INTENT(OUT) :: z
 END SUBROUTINE subfortranku
END INTERFACE
```

- Blok INTERFACE untuk fungsi funcfortranku (Subbab 7.3):

```
INTERFACE
 FUNCTION funcfortranku(x) RESULT(z)
 IMPLICIT NONE
 REAL,INTENT(IN) :: x
 REAL :: z
 END FUNCTION funcfortranku
END INTERFACE
```

- Blok INTERFACE untuk fungsi rekursif faktorial (Subbab 7.4):

```
INTERFACE
 RECURSIVE FUNCTION faktorial(n) RESULT(m)
 IMPLICIT NONE
 INTEGER, INTENT(IN) :: n
 INTEGER :: m
 END FUNCTION faktorial
END INTERFACE
```

Nama subprogram yang disebutkan di dalam blok `INTERFACE` harus sama dengan nama subprogram, yang untuk subprogram tersebut blok `INTERFACE` itu dibuat. Namun, nama variabel dan konstanta boleh berbeda. Contoh, nama variabel x, y, z dalam blok `INTERFACE` untuk subrutin subfortranku di atas dapat diganti dengan u, v, w, sehingga blok `INTERFACE` untuk subrutin subfortranku menjadi:

```
INTERFACE
 SUBROUTINE subfortranku(u,v,w)
 IMPLICIT NONE
 REAL, INTENT(IN) :: u,v
 REAL, INTENT(OUT) :: w
 END SUBROUTINE subfortranku
END INTERFACE
```

9.6 Modul

Mula-mula mari kita lihat struktur modul. Modul memiliki struktur yang dapat diringkas sebagai berikut:

```
MODULE [nama modul]
[bagian spesifikasi]
CONTAINS
[subprogram modul 1]
[subprogram modul 2]
...
END MODULE [nama modul]
```

Modul tidak memiliki bagian yang berisi perintah & pernyataan modul. Subprogram yang berada di dalam modul kita sebut saja subprogram modul.

Sepintas struktur modul tampak sederhana. Namun, subprogram modul memiliki struktur seperti subprogram eksternal, yaitu dapat berisi subprogram internal. Jika ditampilkan secara lebih detil struktur modul bisa tampak seperti berikut:

```
MODULE [nama modul]
[bagian spesifikasi]
CONTAINS
SUBROUTINE [nama subrutin] ([argumen])
[bagian spesifikasi]
[bagian perintah & pernyataan subrutin]
CONTAINS
[subprogram internal 1]
[subprogram internal 2]
...
```

```
END SUBROUTINE [nama subrutin]
...
FUNCTION [nama fungsi] ([argumen]) RESULT([variabel hasil])
[bagian spesifikasi]
[bagian perintah & pernyataan fungsi]
CONTAINS
[subprogram internal 1]
[subprogram internal 2]
...
END FUNCTION [nama fungsi]
...
END MODULE [nama modul]
```

Awalan `RECURSIVE` tentu saja dapat ditambahkan di pernyataan subprogram yang bersifat rekursif.

Sebuah modul dapat berisi tiga hal berikut:

- variabel dan konstanta yang dideklarasikan di bagian spesifikasi modul,
- blok `INTERFACE` yang dinyatakan di bagian spesifikasi modul,
- subprogram modul.

Tiga hal tersebut di atas dapat dipakai atau dikenal oleh unit / subunit program lain (program utama, subprogram eksternal, modul, subprogram modul, subprogram internal). Caranya adalah dengan memberikan perintah

```
USE [nama modul]
```

di awal bagian spesifikasi unit / subunit program lain tersebut, sebelum deklarasi variabel dan konstanta, serta pernyataan blok `INTERFACE`. Contoh, misalkan program utama fortranku menggunakan modul yang bernama modulku, maka bagian kepala program utama fortranku tampak seperti berikut:

```
PROGRAM fortranku

USE modulku

IMPLICIT NONE
...
```

Sesuai keperluan mungkin saja sebagian variabel dan konstanta yang dideklarasikan di bagian spesifikasi modul dibuat untuk berlaku hanya di dalam modul tersebut, tidak untuk dikenal dan dipakai oleh unit / subunit program lain. Dalam hal ini variabel dan konstanta itu harus diberi atribut `PRIVATE`. Perhatikan contoh berikut, anggaplah itu diambil dari bagian spesifikasi sebuah modul:

```
INTEGER, PARAMETER :: dpr=KIND(1.0D0)
INTEGER, PRIVATE :: i,j
REAL(dpr), PRIVATE :: p,q
```

Pada contoh itu konstanta dpr berlaku di dalam modul tersebut dan di semua unit / subunit program lain yang memakai modul itu, sedangkan variabel i, j, p, dan q hanya berlaku di dalam modul tersebut, untuk keperluan internal.

Modul, dengan demikian, menjadi suatu alternatif yang baik apabila beberapa unit / subunit program memakai secara bersama beberapa variabel, konstanta, blok `INTERFACE`, maupun subprogram. Variabel, konstanta, blok `INTERFACE`, dan subprogram tersebut cukup ditaruh di dalam sebuah modul, yang kemudian dengan perintah singkat `USE [nama modul]` dapat dipakai oleh unit / subunit program lain. Sebuah modul bisa saja dimanfaatkan untuk menyimpan hanya variabel dan konstanta, hanya blok `INTERFACE`, atau hanya subprogram, yang dipakai secara bersama oleh beberapa unit / subunit program. Di Sublampiran C.13 dan C.14 kami berikan beberapa contoh program yang menggunakan modul.

9.7 Subprogram sebagai Argumen

Sebagaimana disampaikan di Subbab 7.1 argumen aktual bagi suatu subprogram dapat berupa variabel, konstanta, atau ekspresi data. Sesungguhnya lebih dari itu argumen aktual dapat juga berupa sebuah subprogram. Subprogram yang dapat menjadi argumen aktual adalah subprogram eksternal dan subprogram modul. Contoh-contoh program yang menunjukkan penggunaan suatu subprogram sebagai argumen aktual bagi subprogram lain diberikan di Sublampiran C.14.

Argumen aktual yang berupa subprogram itu dihubungkan dengan argumen *dummy*, yang tentunya juga berupa subprogram. Deklarasi argumen *dummy* yang berupa subprogram tidak seperti deklarasi variabel, melainkan menggunakan blok `INTERFACE`. Blok `INTERFACE` untuk subprogram yang menjadi argumen *dummy* tersebut sama seperti blok `INTERFACE` untuk subprogram yang menjadi argumen aktual, hanya saja nama subprogram dan nama variabel serta konstanta di dalamnya boleh berbeda. Contoh, misalkan ada satu fungsi eksternal atau fungsi modul yang bernama polinomial sebagai berikut:

```
FUNCTION polinomial(x,y) RESULT(z)
INTEGER, PARAMETER :: dpr=KIND(1.0D0)
REAL(dpr), INTENT(IN) :: x,y
REAL(dpr) :: z
...
END FUNCTION polinomial
```

Fungsi polinomial itu menjadi argumen aktual bagi, misalkan saja, subrutin soalmath, yang dihubungkan dengan argumen *dummy* bernama farg, sehingga di dalam subrutin

soalmath fungsi polinomial dijalankan dengan memanggil fungsi farg. Argumen *dummy* farg dideklarasikan di bagian spesifikasi subrutin soalmath dengan menggunakan blok `INTERFACE` sebagai berikut:

```
SUBROUTINE soalmath(a,b,c,farg)
INTEGER, PARAMETER :: dpr=KIND(1.0D0)
REAL(dpr), INTENT(IN) :: a,b
REAL(dpr), INTENT(OUT) :: c

INTERFACE
 FUNCTION farg(u,v) RESULT(w)
 INTEGER, PARAMETER :: dpr=KIND(1.0D0)
 REAL(dpr), INTENT(IN) :: u,v
 REAL(dpr) :: w
 END FUNCTION farg
END INTERFACE
...
END SUBROUTINE soalmath
```

Atribut `INTENT` tidak perlu dan tidak bisa diberikan untuk argumen *dummy* yang berupa subprogram. Namun, argumen *dummy* yang berupa subprogram dapat memiliki atribut `OPTIONAL`, yang artinya dapat bersifat pilihan, seperti juga argumen *dummy* yang berupa variabel. Dalam hal ini atribut `OPTIONAL` diberikan tidak dengan cara yang ditunjukkan di Subbab 9.3, melainkan dengan pernyataan `OPTIONAL` di bagian spesifikasi subprogram sebagai berikut:

```
OPTIONAL :: [daftar argumen dummy pilihan]
```

Secara umum, pernyataan `OPTIONAL` tersebut dapat dipakai tidak hanya untuk argumen *dummy* yang berupa subprogram, melainkan juga untuk yang berupa variabel. Pernyataan `OPTIONAL` dapat dicantumkan pada baris sebelum atau sesudah deklarasi argumen *dummy* pilihan. Jika sebuah argumen *dummy* pilihan dicantumkan pada pernyataan `OPTIONAL`, maka argumen *dummy* pilihan itu tidak perlu lagi mendapat atribut `OPTIONAL` (Subbab 9.3) ketika dideklarasikan; begitu juga sebaliknya. Dengan demikian, tidak terjadi pemberian ganda atribut `OPTIONAL` pada sebuah argumen *dummy* pilihan.

Lampiran

Lampiran A

Program Komputer serta Cara Kompilasi dan Penautan Program

Sebuah program komputer adalah sekumpulan perintah untuk dikerjakan oleh komputer. Perintah-perintah itu disusun mengikuti aturan bahasa pemrograman yang digunakan, contohnya Fortran 90/95. Sesuai bahasa pemrograman yang dipakai perintah-perintah itu disampaikan tidak dengan kalimat-kalimat yang biasa kita pakai dalam berkomunikasi antar sesama manusia, melainkan dalam bentuk kode. Oleh karena itu, program komputer juga biasa disebut sebagai kode sumber (*source code*) atau kadang disingkat sebagai kode.

Program komputer disimpan dalam sebuah file, yang formatnya adalah teks biasa (*plain text*). Sekedar untuk memberi gambaran seperti apa format teks biasa, ambillah file yang namanya memiliki akhiran (*extension*) ".txt", karena file seperti itu umumnya memiliki format teks biasa. Namun, sebuah file yang formatnya teks biasa tidak harus memiliki nama dengan akhiran ".txt", melainkan apa saja atau bahkan tanpa akhiran, karena nama file tidak menentukan format file. Dalam hal ini, akhiran nama file program komputer ditentukan oleh bahasa pemrograman yang dipakai. Nama file program komputer yang disusun dalam bahasa Fortran 90/95, disingkat file program Fortran 90/95, diberi akhiran ".f90". Nama file tersebut juga tidak harus sama dengan nama program yang disimpan di dalamnya. Untuk menyusun atau mengedit sebuah program komputer tidak diperlukan suatu program atau perangkat lunak (*software*) editor teks khusus. Sembarang editor teks bisa dipakai asalkan dapat menyimpan program komputer sebagai file berformat teks biasa.

Sesungguhnya kode-kode yang diatur menurut suatu bahasa pemrograman tidak dapat dipahami secara langsung oleh komputer. Bahasa pemrograman dibuat sebagai bahasa yang lebih mudah dimengerti oleh kita manusia untuk menyampaikan perintah kepada komputer. Pada akhirnya, kode-kode yang termuat dalam sebuah program komputer harus diproses, atau istilahnya dikompilasi (*compile*), sehingga menjadi perintah-perintah yang dipahami oleh komputer. Kompilasi dilakukan oleh suatu perangkat lunak yang disebut sebagai *compiler*. *Compiler* kemudian juga menautkan (*link*) bagian-bagian program yang disimpan dalam satu atau beberapa file menjadi satu file yang dapat dijalankan (*executable file*). Proses ini seringkali juga disebut sebagai pembuatan (*build*),

karena pada proses ini dibuat sebuah file yang dapat dijalankan. *Compiler* dapat ditemukan di internet, khususnya *compiler* Fortran 90/95, baik yang berbayar, maupun yang tak berbayar.

Berikut ini kita lihat cara mengkompilasi dan menautkan program. Anggaplah kita menggunakan *compiler* yang bernama "comfort". Kita bekerja dalam modus teks. Untuk itu, kita jalankan *command-line interface*, yang secara praktis adalah kita buka sebuah terminal (linux) atau *command prompt* (windows). Dalam terminal maupun *command prompt* itu kita dapat tuliskan perintah, yang kemudian dijalankan dengan menekan tombol `ENTER`.

Ambillah sebuah file program Fortran 90/95 yang bernama prosatu.f90. Untuk mengkompilasi dan menautkan program yang disimpan dalam prosatu.f90 dijalankan perintah berikut dalam terminal atau *command prompt*:

```
comfort prosatu.f90
```

Perintah di atas menghasilkan sebuah *executable file* bernama a.out (linux) atau a.exe (windows). Untuk menjalankannya ketik a.out di terminal atau a.exe (atau cukup a) di *command prompt*, lalu tekan tombol `ENTER`.

Jika cara di atas menghasilkan *executable file* a.out atau a.exe, cara kompilasi dan penautan berikut ini menghasilkan *executable file*, yang namanya dapat kita tentukan dengan menggunakan opsi *compiler* (*compiler option*) -o. Contoh, di terminal kita dapat jalankan perintah

```
comfort -o bagus prosatu.f90
```

untuk menghasilkan *executable file* bernama bagus, di *command prompt* dapat kita jalankan perintah

```
comfort -o bagus.exe prosatu.f90
```

untuk menghasilkan *executable file* bernama bagus.exe.

Apabila program komputer yang akan dikompilasi dan ditautkan melibatkan lebih dari satu file, misalkan prosatu.f90, produa.f90, dan protiga.f90, maka nama-nama file itu dituliskan berturut-turut. Contoh, perintah

```
comfort prosatu.f90 produa.f90 protiga.f90
```

menghasilkan *executable file* a.out (linux) atau a.exe (windows), perintah di terminal

```
comfort -o bagus prosatu.f90 produa.f90 protiga.f90
```

menghasilkan *executable file* bagus, dan perintah di *command prompt*

```
comfort -o bagus.exe prosatu.f90 produa.f90 protiga.f90
```

menghasilkan *executable file* bagus.exe. Urutan nama file yang dituliskan bebas, KECUALI bahwa nama file yang berisi modul (lihat Subbab 9.6) harus dituliskan lebih dulu dari nama file yang berisi unit / subunit program, yang menggunakan modul tersebut. Dengan kata lain, modul harus lebih dulu dikompilasi daripada unit / subunit program yang menggunakannya. Apabila, misalkan, file protiga.f90 berisi modul yang dipakai oleh unit / subunit program yang disimpan dalam file produa.f90, maka semua yang di bawah ini merupakan urutan penulisan nama file yang benar:

