BAB 5 ALGORITMA PEMROGRAMAN DASAR

Perangko dari Rusia pada Gambar 5.1. di samping ini bergambar seorang pria dengan nama Muhammad ibn Mūsā al-Khwārizmī. Bagi kalian yang sedang berkecimpung dalam dunia komputer maka seharusnya mengetahui siapa orang di samping ini. Dia adalah seorang ilmuwan Islam yang karya-karyanya dalam bidang matematika, astronomi, astrologi dan geografi banyak menjadi dasar perkembangan ilmu modern. Dan dari namanya istilah yang akan kita pelajari dalam bab ini muncul. Dari Al-Khawarizmi kemudian berubah menjadi ***algorithm*** dalam Bahasa Inggris dan diterjemahkan menjadi ***algoritma*** dalam Bahasa Indonesia.

(Sumber: www.wikipedia.org)

Gambar 5.1. Perangko bergambar Muhammad ibn Mūsā al-Khwārizmī.

Standar kompetensi algoritma pemrograman dasar terdiri atas empat kompetensi dasar. Dalam penyajian pada buku ini, setiap kompetensi dasar memuat uraian materi, dan latihan. Ringkasan diletakkan pada setiap akhir bab. Kompetensi dasar pada bab ini adalah menjelaskan variabel, konstanta dan tipe data, membuat algoritma/logika alur pemrograman, menerapkan pengelolaan array, dan mengoperasikan file. Sebelum mempelajari kompetensi ini ingatlah kembali sistem operasi, prinsip pemecahan masalah, dan materi-materi pendukung dari mata pelajaran matematika.

Pada akhir bab, tercantum soal-soal latihan yang disusun dari soal-soal yang mudah hingga soal-soal yang sulit. Latihan soal ini digunakan untuk mengukur kemampuan terhadap kompetensi dasar ini. Artinya setelah mempelajari kompetensi dasar ini secara mandiri dengan bimbingan guru sebagai fasilitator, ukurlah sendiri kemampuan dengan mengerjakan soal-soal latihan tersebut.

TUJUAN

Setelah mempelajari bab ini diharapkan pembaca akan mampu :

- Menjelaskan variabel, konstanta dan tipe data
- Membuat algoritma/logika alur pemrograman
- Menerapkan pengelolaan array
- Mengoperasikan file

5.1. VARIABEL, KONSTANTA DAN TIPE DATA

Variabel, konstanta dan tipe data merupakan tiga hal yang akan selalu kita jumpai ketika kita membuat program. Bahasa pemrograman apapun dari yang paling sederhana sampai yang paling kompleks, mengharuskan kita untuk mengerti ketiga hal tersebut.

5.1.1. Variabel

Variabel adalah tempat dimana kita dapat mengisi atau mengosongkan nilainya dan memanggil kembali apabila dibutuhkan. Setiap variabel akan mempunyai **nama (identifier)** dan **nilai**. Perhatikan contoh berikut.

Contoh 5.1. Nama variabel dan nilai.

```
username = "joni"
Nama = "Al-Khawarizmi"
Harga = 2500
HargaTotal = 34000
```

Pada contoh 5.1. di atas, **username, Nama, harga** dan **HargaTotal** adalah nama dari variabel sedangkan **"joni", "Al-Khawarizmi", 2500** dan **34000** adalah nilai dari masing-masing variabel. Nilai-nilai ini akan tersimpan di dalam nama variabel masing-masing sepanjang tidak kita rubah.

Pada sebagian besar bahasa pemrograman, variabel harus dideklarasikan lebih dulu untuk mempermudah *compiler* bekerja. Apabila variabel tidak dideklarasikan maka setiap kali *compiler* bertemu dengan variabel baru pada kode program akan terjadi waktu tunda karena *compiler* harus membuat variabel baru. Hal ini memperlambat proses kerja compiler. Bahkan pada beberapa bahasa pemrograman, *compiler* akan menolak untuk melanjutkan proses kompilasi.

Pemberian nama variabel harus mengikuti aturan yang ditetapkan oleh bahasa pemrograman yang kita gunakan. Namun secara umum ada aturan yang berlaku untuk hampir semua bahasa pemrograman. Aturan-aturan tersebut yaitu:

- Nama variabel harus diawali dengan huruf.
- Tidak boleh menggunakan spasi pada satu nama variabel. Spasi bisa diganti dengan karakter underscore (_).

- Nama variabel tidak boleh mengandung karakter-karakter khusus, seperti : .,+, -, *, /, <, >, &, (,) dan lain-lain.
- Nama variabel tidak boleh menggunakan kata-kata kunci d bahasa pemrograman

Contoh 5.2. Contoh penamaan variabel.

Penamaan yang benar	Penamaan yang salah
namasiswa **XY12** **harga_total** **JenisMotor** **alamatRumah**	**nama siswa** (salah karena menggunakan spasi) **12X** (salah karena dimulai dengan angka) **harga.total** (salah karena menggunakan karakter .) **Jenis Motor** (salah karena menggunakan spasi) **for** (salah karena menggunakan kata kunci bahasa pemrograman)

5.1.2. Konstanta

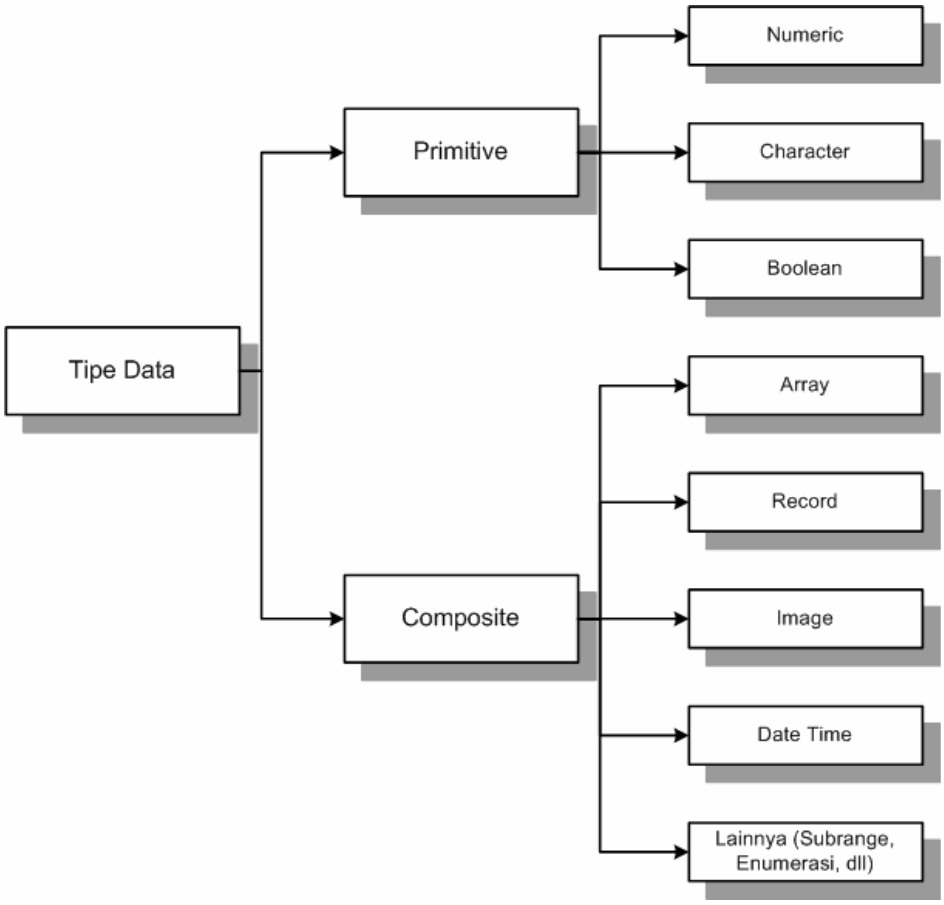
Konstanta adalah variabel yang nilai datanya bersifat tetap dan tidak bisa diubah. Jadi konstanta adalah juga variabel bedanya adalah pada nilai yang disimpannya. Jika nilai datanya sepanjang program berjalan tidak berubah-ubah, maka sebuah varibel lebih baik diperlakukan sebagai konstanta. Pada sebuah kode program, biasanya nilai data dari konstanta diberikan langsung di bagian deklarasi konstanta. Sedangkan untuk variabel biasanya hanya ditentukan nama variabel dan tipe datanya tanpa isian nilai data. Aturan penamaan variabel juga berlaku untuk penamaan konstanta. Demikian juga aturan penetapan tipe data.

Sebagai contoh, jika kita membuat program perhitungan matematik yang menggunakan nilai pi (3.14159) yang mungkin akan muncul dibanyak tempat pada kode program, kita dapat membuat pi sebagai konstanta. Penggunaan konstanta pi akan lebih memudahkan penulisan kode program dibanding harus mengetikkan nilai 3.14159 berulang-ulang.

5.1.3. Tipe Data

Tipe data adalah jenis data yang dapat diolah oleh komputer untuk memenuhi kebutuhan dalam pemrograman komputer. Setiap variabel atau konstanta yang ada dalam kode program, sebaiknya kita tentukan dengan pasti tipe datanya. Ketepatan pemilihan tipe data pada variabel atau konstanta akan sangat menentukan pemakaian sumberdaya komputer (terutama memori komputer). Salah satu tugas penting seorang *programmer* adalah memilih tipe data yang sesuai untuk menghasilkan program yang efisien dan berkinerja tinggi.

Ada banyak tipe data yang tersedia tergantung jenis bahasa pemrograman yang dipakai. Namun secara umum dapat dikelompokkan seperti pada Gambar 5.2.



Gambar 5.2. Pengelompokkan tipe data.

Tipe data *primitive* adalah tipe data dasar yang tersedia secara langsung pada suatu bahasa pemrograman. Sedangkan tipe data *composite* adalah tipe data bentukan yang terdiri dari dua atau lebih tipe data *primitive*.

Tipe data *numeric*

Tipe data *numeric* digunakan pada variabel atau konstanta untuk menyimpan nilai dalam bentuk bilangan atau angka. Semua bahasa pemrograman menyediakan tipe data *numeric*, hanya berbeda dalam jenis *numeric* yang diakomodasi.

Jenis yang termasuk dalam tipe data *numeric* antara lain *integer* (bilangan bulat), dan *float* (bilangan pecahan). Selain jenis, dalam bahasa pemrograman juga diterapkan presisi angka yang digunakan, misalnya tipe data *Single* adalah tipe data untuk bilangan pecahan dengan presisi yang terbatas, sedangkan tipe data *Double* adalah tipe data untuk bilangan pecahan dengan presisi yang lebih akurat. Pada bab-bab berikutnya yang membahas aplikasi bahasa pemrograman bagian ini akan diuraikan lebih lanjut.

Penentuan tipe data numeric untuk suatu variabel/konstanta harus sangat berhati-hati. Manual dan petunjuk pada masing-masing bahasa pemrograman pada bagian tipe data harus diperhatikan dengan seksama. Perhatikan contoh berikut.

Contoh 5.3. Penggunaan tipe data *numeric*.

Kode Program A

```
#include <iostream>
using namespace std;
int main() {
      int x, z;
      float y;
      x = 12;
      y = 2.15;
      z = x * y;
      cout << "X =" << x << endl;
      cout << "Y =" << y << endl;
      cout << "Z =" << z << endl;
      return 0; }
```

Hasil eksekusi Program A

```
X =12
Y =2.15
Z =25
```

Kode Program B

```
#include <iostream>
using namespace std;
int main() {
      int x;
      float y, z;
      x = 12.8;
      y = 2.15;
      z = x * y;
      cout << "X =" << x << endl;
      cout << "Y =" << y << endl;
      cout << "Z =" << z << endl;
      return 0; }
```

Hasil eksekusi Program B

```
X =12
Y =2.15
Z =25.8
```

Kode Program C

```
#include <iostream>
using namespace std;
int main() {
      int x;
      float y, z;
      x = 12;
      y = 2.15;
      z = x * y;
      cout << "X =" << x << endl;
      cout << "Y =" << y << endl;
      cout << "Z =" << z << endl;
      return 0; }
```

Hasil eksekusi Program C

```
X =12
Y =2.15
Z =25.8
```

Ketiga kode program di atas (A, B dan C) ditulis dengan bahasa C++. Sekilas sama namun berbeda pada penggunaan tipe data dan pengisian nilai.

