**BAB III**

**STRUKTUR PENGULANGAN**

**3.1. Pendahuluan**

Dalam bab sebelumnya sudah dibahas 2 macam struktur logika, yaitu struktur sekuensial dan struktur seleksi. Struktur logika yang ketiga dalam metode pemrograman terstruktur adalah struktur pengulangan atau *looping* (*repetition*). Struktur ini digunakan untuk mengulangi eksekusi sebuah atau sekelompok instruksi. Sehari-hari sering ditemui pekerjaan yang sama dilakukan berulang-ulang untuk sekelompok data seperti contohnya menghitung gaji untuk sejumlah pegawai di akhir minggu atau bulan, menghitung nilai akhir dari sejumlah mahasiswa di akhir semester, dan sebagainya. Pekerjaan semacam ini cocok untuk dikerjakan komputer karena komputer tidak akan merasa bosan atau lelah seperti manusia. Struktur pengulangan merupakan salah satu struktur pemrograman yang penting. Ada macam-macam bentuk struktur pengulangan yang akan dibahas yaitu *conditional loop, counted loop* dan pengulangan bertingkat (*nested loop*).

Sebagai contoh, lihat algoritma untuk menampilkan kata “HALLO” sebanyak 3 kali.

Algoritma HALLO3

Algoritma untuk menampilkan kata HALLO sebanyak 3 kali. Variabel N bertipe integer.

* 1. [initialisasi variabel]

N = 1

* 1. [membuat pengulangan]

While ( N ≤ 3 )

{

2.a [menampilkan kata]

Write (“HALLO”)

2.b [increment N]

N = N + 1

}

* 1. [selesai]

Halt

Dalam algoritma di atas, langkah 2.a dan 2.b yang ada di dalam instruksi WHILE (langkah 2) akan diulangi selama nilai N lebih kecil atau sama dengan 3 atau dengan kata lain kondisi pada instruksi WHILE bernilai TRUE. Dalam kasus ini, setelah langkah 2.b, kontrol program akan kembali ke langkah 2 untuk memeriksa apakah kondisi masih bernilai TRUE. Jika kondisi bernilai FALSE atau jika N > 3, maka kontrol program akan menuju ke langkah berikutnya, yaitu langkah 3. Langkah 2.a dan 2.b di atas adalah instruksi-instruksi yang akan diulangi selama kondisi pada intruksi While bernilai TRUE, instruksi ini disebut sebagai rentang pengulangan (*range of loop*). Untuk memudahkan pembacaan algoritma, semua contoh-contoh algoritma yang akan dibahas nanti akan menggunakan tanda buka kurung yang besar untuk menandai instruksi yang berada dalam rentang pengulangan.

Untuk mengetahui apakah proses pengulangan dalam algoritma sudah sesuai dengan kebutuhan, maka dapat dilakukan dengan membuat:

1. Tabel telusur (*trace table*): tabel yang berisi perubahan nilai variabel-variabel ketika algoritma dieksekusi mulai dari langkah pertama sampai terakhir.
2. *Loop behaviour table*: mirip seperti tabel telusur, tetapi tabel ini hanya berisi perubahan nilai variabel-variabel yang ada dalam suatu pengulangan ketika pengulangan tersebut dieksekusi. Pengulangan ke 0 adalah keadaan tepat sebelum pengulangan dieksekusi.

Sebagai contoh dibuat tabel telusur untuk algoritma HALLO (lihat tabel 3.1) dan loop behavior table-nya (lihat tabel 3.2) di bawah ini.

Tabel 3.1 Tabel telusur untuk Algoritma HALLO

|  |  |  |  |
| --- | --- | --- | --- |
| Langkah | N | Kondisi While | Output |
| 1 | 1 |  |  |
| 2 | 1 | True |  |
| 2.a | 1 |  | HALLO |
| 2.b | 2 |  |  |
| 2 | 2 | True |  |
| 2.a | 2 |  | HALLO |
| 2.b | 3 |  |  |
| 2 | 3 | True |  |
| 2.a | 3 |  | HALLO |
| 2.b | 4 |  |  |
| 2 | 4 | False |  |
| 3 | Selesai |  |  |

Pada tabel telusur di atas, dapat dilihat bahwa instruksi-instruksi yang berada dalam rentang pengulangan dieksekusi berkali-kali sampai kondisi menjadi FALSE. Perhatikan deretan langkah 2, 2.a, dan 2.b. Langkah 2.a dan 2.b akan dikerjakan selama kondisi pada WHILE bernilai TRUE. Jika nilai pada kondisi pada WHILE menjadi FALSE maka langkah 2.a dan 2.b tidak dikerjakan dan eksekusi akan dilanjutkan ke langkah 3.

Tabel 3.2 Loop behavior table untuk algoritma HALLO

|  |  |  |
| --- | --- | --- |
| Loop | N | Output |
| 0 | 1 |  |
| 1 | 2 | HALLO |
| 2 | 3 | HALLO |
| 3 | 4 | HALLO |

Sesuai namanya, tabel ini hanya berisi perubahan nilai variabel setelah melewati loop, tidak terlihat pada langkah berapa terjadi perubahan nilai tersebut. Dengan demikian dapat dikatakan tabel ini merupakan ringkasan dari tabel telusur (tabel 3.1).

Sebagai contoh berikutnya, akan dibuat algoritma untuk menghitung faktorial. Untuk suatu nilai N, maka faktorial dari N (biasanya ditulis N!) adalah hasil kali dari N(N-1)(N-2) ... (2)(1). Ini merupakan deret dari perkalian yang berulang-ulang yang pengalinya dikurangi satu sebelum dilakukan perkalian lagi. Penyelesaian dengan algoritma adalah dengan menggunakan pengulangan seperti algoritma berikut ini.

Algoritma FAKTORIAL

Algoritma untuk menghitung faktorial dari suatu nilai yang diinputkan. Variabel N, FAKT, dan PENGALI bertipe integer.

1. [memasukkan input]

Read(PENGALI)

1. [initialisasi variabel]

FAKT = 1

1. [membuat pengulangan]

While ( PENGALI ≥ 1 )

{

3.a [menghitung hasil perkalian]

FAKT = FAKT \* PENGALI

3.b [mengurangi pengali]

PENGALI = PENGALI – 1

}

1. [menampilkan hasil]

Write(“Faktorial dari “, N,” adalah “, FAKT)

1. [selesai]

Halt

Dalam contoh di atas, banyaknya pengulangan dikontrol oleh instruksi WHILE di langkah ke 3. Instruksi ini akan menyebabkan eksekusi langkah 3.a dan 3.b (*range of loop*) diulangi selama nilai PENGALI lebih besar atau sama dengan 1 atau kondisi dalam instruksi WHILE = = TRUE. Jika suatu saat kondisi menjadi FALSE maka eksekusi dilanjutkan dengan langkah setelah rentang pengulangan, dalam hal ini langkah ke 4.

Untuk mengetahui apakah proses pengulangan dalam algoritma di atas sudah sesuai maka dibuat tabel telusur dan *loop behavior table* dari algoritma FAKTORIAL untuk nilai N = 4, lihat tabel 3.3 dan tabel 3.4 berikut ini.

**Tabel 3.3 Tabel telusur dari Algoritma FAKTORIAL untuk N = 4**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Langkah** | **FAKT** | **PENGALI** | **Kondisi** | **Output** |
| 1  2  3  3.a  3.b  3  3.a  3.b  3  3.a  3.b  3  3.a  3.b  3  4  5 | ?  1  1  4  4  4  12  12  12  24  24  24  24  24  24  24 | 4  4  4  4  3  3  3  2  2  2  1  1  1  0  0  0 | True  True  True  True  False | Faktorial dari 4 adalah 24 |

**Tabel 3.4 *Loop behavior table* dari Algoritma FAKTORIAL untuk N = 4**

|  |  |  |
| --- | --- | --- |
| **Pengulangan ke** | **Nilai FAKT** | **Nilai PENGALI** |
| 0  1  2  3  4 | 1  4  12  24  24 | 4  3  2  1  0 |

**3.2. Pengulangan dengan Kondisi**

Algoritma HALLO3 dan FAKTORIAL yang sudah dibahas di atas, adalah contoh dari **Pengulangan dengan Kondisi** (***Conditional loop*)**, yaitu pengulangan yang dikontrol oleh suatu kondisi. Instruksi yang digunakan dalam *conditional loop* ini adalah instruksi WHILE sehingga dapat disebut sebagai **While Loop**. Bentuk umum dari *conditional loop* ini adalah:

**WHILE ( kondisi )**

**{ ................ instruksi yang dikerjakan jika kondisi = = TRUE**

**}**

Artinya:

Instruksi-instruksi yang berada di antara tanda kurung {......} yang disebut sebagai *range of loop*, akan diulangi eksekusinya jika kondisi = = TRUE.