```
prosatu.f90 protiga.f90 produa.f90
protiga.f90 prosatu.f90 produa.f90
protiga.f90 produa.f90 prosatu.f90
```

Kompilasi dan penautan dapat dilakukan secara terpisah. Kompilasi dilakukan dengan memberikan opsi *compiler* -c dan menghasilkan file obyek (namanya berakhiran ".o"). Contoh,

```
comfort -c prosatu.f90 produa.f90 protiga.f90
```

menghasilkan file-file obyek prosatu.o, produa.o, dan protiga.o. Kompilasi ini juga dapat dilakukan untuk tiap file secara sendiri-sendiri, yaitu

```
comfort -c prosatu.f90
comfort -c produa.f90
comfort -c protiga.f90
```

Perhatikan bahwa urutan penulisan nama file maupun urutan kompilasi yang sesungguhnya harus sesuai dengan pemakaian modul di dalam program, yaitu modul harus dikompilasi lebih dulu daripada unit / subunit program yang menggunakannya. Kemudian, penautan dilakukan sebagai berikut, misalkan, untuk menghasilkan *executable file* bagus di linux:

```
comfort -o bagus prosatu.o produa.o protiga.o
```

atau bagus.exe di windows:

```
comfort -o bagus.exe prosatu.o produa.o protiga.o
```

Pada penautan urutan penulisan nama file bebas, tidak bergantung pada pemakaian modul di dalam program. Jadi, nama file yang berisi modul tidak harus berada lebih dulu dari nama file yang berisi unit / subunit program yang menggunakan modul itu.

Bahwa kompilasi dan penautan dapat dilakukan terpisah, ini memungkinkan orang melakukan hal-hal sebagai berikut, contoh:

- Anggaplah prosatu.f90 berisi program utama. Anggap juga sudah dilakukan kompilasi, sehingga file-file obyek prosatu.o, produa.o, protiga.o sudah ada. Kemudian, orang lakukan perubahan pada program utama dalam file prosatu.f90. Untuk menghasilkan *executable file* orang harus mengkompilasi ulang cukup hanya prosatu.f90:

  ```
  comfort -c prosatu.f90
  ```

  dilanjutkan dengan melakukan penautan ulang, di linux:

  ```
  comfort -o bagus prosatu.o produa.o protiga.o
  ```

  atau di windows:

  ```
  comfort -o bagus.exe prosatu.o produa.o protiga.o
  ```

- Beberapa modul dan subprogram eksternal (lihat Subbab 9.5) yang umum dipakai dapat dikompilasi dan kemudian file obyeknya disimpan di suatu tempat. Kumpulan file obyek ini berfungsi sebagai perpustakaan (*library*). Jika suatu program memerlukan modul dan / atau subprogram eksternal tersebut, maka pada saat penautan file-file obyeknya atau *library* itu dapat disertakan. Pada pusat-pusat komputasi biasanya sudah diatur cara menyertakan *library* pada proses penautan program.

Lampiran B

Beberapa Subprogram Intrinsik Fortran 90/95

Fortran 90/95 menyediakan banyak subprogram intrinsik, baik berupa subrutin maupun fungsi. Di sini kami berikan beberapa subprogram intrinsik, khususnya fungsi, yang menurut kami biasa diperlukan. Subprogram-subprogram intrinsik tersebut kami sampaikan dalam kelompok dan secara alfabetis.

Nama argumen yang disebutkan dalam daftar subprogram intrinsik di bawah merupakan nama argumen *dummy* subprogram tersebut. Dengan demikian, jika diperlukan subprogram itu dapat dipanggil dengan menggunakan kata kunci yang menghubungkan argumen *dummy* dengan argumen aktual, seperti dijelaskan di Subbab 9.3.

Beberapa subprogram intrinsik Fortran 90/95 memiliki argumen *dummy* pilihan (lihat Subbab 9.3). Bahkan ada yang semua argumen *dummy*-nya bersifat pilihan. Argumen *dummy* pilihan kami tuliskan di sini dalam tanda kurung { } hanya untuk membedakannya dari argumen *dummy* bukan pilihan. Perhatikan bahwa dalam pemakaiannya tanda kurung { } tidak dicantumkan.

B.1 Subrutin Intrinsik

`CPU_TIME(TIME)`

Argumen `TIME` bertipe `REAL` skalar. Subrutin ini memberikan suatu nilai pendekatan waktu prosesor (*processor time*) dalam detik untuk `TIME`. Nilai yang diberikan bergantung pada prosesor (*processor dependent*). Jika subrutin ini dipanggil di dua tempat berbeda dalam suatu program, maka dapat dihitung waktu yang diperlukan komputer untuk menjalankan program di antara dua pemanggilan subrutin tersebut.

`DATE_AND_TIME({DATE},{TIME},{ZONE},{VALUES})`

Argumen `DATE`, `TIME`, `ZONE` bertipe `CHARACTER` skalar dan argumen `VALUES` bertipe `INTEGER` array berbentuk (8). Subrutin ini memberikan:

- untuk `DATE` data tanggal dalam format ccyymmdd (cc = abad, yy = tahun, mm = bulan, dd = hari),

- untuk `TIME` data waktu dalam format hhmmss.sss (hh = jam, mm = menit, ss = detik, sss = milidetik),
- untuk `ZONE` data beda waktu antara waktu setempat dan UTC (dikenal juga dengan GMT) dalam format Shhmm (S = tanda + atau -, hh = jam, mm = menit),
- untuk `VALUES` 8 elemen data, yaitu secara berurutan tahun, bulan, hari, beda waktu antara waktu setempat dan UTC dalam menit, jam, menit, detik, dan milidetik.

`RANDOM_NUMBER(HARVEST)`

Argumen `HARVEST` bertipe `REAL` dan jenisnya bisa skalar maupun array. Subrutin ini memberikan ke `HARVEST` sebuah atau array bilangan acak "palsu" (*pseudorandom number*), yang rentang nilainya dari 0 sampai 1.

`RANDOM_SEED({SIZE})`, `RANDOM_SEED({PUT})`, `RANDOM_SEED({GET})`

Bilangan acak "palsu" dihasilkan dari suatu benih berupa sebuah array rank 1 bilangan bulat, yang ukuran arraynya bergantung pada prosesor. Subrutin `RANDOM_SEED` digunakan untuk mendapatkan dan membuat benih tersebut. Subrutin ini menerima tak lebih dari satu argumen saja, yaitu salah satu dari `SIZE`, `PUT`, dan `GET`. Argumen `SIZE` bertipe `INTEGER` skalar, sedangkan `PUT` dan `GET` bertipe `INTEGER` array rank 1, yang ukurannya sama dengan ukuran array benih. Subrutin `RANDOM_SEED`:

- memberikan untuk `SIZE` ukuran array benih, yang dipakai untuk menentukan ukuran array `PUT` dan `GET`,
- mengambil dari `PUT` suatu array rank 1 bilangan bulat, yang dipakai oleh prosesor untuk membuat benih,
- memberikan untuk `GET` benih yang dibuat atau dipakai prosesor pada saat itu.

`SYSTEM_CLOCK({COUNT},{COUNT_RATE},{COUNT_MAX})`

Argumen `COUNT`, `COUNT_RATE`, `COUNT_MAX` bertipe `INTEGER` skalar. Subrutin ini memberikan:

- untuk `COUNT` suatu nilai berdasarkan nilai jam prosesor (*processor clock*) saat itu; nilai ini bergantung pada prosesor (*processor dependent*),
- untuk `COUNT_RATE` jumlah hitungan jam prosesor (*processor clock counts*) per detik,
- untuk `COUNT_MAX` nilai maksimum yang mungkin untuk `COUNT`.

B.2 Fungsi Intrinsik

B.2.1 Fungsi Matematis

Argumen fungsi-fungsi berikut bisa berupa skalar maupun array. Jika argumen itu array, maka fungsi ini bekerja untuk tiap elemen argumen array tersebut dan luaran fungsi berupa array, yang bentuknya sama dengan bentuk array argumennya. Tipe data dan *kind type parameter* luaran fungsi sesuai dengan argumennya.

`ACOS(X)`
: Argumen `X` bertipe `REAL` dan nilainya adalah -1 $\leqslant$ `X` $\leqslant$ 1. Fungsi ini memberikan nilai fungsi arkus kosinus untuk `X` dalam satuan radian, dengan rentang nilai 0 sampai π (kuadran 1 dan 2).

`ASIN(X)`
: Argumen `X` bertipe `REAL` dan nilainya adalah -1 $\leqslant$ `X` $\leqslant$ 1. Fungsi ini memberikan nilai fungsi arkus sinus untuk `X` dalam satuan radian, dengan rentang nilai $-\pi/2$ sampai $\pi/2$ (kuadran 4 dan 1).

`ATAN(X)`
: Argumen `X` bertipe `REAL`. Fungsi ini memberikan nilai fungsi arkus tangen untuk `X` dalam satuan radian, dengan rentang nilai $-\pi/2$ sampai $\pi/2$ (kuadran 4 dan 1).

`COS(X)`
: Argumen `X` bertipe `REAL` atau `COMPLEX` dan nilainya dinyatakan dalam satuan radian. Fungsi ini memberikan nilai fungsi kosinus untuk `X`.

`COSH(X)`
: Argumen `X` bertipe `REAL`. Fungsi ini memberikan nilai fungsi kosinus hiperbolik untuk `X`.

`EXP(X)`
: Argumen `X` bertipe `REAL` atau `COMPLEX`. Fungsi ini memberikan nilai fungsi eksponensial untuk `X`.

`LOG(X)`
: Argumen `X` bertipe `REAL` atau `COMPLEX`. Jika `X` bilangan riil, maka `X` harus positif. Jika `X` bilangan kompleks, maka `X` tidak boleh nol. Fungsi ini memberikan nilai fungsi logaritma natural untuk `X`. Jika `X` bilangan kompleks, maka luaran fungsi memiliki komponen imajiner b yang nilainya adalah $-\pi < b \leqslant \pi$.

`LOG10(X)`
: Argumen `X` bertipe `REAL` dan harus positif. Fungsi ini memberikan nilai fungsi logaritma biasa untuk `X`.

`SIN(X)`
: Argumen `X` bertipe `REAL` atau `COMPLEX` dan nilainya dinyatakan dalam satuan radian. Fungsi ini memberikan nilai fungsi sinus untuk `X`.

`SINH(X)`
: Argumen `X` bertipe `REAL`. Fungsi ini memberikan nilai fungsi sinus hiperbolik untuk `X`.

`SQRT(X)`
: Argumen `X` bertipe `REAL` atau `COMPLEX`. Jika `X` bilangan riil, maka `X` tidak boleh negatif. Fungsi ini memberikan nilai akar kuadrat untuk `X`. Jika `X` bilangan kompleks, maka luaran fungsi memiliki komponen riil yang tidak negatif, dan jika komponen riilnya ternyata nol, maka komponen imajinernya tidak negatif.

`TAN(X)`
: Argumen `X` bertipe `REAL` dan nilainya dinyatakan dalam satuan radian. Fungsi ini memberikan nilai fungsi tangen untuk `X`.

`TANH(X)`
: Argumen `X` bertipe `REAL`. Fungsi ini memberikan nilai fungsi tangen hiperbolik untuk `X`.

B.2.2 Fungsi untuk Operasi Array

Fungsi-fungsi berikut melakukan suatu operasi pada sebuah obyek berbentuk array. Obyek tersebut disimpan dalam sebuah argumen *dummy* bukan pilihan, yang diberi nama `ARRAY`. Luaran fungsi memiliki tipe data yang sama dengan tipe data `ARRAY`, begitu juga *kind type parameter*-nya.

Semua fungsi tersebut memiliki dua argumen *dummy* pilihan, yaitu `DIM` dan `MASK`, yang mengatur secara lebih khusus kerja fungsi-fungsi itu pada `ARRAY`. Argumen `DIM` bertipe `INTEGER` skalar dan argumen `MASK` bertipe `LOGICAL` array, yang bentuk arraynya sama dengan argumen `ARRAY`. Contoh penggunaan `DIM` dan `MASK` sebagai berikut:

- Misalkan argumen `ARRAY` berdimensi 3. Jika fungsi di bawah dipanggil dengan argumen `DIM=2`, maka fungsi tersebut bekerja hanya pada dimensi ke-2 argumen `ARRAY`.

- Jika argumen `MASK` diberikan, maka fungsi itu hanya bekerja untuk elemen `ARRAY` yang bersesuaian dengan elemen `MASK`, yang bernilai `.TRUE.`. Contoh, jika pada pemanggilan fungsi di bawah dicantumkan `MASK=A>1`, dengan A merupakan argumen aktual yang dihubungkan dengan `ARRAY`, maka fungsi itu bekerja hanya untuk elemen `ARRAY` yang nilainya lebih dari 1, karena nilai elemen `MASK` yang bersesuaian dengan itu adalah `.TRUE.`.

`MAXVAL(ARRAY,{DIM},{MASK})`
: Argumen `ARRAY` bertipe `REAL` atau `INTEGER`. Fungsi ini memberikan nilai terbesar elemen `ARRAY`.

`MINVAL(ARRAY,{DIM},{MASK})`
: Argumen `ARRAY` bertipe `REAL` atau `INTEGER`. Fungsi ini memberikan nilai terkecil elemen `ARRAY`.

`PRODUCT(ARRAY,{DIM},{MASK})`

Argumen `ARRAY` bertipe `REAL`, `INTEGER`, atau `COMPLEX`. Fungsi ini memberikan hasil perkalian elemen-elemen `ARRAY`.

`SUM(ARRAY,{DIM},{MASK})`

Argumen `ARRAY` bertipe `REAL`, `INTEGER`, atau `COMPLEX`. Fungsi ini memberikan hasil penjumlahan elemen-elemen `ARRAY`.

B.2.3 Fungsi Lain

Sama seperti argumen fungsi matematis, argumen fungsi-fungsi berikut bisa berupa skalar maupun array. Jika argumen itu array, maka fungsi ini bekerja untuk tiap elemen argumen array tersebut dan luaran fungsi berupa array, yang bentuknya sama dengan bentuk array argumennya. Sebagian fungsi di bawah memiliki argumen *dummy* pilihan.

`ABS(A)`

Argumen `A` bertipe `REAL`, `INTEGER`, atau `COMPLEX`. Fungsi ini memberikan nilai mutlak `A`. Luaran fungsi bertipe `INTEGER`, jika `A` bertipe `INTEGER`, atau luaran bertipe `REAL`, jika `A` bertipe `REAL` atau `COMPLEX`, dengan *kind type parameter* sesuai `A`.

`AIMAG(Z)`

Argumen `Z` bertipe `COMPLEX`. Fungsi ini memberikan komponen imajiner `Z`, dengan tipe `REAL` dan *kind type parameter* sesuai `Z`.

`CMPLX(X,{Y},{KIND})`

Argumen `X` dan `Y` bertipe `REAL` atau `INTEGER`, apabila `Y` ada. Jika `Y` tidak ada, maka argumen `X` juga dapat bertipe `COMPLEX`. Fungsi ini memberikan bilangan bertipe `COMPLEX` berdasarkan argumennya `X` dan `Y` (jika ada) sebagai berikut:

- Jika `Y` tidak ada dan `X` bertipe bukan `COMPLEX`, maka dihasilkan bilangan kompleks yang memiliki komponen riil `X` dan komponen imajiner nol.
- Jika `Y` tidak ada dan `X` bertipe `COMPLEX`, maka dihasilkan bilangan kompleks `X`.
- Jika `Y` ada, maka dihasilkan bilangan kompleks yang memiliki komponen riil `X` dan komponen imajiner `Y`.

Jika argumen `KIND` diberikan, maka luaran fungsi memiliki *kind type parameter* sesuai argumen `KIND`. Jika tidak, maka luaran fungsi memiliki *kind type parameter* standar untuk `COMPLEX`.

`CONJG(Z)`

Argumen `Z` bertipe `COMPLEX`. Fungsi ini memberikan `Z`*, yaitu konjugat kompleks (*complex conjugate*) `Z`, dengan tipe dan *kind type parameter* sesuai `Z`.

`INT(A,{KIND})`

Argumen `A` bertipe `REAL`, `INTEGER`, atau `COMPLEX`. Fungsi ini memberikan bilangan bertipe `INTEGER` berdasarkan argumennya `A` sebagai berikut:

- Jika `A` bertipe `INTEGER`, maka dihasilkan bilangan bulat `A`.
- Jika `A` bertipe `REAL`, contoh 3.7, maka dihasilkan bilangan bulat dengan membuang pecahan desimal dari `A`, sesuai contoh, 3.
- Jika `A` bertipe `COMPLEX`, contoh (-3.7,2.5), maka dihasilkan bilangan bulat dari komponen riil `A` dengan membuang pecahan desimalnya, sesuai contoh, -3.

Jika argumen `KIND` diberikan, maka luaran fungsi memiliki *kind type parameter* sesuai argumen `KIND`. Jika tidak, maka luaran fungsi memiliki *kind type parameter* standar untuk `INTEGER`.

`MAX(A1,A2,{A3},...)`

Argumen `A1`, `A2`, `A3`, ... dapat bertipe `REAL` atau `INTEGER`, namun semuanya harus memiliki tipe dan *kind type parameter* yang sama. Fungsi ini memberikan nilai terbesar dari argumen-argumennya, dengan tipe dan *kind type parameter* sesuai argumen-argumennya.

`MIN(A1,A2,{A3},...)`

Argumen `A1`, `A2`, `A3`, ... dapat bertipe `REAL` atau `INTEGER`, namun semuanya harus memiliki tipe dan *kind type parameter* yang sama. Fungsi ini memberikan nilai terkecil dari argumen-argumennya, dengan tipe dan *kind type parameter* sesuai argumen-argumennya.

`REAL(A,{KIND})`

Argumen `A` bertipe `REAL`, `INTEGER`, atau `COMPLEX`. Fungsi ini memberikan bilangan bertipe `REAL` berdasarkan argumennya `A` sebagai berikut:

- Jika `A` bertipe `INTEGER`, maka dihasilkan bilangan `A` bertipe `REAL`.
- Jika `A` bertipe `REAL`, maka dihasilkan bilangan riil `A`.
- Jika `A` bertipe `COMPLEX`, contoh (-3.7,2.5), maka dihasilkan bilangan dari komponen riil `A` bertipe `REAL`, sesuai contoh, -3.7.

Jika argumen `KIND` diberikan, maka luaran fungsi memiliki *kind type parameter* sesuai argumen `KIND`. Jika tidak, maka luaran fungsi memiliki *kind type parameter* standar untuk `REAL`, apabila `A` bertipe `REAL` atau `INTEGER`, atau *kind type parameter* sesuai `A`, apabila `A` bertipe `COMPLEX`.

Lampiran C

Contoh-Contoh Program Fortran 90/95

Kami berikan di sini beberapa contoh program dan subprogram kecil. Sebagian dari contoh tersebut mengerjakan hal-hal yang sangat sederhana, seperti mempertukarkan nilai dua variabel, mencari nilai terkecil dan nilai terbesar, menghitung hasil penjumlahan dan hasil perkalian, namun itu dapat menjadi dasar bagi pemrograman untuk pekerjaan yang lebih kompleks. Catatan, di sini kami tidak lagi menggunakan lambang "˜" untuk menunjukkan satu spasi dalam data `CHARACTER`, karena kami pikir itu tidak lagi perlu.

C.1 Mempertukarkan Nilai Dua Variabel

Program berikut mempertukarkan isi variabel nilai1 dan nilai2. Untuk melakukan itu diperlukan variabel ketiga, yang menyimpan sementara nilai salah satu variabel nilai1 atau nilai2. Langkah-langkah yang dilakukan adalah:

1. menyalin isi variabel nilai1 ke variabel ketiga,
2. menyalin isi variabel nilai2 ke variabel nilai1,
3. menyalin isi variabel ketiga ke variabel nilai2.