Pada kode program A, variabel *x* dan *z* kita deklarasikan bertipe data *int* (Integer = bilangan bulat) dan *y* bertipe data *float* (pecahan). Hasil eksekusi program A menunjukkan hasil yang tidak kita inginkan. Nilai z yang merupakan perkalian *x* dengan *y* harusnya bernilai 25.8 (hasil dari 12 x 2.15). Namun karena *z* dideklarasikan bertipe data *int* maka hasilnya menjadi 25. Dari ketiga kode program di atas yang paling benar adalah kode program C. Mengapa kode program B salah? Cobalah cermati bagian kode yang dicetak tebal kemudian tentukan dimana terjadi kesalahan.

Character

Bersama dengan tipe data *numeric*, *character* merupakan tipe data yang paling banyak digunakan. Tipe data *character* kadang disebut sebagai *char* atau *string*. Tipe data *string* hanya dapat digunakan menyimpan teks atau apapun sepanjang berada dalam tanda petik dua ("...") atau petik tunggal ('...'). Perhatikan contoh berikut.

Contoh 5.4. Penggunaan tipe data character.

Kode program

```
#include <iostream>
using namespace std;

int main() {
    int x;
    x = 5;
    char huruf = 'A';
    char* kata = "Java";

    cout << "X = " << x << endl;
    cout << "Isi variabel huruf = "
    << huruf << endl;
    cout << "Isi variabel kata = " <<
    kata << endl;
    return 0; }
```

Hasil eksekusi program

```
X = 5
Isi variabel huruf = A
Isi variabel kata = Java
```

Pada contoh ini kita mendeklarasikan variabel *x* sebagai int (Integer), sedangkan variabel *huruf* dan *kata* bertipe data char (character). Perhatikan hasil eksekusi kode program di atas.

Boolean

Tipe data Boolean digunakan untuk menyimpan nilai True/False (Benar/Salah). Pada sebagian besar bahasa pemrograman nilai selain 0 menunjukkan True dan 0 melambangkan False. Tipe data ini banyak digunakan untuk pengambilan keputusan pada struktur percabangan dengan IF ... THEN atau IF ... THEN ... ELSE.

Array

Array atau sering disebut sebagai larik adalah tipe data yang sudah terstruktur dengan baik, meskipun masih sederhana. Array mampu menyimpan sejumlah data dengan tipe yang sama (homogen) dalam sebuah variabel. Setiap lokasi data array diberi nomor indeks yang berfungsi sebagai alamat dari data tersebut. Penjelasan tentang array akan disampaikan lebih detil pada bagian lain dari bab ini.

Record atau Struct

Seperti halnya Array, Record atau Struct adalah termasuk tipe data komposit. Record dikenal dalam bahasa Pascal/Delphi sedangkan Struct dikenal dalam bahasa C++. Berbeda dengan array, tipe data record mampu menampung banyak data dengan tipe data berbeda-beda (heterogen). . Sebagai ilustrasi array mampu menampung banyak data namun dengan satu tipe data yang sama, misalnya integer saja. Sedangkan dalam record, kita bisa menggunakan untuk menampung banyak data dengan tipe data yang berbeda, satu bagian integer, satu bagian lagi character, dan bagian lainnya Boolean. Biasanya record digunakan untuk menampung data suatu obyek. Misalnya, siswa memiliki nama, alamat, usia, tempat lahir, dan tanggal lahir. Nama akan akan menggunakan tipe data string, alamat bertipe data string, usia bertipe data single (numeric), tempat lahir bertipe data string dan tanggal lahir bertipe data date. Berikut ini contoh pendeklarasian record dalam Delphi.

Contoh 5.5. Deklarasi tipe data record pada Delphi.

```
Type TRecord_Siswa = Record
                Nama_Siswa          : String[30]
                Alamat              : String[50]
                Usia                : Real
           EndRecord
```

Image

Image atau gambar atau citra merupakan tipe data grafik. Misalnya grafik perkembangan jumlah siswa SMK, foto keluarga kita, video perjalanan dan lain-lain. Pada bahasa-bahasa pemrograman modern terutama yang berbasis visual tipe data ini telah didukung dengan sangat baik.

Date Time

Nilai data untuk tanggal (Date) dan waktu (Time) secara internal disimpan dalam format yang spesifik. Variabel atau konstanta yang dideklarasikan dengan tipe data Date dapat digunakan untuk menyimpan baik tanggal maupun jam. Tipe data ini masuk dalam kelompok tipe data composite karena merupakan bentukan dari beberapa tipe data. Berikut ini contoh tipe data dalam Visual Basic.

Contoh 5.6. Penggunaan tipe data date time pada Visual Basic.

```
Dim WaktuLahir As Date
WaktuLahir = “01/01/1997”
WaktuLahir = “13:03:05 AM”
WaktuLahir = “02/23/1998 13:13:40 AM”
WaktuLahir = #02/23/1998 13:13:40 AM#
```

Tipe data lain

Subrange

Tipe data subrange merupakan tipe data bilangan yang mempunyai jangkauan nilai tertentu sesuai dengan yang ditetapkan programmer. Biasanya tipe data ini mempunyai nilai batas minimum dan nilai batas maksimum. Tipe data ini didukung dengan sangat baik dalam Delphi. Berikut ini contoh deklarasi tipe data subrange dalam Delphi.

Contoh 5.7. Deklarasi tipe data subrange pada Delphi.

```
Type
     BatasIndeks = 1..20
     RentangTahun = 1950..2030
Var
     Indeks          : BatasIndeks
     Tahun           : RentangTahun
```

Enumerasi

Tipe data ini merupakan tipe data yang mempunyai elemen-elemen yang harus disebut satu persatu dan bernilai konstanta integer sesuai dengan urutannya. Nilai konstanta integer elemen ini diwakili oleh suatu nama variable yang ditulis di dalam kurung. Tipe data ini juga dijumpai pada Delphi dan bahasa pemrograman deklaratif seperti SQL. Berikut ini contoh deklarasi tipe data enumerasi dalam Delphi.

Contoh 5.8. Penggunaan tipe data enumerasi.

```
Type
     Hari_dlm_Minggu = (Nol, Senin, Selasa, Rabu,
                        Kamis, Jumat, Sabtu,
                        Minggu)
     Nama_Bulan = (Nol, Januari, Pebruari, Maret,
                   April, Mei, Juni, Juli,
                   Agustus,
                   September, Oktober, Nopember,
                   Desember)
Var
     No_Hari         : Hari_dlm_Minggu
     No_Bulan        : Nama_Bulan
```

Pada contoh di atas tipe data Hari_dlm_Minggu termasuk enumerasi dengan rentang nilai Nol, Senin sampai dengan Minggu dan nilai data dari 0, 1, sampai dengan 7. Sedangkan tipe data Nama_Bulan termasuk enumerasi dengan rentang nilai Nol, Januari sampai dengan Desember dan nilai data dari 0, 1, sampai dengan 12.

Object

Tipe data object digunakan untuk menyimpan nilai yang berhubungan dengan obyek-obyek yang disediakan oleh Visual Basic, Delphi dan dan bahasa pemrograman lain yang berbasis GUI. Sebagai contoh, apabila kita mempunyai form yang memiliki control Command button yang kita beri nama Command1, kita dapat mendeklarasikan variabel sebagai berikut :

Contoh 5.9. Penggunaan tipe data *object.*

```
Dim A As CommandButton
Set A = Command1
A.Caption = "HEY!!!"
A.FontBold = True
```

Pada contoh ini variabel A dideklarasikan bertipe data Object yaitu CommandButton. Kemudian kita set variabel A dengan control Command button yang ada pada form (Command1). Dengan cara ini kita dapat mengakses seluruh *property, method* dan *event* obyek Command1 dengan menggunakan variabel A.

Variant

Tipe data hanya ada di Visual Basic. Tipe ini adalah tipe data yang paling fleksibel di antara tipe data yang lain, karena dapat mengakomodasi semua tipe data yang lain seperti telah dijelaskan.

5.2. STRUKTUR ALGORITMA PEMROGRAMAN

5.2.1. Pengertian Algoritma

Algoritma adalah urutan langkah-langkah logis penyelesaian masalah yang disusun secara sistematis. Masalah dapat berupa apa saja, dengan catatan untuk setiap masalah, ada syarat kondisi awal yang harus dipenuhi sebelum menjalankan algoritma. Konsep algoritma sering kali disetarakan dengan sebuah resep. Sebuah resep biasanya memiliki daftar bahan atau bumbu yang akan digunakan, urutan pengerjaan dan bagaimana hasil dari urutan pengerjaan tersebut. Apabila bahan yang digunakan tidak tertera (tidak tersedia) maka resep tersebut tidak akan dapat dikerjakan. Demikian juga jika urutan pengerjaannya tidak beraturan, maka hasil yang diharapkan tidak akan dapat diperoleh.

Algoritma yang berbeda dapat diterapkan pada suatu masalah dengan syarat yang sama. Tingkat kerumitan dari suatu algoritma merupakan ukuran seberapa banyak komputasi yang dibutuhkan algoritma tersebut untuk menyelesaikan masalah. Umumnya, algoritma yang dapat menyelesaikan suatu permasalahan dalam waktu yang singkat memiliki tingkat kerumitan yang rendah, sementara algoritma yang membutuhkan waktu lama untuk menyelesaikan suatu masalah membutuhkan tingkat kerumitan yang tinggi.

Perhatikan algoritma sederhana berikut.

Contoh 5.10. Algoritma menghitung luas segitiga.

1. **Start**
2. **Baca data alas dan tinggi.**
3. **Luas adalah alas kali tinggi kali 0.5**
4. **Tampilkan Luas**
5. **Stop**

Algoritma di atas adalah algoritma yang sangat sederhana, hanya ada lima langkah. Pada algoritma ini tidak dijumpai perulangan ataupun pemilihan. Semua langkah dilakukan hanya satu kali.

Sekilas algoritma di atas benar, namun apabila dicermati maka algoritma ini mengandung kesalahan yang mendasar, yaitu tidak ada pembatasan pada nilai data untuk alas dan tinggi. Bagaimana jika nilai data alas atau tinggi adalah bilangan 0 atau bilangan negatif ? Tentunya hasil yang keluar menjadi tidak sesuai dengan yang diharapkan. Dalam kasus seperti ini kita perlu menambahkan langkah untuk memastikan nilai alas dan tinggi memenuhi syarat, misalnya dengan melakukan pengecekan pada input yang masuk. Apabila input nilai alas dan tinggi kurang dari 0 maka program tidak akan dijalankan. Sehingga algoritma di atas dapat dirubah menjadi seperti contoh berikut.

Contoh 5.11. Hasil perbaikan algoritma perhitungan luas segitiga.

1. **Start**
2. **Baca data alas dan tinggi.**
3. **Periksa data alas dan tinggi, jika nilai data alas dan tinggi lebih besar dari nol maka lanjutkan ke langkah ke 4 jika tidak maka stop**
4. **Luas adalah alas kali tinggi kali 0.5**
5. **Tampilkan Luas**
6. **Stop**

Dari penjelasan di atas dapat diambil kesimpulan pokok tentang algoritma. Pertama, **algoritma harus benar**. Kedua **algoritma harus berhenti**, dan setelah berhenti, **algoritma memberikan hasil yang benar**.

