Cara Mudah

Mempelajari

Pemrograman C

&

Implementasinya

Membahas:

- Semua konsep dan elemen dasar bahasa C
- Struktur dan union
- Konsep pointer, teknik alokasi dan dealokasi memori
- Array dan penggunaannya
- Linked list (stack dan queue)
- Operasi file
- Preprocessor directive
- Pemrograman port (paralel dan serial)

I Made Joni
Budi Raharjo

Saya persembahkan buku ini untuk Ibu Ni Wayan Ranis dan kakak-kakak tercinta. Ucapan terima kasih saya tujukan kepada rekan-rekan dosen di Jurusan Fisika Universitas Padjadjaran: Prof. Dr. Rustam E. Siregar; Dr. Qomarrudin; Dr. Bernard YT; Camellia Panatarani, D.Eng; Darmawan Hidayat, MT; Bambang Mukti Wibawa, MS; dan yang lainnya – atas motivasi yang telah diberikan.

-- I Made Joni

Saya persembahkan buku ini untuk kedua orang tua tercinta, terima kasih atas semua doa, didikan, dan kasih sayangnya selama ini; juga untuk kakak-kakak, dan adikku -- Mira -- atas dukungan dan kasih sayangnya. Terima kasih juga saya ucapkan untuk Bapak Totok Triwibowo atas arahan dan saran-saran yang telah diberikan, rekan-rekan di PT. Sigma Delta Duta Nusantara (SDDN) dan PT. Telekomunikasi Indonesia.

-- Budi Raharjo

Kata Pengantar

Puji syukur kami panjatkan kepada Tuhan YME atas karunia dan berkah-Nya sehingga penulis berdua dapat menyelesaikan penyusunan buku ini.

Bahasa C merupakan bahasa pemrograman yang sudah tidak diragukan lagi kehandalannya dan banyak digunakan untuk membuat program-program dalam berbagai bidang, termasuk pembuatan compiler dan sistem operasi. Sampai saat ini, C masih tetap menjadi bahasa populer dan berwibawa dalam kancah pemrograman. Sejauh ini, C juga telah menjadi inspirasi bagi kelahiran bahasa-bahasa pemrograman baru, seperti C++, Java, dan juga yang lainnya; sehingga dari sisi sintak kontrol programnya, ketiga bahasa ini bisa dikatakan sama. Bahasa pemrograman C sangatlah fleksibel dan portabel, sehingga dapat ditempatkan dan dijalankan di dalam beragam sistem operasi. Pada umumnya, C banyak digunakan untuk melakukan interfacing antar perangkat keras (*hardware*) agar dapat berkomunikasi satu sama lainnya.

Dalam buku ini akan dibahas konsep-konsep dasar yang mutlak diperlukan dalam pemrograman menggunakan bahasa C sehingga Anda dapat mengimplementasikannya langsung ke dalam kasus-kasus pemrograman yang Anda hadapi. Sesuai dengan kata penciptanya -- Dennis M. Ritchie -- dalam bukunya "*The C Programming Language*" yang disusun bersama rekan kerjanya -- Brian W. Kernighan -- menyebutkan bahwa bahasa C bukan merupakan bahasa besar sehingga tidaklah diperlukan buku yang tebal untuk membahas semua elemen dari bahasa C. Dengan banyaknya contoh program yang ada, mudah-mudahan buku yang relatif tipis ini dapat menjadi referensi yang membantu Anda dalam mempelajari pemrograman C secara mudah dan cepat. Selain itu, pada bagian akhir buku ini disertakan pula beberapa konsep dasar dan implementasi C untuk melakukan pemrograman port.

Terakhir, penulis berdua menghaturkan banyak terima kasih kepada Pak Benie Ilman beserta rekan-rekan di penerbit Informatika yang telah banyak membantu dalam proses pencetakan dan penerbitan buku ini.

Bandung, 10 Juni 2006
Penulis

Daftar Isi

Kata Pengantar

Daftar Isi

Bab 1. Pengenalan Bahasa C

1.1. Pendahuluan
1.2. Apa Itu Program Komputer?
1.3. Apa Itu Kompiler?
1.4. Apa Itu Bahasa Pemrograman?
1.5. Mengapa Menggunakan Bahasa C?
1.6. Sejarah Singkat Lahirnya Bahasa C
1.7. Kerangka Program dalam Bahasa C
1.8. File Header (*.h)
1.9. Proses Pembentukan Program dalam Bahasa C
    1.9.1. Menuliskan Kode Program
    1.9.2. Melakukan Kompilasi Kode Program
    1.9.3. Proses *Linking*
1.10. Mengenal Fungsi *printf()* dan *scanf()*
1.11. Membuat Program di dalam Sistem Operasi Windows dan Linux
    1.11.1. Kompilasi dan Eksekusi Program di dalam Windows
    1.11.2. Kompilasi dan Eksekusi Program di dalam Linux

Bab 2. Komentar, Variabel dan Tipe Data

2.1. Pendahuluan
2.2. Komentar Program
    2.2.1. Komentar Sisipan
    2.2.2. Komentar Bersarang
2.3. Variabel
    2.3.1. Batasan Penamaan Variabel
    2.3.2. Inisialisasi Variabel
    2.3.3. Lingkup Variabel
        2.3.3.1. Variabel Global
        2.3.3.2. Variabel Lokal
    2.3.4. Jenis Variabel
        2.3.4.1. Variabel Otomatis
        2.3.4.2. Variabel Statis
        2.3.4.3. Variabel Eksternal
        2.3.4.4. Variabel Register
2.4. Konstanta

2.5. Tipe Data
2.5.1 Tipe Data Dasar
2.5.1.1. Tipe Bilangan Bulat
2.5.1.2. Tipe Bilangan Riil
2.5.1.3. Tipe Karakter dan String
2.5.1.4. Tipe Logika
2.5.2. Tipe Data Bentukan
2.5.3. Enumerasi

Bab 3. Operator

3.1. Pendahuluan
3.2. Operator *Assignment*
3.3. Operator *Unary*
3.3.1. *Increment*
3.3.1.1. *Pre-Increment*
3.3.1.2. *Post-Increment*
3.3.2. *Decrement*
3.4. Operator *Binary*
3.4.1. Operator Aritmetika
3.4.2. Operator Logika
3.4.2.1. Operator && (AND)
3.4.2.2. Operator || (OR)
3.4.2.3. Operator ! (NOT)
3.4.3. Operator Relasional
3.4.4. Operator Bitwise
3.4.4.1. Operator ^ (*Exclusive* OR)
3.4.4.2. Operator >> (*Shift Right*)
3.4.4.3. Operator << (*Shift Left*)
3.5. Operator *Ternary*

Bab 4. Kontrol Program

4.1. Pendahuluan
4.2. Pemilihan
4.2.1. Statemen *if*
4.2.1.1. Satu Kasus
4.2.1.2. Dua Kasus
4.2.1.3. Lebih dari Dua Kasus
4.2.2. Statemen *switch*
4.3. Pengulangan
4.3.1. Struktur *for*
4.3.1.1. Melakukan Beberapa Inisialisasi dalam Struktur *for*
4.3.1.2. Struktur *for* Bersarang
4.3.2. Struktur *while*
4.3.3. Struktur *do-while*
4.4. Statemen Peloncatan
4.4.1. Menggunakan Keyword *break*
4.4.2. Menggunakan Keyword *continue*
4.4.3. Menggunakan Keyword *goto*

4.4.4. Menggunakan Fungsi *exit()*

Bab 5. Fungsi

5.1. Pendahuluan
5.2. Apa Nilai yang dikembalikan Oleh Fungsi *main()*?
5.3. Fungsi Tanpa Nilai Balik
5.4. Fungsi dengan Nilai Balik
5.5. Fungsi dengan Parameter
    5.5.1. Jenis Parameter
    5.5.2. Melewatkan Parameter Berdasarkan Nilai (*Pass By Value*)
    5.5.3. Melewatkan Parameter Berdasarkan Alamat (*Pass By Reference*)
5.6. Melewatkan Parameter *argc* dan *argv* ke dalam Fungsi *main()*
5.7. Membuat Prototipe Fungsi
5.8. Rekursi
    5.8.1. Menentukan Nilai Faktorial
    5.8.2. Menentukan Nilai Perpangkatan
    5.8.3. Konversi Bilangan Desimal ke Bilangan Biner
    5.8.4. Konversi Bilangan Desimal ke Bilangan Heksadesimal

Bab 6. Array dan String

6.1. Pendahuluan
6.2. Apa Itu Array?
6.3. Mengapa Harus Menggunakan Array?
6.4. Inisialisasi Array
6.5. Array Konstan
6.6. Array Sebagai Parameter
6.7. Array Sebagai Tipe Data Bentukan
6.8. Pencarian pada Elemen Array
6.9. Pengurutan pada Elemen Array
    6.9.1. Menggunakan Metode Gelembung (*Bubble Sort*)
    6.9.2. Menggunakan Metode Maksimum/Minumun (*Maximum/Minimum Sort*)
6.10. Array Multidimensi
    6.10.1. Array Dua Dimensi
    6.10.2. Array Tiga Dimensi
6.11. String (Array dari Karakter)
6.12. Menampilkan String ke Layar
6.13. Membaca String dari Keyboard
6.14. Manipulasi String
    6.14.1. Menggabungkan String
    6.14.2. Menentukan Panjang String
    6.14.3. Menyalin String
    6.14.4. Membandingkan String
    6.14.5. Melakukan Pencaraian String
    6.14.6. Melakukan Konversi String

Bab 7. Pointer

7.1. Pendahuluan

7.2. Apa Itu Memori Komputer?
7.3. Apa Itu Pointer?
7.4. Mendeklarasikan Tanpa Tipe
7.5. Pointer dan Array
7.6. Pointer dan Fungsi
    7.6.1. Pointer sebagai Parameter Fungsi
    7.6.2. Pointer ke Fungsi
7.7. Pointer NULL
7.8. Pointer ke Pointer (*Multiple Indirection*)
7.9. Konstanta pada Pointer
    7.9.1. Keyword *const* Sebelum Tipe Data
    7.9.2. Keyword *const* Setelah Tipe Data
    7.9.3. Keyword *const* Sebelum dan Setelah Tipe Data
7.10. Mengalokasikan Memori
    7.10.1. Menggunakan Fungsi *malloc()*
    7.10.2. Menggunakan Fungsi *calloc()*
    7.10.3. Menggunakan Fungsi *realloc()*
7.11. Mendealokasikan Memori

7.12. **Memory Leak**

Bab 8. Struktur dan Union

8.1. Pendahuluan
8.2. Dasar-Dasar Struktur
    8.2.1. Mendefinisikan Struktur
    8.2.2. Inisialisasi Struktur
    8.2.3. Ukuran Strukur dalam Memori
    8.2.4. Mendefinisikan Struktur yang Berisi Struktur Lain
8.3. Struktur sebagai Tipe Data Bentukan
8.4. Struktur dan Fungsi
    8.4.1. Struktur sebagai Parameter Fungsi
    8.4.2. Struktur sebagai Nilai Kembalian Fungsi
8.5. Struktur dan Array
    8.5.1. Struktur dari Array
    8.5.2. Array dari Struktur
8.6. Struktur dan Pointer
    8.6.1. Struktur yang Berisi Pointer
    8.6.2. Pointer ke Struktur
8.7. Union

Bab 9. Linked List

9.1. Pendahuluan
9.2. *Stack* (Tumpukan)
9.3. *Queue* (Antrian)
9.4. *Linked List* dengan Dua Buah Pointer
9.5. Mengimplementasikan *Lisked List* dengan Rekursi
9.6. *Binary Tree* (Pohon Biner)

Bab 10. Operasi File

10.1. Pendahuluan
10.2. Membuka File
10.3. Menutup File
10.4. Membaca Data dari File
    10.4.1. Fungsi getc()
    10.4.2. Fungsi fgets()
    10.4.3. Fungsi fscanf()
10.5. Menulis Data ke dalam File
    10.5.1. Fungsi putc()
    10.5.2. Fungsi fputs()
    10.5.3. Fungsi fprintf()
10.6. Mendeteksi EOF (*End Of File*)
10.7. Manajemen File
    10.7.1. Menyalin File
    10.7.2. Mengubah Nama File
    10.7.3. Menghapus File
10.8. File Temporari

Bab 11. Preprosesor Directive

11.1. Pendahuluan
11.2. Directive #include
11.3. Directive #define
11.4. Pengkondisian dalam Proses Kompilasi
    11.4.1. Directive #if dan #endif
    11.4.2. Directive #else dan #elif
    11.4.3. Directive #ifdef
    11.4.4. Directive #ifndef
11.5. Directive #undef
11.6. Directive #pragma
11.7. Directive #line
11.8. Directive #error
11.9. Makro yang Telah Didefinisikan dalam Bahasa C
    11.9.1. Makro __FILE__
    11.9.2. Makro __LINE__
    11.9.3. Makro __DATE__
    11.9.4. Makro __TIME__
    11.9.5. Makro __STDC__

Bab 12. Implementasi C untuk Pemrograman Paralel dan Serial

12.1. Pendahuluan
12.2. Input/Output (I/O) PC
    12.2.1. Mengenal Transfer Data pada Port Paralel
    12.2.2. Standar Komunikasi Paralel menurut IEEE 1284
    12.2.3. Pengalamatan Port Paralel LPT
12.3. Aplikasi Pemrograman Port C untuk Tampilan 8 Buah Led
12.4. Aplikasi Pemrograman Port C untuk Musik

12.5. Aplikasi Pemrograman Port C untuk Menggerakkan Motor DC
12.5.1. Motor DC dengan Koil Medan
12.5.1.1. Motor DC Jenis Seri
12.5.1.2. Motor DC Jenis Shunt
12.5.1.3. Motor DC Jenis Gabungan
12.5.2. Motor DC Magnet Permanen
12.5.3. Pengaturan Gerak Motor DC
12.5.3.1. Driver Motor DC
12.5.3.2. Sumber Tegangan
12.5.3.3. Pemrograman Port Paralel untuk Pengaturan Motor DC
12.6. Aplikasi Pemrograman Port C untuk Menggerakkan Motor Stepper
12.7. Aplikasi Pemrograman Port C untuk Tampilan LCD Dua Baris 16 Karakter
12.7.1. Gambaran Umum Pengoperasian LCD
12.7.2. Inisialisasi LCD
12.7.3. Pengiriman Alamat Tujuan Penulisan Karakter
12.7.4. Mengirim Karakter ASCII
12.7.5. Antarmuka LCD ke Port Paralel LPT PC
12.8. Port Paralel Dwi-Arah
12.8.1. Menggunakan Port Paralel untuk Input 8 Bit
12.8.2. Mode Nibble
12.9. Pemrograman Port Paralel dan Serial Menggunakan Fungsi DOS dan BIOS
12.9.1. Pemrograman Port Paralel Menggunakan Fungsi DOS dan BIOS
12.9.2. Pemrograman Port Serial Menggunakan Fungsi DOS dan BIOS

Lampiran A – Standard Library

Lampiran B – Membuat File Header (.h)*

Bab

1

Pengenalan Bahasa C

1.1. Pendahuluan

C merupakan bahasa pemrograman yang berkekuatan tinggi (*powerful*) dan fleksibel yang telah banyak digunakan oleh para programmer profesional untuk mengembangkan program-program yang sangat bervariasi dalam berbagai bidang. Namun sebelum Anda mempelajari lebih jauh mengenai bahasa C beserta implementasinya, sebaiknya Anda mengetahui terlebih dahulu komponen dan pengetahuan dasar tentang ilmu pemrograman komputer.

1.2. Apa Itu Program Komputer?

Bagi Anda yang merupakan pemula di dunia pemrograman komputer, mungkin Anda akan bertanya apa sebenarnya yang dimaksud dengan program komputer? Program komputer tidak lain adalah suatu perangkat lunak (*software*) yang digunakan untuk keperluan-keperluan aplikatif tertentu di berbagai bidang, baik di lingkungan perusahaan, pendidikan ataupun yang lainnya. Perangkat lunak tersebut sebenarnya merupakan suatu runtunan kode-kode program yang ditulis dengan salah satu bahasa pemrograman tertentu dan telah dikompilasi melalui kompilator yang sesuai dengan bahasa pemrograman tersebut. Dengan demikian, untuk membuat sebuah perangkat lunak atau yang lazim dikenal dengan sebutan ‘program’, kita tentu harus memiliki keahlian dan menguasai salah satu bahasa pemrograman tertentu.

1.3. Apa Itu Kompilator?

Kompilator (*compiler*) dapat diartikan sebagai suatu penerjemah, artinya kumpulan kode program yang ditulis dalam suatu bahasa pemrograman tertentu akan diterjemahkan oleh kompilator ke dalam bahasa assembly; yang selanjutnya akan diterjemahkan lagi menjadi kode objek sehingga perintah-perintahnya akan dikenali oleh komputer (dalam hal ini *mesin*). Dengan demikian komputer akan dapat merespon permintaan kita dengan melakukan sesuatu sesuai dengan apa yang kita perintahkan.