Ketika eksekusi algoritma sampai ke instruksi WHILE, kondisi dievaluasi lebih dulu sebelum menjalankan instruksi dalam *range of loop*. Jika hasil evaluasi kondisi = = TRUE maka semua instruksi yang ada dalam *range of loop* dieksekusi. Lalu kontrol program kembali ke instruksi WHILE untuk mengevaluasi kondisi lagi. Jika hasil evaluasi kondisi = = FALSE maka eksekusi algoritma dilanjutkan dengan instruksi setelah instruksi WHILE dalam hal ini *range of loop* tidak dieksekusi lagi. *Flowchart* dari *conditional loop* ini dapat dilihat pada gambar 3.1 di bawah ini.



Gambar 3.1 Flowchart dari bentuk umum *condional loop*

Sebagai ilustrasi masalah, akan dibuat algoritma untuk menghitung nilai akhir mahasiswa yang ditentukan oleh nilai praktikum, nilai UTS, dan nilai UAS. Bobot dari masing-masing nilai terhadap nilai akhir adalah 20%, 30% dan 50%. Sebagai output, akan dicetak nama mahasiswa, nilai praktikum, nilai UTS, nilai UAS dan nilai akhirnya. Algoritma untuk menghitung nilai akhir dari seorang mahasiswa dapat dilihat pada Algoritma LAPORAN berikut ini.

Algoritma LAPORAN

Algoritma untuk menghitung nilai akhir (AKHIR) yang berasal dari 20% nilai praktikum (PRAK), 30% nilai UTS (UTS), dan 50% nilai UAS (UAS). Sebagai input adalah nama mahasiswa (NAMA), nilai praktikum, nilai UTS dan nilai UAS. Variabel NAMA bertipe karakter, variabel PRAK, UTS dan UAS bertipe integer dan variabel AKHIR bertipe real.

1. [memasukkan data]

Read (NAMA, PRAK, UTS, UAS)

1. [hitung nilai akhir]

AKHIR = 0.2 \* PRAK + 0.3 \* UTS + 0.5 \* UAS

1. [menampilkan hasil]

Write(“Nama mahasiswa: “, NAMA)

Write(“Nilai praktikum : “, PRAK)

Write(“Nilai UTS: “, UTS)

Write(“Nilai UAS: “, UAS)

Write(“Nilai akhir: “, AKHIR)

1. [selesai]

Halt

Algoritma di atas hanya dapat digunakan untuk menghitung nilai akhir dari 1 orang mahasiswa. Untuk menghitung nilai akhir dari seluruh mahasiswa dalam kelas, maka algoritma ini harus dijalankan berulang kali. Untuk mengatasi hal ini perlu ditambahkan instruksi pengulangan (*loop*) dalam algoritma di atas sehingga proses input data dan perhitungan nilai (lihat langkah 1 dan 2 pada algoritma di atas) diulangi sesuai dengan banyaknya mahasiswa dalam kelas. Yang menjadi masalah adalah bagaimana menghentikan eksekusi suatu pengulangan. Untuk *conditional loop*, eksekusi pengulangan akan berhenti jika kondisi yang mengkontrolnya tidak dipenuhi lagi, atau (KONDISI = FALSE). Dalam *conditional loop*, ada 3 cara untuk mengontrol banyaknya pengulangan yaitu: *counter-controlled input*, *sentinel-controlled input*, dan sinyal *end of file* (eof).

**3.2.1 Counter-Controlled Input**

Dalam *counter-controlled input*, banyaknya data yang akan diproses ditentukan lebih dulu sehingga banyaknya pengulangan sesuai dengan banyaknya data yang akan diproses. Dalam hal ini, diperlukan suatu *counter* (pencacah) untuk mengontrol banyaknya pengulangan. Informasi mengenai banyaknya data (atau banyaknya pengulangan) ditaruh di awal kumpulan data yang akan diproses. Dengan demikian, dalam algoritma ini cara memasukkan inputnya adalah seperti ilustrasi pada gambar 3.2 di halaman berikut.

← Pencacah (banyaknya data)

← data baris pertama

•

•

•

← data baris terakhir

Gambar 3.2 Bentuk input untuk *counter-controlled input*

Data yang pertama dimasukkan adalah banyaknya data yang akan diproses, selanjutnya baru dimasukkan data baris pertama, data baris kedua dan seterusnya. Untuk menghitung nilai akhir dari seluruh mahasiswa, algoritma LAPORAN akan direvisi dengan menggunakan *counter-controlled input* ini. Untuk itu diperlukan dua variabel tambahan yaitu variabel BANYAK untuk menyimpan data banyaknya mahasiswa yang akan diproses dan variabel COUNT untuk menghitung banyaknya data mahasiswa yang **telah** diproses.

Algoritma LAPORAN\_revisi\_1

Algoritma untuk menghitung nilai akhir (AKHIR) yang berasal dari 20% nilai praktikum (PRAK), 30% nilai UTS (UTS), dan 50% nilai UAS (UAS) dari sejumlah mahasiswa dengan menggunakan counter-controlled input. Sebagai input adalah nama mahasiswa, nilai praktikum, nilai UTS, dan nilai UAS. Variabel NAMA bertipe karakter. Variabel PRAK, UTS, UAS, BANYAK dan COUNT bertipe integer. Variabel AKHIR bertipe real.

1. [memasukkan data banyaknya mahasiswa]

Read (BANYAK)

1. [inisialisasi counter]

COUNT = 0

1. [membuat counter controlled loop]

While ( COUNT < BANYAK )

{

3.a [memasukkan data]

Read (NAMA, PRAK, UTS, UAS)

3.b [hitung nilai akhir]

AKHIR = 0.2 \* PRAK + 0.3 \* UTS + 0.5 \* UAS

3.c [menampilkan hasil]

Write(“Nama mahasiswa: “, NAMA)

Write(“Nilai praktikum : “, PRAK)

Write(“Nilai UTS: “, UTS)

Write(“Nilai UAS: “, UAS)

Write(“Nilai akhir: “, AKHIR)

3.d [memperbaiki counter banyaknya mahasiswa yang sudah diproses]

COUNT = COUNT + 1

}

1. [selesai]

Halt

Nilai COUNT akan bertambah satu (lihat langkah 3.d) setiap kali pengulangan dieksekusi (dalam hal ini, artinya satu data mahasiswa telah selesai diproses). Jika suatu saat, nilai COUNT sama dengan nilai BANYAK maka berarti seluruh data telah selesai diproses. Kondisi inilah yang akan menghentikan eksekusi pengulangan dan eksekusi program dilanjutkan dengan langkah selanjutnya (lihat Algoritma LAPORAN\_revisi\_1 langkah yang ke 4). Ilustrasi flowchart untuk algoritma LAPORAN\_revisi\_1 ini dapat dilihat pada gambar 3.3 di bawah ini.



Gambar 3.3 Flowchart dari Algoritma LAPORAN\_revisi\_1

Kelebihan dari *counter-controlled input* ini adalah bentuknya sederhana, tetapi jika datanya sangat banyak, ada kemungkinan salah menghitung banyaknya data sehingga pengulangan dapat berhenti sebelum semua data diproses atau sebaliknya pengulangan belum berhenti meskipun semua data sudah diproses.

**3.2.2 Sentinel-Controlled Input**

Untuk mengatasi terjadinya kesalahan dalam menghitung banyaknya data yang akan diproses pada metode *counter-controlled input*, maka digunakan suatu informasi untuk menghentikan pengulangan yang diletakkan di akhir kumpulan data yang akan diproses. Informasi ini bentuknya seperti data, tetapi nilainya tidak sama dengan data sebenarnya dan disebut sebagai *sentinel* (tanda). Ketika *sentinel* ini dibaca, maka eksekusi pengulangan berhenti karena kondisi yang menghentikan pengulangan telah terpenuhi. Pengulangan yang menggunakan metode ini disebut *sentinel-controlled input*. Dengan menggunakan sentinel, banyaknya data atau pengulangan tidak diketahui, tetapi pengulangan akan berhenti (keluar dari loop) ketika program membaca sentinel. Dengan demikian, dalam algoritma ini cara memasukkan inputnya adalah seperti ilustrasi pada gambar 3.4 berikut ini

.

← data baris pertama

•

•

← data baris terakhir

← baris data *sentinel*

Gambar 3.4 Bentuk input untuk *sentinel*-*controlled* *loop*

Sebagai contoh, *sentinel-controlled input* akan digunakan untuk menyelesaikan masalah yang sama seperti di atas. Untuk itu algoritma LAPORAN akan direvisi lagi sehingga *conditional loop*-nya menggunakan metode *sentinel-controlled input*. Sebagai sentinel digunakan data “END” sebagai nama mahasiswa (lihat Algoritma LAPORAN\_ revisi\_2).

Algoritma LAPORAN\_revisi\_2

Algoritma untuk menghitung nilai akhir (AKHIR) yang berasal dari 20% nilai praktikum (PRAK), 30% nilai UTS (UTS), dan 50% nilai UAS (UAS) dari sejumlah mahasiswa dengan menggunakan sentinel-controlled loop. Sebagai input adalah nama mahasiswa, nilai praktikum, nilai UTS dan nilai UAS. Variabel NAMA bertipe karakter, variabel PRAK, UTS dan UAS bertipe integer dan variabel AKHIR bertipe real. Data input “END” untuk variabel NAMA adalah sentinel yang menunjukkan akhir dari deretan input data.