5.2.2. Cara Penulisan Algoritma

Ada tiga cara penulisan algoritma, yaitu :

- ***Structured English (SE)***

SE merupakan alat yang cukup baik untuk menggambarkan suatu algoritma. Dasar dari SE adalah Bahasa Inggris, namun kita dapat memodifikasi dengan Bahasa Indonesia sehingga kita boleh menyebutnya sebagai Structured Indonesian (SI). Algoritma seperti pada Contoh 5.10 dan 5.11 merupakan algoritma yang ditulis menggunakan SI. Karena dasarnya adalah bahasa sehari-hari, maka SE atau SI lebih tepat untuk menggambarkan suatu algoritma yang akan dikomunikasikan kepada pemakai perangkat lunak.

- ***Pseudocode***

Pseudocode mirip dengan SE. Karena kemiripan ini kadang-kadang SE dan Pseudocode dianggap sama. *Pseudo* berarti imitasi atau tiruan atau menyerupai, sedangkan *code* menunjuk pada kode program. Sehingga *pseudocode* adalah kode yang mirip dengan instruksi kode program sebenarnya. *Pseudocode* didasarkan pada bahasa pemrograman yang sesungguhnya seperti BASIC, FORTRAN atau PASCAL. *Pseudocode* yang berbasis bahasa PASCAL merupakan *pseudocode* yang sering digunakan. Kadang-kadang orang menyebut *pseudocode* sebagai *PASCAL-LIKE* algoritma. Apabila Contoh 5.10 ditulis dalam pseudocode berbasis bahasa BASIC akan tampak seperti pada contoh 5.12.

Contoh 5.12. *Pseudocode.*

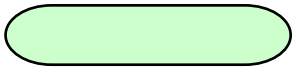
```
1. Start
2. READ alas, tinggi
3. Luas = 0.5 * alas * tinggi
4. PRINT Luas
5. Stop
```

Pada Contoh 5.12 tampak bahwa algoritma sudah sangat mirip dengan bahasa BASIC. Pernyataan seperti READ dan PRINT merupakan *keyword* yang ada pada bahasa BASIC yang masing-masing menggantikan kata “baca data” dan “tampilkan”. Dengan menggunakan *pseudocode* seperti di atas maka proses penterjemahan dari algoritma ke kode program menjadi lebih mudah.

- ***Flowchart***

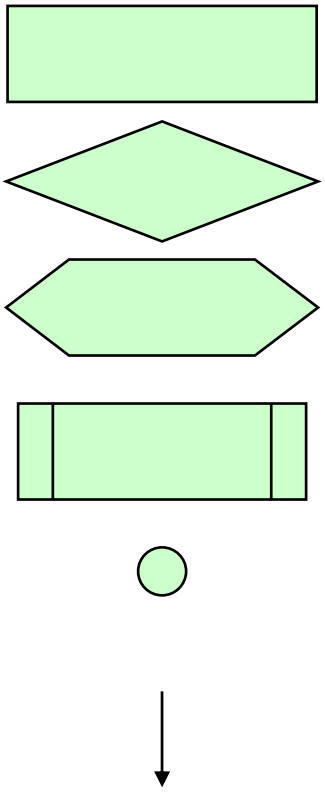
Flowchart atau bagan alir adalah skema/bagan (*chart*) yang menunjukkan aliran (*flow*) di dalam suatu program secara logika. *Flowchart* merupakan alat yang banyak digunakan untuk menggambarkan algoritma dalam bentu notasi-notasi tertentu. Secara lebih detil bagian ini akan dibahas pada bagian berikutnya.

Pada *flowchart* ada beberapa simbol penting yang digunakan untuk membuat algoritma sebagaimana tercantum pada Gambar 5.3.



Notasi ini disebut *Terminator* yang berarti digunakan untuk menunjukkan awal dan akhir suatu algoritma

Notasi ini disebut *Data* yang digunakan untuk mewakili data input atau output atau menyatakan operasi pemasukan data dan pencetakan hasil.



Notasi ini disebut *Process* yang digunakan untuk mewakili suatu proses.

Notasi ini disebut *Decision* yang digunakan untuk suatu pemilihan, penyeleksian kondisi di dalam suatu program

Notasi ini disebut *Preparation* yang digunakan untuk memberi nilai awal, nilai akhir, penambahan/pengurangan bagi suatu variable *counter*.

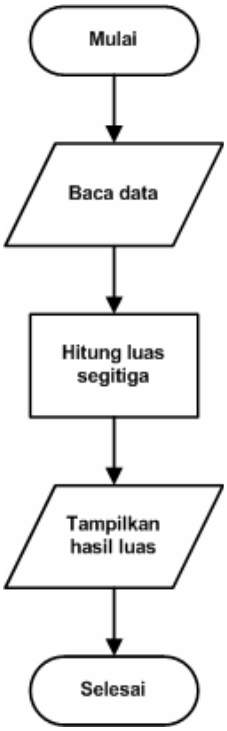
Notasi ini disebut *Predefined Process* yang digunakan untuk menunjukkan suatu operasi yang rinciannya ditunjukkan ditempat lain (prosedur, sub-prosedur, fungsi)

Notasi ini disebut *Connector* yang digunakan untuk menunjukkan sambungan dari *flowchart* yang terputus di halaman yang sama atau haraman berikutnya.

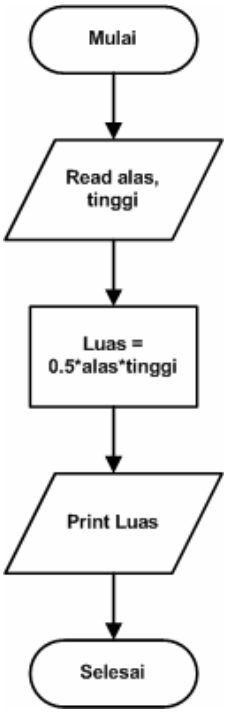
Notasi ini disebut *Arrow* yang digunakan untuk menunjukkan arus data atau aliran data dari proses satu ke proses lainnya.

Gambar 5.3. Simbol-simbol yang digunakan dalam *flowchart.*

Program *Flowchart* dapat terdiri dari dua macam, yaitu bagan alir logika program (*program logic flowchart*) dan bagan alir program komputer terinci (*detailed computer program flowchart*). Bagan alir logika program digunakan untuk menggambarkan tiap-tiap langkah di dalam program komputer secara logika dan biasanya dipersiapkan oleh seorang analis system. Sedangkan bagan alir program komputer terinci digunakan untuk menggambarkan instruksi-instruksi program komputer secara terinci dan biasanya dipersiapkan oleh seorang programmer. Apabila Contoh 5.10 dibuat program *flowchart*nya maka akan tampak pada gambar 5.4.



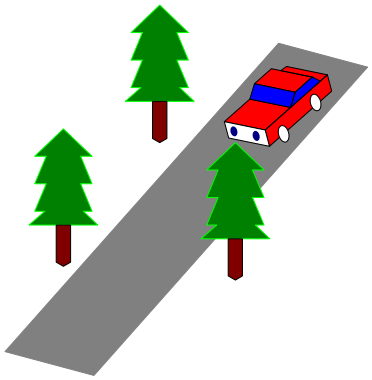
Bagan alir logika program



Bagan alir program komputer terinci

Gambar 5.4. *Program flowchart*.

5.2.3. Struktur Algoritma Berurutan



Gambar 5.5. Mobil sedang berjalan pada jalur lurus.

Ada tiga struktur dasar yang digunakan dalam membuat algoritma yaitu struktur berurutan (*sequencing*), struktur pemilihan/keputusan/percabangan (*branching*) dan struktur pengulangan (*looping*). Sebuah algoritma biasanya akan menggabungkan ketiga buah struktur ini untuk menyelesaikan masalah.

Pada bagian ini kita akan bahas lebih dulu struktur algoritma berurutan. Struktur berurutan dapat kita samakan dengan mobil yang sedang berjalan pada jalur lurus yang tidak terdapat persimpangan seperti tampak pada Gambar 5.5. Mobil tersebut akan melewati kilometer demi kilometer jalan sampai tujuan tercapai.

Struktur berurutan terdiri satu atau lebih instruksi. Tiap instruksi dikerjakan secara berurutan sesuai dengan urutan penulisannya, yaitu sebuah instruksi dieksekusi setelah instruksi sebelumnya selesai dieksekusi. Urutan instruksi menentukan keadaan akhir dari algoritma. Bila urutannya diubah, maka hasil akhirnya mungkin juga berubah. Menurut Goldshlager dan Lister (1988) struktur berurutan mengikuti ketentuan-ketentuan sebagai berikut:

- tiap instruksi dikerjakan satu persatu
- tiap instruksi dilaksanakan tepat sekali, tidak ada yang diulang
- urutan instruksi yang dilaksanakan pemroses sama dengan urutan aksi sebagaimana yang tertulis di dalam algoritmanya
- akhir dari instruksi terakhir merupakan akhir algoritma.

Contoh 5.13. *Flowchart* untuk menghitung luas bangun.

Buatlah *flowchart* untuk menghitung:

a. volume balok

b. luas lingkaran

Penyelesaian:

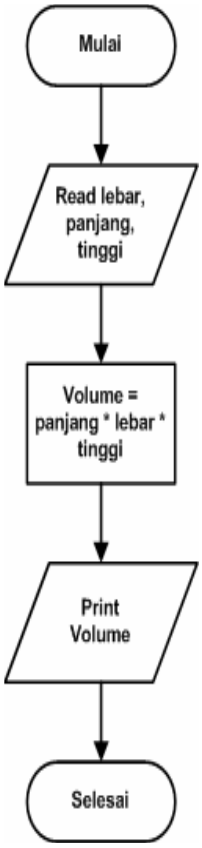
Soal ini merupakan permasalahan dengan algoritma struktur berurutan karena tidak ada proses pemilihan atau pengulangan. Untuk volume balok, kita harus menentukan variabel input dan output yang dibutuhkan. Untuk menghitung volume balok dibutuhkan variabel input panjang, lebar dan tinggi. Sedangkan variabel outputnya adalah volume. Pada luas lingkaran dibutuhkan variabel input radius dan variabel output luas. Untuk menghitung luas lingkaran ini kita juga membutuhkan konstanta phi. *Flowchart* untuk dua masalah ini dapat dilihat pada Gambar 5.6.

Contoh 5.14. *Flowchart* untuk konversi suhu.

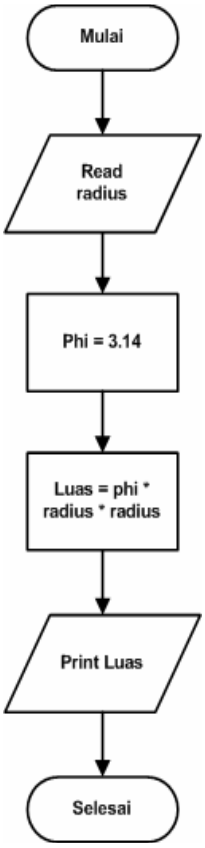
Buat *flowchart* untuk mengubah temperatur dalam Fahrenheit menjadi temperatur dalam Celcius dengan rumus $^{o}C = 5/9 \times (^{o}F - 32)$.

Penyelesaian:

Soal ini juga masih menggunakan algoritma dengan struktur berurutan. Variabel input yang dibutuhkan adalah F dan variabel outputnya adalah C. *Flowchart* untuk dua masalah ini dapat dilihat pada Gambar 5.7.