Tidak semua bahasa pemrograman dapat dikompilasi di dalam satu kompilator tertentu, artinya sebuah kompilator hanya akan mengenali bahasa-bahasa tertentu saja sesuai dengan yang telah dibuat oleh pencipta kompilator tersebut. Sebagai contoh, Anda tidak dapat melakukan kompilasi program yang ditulis dalam bahasa C di dalam kompilator Pascal, begitupun sebaliknya. Namun sebagai pengetahuan bagi Anda bahwa untuk semua kompilator C++, selain digunakan untuk mengkompilasi program yang ditulis dalam bahasa C++, kompilator tersebut juga dapat digunakan untuk melakukan kompilasi terhadap kode-kode program yang ditulis dalam bahasa C.

1.4. Apa Itu Bahasa Pemrograman?

Pada sub bab sebelumnya Anda telah banyak membaca istilah ‘bahasa pemrograman’. Sebenarnya apa yang dimaksud dengan bahasa pemrograman? Bahasa pemrograman adalah suatu kumpulan kata (perintah) yang siap digunakan untuk menulis suatu kode program sehingga kode-kode program yang kita tulis tersebut akan dapat dikenali oleh kompilator yang sesuai. Kata-kata tersebut dalam dunia pemrograman sering dikenal dengan istilah *keyword* (terkadang disebut *reserved word*). Untuk mempelajari salah satu bahasa pemrograman tertentu, tentunya kita tidak hanya menghafal semua *keyword* (kata kunci) yang ada di dalamnya, namun kita juga perlu untuk memahami fungsi dan aturan penggunaannya.

Sekarang ini banyak sekali bahasa pemrograman yang dapat digunakan untuk mengembangkan suatu perangkat lunak, diantaranya bahasa C, C++, Pascal, Java dan banyak lagi lainnya. Bahkan untuk pembuatan pemrograman visual pun, telah banyak tersedia perangkat lunak seperti C++Builder, Delphi, JBuilder, Visual C++ dan yang lainnya.

1.5. Mengapa Menggunakan Bahasa C?

Seperti yang telah dikatakan sebelumnya bahwa sekarang banyak sekali terdapat bahasa permograman tingkat tinggi (*high level language*) seperti Pascal, BASIC, COBOL dan lainnya. Walaupun demikian, sebagian besar dari para programmer profesional masih tetap memilih bahasa C sebagai bahasa yang lebih unggul, berikut ini alasan-alasannya:

- Bahasa C merupakan bahasa yang *powerful* dan fleksibel yang telah terbukti dapat menyelesaikan program-program besar seperti pembuatan sistem operasi, pengolah kata, pengolahan gambar (seperti pembuatan *game*) dan juga pembuatan kompilator untuk bahasa pemrograman baru.

- Bahasa C merupakan bahasa yang portabel sehingga dapat dijalankan di beberapa sistem operasi yang berbeda. Sebagai contoh program yang kita tulis dalam sistem operasi Windows dapat kita kompilasi di dalam sistem operasi Linux dengan sedikit ataupun tanpa perubahan sama sekali.

- Bahasa C merupakan bahasa yang sudah populer dan banyak digunakan oleh para programmer berpengalaman sehingga kemungkinan besar *library* (pustaka) dan aksesoris program lainnya yang diperlukan dalam pemrograman telah banyak disediakan oleh pihak luar/lain dan dapat diperoleh dengan mudah.

- Bahasa C merupakan bahasa yang bersifat modular, yaitu yang tersusun atas rutin-rutin tertentu yang dinamakan dengan fungsi (*function*) dan fungsi-fungsi tersebut dapat digunakan kembali untuk pembuatan program-program lainnya tanpa harus menulis ulang implementasinya.

- Bahasa C merupakan bahasa tingkat menengah (*middle level language*) sehingga mudah untuk melakukan *interfacing* (pembuatan program antar muka) ke perangkat keras (*hardware*).

1.6. Sejarah Singkat Lahirnya Bahasa C

Sebelum melangkah lebih jauh ke pembentukan program dengan menggunakan bahasa C, ada baiknya kita menengok ke belakang terlebih dahulu untuk mengetahui sejarah terbentuknya bahasa C serta kepopulerannya di bidang industri perangkat lunak.

Lahirnya bahasa pemrograman diawali oleh terbentuknya bahasa assembly yang dikembangkan oleh IBM dalam tahun 1956-1963. Bahasa ini termasuk dalam bahasa tingkat rendah (*low level language*). Pada tahun 1957, sebuah tim yang dipimpin oleh John W. Backus berhasil mengembangkan sebuah bahasa pemrograman baru yang lebih diarahkan untuk proses analisa numerik. Bahasa pemrograman tersebut dinamai dengan bahasa FORTRAN (*Formula Translation*). Setahun kemudian, yaitu pada tahun 1958, para ilmuwan komputer dari Eropa dan Amerika yang tergabung dalam sebuah komite menciptakan bahasa pemrograman baru yang lebih bersifat struktural dan dinamakan dengan bahasa ALGOL (*Algorithmic Language*). Kemudian pada tahun 1964, IBM kembali menciptakan bahasa pemrograman baru dengan nama PL/I (*Programming Language 1*) yang lebih ditujukan untuk keperluan bisnis dan penelitian.

Tahun 1969 laboratorium Bell AT&T di Murray, New Jersey menggunakan bahasa assembly untuk mengembangkan sistem operasi Unix yang bertujuan untuk membuat program antar muka yang bersifat *programmer friendly*. Setelah Unix berjalan, lahirlah bahasa pemrograman baru yang ditulis oleh Martin Richards dengan nama bahasa BCPL (*Basic Combined Programming Language*). Kemudian pada tahun 1970, seorang pengembang sistem dari laboratorium tersebut yang bernama Ken Thompson membuat bahasa B yang akan digunakan untuk menulis ulang sistem operasi Unix. Nama ‘B’ ini konon diambil dari huruf pertama dalam kata BCPL. Karena alasan bahwa bahasa B masih terkesan lambat, maka pada tahun 1971 seorang pengembang sistem bernama Dennis Ritchie, yang juga bekerja di laboratorium yang sama, menciptakan bahasa baru dengan nama C yang bertujuan untuk menulis ulang dan menutupi kelemahan-kelemahan yang ada pada sistem operasi Unix sebelumnya. Menurut sumber yang ada, nama ‘C’ ini juga konon diambil dari huruf kedua dalam kata BCPL.

Sejak itu bahasa C terus digunakan untuk memelihara sitem operasi Unix, Sampai akhirnya pada tahun 90-an, bahasa C ini digunakan untuk mengembangkan sistem operasi Windows dan sekarang ini digunakan untuk mengembangkan sistem operasi Linux. Selain untuk menulis program yang merupakan *embedded system*, di kalangan industri hiburan, bahasa C juga banyak digunakan dalam mengembangkan perangkat lunak untuk permainan (*game*). Hal-hal inilah yang menyebabkan bahasa C menjadi bahasa yang sangat populer di kalangan industri perangkat lunak.

1.7. Kerangka Program dalam Bahasa C

Setiap program yang ditulis dengan menggunakan bahasa C harus mempunyai fungsi utama, yang bernama **`main()`**. Fungsi inilah yang akan dipanggil pertama kali pada saat proses eksekusi program. Artinya apabila kita mempunyai fungsi lain selain fungsi utama, maka fungsi lain tersebut baru akan dipanggil pada saat digunakan. Fungsi `main()` ini dapat mengembalikan nilai 0 ke sistem operasi yang berarti bahwa program tersebut berjalan dengan baik tanpa adanya kesalahan.

Berikut ini dua bentuk kerangka fungsi `main()` di dalam bahasa C yang sama-sama dapat digunakan.

a. Bentuk Pertama (*tanpa pengembalian nilai ke sistem operasi*)

```
void main(void) {
  Statemen_ yang_akan_dieksekusi;
  ...
}
```

Kata kunci `void` di atas bersifat opsional, artinya bisa dituliskan atau bisa juga tidak.

b. Bentuk Kedua (*dengan mengembalikan nilai 0 ke sistem operasi*)

```
int main(void) {
  Statemen_yang_akan_dieksekusi;
  ...
  return 0;
}
```

Kata kunci `void` di atas juga bersifat opsional. Namun, para programmer C pada umumnya menuliskan kata kunci tersebut di dalam fungsi yang tidak memiliki parameter. Dalam buku ini penulis akan banyak menggunakan bentuk kedua untuk setiap contoh-contoh program yang ada.

Adapun untuk kerangka lengkap dari program yang ditulis dalam bahasa C adalah seperti yang tertulis di bawah ini.

```
#include <nama_header_file>
...

/* Prototipe fungsi */
tipe_data nama_fungsi1(parameter1, parameter2, ...);
tipe_data nama_fungsi2(parameter1, parameter2, ...);
...

/* Fungsi utama */
int main(void) {
  Statemen_yang_akan_dieksekusi;
  ...
  return 0;
}

/* Implementasi fungsi */
tipe_data nama_fungsi1(parameter1, parameter2, ...) {
  Statemen_yang_akan_dieksekusi;
  ...
```

```
}

tipe_data nama_fungsi1(parameter1, parameter2, ...) {
  Statemen_yang_akan_dieksekusi
  ...
}
```

Oleh karena bahasa C merupakan bahasa prosedural yang menerapkan konsep runtunan (program dieksekusi per baris dari atas ke bawah secara berurutan), maka apabila kita menuliskan fungsi-fungsi lain tersebut di bawah fungsi utama, maka kita harus menuliskan bagian prototipe (*prototype*), hal ini dimaksudkan untuk mengenalkan terlebih dahulu kepada kompilator daftar fungsi yang akan digunakan di dalam program. Namun apabila kita menuliskan fungsi-fungsi lain tersebut di atas atau sebelum fungsi utama, maka kita tidak perlu lagi untuk menuliskan bagian prototipe di atas. Untuk lebih jelasnya, perhatikan kerangka program di bawah ini dimana fungsi-fungsi yang akan digunakan dituliskan sebelum fungsi utama.

```
#include <nama_header_file>
...

/* Fungsi-fungsi yang dibutuhkan ditulis sebelum fungsi main()
   sehingga tidak membutuhkan prototipe fungsi */
tipe_data nama_fungsi1(parameter1, parameter2, ...) {
  Statemen_yang_akan_dieksekusi;
  ...
}

tipe_data nama_fungsi1(parameter1, parameter2, ...) {
  Statemen_yang_akan_dieksekusi;
  ...
}
...

/* Fungsi utama */
int main(void) {
  Statemen_yang_akan_dieksekusi;
  ...
  return 0;
}
```

Program yang ditulis di dalam bahasa C akan disimpan dalam file yang berekstensi C (*.c), misalnya **contoh1.c**. Sebagai tambahan pengetahuan bagi Anda bahwa program yang ditulis dengan bahasa C ini dapat dikenali dan dikompilasi dengan menggunakan kompilator C++. Sehingga apabila Anda tidak memiliki kompilator C, maka Anda juga dapat mencoba menjalankan contoh-contoh program di dalam buku ini dengan menggunakan kompilator C++ (dalam segala bentuk variannya, seperti Turbo C++, Borland C++, MinGW, C++Builder, Visual C++ atau lainnya), tentunya selama program yang kita buat tidak menggunakan file header yang spesifik dari kompilator tertentu.

1.8. File Header (*.h)

File header adalah file dengan ekstensi h (*.h), yaitu file bantuan yang digunakan untuk menyimpan daftar-daftar fungsi yang akan digunakan di dalam program. Bagi Anda yang sebelumnya pernah mempelajari bahasa Pascal, file header ini serupa dengan *unit*. Dalam bahasa C, file header standar yang untuk proses input/output adalah **<stdio.h>**. Maka dari itu untuk hampir setiap kode program yang ditulis dalam bahasa C, akan mencantumkan file header <stdio.h>. Perlu sekali untuk diperhatikan bahwa apabila kita menggunakan file header yang telah disediakan oleh kompilator, maka kita harus menuliskannya di dalam tanda '<' dan '>' (misalnya *<stdio.h>*). Namun apabila kita menggunakan file header yang kita buat sendiri, maka file tersebut ditulis di antara tanda '""' (misalnya *"CobaHeader.h"*). Perbedaan antara keduanya terletak pada saat pencarian file tersebut. Apabila kita menggunakan tanda <>, maka file header tersebut akan dianggap berada pada direktori default yang telah ditentukan oleh kompilator. Sedangkan apabila kita menggunakan tanda "", maka file header dapat kita tentukan sendiri lokasinya.

File header yang akan kita gunakan harus kita daftarkan dengan menggunakan *directive* **#include** (lihat bab 11 - *Preprocessor Directive*). *Directive* #include ini berfungsi untuk memberitahu kepada kompilator bahwa program yang kita buat akan menggunakan file-file yang didaftarkan. Berikut ini contoh penggunaan *directive* **#include**.

```
#include <stdio.h>
#include <stdlib.h>
#include "MyHeader.h"
```

Setiap kita akan menggunakan fungsi tertentu yang disimpan dalam sebuah file header, maka kita juga harus mendaftarkan file header-nya dengan menggunakan *directive* #include. Sebagai contoh, kita akan menggunakan fungsi getch() dalam program, maka kita harus mendaftarkan file header <conio.h>.

1.9. Proses Pembentukan Program dalam Bahasa C

Secara umum, terdapat tiga buah proses untuk membentuk suatu program dengan menggunakan bahasa C, yaitu yang akan diterangkan secara detail pada sub bab berikut.

1.9.1. Menuliskan Kode Program

Hal dasar yang harus dilakukan untuk membuat suatu program adalah menuangkan permasalahan yang kita hadapi ke dalam bentuk kode program, yaitu dengan menerapkan konsep algoritma.

Kode program adalah kumpulan atau runtunan yang digunakan untuk memerintahkan komputer agar dapat menjalankan pekerjaan-pekerjaan tertentu sesuai yang kita kehendaki. Kode program sering juga dinamakan dengan istilah 'sintak'. Sebagai contoh, berikut ini contoh baris perintah dalam bahasa C.

```
printf(“Halo semua, apa kabar?”);
```

Perintah di atas akan menampilkan teks “Halo semua, apa kabar?” di layar. Anda tidak perlu bingung dan cemas dengan kehadiran fungsi `printf()` di atas karena hal tersebut akan kita bahas pada materi selanjutnya dalam buku ini.

Untuk menuliskan kode program, Anda dapat menggunakan program-program editor yang telah tersedia di dalam sistem operasi yang Anda gunakan. Misalnya apabila Anda menggunakan Microsoft Windows, maka Anda dapat menggunakan **Notepad**. Apabila Anda menggunakan MS-DOS, maka dapat menggunakan **Edit** serta di dalam Linux atau Unix, Anda dapat menggunakan editor **ed**, **joe**, **ex, emacs, pico** ataupun **vi**. Namun, sekarang telah banyak kompilator C yang menyediakan *built-in editor* untuk keperluan penulisan dan penyuntingan kode program yang akan kita buat sehingga kita tidak perlu menuliskannya dengan editor lain di luar kompilator.

1.9.2. Melakukan Kompilasi Kode Program

Sampai di sini, komputer belum mengetahui arti kode-kode program yang ditulis dalam bahasa C tersebut karena komputer hanya mengenal instruksi-instruksi biner yang dikenal dengan bahasa mesin. Maka dari itu kita membutuhkan suatu program lain untuk dapat menerjemahkan kode program (dalam bahasa C) tersebut ke dalam bahasa mesin. Program seperti inilah yang dinamakan dengan kompilator.

Kompilator akan menerima masukan kode program dan akan menghasilkan suatu kode objek yang disimpan dalam file objek. File objek tersebut berisi kode-kode mesin yang merupakan terjemahan dari kode program. Dalam sistem operasi Windows, biasanya file objek ini akan berekstensi **.obj**, sedangkan dalam sistem operasi Unix atau Linux pada umumnya file objek tersebut akan berekstensi **.o** (secara *default* dalam Linux akan menghasilkan file **a.out**).

Sebagai tambahan bagi Anda, berikut ini disajikan tabel yang berisi cara untuk melakukan kompilasi sesuai dengan beberapa kompilator C (dalam Microsoft Windows) yang sering digunakan. Sebagai contoh di sini kita memiliki kode program yang telah disimpan dalam file **coba.c**.

Kompilator	Perintah untuk melakukan kompilasi program
Turbo C	**`tcc`** `coba.c`
Borland C	**`bcc`** `coba.c`
Microsoft C	**`cl`** `coba.c`

Apabila Anda menggunakan sistem operasi Unix atau Linux, maka Anda dapat melakukan kompilasi kode program tersebut dengan menggunakan kompilator `cc` atau `gcc`, yaitu dengan cara menuliskan perintah seperti di bawah ini.

```
cc coba.c
```

atau

```
gcc coba.c
```

1.9.3. Proses *Linking*

Proses terakhir yang terdapat pada pembentukan suatu file eksekusi (*executable file*) atau file yang dapat dijalankan di komputer adalah proses *linking* (menghubungkan). Proses ini akan dilakukan secara internal pada saat selesai proses kompilasi.

Pada program di atas, kita menggunakan fungsi `printf()` yang merupakan fungsi pustaka (*library function*) yang telah disediakan oleh kompilator dalam file `<stdio.h>`. Di sini berarti kita menggunakan file lain untuk menjalankan kode program kita. Maka dari itu, file objek yang dihasilkan dari proses kompilasi di atas akan dikombinasikan atau dihubungkan dengan kode objek dari *library function* bersangkutan. Hal inilah yang disebut dengan proses *linking*. Adapun proses semacam ini dilakukan oleh program yang dinamakan dengan *linker*.