1. [membaca data baris pertama]

Read (NAMA, PRAK, UTS, UAS)

1. [membuat sentinel-controlled loop]

While ( NAMA != “END” )

{

2.a [menghitung nilai akhir]

AKHIR = 0.2 \* PRAK + 0.3 \* UTS + 0.5 \* UAS

2.b [menampilkan hasil]

Write(“Nama mahasiswa: “, NAMA)

Write(“Nilai praktikum : “, PRAK)

Write(“Nilai UTS: “, UTS)

Write(“Nilai UAS: “, UAS)

Write(“Nilai akhir: “, AKHIR)

2.c [membaca data baris berikutnya]

Read (NAMA, PRAK, UTS, UAS)

}

3. [selesai]

Halt

Kelebihan dari *sentinel-controlled input* dibandingkan dengan *counter-controlled input* adalah tidak perlu menghitung banyaknya data yang akan diproses. Tetapi harus hati-hati dalam menentukan nilai *sentinel*, karena jika nilai *sentinel* ini sama dengan nilai data maka pengulangan akan berhenti meskipun belum semua data diproses. Jadi dalam menentukan nilai *sentinel* harus yakin bahwa tidak ada data yang nilainya sama dengan nilai sentinel. Dalam algoritma LAPORAN\_revisi\_2, nilai sentinel yang digunakan adalah “END” untuk variabel NAMA. Hal ini digunakan karena dapat dipastikan tidak ada mahasiswa yang namanya “END”. Ilustrasi flowchart untuk algoritma LAPORAN\_revisi\_2 ini dapat dilihat pada gambar 3.5.

Untuk menggambarkan proses eksekusi algoritma di atas, digunakan data berikut ini sebagai input dan hasil penelusurannya dapat dilihat pada tabel 3.5.

“Mark Robin”, 58, 63, 72

“Edi Hasan”, 77, 51, 60

“END”, 0, 0, 0

Sesuai ilustrasi pada gambar 3.4 di atas, data baris terakhir adalah sentinel yang akan menghentikan proses pengulangan. Perhatikan perbedaan format data input dengan metode *counter-controlled input* yang telah dibahas sebelum ini.



Gambar 3.5 Flowchart dari Algoritma LAPORAN\_revisi\_2

Tabel 3.5 Tabel telusur untuk Algoritma LAPORAN\_revisi\_2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Langkah** | **NAMA** | **PRAK** | **UTS** | **UAS** | **AKHIR** | **Kondisi** | **Output** |
| 1  2  2.a  2.b  2.c  2  2.a  2.b  2.c  2  3 | Mark Robin  Mark Robin  Mark Robin  Mark Robin  Edi Hasan  Edi Hasan  Edi Hasan  Edi Hasan  END  END  Selesai | 58  58  58  58  77  77  77  77  0  0 | 63  63  63  63  51  51  51  51  0  0 | 72  72  72  72  60  60  60  60  0  0 | ?  ?  66.5  66.5  66.5  66.5  60.7  60.7  60.7  60.7 | True  True  False | Sesuai format  Sesuai format |

**3.2.3 Sinyal *End of File***

Jika data yang akan diproses telah disimpan dalam file, beberapa bahasa pemrograman menyediakan fasilitas sinyal *end of file* (eof) yang bernilai FALSE atau TRUE. Nilai FALSE dihasilkan jika masih ada data yang dibaca dan nilai TRUE dihasilkan jika tidak ada data yang dibaca lagi karena pembacaan data sudah sampai ke akhir file. Sinyal eof ini dapat digunakan sebagai input untuk mengontrol pengulangan sehingga dapat berfungsi seperti *sentinel* untuk menghentikan pengulangan. Bentuk umum penggunaan sinyal eof dalam *conditional loop* adalah :

**While ( not eof )**

**{ ............... }**

Algoritma LAPORAN\_revisi\_3 adalah contoh algoritma yang menggunakan sinyal eof. Dalam algoritma ini diasumsikan data sudah disimpan dalam file sehingga instruksi Read pada langkah ke 1 dan langkah ke 2.c, dimaksudkan untuk memasukkan data dari file. Sebenarnya ada perbedaan bentuk instruksi Read untuk membaca data dari *keyboard* dan untuk membaca data dari file, tetapi untuk sementara ini, kedua bentuk instruksi Read tersebut disamakan, tetapi pada bagian deklarasi diberikan keterangan bahwa data disimpan dalam file.

Algoritma LAPORAN\_revisi\_3

Algoritma untuk menghitung nilai akhir (AKHIR) yang berasal dari 20% nilai praktikum (PRAK), 30% nilai UTS (UTS), dan 50% nilai UAS (UAS) dari sejumlah mahasiswa dengan menggunakan sinyal end-of-file (eof). Sebagai input adalah nama mahasiswa (NAMA), nilai praktikum, nilai UTS dan nilai UAS yang disimpan dalam file. Variabel NAMA bertipe karakter, variabel PRAK, UTS dan UAS bertipe integer dan variabel AKHIR bertipe real.

* 1. [membaca data baris pertama]

Read (NAMA, PRAK, UTS, UAS)

* 1. [membuat pengulangan]

While ( not eof )

{

2.a. [menghitung nilai akhir]

AKHIR = 0.2 \* PRAK + 0.3 \* UTS + 0.5 \* UAS

2.b. [menampilkan hasil]

Write(“Nama mahasiswa: “, NAMA)

Write(“Nilai praktikum : “, PRAK)

Write(“Nilai UTS: “, UTS)

Write(“Nilai UAS: “, UAS)

Write(“Nilai akhir: “, AKHIR)

2.c. [membaca data baris berikutnya]

Read (NAMA, PRAK, UTS, UAS)

}

3. [selesai]

Halt

**Latihan Soal subbab 3.2**

1. Untuk tiap potongan algoritma berikut ini, tunjukkanlah apakah pengulangan akan berakhir atau tidak. Jika tidak, mengapa?

a. 1. COUNTER = 0

2. TOTAL = 0

3. While ( COUNTER >= 0 )

{ TOTAL = TOTAL + 2

}

b. 1. COUNTER = 0

2. TOTAL = 0

3. While ( COUNTER <= 10 )

{ TOTAL = TOTAL + 2

COUNTER = COUNTER + 1

}

2. Buatlah trace table untuk tiap potongan algoritma ini. Berapakah nilai yang dicetak untuk variabel VAR?. Diasumsikan semua variabel bertipe integer.

a. 1. VAR = 0

2. INDEKS = 1

3. WHILE (INDEKS <= 10)

3.a. { INDEKS = INDEKS +1

3.a. VAR = VAR + 1

}

4. Write (VAR)

b. 1. VAR = 0

2. INDEKS = 4

3. WHILE (INDEKS <= 36

3.a. { INDEKS = INDEKS + 4 )

3.b. VAR = VAR + 2

}

4. Write (VAR)

3. Buatlah potongan algoritma untuk masalah berikut ini:

a. Mencetak dan menjumlahkan 100 angka integer yang pertama, lalu hitung rata-ratanya.

b. Mencetak nilai variabel X dan kurangi X dengan 0.5 selama X positif

c. Membaca variabel A, B dan C lalu hitung jumlahnya. Ulangi proses ini selama tidak satupun dari nilai variabel A,B dan C yang negatif.

d. Menghitung dan mencetak kuadrat dari deretan integer sampai perbedaan antara kuadrat dari angka sebelumnya dan sesudahnya lebih besar dari 50.

3. Buatlah algoritma untuk memasukkan bilangan sebanyak n buah bilangan, lalu memeriksa apakah bilangan yang dimasukkan bilangan genap atau bilangan ganjil. Jika bilangan genap, hitung banyaknya bilangan dan hitung rata-ratanya.

4. Tentukan suatu jumlah yang cukup besar, misal B. Buat algoritma untuk menerima beberapa angka, dan menjumlahkan angka setiap kali angka dimasukkan sampai mencapai B lalu berhenti. Tampilkan banyaknya angka yang dimasukkan.

5. Bilangan prima adalah bilangan yang hanya habis dibagi dirinya sendiri dan angka 1. Buatlah algoritma untuk menentukan suatu bilangan apakah bilangan prima atau bukan.

6. Buatlah algoritma untuk menampilkan tabel konversi dari centimeter ke inci dan dari meter ke feet dan yard mulai dari nilai 0 sampai 100 dengan increment 5.

7. Buatlah algoritma untuk menerima jumlah penjualan setiap hari selama 1 minggu lalu hitung total penjualan dan rata-ratanya.