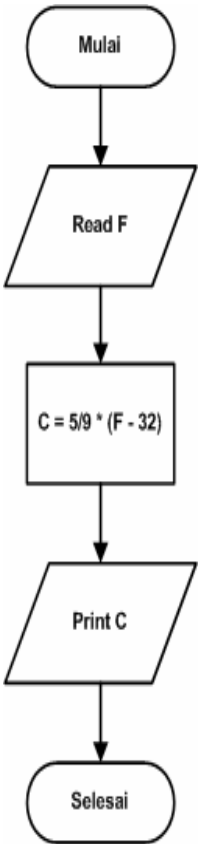


a. *flowchart* menghitung volume balok



b. *flowchart* menghitung luas lingkaran

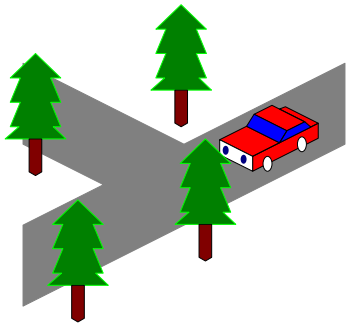
Gambar 5.6. *Flowchart* menghitung volume balok dan luas lingkaran.



Gambar 5.7. *Flowchart* untuk konversi suhu.

5.2.4. Struktur Algoritma Percabangan

Sebuah program tidak selamanya akan berjalan dengan mengikuti struktur berurutan, kadang-kadang kita perlu merubah urutan pelaksanaan program dan menghendaki agar pelaksanaan program meloncat ke baris tertentu. Peristiwa ini kadang disebut sebagai percabangan/pemilihan atau keputusan. Hal ini seperti halnya ketika mobil berada dalam persimpangan seperti pada Gambar 5.7. Pengemudi harus memutuskan apakah harus menempuh jalur yang kanan atau yang kiri.



Gambar 5.8. Mobil sedang dalam persimpangan.

Pada struktur percabangan, program akan berpindah urutan pelaksanaan jika suatu kondisi yang disyaratkan dipenuhi. Pada proses seperti ini simbol *flowchart* Decision harus digunakan. Simbol decision akan berisi pernyataan yang akan diuji kebenarannya. Nilai hasil pengujian akan menentukan cabang mana yang akan ditempuh.

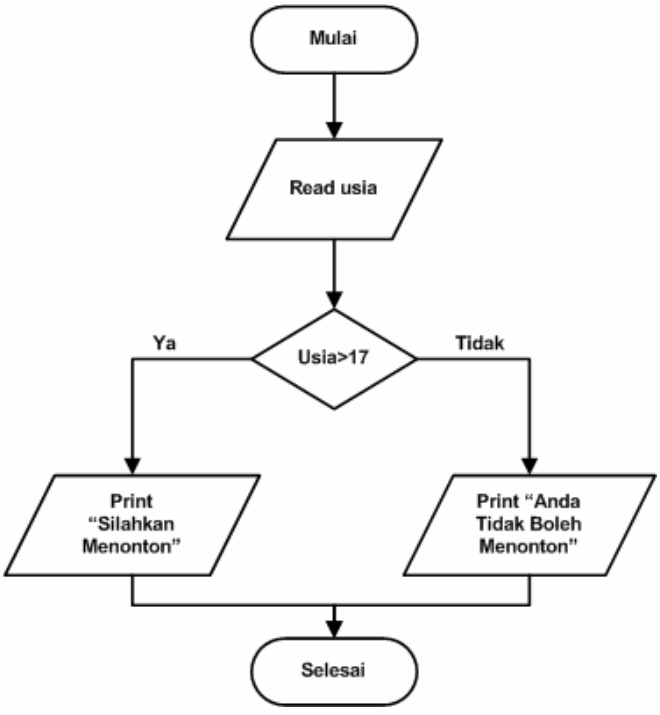
Contoh 5.15. Struktur percabangan untuk masalah batasan umur.

Sebuah aturan untuk menonton sebuah film tertentu adalah sebagai berikut, jika usia penonton lebih dari 17 tahun maka penonton diperbolehkan dan apabila kurang dari 17 tahun maka penonton tidak diperbolehkan nonton. Buatlah *flowchart* untuk permasalahan tersebut.

Penyelesaian:

Permasalahan diatas merupakan ciri permasalahan yang menggunakan struktur percabangan. Hal ini ditandai dengan adanya pernyataan *jika .. maka ...*(atau If ... Then dalam Bahasa Inggris.

Flowchart penyelesaian masalah tampak pada Gambar 5.9. Pada gambar tersebut, tampak penggunaan simbol Decision. Pada simbol ini terjadi pemeriksaan kondisi, yaitu apakah usia lebih dari 17 tahun atau tidak. Jika jawaban ya maka program akan menghasilkan keluaran teks "Silahkan Menonton", sedangkan jika input usia kurang dari 17 tahun maka program akan menghasilkan keluaran teks "Anda Tidak Boleh Menonton".



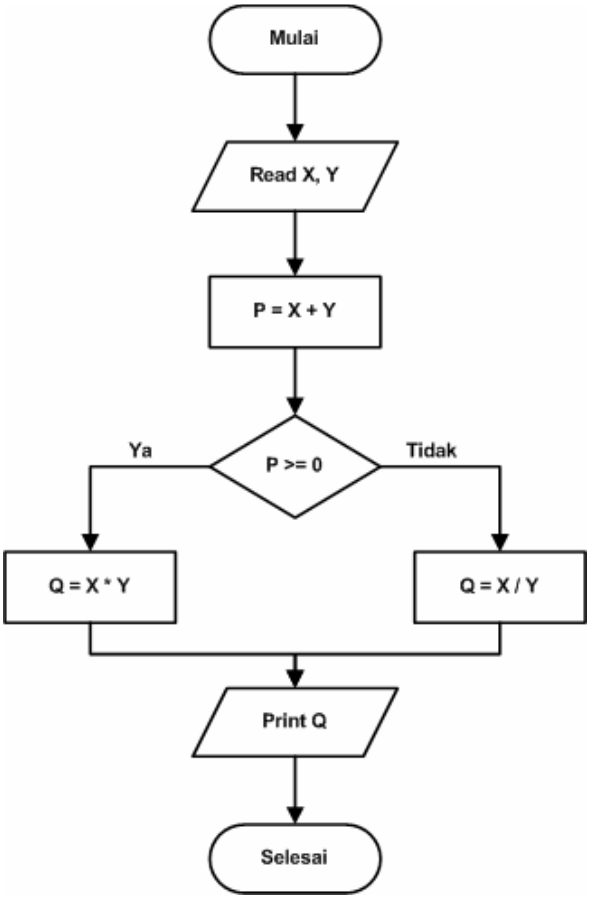
Gambar 5.9. *Flowchart* penyelesaian masalah nonton film.

Contoh 5.16. Struktur percabangan untuk perhitungan dua buah bilangan.

Dalam suatu perhitungan nilai P = X + Y. Jika P positif, maka Q = X * Y, sedangkan jika negative maka nilai Q = X/Y. Buatlah *flowchart* untuk mencari nilai P dan Q

Penyelesaian:

Pada contoh ini input yang dibutuhkan adalah nilai X dan Y, sedangkan proses pemeriksaan kondisi dilakukan pada nilai P apakah positif (termasuk 0) ataukah negative. Perhatikan *flowchart* penyelesaian masalah pada Gambar 5.10.



Gambar 5.10. *Flowchart* penyelesaian untuk perhitungan dua buah bilangan.

Kedua contoh di atas (5.15 dan 5.16) merupakan contoh struktur percabangan sederhana yang melibatkan hanya satu percabangan. Pada masalah-masalah yang lebih rumit, kita akan menjumpai lebih banyak percabangan. Kita juga akan menjumpai suatu struktur percabangan berada di

dalam struktur percabangan yang lain, atau yang biasa disebut nested (bersarang). Perhatikan contoh-contoh berikut.

Contoh 5.17. Struktur percabangan bersarang untuk masalah fotokopi.

Sebuah usaha fotokopi mempunyai aturan sebagai berikut :

- jika yang fotokopi statusnya adalah langganan, maka berapa lembar pun dia fotokopi, harga perlembarnya Rp. 75,-
- jika yang fotokopi bukan langganan, maka jika dia fotokopi kurang dari 100 lembar harga perlembarnya Rp. 100,-. Sedangkan jika lebih atau sama dengan 100 lembar maka harga perlembarnya Rp. 85,-.

Buat *flowchart* untuk menghitung total harga yang harus dibayar jika seseorang memfotokopi sejumlah X lembar.

Penyelesaian:

Pada contoh ini, masalah terlihat lebih rumit. Ada dua percabangan yang terjadi. Yang pertama adalah pemeriksaan apakah status seseorang pelanggan atau bukan. Kedua, apabila status seseorang bukan pelanggan, maka dilakukan pemeriksaan berapa jumlah lembar fotokopi, apakah lebih dari 100 lembar atau tidak.

Pada soal ini kita juga menjumpai apa yang disebut sebagai nested. Perhatikan pernyataan pada syarat kedua dari persoalan di atas.

> ***jika*** *yang fotokopi bukan langganan, maka* ***jika*** *dia fotokopi kurang dari 100 lembar harga perlembarnya Rp. 100*

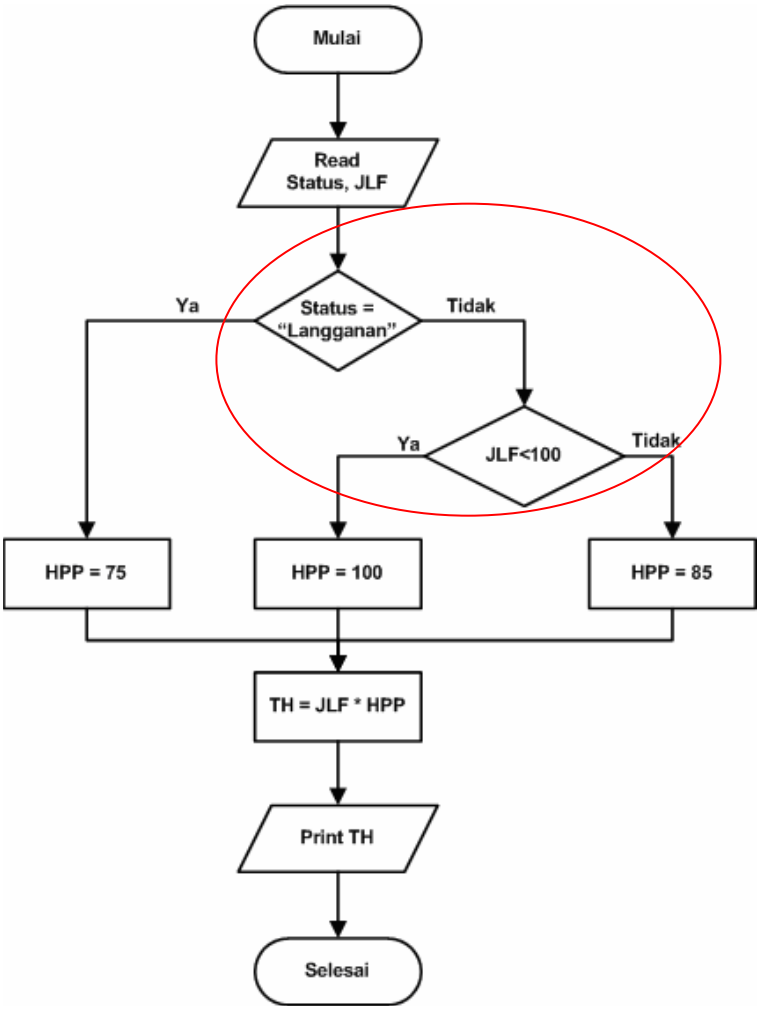
pernyataan jika yang kedua berada di dalam jika yang pertama.