1.10. Mengenal Fungsi `printf()` dan `scanf()`

Dalam membuat suatu program komputer, kita tidak akan terlepas dari proses masukan (*input*) dan keluaran (*output*) data. Untuk melakukan hal tersebut, di dalam bahasa C telah disediakan fungsi pustaka, yaitu fungsi `printf()` yang berguna untuk menampilkan keluaran data dan fungsi `scanf()` yang berguna untuk membaca masukan data. Adapun prototipe dari kedua fungsi tersebut dapat Anda lihat di bawah ini.

```
printf(const char *format, ...);
scanf(const char *format, ...);
```

Setelah Anda mengetahui konsep dasar dan kerangka dari program dalam bahasa C yang telah diterangkan di atas, sekarang kita akan memulai penulisan kode program dengan menuliskan program yang sangat sederhana. Di sini kita akan menulis kode program di mana program tersebut dapat menampilkan teks `'Saya sedang belajar bahasa C'` ke layar monitor sehingga kita membutuhkan file header `<stdio.h>`. Adapun sintak programnya adalah seperti yang tertera di bawah ini.

```
#include <stdio.h>

int main(void) {

  /* Mencetak teks ke layar */
  printf("Saya sedang belajar bahasa C");

  return 0;
}
```

Apabila dijalankan, program tersebut akan memberikan hasil sebagai berikut:

```
Saya sedang belajar bahasa C
```

Sekarang kita akan membuat program yang akan membaca data masukan dari *keyboard*, yaitu dengan menggunakan fungsi `scanf()`. Adapun sintak programnya adalah seperti yang tertera di bawah ini.

```
#include <stdio.h>

int main(void) {

  /* Mendeklarasikan variabel X yang bertipe int */
  int X;
  /* Menampilkan teks sebagai informasi bagi pengguna
  program (user) */
  printf("Masukkan sebuah bilangan bulat : ");

  /* Membaca data masukan dari keyboard */
  scanf("%d", &X);

  /* Menampilkan kembali data yang telah dimasukkan dari
  keyboard */
  printf("Anda telah memasukkan bilangan %d", X);

  return 0;
}
```

Apabila Anda masih merasa bingung dengan kehadiran variabel `X` dan tipe data `int` yang terdapat pada sintak di atas, Anda tidak perlu cemas karena semua itu akan kita bahas lebih lanjut pada bab berikutnya dalam buku ini. Perhatikan juga statemen di bawah ini.

```
scanf("%d", &X);
```

Maksud dari sintak di atas adalah membaca nilai yang bertipe `int` dari *keyboard* dan menyimpan nilai tersebut ke dalam alamat memori yang ditempati oleh variabel `X`. Materi mengenai alamat memori akan kita bahas lebih lanjut di dalam *bab 7–Pointer*. Sekarang apabila program tersebut dijalankan, maka contoh hasil yang akan diberikan adalah sebagai berikut:

```
Masukkan sebuah bilangan bulat : 10
```

```
Anda telah memasukkan bilangan 10
```

Hal yang perlu Anda ketahui dalam menggunakan fungsi `printf()` adalah format argumen yang terdapat di dalamnya. Perhatikan sintak berikut.

```
...
printf("Nilai : %d", 10);
...
```

Hasil yang akan tampil di layar adalah sebagai berikut.

```
Nilai : 10
```

`%d` di atas menunjukkan argumen yang digunakan untuk menampilkan nilai dengan tipe data `int`. Sedangkan nilai 10 menunjukkan nilai yang akan diisikan untuk menggantikan argumen tersebut. Selanjutnya, karakter **d** yang mengikuti tanda `%` di sini berguna untuk memberitahu kepada kompilator bahwa nilai yang akan ditampilkan tersebut bertipe `int`. Artinya, apabila kita akan menampilkan nilai dengan tipe data lain (misalnya `char`, `float` ataupun `char*`) maka karakter yang mengikuti tanda `%` pun akan berbeda. Berikut ini daftar karakter yang dapat dijadikan sebagai format untuk menentukan tipe argumen pada fungsi `printf()`.

Karakter	Tipe Argumen	Keterangan
`d, i`	`int`	Untuk menampilkan tipe bilangan bulat dalam bentuk desimal (basis 10)
`o`	`int`	Untuk menampilkan tipe bilangan bulat dalam bentuk oktal (basis 8) tanpa diawali angka 0
`x, X`	`int`	Untuk menampilkan tipe bialangan bulat dalam bentuk heksadesimal (basis 16) tanpa diawali dengan tanda `0x` atau `0X`. format `x` digunakan untuk menampilkan hasil dalam huruf kecil, sedangkan `X` untuk huruf besar.
`u`	`int`	Menampilkan bilangan bulat tanpa tanda (*unsigned*)
`c`	`char`	Menampilkan karakter
`s`	`char*`	Menampilkan string (kumpulan karakter)
`f`	`float`	Menampilkan bilangan rii dengan tipe `float`. Apabila tipenya `double` maka akan ditulis `lf`.
`e, E`	`double`	Menampilkan bilangan riil dalam bentuk eksponen
`g, G`	`double`	Menampilkan bilangan riil, format ini akan secara otomatis memanggil `%e`, `%E` ataupun `%f` sesuai dengan nilai yang dimasukkan.
`p`	`void*`	Menampilkan pointer (alamat memori)
`%`	–	Apabila tanda `%` diikuti karakter `%`, maka program akan menampilkan tanda `%` sebagai keluarannya.

Berikut ini contoh penggunaan fungsi `printf()` yang mengandung lebih dari satu argumen dan dari tipe yang berbeda.

```
printf("Karakter: %c, Bilangan bulat: %d, String: %s,
  Bilangan riil: %2.3f", 'A', 23, "Mira", 19.4);
...
```

Sintak di atas akan memberikan hasil seperti di bawah ini.

```
Karakter: A, Bilangan bulat: 23, String: Mira, Bilangan riil :  19.400
```

Bilangan 2.3 yang terdapat pada `%2.3f` di atas menunjukkan bahwa bilangan riil tersebut akan ditampilkan dalam dua presisi dengan tiga angka di belakang koma. Berikut ini contoh format untuk menampilkan bilangan yang bertipe `int` dan `float` atau `double`.

`%d`	mencetak bilangan bulat (*integer*)
`%2d`	mencetak bilangan bulat dengan lebar 2 karakter
`%f`	mencetak bilangan riil (*floating point*)
`%2f`	mencetak bilangan riil dengan lebar 2 karakter
`%.3f`	mencetak bilangan riil dengan 3 angka di belakang koma
`%2.3f`	mencetak bilangan riil dengan lebar 2 karakter dan 3 angka di belakang koma

Satu hal lagi yang perlu untuk dipahami dalam menggunakan fungsi `printf()` adalah konstanta karakter, yaitu suatu konstanta yang diawali dengan tanda *backslash* (‘\’). Berikut ini daftar konstanta karakter yang telah didefinisikan dalam bahasa C.

Sequence	**Arti dan kegunaan**
`\a`	*Alert*; untuk membangkitkan suara dari spreaker
`\b`	*Backspace*; untuk meletakkan karakter *backspace*, kursor akan kembali ke depan sebanyak satu karakter
`\f`	*Formfeed*; untuk meletakkan karakter *formfeed*
`\n`	*Newline*; untuk meletakkan baris baru
`\r`	*Carriage return*; untuk meletakkan kursor di awal baris bersangkutan
`\t`	*Horizontal tab*; untuk meletakkan tab horisontal
`\v`	*Vertical tab*; untuk meletakkan tab vertikal
`\\`	*Backslash*; untuk menampilkan karakter \
`\?`	*Question mark*; menampilkan karakter tanda tanya (?)
`\'`	*Single quote*; menampilkan karakter petik tunggal (‘)
`\"`	*Double quote*; menampilkan karakter petik ganda (“)
`\ooo`	*Octal number*; menampilkan bilangan dalam bentuk oktal (basis 8)

\x*hh*	*Hexadecimal number*; menampilkan bilangan dalam bentuk heksadesimal (basis 16)

Apabila kita amati bahwa konstanta di atas tersusun dari dua karakter atau lebih, namun dalam bahasa C konstanta tersebut akan dianggap sebagai satu karakter. Untuk lebih jelasnya, coba Anda perhatikan program berikut ini yang akan menggunakan konstanta karakter `\t`, `\"` dan `\n`. Adapun sintaknya adalah sebagai berikut.

```
#include <stdio.h>

int main(void) {

  printf("Judul\t\t: \"Pemrograman Menggunakan Bahasa C\"\n");
  printf("Penulis\t\t: Budi Raharjo & I Made Joni\n");
  printf("Penerbit\t: INFORMATIKA Bandung");

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Judul       : "Pemrograman Menggunakan Bahasa C"
Penulis     : Budi Raharjo & I Made Joni
Penerbit    : INFORMATIKA Bandung
```

Sama halnya seperti pada fungsi `printf()`, kita juga harus menentukan format nilai yang akan dibaca pada saat kita menggunakan fungsi `scanf()`. Perhatikan kembali sintak berikut.

```
scanf("%d", &X);
```

`%d` di atas menunjukkan bahwa nilai yang akan dibaca dari *keyboard* tersebut akan dianggap sebagai tipe bilangan bulat (*integer*). Adapun format yang dapat digunakan dalam fungsi `scanf()` adalah sama seperti format yang digunakan dalam fungsi `printf()`.

1.11. Membuat Program di dalam Sistem Operasi Windows dan Linux

Pada bagian ini kita akan sedikit membahas mengenai pembuatan program dengan menggunakan bahasa C di dalam sistem operasi Windows dan Linux. Kenapa Window dan Linux? Hal ini disebabkan karena dua sistem operasi inilah yang saat ini banyak digunakan oleh para pengguna komputer di Indonesia. Dengan demikian, bagi Anda

yang menggunakan salah satu dari sistem operasi tersebut tetap dapat menggunakan buku ini sebagai bahan acuan untuk membuat program yang Anda inginkan.

Sebenarnya perbedaan pembuatan program di dalam Windows dan Linux (dalam hal ini menggunakan bahasa C) hanya terletak pada cara melakukan kompilasi dan eksekusinya saja. Berikut ini sebuah contoh program yang akan kita kompilasi di dalam Windows dan Linux. Adapun sintak yang dimaksud di sini adalah sebagai berikut.

```
#include <stdio.h>

int main(void) {
  int a, b, c, D;
  printf("Masukkan nilai a : "); scanf("%d", &a);
  printf("Masukkan nilai b : "); scanf("%d", &b);
  printf("Masukkan nilai c : "); scanf("%d", &c);
  D = (b*b) – 4 * a * c;
  printf("\nDeterminan = %d", D);
  return 0;
}
```

Simpan kode program di atas ke dalam file **determinan.c**. Untuk saat ini, Anda tidak perlu mempedulikan apa maksud dari sintak di atas. Hal yang perlu Anda perhatikan di sini adalah bagaimana cara yang harus digunakan untuk melakukan kompilasi dan eksekusi kode program tersebut di dalam kedua sistem operasi di atas.

1.11.1. Kompilasi dan Eksekusi Program di dalam Windows

Apabila Anda menggunakan kompilator C atau C++ yang telah menyediakan editor khusus (misalnya Turbo C, Turbo C++, Borland C, Borland C++, Dev-C++, MinGW Developer Studio, dll), tentu Anda akan mudah sekali dalam menjalankan kode program yang Anda buat, yaitu dengan memilih menu ***Compile*** (untuk proses kompilasi) dan menu ***Run*** (untuk proses eksekusi). Walaupun demikian, sebenarnya Anda dapat melakukannya dengan menuliskan perintah berikut pada *command prompt* yang tersedia (pada mode MS-DOS). Penulis berasumsi bahwa di sini kita menggunakan kompilator Turbo C, sehingga perintahnya adalah sebagai berikut.

- Untuk melakukan kompilasi program:

  `tcc determinan.c`

- Untuk melakukan eksekusi program:

  `determinan`

  atau

  `determinan.exe`

1.11.2. Kompilasi dan Eksekusi Program di dalam Linux

Di dalam sistem operasi Linux, kita dapat melakukan kompilasi program yang ditulis dengan bahasa C atau C++ dengan kompilator **cc**, **gcc** maupun **g++**. Seperti yang telah disinggung pada sub bab sebelumnya bahwa proses kompilasi di dalam sistem operasi Linux akan menghasilkan file objek dengan ekstensi **.o**. Namun, apabila kita tidak mencantumkan nama file objek yang dimaksud, maka kompilator secara *default* akan menyimpannya ke dalam file **a.out**. Kita dapat mendefinisikan file objek tersebut dengan menyertakan option **–o** pada saat melakukan kompilasi. Sebagai contoh apabila kita akan melakukan kompilasi terhadap file **determinan.c** dan akan menyimpan file objeknya ke dalam file **determinan.o**, maka kita akan menuliskan perintah seperti berikut.

- Untuk melakukan kompilasi:

  cc determinan.c –o determinan.o

- Untuk melakukan eksekusi program:

  ./determinan.o

Adapun hasil dari eksekusi program di atas (baik di Windows maupun di Linux) adalah sebagai berikut.

```
Masukkan nilai a : 2
Masukkan nilai b : 3
Masukkan nilai c : 1

Determinan : 1
```

Bab

2

Komentar, Variabel dan Tipe Data

2.1. Pendahuluan

Sebelum melangkah lebih jauh ke pembahasan selanjutnya, pada bab ini kita akan membahas elemen-elemen dasar yang terdapat dalam bahasa C. Hal ini dimaksudkan agar Anda tidak kesulitan dalam memahami sintak-sintak program yang terdapat pada bab-bab selanjutnya. Adapun elemen-elemen dasar yang dimaksud di sini adalah komentar program, variabel, konstanta dan tipe data. Melalui bab ini diharapkan Anda akan dapat memahami materi-materi berikut ini.

- Bagaimana cara serta pada saat kapan Anda seharusnya membuat komentar program
- Mengenal apa arti variabel beserta jenis-jenisnya di dalam bahasa C
- Bagaimana cara mendeklarasikan suatu variabel dengan tipe data tertentu
- Memahami perbedaan antara variabel dan konstanta
- Tipe-tipe data beserta rentang nilai yang digunakan di dalam bahasa C

2.2. Komentar Program

Untuk memudahkan pembacaan alur proses dari sebuah program, seorang programmer sebaiknya menuliskan komentar-komentar di dalam sintak program tersebut. Adapun definisi dari komentar program itu sendiri adalah bagian (berupa teks) di dalam program yang tidak ikut dieksekusi pada saat proses kompilasi. Tidak seperti bahasa pemrograman lainnya lainnya seperti Pascal, C++, Java dan lainnya yang memiliki beberapa cara untuk pembuatan komentar program, bahasa C hanya menyediakan sebuah cara, yaitu dengan menggunakan menuliskan tanda '`/*`' dan mengakhirinya dengan tanda '`*/`'. Artinya, setiap teks yang berada di belakang tanda `/*` akan dianggap sebagai komentar sampai ditemukan tanda `*/`. Jenis komentar yang terdapat dalam bahasa C ini dapat digunakan untuk komentar yang banyaknya hanya satu baris, dua baris atau lebih, bahkan bisa juga berfungsi sebagai komentar yang bersifat sisipan. Untuk lebih memahaminya, perhatikan contoh pembuatan komentar-komentar program di bawah ini.

```
/* Ini adalah komentar program yang banyaknya satu baris */
```

```
/* Ini adalah komentar program
    yang banyaknya dua baris */

/* Ini adalah
    komentar program
    yang banyaknya tiga baris */
```

Selain untuk menuliskan alur proses (algoritma) dari suatu program, para programmer juga umumnya menggunakan komentar untuk menuliskan deskripsi dari file kode yang mereka buat, misalnya sebagai berikut.

```
/* ------------------------------------------- */
/* Nama File               : Coba.c            */
/* Oleh                    : Budi Raharjo      */
/* Dibuat tanggal          : 21 Maret 2005     */
/* Dimodifikasi tanggal    : 11 Februari 2006  */
/* dst...                                      */
/* ------------------------------------------- */
```

2.2.1. Komentar Sisipan

Dalam kasus-kasus tertentu kita juga dapat menggunakan komentar program yang merupakan komentar sisipan, artinya komentar tersebut berada di tengah-tengah baris program. Berikut ini contoh pembuatan komentar sisipan yang dimaksud di sini.

```
int /* Ini adalah komentar sisipan */ X;
```

Meskipun hal tersebut diperbolehkan oleh kompilator, namun sebaiknya Anda menghindari penggunaannya. Alasannya adalah, komentar seperti itu justru akan mempersulit pembacaan alur proses dari sebuah program. Dengan demikian, seharunya Anda menuliskan komentar tersebut di bagian atas baris program ataupun setelah baris program berakhir, misalnya sebagai berikut.

```
/* Mendeklarasikan variabel X */
int X;

atau

int X;     /* Mendeklarasikan variabel X */
```

2.2.2. Komentar Bersarang

Beberapa kompilator C/C++ ada yang mendukung adanya pembuatan komentar yang bersarang (*nested comment*). Meskipun demikian Anda harus berhati-hati dalam menuliskan komentar seperti ini. Untuk mempermudah pembahasan, perhatikan contoh penulisan komentar program di bawah ini.

```
/* Ini adalah komentar /*beruntun*/ di dalam program yang
ditulis dengan bahasa C */
```

Pada penulisan komentar di atas, yang dikenali sebagai komentar hanya teks `'Ini adalah komentar /*beruntun'` dan teks sisanya, yaitu `'di dalam program yang ditulis dengan bahasa C */'` akan dianggap sebagai variabel-variabel yang tidak dikenali oleh kompilator. Hal ini disebabkan karena tanda `*/` akan mengakhiri tanda `/*` yang pertama, dan ini tentu akan menyebabkan terjadinya kesalahan pada saat proses kompilasi. Untuk itu, apabila kompilator C yang Anda gunakan mendukung penulisan komentar bersarang, sebaiknya hindari penggunaannya di dalam program.