```
PROGRAM tukar

IMPLICIT NONE
REAL :: nilai1,nilai2,r

! masukkan nilai1 dan nilai2
WRITE(*,*) 'masukkan nilai 1 dan nilai 2'
READ(*,*) nilai1, nilai2

! tampilkan di layar
WRITE(*,*) 'nilai 1 = ',nilai1
```

```
WRITE(*,*) 'nilai 2 = ',nilai2

! tukar
r     =nilai1 ! isi nilai1 disalin ke r
nilai1=nilai2 ! isi nilai2 disalin ke nilai1
nilai2=r      ! isi r disalin ke nilai2

! tampilkan di layar
WRITE(*,*)
WRITE(*,*) 'nilai 1 = ',nilai1
WRITE(*,*) 'nilai 2 = ',nilai2

STOP

END PROGRAM tukar
```

C.2 Mempertukarkan Dua Baris dan Dua Kolom Sebuah Matriks

Kita manfaatkan langkah-langkah dalam program tukar di Sublampiran C.1 untuk mempertukarkan dua baris dan dua kolom sebuah matriks. Pertukaran baris-baris matriks diperlukan, sebagai contoh, untuk menyelesaikan sistem persamaan linier dengan metode numerik Eliminasi Gauss dan juga Dekomposisi LU. Pada pertukaran dua baris elemen-elemen matriks di kedua baris itu dipertukarkan kolom per kolom. Sebaliknya, pada pertukaran dua kolom elemen-elemen matriks di kedua kolom itu dipertukarkan baris per baris. Program tubarom berikut ini mempertukarkan baris ke-2 dan ke-3, kemudian kolom ke-1 dan ke-3.

```
PROGRAM tubarom

IMPLICIT NONE
INTEGER, PARAMETER :: n=4, m=3 ! jumlah baris dan kolom matriks
INTEGER :: i,j
REAL :: r
REAL, DIMENSION(n,m) :: mata ! matriks n x m

! masukkan elemen matriks
WRITE(*,*) 'masukkan elemen matriks per baris'
WRITE(*,*) 'ukuran matriks: ',n,' x ',m
DO i=1,n
 READ(*,*) (mata(i,j), j=1,m)
END DO
```

```
! tampilkan di layar
DO i=1,n
 WRITE(*,*) (mata(i,j), j=1,m)
END DO

! tukar baris 2 dan 3, lakukan per kolom
DO j=1,m
 r        =mata(2,j)
 mata(2,j)=mata(3,j)
 mata(3,j)=r
END DO

! tampilkan di layar
WRITE(*,*)
DO i=1,n
 WRITE(*,*) (mata(i,j), j=1,m)
END DO

! tukar kolom 1 dan 3, lakukan per baris
DO i=1,n
 r        =mata(i,1)
 mata(i,1)=mata(i,3)
 mata(i,3)=r
END DO

! tampilkan di layar
WRITE(*,*)
DO i=1,n
 WRITE(*,*) (mata(i,j), j=1,m)
END DO

STOP

END PROGRAM tubarom
```

C.3 Mencari Nilai Terkecil dan Nilai Terbesar

Program minmax di bawah ini mencari nilai terkecil dan nilai terbesar dari suatu set nilai. Mencari nilai terkecil dan nilai terbesar diperlukan, contoh, untuk mengurutkan nilai dari yang terkecil sampai yang terbesar dan sebaliknya dari yang terbesar sampai yang terkecil, sebagaimana ditunjukkan dalam Sublampiran C.4. Langkah-langkah untuk mencari nilai terkecil / terbesar adalah:

1. anggap dan ambil nilai pertama sebagai nilai terkecil / terbesar,

2. bandingkan nilai terkecil / terbesar itu dengan nilai kedua, jika nilai kedua lebih kecil / besar, maka ambil nilai kedua sebagai nilai terkecil / terbesar,

3. ulangi untuk nilai ketiga dan seterusnya sampai nilai terakhir.

```
PROGRAM minmax

IMPLICIT NONE
INTEGER, PARAMETER :: n=5 ! jumlah nilai
INTEGER :: i
REAL :: kecil,besar
REAL, DIMENSION(n) :: nilai

! masukkan nilai-nilai
WRITE(*,*) 'masukkan dalam satu baris nilai-nilai sebanyak ',n
READ(*,*) (nilai(i), i=1,n)

! tampilkan nilai ke layar
DO i=1,n
 WRITE(*,*) nilai(i)
END DO

! inisialisasi kecil dan besar
kecil=nilai(1)
besar=nilai(1)
DO i=2,n
! pilih nilai yang lebih kecil untuk variabel kecil
 IF (nilai(i) < kecil) kecil=nilai(i)
! pilih nilai yang lebih besar untuk variabel besar
 IF (nilai(i) > besar) besar=nilai(i)
END DO

! tampilkan hasil ke layar
WRITE(*,*) 'nilai terkecil = ',kecil
WRITE(*,*) 'nilai terbesar = ',besar

STOP

END PROGRAM minmax
```

C.4 Mengurutkan Nilai

Satu set nilai dapat diurutkan mulai dari yang terkecil sampai yang terbesar (*ascending*) atau sebaliknya mulai dari yang terbesar sampai yang terkecil (*descending*). Mengurutkan nilai secara *ascending* dimulai dengan mencari nilai terkecil, lalu menempatkannya

di urutan pertama. Kemudian, tanpa melibatkan nilai yang sudah ditaruh di urutan pertama, ulangi lagi untuk mendapatkan nilai terkecil kedua dan menempatkannya di urutan kedua. Begitu seterusnya sampai semua nilai sudah diurutkan. Jadi, ini merupakan kombinasi dua pekerjaan, yaitu mencari nilai terkecil dan mempertukarkan urutan nilai. Program sortinga berikut ini mengurutkan nilai secara *ascending*.

```
PROGRAM sortinga

IMPLICIT NONE
INTEGER, PARAMETER :: n=5
INTEGER :: i,j
REAL :: r
REAL, DIMENSION(n) :: nilai

! masukkan nilai-nilai
WRITE(*,*) 'masukkan dalam satu baris nilai-nilai sebanyak ',n
READ(*,*) (nilai(i), i=1,n)

! tampilkan nilai ke layar
DO i=1,n
 WRITE(*,*) nilai(i)
END DO

! urutkan dari kecil ke besar
DO i=1,n-1
 DO j=i+1,n
! nilai ke-i tiap kali dibandingkan dengan nilai-nilai setelahnya
! jika nilai setelahnya lebih kecil dari nilai ke-i, maka keduanya
! saling bertukar tempat
   IF (nilai(j) < nilai(i)) THEN
    r       =nilai(j)
    nilai(j)=nilai(i)
    nilai(i)=r
   END IF
 END DO
END DO

! tampilkan hasil ke layar
WRITE(*,*)
DO i=1,n
 WRITE(*,*) nilai(i)
END DO

STOP

END PROGRAM sortinga
```

Untuk mengurutkan nilai secara *descending* berlaku cara yang sama, namun dilakukan terhadap nilai terbesar. Program sortingd di bawah ini mengurutkan nilai secara *descending*. Perhatikan bahwa perbedaan signifikan antara program sortingd dan program sortinga terdapat hanya di baris ke-24.

```
PROGRAM sortingd

IMPLICIT NONE
INTEGER, PARAMETER :: n=5
INTEGER :: i,j
REAL :: r
REAL, DIMENSION(n) :: nilai

! masukkan nilai-nilai
WRITE(*,*) 'masukkan dalam satu baris nilai-nilai sebanyak ',n
READ(*,*) (nilai(i), i=1,n)

! tampilkan nilai ke layar
DO i=1,n
 WRITE(*,*) nilai(i)
END DO

! urutkan dari besar ke kecil
DO i=1,n-1
 DO j=i+1,n
! nilai ke-i tiap kali dibandingkan dengan nilai-nilai setelahnya
! jika nilai setelahnya lebih besar dari nilai ke-i, maka keduanya
! saling bertukar tempat
   IF (nilai(j) > nilai(i)) THEN
    r       =nilai(j)
    nilai(j)=nilai(i)
    nilai(i)=r
   END IF
 END DO
END DO

! tampilkan hasil ke layar
WRITE(*,*)
DO i=1,n
 WRITE(*,*) nilai(i)
END DO

STOP

END PROGRAM sortingd
```

C.5 Menghitung Hasil Penjumlahan dan Hasil Perkalian

Ambillah sebagai contoh hasil penjumlahan S dan hasil perkalian P diberikan sebagai berikut:

$$S = \sum_{i=1}^{n} x_i$$
$$P = \prod_{i=1}^{n} x_i$$

Keduanya S dan P dihitung oleh program sumkali di bawah ini. Prinsip yang digunakan adalah S dan P tiap kali diperbarui nilainya dalam suatu pengulangan. Perhatikan baris ke-21 dan ke-22 bahwa inisialisasi untuk menghitung hasil penjumlahan berbeda dari inisialisasi untuk menghitung hasil perkalian.

```
PROGRAM sumkali

IMPLICIT NONE
INTEGER, PARAMETER :: n=5
INTEGER :: i
REAL :: s,p
REAL, DIMENSION(n) :: x

! masukkan nilai-nilai
WRITE(*,*) 'masukkan dalam satu baris nilai-nilai sebanyak ',n
READ(*,*) (x(i), i=1,n)

! tampilkan nilai ke layar
DO i=1,n
 WRITE(*,*) x(i)
END DO

! menghitung hasil penjumlahan dan hasil perkalian

s=0.0 ! inisialisasi untuk menghitung hasil penjumlahan
p=1.0 ! inisialisasi untuk menghitung hasil perkalian
DO i=1,n
 s=s+x(i) ! perbarui nilai s
 p=p*x(i) ! perbarui nilai p
END DO

! tampilkan hasil ke layar
```

```
WRITE(*,*)
WRITE(*,*) 'S = ',s
WRITE(*,*) 'P = ',p

STOP

END PROGRAM sumkali
```

C.6 Menghitung Perkalian Matriks

Di sini kita terapkan cara menghitung hasil penjumlahan yang ditunjukkan di Sub-lampiran C.5 untuk menghitung perkalian dua matriks. Misalkan C adalah sebuah matriks hasil perkalian matriks A dan matriks B, yaitu $C = AB$. Elemen-elemen matriks C dihitung sebagai berikut:

$$C_{ij} = \sum_k A_{ik} B_{kj} \,.$$

Contoh program di bawah ini menghitung perkalian matriks $C = AB$. Di situ A matriks 2×3 dan B matriks 3×2, sehingga C matriks 2×2.