Input yang dibutuhkan untuk permasalahan ini adalah status orang yang fotokopi dan jumlah lembar yang difotokopi. Sehingga variable input yang digunakan adalah:

- **Status** untuk status orang yang fotokopi
- **JLF** untuk jumlah lembar yang difotokopi

Selain itu terdapat variable dengan nama HPP yang digunakan untuk menyimpan harga per lembar dan TH untuk menyimpan nilai total harga. Perhatikan, variable Status bertipe data char, sehingga penulisannya harus menggunakan tanda " ".

Flowchart penyelesaian masalah ini dapat dilihat pada Gambar 5.11.



Gambar 5.11. *Flowchart* penyelesaian untuk masalah fotokopi.

Contoh 5.18. Struktur percabangan bersarang untuk masalah kelulusan siswa.

Aturan kelulusan siswa pada mata pelajaran Pemrograman Web diterapkan sebagai berikut :

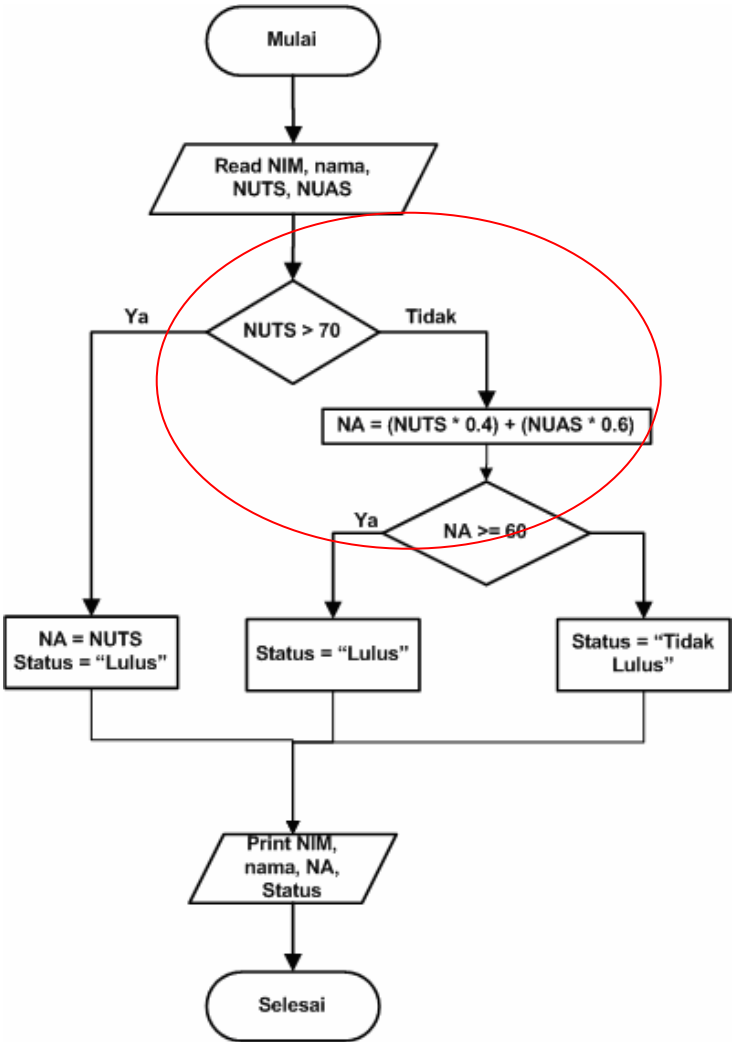
- Jika nilai ujian tengah semester (UTS) lebih besar dari 70 maka siswa dinyatakan lulus dan Nilai Akhir sama dengan nilai UTS.
- Jika nilai UTS kurang atau sama dengan 70 maka siswa dinyatakan lulus jika Nilai Akhir lebih besar atau sama dengan 60 dimana Nilai Akhir = (nilai UTS x 40%) + (nilai UAS x 60%).

Buatlah *flowchart* penyelesaian masalah tersebut apabila output yang diinginkan adalah NIM, Nama Siswa, Nilai Akhir dan Status Kelulusan.

Penyelesaian:

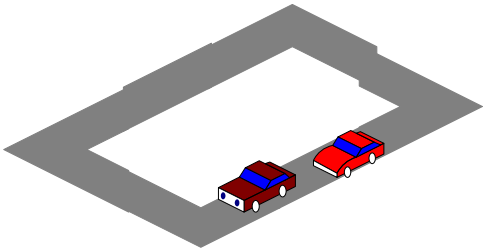
Pada contoh ini, ada dua percabangan. Yang pertama adalah pemeriksaan apakah nilai UTS siswa lebih dari 70. Kedua, apabila nilai UTS tidak lebih dari 70, maka dilakukan pemeriksaan apakah nilai akhir lebih dari 60.

Input yang dibutuhkan untuk permasalahan ini adalah NIM, nama siswa, nilai UTS, dan nilai UAS. Sehingga variable input yang digunakan adalah: NIM untuk Nomor induk siswa, nama untuk nama siswa, NUTS untuk nilai ujian tengah semester, dan NUAS untuk nilai ujian akhir semester. Sedangkan variabel ouput terdiri dari NA yang digunakan untuk menyimpan nilai akhir dan Status untuk menyimpan status kelulusan.



Gambar 5.12. *Flowchart* penyelesaian untuk kelulusan siswa.

5.2.5. Struktur Algoritma Pengulangan



Gambar 5.13. Lomba balap mobil di sirkuit.

Dalam banyak kasus seringkali kita dihadapkan pada sejumlah pekerjaan yang harus diulang berkali. Salah satu contoh yang gampang kita jumpai adalah balapan mobil seperti tampak pada gambar 5.13. Mobil-mobil peserta harus mengelilingi lintasan sirkuit berkali-kali sesuai yang ditetapkan dalam aturan lomba. Siapa yang mencapai garis akhir paling cepat, dialah yang menang.

Pada pembuatan program komputer, kita juga kadang-kadang harus mengulang satu atau sekelompok perintah berkali-kali agar memperoleh hasil yang diinginkan. Dengan menggunakan komputer, eksekusi pengulangan mudah dilakukan. Hal ini karena salah satu kelebihan komputer dibandingkan dengan manusia adalah kemampuannya untuk mengerjakan tugas atau suatu instruksi berulangkali tanpa merasa lelah, bosan, atau malas. Bandingkan dengan pengendara mobil balap, suatu ketika pasti dia merasa lelah dan bosan untuk berputar-putar mengendarai mobil balapnya.

Struktur pengulangan terdiri dari dua bagian :

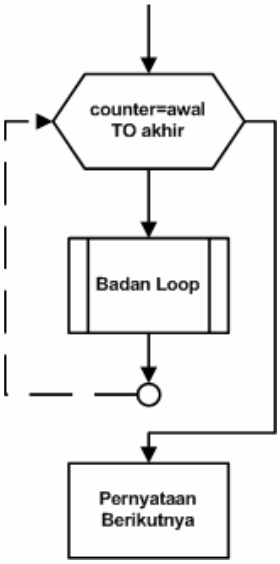
1. **Kondisi pengulangan**, yaitu syarat yang harus dipenuhi untuk melaksanakan pengulangan. Syarat ini biasanya dinyatakan dalam ekspresi Boolean yang harus diuji apakah bernilai benar (true) atau salah (false)
2. **Badan pengulangan (loop body)**, yaitu satu atau lebih instruksi yang akan diulang

Pada struktur pengulangan, biasanya juga disertai bagian inisialisasi dan bagian terminasi. **Inisialisasi** adalah instruksi yang dilakukan sebelum pengulangan dilakukan pertama kali. Bagian insialisasi umumnya digunakan untuk memberi nilai awal sebuah variable. Sedangkan **terminasi** adalah instruksi yang dilakukan setelah pengulangan selesai dilaksanakan.

Ada beberapa bentuk pengulangan yang dapat digunakan, masing-masing dengan syarat dan karakteristik tersendiri. Beberapa bentuk dapat dipakai untuk kasus yang sama, namun ada bentuk yang hanya cocok untuk kasus tertentu saja. Pemilihan bentuk pengulangan untuk masalah tertentu dapat mempengaruhi kebenaran algoritma. Pemilihan bentuk pengulangan yang tepat bergantung pada masalah yang akan diprogram.

- **Struktur pengulangan dengan For**

Pengulangan dengan menggunakan *For*, merupakan salah teknik pengulangan yang paling tua dalam bahasa pemrograman. Hampir semua bahasa pemrograman menyediakan metode ini, meskipun sintaksnya mungkin berbeda. Pada struktur *For* kita harus tahu terlebih dahulu seberapa banyak badan loop akan diulang. Struktur ini menggunakan sebuah variable yang biasa disebut sebagai *loop's counter*, yang nilainya akan naik atau turun selama proses pengulangan. *Flowchart* umum untuk struktur *For* tampak pada Gambar 5.14. Perhatikan penggunaan simbol *preparation* pada *flowchart* tersebut.



Gambar 5.14. Struktur algoritma pengulangan dengan For.

Dalam mengeksekusi sebuah pengulangan dengan For, urutan langkah-langkah adalah sebagai berikut :

1. Menetapkan nilai *counter* sama dengan awal.
2. Memeriksa apakah nilai *counter* lebih besar daripada nilai akhir. Jika benar maka keluar dari proses pengulangan. Apabila kenaikan bernilai negatif, maka proses akan memeriksa apakah nilai *counter* lebih kecil daripada nilai akhir. Jika benar maka keluar dari proses pengulangan.
3. Mengeksekusi pernyataan yang ada di badan loop
4. Menaikkan/menurunkan nilai *counter* sesuai dengan jumlah yang ditentukan pada argument *increment*. Apabila argument *increment* tidak ditetapkan maka secara default nilai *counter* akan dinaikkan 1.
5. Ulang kembali mulai langkah no 2.

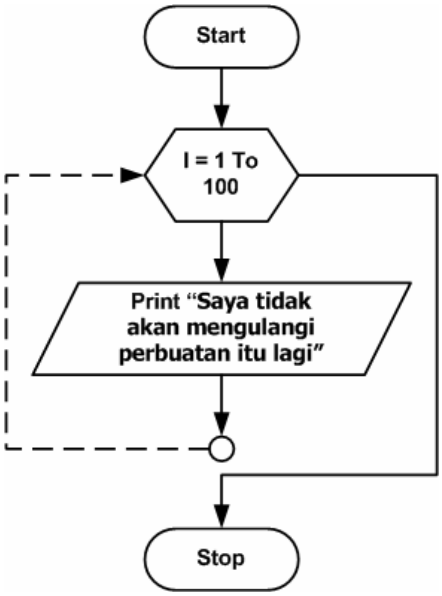
Satu hal yang penting yang harus kita perhatikan adalah nilai *counter* selalu ditetapkan diawal dari pengulangan. Apabila kita mencoba merubah nilai akhir pada badan *loop*, maka tidak akan berdampak pada berapa banyak pengulangan akan dilakukan.

Contoh 5.19. Algoritma untuk mencetak pernyataan sebanyak 100 kali

Mungkin kalian pernah ketika masih di sekolah dasar melakukan perbuatan nakal yang membuat kalian disuruh menuliskan pernyataan tertentu sebanyak 100 kali sebagai hukuman atas kenakalan tersebut. Misalkan pernyataan yang harus ditulis adalah "Saya tidak akan mengulangi perbuatan itu lagi". Bagaimanakah caranya algoritma untuk kasus ini?

Penyelesaian:

Pada contoh ini, kita memerlukan variabel counter, misalkan kita beri nama I. Nilai awalnya adalah 1 dan nilai akhirnya adalah 100. Sedangkan increment atau kenaikan tiap kali pengulangan dari I adalah satu. Perintah untuk mencetak pernyataan akan diulang satu persatu sampai nilai akhir dari *counter* terpenuhi (100). *Flowchart* penyelesaian untuk contoh ini dapat dilihat pada Gambar 5.15.