2.3. Variabel

Variabel adalah suatu pengenal di dalam program yang berguna untuk menyimpan nilai dari tipe data tertentu. Adapun nilai yang disimpan dalam sebuah variabel nilainya bersifat dinamis, artinya nilai tersebut dapat diubah selama program berjalan. Untuk menggunakan variabel tentu kita harus mendeklarasikannya terlebih dahulu agar kompilator dapat mengenalinya. Berikut ini bentuk umum untuk proses pendeklarasian variabel di dalam bahasa C.

```
tipe_data nama_variabel;
```

Tipe data di sini berguna untuk memberitahu kepada kompilator bahwa variabel tersebut hanya dapat menampung nilai sesuai dengan tipe data tertentu yang didefinisikan ataupun tipe lain yang kompatibel. Adapun pembahasan mengenai tipe data itu sendiri baru akan diterangkan pada bagian akhir dari bab ini.

Berikut ini contoh pendeklarasian beberapa buah variabel dengan tipe data yang berbeda.

```
int x;   /* Mendeklarasikan variabel x dengan tipe data int */
float y; /* Mendeklarasikan variabel y dengan tipe data float */
char z;  /* Mendeklarasikan variabel z dengan tipe data char */
```

Bahasa C mengizinkan kita untuk melakukan pendeklarasian beberapa buah variabel dengan tipe data yang sama dalam satu baris, misalnya seperti berikut.

```
int a, b, c;    /* Mendeklarasikan variabel a, b dan c yang
                   bertipe int */
double x, y, z; /* Mendeklarasikan variabel x, y dan z yang
                   bertipe double */
```

2.3.1. Batasan Penamaan Variabel

Dalam mendeklarasikan suatu variabel, terdapat beberapa batasan yang perlu Anda ketahui, yaitu sebagai berikut:

- Bahasa C merupakan bahasa yang bersifat *case-sensitive* (membedakan penulisan huruf kecil dan huruf besar) sehingga nama variabel pada saat dideklarasikan dan digunakan, penulisannya harus sama. Dalam bahasa C, variabel `a` dan `A` akan dianggap sebagai dua buah variabel yang berbeda.

- Nama variabel tidak boleh berupa angka ataupun diawali oleh karakter yang berupa angka.

  Contoh:
  ```
  int 234;       /* SALAH, karena nama variabel berupa angka */
  int 3Dimensi; /* SALAH, karena nama variabel diawali oleh angka */
  int S1;        /* BENAR, karena nama variabel diawali oleh karakter huruf*/
  ```

- Nama variabel tidak boleh mengandung spasi.

  Contoh:
  ```
  float Bilangan Riil;      /* SALAH, karena mengandung spasi */
  float BilanganRiil;       /* BENAR */
  float _BilanganRiil;      /* BENAR */
  float Bilangan_Riil;      /* BENAR */
  ```

- Nama variabel tidak boleh menggunakan karakter-karakter yang merupakan simbol (`@`, `?`, `#`, `!`, `dll`), meskipun karakter tersebut terletak di tengah atau di belakang nama variabel.

  Contoh:
  ```
  int #lima;          /* SALAH, karena mengandung tanda # */
  int enam@;          /* SALAH, karena mengandung tanda @ */
  int tu?juh;         /* SALAH, karena mengandung tanda ? */
  ```

- Nama variabel tidak boleh menggunakan kata kunci maupun makro yang telah didefinisikan di dalam bahasa C.

  Contoh:
  ```
  int void;     /* SALAH, karena nama variabel berupa keyword void */
  int return;   /* SALAH, karena nama variabel berupa keyword return */
  ```

- Biasakan untuk memberi nama variabel se-deskriptif mungkin sehingga program akan mudah untuk dibaca dan dimengerti oleh orang lain. Sebagai contoh apabila kita akan mendeklarasikan variabel untuk menyimpan nilai dari luas lingkaran, maka kita dapat memberinya nama *luas*, *L*, *LuasLingkaran* atau yang lainnya.

2.3.2. Inisialisasi Variabel

Inisialisasi nilai terhadap suatu variabel berguna untuk menentukan nilai *default* ke dalamnya sehingga apabila kita tidak menggantikannya dengan nilai lain, maka nilai yang akan digunakan adalah nilai *default* tersebut. Dalam bahasa C, kita dapat langsung melakukan inisialisasi nilai terhadap variabel pada saat proses deklarasi. Berikut ini bentuk umum untuk melakukan hal tersebut.

```
tipe_data nama_variabel = nilai_untuk_inisialisasi;
```

Tanda sama dengan (=) di atas berfungsi sebagai operator *assigment*, yaitu operator untuk memasukkan sebuah nilai ke dalam suatu variabel. Materi tentang operator ini akan dibahas lebih lanjut dalam bab 3 – *Operator*.

Berikut ini contoh sintak untuk melakukan proses inisialisasi terhadap variabel.

```
int x = 0;      /* Melakukan inisialisasi terhadap variabel x
                   dengan nilai 0 */
char karakter = 'A';  /* Melakukan inisialisasi terhadap
                         variabel karakter dengan
                         nilai 'A' */
```

Untuk lebih memahaminya, di sini kita akan menuliskan program untuk menunjukkan kegunaan dari proses inisialisasi Adapun sintak programnya adalah seperti di bawah ini.

```
#include <stdio.h>

int main(void) {
  /* Mendeklarasikan variabel x tanpa melakukan inisialisasi */
  int x;
  /* Mendeklarasikan variabel y dan melakukan inisialisasi
     dengan nilai 0 */
  int y = 0;

  /*Menampilkan nilai yang terdapat di dalam variabel x dan y */
  printf("Nilai x = %d\n", x);
  printf("Nilai y = %d\n", y);

  return 0;
}
```

Hasil yang akan diperoleh dari program di atas adalah sebagai berikut.

```
Nilai x = 4206596
Nilai y = 0
```

Apa yang dapat Anda simpulkan? Nilai dari variabel x yang belum diinisialisasi (belum diisi nilai apapun) akan menghasilkan nilai yang tidak kita inginkan, yaitu nilai acak yang diberikan oleh kompilator secara otomatis untuk variabel yang belum diisi nilai. Hal inilah yang menjadi alasan kenapa kita sebaiknya melakukan inisialisasi terhadap suatu variabel yang kita deklarasikan.

2.3.3. Lingkup Variabel

Dalam sebuah program, lingkup variabel ditentukan oleh tempat di mana ia dideklarasikan. Pada bagian ini kita akan sedikit menengok ke depan bahwa program yang ditulis dengan bahasa C tersusun dari fungsi-fungsi yang melakukan proses-proses tertentu. Hal ini tentu bertujuan untuk mempermudah pembahasan mengenai lingkup variabel. Adapun menurut lingkupnya, variabel dibedakan menjadi dua, yaitu variabel global dan variabel lokal. Berikut ini penjelasan dari masing-masing topik tersebut.

2.3.3.1. Variabel Global

Variabel global adalah variabel yang dideklarasikan di luar fungsi, baik fungsi utama maupun fungsi pendukung lainnya. Berikut ini contoh penggunaannya dalam program.

```
#include <stdio.h>

/* Mendeklarasikan variabel global */
int x;

/* Memasukkan nilai ke dalam variabel x dengan nilai 10 */
x = 10;

/* Mendefinisikan fungsi lain dengan nama TampilkanNilaiX() */
void TampilkanNilaiX(void) {
  printf("Nilai x yang dipanggil dari fungsi
          TampilkanNilaiX()\t : %d\n", x);
}

/* Fungsi utama */
int main(void) {
  printf("Nilai x yang dipanggil dari fungsi main()\t\t\t :
         %d\n", x);
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Nilai x yang dipanggil dari fungsi TampilkanNilaiX()    : 10
Nilai x yang dipanggil dari fungsi main()               : 10
```

Dari hasil tersebut terlihat bahwa variabel x bersifat global, artinya dia akan dikenali dan dapat diakses oleh setiap fungsi yang terdapat di dalam program (*dalam hal ini fungsi `main()` dan `TampilkanNilaiX()`*). Apabila Anda masih merasa bingung dengan kehadiran pendefinisian fungsi di atas, Anda tidak perlu cemas karena materi tersebut akan kita bahas lebih detil di bab selanjutnya, yaitu bab 5 – *Fungsi*.

2.3.3.2. Variabel Lokal

Berbeda dengan variabel global, variabel lokal ini dideklarasikan di dalam sebuah fungsi sehingga hanya dapat dikenali dan diakses oleh fungsi itu saja. Dengan kata lain, sebuah fungsi tidak dapat mengenali dan mengakses variabel yang dideklarasikan di dalam fungsi lainnya. Oleh karena itu, nama variabel lokal dari fungsi yang satu dengan fungsi lainya bisa sama. Sebagai contoh, apabila kita mempunyai dua buah fungsi, yaitu `TentukanTitik2D()` dan `TentukanTitik3D()` yang berfungsi untuk menentukan sebuah titik dari bangun dimensi dua dan dimensi tiga. Di sini kita dapat mendeklarasikan suatu variabel `x` di setiap fungsi, dan ini akan dianggap sebagai dua variabel yang berbeda karena `x` di sini bersifat lokal. Untuk lebih memahaminya, perhatikan sintak berikut.

```
void TentukanTitik2D(void) {
  int x;   /* Mendeklarasikan variabel x di dalam fungsi
              TentukanTitik2D */
  int y;
  ...
}

void TentukanTitik3D(void) {
  int x;   /* Mendeklarasikan variabel x di dalam fungsi
              TentukanTitik3D */
  int y, z;
  ...
}
```

Berikut ini contoh program yang akan menunjukkan penggunaan variabel lokal.

```
#include <stdio.h>

void Fungsi1(void) {
  /* Mendeklarasikan variabel x yang bersifat lokal */
  int x;
  x = 12;
```

```
  printf("Nilai x di dalam Fungsi1() : %d\n", x);
}

void Fungsi1(void) {
  /* Mendeklarasikan variabel x yang bersifat lokal */
  int x;
  /* Mendeklarasikan variabel y yang bersifat lokal */
  int y = 50;
  x = 35;
  printf("Nilai x di dalam Fungsi2() : %d\n", x);
  printf("Nilai y di dalam Fungsi2() : %d\n", y);
}

/* Fungsi utama */
int main() {
  /* Melakukan pemanggilan terhadap fungsi Fungsi1() dan
     Fungsi2() */
  Fungsi1();
  Fungsi2();

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Nilai x di dalam Fungsi1() : 12
Nilai x di dalam Fungsi2() : 35
Nilai y di dalam Fungsi2() : 50
```

Nilai `y` yang terdapat dalam `Fungsi2()` hanya akan dikenali oleh `Fungsi2()` saja sedangkan `Fungsi1()` dan fungsi `main()` tidak dapat mengenailnya. Begitupun dengan variabel `x` yang dimiliki oleh `Fungsi1()` dan `Fungsi2()`, keduanya merupakan variabel yang berbeda dan hanya dikenali dalam fungsi bersangkutan saja.

2.3.4. Jenis Variabel

Variabel yang terdapat pada bahasa C cukup beragam, untuk itu kita harus mengenalnya satu per satu sehingga kita tidak akan merasa kesulitan dalam mengimplementasikannya ke dalam sebuah program. Menurut jenisnya, variabel dalam bahasa C dibedakan menjadi empat macam, yaitu variabel *otomatis*, *statis*, *eksternal* dan *register*. Berikut ini penjelasan masing-masing topik tersebut.

2.3.4.1. Variabel Otomatis

Variabel otomatis adalah variabel yang hanya dikenal di dalam suatu blok saja (dalam tanda **`{...}`**), baik itu blok pemilihan, pengulangan ataupun fungsi. Kita tahu bahwa dalam bahasa C, variabel dapat dideklarasikan dimana saja sesuai keinginan kita, ini tentu berbeda dengan kebanyakan bahasa pemrograman lainnya (misalnya Pascal) dimana variabel harus dideklarasikan sebelumnya ditempat khusus, yaitu pada bagian

deklarasi. Oleh karena itu, apabila setelah tanda **{** (permulaan blok), kita melakukan deklarasi variabel, maka variabel tersebut hanya akan dikenali oleh program sampai ditemukan tanda **}** (akhir blok) pertama yang ditemukan. Variabel seperti inilah yang dinamakan dengan variabel otomatis. Dikatakan 'otomatis' karena variabel ini dialokasikan pada saat pendeklarasian dan akan didealokasikan secara otomatis ketika program keluar dari suatu blok. Walaupun bersifat opsional, namun untuk mempertegas bahwa variabel tersebut sebagai variabel otomatis, kita dapat menggunakan kata kunci `auto` pada saat pendeklarasian. Berikut ini bentuk umumnya.

```
auto tipe_data nama_variabel;
```

Untuk lebih memahaminya, perhatikan sintak berikut.

```
#include <stdio.h>

int main(void) {
  int a;

  /* Melakukan blok pemilihan */
  if (a > 0) {
    auto int var_otomatis;  /* Mendeklarasikan variabel
                               otomatis. */
    /* Dalam blok ini variabel var_otomatis dikenali */
    ...
  } /* akhir blok pemilihan */

  /* di sini variabel var_otomatis sudah tidak dikenali lagi */
  return 0;
}
```

Berikut ini contoh penggunaannya di dalam sebuah program.

```
#include <stdio.h>
#include <math.h>     /* untuk menggunakan fungsi pow() */

int main(void) {
  int B = 5, e;
  printf("Masukkan nilai pangkat : "); scanf("%d", &e);
  if (e >= 0) {
    auto int hasil;
    hasil = pow (B, e);
    printf("%d^%d = %d", B, e, hasil);
  }

  /* printf("hasil = %d", hasil); */ /* SALAH, karena variabel
                                        hasil tidak
                                        dikenal */
  return 0;
}
```

Contoh hasil yang akan diperoleh dari program di atas adalah sebagai berikut.

```
Masukkan nilai pangkat : 3
5^3 = 125
```

2.3.4.2. Variabel Statis

Variabel statis adalah suatu variabel yang menyimpan nilai permanen dalam memori, artinya variabel tersebut akan menyimpan nilai terakhir yang diberikan. Untuk menyatakan bahwa suatu variabel adalah variabel statis adalah dengan menggunakan kata kunci `static`. Adapun bentuk umum dari pendeklarasiannya adalah sebagai berikut.

```
static tipe_data nama_variabel;
```

Untuk lebih memahami tentang variabel statis, di sini kita akan membuat dua buah program dimana program pertama akan menggunakan variabel biasa, sedangkan program kedua menggunakan variabel statis. Hal ini bertujuan agar Anda dapat mengetahui perbedaan yang tampak pada variabel statis.

a. Menggunakan variabel biasa

```
#include <stdio.h>

/* Mendefinisikan sebuah fungsi dengan nama KaliSepuluh() */
int KaliSepuluh(void) {
  int a = 1;          /* Mendeklarasikan variabel biasa */
  a = a * 10;
  return a;
}

/* Fungsi utama */
int main(void) {
  /* Mendeklarasikan variabel x, y dan z untuk menampung
     nilai dari fungsi */
  int x, y, z;

  x = KaliSepuluh();  /* Melakukan pemanggilan fungsi untuk
                         pertama kali */
  y = KaliSepuluh();  /* Melakukan pemanggilan fungsi untuk
                         kedua kali */
  z = KaliSepuluh();  /* Melakukan pemanggilan fungsi
                         untuk ketiga kali */

  /* Menampilkan nilai yang terdapat pada variabel x, y dan z */
  printf("Nilai x = %d\n", x);
  printf("Nilai y = %d\n", y);
```

```
  printf("Nilai z = %d\n", z);

  return 0;
}
```

Hasil yang akan diperoleh adalah sebagai berikut.

```
Nilai x = 10
Nilai y = 10
Nilai z = 10
```

b. Menggunakan variabel statis

```
#include <stdio.h>

/* Mendefinisikan sebuah fungsi dengan nama KaliSepuluh() */
int KaliSepuluh(void) {
  static int a = 1;   /* Mendeklarasikan variabel statis */
  a = a * 10;
  return a;
}

/* Fungsi utama */
int main(void) {
  /* Mendeklarasikan variabel x, y dan z untuk menampung nilai
     dari fungsi */
  int x, y, z;

  x = KaliSepuluh();  /* Melakukan pemanggilan fungsi untuk
                         pertama kali */
  y = KaliSepuluh();  /* Melakukan pemanggilan fungsi untuk
                         kedua kali */
  z = KaliSepuluh(); /* Melakukan pemanggilan fungsi untuk
                        ketiga kali */

  /* Menampilkan nilai yang terdapat pada variabel x, y dan z */
  printf("Nilai x = %d\n", x);
  printf("Nilai y = %d\n", y);
  printf("Nilai z = %d\n", z);

  return 0;
}
```

Hasil yang akan diperoleh adalah sebagai berikut.

```
Nilai x = 10
```

```
Nilai y = 100
Nilai z = 1000
```

Apa yang dapat Anda simpulkan dari hasil di atas? Kita lihat bahwa pada saat kita menggunakan variabel biasa, setiap kali pemanggilan fungsi tersebut, kompilator akan mengalokasikan variabel `a` dan menginisialisasinya dengan nilai 1, yang selanjutnya akan dikalikan dengan 10. Namun yang perlu diperhatikan adalah pada saat program selesai melakukan proses yang terdapat fungsi, maka variabel `a` akan didealokasikan secara otomatis. Hal ini yang menyebabkan setiap pemanggilan fungsi akan selalu menghasilkan nilai yang tetap, yaitu 10.