8. Sebuah ruang laboratorium harus dikontrol suhunya agar tidak terlalu panas atau dingin dengan range +- 2 derajat dari suhu yang ditentukan. Ruang tersebut dikontrol oleh pengatur suhu otomatis yang menerima input suhu ruangan setiap 5 menit. Buatlah algoritma yang menerima input suhu yang ditentukan dan suhu ruangan, lalu memberikan perintah ke pengatur suhu:

a. Menurunkan suhu pada AC sebesar 2 derajat jika suhu ruangan terlalu panas

b. Menaikkan suhu pada AC sebesar 2 derajat jika suhu ruangan terlalu dingin

**3.3. Pengulangan dengan Penghitung**

Dalam beberapa kasus, *conditional loop* tidak diperlukan karena banyaknya pengulangan sudah diketahui lebih dulu ketika algoritma dibuat. Dalam hal semacam ini digunakan **Pengulangan dengan penghitung (*counted loop*)**. Disebut *counted loop* karena banyaknya pengulangan dikontrol oleh suatu variabel pengontrol pengulangan (*loop control variable*) yaitu variabel yang menghitung berapa kali pengulangan sudah dieksekusi. Eksekusi pengulangan akan berhenti jika nilai variabel pengontrol pengulangan sudah sama dengan jumlah pengulangan yang diinginkan. Pengulangan ini menggunakan instruksi FOR sehingga dinamakan **For loop**. Bentuk umum dari *counted loop* adalah:

**For ( VP = INIT ; VP <= LIMIT ; ekspresi )**

**{ ........... range of loop ........**

**}**

Dengan:

- VP : variabel pengontrol pengulangan

- INIT : nilai awal VP

- LIMIT : nilai akhir variabel pengontrol pengulangan

- ekspresi : *increment* atau *decrement* dari VP

Sebagai contoh lihat macam-macam bentuk instruksi dengan *counted loop* berikut ini:

a. For ( J = 1 ; J <= 6 ; J = J + 1 ) atau For ( J = 1 ; J <= 6 ; J ++ )

⇒ instruksi dalam *range of loop* diulangi mulai dari J (variabel pengontrol pengulangan) bernilai 1, J bernilai 2, J bernilai 3, sampai J bernilai 6.

b. For ( COUNT = 0 ; COUNT <= JUM ; COUNT ++ )

⇒ instruksi dalam range of loop diulangi mulai dari COUNT bernilai 0, lalu 1, 2, ..., sampai COUNT = = JUM. Variabel JUM adalah variabel sudah mempunyai nilai sebelum instruksi FOR ini dieksekusi.

c. For ( INDEKS = 3 ; INDEKS <= 20 ; INDEKS = INDEKS + 3 )

atau For ( INDEKS = 3 ; INDEKS <= 20 ; INDEKS += 3 )

⇒ instruksi dalam range of loop diulangi selama INDEKS bernilai 3, 6, 9, ..., 18. Perhatikan, karena *increment* bernilai 3 maka nilai INDEKS ditambah 3 setiap kali masuk ke pengulangan berikutnya.

d. For ( K = 15 ; K >= 1 ; K = K - 2 ) atau For ( K = 15 ; K = 1 ; K -= 2 )

⇒ instruksi dalam range of loop akan diulangi selama K bernilai 15, 13, 11, ..., 1. Perhatikan, karena *decrement* bernilai (2) maka nilai K dikurangi 2 setiap kali masuk ke pengulangan berikutnya.

Dengan menggunakan *counted loop*, nilai awal dan nilai akhir dari variabel pengontrol pengulangan (VP) harus sudah diketahui. Secara otomatis VP akan diberi nilai awal INIT ketika pengulangan mulai dieksekusi, lalu ditambah dengan *increment* ketika eksekusi kembali ke instruksi FOR dan dibandingkan dengan nilai LIMIT. Jika nilai VP yang baru (setelah ditambah *increment*) ≤ LIMIT maka intruksi dalam rentang pengulangan dieksekusi lagi, begitu seterusnya sampai nilai VP > LIMIT. Sebagai pembanding dengan bentuk instruksi *conditional loop* yang telah dibahas, instruksi *counted loop* akan digunakan dalam algoritma LAPORAN\_revisi\_4.

Algoritma LAPORAN\_revisi\_4

Algoritma untuk menghitung nilai akhir (AKHIR) yang berasal dari 20% nilai praktikum (PRAK), 30% nilai UTS (UTS), dan 50% nilai UAS (UAS) dari sejumlah mahasiswa dengan menggunakan metode counted loop. Sebagai input adalah nama mahasiswa (NAMA), nilai praktikum, nilai UTS dan nilai UAS. Variabel NAMA bertipe karakter, variabel PRAK, UTS dan UAS bertipe integer dan variabel AKHIR bertipe real. Variabel JUM menunjukkan jumlah mahasiswa yang akan diproses nilai akhirnya, variabel COUNT menunjukkan jumlah mahasiswa yang telah diproses dan juga berfungsi sebagai variabel pengontrol pengulangan. Kedua variabel ini bertipe integer.

1. [memasukkan jumlah mahasiswa]

Read (JUM)

2. [membuat counted loop]

For ( COUNT = 1 ; COUNT <= JUM ; COUNT = COUNT + 1)

{

2.a [memasukkan data]

Read (NAMA, PRAK, UTS, UAS)

2.b [hitung nilai akhir]

AKHIR = 0.2 \* PRAK + 0.3 \* UTS + 0.5 \* UAS

2.c [menampilkan hasil]

Write(“Nama mahasiswa: “, NAMA)

Write(“Nilai praktikum : “, PRAK)

Write(“Nilai UTS: “, UTS)

Write(“Nilai UAS: “, UAS)

Write(“Nilai akhir: “, AKHIR)

}

3. [selesai]

Halt

Bila algoritma LAPORAN\_revisi\_4 dibandingkan dengan algoritma LAPORAN\_revisi\_1, maka terlihat, pada algoritma LAPORAN\_revisi\_4 tidak ada instruksi untuk menginisialisasi variabel pengontrol pengulangan, COUNT, maupun instruksi yang mengubah nilai COUNT karena dengan menggunakan *counted loop* ini, secara otomatis variabel COUNT diinisialisasi dengan nilai 1 dan ditambah dengan 1 ketika eksekusi pengulangan kembali ke instruksi FOR (lihat langkah ke 2). Dengan demikian penulisan algoritma menjadi lebih singkat. Untuk lebih jelasnya dapat dilihat pada gambar flowchart untuk algoritma LAPORAN\_revisi\_4 (gambar 3.6) di halaman berikut.

Untuk contoh berikutnya, akan ditulis kembali algoritma FAKTORIAL dengan menggunakan *counted loop*, hasilnya dapat dilihat pada algoritma FAKTORIAL\_revisi\_1 berikut ini.

Algoritma FAKTORIAL\_revisi\_1

Algoritma untuk menghitung faktorial dari suatu nilai yang diinputkan dengan menggunakan counted loop. Semua variabel bertipe integer.

1. [memasukkan angka yang akan dihitung faktorialnya]

Read(N)

1. [initialisasi variabel]

FAKT = 1

1. [melakukan perhitungan]

For ( PENGALI = N ; PENGALI >=1 ; PENGALI=PENGALI –1)

{

FAKT = FAKT \* PENGALI

}

1. [menampilkan hasil]

Write(“Faktorial dari “, N,” adalah “, FAKT)

1. [selesai]

Halt



Gambar 3.6 Flowchart dari Algoritma LAPORAN\_revisi\_4

Sebagai latihan, buatlah tabel telusur untuk algoritma LAPORAN (revisi 4) dengan data input yang sama seperti *counter-controlled input* dan untuk algoritma FAKTORIAL (revisi 1) untuk N = 4. Lalu bandingkan hasilnya dengan tabel 3.3. Kesimpulan apa yang anda dapatkan dari perbandingan kedua tabel tersebut?.

**Latihan soal subbab 3.3**

1. Buatlah trace table untuk tiap potongan algoritma ini. Berapakah nilai yang dicetak untuk variabel VAR?. Diasumsikan semua variabel bertipe integer.

a. 1. VAR = 0

2. For ( INDEKS = 1; INDEKS <= 10 ; INDEKS ++ )

2.a. { VAR = VAR + 1

}

3. Write (VAR)

b. 1. VAR = 0

2. For ( INDEKS = 4 ; INDEKS <= 36 ; INDEKS = INDEKS + 4 )

2.a. VAR = VAR + 2

}

3. Write (VAR)

2. Buatlah potongan algoritma dengan instruksi FOR untuk masalah berikut ini:

a. Mencetak akar kuadrat dari 25 angka integer yang pertama

b. Mencetak dan menjumlahkan 100 angka integer yang pertama lalu hitung rata-ratanya.

c. Mencetak titik-titik (X,Y) dari persamaan Y = X2 – 3x + 1 untuk nilai X mulai dari –2 sampai 2 dengan pertambahan (*increment*) 0.1.

3. Buatlah algoritma dengan menggunakan instruksi FOR untuk soal pada subbab 3.2 no. 3.

4. Buatlah algoritma dengan mneggunakan instruksi FOR untuk soal pada subbab 3.2 no. 6.

5. Buatlah algoritma dengan mneggunakan instruksi FOR untuk soal pada subbab 3.2 no. 7

6. Bagian R & D sebuah pabrik mobil melakukan percobaan mengenai efisiensi pemakaian bahan bakar. Hasil percobaan dapat dilihat pada tabel di bawah ini. Buat algoritma untuk membaca sepasang nilai data dalam tabel, lalu menghitung efisiensi pemakaian bahan bakar (km per liter bensin) untuk tiap pasang data dan menampilkan hasil perhitungannya.