Gambar 5.15. *Flowchart* menulis pernyataan 100 kali.

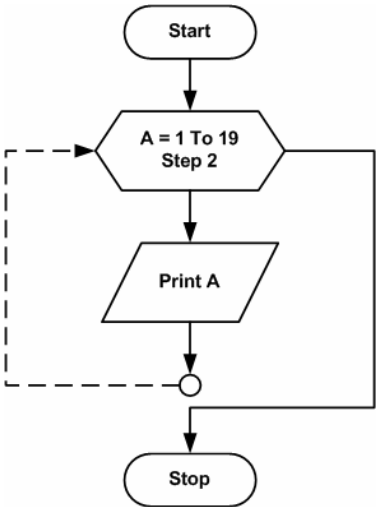
Perhatikan bagaimana mudahnya kita melakukan pengulangan. Pada Gambar 5.15 tersebut *increment* tidak dicantumkan, karena sesuai langkah-langkah yang dijelaskan sebelumnya, jika tidak dicantumkan maka otomatis nilai *increment* adalah satu.

Contoh 5.20. *Flowchart* untuk mencetak anggota suatu himpunan.

Diketahui sebuah himpunan A yang beranggotakan bilangan 1, 3, 5, .., 19. Buatlah *flowchart* untuk mencetak anggota himpunan tersebut.

Penyelesaian:

Pada contoh ini, kita memerlukan variabel *counter*, misalkan kita beri nama A (sesuai dengan nama himpunan). Nilai awalnya adalah 1 dan nilai akhirnya adalah 19. Dari pola himpunan kita tahu bahwa kenaikan bilangan adalah 2 (1 ke 3, 3 ke 5, dan seterusnya). Sehingga bisa kita nyatakan *increment* atau kenaikan tiap kali pengulangan dari A adalah 2. *Flowchart* penyelesaian untuk contoh ini dapat dilihat pada Gambar 5.16.



Gambar 5.16. *Flowchart* mencetak anggota himpunan.

Pada Gambar 5.16 tersebut, perhatikan pada simbol *preparation*. Terdapat tambahan pernyataan **step 2**. Inilah yang disebut sebagai *increment*. Setiap kali pengulangan, maka nilai *counter* yaitu A akan bertambah 2 sehingga yang akan tercetak adalah 1, 3, 5, .., 19.

Contoh 5.21. Menentukan hasil dari suatu *flowchart* pengulangan.

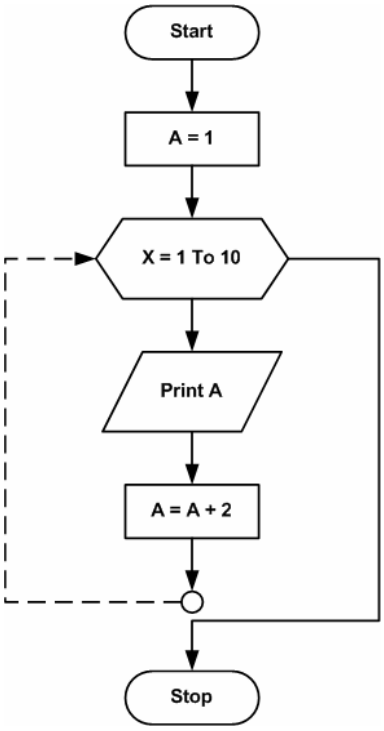
Perhatikan *flowchart* pada Gambar 5.17. Tentukan hasil dari *flowchart* tersebut.

Penyelesaian:

Pada contoh ini, kita mencoba menentukan hasil dari sebuah *flowchart*. Bagaimana menurut kalian jawabannya? Marilah kita uraikan jalannya fowchart tersebut.

Pada *flowchart*, setelah Start, kita meletakkan satu proses yang berisi pernyataan A = 1. Bagian inilah yang disebut **inisialisasi**. Kita memberi nilai awal untuk A = 1. Variabel *counter*-nya adalah X dengan nilai awal 1 dan nilai akhir 10, tanpa *increment* (atau secara *default increment*-nya

adalah 1). Ketika masuk ke badan *loop* untuk pertama kali maka akan dicetak langsung nilai variabel A. Nilai variabel A masih sama dengan 1. Kemudian proses berikutnya adalah pernyataan A = A + 2. Pernyataan ini mungkin agak aneh, tapi ini adalah sesuatu yang pemrograman. Arti dari pernyataan ini adalah gantilah nilai A yang lama dengan hasil penjumlah nilai A lama ditambah 2. Sehingga A akan bernilai 3. Kemudian dilakukan pengulangan yang ke-dua. Pada kondisi ini nilai A adalah 3, sehingga yang tercetak oleh perintah print adalah 3. Baru kemudian nilai A kita ganti dengan penjumlahan A + 2. Nilai A baru adalah 5. Demikian seterusnya. Sehingga output dari *flowchart* ini adalah 1, 3, 5, 7, .., 19.



Gambar 5.17. *Flowchart* mencetak bilangan tertentu.

Kita dapat melihat sekarang bahwa Gambar 5.16 dan 5.17 memberikan output yang sama. Dari kedua *flowchart* tersebut, Gambar 5.17 merupakan *flowchart* yang disarankan. Meskipun lebih panjang tetapi lebih terstruktur. Selain itu tidak semua bahasa pemrograman memberi fasilitas pengaturan *increment.*

Flowchart pada Gambar 5.17 harus kita perhatikan benar-benar, terutama posisi pernyataan Print A. Cobalah membalik posisinya sehingga letak pernyataan A = A + 2 berada di atas pernyataan Print A. Bagaimanakah hasilnya?

Seperti halnya struktur percabangan, kita juga akan menjumpai bentuk struktur pengulangan bersarang (*nested*). Artinya, ada suatu pengulangan yang berada di dalam pengulangan yang lain. Perhatikan Contoh 5.22 berikut ini.

Contoh 5.21. Menentukan hasil dari suatu *flowchart* pengulangan.

Perhatikan *flowchart* pada Gambar 5.18. Tentukan hasil dari *flowchart* tersebut.

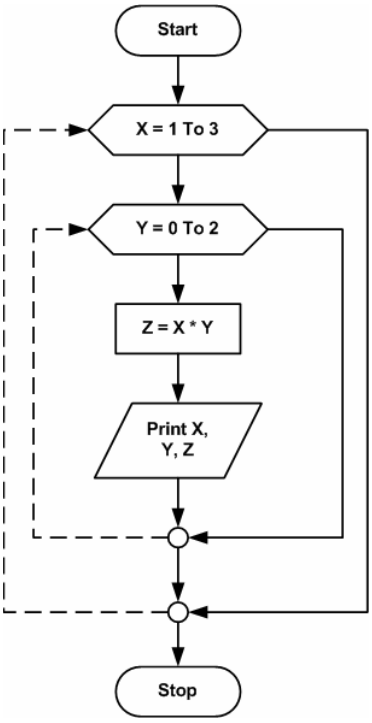
Penyelesaian:

Pada contoh ini, kita mencoba menentukan hasil dari sebuah *flowchart* dengan pengulangan bersarang. Bagaimana menurut kalian jawabannya? Marilah kita uraikan jalannya *flowchart* tersebut.

Pada Gambar 5.18 tersebut, terlihat ada dua simbol *preparation*. Yang pertama dengan variabel *counter* X dan yang kedua dengan variabel *counter* Y. Dalam posisi, variabel *counter* Y terletak setelah variabel *counter* X. Hal ini berarti pengulangan dengan variabel *counter* Y terletak di dalam variabel *counter* X. Inilah yang disebut sebagai pengulangan bersarang.

Pada bentuk pengulangan seperti ini, alur eksekusi program akan berjalan sebagai berikut:

- Variabel X akan diisi dengan nilai awal *counter*-nya yaitu 1.
- Variabel Y akan diisi dengan nilai awal *counter*-nya yaitu 0
- Nilai Z dihitung dengan mengalikan X dengan Y. Nilai X = 1 dan nilai Y = 0 jadi nilai Z = 0
- Nilai X, Y dan Z dicetak di layar.
- Alur berputar, dan nilai Y dinaikkan menjadi 1 sedangkan nilai X masih satu (karena bagian X belum berputar).
- Nilai Z = 1 hasil perkalian X = 1 dan Y = 1.
- Nilai X, Y, dan Z akan dicetak lagi di layar.
- Alur berputar kembali sehingga nilai Y menjadi 2, sehingga nilai Z akan menjadi 2.



Gambar 5.18. *Flowchart* dengan pengulangan bersarang.

- Setelah ini perputaran akan keluar dari Y karena nilai akhir *counter* yaitu sudah tercapai.
- Nilai X akan dinaikkan 1 menjadi 2, dan proses kembali melakukan pengulangan pada bagian Y seperti di atas.
- Proses pengulangan diulang terus sampai nilai akhir dari *counter* X yaitu 3 tercapai. Sehingga hasil akhir dari *flowchart* tersebut adalah sebagai berikut:

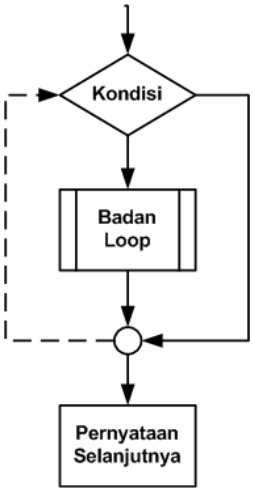
```
X       Y       Z
1       0       0
1       1       1
1       2       2
2       0       0
2       1       2
2       2       4
3       0       0
3       1       3
3       2       6
```

Dari Contoh 5.21. di atas kita bisa melihat ada aturan-aturan yang harus dipenuhi dalam pengulangan bersarang, yaitu:

- Masing-masing pengulangan (badan loop) mempunyai variabel *counter* sendiri-sendiri.
- Pengulangan-pengulangan tersebut tidak boleh tumpang tindih.

- **Struktur pengulangan dengan While**

Pada pengulangan dengan *For,* banyaknya pengulangan diketahui dengan pasti karena nilai awal (start) dan nilai akhir (end) sudah ditentukan diawal pengulangan. Bagaimana jika kita tidak tahu pasti harus berapa kali mengulang? Pengulangan dengan *While* merupakan jawaban dari permasalahan ini. Seperti halnya *For*, struktur pengulangan dengan *While* juga merupakan struktur yang didukung oleh hampir semua bahasa pemrograman namun dengan sintaks yang berbeda.



Gambar 5.19. *Flowchart* umum *While*.

Struktur *While* akan mengulang pernyataan pada badan loop sepanjang kodisi pada *While* bernilai benar. Dalam artian kita tidak perlu tahu pasti berapa kali diulang. Yang penting sepanjang kondisi pada *While* dipenuhi maka pernyataan pada badan loop akan diulang. *Flowchart* umum untuk struktur *While* dapat dilihat pada Gambar 5.19.

Pada Gambar 5.19., tampak bahwa simbol preparasi untuk pengulangan seperti pada For tidak digunakan lagi. Namun kita menggunakan simbol decision untuk mengendalikan pengulangan. Selain kondisi, biasanya pada

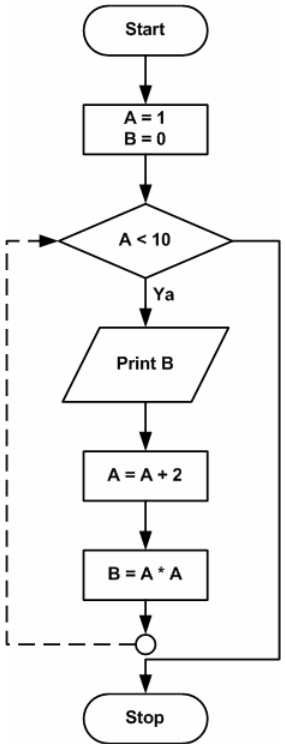
pengulangan *While* harus dilakukan inisialisasi variabel terlebih dahulu.