Sedangkan apabila kita menggunakan variabel statis, maka ketika pertama kali fungsi dipanggil, variabel `a` akan dialokasikan dan akan tetap bersarang di memori sampai program dihentikan. Adapun nilai `a` yang ada pada saat ini adalah 1 sehingga apabila dikalikan 10, maka hasilnya adalah 10. Kemudian ketika fungsi itu dipanggil untuk yang kedua kalinya, nilai `a` sudah menjadi 10, *bukan 1*, dan nilai terakhir tersebut kemudian dikalikan lagi dengan 10, menghasilkan nilai 100. Begitu juga dengan pemanggilan fungsi yang ketiga, nilai `a` yang ada adalah 100 sehingga apabila dikalikan 10 maka hasilnya 1000.

2.3.4.3. Variabel Eksternal

Bahasa C mengizinkan kita untuk menuliskan sintak program ke dalam file yang terpisah dengan bertujuan untuk modularisasi program. Untuk itu apabila kita ingin mendeklarasikan variabel yang dapat dikenali dan diakses oleh masing-masing file yang terpisah tersebut, maka variabel itu harus kita deklarasikan sebagai variabel eksternal. Adapun caranya adalah dengan menambahkan kata kunci `extern` pada saat pendeklarasikan. Berikut ini bentuk umumnya.

```
extern tipe_data nama_variabel;
```

Sebagai contoh, asumsikan kita memiliki dua buah file program, yaitu eksternal.c dan utama.c, maka apabila kita mendeklarasikan suatu variabel eksternal di file utama.c, maka dalam file eksternal.c variabel itu juga dapat diakses, yaitu dengan mendefinisikannya melalui kata kunci `extern`. Untuk mengilustrasikan kasus ini, coba Anda perhatikan potongan sintak di bawah ini.

Dalam file **utama.c**

```
...
int var_eksternal;    /* deklarasi variabel */

int main(void) {
  var_eksternal = 100;

  printf("Nilai var_eksternal : %d\n", var_eksternal);

  /* Memanggil fungsi SetNilai() yang terdapat pada file
     eksternal.c */
  SetNilai();
```

```
  printf("Nilai var_eksternal : %d\n", var_eksternal);

  return 0;
}
```

Dalam file **eksternal.c**

```
...
extern int var_eksternal;
void SetNilai(void) {
      var_eksternal = 500;
}
...
```

Apabila dikompilasi dan dijalankan, maka program di atas akan memberikan hasil sebagai berikut.

```
Nilai var_eksternal : 100
Nilai var_eksternal : 500
```

Apabila Anda masih bingung dengan sintak di atas, itu wajar. Namun Anda tidak perlu cemas karena yang perlu Anda ketahui di sini adalah konsep dari variabel eksternal saja. Variabel eksternal pada umumnya digunakan pada saat pembuatan file header. Pembahasan mengenai pembuatan program yang terdiri dari beberapa file, itu akan dibahas secara tersendiri dalam buku ini, yaitu pada lampiran B – *Membuat File Header (*.h)*.

2.3.4.4. Variabel Register

Berbeda dengan variabel biasa yang akan bertempat di memori, variabel register ini akan disimpan di dalam register CPU (*Central Proccessing Unit*). Dengan demikian apabila kita ingin mengisikan atau mengubah nilai variabel register, maka kita tidak perlu melakukan akses terhadap memori sehingga proses yang dilakukan pun lebih cepat.

Perlu untuk diperhatikan bahwa variabel register ini hanya dapat diterapkan ke tipe bilangan bulat, karakter dan pointer saja. Selain itu variabel ini hanya boleh dideklarasikan sebagai variabel lokal ataupun parameter dari fungsi. Untuk mendeklarasikannya, kita harus menggunakan kata kunci `register`. Adapun bentuk umumnya adalah sebagai berikut.

```
register tipe_data nama_variabel;
```

Berikut ini contoh program yang akan menunjukkan penggunaan variabel register.

```
#include <stdio.h>

/* Mendefinisikan fungsi untuk menghitung
   nilai perpangkatan B^e */
int Pangkat(register int B, register int e) {
  /* Mendeklarasikan variabel register */
  register int hasil;
  hasil = 1;

  for ( ; e; e--) {
    hasil *= B;
  }
  return hasil;
}

/* Fungsi utama */
int main(void) {
  printf("2^6 = %d", Pangkat(2, 6));  /* Menghitung 2^6 */

  return 0;
}
```

Hasil yang akan diperoleh dari program di atas adalah sebagai berikut.

```
2^6 = 64
```

2.4. Konstanta

Konstanta adalah sebuah tetapan yang tidak dapat diubah nilainya ketika program berjalan. Dalam bahasa C, para programmer biasanya menggunakan sebuah makro untuk membuat suatu konstanta, yaitu dengan mendefinisikannya melalui *directive* `#define`. Untuk mengetahui informasi lebih lanjut mengenai *directive* `#define` dan konsep kerja makro, Anda dapat melihat bab 11 – *Preprosesor Directive*.

Berikut ini contoh sintak pembuatan konstanta dengan mendefinisikan makro.

```
#define PI      3.1416     /* Membuat konstanta PI dengan
                              nilai 3.1416 */
#define NULL    0          /* Membuat konstanta NULL
                              dengan nilai 0 */
#define FALSE   0          /* Membuat konstanta FALSE dengan
                              nilai 0 */
#define TRUE    1          /* Membuat konstanta TRUE dengan
                              nilai 1 */
```

Meskipun tidak bersifat mutlak, namun kebanyakan para programmer C pada umumnya mendefinisikan sebuah makro dengan menggunakan huruf besar (kapital). Berikut ini contoh program yang dapat menunjukkan penggunaan makro.

```
#include <stdio.h>

/* Mendefinisikan makro untuk mengeset nilai PI */
#define PI  3.1416

int main(void) {
  /* Mendeklarasikan variabel untuk menampun nilai jari-jari
     dan luas lingkaran */
  double jari2, luas;

  /* Meminta pengguna program (user) untuk memasukkan panjang
     jari-jari */
  printf("Masukkan panjang jari-jari lingkaran : ");
  scanf("%lf", &jari2);

  /* Melakukan perhitungan luas lingkaran dan memasukkan
     hasilnya ke dalam
  variabel luas */
  luas = PI * jari2 *jari2;
  /* Menampilkan nilai dari variabel luas */
  printf("Luas lingkaran = %.2lf", luas);

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Masukkan panjang jari-jari lingkaran : 2
Luas lingkaran = 12.57
```

Selain menggunakan directive `#define`, bahasa C juga menyediakan kata kunci `const` untuk membentuk suatu kontanta. Berikut ini bentuk umum pembuatannya.

```
const tipe_data nama_konstanta = nilai_kontan;
```

Sebagai contoh apabila kita akan membuat konstanta `PI` dengan menggunakan kata kunci `const`, maka kita akan menuliskannya sebagai berikut.

```
const double PI = 3.1416;
```

2.5. Tipe Data

Dalam dunia pemrograman, tipe data adalah sesuatu yang digunakan untuk merepresentasikan jenis dari suatu nilai tertentu. Sebagai contoh nilai 10 adalah nilai yang bertipe bilangan bulat, 12.23 bertipe bilangan riil serta 'A' bertipe karakter. Untuk menulis suatu program dalam bahasa tertentu, tentunya kita harus mengetahui terlebih dahulu akan tipe data yang terdapat di dalamnya. Kesalahan dalam menentukan tipe data dapat menyebabkan nilai yang dihasilkan tidak akurat. Maka dari itu, pada bagian ini Anda akan diperkenalkan macam-macam tipe data yang terdapat di dalam bahasa C agar Anda dapat terhindar dari masalah akurasi nilai yang tidak diinginkan.

Secara umum tipe data dapat dibedakan menjadi tiga bagian besar, yaitu tipe data dasar, bentukan dan enumerasi. Berikut ini penjelasan dari masing-masing tipe tersebut.

2.5.1. Tipe Data Dasar

Tipe data dasar dalam bahasa C dikelompokkan ke dalam empat kategori, yaitu tipe bilangan bulat (*integer*), bilangan riil (*floating-point*), karakter atau string serta tipe logika (*boolean*).

2.5.1.1. Tipe Bilangan Bulat

Sesuai dengan namanya, tipe bilangan bulat adalah suatu tipe data yang digunakan untuk menyimpan nilai-nilai yang berbentuk bilangan bulat (bilangan yang tidak mengandung koma), misalnya 10, 23, 200 dan sebagainya. Namun yang perlu diperhatikan juga bahwa bilangan bulat juga dikelompokkan lagi menjadi dua jenis, yaitu bilangan bulat positif dan negatif. Untuk itu, di dalam bahasa C tipe bilangan bulat juga dibedakan lagi menjadi beberapa macam dengan rentang nilai tertentu. Adapun yang termasuk ke dalam tipe bilangan bulat di dalam bahasa C adalah seperti yang tampak pada tabel di bawah ini.

Tipe Data	Ukuran (dalam bit)	Rentang	Format
`int`	16 atau 32	-32768 sampai 32767	`%d`
`unsigned int`	16 atau 32	0 sampai 65535	`%u`
`signed int`	16 atau 32	Sama seperti `int`	`%d`
`short int`	16	-32768 sampai 32767	`%d`
`unsigned short int`	16	0 sampai 65535	`%u`
`signed short int`	16	Sama seperti `short int`	`%d`
`long int`	32	-2147483648 sampai 2147483647	`%l`
`signed long int`	32	Sama seperti `long int`	`%l`
`unsigned long int`	32	0 sampai 4294967295	`%L`

2.5.1.2. Tipe Bilangan Riil

Selain tipe data untuk menyimpan nilai-nilai bilangan bulat, bahasa C juga menyediakan tipe data yang digunakan untuk menyimpan nilai-nilai bilangan riil (bilangan yang mengandung koma), misalnya 2.13, 5.64, 1.98 dan lainnya. Adapun yang termasuk ke dalam tipe tersebut adalah seperti yang tertera pada tabel di bawah ini.

Tipe Data	Ukuran (dalam bit)	Rentang	Format
`float`	32	3.4e-38 sampai 3.4e+38	`%f`
`double`	64	1.7e-308 sampai 1.7e+308	`%lf`
`long double`	80	1.7e-308 sampai 1.7e+308	`%lf`

Sebagai catatan apabila Anda akan menuliskan bilangan riil tersebut ke dalam bentuk eksponen, maka format yang akan digunakan adalah %e atau %E.

2.5.1.3. Tipe Karakter dan String

Tipe ini digunakan untuk merepresentasikan data-data yang berupa karakter. Adapun yang termasuk ke dalam tipe data karakter di dalam bahasa C adalah seperti yang tertera pada tabel di bawah ini.

Tipe Data	Ukuran (dalam bit)	Rentang	Format
`char`	8	-128 sampai +127	`%c`
`signed char`	8	-128 sampai +127	`%c`
`unsigned char`	8	0 sampai 255	`%c`

Data akan dianggap sebagai karakter apabila diapit oleh tanda petik tunggal (‘’), misalnya ‘A’, ’B’, ’a’, ’b’, dan sebagainya. Sedangkan apabila diapit oleh tanda petik ganda (“”), maka akan dianggap sebagai string, misalnya “A”, “B”, “a”, “b”. Adapun yang dinamakan dengan string itu sendiri adalah kumpulan dari karakter, misalnya “Saya sedang belajar”, “Bahasa C” dan sebagainya. Format yang digunakan untuk tipe string adalah `%s`. Tipe string ini akan kita bahas lebih lanjut pada bagian lain dalam buku ini, yaitu pada bab 6 – *Array dan String*.

2.5.1.4. Tipe Logika

Tipe logika adalah tipe data yang merepresentasikan nilai benar (*true*) dan salah (*false*). Bahasa C tidak mendefinisikan tipe khusus untuk menampung nilai-nilai tersebut. Hal ini tentu berbeda dengan bahasa pemrograman lain (misalnya bahasa Pascal) yang telah menyediakan tipe `boolean` untuk merepresentasikan nilai logika. Dalam bahasa C nilai `true` direpresentasikan dengan nilai selain 0 (biasanya dengan nilai 1), sedangkan nilai `false` direpresentasikan dengan nilai 0. Pada umumnya para programmer C mendefinisikan tipe logika melalui pembuatan makro maupun tipe enumerasi. Berikut ini contoh pendefinisian yang biasa dilakukan untuk membuat tipe logika di dalam bahasa C.

```
#define TRUE    1
#define FALSE   0
```

Apabila menggunakan enumerasi, maka contoh sintaknya adalah sebagai berikut.

```
typedef enum {FALSE, TRUE} boolean;
```

Dengan demikian tipe `boolen` yang kita definisikan di atas dapat digunakan untuk mendeklarasikan variabel.

2.5.2. Tipe Data Bentukan

Tipe data bentukan adalah suatu tipe data yang didefinisikan sendiri untuk memenuhi kebutuhan-kebutuhan progam yang akan kita buat. Adapun yang termasuk ke dalam tipe data bentukan adalah tipe array (larik) dan struktur. Materi mengenai array dan struktur baru akan kita bahas secara rinci pada bagian tersendiri dalam buku ini, yaitu pada bab 6 – *Array dan String* dan bab 8 – *Struktur dan Union*.

2.5.3. Enumerasi

Enumerasi adalah suatu tipe data yang nilainya telah didefinisikan secara pasti pada saat pembuatan tipe tersebut. Tipe ini umumnya digunakan untuk menyatakan sesuatu yang nilainya sudah pasti, seperti nama hari, nama bulan, jenis kelamin dan lain sebagainya. Adapun untuk membuat sebuah enumerasi di dalam bahasa C adalah dengan menggunakan kata kuci `enum`. Bentuk umum untuk mendefinisikannya adalah sebagai berikut.

```
enum nama_enumerasi { nilai1, nilai2, nilai3, … };
```

Berikut ini contoh sintak pendefinisian enumerasi yang terdapat dalam bahasa C.

```
enum JenisKelamin { pria, wanita };
enum boolean      { false, true };
enum WarnaPrimer  { merah, hijau, biru };
enum NamaHari     { minggu, senin, selasa, rabu, kamis, jumat,
                    sabtu };
enum NamaBulan    { januari, februari, maret, april, mei, juni,
                    juli, agustus, september,
                    oktober, november, desember };
```

Untuk mengakses nilai dari enumerasi ini kita dapat langsung mengisikan nilainya ataupun dengan menggunakan nilai integer yang secara *default* indeksnya dimulai dari nol. Pada contoh enumerasi `JenisKelamin` di atas, nilai 0 berati `pria` dan nilai 1

berarti `wanita`. Begitu juga untuk enumerasi `WarnaPrimer`, nilai 0 berarti `merah`, nilai 1 berarti `hijau` dan nilai 2 berarti `biru`.

Untuk lebih jelasnya, coba Anda perhatikan contoh program di bawah ini yang merupakan implementasi dari tipe enumerasi di atas.

```
#include <stdio.h>
#include <string.h>   /* untuk menggunakan fungsi strcpy() */

/* Mendefinisikan enumerasi yang bernama JenisKelamin */
enum JenisKelamin { pria, wanita };

int main(void) {

      /* Mendeklarasikan variabel bertipe string */
      char nama[25];
      /* Mendeklarasikan variabel gender yang bertipe
         JenisKelamin */
      enum JenisKelamin gender;

      /* Mengisikan nilai ke dalam variabel nama dan gender */
      strcpy(nama, "Mira");
      gender = wanita;  /* dapat ditulis dengan gender = 1 */

      /* Menampilkan nilai yang disimpan di dalam variabel nama
         dan gender */
      printf("Nama \t\t: %s\n", nama);
      printf("Jenis kelamin \t: %d", gender);

      return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Nama          : Mira
Jenis Kelamin : 1
```

Walaupun demikian, kita juga diizinkan untuk mendefinisikan nilai dari enumerasi sesuai keinginan kita, artinya nilai indeks tidak selalu diawali dengan nilai 0. Berikut ini contohnya.

```
enum JenisKelamin { pria = 10, wanita = 20 };
enum Hari { minggu=1, senin=2, selasa=3, rabu=4, kamis=5,
            jumat=6, sabtu=7 };
```

Bab

3

3.1. Pendahuluan

Dalam kegiatan pemrograman kita selalu dilibatkan dalam kesibukan melakukan operasi-operasi tertentu, misalnya seperti perhitungan-perhitungan matematik, pemanipulasian string, pemanipulasian bit ataupun operasi-operasi lainnya. Untuk melakukan hal tersebut tentu kita harus menguasai dengan benar akan penggunaan operator-operator yang digunakan dalam suatu bahasa pemrograman tertentu. Operator itu sendiri adalah tanda yang digunakan untuk meyelesaikan suatu operasi tertentu.