Jarak tempuh (km) Jumlah pemakaian bensin (liter)

* 1. 14.8

248 15.1

302 12.8

147 9.25

265 13.3

7. Jika meminjam sebanyak M rupiah yang dikenai bunga sebesar B% per bulan dan ingin dilunasi dalam waktu N bulan, maka tiap bulan harus membayar cicilan sebanyak C rupiah dengan C adalah :

C = 

Selama waktu melunasi cicilan, sebagian pembayaran cicilan digunakan untuk membayar bunga bulanan dan sisanya untuk mengurangi saldo pinjaman.

Buatlah algoritma untuk membuat tabel amortisasi yang menampilkan nomor, jumlah pembayaran tiap bulan, bunga untuk bulan tersebut, jumlah pengurangan modal dan saldo yang baru. 

8. Buat algoritma untuk menghitung jumlah dari deret berikut ini sampai 100 suku yang pertama:

1 – ½ + ¼ - 1/6 + 1/8 – 1/10 + 1/12 - ....

**3.4. Pengulangan Bertingkat**

Sama seperti struktur If-else yang dapat berada dalam struktur If-else yang lain (struktur If bertingkat), struktur pengulangan juga dapat berada dalam struktur pengulangan yang lain atau disebut pengulangan bertingkat. Yang perlu diingat dalam membuat struktur pengulangan bertingkat ini adalah:

1. Struktur pengulangan yang ada di dalam harus sepenuhnya berada di dalam pengulangan yang di luar, tidak boleh ada *overlapping* di antara struktur pengulangan yang di dalam dengan struktur pengulangan yang berada di luar, lihat ilustrasi pada gambar 3.7.

2. Setiap pengulangan dalam struktur pengulangan bertingkat harus mempunyai variabel pengontrolnya sendiri-sendiri, tidak boleh ada struktur pengulangan yang mempunyai variabel pengontrol yang sama.



Gambar 3.7 Ilustrasi pengulangan bertingkat

Gambar 3.7.a dan 3.7.b adalah ilustrasi pengulangan bertingkat yang benar, karena semua pengulang yang didalam berakhir sebelum pengulangan yang berada di luar. Sementara itu, gambar 3.7.c adalah ilustrasi yang salah dari pengulangan bertingkat karena pengulangan yang berada di dalam berakhir setelah pengulangan yang berada di luar.

Sebagai contoh, lihat kembali masalah menghitung nilai akhir mahasiswa. Diasumsikan, nilai praktikum merupakan rata-rata dari 4 nilai praktikum sehingga akan digunakan pengulangan bertingkat. Pengulangan yang di luar digunakan untuk menginput data Nama, UTS dan UAS, lalu menghitung rata-rata praktikum dan nilai akhir. Pengulangan yang di dalam digunakan untuk menginput data praktikum dan menjumlahkannya. Dengan demikian algoritma LAPORAN­\_revisi\_4 direvisi lagi dengan menggunakan pengulangan bertingkat yang terdiri dari 2 buah struktur pengulangan. Hasilnya dapat dilihat pada Algoritma LAPORAN\_revisi\_5 di bawah ini. Struktur pengulangan yang pertama (luar) ada pada langkah 2.a sampai langkah 2.f. Sedang struktur pengulangan yang kedua (dalam) ada pada langkah 2.c.i dan 2.c.ii. Perhatikan nomor langkah dari masing-masing struktur pengulangan.

Algoritma LAPORAN\_revisi\_5

Algoritma untuk menghitung nilai akhir (AKHIR) yang berasal dari 20% nilai praktikum (PRAK), 30% nilai UTS (UTS), dan 50% nilai UAS (UAS) dari sejumlah mahasiswa dengan menggunakan metode counted loop. Nilai praktikum merupakan rata-rata dari 4 nilai praktikum yang dihitung dengan menggunakan nested loop. Sebagai input adalah nama mahasiswa (NAMA), 4 buah nilai praktikum, nilai UTS dan nilai UAS. Variabel NAMA bertipe karakter. Variabel P, UTS dan UAS bertipe integer. Variabel AKHIR dan PRAK bertipe real. Variabel J menunjukkan banyaknya mahasiswa yang akan diproses nilai akhirnya, variabel C menunjukkan banyaknya mahasiswa yang telah diproses dan juga berfungsi sebagai variabel pengontrol pengulangan. Variabel PR merupakan variabel untuk menunjukkan banyaknya praktikum dan juga berfungsi sebagai pengontrol loop yang di dalam. Ketiga variabel ini bertipe integer.

1. [memasukkan banyaknya mahasiswa]

Read (J)

1. [membuat counted loop]

For ( C = 1 ; C <= J ; C = C + 1)

{

2.a [memasukkan data]

Read (NAMA, UTS, UAS)

2.b [inisialisasi variabel PRAK]

PRAK = 0

2.c. [memasukkan nilai praktikum satu persatu dan menjumlahkannya]

For ( PR = 1 ; PR <= 4 ; PR++ )

{

2.c.i Read (P)

2.c.ii PRAK = PRAK + P

}

2.d. [menghitung rata-rata praktikum]

PRAK = PRAK/4

2.e. [hitung nilai akhir]

AKHIR = 0.2 \* PRAK + 0.3 \* UTS + 0.5 \* UAS

2.f. [menampilkan hasil]

Write(“Nama mahasiswa: “, NAMA)

Write(“Nilai rata-rata praktikum : “, PRAK)

Write(“Nilai UTS: “, UTS)

Write(“Nilai UAS: “, UAS)

Write(“Nilai akhir: “, AKHIR)

}

1. [selesai]

Halt

Perhatikan langkah 2.c.i dan 2.c.ii yang merupakan badan loop dari loop yang di dalam untuk menginput dan menjumlahkan total nilai praktikum dari seorang mahasiswa. Loop ini dieksekusi sampai selesai sebelum eksekusi dilanjutkan ke langkah berikutnya (langkah 2.d). Penelusuran algoritma LAPORAN\_revisi\_5 dilakukan untuk contoh data mahasiswa berikut ini, dan hasilnya dapat dilihat pada trace table di bawah ini (tabel 3.6).