Contoh 5.22. Pengulangan dengan *While* untuk mencetak nilai tertentu.

Perhatikan *flowchart* pada Gambar 5.20. Bagaimanakah *output* dari *flowchart* tersebut?

Penyelesaian:

Perhatikan Gambar 5.20. bisakah kalian menentukan hasil dari *flowchart* tersebut? Perhatikan tahapan eksekusi *flowchart* berikut ini.



Gambar 5.20. *Flowchart* pengulangan dengan while untuk mencetak nilai tertentu.

1. Pada *flowchart* ini ada dua variabel yang kita gunakan yaitu A dan B. Kedua variabel tersebut kita inisialisasi nilai awalnya (A = 1 dan B = 0) sebelum proses loop terjadi. Variabel A adalah variabel *counter*.
2. Pada simbol decision, nilai A akan diperiksa apakah memenuhi kondisi (< 10). Jika Ya maka perintah berikutnya dieksekusi, jika tidak maka program akan berhenti. Pada awal eksekusi ini kondisi akan terpenuhi karena nilai A = 1.
3. Jalankan perintah Print B.
4. Nilai variabel A kemudian diganti dengan nilai A lama (1) ditambah 2. Sehingga nilai variabel A baru adalah 3. Sedangkan nilai variabel B = 9 (hasil perkalian A = 3).
5. Program akan berputar kembali untuk memeriksa apakah nilai variabel A masih lebih kecil dari 10. Pada kondisi ini nilai A = 3, sehingga kondisi masih terpenuhi. Kemudian langkah berulang ke langkah ke 3. Begitu seterusnya sampai nilai variabel A tidak lagi memenuhi syarat kurang dari 10. Sehingga output dari *flowchart* ini adalah : 0, 9, 25, 49, 81.

Seperti halnya pengulangan dengan *For*, pengulangan dengan *While* juga memungkinkan terjadinya pengulangan bersarang. Aturan dan cara yang dilakukan sama dengan pengulangan dengan *For*.

Pada beberapa bahasa pemrograman juga disediakan pengulangan dengan cara *Do ... Loop* dan *Repeat .. Until*. Kedua cara ini mirip dengan *While*, perbedaannya adalah letak dari kondisi. Pada *While* pemeriksaan kondisi diletakkan sebelum badan *loop*. Sedangkan *Do ... Loop* dan *Repeat ... Until*, pemeriksaan kondisi dilakukan setelah badan loop.

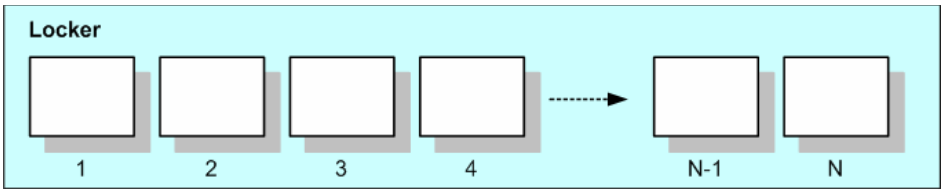
5.3. PENGELOLAAN ARRAY

Variabel array telah kita singgung di bagian depan, namun masih sangat terbatas. Pada bagian ini kita akan pelajari lebih detil tentang array.

5.3.1. Pengertian Array

Variabel-variabel yang kita gunakan selama ini adalah variable biasa yang memiliki sifat bahwa sebuah nama variable hanya dapat menyatakan sebuah nilai *numeric* atau *string* pada suatu saat. Apabila kita ingin memberi nilai yang baru pada variable tersebut maka nilai lama akan hilang tergantikan oleh nilai yang baru. Bagaimana apabila kita ingin menyimpan beberapa nilai/data dalam sebuah variable dengan nama yang sama, tetapi semua nilai tetap tersimpan? Solusi yang dapat dilakukan adalah dengan menggunakan indeks pada nama variable tersebut. Cara ini biasa disebut dengan array.

Array adalah struktur data yang menyimpan sekumpulan elemen yang bertipe sama, setiap elemen diakses langsung melalui indeksnya. Indeks array haruslah tipe data yang menyatakan keterurutan, misalnya integer atau string. Array dapat dianalogikan sebagai sebuah lemari atau locker yang memiliki sederetan kotak penyimpanan yang diberi nomor berurutan (lihat Gambar 5.21). Untuk menyimpan atau mengambil sesuatu dari kotak tertentu kita hanya cukup mengetahui nomor kotaknya saja.



Gambar 5.21. Lemari dengan banyak kotak laci di dalamnya

Pada variabel array, kita tidak hanya menentukan tipe datanya saja, tetapi juga jumlah elemen dari array tersebut atau dalam hal ini adalah batas atas indeksnya. Pada banyak bahasa pemrograman seperti C++, Visual Basic, dan beberapa yang lainnya, nilai indeks awal adalah 0 bukan 1. Cara menuliskan variabel array berbeda-beda tergantung bahasa pemrograman apa yang dipakai. Tetapi yang pasti tipe data harus disebutkan dan batas atas indeks harus

ditentukan. Untuk mengisi data pada array kita dapat langsung menentukan pada indeks berapa kita akan isikan demikian juga untuk memanggil atau menampilkan data dari array.

Contoh deklarasi, pengisian dan pemanggilan array adalah sebagai berikut.

Contoh 5.23. Penulisan array pada C++ dan Visual Basic.

Array pada C++

```
#include <iostream>
using namespace std;

int main() {

// Mendeklarasikan array A dengan 3 buah elemen bertipe int
  int A[3];

// Mengisikan nilai elemen array
A[0] = 5;
A[1] = 10;
A[2] = 20;

  // Menampilkan nilai elemen array
cout<<"Nilai elemen ke-1 = "<<A[0];
cout<<"Nilai elemen ke-2 = "<<A[1];
cout<<"Nilai elemen ke-3 = "<<A[2];

  return 0;
}
```

Array pada Visual Basic

```
'Mendeklarasikan array A dengan 3 buah elemen bertipe integer

Dim A (2) as Integer

'Mengisikan nilai elemen array
A(0) = 5;
A(1) = 10;
A(2) = 20;

'Menampilkan nilai elemen array
Print A(0);
Print A(1);
Print A(2);
```

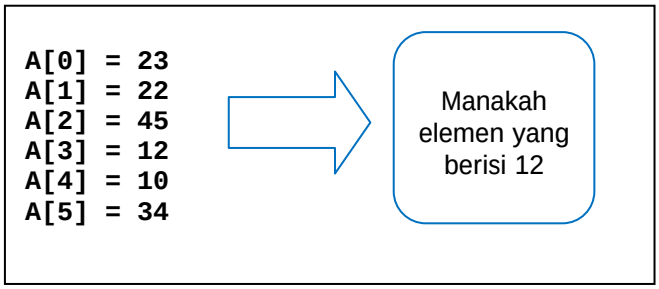
Perhatikan pada kedua kode di atas. Pada pendeklarasian variabel array nilai maksimal indeks adalah 2 tetapi jumlah elemennya ada 3 karena indeks dimulai dari 0 bukan dari 1.

5.3.2. Pencarian Data dalam Array

Salah satu permasalahan yang sering dijumpai dalam array adalah bagaimana mencari elemen tertentu dari array. Misalnya pada kasus loker pada Gambar 5.21 di atas tersedia 100 kotak. Kemudian kita diminta untuk mencari nomor kotak keberapa yang dimiliki oleh seorang siswa bernama “Rudi”. Contoh yang lain, misalkan ada banyak siswa dalam satu sekolah dan kita diminta

mencari data seorang siswa dengan nama tertentu atau alamat tertentu. Perhatikan contoh berikut.

Contoh 5.24. Pencarian pada array.



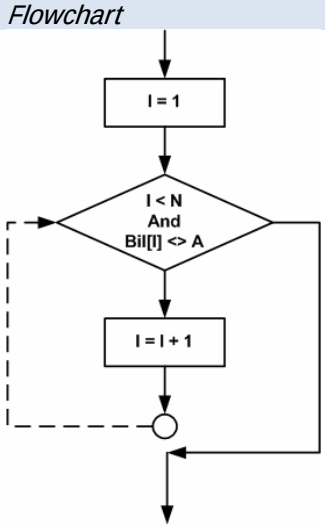
Pada contoh ini kita diminta mencari elemen yang berisi angka 12 dari sekumpulan elemen dalam array. Ada 6 elemen pada array tersebut. Menurut kalian bagaimanakah algoritma penyelesaiannya?

Cara yang paling umum dan paling mudah dilakukan adalah dengan cara pencarian berurutan (*linear search*). Pada masa lalu cara ini dianggap tidak efisien karena membutuhkan waktu lama. Namun dengan perkembangan komputer yang sangat cepat, waktu eksekusi algoritma ini tidak terlalu dipermasalahkan. Cara ini dilakukan dengan cara membandingkan isi dari elemen dengan apa yang kita cari. Satu per satu dimulai dari elemen yang paling awal.

Apabila kita terapkan pada Contoh 5.24, maka eksekusi program akan berlangsung berurutan sebagai berikut:

- Tetapkan bilangan yang ingin kita cari (yaitu 12)
- Ambil elemen paling awal (yaitu A[0]), bandingkan isi elemen tersebut (yaitu 23) dengan bilangan yang kita cari. Jika sama maka stop.
- Jika tidak maka lanjutkan dengan elemen berikutnya (yaitu A[1]), bandingkan isi elemen tersebut dengan bilangan yang kita cari. Jika sama maka stop.
- Jika tidak maka lanjutkan dengan elemen berikutnya. Dan seterusnya sampai dijumpai elemen yang berisi sama dengan bilangan yang kita cari.

Contoh 5.24 akan memberikan hasil elemen A[3] yang memiliki isi 12. Apabila digambarkan dalam bentuk *flowchart* maka akan tampak seperti pada Gambar 5.22.

Flowchart	Keterangan
	- I adalah variabel counter sekaligus indeks - N adalah batas atas dari indeks - Bil[I] adalah nama variabel array yang berisi bilangan - A adalah bilangan yang kita cari

Gambar 5.22. *Flowchart* untuk pencarian bilangan.

Pada *flowchart* di Gambar 5.22, kita menggunakan pengulangan model *While*. Kondisi yang harus dipenuhi disini ada dua, yaitu I < N dan Bil[I] <> A. Arti dari kondisi ini adalah jika nilai indeks I kurang dari batas atas indeks dan isi dari Bil[I] tidak sama dengan bilangan yang kita cari, maka pencarian akan diteruskan pada indeks yang lebih tinggi. Selama kondisi ini dipenuhi maka pencarian akan terus dilakukan. Perhatikan bahwa di sini kita menggunakan "dan" yang artinya kedua kondisi harus dipenuhi agar dianggap benar. Pencarian akan hanya akan berhenti jika salah satu kondisi atau kedua kondisi tidak dipenuhi lagi. Sehingga misalnya Bil[I] mempunyai isi yang sama dengan A maka pencarian akan dihentikan karena kondisi pada *While* sudah tidak dipenuhi lagi.

5.3.3. Pengurutan Data pada Array

Permasalahan lain dalam array yang juga banyak digunakan adalah bagaimana mengurutkan elemen-elemen dari variabel array tersebut. Perhatikan kembali Contoh 5.24. Pada contoh tersebut terlihat bahwa isi elemen-elemen dari array tidak dalam posisi berurutan. Bagaimanakah caranya agar isi elemen-elemen tersebut terurut dari besar ke kecil atau sebaliknya?