Pada bagian ini kita akan membahas mengenai arti dan penggunaan operator-operator yang terkandung dalam bahasa C, yang akan kita kelompokkan ke dalam empat bagian besar yaitu operator *assigment*, *unary*, *binary*, dan *ternary*.

3.2. Operator *Assignment*

Operator *assignment* adalah suatu operator penugasan yang digunakan untuk memasukkan nilai ke dalam suatu variabel. Dalam bahasa C, operator *assignment* ini dilambangkan dengan tanda sama dengan (=). Untuk lebih memahaminya, perhatikan contoh sintak berikut.

```
C = 15;
```

Sintak tersebut berarti memasukkan nilai 15 ke dalam variabel `C`. Selain itu bahasa C juga menawarkan kita untuk memasukkan nilai ke dalam beberapa variabel secara sekaligus, berikut ini contoh yang akan menunjukkan hal tersebut.

```
A = B = C = 20;
```

Di sini kita memasukkan nilai 20 ke dalam variabel `A`, `B` dan `C`. Berikut ini program yang akan menunjukkan penggunaan operator *assignment*.

```
#include <stdio.h>

int main(void) {
  /* Mendeklarasikan variabel a, b, c dab d bertipe int */
```

```
  int a, b, c, d;

  /* Melakukan assignment ke dalam variabel a dengan nilai 10 */
  a = 10;

  /* Melakukan assignment ke dalam variabel b dan c dengan nilai
     35 */
  b = c = 35;

  /* Melakukan assigment ke dalam variabel d dengan nilai yang
     terdapat dalam varaibel a */
  d = a;

  /* Menampilkan nilai yang terdapat pada variabel a, b,
     c dan d */
  printf("Nilai a \t= %d\n", a);
  printf("Nilai b \t= %d\n", b);
  printf("Nilai c \t= %d\n", c);
  printf("Nilai d \t= %d\n", d);
  return 0;
}
```

Program di atas akan memberikan hasil seperti di bawah ini.

```
Nilai a = 10
Nilai b = 35
Nilai c = 35
Nilai d = 10
```

Berbeda dengan kebanyakan bahasa pemrograman lainnya, bahasa C juga menawarkan cara penulisan sintak untuk mempersingkat proses *assignment*. Sebagai contoh apabila terdapat statemen **`j = j+4`**, maka statemen tersebut dapat kita tulis dengan **`j += 4`**. Berikut ini bentuk penyingkatan yang terdapat dalam bahasa C.

Contoh Statemen	Bentuk Penyingkatan
`J = J + 4`	`J += 4`
`J = J - 4`	`J -= 4`
`J = J * 4`	`J *= 4`
`J = J / 4`	`J /= 4`
`J = J % 4`	`J %= 4`
`J = J << 4`	`J <<= 4`
`J = J >> 4`	`J >>= 4`
`J = J & 4`	`J &= 4`
`J = J ^ 4`	`J ^= 4`
`J = J | 4`	`J |= 4`

Apabila Anda masih merasa bingung dengan kehadiran operator-operator di atas, Anda tidak perlu cemas karena semua itu akan kita bahas dalam bab ini.

3.3. Operator *Unary*

Operator *unary* adalah operator yang digunakan untuk melakukan operasi-operasi matematik yang hanya melibatkan satu buah *operand*. Dalam bahasa C, yang termasuk ke dalam operator *unary* adalah seperti yang tampak pada tabel di bawah ini.

Operator	**Jenis Operasi**	**Contoh Penggunaan**
+	Membuat nilai positif	+10
-	Membuat nilai negatif	-10
++	**Increment *(menambahkan nilai 1)***	x++
--	**Decrement *(mengurangi nilai 1)***	x--

Berikut ini program yang akan menunjukkan penggunaan operator unary + dan -.

```
#include <stdio.h>

int main(void) {
  int x, y;
  /* Menjadikan nilai positif pada bilangan 5 dan
     memasukkannya ke variabel x */
  x = +5;  /* Sama artinya dengan x = 5; */

  /* Menjadikan nilai negatif pada bilangan 10 dan
     memasukkannya ke variabel y */
  y = -10;

  printf("%d x (%d) = %d", x, y, x*y);

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah seperti yang tampak di bawah ini.

```
5 x (–10) = –50
```

3.3.1. *Increment*

Increment adalah suatu proses menaikkan (menambahkan) nilai dengan nilai 1. Adapun operator dalam bahasa C yang digunakan untuk melakukan proses tersebut adalah operator ++. Maka dari itu operator ++ ini disebut dengan *operator increment*. Sebagai contoh apabila kita memiliki variabel x yang bertipe int dengan nilai 10, maka setelah operasi ++x atau x++, maka nilai x akan bertambah satu, yaitu menjadi 11.

Dalam bahasa C, *increment* terbagi lagi ke dalam dua bagian, yaitu *pre-increment* dan *post-increment*. Berikut ini penjelasan dari masing-masing topik tersebut.

3.3.1.1. Pre-Increment

Pre-increment berarti menaikkan nilai yang terdapat pada sebuah variabel sebelum nilai dari variabel tersebut diproses di dalam program. Operator ++ akan dianggap sebagai *pre-increment* apabila dituliskan di depan nama variabel atau nilai yang akan dinaikkan. Sebagai contoh, misalnya kita memiliki variabel x bertipe int dengan nilai 10 dan di sini kita akan melakukan pemrosesan terhadap variabel tersebut dengan cara menampilkan nilainya ke layar monitor. Apabila kita melakukan operasi *pre-increment*, yaitu dengan menuliskan ++x, maka nilai x akan dinaikkan terlebih dahulu sebelum nilai tersebut ditampilkan ke layar. Hal ini menyebabkan nilai yang akan ditampilkan adalah 11. Untuk membuktikannya, coba perhatikan program di bawah ini.

```
#include <stdio.h>

int main(void) {
  int x=10;

  printf(“Nilai x awal \t= %d\n”, x);
  printf(“Nilai ++x \t= %d\n”, ++x);
  printf(“Nilai x akhir \t= %d\n”, x);
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah seperti yang tampak di bawah ini.

```
Nilai x awal    = 10
Nilai ++x       = 11
Nilai x akhir   = 11
```

Dari hasil tersebut dapat kita lihat bahwa nilai x mula-mula adalah 10, kemudian nilai ++x adalah 11. Hal ini disebabkan karena pada operasi ++x, nilai 10 dinaikkan terlebih dahulu sebelum ditampilkan. Sampai di sini, nilai x adalah 11 sehingga pada saat pemanggilan berikutnya, x yang akan ditampilkam juga bernilai 11.

3.3.1.2. Post-Increment

Post-increment berarti menaikkan nilai yang terdapat pada sebuah variabel setelah nilai dari variabel tersebut diproses di dalam program. Pada *post-increment* operator ++ ditulis setelah variabel atau nilai yang akan dinaikkan. Sebagai contoh, misalkan kita memiliki variabel x yang bernilai 10, maka nilai x++ yang akan ditampilkan di layar adalah 10 (bukan 11). Kenapa demikian? Hal ini disebabkan karena nilai dari variabel x

tersebut akan ditampilkan terlebih dahulu, selanjutnya baru dinaikkan nilainya. Untuk lebih jelasnya, perhatikan kembali program di bawah ini.

```
#include <stdio.h>

int main(void) {
  int x=10;
  printf("Nilai x awal \t= %d\n", x);
  printf("Nilai x++ \t= %d\n", x++);
  printf("Nilai x akhir \t= %d\n", x);
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah seperti yang tampak di bawah ini.

```
Nilai x awal    = 10
Nilai x++       = 10
Nilai x akhir   = 11
```

3.3.2. *Decrement*

Decrement merupakan kebalikan dari *increment*, yang merupakan proses penurunan nilai dengan nilai 1. *Decrement* juga dibagi menjadi dua macam, yaitu *pre-decrement* dan *post-decrement*. Namun di sini kita tidak akan membahas tentang kedua jenis *decrement* tersebut karena konsepnya sama persis dengan *pre-increment* dan *post-increment*.

Untuk lebih memahaminya, berikut ini program yang akan menunjukkan penggunaan operator *decrement*.

```
#include <stdio.h>

int main(void) {
  int x=10, y=10;

  /* Melakukan pre-decrement pada variabel x */
  printf("Nilai x awal \t= %d\n", x);
  printf("Nilai --x \t= %d\n", --x);
  printf("Nilai x akhir \t= %d\n\n", x);

  /* Melakukan post-decrement pada variabel y */
  printf("Nilai y awal \t= %d\n", y);
  printf("Nilai y-- \t= %d\n", y--);
  printf("Nilai y akhir \t= %d\n", y);

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Nilai x awal    = 10
Nilai --x       = 9
Nilai x akhir   = 9

Nilai y awal    = 10
Nilai y--       = 10
Nilai y akhir   = 9
```

3.4. Operator *Binary*

Dalam ilmu matematika, operator *binary* adalah operator yang digunakan untuk melakukan operasi yang melibatkan dua buah *operand*. Pada pembahasan ini, kita akan mengelompokkan operator *binary* ini ke dalam empat jenis, yaitu operator *aritmetika*, *logika*, *relasional* dan *bitwise*.

3.4.1. Operator Aritmetika

Operator aritmetika adalah operator yang berfungsi untuk melakukan operasi-operasi aritmetika seperti penjumlahan, pengurangan, perkalian dan pembagian. Berikut ini operator aritmetika yang terdapat dalam bahasa C.

Operator	**Jenis Operasi**	**Contoh Penggunaan**
+	Penjumlahan	`12 + 13 = 25`
-	Pengurangan	`15 – 12 = 13`
*	***Perkalian***	`12 * 3 = 36`
/	***Pembagian***	`10.0 / 3.0 = 3.3333`
%	***Sisa bagi (*modulus*)***	`10 % 4 = 2`

Agar Anda dapat lebih memahaminya, di sini dituliskan beberapa program yang akan menunjukkan penggunaan dari masing-masing operator tersebut.

a. Menggunakan Operator +

```
#include <stdio.h>

int main(void) {
  /* Mendeklarasikan variabel a dan b yang akan dijadikan
     operand */
  int a=12, b=13;
```

```
  /* Mendeklarasikan variabel c sebagai penampung nilai hasil
     penjumlahan */
  int c;

  /* Melakukan operasi penjumlahan */
  c = a + b;

  /* Menampilkan nilai hasil penjumlahan */
  printf("%d + %d = %d", a, b, c);

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
12 + 13 = 25
```

b. Menggunakan Operator –

```
#include <stdio.h>

int main(void) {
  /* Mendeklarasikan variabel a dan b yang akan dijadikan
     operand */
  int a=25, b=12;

  /* Mendeklarasikan variabel c sebagai penampung nilai hasil
     pengurangan */
  int c;
  /* Melakukan operasi pengurangan */
  c = a - b;

  /* Menampilkan nilai hasil pengurangan */
  printf("%d - %d = %d", a, b, c);

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
25 - 12 = 13
```

*c. Menggunakan Operator **

```
#include <stdio.h>

int main(void) {
  /* Mendeklarasikan variabel a dan b yang akan dijadikan
     operand */
  int a=12, b=3;

  /* Mendeklarasikan variabel c sebagai penampung nilai hasil
     perkalian */
  int c;

  /* Melakukan operasi perkalian */
  c = a * b;

  /* Menampilkan nilai hasil perkalian */
  printf("%d x %d = %d", a, b, c);

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
12 x 3 = 36
```

d. Menggunakan Operator /

Dalam menggunakan operator `/`, kita harus memperhatikan tipe data dari *operand* yang kita definisikan. Pasalnya, perbedaan tipe data akan menyebabkan hasil yang berbeda pula. Apabila kita mendefinisikan *operand* dengan tipe data riil (`float` atau `double`), maka hasilnya juga berupa data riil. Misalnya `10.0 / 3.0`, maka hasil yang akan didapatkan adalah `3.3333`. Sedangkan apabila kita mendefinisikan *operand* dengan tipe data bilangan bulat (`int` atau `long`), maka nilai yang dihasilkan juga berupa bilangan bulat (tanpa memperdulikan sisa baginya). Misalnya `10 / 3`, maka hasil yang didapatkan adalah `3`. Adapun nilai 1 yang merupakan sisa bagi dari operasi pembagian tersebut akan diabaikan oleh program. Berikut ini contoh program yang akan membuktikan hal tersebut.

```
#include <stdio.h>

int main(void) {
  /* Mendeklarasikan variabel a, b dan c dengan tipe int */
  int a=10, b=3, c;

  /* Mendeklarasikan variabel x, y dan z dengan tipe double */
  double x = 10.0, y = 3.0, z;

  /* Melakukan operasi pembagian pada bilangan bulat */
```

```
  c = a / b;

  /* Melakukan operasi pembagian pada bilangan riil */
  z = x / y;

  /* Menampilkan nilai hasil operasi pembagian */
  printf("%d / %d \t\t = %d\n", a, b, c);
  printf("%.1f / %.1f \t = %.4f\n", x, y, z);

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
10 / 3          = 3
10.0 / 3.0      = 3.3333
```

e. Menggunakan Operator %

Operator `%` ini digunakan untuk mendapatkan sisa bagi dari operasi pembagian pada bilangan bulat. Misalnya `10 % 3`, maka hasilnya adalah `1`. Nilai tersebut merupakan sisa bagi yang diperoleh dari proses pembagian 10 dibagi 3. Berikut ini contoh program yang menunjukkan penggunaan operator `%`.

```
#include <stdio.h>

int main(void) {
  printf("%2d %s %d \t = %d\n", 10, "%", 3, (10 % 3));
  printf("%2d %s %d \t = %d\n", 8, "%", 3, (8 % 3));
  printf("%2d %s %d \t = %d\n", 15, "%", 4, (15 % 4));
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
10 % 3 = 1
 8 % 3 = 2
15 % 4 = 3
```

3.4.2. Operator Logika

Operator logika adalah operator digunakan di dalam operasi yang hanya dapat menghasilkan nilai benar (`true`) dan salah (`false`). Nilai seperti ini disebut dengan nilai boolean. Berbeda dengan bahasa pemrograman lain (misalnya Pascal), bahasa C

tidak menyediakan tipe data khusus untuk merepresentasikan nilai boolean tersebut. Dalam bahasa C, nilai benar akan direpresentasikan dengan menggunakan nilai selain nol. Namun, pada kenyataannya para programmer C umumnya menggunakan nilai 1 untuk merepresentasikan nilai benar. Adapun nilai salah akan direpresentasikan dengan nilai 0. Untuk memudahkan dalam proses penulisan sintak, maka biasanya nilai-nilai tersebut dijadikan sebagai makro dengan sintak berikut.

```
#define TRUE   1
#define FALSE  0
```

Adapun yang termasuk ke dalam operator logika dalam bahasa C adalah seperti yang tampak dalam tabel berikut.

Operator	**Jenis Operasi**	**Contoh Penggunaan**
&&	AND (dan)	`1 && 0 = 0`
||	OR (atau)	`1 || 0 = 1`
!	***NOT (negasi / ingkaran)***	`!1 = 0`

3.4.2.1. Operator && (AND)

Operasi AND akan memberikan nilai benar apabila semua *operand*-nya bernilai benar, selain itu operasi ini akan menghasilkan nilai salah. Berikut ini tabel yang menyatakan nilai yang dihasilkan dari operasi AND.

X	**Y**	**X && Y**
1	1	1
1	0	0
0	***1***	0
0	***0***	0

Berikut ini contoh program yang akan menunjukkan penggunaan operator &&.

```
#include <stdio.h>

int main(void) {
  printf("1 && 1 = %d\n", (1 && 1));
  printf("1 && 0 = %d\n", (1 && 0));
  printf("0 && 1 = %d\n", (0 && 1));
  printf("0 && 0 = %d\n", (0 && 0));
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
1 && 1 = 1
1 && 0 = 0
0 && 1 = 0
0 && 0 = 0
```

3.4.2.2. Operator || (OR)

Operasi OR akan menghasilkan nilai salah apabila semua *operand*-nya bernilai salah, selain itu operasi tersebut akan menghasilkan nilai benar. Berikut ini tabel yang menyatakan hasil dari operasi OR.

X	Y	X || Y
1	1	1
1	0	1
0	***1***	1
0	***0***	0

Berikut ini contoh program yang akan membuktikan hal tesebut

```
#include <stdio.h>

int main(void) {
  printf("1 || 1 = %d\n", (1 || 1));
  printf("1 || 0 = %d\n", (1 || 0));
  printf("0 || 1 = %d\n", (0 || 1));
  printf("0 || 0 = %d\n", (0 || 0));
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
1 || 1 = 1
1 || 0 = 1
0 || 1 = 1
0 || 0 = 0
```

3.4.2.3. Operator ! (NOT)

Operasi NOT merupakan operasi yang menghasilkan nilai ingkaran (negasi) dari nilai *operand* yang diberikan. Berikut ini tabel yang menyatakan hasil dari opearsi NOT.

X	**!X**
1	0
0	1

Adapun contoh program yang akan menunjukkan hal tersebut adalah sebagai berikut.