```
PROGRAM matriksab

IMPLICIT NONE
INTEGER, PARAMETER :: na=2, ma=3, & ! jumlah baris & kolom matriks A
                      nb=3, mb=2    ! jumlah baris & kolom matriks B
INTEGER :: i,j,k
REAL, DIMENSION(na,ma) :: mata ! matriks A
REAL, DIMENSION(nb,mb) :: matb ! matriks B
REAL, DIMENSION(na,mb) :: matc ! matriks C

! masukkan elemen matriks A
WRITE(*,*) 'masukkan elemen matriks A per baris'
WRITE(*,*) 'ukuran matriks: ',na,' x ',ma
DO i=1,na
 READ(*,*) (mata(i,j), j=1,ma)
END DO

! tampilkan di layar
DO i=1,na
 WRITE(*,*) (mata(i,j), j=1,ma)
END DO

! masukkan elemen matriks B
WRITE(*,*) 'masukkan elemen matriks B per baris'
WRITE(*,*) 'ukuran matriks: ',nb,' x ',mb
```

```
DO i=1,nb
 READ(*,*) (matb(i,j), j=1,mb)
END DO

! tampilkan di layar
DO i=1,nb
 WRITE(*,*) (matb(i,j), j=1,mb)
END DO

! hitung matriks C=AB
DO j=1,mb
 DO i=1,na
  matc(i,j)=0.0 ! inisialisasi
  DO k=1,nb
   matc(i,j)=matc(i,j)+mata(i,k)*matb(k,j)
  END DO
 END DO
END DO

! tampilkan di layar
WRITE(*,*)
DO i=1,na
 WRITE(*,*) (matc(i,j), j=1,mb)
END DO

STOP

END PROGRAM matriksab
```

C.7 Interpolasi Lagrange

Misalkan kita mempunyai satu set data $f(x_i)$, $i = 1, ..., n$. Jika kita ingin mendapatkan nilai data di titik x, sebut saja $p(x)$, dengan $x \neq x_i$ dan $x_1 < x < x_n$, maka $p(x)$ dapat dihitung menggunakan interpolasi Lagrange sebagai berikut:

$$p(x) = \sum_{i=1}^{n} l(x, x_i) f(x_i) \,,$$

dengan $l(x, x_i)$ adalah koefisien Lagrange:

$$l(x, x_i) = \prod_{j \neq i}^{n} \left(\frac{x - x_j}{x_i - x_j} \right) \,.$$

Nilai $p(x)$ dan $l(x, x_i)$ dapat dihitung dengan langkah-langkah seperti yang dicontohkan oleh program sumkali di Sublampiran C.5. Namun, perhatikan pada rumus

koefisien Lagrange di atas bahwa perkalian dikerjakan untuk semua nilai j, KECUALI $j = i$. Berikut ini contoh subrutin untuk menghitung koefisien Lagrange (menghitung hasil perkalian yang tidak melibatkan semua suku):

```
SUBROUTINE colag(n,x,x0,l)

IMPLICIT NONE
INTEGER, INTENT(IN) :: n ! jumlah titik data
REAL, INTENT(IN) :: x0 ! titik data yang diinterpolasi
REAL, DIMENSION(n), INTENT(IN) :: x  ! titik data
REAL, DIMENSION(n), INTENT(OUT) :: l ! koefisien lagrange
INTEGER :: i,j

! menghitung koefisien lagrange

DO i=1,n
 l(i)=1.0 ! inisialisasi
 DO j=1,n
! perkalian dilakukan hanya jika j /= i
  IF (j /= i) l(i)=l(i)*(x0-x(j))/(x(i)-x(j))
 END DO
END DO

RETURN

END SUBROUTINE colag
```

C.8 Fungsi Faktorial dan Fungsi Faktorial Ganda

Pada Subbab 7.4 kita sudah melihat contoh fungsi rekursif untuk menghitung nilai fungsi faktorial, yaitu $n! = n(n-1)(n-2)...1$, sebagai berikut:

```
RECURSIVE FUNCTION faktorial(n) RESULT(m)

IMPLICIT NONE
INTEGER, INTENT(IN) :: n
INTEGER :: m

IF (n==1 .OR. n==0) THEN
 m=1                 ! 0! = 1! = 1
ELSE IF (n>1) THEN
 m=n*faktorial(n-1)  ! m = n(n-1)(n-2)...1 = n!
ELSE
 STOP                ! program berhenti jika n negatif
```

```
END IF

RETURN

END FUNCTION faktorial
```

Selain fungsi faktorial $n!$ ada juga fungsi faktorial ganda, yaitu $n!! = n(n-2)(n-4)...n_0$, dengan $n_0 = 1$ jika n ganjil dan $n_0 = 2$ jika n genap. Berikut ini contoh fungsi rekursif untuk menghitung $n!!$:

```
RECURSIVE FUNCTION faktorial2(n) RESULT(m)

IMPLICIT NONE
INTEGER, INTENT(IN) :: n
INTEGER :: m

IF (n==1) THEN
 m=1                 ! 1!! = 1
ELSE IF (n==2) THEN
 m=2                 ! 2!! = 2
ELSE IF (n>2) THEN
 m=n*faktorial2(n-2) ! m = n(n-2)(n-4)... = n!!
ELSE
 STOP                ! program berhenti jika n < 1
END IF

RETURN

END FUNCTION faktorial2
```

C.9 Ekspansi Binomial

Nilai $(a+b)^p$ dapat dihitung menurut ekspansi binomial sebagai sebuah deret berikut:

$$(a+b)^p = \sum_{n=0}^{\infty} \begin{pmatrix} p \\ n \end{pmatrix} a^{p-n} b^n \,,$$

dengan

$$\begin{pmatrix} p \\ n \end{pmatrix} = \begin{cases} 1 & , (n=0) \\ \frac{1}{n!} \prod_{i=0}^{n-1} (p-i) & , (n>0) \end{cases}$$

disebut koefisien binomial. Secara umum p dapat berupa bilangan bulat maupun riil, bernilai positif maupun negatif. Khusus untuk p bilangan bulat positif, koefisien binomial bernilai nol untuk $n > p$, sehingga ekspansi binomial menjadi deret berhingga,

yaitu hanya sampai suku ke-p. Berikut ini contoh subrutin yang menghitung koefisien binomial untuk p bilangan bulat positif, yang dipakai bersama dengan fungsi faktorial dari Sublampiran C.8.

```
SUBROUTINE cobino(p,b)

IMPLICIT NONE
INTEGER, INTENT(IN) :: p
INTEGER, DIMENSION(0:p), INTENT(OUT) :: b
INTEGER :: i,n

! menghitung koefisien binomial

b(0)=1
DO n=1,p
 b(n)=1 ! inisialisasi
 DO i=0,n-1
  b(n)=b(n)*(p-i)
 END DO
 b(n)=b(n)/faktorial(n)
END DO

RETURN

END SUBROUTINE cobino
```

C.10 Koefisien Clebsch-Gordan

Kini kami berikan satu contoh dari bidang fisika, yaitu berkenaan dengan penjumlahan momentum angular dalam mekanika kuantum. Kita berurusan dengan koefisien Clebsch-Gordan, yang nilainya dapat dihitung menurut rumus berikut:

$$\begin{aligned}
C\left(j_1 j_2 j; m_1 m_2 m\right) = {} & \delta_{m,m_1+m_2}(2j+1)^{\frac{1}{2}} \\
& \times \left[\frac{(j_1+j_2-j)!(j+j_1-j_2)!(j_2+j-j_1)!}{(j_1+j_2+j+1)!}\right]^{\frac{1}{2}} \\
& \times \left[(j_1+m_1)!(j_2+m_2)!(j+m)!\right]^{\frac{1}{2}} \\
& \times \left[(j_1-m_1)!(j_2-m_2)!(j-m)!\right]^{\frac{1}{2}} \\
& \times \sum_k \frac{(-1)^k}{k!}\left[(j_1+j_2-j-k)!\right]^{-1} \\
& \qquad \times \left[(j_1-m_1-k)!(j_2+m_2-k)!\right]^{-1} \\
& \qquad \times \left[(j-j_2+m_1+k)!(j-j_1-m_2+k)!\right]^{-1} .
\end{aligned}$$

Pasangan-pasangan argumen koefisien Clebsch-Gordan j_1 dan m_1, j_2 dan m_2, j dan m masing-masing bisa bernilai bulat maupun kelipatan ganjil dari setengah. Namun,

hubungan antar nilai j_1, j_2, j, m_1, m_2, dan m harus memenuhi beberapa syarat. Apabila salah satu syarat saja tidak dipenuhi, maka koefisien Clebsch-Gordan bernilai nol. Syarat-syarat tersebut adalah:

1. $|j_1 - j_2| \leqslant j \leqslant j_1 + j_2$,
2. $m = m_1 + m_2$,
3. $-j_1 \leqslant m_1 \leqslant j_1$,
4. $-j_2 \leqslant m_2 \leqslant j_2$,
5. $-j \leqslant m \leqslant j$.

Pada rumus koefisien Clebsch-Gordan di atas variabel deret k tidak memiliki batas bawah dan batas atas yang tetap, melainkan dinamis. Kita ketahui bahwa argumen fungsi faktorial tidak boleh negatif. Dengan demikian, batas bawah dan batas atas k ditentukan sedemikian agar di dalam deret itu tidak ada argumen fungsi faktorial yang kurang dari nol. Dari tiga fungsi faktorial $(j_1+j_2-j-k)!$, $(j_1-m_1-k)!$, dan $(j_2+m_2-k)!$ kita lihat bahwa k tidak boleh lebih besar dari ketiga-tiganya $j_1 + j_2 - j$, $j_1 - m_1$, dan j_2+m_2. Karena itu, batas atas k adalah salah satu dari j_1+j_2-j, j_1-m_1, dan j_2+m_2, yang nilainya terkecil. Kemudian, dari tiga fungsi faktorial $k!$, $(j - j_2 + m_1 + k)!$, dan $(j - j_1 - m_2 + k)!$ kita simpulkan bahwa k tidak boleh kurang dari ketiga-tiganya 0, $-j + j_2 - m_1$, dan $-j + j_1 + m_2$. Dengan demikian, batas bawah k adalah salah satu dari 0, $-j + j_2 - m_1$, dan $-j + j_1 + m_2$, yang nilainya terbesar.

Contoh fungsi berikut ini menghitung koefisien Clebsch-Gordan dengan presisi ganda, khusus untuk j_1, j_2, j, m_1, m_2, dan m semuanya bernilai bulat. Fungsi tersebut membutuhkan fungsi faktorial dari Sublampiran C.8.

```
FUNCTION clebsch(j1,j2,j,m1,m2,m) RESULT(cg)

IMPLICIT NONE
INTEGER, PARAMETER :: dpr=KIND(1.0D0)
INTEGER, INTENT(IN) :: j1,j2,j,m1,m2,m
INTEGER :: minplus,k,kmin,kmax
REAL(dpr) :: cg,deretk,ff

! periksa hubungan nilai-nilai argumen, apakah memenuhi syarat

IF (j > j1+j2 .OR. j < ABS(j1-j2)) THEN
 cg=0.0_dpr
ELSE IF (m /= m1+m2) THEN
 cg=0.0_dpr
ELSE IF (ABS(m1) > j1) THEN
 cg=0.0_dpr
ELSE IF (ABS(m2) > j2) THEN
 cg=0.0_dpr
```

```
ELSE IF (ABS(m) > j) THEN
 cg=0.0_dpr
ELSE

! syarat dipenuhi, hitung koefisien Clebsch-Gordan

! hitung deret k

 kmin=MAX(0,-j+j2-m1,-j+j1+m2) ! batas bawah deret k
 kmax=MIN(j1+j2-j,j1-m1,j2+m2) ! batas atas deret k

 deretk=0.0_dpr ! inisialisasi nilai deret k
 minplus=(-1)**kmin ! inisialisasi (-1)^(k)
 DO k=kmin,kmax
  ff= faktorial(k)*faktorial(j1+j2-j-k) &
      *faktorial(j1-m1-k)*faktorial(j2+m2-k) &
      *faktorial(j-j2+m1+k)*faktorial(j-j1-m2+k)
  deretk=deretk+minplus/ff
  minplus=-minplus    ! untuk berikutnya, (-1)^(k+1)
 END DO

! koefisien Clebsch-Gordan

 cg= SQRT( (2.0_dpr*j+1.0_dpr) &
          *faktorial(j1+j2-j)*faktorial(j+j1-j2)*faktorial(j2+j-j1) &
          /faktorial(j1+j2+j+1) &
          *faktorial(j1+m1)*faktorial(j2+m2)*faktorial(j+m) &
          *faktorial(j1-m1)*faktorial(j2-m2)*faktorial(j-m))*deretk

END IF

RETURN

END FUNCTION clebsch
```

Catatan:

- Di bagian awal fungsi, dengan rangkaian pernyataan `IF` dan `ELSE IF`, dilakukan pemeriksaan apakah syarat-syarat untuk hubungan nilai-nilai j_1, j_2, j, m_1, m_2, dan m semuanya dipenuhi. Jika satu saja tidak dipenuhi, maka fungsi clebsch memberikan hasil nol.

- Batas bawah dan batas atas variabel deret k ditentukan dengan menggunakan fungsi intrinsik `MAX` dan `MIN`.

C.11 Program yang Memanfaatkan Atribut SAVE untuk Variabel Lokal

Program prodegrad berikut ini mengubah satuan sudut dari derajat ke radian. Pengubahan satuan dilakukan oleh fungsi degrad. Di akhir program pengguna dapat memilih untuk berhenti atau memasukkan lagi nilai sudut yang lain dalam derajat untuk diubah ke radian. Dengan demikian, fungsi internal degrad mungkin saja dipanggil lebih dari satu kali.

Untuk mengubah satuan sudut dari derajat ke radian fungsi degrad memerlukan nilai variabel ff, yaitu $\pi/180$. Setelah pemanggilan pertama fungsi degrad nilai variabel ff tidak perlu dihitung lagi. Karena itu, variabel ff diberi atribut `SAVE`. Di dalam fungsi degrad variabel i digunakan untuk menentukan apakah nilai variabel ff harus dihitung atau tidak. Variabel i juga memiliki atribut `SAVE`, karena mendapat nilai awal i = 1 saat dideklarasikan. Setelah pemanggilan pertama fungsi degrad nilai variabel i tetap sama dengan 0, sesuai nilainya yang terakhir sebelum eksekusi program keluar dari fungsi degrad pada pemanggilan pertama. Dengan demikian, nilai variabel ff hanya dihitung sekali, yaitu pada pemanggilan pertama fungsi degrad.