2

“Mark Robin”, 63, 72, 58, 70, 73, 63

“Edi Hasan”, 51, 60, 70, 75, 80, 83

Tabel 3.6 Trace Table untuk Algoritma LAPORAN\_revisi\_5

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Langkah** | **J** | **C** | **NAMA** | **UTS** | **UAS** | **PRAK** | **PR** | **P** | **AKHIR** | **K 1** | **K 2** | **Output** |
| 1 | 2 | ? | ? | ? | ? | ? | ? | ? | ? |  |  |  |
| 2 | 2 | 1 | ? | ? | ? | ? | ? | ? | ? | T |  |  |
| 2.a | 2 | 1 | Mark Robin | 63 | 73 | ? | ? | ? | ? |  |  |  |
| 2.b | 2 | 1 | Mark Robin | 63 | 73 | 0 | ? | ? | ? |  |  |  |
| 2.c | 2 | 1 | Mark Robin | 63 | 73 | 0 | 1 | ? | ? |  | T |  |
| 2.c.i | 2 | 1 | Mark Robin | 63 | 73 | 0 | 1 | 58 | ? |  |  |  |
| 2.c.ii | 2 | 1 | Mark Robin | 63 | 73 | 58 | 1 | 58 | ? |  |  |  |
| 2.c | 2 | 1 | Mark Robin | 63 | 73 | 58 | 2 | 58 | ? |  | T |  |
| 2.c.i | 2 | 1 | Mark Robin | 63 | 73 | 58 | 2 | 70 | ? |  |  |  |
| 2.c.ii | 2 | 1 | Mark Robin | 63 | 73 | 128 | 2 | 70 | ? |  |  |  |
| 2.c | 2 | 1 | Mark Robin | 63 | 73 | 128 | 3 | 70 | ? |  | T |  |
| 2.c.i | 2 | 1 | Mark Robin | 63 | 73 | 128 | 3 | 73 | ? |  |  |  |
| 2.c.ii | 2 | 1 | Mark Robin | 63 | 73 | 128 | 3 | 73 | ? |  |  |  |
| 2.c | 2 | 1 | Mark Robin | 63 | 73 | 201 | 4 | 73 | ? |  | T |  |
| 2.c.i | 2 | 1 | Mark Robin | 63 | 73 | 201 | 4 | 63 | ? |  |  |  |
| 2.c.ii | 2 | 1 | Mark Robin | 63 | 73 | 264 | 4 | 63 | ? |  |  |  |
| 2.c | 2 | 1 | Mark Robin | 63 | 73 | 264 | 5 | 63 | ? |  | F |  |
| 2.d | 2 | 1 | Mark Robin | 63 | 73 | 66 | 5 | 63 | ? |  |  |  |
| 2.e | 2 | 1 | Mark Robin | 63 | 73 | 66 | 5 | 63 | 68.6 |  |  |  |
| 2.f | 2 | 1 | Mark Robin | 63 | 73 | 66 | 5 | 63 | 68.6 |  |  | Sesuai format |
| 2 | 2 | 2 | Mark Robin | 63 | 73 | 66 | 5 | 63 | 68.6 | T |  |  |
| 2.a | 2 | 2 | Edi Hasan | 51 | 60 | 66 | 5 | 63 | 68.6 |  |  |  |
| 2.b | 2 | 2 | Edi Hasan | 51 | 60 | 0 | 5 | 63 | 68.6 |  |  |  |
| 2.c | 2 | 2 | Edi Hasan | 51 | 60 | 0 | 1 | 63 | 68.6 |  | T |  |
| 2.c.i | 2 | 2 | Edi Hasan | 51 | 60 | 0 | 1 | 70 | 68.6 |  |  |  |
| 2.c.ii | 2 | 2 | Edi Hasan | 51 | 60 | 70 | 1 | 70 | 68.6 |  |  |  |
| 2.c | 2 | 2 | Edi Hasan | 51 | 60 | 70 | 2 | 70 | 68.6 |  | T |  |
| 2.c.i | 2 | 2 | Edi Hasan | 51 | 60 | 70 | 2 | 75 | 68.6 |  |  |  |
| 2.c.ii | 2 | 2 | Edi Hasan | 51 | 60 | 145 | 2 | 75 | 68.6 |  |  |  |
| 2.c | 2 | 2 | Edi Hasan | 51 | 60 | 145 | 3 | 75 | 68.6 |  | T |  |
| 2.c.i | 2 | 2 | Edi Hasan | 51 | 60 | 145 | 3 | 80 | 68.6 |  |  |  |
| 2.c.ii | 2 | 2 | Edi Hasan | 51 | 60 | 225 | 3 | 80 | 68.6 |  |  |  |
| 2.c | 2 | 2 | Edi Hasan | 51 | 60 | 225 | 4 | 80 | 68.6 |  | T |  |
| 2.c.i | 2 | 2 | Edi Hasan | 51 | 60 | 225 | 4 | 83 | 68.6 |  |  |  |
| 2.c.ii | 2 | 2 | Edi Hasan | 51 | 60 | 308 | 4 | 83 | 68.6 |  |  |  |
| 2.c | 2 | 2 | Edi Hasan | 51 | 60 | 308 | 5 | 83 | 68.6 |  | F |  |
| 2.d | 2 | 2 | Edi Hasan | 51 | 60 | 77 | 5 | 83 | 68.6 |  |  |  |
| 2.e | 2 | 2 | Edi Hasan | 51 | 60 | 77 | 5 | 83 | 60.9 |  |  |  |
| 2.f | 2 | 2 | Edi Hasan | 51 | 60 | 77 | 5 | 83 | 60.9 |  |  | Sesuai format |
| 2 | 2 | 3 | Edi Hasan | 51 | 60 | 77 | 5 | 83 | 60.9 | F |  |  |
| 3 | Selesai | |  |  |  |  |  |  |  |  |  |  |

Keterangan:

K1: Kondisi dari loop yang di luar

K2: kondisi dari loop yang di dalam

T: TRUE

F: FALSE

Sebagai contoh lain penggunaan pengulangan bertingkat dapat dilihat pada Algoritma CONTOH di bawah ini. Pada algoritma ini, ada 3 buah struktur pengulangan, yang di luar adalah pengulangan yang dikontrol oleh variabel I, pengulangan yang di tengah dikontrol oleh variabel J dan pengulangan yang di dalam dikontrol oleh variabel K. Dalam eksekusi pengulangan bertingkat, pengulangan yang berada di dalam adalah pengulangan yang paling sering dieksekusi karena pengulangan yang berada di dalam harus dieksekusi secara penuh sebelum variabel pengontrol pengulangan yang di luar dapat bertambah.

Algoritma CONTOH

Algoritma untuk menghitung perkembangan nilai variabel X, Y, dan Z yang berada dalam pengulangan bertingkat. Variabel I, J an K adalah variabel untuk mengontrol pengulangan. Semua variabel bertipe integer.

1. [inisialisasi variabel]

X = 0

Y = 0

Z =0

2. [membuat pengulangan yang pertama]

For ( I = 1 ; I <= 3 ; I = I + 1 )

{

2.a X = X + 1

[membuat pengulangan yang kedua]

2.b For ( J = 1 ; J <= 3 ; J = J + 1 )

{

2.b.i Y = Y + 1 [membuat pengulangan yang ketiga]

2.b.ii For ( K = 1 ; K <= 3 ; K = K + 1 ) {

2.b.ii.1 Z = Z + 1 }

2.b.iii Write (“Z = “, Z) }

2.c Write (“Y = “, Y, “Z = “, Z)

}

3. Write (“X = “, X, “Y = “, Y, “Z = “, Z)

4. [selesai]

Halt

Hasil penelusuran algoritma CONTOH dapat dilihat tabel 3.7 di bawah ini.

Tabel 3.7 Tabel telusur untuk algoritma CONTOH

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Langkah | X | Y | Z | I | J | K | Output |
| 1 | 0 | 0 | 0 | ? | ? | ? |  |
| 2 | 0 | 0 | 0 | 1 | ? | ? |  |
| 2.a | 1 | 0 | 0 | 1 | ? | ? |  |
| 2.b | 1 | 0 | 0 | 1 | 1 | ? |  |
| 2.b.i | 1 | 1 | 0 | 1 | 1 | ? |  |
| 2.b.ii | 1 | 1 | 0 | 1 | 1 | 1 |  |
| 2.b.ii.1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 2.b.ii | 1 | 1 | 1 | 1 | 1 | 2 |  |
| 2.b.ii.1 | 1 | 1 | 2 | 1 | 1 | 2 |  |
| 2.b.ii | 1 | 1 | 2 | 1 | 1 | 3 |  |
| 2.b.ii.1 | 1 | 1 | 3 | 1 | 1 | 3 |  |
| 2.b.ii | 1 | 1 | 3 | 1 | 1 | 4 |  |
| 2.b.iii | 1 | 1 | 3 | 1 | 1 | 4 | Z = 3 |
| 2.b | 1 | 1 | 3 | 1 | 2 | 4 |  |
| 2.b.i | 1 | 2 | 3 | 1 | 2 | 1 |  |
| 2.b.ii | 1 | 2 | 3 | 1 | 2 | 1 |  |
| 2.b.ii.1 | 1 | 2 | 4 | 1 | 2 | 1 |  |
| ... | ... | ... | ... | ... | ... | ... |  |
| 2.b.iii | 1 | 2 | 6 | 1 | 2 | 4 | Z = 6 |
| 2.b | 1 | 3 | 6 | 1 | 4 | 4 |  |
| 2.b.i | 1 | 3 | 6 | 1 | 4 | 4 |  |
| ... | ... | ... | ... | ... | ... | ... |  |
| 2.b.iii | 1 | 3 | 9 | 1 | 3 | 4 | Z = 9 |
| 2.b | 1 | 3 | 9 | 1 | 4 | 4 |  |
| 2.c | 1 | 3 | 9 | 1 | 4 | 4 | Y = 3 Z = 9 |
| 2 | 1 | 3 | 9 | 2 | 4 | 4 |  |
| 2.a | 2 | 3 | 9 | 2 | 4 | 4 |  |
| 2.b | 2 | 3 | 9 | 2 | 1 | 4 |  |
| 2.b.i | 2 | 4 | 9 | 2 | 1 | 4 |  |
| ... | ... | ... | ... | ... | ... | ... |  |
| 2.b.iii | 2 | 6 | 18 | 2 | 3 | 4 | Z = 18 |
| 2.b | 2 | 6 | 18 | 2 | 4 | 4 |  |
| 2.c | 2 | 6 | 18 | 2 | 4 | 4 | Y = 6 Z = 18 |
| 2 | 2 | 6 | 18 | 3 | 4 | 4 |  |
| 2.a | 3 | 6 | 18 | 3 | 4 | 4 |  |
| 2.b | 3 | 6 | 18 | 3 | 1 | 4 |  |
| 2.b.i | 3 | 7 | 18 | 3 | 1 | 4 |  |
| ... | ... | ... | ... | ... | ... | ... |  |
| 2.b.iii | 3 | 9 | 27 | 3 | 3 | 4 | Z = 27 |
| 2.b | 3 | 9 | 27 | 3 | 4 | 4 |  |
| 2.c | 3 | 9 | 27 | 3 | 4 | 4 | Y = 9 Z = 27 |
| 2 | 3 | 9 | 27 | 4 | 4 | 4 |  |
| 3 | 3 | 9 | 27 | 4 | 4 | 4 | X = 3 Y = 9 Z = 27 |
| 4 | Selesai | |  |  |  |  |  |

Untuk mempersingkat tabel, tanda **....** menunjukkan ada baris yang tidak ditampilkan. Perhatikan, variabel Z yang bertambah dengan cepat dibanding variabel Y dan X. Dengan demikian, loop yang berada paling dalam adalah loop yang paling banyak dieksekusi.