```
#include <stdio.h>

int main(void) {
  printf("!1 = %d\n", (!1));
  printf("!0 = %d\n", (!0));
  return 0;
}
```

Program di atas akan memberikan hasil seperti di bawah ini.

```
!1 = 0
!0 = 1
```

3.4.3. Operator Relasional

Operator relasional adalah operator yang digunakan untuk menentukan relasi atau hubungan dari dua buah nilai atau *operand*. Operator ini terdapat dalam sebuah ekspresi yang selanjutnya akan menentukan benar atau tidaknya ekspresi tersebut. Berikut ini operator yang termasuk ke dalam operator relasional dalam bahasa C.

Operator	**Jenis Operasi**	**Contoh Penggunaan**
==	Sama dengan	(8 == 8) = 1
>	Lebih besar	(8 > 9) = 0
<	***Lebih kecil***	(8 < 9) = 1
>=	Lebih besar atau sama dengan	(8 >= 7) = 0
<=	Lebih kecil atau sama dengan	(7 <= 8) = 1
!=	***Tidak sama dengan***	(8 != 9) = 1

Operator ini pada umumnya digunakan untuk melakukan pengecekan terhadap ekspresi di dalam blok pemilihan, yang baru akan kita bahas pada bab selanjutnya, yaitu bab 4 – *Kontrol Program*. Namun sebagai gambaran bagi Anda, berikut ini contoh program yang akan menunjukkan penggunaan salah satu dari operator di atas.

```
#include <stdio.h>

int main(void) {
```

```
  int x;
  printf("Masukkan sebuah bilangan bulat : "); scanf("%d", &x);

  /* Menggunakan operator relasional == */
  if (x % 2 == 0 ) {
    printf("%d adalah bilangan genap", x);
  } else {
    printf("%d adalah bilangan ganjil", x);
  }

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas dalah sebagai berikut.

```
Masukkan sebuah bilangan bulat : 5
5 adalah bilangan ganjil
```

3.4.4. Operator Bitwise

Operator bitwise digunakan untuk menyelesaikan operasi-operasi bilangan dalam bentuk biner yang dilakukan bit demi bit. Dengan kata lain, operator bitwise ini berfungsi untuk melakukan pemanipulasian bit. Operasi ini merupakan hal vital apabila program yang kita buat kan melakukan interaksi dengan perangkat keras (*hardware*). Meskipun bahasa pemrograman lebih bersifat *data-oriented*, namun perangkat keras masihlah bersifat *bit-oriented*. Ini artinya, perangkat keras menginginkan input dan output data yang dilakukan terhadapnya tetap dalam bentuk bit tersendiri.

Perlu ditekankan di sini bahwa operasi pemanipulasian bit ini hanya dapat dilakukan pada bilangan-bilangan yang bertipe `char` dan `int` saja karena keduanya dapat berkoresponden dengan tipe `byte` dan `word` di dalam bit. Adapun yang termasuk ke dalam operator bitwise di dalam bahasa C adalah seperti yang tertera dalam tabel di bawah ini.

Operator	**Jenis Operasi**	**Contoh Penggunaan**
`&`	Bitwise AND	`1 & 1 = 1`
`|`	Bitwise OR	`1 | 0 = 1`
`^`	***Bitwise XOR (Exclusive OR)***	`1 ^ 1 = 0`
`~`	Bitwise Complement (NOT)	`~0 = 1`
`>>`	***Shift right (geser kanan)***	`4 << 1 = 8`
`<<`	Shift left (geser kiri)	`4 >> 1 = 2`

Fungsi dari operator `&`, `|` dan `~` di atas sebenarnya sama dengan fungsi operator logika `&&`, `||` dan `!`. Perbedaannya hanya operator *bitwise* ini melakukan operasinya bit demi

bit, sedangkan operator logika melakukan operasi pada nilai totalnya. Sebagai contoh apabila kita melakukan operasi logika `7 || 8`, maka hasil yang akan didapatkan adalah `1`, pasalnya nilai 7 dan 8 akan dianggap sebagai nilai benar (`true`) sehingga operasi OR tersebut juga akan menghasilkan nilai `true` yang direpresentasikan dengan nilai 1. Namun, jika kita melakukan operasi bitwise `7 | 8`, maka nilai 7 dan 8 tersebut akan dikonversi ke dalam bilangan biner, setelah itu baru dilakukan operasi OR untuk setiap bit-nya. Proses ini dapat kita representasikan dengan cara berikut.

```
0 0 0 0 1 0 0 0     nilai 8 dalam bentuk biner
0 0 0 0 0 1 1 1     nilai 7 dalam bentuk biner
----------------- |
0 0 0 0 1 1 1 1     hasil = 15
```

Cara kerja dari operator `&` dan `~` juga sama seperti di atas. Untuk itu di sini kita tidak akan membahas lebih detil tentang kedua operator tersebut. Adapun operator lain yang perlu Anda ketahui di sini adalah operator `^` (*bitwise XOR*), `>>` (*shift right*) dan `<<` (*shift left*). Penjelasan dari masing-masing operator tersebut dapat Anda lihat dalam sub bab di bawah ini.

3.4.4.1. Operator ^ (Exclusive OR)

Operasi XOR (*exlusive OR*) akan memberikan nilai benar apabila hanya terdapat satu buah *operand* yang bernilai benar, selain itu akan menghasilkan nilai salah. Dengan demikian, apabila kedua *operand*-nya bernilai benar, operasi ini tetap akan menghasilkan nilai salah. Berikut ini tabel yang menunjukkan hasil dari operasi XOR.

X	**Y**	**X ^ Y**
1	1	0
1	0	1
0	***1***	1
0	***0***	0

Sebagai contoh apabila kita ingin melakukan operasi `45 ^ 23`, maka hasilnya adalah `58`. Berikut ini proses yang menunjukkan operasi tersebut.

```
0 0 1 0 1 1 0 1     nilai 45 dalam bentuk biner
0 0 0 1 0 1 1 1     nilai 23 dalam bentuk biner
----------------- ^
0 0 1 1 1 0 1 0     hasil = 58
```

Anda dapat membuktikannya ke dalam sebuah program sederhana seperti di bawah ini.

```
#include <stdio.h>

#define X  0x2D /* nilai 45 dalam bentuk heksadesimal */
#define Y  0x17 /* nilai 23 dalam bentuk heksadesimal */
```

```
int main(void) {
  printf("%d ^ %d = %d", (X ^ Y));
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
45 ^ 23 = 58
```

3.4.4.2. Operator >> (Shift Right)

Operator *shift right* (geser kanan) ini digunakan untuk melakukan penggeseran bit ke arah kanan sebanyak nilai yang didefinisikan. Apabila terdapat operasi `X >> 3` berarti melakukan penggeseran 3 bit ke kanan dari nilai `X` yang telah dikonversi ke dalam bilangan biner. Adapun bentuk umum dari penggunaan operator `>>` adalah sebagai berikut.

```
nilai >> banyaknya_pergeseran_bit_ke_arah_kanan
```

Untuk memudahkan Anda dalam menentukan hasil yang diberikan dari operasi ini, ingatlah bahwa setiap proses pergeseran bit yang terjadi, operator `>>` akan membagi suatu nilai dengan 2. Sebagai contoh `128 >> 1`, maka hasil yang akan didapatkan adalah `64`. Sedangkan `128 >> 2` akan menghasilkan nilai `32`, begitu seterusnya.
Berikut ini contoh program yang akan membuktikan hal tersebut.

```
#include <stdio.h>

#define X 0x80  /* nilai 128 dalam bentuk heksadesimal */

int main(void) {
  printf("%d >> 1 = %d\n", X, (X>>1));
  printf("%d >> 2 = %d\n", X, (X>>2));
  printf("%d >> 3 = %d\n", X, (X>>3));
  printf("%d >> 4 = %d\n", X, (X>>4));
  printf("%d >> 5 = %d\n", X, (X>>5));
  printf("%d >> 6 = %d\n", X, (X>>6));
  printf("%d >> 7 = %d\n", X, (X>>7));

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```

```

```
128 >> 1 = 64
128 >> 2 = 32
128 >> 3 = 16
128 >> 4 = 8
128 >> 5 = 4
128 >> 6 = 2
128 >> 7 = 1
```

Berikut ini tabel yang akan mengilustrasikan proses yang terjadi dalam program di atas.

Nilai X	X dalam bentuk bilangan biner	Hasil
X = 128	**1** 0 0 0 0 0 0 0	128
X = 128 >> 1	0 **1** 0 0 0 0 0 0	64
X = 128 >> 2	0 0 **1** 0 0 0 0 0	32
X = 128 >> 3	0 0 0 **1** 0 0 0 0	16
X = 128 >> 4	0 0 0 0 **1** 0 0 0	8
X = 128 >> 5	0 0 0 0 0 **1** 0 0	4
X = 128 >> 6	0 0 0 0 0 0 **1** 0	2
X = 128 >> 7	0 0 0 0 0 0 0 **1**	1

3.4.4.3. Operator << (Shift Left)

Operator *shift left* (geser kiri) merupakan kebalikan dari operator >>, artinya di sini kita melakukan pergeseran bit ke arah kiri sebanyak nilai yang didefinisikan. Berikut ini bentuk umum penggunaan operator <<.

```
nilai << banyaknya_pergeseran_bit_ke_arah_kiri
```

Dalam setiap pergeseran bit-nya, operator ini akan mengalikan suatu nilai dengan 2. Misalnya `1 >> 1`, maka hasil yang akan didapatkan adalah 2 (berasal dari 1 x 2). Sedangkan `1 >> 2` akan memberikan hasil 4 (berasal dari 1 x 2 x 2), `1 >> 3` memberikan hasil 8 (berasal dari 1 x 2 x 2 x 2), begitu seterusnya. Perhatikan program di bawah ini yang akan membuktikan hal tersebut.

```
#include <stdio.h>

#define X 0x01  /* nilai 1 dalam bentuk heksadesimal */

int main(void) {
  printf("%d << 1 = %d\n", X, (X<<1));
  printf("%d << 2 = %d\n", X, (X<<2));
  printf("%d << 3 = %d\n", X, (X<<3));
  printf("%d << 4 = %d\n", X, (X<<4));
  printf("%d << 5 = %d\n", X, (X<<5));
  printf("%d << 6 = %d\n", X, (X<<6));
  printf("%d << 7 = %d\n", X, (X<<7));

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
1 << 1 = 2
1 << 2 = 4
1 << 3 = 8
1 << 4 = 16
1 << 5 = 32
1 << 6 = 64
1 << 7 = 128
```

Untuk mengilustrasikan proses yang terjadi dalam program atas, perhatikanlah tabel di bawah ini.

Nilai X	X dalam bentuk bilangan biner	Hasil
X = 1	0 0 0 0 0 0 0 **1**	1
X = 1 << 1	0 0 0 0 0 0 **1** 0	2
X = 1 << 2	0 0 0 0 0 **1** 0 0	4
X = 1 << 3	0 0 0 0 **1** 0 0 0	8
X = 1 << 4	0 0 0 **1** 0 0 0 0	16
X = 1 << 5	0 0 **1** 0 0 0 0 0	32
X = 1 << 6	0 **1** 0 0 0 0 0 0	64
X = 1 << 7	**1** 0 0 0 0 0 0 0	128

3.5. Operator *Ternary*

Berbeda dengan bahasa pemrograman lainnya, bahasa C menyediakan operator *ternary*, yaitu operator yang melibatkan tiga buah operand. Operator ini dilambangkan dengan tanda `?:` serta berguna untuk melakukan pemilihan terhadap nilai tertentu dimana pemilihan tersebut didasarkan atas ekspresi tertentu. Adapun bentuk umum dari penggunaan operator *ternary* ini adalah sebagai berikut.

```
ekspresi1 ? ekspresi2 : ekspresi3;
```

Apabila `ekspresi1` bernilai benar maka program akan mengeksekusi `ekspresi2`, sedangkan apabila bernilai salah maka yang akan dieksekusi adalah `ekspresi3`.

Berikut ini contoh program yang akan menunjukkan penggunaan operator *ternary* tersebut.

```
#include <stdio.h>

int main(void) {
  /* Mendeklarasikan variabel yang diperlukan */
  int a, b, abs_a, maks;
```

```
  /* Meminta masukan dari user untuk mengisikan nilai a dan b*/
  printf("Masukkan nilai a : "); scanf("%d", &a);
  printf("Masukkan nilai b : "); scanf("%d", &b);

  /* Menggunakan operator ?: untuk menentukan nilai mutlak
     (absolut) dari variabel a dan memasukkan hasilnya ke dalam
     variabel abs_a */
  abs_a = (a > 0) ? a : (-a);

  /* Menggunakan operator ?: untuk menentukan nilai maksimal
     dari variabel a dan b, selanjutnya memasukkan hasilnya ke
     dalam variabel maks */
  maks = (a > b) ? a : b;

  /* Menampilkan nilai yang dikandung dalam variabel abs_a dan
     maks */
  printf("\nNilai absolut dari %d \t\t = %d\n", a, abs_a);
  printf("Nilai maksimal dari %d dan %d \t = %d\n", a, b, maks);

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Masukkan nilai a : -15
Masukkan nilai b : 20

Nilai absolut dari –15          = 15
Nilai maksimal dari –15 dan 20  = 20
```

Sebagai tambahan bagi Anda, apabila program di atas akan ditulis dalam bentuk struktur `if` (*tidak menggunakan operator ?:*), maka sintaknya menjadi seperti di bawah ini.

```
#include <stdio.h>

int main(void) {
  /* Mendeklarasikan variabel yang diperlukan */
  int a, b, abs_a, maks;

  /* Meminta masukan dari user untuk mengisikan nilai a dan b*/
  printf("Masukkan nilai a : "); scanf("%d", &a);
  printf("Masukkan nilai b : "); scanf("%d", &b);

  /* Menentukan nilai mutlak (absolut) dengan struktur if */
  if (a > 0) {
    abs_a = a;
  } else {
```

```
    abs_a = (-a);
  }

  /* Menentukan nilai maksimal dari variabel a dan b dengan
     struktur if */
  if (a > b) {
    maks = a;
  } else {
    maks = b;
  }

  /* Menampilkan nilai yang dikandung dalam variabel abs_a dan
     maks */
  printf("\nNilai absolut dari %d \t\t = %d\n", a, abs_a);
  printf("Nilai maksimal dari %d dan %d \t = %d\n", a, b, maks);

  return 0;
}
```

Bab

4

Kontrol Program

4.1. Pendahuluan

Setelah Anda memahami tentang variabel dan jenis-jenisnya, tipe data, operator serta ekspresi yang telah dibahas pada bab sebelumnya, maka sekarang tiba saatnya Anda mengetahui bagaimana cara melakukan kontrol terhadap program di dalam bahasa C.

Untuk dapat membuat suatu program dengan bahasa C secara baik dan benar tentu kita harus mengetahui terlebih dahulu aturan-aturan pemrograman yang berlaku, seperti melakukan pemilihan statemen yang didasarkan atas kondisi tertentu maupun melakukan pengulangan statemen di dalam program. Melalui bab ini, diharapkan Anda dapat memahami konsep pengontrolan program dalam bahasa C dan mampu mengimplementasikannya di dalam kasus-kasus program yang Anda hadapi.

Selain itu, dalam bab ini juga kita akan banyak melakukan penulisan contoh-contoh program, hal ini bertujuan untuk memudahkan Anda dalam memahami konsep yang akan disampaikan serta untuk menambah pembendaharaan kasus-kasus program bagi Anda yang merupakan pemula di dalam dunia pemrograman. Untuk dapat lebih memahami konsep dari alur program secara rinci, penulis merekomendasikan agar Anda membaca buku '*Algoritma dan Pemrograman dalam bahasa Pascal dan C (Buku 1)*' yang ditulis oleh Ir. Rinaldi Munir dan diterbitkan oleh CV. Informatika Bandung.

4.2. Pemilihan

Dalam kehidupan sehari-hari, kadang kala kita disudutkan pada beberapa pilihan dimana pilihan-pilihan tersebut didasarkan atas kondisi tertentu. Dengan kata lain, pilihan tersebut hanya dapat dilakukan apabila kondisi telah terpenuhi. Sebagai contoh, perhatikan statemen di bawah ini.

Jika Gunawan memiliki banyak uang, maka ia akan membeli mobil mewah

Pada statemen di atas, Gunawan akan dapat membeli mobil mewah hanya apabila ia memiliki banyak uang. Hal ini berarti apabila ternyata Gunawan tidak memiliki banyak uang (kondisi tidak terpenuhi), maka Gunawan pun tidak akan pernah membeli mobil mewah.

Begitupun di dalam bahasa pemrograman, kita juga dapat melakukan pemilihan statemen yang akan dieksekusi, yaitu dengan melakukan pengecekan terhadap kondisi

tertentu yang didefinisikan. Adapun kondisi yang dimaksud di dalam program tidak lain adalah suatu ekspresi.