```
PROGRAM prodegrad

IMPLICIT NONE
REAL :: sudut
CHARACTER(1) :: ulang

DO

! masukkan nilai sudut

 WRITE(*,*) 'masukkan nilai sudut dalam derajat'
 READ(*,*) sudut

! tampilkan di layar
 WRITE(*,"(1X,A,F7.2)") 'nilai sudut dalam derajat = ',sudut
 WRITE(*,"(1X,A,F7.2)") 'nilai sudut dalam radian  = ',degrad(sudut)

! ulangi?
 WRITE(*,*) 'lagi (y/t)?'
 READ(*,*) ulang

 IF (ulang == 't' .OR. ulang == 'T') EXIT

END DO

STOP
```

```
CONTAINS

FUNCTION degrad(deg) RESULT(rad)

IMPLICIT NONE
REAL, INTENT(IN) :: deg
INTEGER :: i=1
REAL :: rad
REAL, SAVE :: ff

IF (i == 1) THEN
 ff=ACOS(-1.0)/180.0
 i=0
END IF

rad=deg*ff

RETURN

END FUNCTION degrad

END PROGRAM prodegrad
```

C.12 Program dengan Subprogram Eksternal

Untuk menunjukkan contoh program yang berisi subprogram eksternal, kami tampilkan terlebih dulu sebagai pembanding contoh program yang berisi subprogram internal. Berikut ini sebuah program yang menghitung koefisien Clebsch-Gordan. Program ini memanggil fungsi clebsch (Sublampiran C.10) dan fungsi clebsch memerlukan fungsi faktorial (Sublampiran C.8). Dalam program itu baik fungsi clebsch maupun fungsi faktorial merupakan subprogram internal, kedua fungsi itu ditempatkan setelah pernyataan `CONTAINS` di akhir program utama. Untuk memanggil fungsi clebsch dan fungsi faktorial tidak diperlukan pernyataan blok `INTERFACE` di bagian spesifikasi program utama.

```
PROGRAM cocg

IMPLICIT NONE
INTEGER :: j1,j2,j,m1,m2,m

! masukkan nilai-nilai

WRITE(*,*) 'masukkan j1, j2, j, m1, m2, m'
READ(*,*) j1,j2,j,m1,m2,m
```

```
! tampilkan di layar
WRITE(*,"(1X,A,6I3)") 'j1, j2, j, m1, m2, m = ',j1,j2,j,m1,m2,m

! tampilkan koefisien Clebsch-Gordan
WRITE(*,*) 'koefisien Clebsch-Gordan = ',clebsch(j1,j2,j,m1,m2,m)

STOP

CONTAINS

FUNCTION clebsch(j1,j2,j,m1,m2,m) RESULT(cg)

IMPLICIT NONE
INTEGER, PARAMETER :: dpr=KIND(1.0D0)
INTEGER, INTENT(IN) :: j1,j2,j,m1,m2,m
INTEGER :: minplus,k,kmin,kmax
REAL(dpr) :: cg,deretk,ff

! periksa hubungan nilai-nilai argumen, apakah memenuhi syarat

IF (j > j1+j2 .OR. j < ABS(j1-j2)) THEN
 cg=0.0_dpr
ELSE IF (m /= m1+m2) THEN
 cg=0.0_dpr
ELSE IF (ABS(m1) > j1) THEN
 cg=0.0_dpr
ELSE IF (ABS(m2) > j2) THEN
 cg=0.0_dpr
ELSE IF (ABS(m) > j) THEN
 cg=0.0_dpr
ELSE

! syarat dipenuhi, hitung koefisien Clebsch-Gordan

! hitung deret k

 kmin=MAX(0,-j+j2-m1,-j+j1+m2) ! batas bawah deret k
 kmax=MIN(j1+j2-j,j1-m1,j2+m2) ! batas atas deret k

 deretk=0.0_dpr ! inisialisasi nilai deret k
 minplus=(-1)**kmin ! inisialisasi (-1)^(k)
 DO k=kmin,kmax
  ff= faktorial(k)*faktorial(j1+j2-j-k) &
     *faktorial(j1-m1-k)*faktorial(j2+m2-k) &
     *faktorial(j-j2+m1+k)*faktorial(j-j1-m2+k)
  deretk=deretk+minplus/ff
```

```
  minplus=-minplus    ! untuk berikutnya, (-1)^(k+1)
 END DO

! koefisien Clebsch-Gordan

 cg= SQRT( (2.0_dpr*j+1.0_dpr) &
          *faktorial(j1+j2-j)*faktorial(j+j1-j2)*faktorial(j2+j-j1) &
          /faktorial(j1+j2+j+1) &
          *faktorial(j1+m1)*faktorial(j2+m2)*faktorial(j+m) &
          *faktorial(j1-m1)*faktorial(j2-m2)*faktorial(j-m))*deretk

END IF

RETURN

END FUNCTION clebsch

RECURSIVE FUNCTION faktorial(n) RESULT(m)

IMPLICIT NONE
INTEGER, INTENT(IN) :: n
INTEGER :: m

IF (n==1 .OR. n==0) THEN
 m=1                   ! 0! = 1! = 1
ELSE IF (n>1) THEN
 m=n*faktorial(n-1)  ! m = n(n-1)(n-2)...1 = n!
ELSE
 STOP                  ! program berhenti jika n negatif
END IF

RETURN

END FUNCTION faktorial

END PROGRAM cocg
```

Kini kita jadikan fungsi clebsch sebagai subprogram eksternal. Fungsi clebsch dikeluarkan dari program utama dan, dalam contoh program berikut, ditaruh sebelum program utama dalam file yang sama. Jika diinginkan fungsi clebsch dapat ditaruh setelah program utama dalam file yang sama atau bahkan ditempatkan dalam file terpisah dari file yang berisi program utama. Fungsi faktorial dijadikan sebagai subprogram internal di dalam fungsi clebsch, ditaruh setelah pernyataan `CONTAINS`. Fungsi clebsch dipanggil oleh program utama, karena itu di bagian spesifikasi program utama dinyatakan blok `INTERFACE` untuk fungsi clebsch. Fungsi clebsch memanggil fungsi faktorial, namun, di bagian spesifikasi fungsi clebsch tidak perlu dinyatakan blok `INTERFACE` untuk fungsi

faktorial, karena fungsi faktorial merupakan subprogram internal fungsi clebsch.

```
FUNCTION clebsch(j1,j2,j,m1,m2,m) RESULT(cg)

IMPLICIT NONE
INTEGER, PARAMETER :: dpr=KIND(1.0D0)
INTEGER, INTENT(IN) :: j1,j2,j,m1,m2,m
INTEGER :: minplus,k,kmin,kmax
REAL(dpr) :: cg,deretk,ff

! periksa hubungan nilai-nilai argumen, apakah memenuhi syarat

IF (j > j1+j2 .OR. j < ABS(j1-j2)) THEN
 cg=0.0_dpr
ELSE IF (m /= m1+m2) THEN
 cg=0.0_dpr
ELSE IF (ABS(m1) > j1) THEN
 cg=0.0_dpr
ELSE IF (ABS(m2) > j2) THEN
 cg=0.0_dpr
ELSE IF (ABS(m) > j) THEN
 cg=0.0_dpr
ELSE

! syarat dipenuhi, hitung koefisien Clebsch-Gordan

! hitung deret k

 kmin=MAX(0,-j+j2-m1,-j+j1+m2) ! batas bawah deret k
 kmax=MIN(j1+j2-j,j1-m1,j2+m2) ! batas atas deret k

 deretk=0.0_dpr ! inisialisasi nilai deret k
 minplus=(-1)**kmin ! inisialisasi (-1)^(k)
 DO k=kmin,kmax
  ff= faktorial(k)*faktorial(j1+j2-j-k) &
     *faktorial(j1-m1-k)*faktorial(j2+m2-k) &
     *faktorial(j-j2+m1+k)*faktorial(j-j1-m2+k)
  deretk=deretk+minplus/ff
  minplus=-minplus    ! untuk berikutnya, (-1)^(k+1)
 END DO

! koefisien Clebsch-Gordan

 cg= SQRT( (2.0_dpr*j+1.0_dpr) &
          *faktorial(j1+j2-j)*faktorial(j+j1-j2)*faktorial(j2+j-j1) &
          /faktorial(j1+j2+j+1) &
```

```
          *faktorial(j1+m1)*faktorial(j2+m2)*faktorial(j+m) &
          *faktorial(j1-m1)*faktorial(j2-m2)*faktorial(j-m))*deretk

END IF

RETURN

CONTAINS

RECURSIVE FUNCTION faktorial(n) RESULT(m)

IMPLICIT NONE
INTEGER, INTENT(IN) :: n
INTEGER :: m

IF (n==1 .OR. n==0) THEN
 m=1                 ! 0! = 1! = 1
ELSE IF (n>1) THEN
 m=n*faktorial(n-1)  ! m = n(n-1)(n-2)...1 = n!
ELSE
 STOP                ! program berhenti jika n negatif
END IF

RETURN

END FUNCTION faktorial

END FUNCTION clebsch

PROGRAM cocg

IMPLICIT NONE
INTEGER :: j1,j2,j,m1,m2,m

INTERFACE
 FUNCTION clebsch(j1,j2,j,m1,m2,m) RESULT(cg)
  IMPLICIT NONE
  INTEGER, PARAMETER :: dpr=KIND(1.0D0)
  INTEGER, INTENT(IN) :: j1,j2,j,m1,m2,m
  REAL(dpr) :: cg
 END FUNCTION clebsch
END INTERFACE

! masukkan nilai-nilai

WRITE(*,*) 'masukkan j1, j2, j, m1, m2, m'
```

```
READ(*,*) j1,j2,j,m1,m2,m

! tampilkan di layar
WRITE(*,"(1X,A,6I3)") 'j1, j2, j, m1, m2, m = ',j1,j2,j,m1,m2,m

! tampilkan koefisien Clebsch-Gordan
WRITE(*,*) 'koefisien Clebsch-Gordan = ',clebsch(j1,j2,j,m1,m2,m)

STOP

END PROGRAM cocg
```

Pada program berikut fungsi clebsch dan fungsi faktorial kedua-duanya ditempatkan sebagai subprogram eksternal. Fungsi clebsch dan fungsi faktorial diletakkan, pada contoh ini, sebelum program utama dalam file yang sama. Fungsi clebsch dipanggil oleh program utama, karena itu di bagian spesifikasi program utama dinyatakan blok `INTERFACE` untuk fungsi clebsch. Fungsi clebsch memanggil fungsi faktorial, maka di bagian spesifikasi fungsi clebsch dinyatakan blok `INTERFACE` untuk fungsi faktorial.

```
RECURSIVE FUNCTION faktorial(n) RESULT(m)

IMPLICIT NONE
INTEGER, INTENT(IN) :: n
INTEGER :: m

IF (n==1 .OR. n==0) THEN
 m=1                  ! 0! = 1! = 1
ELSE IF (n>1) THEN
 m=n*faktorial(n-1)   ! m = n(n-1)(n-2)...1 = n!
ELSE
 STOP                 ! program berhenti jika n negatif
END IF

RETURN

END FUNCTION faktorial

FUNCTION clebsch(j1,j2,j,m1,m2,m) RESULT(cg)

IMPLICIT NONE
INTEGER, PARAMETER :: dpr=KIND(1.0D0)
INTEGER, INTENT(IN) :: j1,j2,j,m1,m2,m
INTEGER :: minplus,k,kmin,kmax
REAL(dpr) :: cg,deretk,ff
```

```
INTERFACE
 RECURSIVE FUNCTION faktorial(n) RESULT(m)
 IMPLICIT NONE
 INTEGER, INTENT(IN) :: n
 INTEGER :: m
 END FUNCTION faktorial
END INTERFACE

! periksa hubungan nilai-nilai argumen, apakah memenuhi syarat

IF (j > j1+j2 .OR. j < ABS(j1-j2)) THEN
 cg=0.0_dpr
ELSE IF (m /= m1+m2) THEN
 cg=0.0_dpr
ELSE IF (ABS(m1) > j1) THEN
 cg=0.0_dpr
ELSE IF (ABS(m2) > j2) THEN
 cg=0.0_dpr
ELSE IF (ABS(m) > j) THEN
 cg=0.0_dpr
ELSE

! syarat dipenuhi, hitung koefisien Clebsch-Gordan

! hitung deret k

 kmin=MAX(0,-j+j2-m1,-j+j1+m2) ! batas bawah deret k
 kmax=MIN(j1+j2-j,j1-m1,j2+m2) ! batas atas deret k

 deretk=0.0_dpr ! inisialisasi nilai deret k
 minplus=(-1)**kmin ! inisialisasi (-1)^(k)
 DO k=kmin,kmax
  ff= faktorial(k)*faktorial(j1+j2-j-k) &
     *faktorial(j1-m1-k)*faktorial(j2+m2-k) &
     *faktorial(j-j2+m1+k)*faktorial(j-j1-m2+k)
  deretk=deretk+minplus/ff
  minplus=-minplus    ! untuk berikutnya, (-1)^(k+1)
 END DO

! koefisien Clebsch-Gordan

 cg= SQRT( (2.0_dpr*j+1.0_dpr) &
         *faktorial(j1+j2-j)*faktorial(j+j1-j2)*faktorial(j2+j-j1) &
         /faktorial(j1+j2+j+1) &
         *faktorial(j1+m1)*faktorial(j2+m2)*faktorial(j+m) &
         *faktorial(j1-m1)*faktorial(j2-m2)*faktorial(j-m))*deretk
```

```
END IF

RETURN

END FUNCTION clebsch

PROGRAM cocg

IMPLICIT NONE
INTEGER :: j1,j2,j,m1,m2,m

INTERFACE
 FUNCTION clebsch(j1,j2,j,m1,m2,m) RESULT(cg)
  IMPLICIT NONE
  INTEGER, PARAMETER :: dpr=KIND(1.0D0)
  INTEGER, INTENT(IN) :: j1,j2,j,m1,m2,m
  REAL(dpr) :: cg
 END FUNCTION clebsch
END INTERFACE

! masukkan nilai-nilai

WRITE(*,*) 'masukkan j1, j2, j, m1, m2, m'
READ(*,*) j1,j2,j,m1,m2,m

! tampilkan di layar
WRITE(*,"(1X,A,6I3)") 'j1, j2, j, m1, m2, m = ',j1,j2,j,m1,m2,m

! tampilkan koefisien Clebsch-Gordan
WRITE(*,*) 'koefisien Clebsch-Gordan = ',clebsch(j1,j2,j,m1,m2,m)

STOP

END PROGRAM cocg
```

Pada contoh selanjutnya hanya fungsi faktorial yang ditempatkan sebagai subprogram eksternal, sementara fungsi clebsch menjadi subprogram internal di dalam program utama. Sebuah blok `INTERFACE` untuk fungsi faktorial dinyatakan di bagian spesifikasi fungsi clebsch.