**3.5. Instruksi Break**

Dalam beberapa kasus, perlu keluar dari pengulangan sebelum seluruh pengulangan diselesaikan. Hal ini dapat dilakukan dengan menggunakan instruksi **Break.** Jika kontrol program menemui instruksi ini maka eksekusi program akan dilanjutkan pada instruksi setelah loop tempat instruksi Break berada meskipun eksekusi pengulangan belum selesai. Biasanya instruksi Break ini digunakan bersama-sama dengan instruksi If untuk memeriksa kondisi: jika kondisi dipenuhi maka Break. Bentuk umum dari instruksi pengulangan dengan Break ini adalah:

**While ( j <= N )**

**{ ..............**

**..............**

**If ( kondisi )**

**{ Break**

**}**

**..............**

**}**

**.............. ← instruksi yang dikerjakan setelah Break**

Instruksi Break dapat juga digunakan dalam pengulangan bertingkat, untuk keluar dari pengulangan yang di dalam sebelum waktunya lalu diteruskan dengan eksekusi pengulangan yang di luar. Sebagai contoh, ingin dihitung nilai rata-rata dari tugas yang dikumpulkan mahasiswa. Yang menjadi masalah adalah banyaknya tugas yang dikumpulkan mahasiswa tidak sama, ada mahasiswa yang mengumpulkan 4 tugas, ada yang 2 tugas dan seterusnya. Dengan demikian, ketika menghitung nilai rata-rata dari tiap mahasiswa, maka perlu dihitung juga banyaknya tugas yang dikumpulkan. Data yang dimasukkan adalah NPM mahasiswa dan nilai tugas. Diasumsikan deretan data diakhiri dengan sentinel NPM yang bernilai 999. Data dikelompokkan per mahasiswa seperti contoh berikut ini:

NPM Nilai tugas

5010123 80

5010123 75

5010123 85

5010211 80

5010211 75

5010135 85

5010135 80

5010135 75

5010135 85

999 0

Algoritma untuk menghitung nilai rata-rata seperti di atas dapat dilihat pada algoritma Hitung\_rata\_rata di halaman berikut. Dalam algoritma ini, jika nomor mahasiswa yang dibaca berikutnya (NPM2 - lihat langkah ke 2.b.iii) masih sama dengan yang sebelumnya (NPM1), maka instruksi IF pada langkah ke 2.b.iv menghasilkan nilai FALSE sehingga eksekusi kembali ke langkah 2.b (pengulangan yang di dalam). Jika nomor mahasiswa yang dibaca berikutnya (NPM2) berbeda dengan yang sebelumnya (NPM1), maka instruksi IF pada langkah ke 2.b.iv menghasilkan nilai TRUE sehingga data nilai untuk mahasiswa yang sebelumnya (NPM1) sudah habis dibaca. Dengan demikian dapat dihitung nilai rata-rata dari seorang mahasiswa. Sebagai latihan, cobalah membuat tabel telusur untuk algoritma ini dengan menggunakan data contoh di atas.

Instruksi While pada langkah 2.b digunakan untuk membuat pengulangan pembacaan data nilai mahasiswa. Instruksi ini akan menghasilkan *infinite loop* jika tidak ada instruksi Break di langkah ke 2.b.iv.

Algoritma Hitung\_rata\_rata

Algoritma untuk menghitung nilai rata-rata dari tiap mahasiswa (RATA). NPM1 dan NPM2 adalah variabel integer untuk menyimpan nomor mahasiswa. TOTAL adalah variabel integer yang menyimpan jumlah nilai. JUM adalah variabel integer yang menyimpan jumlah tugas untuk tiap mahasiswa. NILAI adalah variabel sementara, bertipe integer.

1. [membaca nomor mahasiswa yang pertama]

Read (NPM1)

1. [membuat pengulangan untuk memroses semua mahasiswa]

While ( NPM1 != 999 )

{

2.a. [inisialisasi variabel]

TOTAL = 0

JUM = 0

2.b. [memroses semua nilai untuk seorang mahasiswa]

While (JUM >= 0)

{

2.b.i. [membaca nilai]

Read (NILAI)

2.b.ii. [meng-update total]

TOTAL = TOTAL + NILAI

JUM = JUM + 1

2.b.iii. [membaca data mahasiswa yang berikutnya]

Read (NPM2)

2.b.iv. [apakah nomor mahasiswa lain?]

If ( NPM2 != NPM1 )

{ [selesai memproses seorang mahasiswa]

RATA = TOTAL / JUM

Write (NPM1, RATA)

NPM1 = NPM2 [siap masuk ke pengulangan luar]

Break

}

}

}

1. [selesai proses]

Write(“Semua mahasiswa telah diproses”)

Halt

**3.6. Mengontrol pengulangan**

Ada dua cara untuk mengontrol pengulangan yaitu:

1. *Top* *tested* atau *pretest*: variabel pengontrol pengulangan dicek lebih dulu sebelum memasuki rentang pengulangan. Jika variabel tersebut memenuhi syarat maka rentang pengulangan dieksekusi, jika tidak maka rentang pengulangan tidak dieksekusi. Semua struktur pengulangan yang telah dibahas termasuk dalam cara ini.
2. *Bottom* *tested* atau *posttest*: rentang pengulangan dieksekusi lebih dulu setelah itu baru dicek apakah variabel pengontrol pengulangan memenuhi syarat atau tidak. Jika masih memenuhi syarat maka badan pengulangan dieksekusi lagi, jika tidak eksekusi dilanjutkan dengan instruksi berikutnya.

Untuk jelasnya, dapat dilihat pada gambar flowchart dari kedua jenis pengontrolan di atas (lihat gambar 3.8) di halaman berikut ini.

Gambar 3.8 *Flowchart* struktur pengulangan untuk:

(a) *Bottom tested* (b) *Top tested*

Bentuk umum instruksi pengulangan yang menggunakan cara *bottom tested* adalah:

**Do**

**{ instruksi\_1**

**instruksi\_2**

**.....**

**instruksi\_n**

**}**

**While ( kondisi )**

Pengulangan dengan cara bottom tested menyebabkan seluruh instruksi dalam rentang pengulangan dieksekusi dulu tanpa memperhatikan apakah kondisi memenuhi syarat atau tidak,

Sebagai contoh, cara ini akan digunakan bersama-sama dengan *sentinel-controlled input* untuk menghitung nilai akhir mahasiswa (lihat algoritma LAPORAN\_revisi\_2) sehingga hasilnya dapat dilihat pada algoritma LAPORAN \_revisi\_ 6 berikut.

Algoritma LAPORAN\_revisi\_6

Algoritma untuk menghitung nilai akhir (AKHIR) yang berasal dari 20% nilai praktikum (PRAK), 30% nilai UTS (UTS), dan 50% nilai UAS (UAS) dari sejumlah mahasiswa dengan menggunakan bottom tested dan sentinel-controlled loop. Sebagai input adalah nama mahasiswa, nilai praktikum, nilai UTS dan nilai UAS. Variabel NAMA bertipe karakter, variabel PRAK, UTS dan UAS bertipe integer dan variabel AKHIR bertipe real. Data input “END” untuk variabel NAMA adalah sentinel yang menunjukkan akhir dari deretan input data.

1. [membaca data baris pertama]

Read (NAMA, PRAK, UTS, UAS)

1. [membuat pengulangan]

Do

{

2.a. [menghitung nilai akhir]

AKHIR = 0.2 \* PRAK + 0.3 \* UTS + 0.5 \* UAS

2.b. [menampilkan hasil]

Write(“Nama mahasiswa: “, NAMA)

Write(“Nilai praktikum : “, PRAK)

Write(“Nilai UTS: “, UTS)

Write(“Nilai UAS: “, UAS)

Write(“Nilai akhir: “, AKHIR)

2.c. [membaca data baris berikutnya]

Read (NAMA, PRAK, UTS, UAS)

}

[menguji kondisi sentinel]

While NAMA != “END”

1. [selesai]

Halt

Pada algoritma LAPORAN\_revisi\_6, pengujian sentinel dilakukan di akhir *range of loop* sehingga kontrol program tidak perlu kembali ke instruksi Do yang berada di awal pengulangan jika kondisi sudah tidak memenuhi lagi. Dengan demikian pembuatan tabel telusurnya menjadi lebih singkat. Sebagai latihan buatlah tabel telusur untuk algoritma LAPORAN\_revisi\_6 dengan input yang sama lalu bandingkan hasilnya.

Perhatikan, untuk algoritma di atas, pada langkah ke 2, langsung mengeksekusi instruksi-instruksi pada rentang pengulangan, lalu setelah selesai baru diperiksa apakah kondisinya memenuhi syarat (lihat statement While NAMA != “END”). Jika memenuhi syarat (kondisi == TRUE) maka eksekusi kembali ke langkah 2. Jika tidak maka selesai.