Ada beberapa algoritma yang dapat digunakan untuk mengurutkan sekumpulan bilangan, antara lain *bubble sort, selection sort, shell sort, quick sort*, dan lain-lain. Pada buku ini kita akan membahas satu algoritma yaitu *bubble sort*. Meskipun kinerjanya tidak sebaik algoritma yang lain, algoritma ini mudah dimengerti dan banyak digunakan. Perhatikan contoh berikut.

Contoh 5.24. Pengurutan dengan *bubble sort*.

Misalkan sebuah variabel array dengan nama Bil yang terdiri dari 5 elemen yang masing-masing berisi bilangan "5 1 4 2 8". Urutkan dari mulai nilai terkecil sampai ke yang paling besar.

Penyelesaian:

Kita akan menggunakan metode bubble sort untuk mengurutkan array ini. Bubble sort dilakukan dengan cara membandingkan dua bilangan yang berurutan letaknya. Jika urutan letaknya benar maka dilanjutkan dengan membandingkan dua bilangan berikutnya. Jika tidak maka tukar letak dari dua bilangan tersebut.

Marilah kita terapkan algoritma ini. Perhatikan tabel array berikut. Kondisi awal adalah pada posisi J = 0. Pertama kita bandingkan antara Bil[0] dengan Bil[1]. Bil[0] = 5 sedangkan Bil[1] = 1. Berdasarkan aturan bubble sort, isi dari Bil[0] tidak sesuai letaknya karena lebih besar dari isi Bil[1]. Sehingga kita perlu menukar isi dari dua elemen array ini. Sehingga Bil[0] = 1 dan Bil[1] = 5 (perhatikan baris pada J = 1). Langkah berikutnya kita membandingkan Bil[1] dengan Bil[2]. Bil[1] = 5 dan Bil[2] = 4, sehingga kembali kita harus menukar isi dari elemen ini (perhatikan baris J = 2). Hal ini terus dilakukan sampai pada perbandingan Bil[3] dengan Bil[4].

J	Bil[0]	Bil[1]	Bil[2]	Bil[3]	Bil[4]
0	5	↔ 1	4	2	8
1	1	5	↔ 4	2	8
2	1	4	5	↔ 2	8
3	1	4	2	5	↔ 8
4	1	4	2	5	8

Pada posisi akhir dari tabel di atas, kita lihat bahwa bilangan belum terurut sepenuhnya. Karena kita baru menggunakan satu kali putaran dengan Bil[0] sebagai patokan. Kita akan lakukan perbandingan lagi, namun dengan Bil[1] sebagai patokan. Hal ini karena Bil[0] pasti sudah berisi bilangan paling kecil. Sehingga tabel baru kita buat seperti berikut.

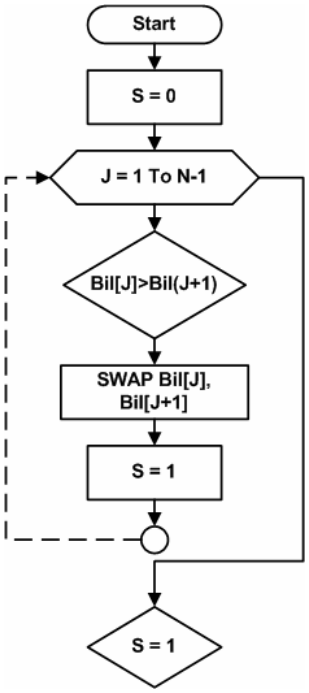
J	Bil[0]	Bil[1]	Bil[2]	Bil[3]	Bil[4]
1	1	4	↔ 2	5	8
2	1	2	4	↔ 5	8
3	1	2	4	5	↔ 8
4	1	2	4	5	8

Pada posisi tabel di atas, sebenarnya urutan bilangan sudah benar, tapi algoritma belum berhenti karena algoritma belum memeriksa putaran yang berikutnya. Sehingga diperlukan dua putaran lagi dengan dengan dasar masing-masing pembanding adalah Bil[2] dan Bil[3]. Kedua tabel tersebut adalah sebagai berikut.

J	Bil[0]	Bil[1]	Bil[2]	Bil[3]	Bil[4]
2	1	2	4	5	8
3	1	2	4	5	8
4	1	2	4	5	8

J	Bil[0]	Bil[1]	Bil[2]	Bil[3]	Bil[4]
3	1	2	4	5	8
4	1	2	4	5	8

Kalau digambarkan dalam bentuk *flowchart* akan tampak seperti pada Gambar 5.23.



Gambar 5.23. *Flowchart* untuk pengurutan bilangan.

5.4. OPERASI FILE

File seringkali digunakan untuk menyimpan data agar data tidak hilang. Data atau yang ada dan dihasilkan pada program akan hilang ketika program diakhiri, sehingga file digunakan untuk menyimpan data tersebut. Ada dua jenis file yaitu file program dan file data. File program berisi kode-kode program sedangkan file data hanya berisi data. File data terdiri dari dua jenis yaitu file data berurutan (*sequential data file*) dan file data acak (*random-access data file*). Perbedaan utama dari kedua jenis file data ini adalah dapat dilihat pada tabel berikut.

File data berurutan	File data acak
- *Record* atau baris data harus dibaca berurutan mulai dari yang pertama - Panjang *field* untuk setiap *record* tidak perlu sama - Pengubahan serta penambahan *record* tertentu sukar dilakukan	- *Record* tidak perlu dibaca berurutan - Panjang *field* untuk setiap *record* harus sama - Pengubahan serta penambahan *record* lebih mudah dilakukan

5.4.1. Algoritma Penulisan Data pada File

Algoritma yang digunakan untuk penulisan data untuk file data berurutan maupun acak secara prinsip sama, hanya modusnya yang berbeda. Berikut ini adalah algoritma penulisan data dalam SE.

```
Open "modus", <buffer number>, "nama file data"
Write <record number>, field 1, field 2, .. field n
Close buffer number
```

Modus O menunjukkan file ini dibuka untuk ditulisi.

Contoh 5.25. Contoh penerapan algoritma penulisan data.

Misalkan kita punya file data dengan nama "siswa.dat" yang *field*-nya adalah nama siswa, alamat, nomor telepon. Maka untuk menuliskan data adalah sebagai berikut.

```
Open "O", #1, "siswa.dat"
Write #1, <nama>, <alamat>, <no.telepon>
Close #1
```

Notasi #1 menunjukkan siswa.dat akan ditempatkan dalam *buffer* no 1. Notasi ini harus sama digunakan di seluruh progam di atas. Artinya kalau kita menempatkan suatu file dengan nomor *buffer* #1 maka ketika membuka, menulis, membaca dan menutup harus menggunakan notasi tersebut. Demikian juga bila kita menempatkan pada *buffer* no #2.

5.4.2. Algoritma Pembacaan Data pada File

Algoritma membaca data algoritmanya hampir sama dengan menuliskan data, tetapi modus yang digunakan tidak O tetapi I. I adalah input yang berarti file data dibuka untuk dibaca datanya sebagai input. Berikut ini algoritmanya dalam SE.

```
Open "modus", <buffer number>, "nama file data"
While not EOF:
       Input <record number>, field 1, field 2, ..
      field n
       Print field 1, field 2, .. field n
End while
Close buffer number
```

Pernyataan *While Not EOF* digunakan untuk memeriksa apakah sudah ada pada baris terakhir dari data. Jika belum maka baris-baris data akan dibaca dan dicetak sampai baris terakhir. Pernyataan input digunakan untuk mengambil data dari file untuk dimuat ke dalam program. Sedangkan pernyataan *print* digunakan untuk mencetak data ke layar komputer.

Contoh 5.26. Contoh penerapan algoritma penulisan data.

File data dengan nama "siswa.dat" seperti pada contoh 5.25 yang *field*-nya adalah nama siswa, alamat, nomor telepon. Maka untuk membaca data adalah sebagai berikut.

```
Open "I", #2, "siswa.dat"
While not EOF:
       Input #2, <nama>, <alamat>, <no.telepon>
       Print <nama>, <alamat>, <no.telepon>
End while
Close buffer number
```

5.5. Ringkasan

- Variabel adalah tempat dimana kita dapat mengisi atau mengosongkan nilainya dan memanggil kembali apabila dibutuhkan. Setiap variabel akan mempunyai nama (*identifier*) dan nilai.
- Konstanta adalah variabel yang nilai datanya bersifat tetap dan tidak bisa diubah.
- Tipe data adalah jenis data yang dapat diolah oleh komputer untuk memenuhi kebutuhan dalam pemrograman komputer.
- Tipe data dapat dikelompokkan menjadi tipe data primitive dan tipe data composite. Tipe data primitive terdiri dari *numeric, character*, dan *bolean*. Sedangkan tipe data composite terdiri dari *array, record/struct, image, date time, subrange*, enumerasi, obyek dan *variant*.
- Algoritma adalah urutan langkah-langkah logis penyelesaian masalah yang disusun secara sistematis. Algoritma harus benar dan harus berhenti. Setelah berhenti, algoritma memberikan hasil yang benar.
- Algoritma dapat ditulis dengan cara *Structured English, Pseudocode* dan *Flowchart*.
- Struktur berurutan terdiri satu atau lebih instruksi. Tiap instruksi dikerjakan secara berurutan sesuai dengan urutan penulisannya.
- Pada struktur percabangan, program akan berpindah urutan pelaksanaan jika suatu kondisi yang disyaratkan dipenuhi.
- Struktur pengulangan terdiri dari dari kondisi pengulangan dan badan pengulangan dan dapat dilakukan dengan *For* dan *While*.

- Array adalah struktur data yang menyimpan sekumpulan elemen yang bertipe sama, setiap elemen diakses langsung melalui indeksnya. Operasi pencarian pada array dapat dilakukan dengan cara linear search sedangkan pengurutan dengan metode *bubble sort*.
- File data ada yang bersifat urut dan ada yang acak. Metode pembacaan dan penulisan dibedakan dari modusnya.

5.6. Soal-Soal Latihan

1. Tentukan salah atau benar pada nama-nama variabel berikut ini. Jika salah cobalah berikan alasan.

   a. nama.guru
   b. NamaGuru
   c. 2x
   d. harga/buku
   e. hargaPerBuku

2. Tentukan tipe data yang cocok untuk hal-hal berikut ini (perhatikan ini bukan nama variabel) dan jelaskan alasannya.

   a. Jumlah murid
   b. Berat badan
   c. Tinggi badan
   d. Nama siswa
   e. Tempat lahir
   f. Tanggal lahir

3. Lihat kembali Contoh 5.3.

   a. Jika pada kode program A semua variabel (x, y, z) dideklarasikan bertipe data int, berapakah nilai z?

   b. Jika pada kode program A semua variabel (x, y, z) dideklarasikan bertipe data float, berapakah nilai z?

   c. Jika pada kode program B variabel x juga dideklarasikan bertipe data float, berapakah nilai z?

4. Ada dua gelas A dan B, gelas A berisi larutan berwarna merah, gelas B berisi larutan berwarna kuning. Pertukarkan isi dari kedua gelas itu sedemikian rupa sehingga gelas A berisi larutan kuning dan gelas B berisi larutan merah. Buatlah algoritma penyelesaiannya dengan menggunakan SI (*Structured Indonesia*).

5. Variabel A = 6, Variabel B = 10. Buatlah *flowchart* untuk menukar nilai variabel A dan B sehingga variabel A = 10 dan variabel B = 6.

6. PT. Sandang Nyaman bermaksud menggunakan komputer untuk menghitung upah mingguan pegawainya. Data yang diperlukan adalah nama pegawai dan jumlah jam kerja selama seminggu. Upah per jam ditetapkan Rp. 4500,-. Buatlah *flowchart* untuk masalah ini jika output