```
RECURSIVE FUNCTION faktorial(n) RESULT(m)

IMPLICIT NONE
INTEGER, INTENT(IN) :: n
INTEGER :: m
```

```
IF (n==1 .OR. n==0) THEN
 m=1                 ! 0! = 1! = 1
ELSE IF (n>1) THEN
 m=n*faktorial(n-1)  ! m = n(n-1)(n-2)...1 = n!
ELSE
 STOP                ! program berhenti jika n negatif
END IF

RETURN

END FUNCTION faktorial

PROGRAM cocg

IMPLICIT NONE
INTEGER :: j1,j2,j,m1,m2,m

! masukkan nilai-nilai

WRITE(*,*) 'masukkan j1, j2, j, m1, m2, m'
READ(*,*) j1,j2,j,m1,m2,m

! tampilkan di layar
WRITE(*,"(1X,A,6I3)") 'j1, j2, j, m1, m2, m = ',j1,j2,j,m1,m2,m

! tampilkan koefisien Clebsch-Gordan
WRITE(*,*) 'koefisien Clebsch-Gordan = ',clebsch(j1,j2,j,m1,m2,m)

STOP

CONTAINS

FUNCTION clebsch(j1,j2,j,m1,m2,m) RESULT(cg)

IMPLICIT NONE
INTEGER, PARAMETER :: dpr=KIND(1.0D0)
INTEGER, INTENT(IN) :: j1,j2,j,m1,m2,m
INTEGER :: minplus,k,kmin,kmax
REAL(dpr) :: cg,deretk,ff

INTERFACE
 RECURSIVE FUNCTION faktorial(n) RESULT(m)
 IMPLICIT NONE
 INTEGER, INTENT(IN) :: n
 INTEGER :: m
```

```
 END FUNCTION faktorial
END INTERFACE

! periksa hubungan nilai-nilai argumen, apakah memenuhi syarat

IF (j > j1+j2 .OR. j < ABS(j1-j2)) THEN
 cg=0.0_dpr
ELSE IF (m /= m1+m2) THEN
 cg=0.0_dpr
ELSE IF (ABS(m1) > j1) THEN
 cg=0.0_dpr
ELSE IF (ABS(m2) > j2) THEN
 cg=0.0_dpr
ELSE IF (ABS(m) > j) THEN
 cg=0.0_dpr
ELSE

! syarat dipenuhi, hitung koefisien Clebsch-Gordan

! hitung deret k

 kmin=MAX(0,-j+j2-m1,-j+j1+m2) ! batas bawah deret k
 kmax=MIN(j1+j2-j,j1-m1,j2+m2) ! batas atas deret k

 deretk=0.0_dpr ! inisialisasi nilai deret k
 minplus=(-1)**kmin ! inisialisasi (-1)^(k)
 DO k=kmin,kmax
  ff= faktorial(k)*faktorial(j1+j2-j-k) &
     *faktorial(j1-m1-k)*faktorial(j2+m2-k) &
     *faktorial(j-j2+m1+k)*faktorial(j-j1-m2+k)
  deretk=deretk+minplus/ff
  minplus=-minplus    ! untuk berikutnya, (-1)^(k+1)
 END DO

! koefisien Clebsch-Gordan

 cg= SQRT( (2.0_dpr*j+1.0_dpr) &
          *faktorial(j1+j2-j)*faktorial(j+j1-j2)*faktorial(j2+j-j1) &
          /faktorial(j1+j2+j+1) &
          *faktorial(j1+m1)*faktorial(j2+m2)*faktorial(j+m) &
          *faktorial(j1-m1)*faktorial(j2-m2)*faktorial(j-m))*deretk

END IF

RETURN
```

```
END FUNCTION clebsch

END PROGRAM cocg
```

Pada contoh terakhir di atas, blok `INTERFACE` untuk fungsi faktorial dinyatakan di bagian spesifikasi fungsi clebsch, sehingga hanya berlaku di dalam fungsi clebsch. Apabila ada subprogram internal lain dalam program utama itu yang memerlukan fungsi eksternal faktorial, maka di bagian spesifikasi subprogram internal lain tersebut harus dinyatakan juga blok `INTERFACE` untuk fungsi faktorial. Pada kasus seperti ini, sebaiknya blok `INTERFACE` dinyatakan bukan di bagian spesifikasi tiap subprogram internal yang membutuhkan, melainkan di bagian spesifikasi program utama, sehingga blok `INTERFACE` itu berlaku di seluruh program utama, termasuk di semua subprogram internalnya. Hal ini dicontohkan oleh program di bawah.

Program utama berikut ini berisi dua subprogram internal, yaitu fungsi clebsch dan subrutin cobino (Sublampiran C.9), masing-masing untuk menghitung koefisien Clebsch-Gordan dan koefisien binomial. Kedua subprogram internal itu memanggil fungsi faktorial, yang menjadi subprogram eksternal. Sebuah blok `INTERFACE` untuk fungsi faktorial dinyatakan di bagian spesifikasi program utama, sehingga berlaku di seluruh bagian program utama. Di bagian spesifikasi fungsi clebsch maupun subrutin cobino masing-masing tidak perlu dinyatakan blok `INTERFACE` untuk fungsi faktorial.

```
RECURSIVE FUNCTION faktorial(n) RESULT(m)

IMPLICIT NONE
INTEGER, INTENT(IN) :: n
INTEGER :: m

IF (n==1 .OR. n==0) THEN
 m=1                  ! 0! = 1! = 1
ELSE IF (n>1) THEN
 m=n*faktorial(n-1)   ! m = n(n-1)(n-2)...1 = n!
ELSE
 STOP                 ! program berhenti jika n negatif
END IF

RETURN

END FUNCTION faktorial

PROGRAM cocgbino

IMPLICIT NONE
INTEGER :: j1,j2,j,m1,m2,m,i,p
INTEGER, DIMENSION(:), ALLOCATABLE :: b
```

```
INTERFACE
 RECURSIVE FUNCTION faktorial(n) RESULT(m)
 IMPLICIT NONE
 INTEGER, INTENT(IN) :: n
 INTEGER :: m
 END FUNCTION faktorial
END INTERFACE

! masukkan nilai-nilai

WRITE(*,*) 'Menghitung Koefisien Clebsch-Gordan'
WRITE(*,*) 'masukkan j1, j2, j, m1, m2, m'
READ(*,*) j1,j2,j,m1,m2,m

! tampilkan di layar
WRITE(*,"(1X,A,6I3)") 'j1, j2, j, m1, m2, m = ',j1,j2,j,m1,m2,m

! tampilkan koefisien Clebsch-Gordan
WRITE(*,*) 'koefisien Clebsch-Gordan = ',clebsch(j1,j2,j,m1,m2,m)

WRITE(*,*)

! masukkan nilai-nilai

WRITE(*,*) 'Menghitung Koefisien Binomial'
WRITE(*,*) 'masukkan nilai p'
READ(*,*) p

ALLOCATE(b(0:p))

CALL cobino(p,b)

! tampilkan hasil ke layar
WRITE(*,*)
WRITE(*,*) 'koefisien binomial:'
DO i=0,p
 WRITE(*,*) b(i)
END DO

DEALLOCATE(b)

STOP

CONTAINS
```

```
FUNCTION clebsch(j1,j2,j,m1,m2,m) RESULT(cg)

IMPLICIT NONE
INTEGER, PARAMETER :: dpr=KIND(1.0D0)
INTEGER, INTENT(IN) :: j1,j2,j,m1,m2,m
INTEGER :: minplus,k,kmin,kmax
REAL(dpr) :: cg,deretk,ff

! periksa hubungan nilai-nilai argumen, apakah memenuhi syarat

IF (j > j1+j2 .OR. j < ABS(j1-j2)) THEN
 cg=0.0_dpr
ELSE IF (m /= m1+m2) THEN
 cg=0.0_dpr
ELSE IF (ABS(m1) > j1) THEN
 cg=0.0_dpr
ELSE IF (ABS(m2) > j2) THEN
 cg=0.0_dpr
ELSE IF (ABS(m) > j) THEN
 cg=0.0_dpr
ELSE

! syarat dipenuhi, hitung koefisien Clebsch-Gordan

! hitung deret k

 kmin=MAX(0,-j+j2-m1,-j+j1+m2) ! batas bawah deret k
 kmax=MIN(j1+j2-j,j1-m1,j2+m2) ! batas atas deret k

 deretk=0.0_dpr ! inisialisasi nilai deret k
 minplus=(-1)**kmin ! inisialisasi (-1)^(k)
 DO k=kmin,kmax
  ff= faktorial(k)*faktorial(j1+j2-j-k) &
      *faktorial(j1-m1-k)*faktorial(j2+m2-k) &
      *faktorial(j-j2+m1+k)*faktorial(j-j1-m2+k)
  deretk=deretk+minplus/ff
  minplus=-minplus    ! untuk berikutnya, (-1)^(k+1)
 END DO

! koefisien Clebsch-Gordan

 cg= SQRT( (2.0_dpr*j+1.0_dpr) &
          *faktorial(j1+j2-j)*faktorial(j+j1-j2)*faktorial(j2+j-j1) &
          /faktorial(j1+j2+j+1) &
          *faktorial(j1+m1)*faktorial(j2+m2)*faktorial(j+m) &
          *faktorial(j1-m1)*faktorial(j2-m2)*faktorial(j-m))*deretk
```

```
END IF

RETURN

END FUNCTION clebsch

SUBROUTINE cobino(p,b)

IMPLICIT NONE
INTEGER, INTENT(IN) :: p
INTEGER, DIMENSION(0:p), INTENT(OUT) :: b
INTEGER :: i,n

! menghitung koefisien binomial

b(0)=1
DO n=1,p
 b(n)=1 ! inisialisasi
 DO i=0,n-1
  b(n)=b(n)*(p-i)
 END DO
 b(n)=b(n)/faktorial(n)
END DO

RETURN

END SUBROUTINE cobino

END PROGRAM cocgbino
```

Program di atas juga memberi contoh alokasi dinamis variabel array. Di program utama variabel b dideklarasikan sebagai variabel array rank 1 `INTEGER`, namun, bentuk dan ukurannya belum ditentukan. Bentuk dan ukuran variabel array b baru ditentukan kemudian ketika diperlukan, karena itu pada deklarasi variabel array b diberikan atribut `ALLOCATABLE`. Pada contoh di atas perintah alokasi variabel array b, yaitu `ALLOCATE(b(0:p))`, menentukan bentuk dan ukuran variabel array b, bahwa ada p+1 elemen variabel array b, dengan indeks yang dimulai dari 0 sampai p, yaitu b(0), b(1), ..., b(p-1), b(p). Setelah data yang tersimpan dalam variabel array b tidak diperlukan lagi, maka variabel array b didealokasi dengan perintah `DEALLOCATE(b)`.

C.13 Program dengan Modul

Program cocgbino di bawah ini memberikan nilai koefisien Clebsch-Gordan dan koefisien binomial. Koefisien Clebsch-Gordan dihitung oleh fungsi clebsch (Sublampiran

C.10) dan koefisien binomial oleh subrutin cobino (Sublampiran C.9). Fungsi clebsch dan subrutin cobino tersebut menjadi subprogram modul, yang ditempatkan dalam modul modkoefisien. Untuk menggunakannya, perintah `USE modkoefisien` diberikan di awal bagian spesifikasi program utama cocgbino, sebelum hal-hal lain, seperti variabel, konstanta, blok `INTERFACE`, dideklarasikan, sehingga fungsi clebsch dan subrutin cobino dikenal dan dapat dipanggil di bagian manapun di dalam program utama. Pada contoh ini modul modkoefisien ditempatkan bersama dengan program utama cocgbino dalam satu file. Dalam file itu modul modkoefisien harus ditempatkan sebelum program utama cocgbino, bukan setelahnya. Modul modkoefisien dapat juga disimpan dalam file terpisah dan, jika demikian, modul modkoefisien harus dikompilasi lebih dulu dari program cocgbino.

Baik fungsi clebsch maupun subrutin cobino keduanya memanggil fungsi faktorial (Sublampiran C.8), yang dalam contoh ini merupakan subprogram eksternal. Oleh karena itu, di bagian spesifikasi modul modkoefisien dinyatakan blok `INTERFACE` untuk fungsi faktorial. Blok `INTERFACE` itu berlaku di seluruh bagian modul modkoefisien, sehingga di bagian spesifikasi fungsi clebsch dan subrutin cobino masing-masing tidak perlu lagi dinyatakan blok `INTERFACE` untuk fungsi faktorial.