**LATIHAN SOAL BAB 3**

1. Buatlah trace table untuk tiap potongan algoritma ini. Berapakah nilai yang dicetak untuk variabel VAR?. Diasumsikan semua variabel bertipe integer.

a. 1. VAR = 0

2. For ( INDEKS1 = 1 ; INDEKS1 <= 5 ; INDEKS1 ++ )

2.a. { For (INDEKS2 = 3 ; INDEKS2 <= 6 ; INDEKS2 ++ )

2.a.i { VAR = VAR + 1

}

2.b. Write (VAR)

}

b. 1. VAR1 = 0; VAR2 = 0

2. For ( INDEKS1 = 10 ; INDEKS1 <= 1 ; INDEKS1 -= 1)

2.a. { VAR1 = VAR1 + 1

2.b. For (INDEKS2 = 1 ; INDEKS2 <= 6 ; INDEKS2 += 2 )

2.b.i. { VAR2 = VAR2 + 1

}

2.c. Write (VAR1, VAR2)

}

2. Buatlah trace table untuk algoritma berikut ini:

Algoritma SOAL\_4

Variabel A, B, X, Y, dan Z adalah variabel integer

1. X = 0
2. Y = 5
3. Z = 25
4. While ( X ≤ 4 )

4.a { Y = Z – Y

4.b A = X + 1

4.c X = X + 1

4.d If ( A > 1 )

{ Z = Z – 5

A = A^2

B = Z – Y

}

}

1. Write (A, B, X, Y, Z)
2. Halt

3. Diketahui algoritma berikut ini:

Algoritma SOAL\_5

Variabel A adalah variabel real, variabel X, S dan T adalah variabel integer

1. Read (A)
2. X = 0
3. T = 0
4. While ( A ≤ 0.5 )

4.a. { A = A + 0.1

4.b. If ( A != 0.3 )

4.c. { T = T + 1

4.d. While ( T ≤ 5 )

4.d.i. { X = X + T

4.d.ii T = T + 2

4.d.iii S = X + T

}

}

Else

4.e. { T = T + 1

4.f. While ( X ≤ 5 )

4.f.i. { T = T \* 2

4.f.ii X = T – 2

4.f.iii S = X \* 3

}

}

1. Write (A, S, T, X)
2. Halt

Buatlah trace table untuk algoritma di atas jika nilai yang dimasukkan untuk A adalah:

a. 0.2

b. 0.5

c. 1.0

4. Untuk menjaga populasi ikan, dibuat aturan yang membatasi berat ikan (dalam kg) yang boleh dipancing dalam satu hari. Jadi setiap pemancing harus dilengkapi dengan timbangan sehingga dapat mengetahui berat ikan yang ditangkapnya dan segera berhenti ketika berat ikan yang ditangkap sudah melebihi batas. Buatlah algoritma untuk membaca batas berat ikan, lalu membaca berat ikan yang ditangkap satu persatu dan mencetak pesan ketika berat ikan sudah melebihi batas yang ditentukan. Berat = 0 menunjukkan akhir dari input. Algoritma yang anda buat juga harus dapat mencetak total berat ikan yang dipancing setiap kali input dimasukkan.

5. Perusahaan Birdie membuat dan menjual kandang burung. Perusahaan mendapatkan keuntungan sebesar Rp. 15750,- untuk setiap kandang burung yang dijualnya, dan sekarang dapat menjual 3500 kandang burung per bulan. Biaya tetap operasi perusahaan adalah Rp. 18500000,- per bulan dan tambanan sebesar Rp.1000000 perbulan untuk biaya iklan. Hasil penelitian menunjukkan bahwa jika melipatduakan biaya iklan maka penjualan akan naik 15%. Buatlah algoritma yang akan menunjukkan bahwa pada suatu saat penambahan biaya iklan menjadi tidak menguntungkan (untuk tiap tingkat, dua kalikan biaya iklan dan kalikan keuntungan sebelumnya dengan 15%).

6. Rekomendasi untuk mendapatkan beasiswa untuk sekolah lanjut diberikan berdasarkan nilai rata-rata siswa sesuai dengan tabel berikut ini:

Nilai rata-rata Rekomendasi .

≥ 90 Sangat direkomendasikan

≥ 80 tetapi < 90 Direkomendasikan

≥ 70 tetapi <80 Kurang direkomendasikan

< 70 Tidak direkomendasikan

Data yang disiapkan untuk tiap siswa yang mengajukan aplikasi beasiswa sesuai dengan format berikut ini:

Nama siswa, nilai rata-rata

Buatlah algoritma untuk membaca data dari semua siswa dan menghasilkan daftar yang berisi nama siswa, nilai rata-rata dan jenis rekomendasi. Input data diakhiri oleh nama siswa = ‘End’. Hitung pula nilai rata-rata dari seluruh siswa yang mengajukan aplikasi beasiswa dan jumlah untuk tiap jenis rekomendasi.

7. Pemilik tim basket ANB yang mempunyai 20 orang pemain, ingin memberikan kenaikan gaji bagi para pemainnya sesuai dengan tabel berikut ini:

Gaji sekarang Persentasi kenaikan

Rp. 0 - Rp. 10.000.000 20 %

Rp. 10.000.001 – Rp. 15.000.000 10 %

Rp. 15.000.001 – Rp. 20.000.000 5 %

Lebih dari Rp. 20.000.000 0 %

Buatlah algoritma untuk membaca nama pemain dan gaji sekarang, lalu menghitung dan mencetak daftar nama pemain, gaji sekarang dan gaji baru. Diakhir daftar hitung pula total gaji sekarang dan total gaji baru.

8. Siswa mengikuti 5 macam ujian akhir yaitu MK1, MK2, MK3, MK4 dan MK5. Siswa dinyatakan lulus jika nilai ujiannya ≥ 60. Wali kelas ingin mengetahui jumlah siswa yang:

1. lulus semua ujian
2. lulus ujian MK1, MK2 dan MK3, tetapi bukan MK4 atau MK5

c. lulus ujian MK1 dan MK2, MK3 atau MK4, tetapi bukan MK5

Buatlah algoritma untuk membaca semua nilai ujian tiap siswa dan menghasilkan banyaknya siswa untuk tiap katagori. Gunakan metode end-of-file untuk mengakhiri input data.

9. Buatlah algoritma untuk pemainan menebak angka integer antara 1 sampai 100. Pemain diberi kesempatan sebanyak 10 kali untuk menebak angka yang disimpan oleh komputer. Jika tebakan pemain lebih besar dari angka yang disimpan maka komputer akan mengeluarkan pesan: “Tebakan anda terlalu besar. Coba lagi”. Sebaliknya jika tebakan pemain lebih kecil maka komputer akan mengeluarkan pesan: “Tebakan anda terlalu kecil. Coba lagi”. Jika pemain menebak dengan tepat, komputer akan menampilkan pesan: “Selamat, tebakan anda benar”. Gunakan fungsi random dengan range antara 1 – 100 untuk menyimpan angka yang akan ditebak.

10. Modifikasi algoritma di atas dengan mengubah banyaknya tebakan menjadi 5 kali dan pesan yang ditampilkan komputer menjadi:

* 1. “Tebakan anda kebesaran. Coba lagi” jika selisih antara tebakan pemain dengan angka yang disimpan komputer lebih besar atau sama dengan 10
  2. “Tebakan anda agak kebesaran. Coba lagi” jika selisih antara 11 - 20
  3. “Tebakan anda terlalu besar. Coba lagi” jika selisih lebih besar dari 20
  4. “Tebakan anda kekecilan. Coba lagi” jika selisih antara tebakan pemain dengan angka yang disimpan komputer lebih besar atau sama dengan -10
  5. “Tebakan anda agak kekecilan. Coba lagi” jika selisih antara -11 sampai -20
  6. “Tebakan anda terlalu kecil. Coba lagi” jika selisih lebih kecil dari -20

11. Buatlah algoritma untuk berlatih matematika seperti berikut:

247

123 +

-----------------

Program menampilkan 2 buah angka dan sebuah operator. Lalu program akan menunggu user untuk memasukkan angka. Jika angka yang dimasukkan benar maka program akan menampilkan soal di atas beserta jawabannya. Jika angka yang dimasukkan salah, maka program akan menampilkan pesan “Anda salah, silahkan mencoba lagi” lalu program menampilkan soal seperti di atas.

12. Buatlah algoritma untuk menghitung dan menabelkan hasil dari persamaan berikut ini:

untuk x = 2, 4, 6, 8 dan y = 6, 9, 12, 15, 18, 21

13. Sebuah toko ritel memberikan promosi seperti tabel di bawah ini. Buatlah algoritma untuk mencetak slip pembayaran yang berisi nama barang, harga satuan, banyaknya barang, total harga barang, menghitung besarnya diskon jika ada, lalu menampilkan harga barang yang harus dibayar.

banyaknya barang yang dibeli per jenis diskon

>= 10 5%.

>= 25 10%

>= 50 15%

>=100 20%

Pembeli dapat membeli beberapa jenis barang dalam satu slip pembayaran. Gunakan struktur loop untuk menangani hal ini. Tambahkan dalam algoritma, instruksi untuk membaca uang yang diterima, jika lebih besar dari harga barang maka hitung uang kembaliannya

14. Modifikasikan soal di bab 2 nomor 6 tentang biaya asuransi ladang pertanian sehingga petani dapat mengasuransikan beberapa jenis ladang sekaligus. Tambahkan dalam algoritma tersebut: apakah petani merupakan pelanggan lama, jika y maka biaya asuransi diberi diskon 10%. Lalu hitunglah biaya asuransi per ladang dan total serta diskon (jika ada).