```
RECURSIVE FUNCTION faktorial(n) RESULT(m)

IMPLICIT NONE
INTEGER, INTENT(IN) :: n
INTEGER :: m

IF (n==1 .OR. n==0) THEN
 m=1                ! 0! = 1! = 1
ELSE IF (n>1) THEN
 m=n*faktorial(n-1)  ! m = n(n-1)(n-2)...1 = n!
ELSE
 STOP               ! program berhenti jika n negatif
END IF

RETURN

END FUNCTION faktorial

MODULE modkoefisien

INTERFACE
 RECURSIVE FUNCTION faktorial(n) RESULT(m)
 IMPLICIT NONE
 INTEGER, INTENT(IN) :: n
 INTEGER :: m
 END FUNCTION faktorial
END INTERFACE
```

```
CONTAINS

FUNCTION clebsch(j1,j2,j,m1,m2,m) RESULT(cg)

IMPLICIT NONE
INTEGER, PARAMETER :: dpr=KIND(1.0D0)
INTEGER, INTENT(IN) :: j1,j2,j,m1,m2,m
INTEGER :: minplus,k,kmin,kmax
REAL(dpr) :: cg,deretk,ff

! periksa hubungan nilai-nilai argumen, apakah memenuhi syarat

IF (j > j1+j2 .OR. j < ABS(j1-j2)) THEN
 cg=0.0_dpr
ELSE IF (m /= m1+m2) THEN
 cg=0.0_dpr
ELSE IF (ABS(m1) > j1) THEN
 cg=0.0_dpr
ELSE IF (ABS(m2) > j2) THEN
 cg=0.0_dpr
ELSE IF (ABS(m) > j) THEN
 cg=0.0_dpr
ELSE

! syarat dipenuhi, hitung koefisien Clebsch-Gordan

! hitung deret k

 kmin=MAX(0,-j+j2-m1,-j+j1+m2) ! batas bawah deret k
 kmax=MIN(j1+j2-j,j1-m1,j2+m2) ! batas atas deret k

 deretk=0.0_dpr ! inisialisasi nilai deret k
 minplus=(-1)**kmin ! inisialisasi (-1)^(k)
 DO k=kmin,kmax
  ff= faktorial(k)*faktorial(j1+j2-j-k) &
     *faktorial(j1-m1-k)*faktorial(j2+m2-k) &
     *faktorial(j-j2+m1+k)*faktorial(j-j1-m2+k)
  deretk=deretk+minplus/ff
  minplus=-minplus    ! untuk berikutnya, (-1)^(k+1)
 END DO

! koefisien Clebsch-Gordan

 cg= SQRT( (2.0_dpr*j+1.0_dpr) &
         *faktorial(j1+j2-j)*faktorial(j+j1-j2)*faktorial(j2+j-j1) &
```

```
          /faktorial(j1+j2+j+1) &
          *faktorial(j1+m1)*faktorial(j2+m2)*faktorial(j+m) &
          *faktorial(j1-m1)*faktorial(j2-m2)*faktorial(j-m))*deretk

END IF

RETURN

END FUNCTION clebsch

SUBROUTINE cobino(p,b)

IMPLICIT NONE
INTEGER, INTENT(IN) :: p
INTEGER, DIMENSION(0:p), INTENT(OUT) :: b
INTEGER :: i,n

! menghitung koefisien binomial

b(0)=1
DO n=1,p
 b(n)=1 ! inisialisasi
 DO i=0,n-1
  b(n)=b(n)*(p-i)
 END DO
 b(n)=b(n)/faktorial(n)
END DO

RETURN

END SUBROUTINE cobino

END MODULE modkoefisien

PROGRAM cocgbino

USE modkoefisien

IMPLICIT NONE
INTEGER :: j1,j2,j,m1,m2,m,i,p
INTEGER, DIMENSION(:), ALLOCATABLE :: b

! masukkan nilai-nilai

WRITE(*,*) 'Menghitung Koefisien Clebsch-Gordan'
WRITE(*,*) 'masukkan j1, j2, j, m1, m2, m'
```

```
READ(*,*) j1,j2,j,m1,m2,m

! tampilkan di layar
WRITE(*,"(1X,A,6I3)") 'j1, j2, j, m1, m2, m = ',j1,j2,j,m1,m2,m

! tampilkan koefisien Clebsch-Gordan
WRITE(*,*) 'koefisien Clebsch-Gordan = ',clebsch(j1,j2,j,m1,m2,m)

WRITE(*,*)

! masukkan nilai-nilai

WRITE(*,*) 'Menghitung Koefisien Binomial'
WRITE(*,*) 'masukkan nilai p'
READ(*,*) p

ALLOCATE(b(0:p))

CALL cobino(p,b)

! tampilkan hasil ke layar
WRITE(*,*)
WRITE(*,*) 'koefisien binomial:'
DO i=0,p
 WRITE(*,*) b(i)
END DO

DEALLOCATE(b)

STOP

END PROGRAM cocgbino
```

Dalam modul modkoefisien di atas konstanta dpr dideklarasikan di fungsi clebsch, sehingga hanya berlaku di dalam fungsi clebsch. Apabila dikehendaki konstanta dpr berlaku di seluruh bagian modul modkoefisien, namun tidak di unit / subunit program lain yang menggunakan modul modkoefisien, maka deklarasi konstanta dpr dipindahkan ke bagian spesifikasi modul modkoefisien, dengan tambahan atribut `PRIVATE`. Dengan begitu, bagian spesifikasi modul modkoefisien dan fungsi clebsch pada contoh program di atas berubah menjadi seperti berikut:

```
...

MODULE modkoefisien

IMPLICIT NONE
```

```
INTEGER, PARAMETER, PRIVATE :: dpr=KIND(1.0D0)

INTERFACE
 RECURSIVE FUNCTION faktorial(n) RESULT(m)
 IMPLICIT NONE
 INTEGER, INTENT(IN) :: n
 INTEGER :: m
 END FUNCTION faktorial
END INTERFACE

CONTAINS

FUNCTION clebsch(j1,j2,j,m1,m2,m) RESULT(cg)

IMPLICIT NONE
INTEGER, INTENT(IN) :: j1,j2,j,m1,m2,m
INTEGER :: minplus,k,kmin,kmax
REAL(dpr) :: cg,deretk,ff

...
```

Jika atribut `PRIVATE` tidak diberikan pada deklarasi konstanta dpr, maka konstanta dpr itu tidak hanya berlaku di seluruh bagian modul modkoefisien, melainkan juga di semua unit / subunit program lain yang menggunakan modul modkoefisien.

Kini, mari kita lakukan dua hal sebagai berikut:

- Blok `INTERFACE` untuk fungsi eksternal faktorial dinyatakan dalam satu modul tersendiri, sebut saja modul modfaktorial. Dengan demikian, blok `INTERFACE` untuk fungsi faktorial tersebut dapat dipakai di sembarang unit / subunit program dengan memberikan perintah `USE modfaktorial` di awal bagian spesifikasi unit / subunit program itu.

- Deklarasi konstanta dpr ditaruh dalam satu modul terpisah, misalkan saja modul modvarpar. Dengan begitu, konstanta dpr dapat dipakai di sembarang unit / subunit program yang melibatkan data `REAL` berpresisi ganda, apabila unit / subunit program tersebut berisi perintah `USE modvarpar` di awal bagian spesifikasinya. Perhatikan bahwa dalam hal ini deklarasi konstanta dpr dalam modul modvarpar harus tanpa disertai atribut `PRIVATE`.

Sesuai dengan dua hal di atas contoh program kita dapat dibuat menjadi seperti di bawah ini. Kita berikan perintah `USE modfaktorial` dan `USE modvarpar` di awal bagian spesifikasi modul modkoefisien. Dalam satu file yang sama modul modfaktorial dan modul modvarpar diletakkan sebelum modul modkoefisien.

```
RECURSIVE FUNCTION faktorial(n) RESULT(m)

IMPLICIT NONE
INTEGER, INTENT(IN) :: n
INTEGER :: m

IF (n==1 .OR. n==0) THEN
 m=1                 ! 0! = 1! = 1
ELSE IF (n>1) THEN
 m=n*faktorial(n-1)  ! m = n(n-1)(n-2)...1 = n!
ELSE
 STOP                ! program berhenti jika n negatif
END IF

RETURN

END FUNCTION faktorial

MODULE modfaktorial

INTERFACE
 RECURSIVE FUNCTION faktorial(n) RESULT(m)
 IMPLICIT NONE
 INTEGER, INTENT(IN) :: n
 INTEGER :: m
 END FUNCTION faktorial
END INTERFACE

END MODULE modfaktorial

MODULE modvarpar

IMPLICIT NONE
INTEGER, PARAMETER :: dpr=KIND(1.0D0)

END MODULE modvarpar

MODULE modkoefisien

USE modfaktorial
USE modvarpar

CONTAINS
```

```
FUNCTION clebsch(j1,j2,j,m1,m2,m) RESULT(cg)

IMPLICIT NONE
INTEGER, INTENT(IN) :: j1,j2,j,m1,m2,m
INTEGER :: minplus,k,kmin,kmax
REAL(dpr) :: cg,deretk,ff

! periksa hubungan nilai-nilai argumen, apakah memenuhi syarat

IF (j > j1+j2 .OR. j < ABS(j1-j2)) THEN
 cg=0.0_dpr
ELSE IF (m /= m1+m2) THEN
 cg=0.0_dpr
ELSE IF (ABS(m1) > j1) THEN
 cg=0.0_dpr
ELSE IF (ABS(m2) > j2) THEN
 cg=0.0_dpr
ELSE IF (ABS(m) > j) THEN
 cg=0.0_dpr
ELSE

! syarat dipenuhi, hitung koefisien Clebsch-Gordan

! hitung deret k

 kmin=MAX(0,-j+j2-m1,-j+j1+m2) ! batas bawah deret k
 kmax=MIN(j1+j2-j,j1-m1,j2+m2) ! batas atas deret k

 deretk=0.0_dpr ! inisialisasi nilai deret k
 minplus=(-1)**kmin ! inisialisasi (-1)^(k)
 DO k=kmin,kmax
  ff= faktorial(k)*faktorial(j1+j2-j-k) &
     *faktorial(j1-m1-k)*faktorial(j2+m2-k) &
     *faktorial(j-j2+m1+k)*faktorial(j-j1-m2+k)
  deretk=deretk+minplus/ff
  minplus=-minplus    ! untuk berikutnya, (-1)^(k+1)
 END DO

! koefisien Clebsch-Gordan

 cg= SQRT( (2.0_dpr*j+1.0_dpr) &
          *faktorial(j1+j2-j)*faktorial(j+j1-j2)*faktorial(j2+j-j1) &
          /faktorial(j1+j2+j+1) &
          *faktorial(j1+m1)*faktorial(j2+m2)*faktorial(j+m) &
          *faktorial(j1-m1)*faktorial(j2-m2)*faktorial(j-m))*deretk
```

```
END IF

RETURN

END FUNCTION clebsch

SUBROUTINE cobino(p,b)

IMPLICIT NONE
INTEGER, INTENT(IN) :: p
INTEGER, DIMENSION(0:p), INTENT(OUT) :: b
INTEGER :: i,n

! menghitung koefisien binomial

b(0)=1
DO n=1,p
 b(n)=1 ! inisialisasi
 DO i=0,n-1
  b(n)=b(n)*(p-i)
 END DO
 b(n)=b(n)/faktorial(n)
END DO

RETURN

END SUBROUTINE cobino

END MODULE modkoefisien

PROGRAM cocgbino

USE modkoefisien

IMPLICIT NONE
INTEGER :: j1,j2,j,m1,m2,m,i,p
INTEGER, DIMENSION(:), ALLOCATABLE :: b

! masukkan nilai-nilai

WRITE(*,*) 'Menghitung Koefisien Clebsch-Gordan'
WRITE(*,*) 'masukkan j1, j2, j, m1, m2, m'
READ(*,*) j1,j2,j,m1,m2,m

! tampilkan di layar
WRITE(*,"(1X,A,6I3)") 'j1, j2, j, m1, m2, m = ',j1,j2,j,m1,m2,m
```

```
! tampilkan koefisien Clebsch-Gordan
WRITE(*,*) 'koefisien Clebsch-Gordan = ',clebsch(j1,j2,j,m1,m2,m)

WRITE(*,*)

! masukkan nilai-nilai

WRITE(*,*) 'Menghitung Koefisien Binomial'
WRITE(*,*) 'masukkan nilai p'
READ(*,*) p

ALLOCATE(b(0:p))

CALL cobino(p,b)

! tampilkan hasil ke layar
WRITE(*,*)
WRITE(*,*) 'koefisien binomial:'
DO i=0,p
 WRITE(*,*) b(i)
END DO

DEALLOCATE(b)

STOP

END PROGRAM cocgbino
```

C.14 Program yang Menggunakan Subprogram sebagai Argumen

Di sini ditunjukkan sebuah program untuk menghitung integral menurut metode integrasi numerik Trapezoid komposit, yaitu:

$$\int_a^b f(x)dx \simeq \frac{1}{2}h\left\{f(a)+f(b)\right\}+h\sum_{i=1}^{n-1}f(x_i)\,,$$

dengan $h = (b-a)/n$ dan $x_i = a + ih$. Dalam program itu diberikan contoh penggunaan subprogram sebagai argumen subprogram lain. Program tersebut terdiri dari program utama integral, modul modvarpar (Sublampiran C.13), fungsi eksternal trapezoid, dan fungsi eksternal kubik. Fungsi trapezoid menghitung integral sesuai rumus di atas, sementara fungsi kubik menghitung nilai integrand $f(x)$, yang dalam contoh ini

merupakan suatu polinomial orde 3:

$$f(x) = -x(x^2 - 9x + 1)\,.$$

Fungsi trapezoid menerima fungsi kubik sebagai salah satu argumen aktual, yang dihubungkan dengan argumen *dummy* integrand, yang juga merupakan sebuah fungsi. Di bagian spesifikasi fungsi trapezoid fungsi integrand dideklarasikan dengan menggunakan blok `INTERFACE`. Perhatikan bahwa tidak diperlukan atribut `INTENT` untuk fungsi integrand. Blok `INTERFACE` untuk fungsi integrand berisi bagian kepala fungsi kubik, dengan catatan bahwa nama fungsi yang dicantumkan bukanlah kubik, melainkan integrand.

Program utama integral memanggil fungsi trapezoid, dengan memasukkan fungsi kubik sebagai salah satu argumen aktual. Karena itu, di bagian spesifikasi program utama integral harus dinyatakan blok `INTERFACE` untuk masing-masing fungsi kubik dan fungsi trapezoid. Perhatikan bahwa blok `INTERFACE` untuk fungsi trapezoid juga memuat blok `INTERFACE` untuk fungsi integrand. Ini dapat dimengerti bahwa mengingat fungsi integrand merupakan salah satu argumen *dummy* fungsi trapezoid, maka pernyataan blok `INTERFACE` untuk fungsi integrand juga termasuk bagian kepala fungsi trapezoid.

```
MODULE modvarpar

IMPLICIT NONE
INTEGER, PARAMETER :: dpr=KIND(1.0D0)

END MODULE modvarpar

FUNCTION kubik(x) RESULT(y)

USE modvarpar

IMPLICIT NONE
REAL(dpr), INTENT(IN) :: x
REAL(dpr) :: y

y=-x*(x**2-9.0_dpr*x+1.0_dpr)

RETURN

END FUNCTION kubik

FUNCTION trapezoid(n,a,b,integrand) RESULT(y)

USE modvarpar

IMPLICIT NONE
INTEGER, INTENT(IN) :: n
REAL(dpr), INTENT(IN) :: a,b
```

```
INTEGER :: i
REAL(dpr) :: y,h

INTERFACE
 FUNCTION integrand(x) RESULT(y)
 USE modvarpar
 IMPLICIT NONE
 REAL(dpr), INTENT(IN) :: x
 REAL(dpr) :: y
 END FUNCTION integrand
END INTERFACE

h=(b-a)/n

y=0.5_dpr*(integrand(a)+integrand(b))
DO i=1,n-1
 y=y+integrand(a+i*h)
END DO
y=y*h

RETURN

END FUNCTION trapezoid

PROGRAM integral

USE modvarpar

IMPLICIT NONE
INTEGER :: n
REAL(dpr) :: a,b,y

INTERFACE
 FUNCTION kubik(u) RESULT(v)
 USE modvarpar
 IMPLICIT NONE
 REAL(dpr), INTENT(IN) :: u
 REAL(dpr) :: v
 END FUNCTION kubik
END INTERFACE

INTERFACE
 FUNCTION trapezoid(n,a,b,integrand) RESULT(y)
 USE modvarpar
 IMPLICIT NONE
 INTEGER, INTENT(IN) :: n
```

```
 REAL(dpr), INTENT(IN) :: a,b
 REAL(dpr) :: y
 INTERFACE
  FUNCTION integrand(u) RESULT(v)
  USE modvarpar
  IMPLICIT NONE
  REAL(dpr), INTENT(IN) :: u
  REAL(dpr) :: v
  END FUNCTION integrand
 END INTERFACE
 END FUNCTION trapezoid
END INTERFACE

WRITE(*,*) 'masukkan batas bawah dan batas atas integral'
READ(*,*) a,b

WRITE(*,*) 'masukkan jumlah titik evaluasi'
READ(*,*) n

y=trapezoid(n,a,b,kubik)

WRITE(*,*) 'hasil integral = ',y

STOP

END PROGRAM integral
```

Metode Trapezoid merupakan satu dari beberapa metode integrasi numerik yang dikenal orang. Bisa saja kita buat sebuah modul yang berisi fungsi-fungsi untuk metode-metode integrasi numerik itu, salah satunya adalah fungsi trapezoid. Jadi, mari kita tempatkan fungsi trapezoid dalam sebuah modul, sebut saja modul quadrature. Dengan begitu, program di atas menjadi seperti di bawah. Di bagian spesifikasi program utama integral tidak perlu lagi dinyatakan blok `INTERFACE` untuk fungsi trapezoid. Sebagai gantinya, di awal bagian spesifikasi program utama integral diberikan perintah `USE quadrature`. Perhatikan bahwa fungsi integrand merupakan argumen *dummy* fungsi trapezoid, karena itu blok `INTERFACE` untuk fungsi integrand harus dinyatakan di bagian spesifikasi fungsi trapezoid. Apabila modul quadrature berisi beberapa fungsi untuk metode integrasi numerik lain dan tiap fungsi memiliki argumen *dummy* integrand, maka blok `INTERFACE` untuk fungsi integrand harus dinyatakan di bagian spesifikasi tiap fungsi tersebut dan tidak dapat dinyatakan hanya di bagian spesifikasi modul quadrature.

```
MODULE modvarpar

IMPLICIT NONE
INTEGER, PARAMETER :: dpr=KIND(1.0D0)
```

```
END MODULE modvarpar

FUNCTION kubik(x) RESULT(y)

USE modvarpar

IMPLICIT NONE
REAL(dpr), INTENT(IN) :: x
REAL(dpr) :: y

y=-x*(x**2-9.0_dpr*x+1.0_dpr)

RETURN

END FUNCTION kubik

MODULE quadrature

USE modvarpar

CONTAINS

FUNCTION trapezoid(n,a,b,integrand) RESULT(y)

IMPLICIT NONE
INTEGER, INTENT(IN) :: n
REAL(dpr), INTENT(IN) :: a,b
INTEGER :: i
REAL(dpr) :: y,h

INTERFACE
 FUNCTION integrand(x) RESULT(y)
 USE modvarpar
 IMPLICIT NONE
 REAL(dpr), INTENT(IN) :: x
 REAL(dpr) :: y
 END FUNCTION integrand
END INTERFACE

h=(b-a)/n

y=0.5_dpr*(integrand(a)+integrand(b))
DO i=1,n-1
 y=y+integrand(a+i*h)
END DO
```

```
y=y*h

RETURN

END FUNCTION trapezoid

END MODULE quadrature

PROGRAM integral

USE modvarpar
USE quadrature

IMPLICIT NONE
INTEGER :: n
REAL(dpr) :: a,b,y

INTERFACE
 FUNCTION kubik(u) RESULT(v)
 USE modvarpar
 IMPLICIT NONE
 REAL(dpr), INTENT(IN) :: u
 REAL(dpr) :: v
 END FUNCTION kubik
END INTERFACE

WRITE(*,*) 'masukkan batas bawah dan batas atas integral'
READ(*,*) a,b

WRITE(*,*) 'masukkan jumlah titik evaluasi'
READ(*,*) n

y=trapezoid(n,a,b,kubik)

WRITE(*,*) 'hasil integral = ',y

STOP

END PROGRAM integral
```

Dalam program di bawah kita jadikan fungsi kubik sebagai sebuah subprogram modul, yang disimpan dalam modul, sebut saja modul fmath. Fungsi kubik dipanggil oleh program utama integral. Karena itu, perintah `USE fmath` harus dicantumkan di awal bagian spesifikasi program utama integral. Namun, perintah `USE fmath` tidak perlu dicantumkan di awal bagian spesifikasi fungsi trapezoid atau modul quadrature, karena fungsi kubik tidak dipanggil secara langsung dari dalam fungsi trapezoid atau modul qu-

adrature. Fungsi trapezoid memanggil fungsi integrand sebagai argumen *dummy*, yang dalam hal ini dihubungkan dengan fungsi kubik sebagai argumen aktual.

```
MODULE modvarpar

IMPLICIT NONE
INTEGER, PARAMETER :: dpr=KIND(1.0D0)

END MODULE modvarpar

MODULE fmath

USE modvarpar

CONTAINS

FUNCTION kubik(x) RESULT(y)

IMPLICIT NONE
REAL(dpr), INTENT(IN) :: x
REAL(dpr) :: y

y=-x*(x**2-9.0_dpr*x+1.0_dpr)

RETURN

END FUNCTION kubik

END MODULE fmath

MODULE quadrature

USE modvarpar

CONTAINS

FUNCTION trapezoid(n,a,b,integrand) RESULT(y)

IMPLICIT NONE
INTEGER, INTENT(IN) :: n
REAL(dpr), INTENT(IN) :: a,b
INTEGER :: i
REAL(dpr) :: y,h

INTERFACE
 FUNCTION integrand(x) RESULT(y)
```

```
 USE modvarpar
 IMPLICIT NONE
 REAL(dpr), INTENT(IN) :: x
 REAL(dpr) :: y
 END FUNCTION integrand
END INTERFACE

h=(b-a)/n

y=0.5_dpr*(integrand(a)+integrand(b))
DO i=1,n-1
 y=y+integrand(a+i*h)
END DO
y=y*h

RETURN

END FUNCTION trapezoid

END MODULE quadrature

PROGRAM integral

USE modvarpar
USE fmath
USE quadrature

IMPLICIT NONE
INTEGER :: n
REAL(dpr) :: a,b,y

WRITE(*,*) 'masukkan batas bawah dan batas atas integral'
READ(*,*) a,b

WRITE(*,*) 'masukkan jumlah titik evaluasi'
READ(*,*) n

y=trapezoid(n,a,b,kubik)

WRITE(*,*) 'hasil integral = ',y

STOP

END PROGRAM integral
```

Daftar Pustaka

Boas, M. L., 2006. *Mathematical Methods in The Physical Sciences, 3rd Ed.*. New York: John Wiley & Sons, Inc.

DeVries, P. L. 1994. *A First Course in Computational Physics*. New York: Wiley.

Edmonds, A. R., 1996. *Angular Momentum in Quantum Mechanics*. Princeton: Princeton University Press.

Metcalf, M., dan J. Reid. 1998. *Fortran 90/95 Explained*. New York: Oxford University Press.

Press, W. H., dkk. 1992. *Numerical Recipes in Fortran*. New York: Cambridge University Press.

Rose, M. E., 1957. *Elementary Theory of Angular Momentum*. New York: John Wiley & Sons, Inc.

Indeks

A
ALLOCATABLE, 67, 119
ALLOCATE, 67, 119

B
bagian spesifikasi, 6
baris program
  baris kosong, 7
  case insensitive, 7
  lebih dari satu baris, 8
  rata kiri, rata kanan, rata kiri-kanan, 8

C
CALL, 46, 47
CASE, 33
CASE DEFAULT, 33
CLOSE, 52
compiler, 6, 72, 81
CONTAINS, 6, 10, 106
CYCLE, 40

D
data, 2, 11
  array, 14
    bentuk, 14
    elemen data array, 14
    jumlah dimensi, 14
    perintah pembentuk array rank 1, 15, 71
    rank, 14
    rank 1, 15
    sumbu dimensi, 14
    ukuran, 14
    urutan elemen data array, 14
  skalar, 14
  tipe, 8, 87, 88
    intrinsik, 11
    turunan, 11
data CHARACTER, 12
  ekspresi, 12
  panjang data, 13
data COMPLEX, 12
  ekspresi, 12
data INTEGER, 12
  ekspresi, 12
data LOGICAL, 13
  ekspresi, 13
  nilai data, 13
    .FALSE., 13, 17, 18, 27, 30, 68, 70
    .TRUE., 13, 17, 18, 27, 30, 68, 70
data REAL, 11
  ekspresi, 11
  tanda desimal, 12
DEALLOCATE, 67, 68, 119
deklarasi
  konstanta, 6
    array rank 1, 18
    skalar, 17
  variabel, 6
    array, 17
    skalar, 16
DIMENSION
  atribut, 17, 18
  batas bawah dan batas atas, 18
DO
  konstruksi, 37
    bersusun, 38
    tanpa variabel pengulangan, 41
    variabel pengulangan, 37
  lingkaran, 37
  pernyataan, 37

E
eksekusi program, 8
ELSE, 31
ELSE IF, 32
END DO, 37
END FUNCTION, 10, 46
END IF, 31
END PROGRAM, 5
END SELECT, 33

END SUBROUTINE, 10, 45
executable file, 1
EXIT, 39, 41

F
FORMAT, 56
Fortran, 1
Fortran 66, 1
Fortran 77, 1
Fortran 90, 1, 68
Fortran 90/95, 1
  file program, 81
Fortran 95, 1, 68
Fortran Berkinerja Tinggi (*High Performance Fortran / HPF*), 1
FUNCTION, 10, 46, 47

G
GOTO, 29

I
IF
  konstruksi, 31, 70
    struktur, 32
  pernyataan, 30, 31, 40
IMPLICIT NONE, 8
  nama konstanta, 8
  nama variabel, 8
INTENT, 45, 77
  IN, 45, 47, 68
  INOUT, 46, 68
  OUT, 45, 68
INTERFACE, 43, 72, 75, 76, 108, 111, 113, 116, 120, 129, 131

K
kind type parameter, 65, 87, 88
komentar, 8
kompilasi dan penautan program, 1, 81
kompilasi dan penautan program secara terpisah, 83
  file obyek, 83, 84
  library, 84
konstanta, 2, 11, 16
  global, 10
  lokal, 10
  tipe, 16, 17
konstanta array, 16
konstanta skalar, 16

L
label, 29, 56

M
masukan & luaran, 2
memori komputer, 66
modul, 43, 68, 70, 74, 120, 128, 131
  isi, 75
  struktur, 74
modus teks, 82
  command-line interface, 82
    command prompt (windows), 82
    terminal (linux), 82

O
OPEN, 52
operator, 2, 23
  buatan, 23
  diadik, 23
  hirarki, 23, 24
  intrinsik, 2, 23
  jenis operasi, 23
  monadik, 23
  tipe data hasil operasi, 24
  tipe data operand, 24
OPTIONAL
  atribut, 68, 77
  pernyataan, 77

P
PARAMETER, 17
pengulangan, 2, 37
percabangan, 2, 29
pernyataan [nilai1]:[nilai2], 18
pernyataan [variabel]=[nilai1], [nilai2], 16
pernyataan [variabel]=[nilai1], [nilai2],[langkah], 16
presisi
  ganda, 65, 103
    data COMPLEX, 66
    data REAL, 65
    konstanta COMPLEX, 66
    konstanta REAL, 65
    variabel COMPLEX, 66
    variabel REAL, 65

tunggal, 65
PRIVATE, 75, 123
PROGRAM, 5
program
unit, 1, 5
modul, 1, 5
program utama, 1, 5
subprogram eksternal, 1, 5
program utama
struktur, 5
dengan subprogram internal, 6, 9
tanpa subprogram internal, 6, 7
proses *I/O*, 49
daftar luaran, 50
daftar masukan, 50
format, 50
redirection, 51
tak terformat, 49, 50, 51, 53
terformat, 49, 50, 53
deskriptor edit, 56, 58
deskriptor edit data, 58
deskriptor edit kontrol, 61
deskriptor edit yang berulang, 63
format bebas, 53
format yang ditentukan, 53, 56
karakter pertama data luaran, 64
pengelompokan deskriptor edit, 63
unit, 50
file eksternal, 52
file internal, 51
keyboard & monitor, 51
nomor unit, 52

R

READ, 50
RECURSIVE, 47, 72, 75
RETURN, 10, 46, 47, 68, 70

S

SAVE, 68, 70, 71, 105
SELECT CASE
konstruksi, 33
struktur, 33
variabel selektor, 33
pernyataan, 33
source code, 81
STOP, 8, 10
subprogram, 2, 6, 43
argumen, 44
aktual, 44, 76, 129
dummy, 44, 67, 68, 76, 129
argumen *dummy*
dihubungkan dengan argumen aktual berdasarkan kata kunci, 69, 85
dihubungkan dengan argumen aktual berdasarkan posisi, 44
pilihan, 44, 68, 85
fungsi, 44
struktur, 46
subrutin, 44
struktur, 44
subprogram eksternal, 43, 71, 76, 108, 111, 113, 116, 120, 128
struktur fungsi, 71
struktur subrutin, 71
subprogram internal, 10, 43, 71, 74, 106, 116
struktur fungsi, 46
struktur subrutin, 44
subprogram intrinsik, 2, 44, 85
fungsi, 87
ABS, 89
ACOS, 87
AIMAG, 89
ALLOCATED, 68
ASIN, 87
ATAN, 87
CMPLX, 20, 89
CONJG, 89
COS, 87
COSH, 87
EXP, 87
INT, 90
KIND, 65
LOG, 87
LOG10, 87
MAX, 90, 104
MAXVAL, 88
MIN, 90, 104
MINVAL, 88
PRESENT, 70
PRODUCT, 89
REAL, 90
RESHAPE, 15

SIN, 87
SINH, 88
SQRT, 88
SUM, 89
TAN, 88
TANH, 88
subrutin, 85
CPU_TIME, 85
DATE_AND_TIME, 85
RANDOM_NUMBER, 86
RANDOM_SEED, 86
SYSTEM_CLOCK, 86
subprogram modul, 5, 43, 74, 76, 120, 133
struktur fungsi, 74
struktur subrutin, 74
subprogram rekursif, 47
struktur, 47
SUBROUTINE, 10, 45, 47

U
USE, 75, 120, 131, 133

V
variabel, 2, 11, 16
global, 10
lokal, 10, 44, 45, 47, 68, 70
tipe, 16
variabel array, 16
alokasi dinamis, 67, 119
status alokasi, 68
teralokasikan, 67, 71
variabel pointer, 16
variabel skalar, 16

W
WRITE, 50