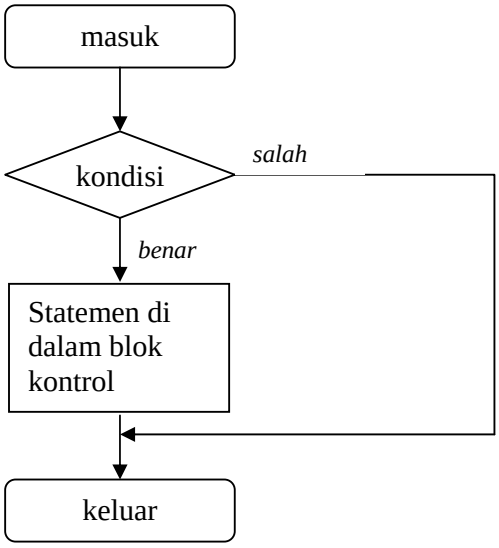
Dalam bahasa C, pemilihan statemen dapat dilakukan melalui dua buah cara, yaitu dengan menggunakan statemen `if` dan statemen `switch`.

4.2.1. Statemen `if`

Untuk memudahkan pembahasan, di sini kita akan mengklasifikasikan pemilihan dengan menggunakan statemen `if` tersebut ke dalam tiga bagian, yaitu pemilihan yang didasarkan atas satu kasus, dua kasus dan lebih dari dua kasus.

4.2.1.1. Satu Kasus

Pemilihan jenis ini adalah pemilihan yang paling sederhana karena hanya mengandung satu kondisi yang akan diperiksa. Berikut ini gambar yang akan menunjukkan konsep dari pemilihan yang didasarkan atas satu kasus.



Gambar 4.1. Statemen `if` untuk satu kasus

Pada gambar di atas terlihat bahwa mula-mula program akan mengecek kondisi yang didefinisikan. Apabila kondisi bernilai benar (kondisi terpenuhi), maka program akan melakukan statemen-statemen yang terdapat di dalam blok pengecekan. Namun apabila ternyata kondisi bernilai salah (kondisi tidak terpenuhi), maka program akan langsung keluar dari blok pengecekan dengan melanjutkan eksekusi terhadap statemen-statemen berikutnya di luar blok pengecakan (jika ada).

Adapun bentuk umum atau kerangka dari blok pemilihan menggunakan statemen `if` untuk satu kasus di dalam bahasa C adalah seperti yang tampak di bawah ini.

```
if (kondisi)
    Statemen_yang_akan_dieksekusi;
```

Bentuk umum di atas berlaku apabila Anda hanya memiliki sebuah statemen di dalam blok pengecekan. Namun, apabila Anda memiliki dua statemen atau lebih, maka bentuk umumnya menjadi seperti di bawah ini.

```
if (kondisi) {
     Statemen_yang_akan_dieksekusi1;
     Statemen_yang_akan_dieksekusi2;
     ...
}
```

Perlu sekali untuk diperhatikan bahwa dalam bahasa C, kondisi harus diapit oleh tanda kurung. Selain itu bahasa C juga tidak memiliki kata kunci `then` seperti yang terdapat pada kebanyakan bahasa pemrograman lainnya, misalnya bahasa Pascal.

Untuk lebih memahami konsep yang terdapat di dalamnya, perhatikan program berikut.

```
#include <stdio.h>

int main(void) {
  int x;

  /* Meminta masukan nilai yang akan ditampung ke dalam
     variabel x */
  printf("Masukkan sebuah bilangan bulat : "); scanf("%d", &x);

  /* Melakukan pengecekan terhadap nilai x yang telah
     dimasukkan */
  if (x > 0)
    printf("\n%d adalah bilangan positif\n", x);

  printf("Statemen di luar blok kontrol pengecekan");
  return 0;
}
```

Coba Anda lakukan kompilasi dan jalankan program tersebut, kemudian masukkan nilai `x` dengan nilai 10, maka Anda akan melihat hasil sebagai berikut.

```
Masukkan sebuah bilangan bulat : 10

10 adalah bilangan positif
Statemen di luar blok kontrol pengecekan
```

Dari hasil di atas dapat kita lihat bahwa nilai `x` sama dengan 10 dan ekspresi (10 > 0) bernilai benar. Hal ini tentu menyebabkan statemen di dalam blok pengecekan akan dieksekusi oleh program.

Namun apabila Anda memasukkan nilai `x` dengan nilai 0 atau negatif (misalnya –5), maka hasil yang akan diberikan adalah sebagai berikut.

```
Masukkan sebuah bilangan bulat : 10
Statemen di luar blok kontrol pengecekan
```

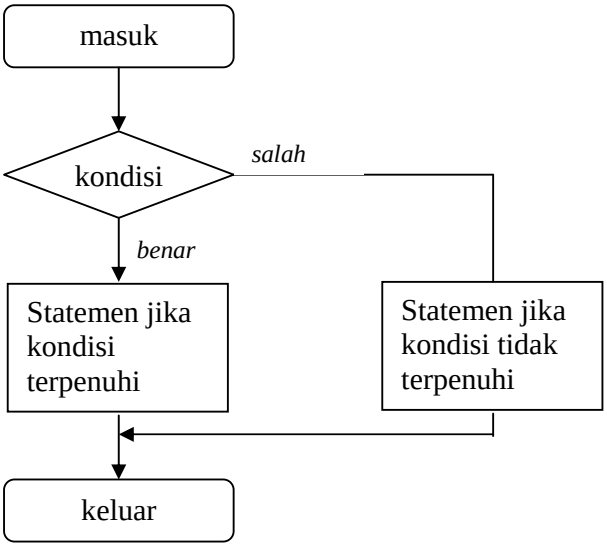
Sekarang, statemen di dalam blok pengecekan tidak ikut dieksekusi. Hal ini disebabkan karena ekspresi (-5 > 0) bernilai salah sehingga program akan langsung keluar dari blok pengecekan.

Sebagai contoh penggunaan statemen `if`, di sini kita akan membuat program untuk menentukan apakah suatu tahun merupakan tahun kabisat atau bukan. Adapun sintaknya adalah sebagai berikut.

```
#include <stdio.h>

int main(void) {
  long tahun;
  printf("Masukkan tahun yang akan diperiksa : ");
  scanf("%ld", &tahun);

  /* Mengecek tahun kabisat */
  if ((tahun % 4 == 0) &&
      ((tahun % 100 != 0) || (tahun % 400 = 0)))
    printf("\n%ld merupakan tahun kabisat", tahun);
  return 0;
}
```

4.2.1.2. Dua Kasus

Bentuk pemilihan ini merupakan perluasan dari bentuk pertama, hanya saja di sini didefinisikan pula statemen yang akan dilakukan apabila kondisi yang diperiksa bernilai salah (tidak terpenuhi). Adapun cara yang digunakan untuk melakukan hal tersebut adalah dengan menambahkan kata kunci `else` di dalam blok pengecekan. Hal ini menyebabkan statemen untuk pemilihan untuk dua kasus sering dikenal dengan statemen `if-else`. Berikut ini gambar yang akan menunjukkan konsep dari pemilihan yang didasarkan atas dua kasus.



Gambar 4.2. Statemen `if` untuk dua kasus

Dari gambar di atas terlihat jelas bahwa pada pemilihan untuk dua kasus, apabila kondisi yang diperiksa tidak terpenuhi maka terlebih dahulu program akan mengeksekusi sebuah (atau lebih) statemen sebelum program melanjutkan eksekusi ke statemen-statemen berikutnya di luar atau setelah blok pengecekan.

Bentuk umum atau kerangka yang digunakan dalam bahasa C untuk melakukan pemilihan dua kasus adalah sebagai berikut.

```
if (kondisi)
  Statemen_jika_kondisi_benar; /* Ingat, harus menggunakan
                                  tanda titik koma */
else
  Statemen_jika_kondisi_salah;
```

Bentuk umum di atas dilakukan apabila statemen yang kita definisikan untuk sebuah nilai kondisi tertentu (benar atau salah) hanya terdiri dari satu statemen. Namun apabila kita akan mendefinisikan lebih dari satu statemen, maka bentuk umumnya adalah sebagai berikut.

```
if (kondisi) {
  Statemen_jika_kondisi_benar1;
  Statemen_jika_kondisi_benar2;
  …
} else {
  Statemen_jika_kondisi_salah1;
  Statemen_jika_kondisi_salah2;
  …
}
```

Sebagai contoh termudah untuk menunjukkan penggunaan statemen `if-else` di dalam program adalah untuk menentukan suatu bilangan bulat apakah merupakan bilangan genap atau ganjil. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>

int main(void) {
  /* Mendeklarasikan variabel untuk menampung nilai yang akan
     diperiksa */
  int x;

  /* Menampilkan teks sebagai informasi bagi user */
  printf("Masukkan bilangan bulat yang akan diperiksa : ");
  /* Meminta user untuk memasukkan nilai ke dalam variabel x */
  scanf("%d", &x);

  /* Melakukan pengecekan terhadap bilangan yang dimasukkan ke
     variabel x */
  if (x % 2 == 0)
    printf("%d merupakan bilangan GENAP", x);
  else
    printf("%d merupakan bilangan GANJIL", x);

  return 0;
}
```

Contoh hasil yang akan diberikan dari program tersebut adalah sebagai berikut.

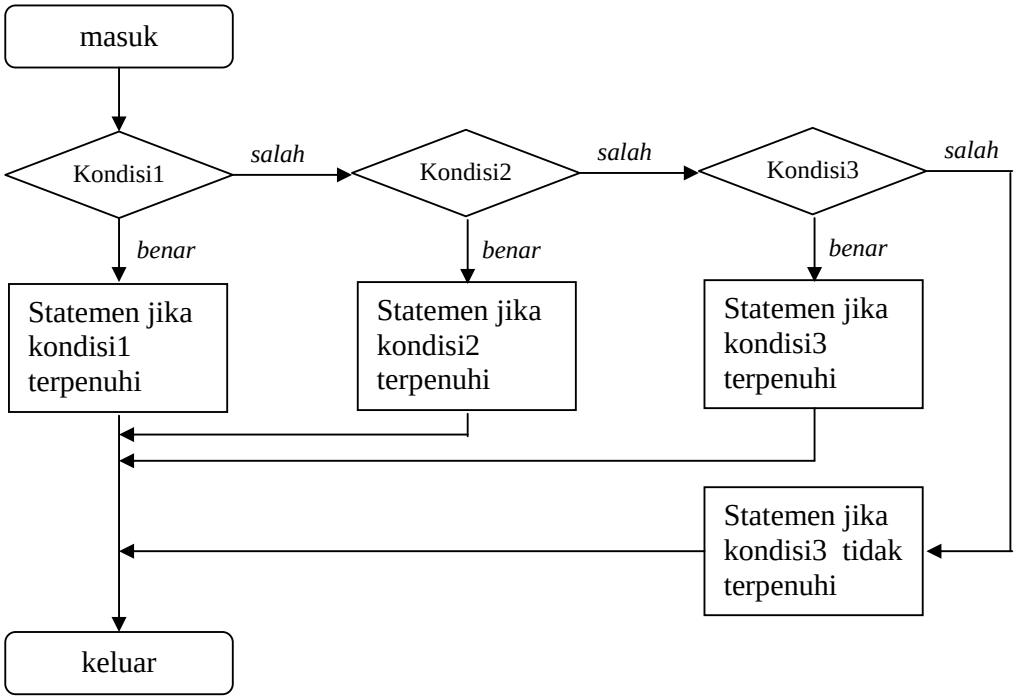
```
Masukkan bilangan bulat yang akan diperiksa : 5
5 merupakan bilangan GANJIL
```

Perhatikan kembali sintak program di atas, di sana tertulis ekspresi `(x % 2 == 0)` yang berarti melakukan pengecekan terhadap suatu bilangan yang dibagi dengan 2, apakah sisa baginya sama dengan 0 atau tidak. Apabila bilangan tersebut habis dibagi 2 (artinya sisa = 0) maka bilangan tersebut merupakan bilangan genap, sebaliknya apabila sisa bagi = 1 maka bilangan tersebut termasuk ke dalam bilangan ganjil. Pada kasus ini, kita melakukan pengecekan terhadap bilangan 5. Kita tahu bahwa 5 dibagi 2 menghasilkan nilai 2 dan sisa baginya 1, ini berarti ekspresi di atas bernilai salah (tidak terpenuhi) dan menyebabkan program akan mengeksekusi statemen yang berada pada bagian `else`, yaitu statemen di bawah ini.

```
printf("%d merupakan bilangan GANJIL", x);
```

4.2.1.3. Lebih dari Dua Kasus

Pada pemilihan jenis ini kita diizinkan untuk menempatkan beberapa (lebih dari satu) kondisi sesuai dengan kebutuhan program yang akan kita buat. Berikut ini gambar yang akan menunjukkan konsep dari pemilihan statemen yang didasarkan atas tiga kasus atau lebih.



Gambar 4.3. Statemen `if` untuk tiga kasus atau lebih

Pada gambar di atas, mula-mula program akan melakukan pengecekan terhadap kondisi1. Apabila kondisi1 benar, maka program akan langsung mengeksekusi statemen yang didefinisikan di dalamnya. Namun, apabila kondisi1 bernilai salah maka program akan melakukan pengecekan terhadap kondisi2. Apabila kondisi2 juga bernilai salah maka program akan melanjutkan ke pengecekan kondisi3. Apabila ternyata kondisi3 juga bernilai salah maka program akan mengeksekusi statemen alternatif yang didefinisikan, yaitu statemen yang terdapat pada bagian akhir blok pengecekan (pada bagian **else**). Adapun bentuk umum dari pemilihan yang melibatkan tiga buah kasus atau lebih adalah sebagai berikut.

```
if (kondisi1) {
  Statemen_yang_akan_dieksekusi;
  …
} else if (kondisi2) {
  Statemen_yang_akan_dieksekusi;
  …
} else {
  Statemen_alternatif;   /* Apabila semua kondisi di atas
                            tidak terpenuhi */
}
```

Sebagai contoh sederhana untuk mengimplementasikan pemilihan yang didasarkan atas tiga kasus adalah pembuatan program untuk menentukan wujud air yang berada pada suhu tertentu. Adapun ketentuan-ketentuannya adalah sebagai berikut.

suhu ≤ 0	air akan berwujud padat (es)
0 < suhu < 100	air akan berwujud cair
suhu ≥ 100	air akan berwujud gas

Apabila dituliskan dalam bentuk program, maka sintaknya adalah seperti yang tertera di bawah ini.

```
#include <stdio.h>

int main(void) {
  int suhu;

  printf("Masukkan besarnya suhu : "); scanf("%d", suhu);

  /* Melakukan pengecekan terhadap suhu */
  if (suhu <= 0) {
    printf("Pada suhu %d derajat Celcius, air akan berwujud " \
           "padat (es)", suhu);
  } else if ((suhu > 0) && (suhu < 100)) {
    printf("Pada suhu %d derajat Celcius, air akan berwujud " \
           "cair", suhu);
  } else {
    printf("Pada suhu %d derajat Celcius, air akan berwujud " \
           "gas", suhu);
  }

  return 0;
}
```

Berikut ini contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Masukkan besarnya suhu : 28
Pada suhu 28 derajat Celcius, air akan berwujud cair
```

Contoh lain yang dapat kita ambil untuk memperjelas materi ini adalah pembuatan program untuk menentukan akar-akar dari persamaan kuadrat, yaitu dengan menggunakan rumus ABC. Kita semua telah mengetahui bahwa bentuk umum dari persamaan kuadrat adalah $y = ax^2 + bx + c$. Untuk menentukan akar-akar dari persamaan kuadrat tersebut sebelumnya kita harus menentukan nilai determinan terlebih dahulu, yaitu dengan rumus $D = b – 4ac$.

Setelah nilai determinan didapatkan, selanjutnya terdapat tiga buah kemungkinan, yaitu sebagai berikut:

- Apabila D > 0, maka akar-akar persamaan kuadrat bersifat riil dan berbeda. Adapun formula untuk mendapatkannya adalah sebagai berikut:

  $x_1 = (-b + \sqrt{D}) / 2a$
  $x_2 = (-b - \sqrt{D}) / 2a$

- Apabila D = 0, maka akar-akar persamaan kuadrat bersifat riil dan sama. Di sini artinya $x_1 = x_2$.

- Apabila D < 0, maka akar-akar persamaan kuadrat bersifat imaginer.

Adapun sintak program yang merupakan implementasi dari kasus di atas adalah seperti yang tertulis di bawah ini.

```
#include <stdio.h>
#include <math.h>     /* untuk menggunakan fungsi sqrt() */

#define TRUE    1
#define FALSE   0

int main(void) {
  /* Mendeklarasikan variabel penampung koefisien persamaan
     kuadrat */
  int a, b, c;
  /* Mendeklarasikan variabel untuk penampung nilai D,
     x1 dan x2 */
  float D, x1, x2;
  int status;     /* sebagai penentu status apakah akar-akar ada
                     atau imaginer */

  /* Meminta user untuk memasukkan koefisien persamaan kuadrat*/
  printf("Masukkan nilai a : "); scanf("%d", &a);
  printf("Masukkan nilai b : "); scanf("%d", &b);
  printf("Masukkan nilai c : "); scanf("%d", &c);

  /* Menghitung nilai determinan */
  D = (b*b) - (4*a*c) ;

  /* Melakukan pengecekan terhadap nilai D */
  if (D > 0) {
    x1 = ((-b) + sqrt(D)) / (2*a);
    x2 = ((-b) - sqrt(D)) / (2*a);
    status = TRUE;
  } else if (D == 0) {
    x1 = ((-b) + sqrt(D)) / (2*a);
    x2 = x1;
    status = TRUE;
  } else {
    status = FALSE;
  }
```

```
  /* Mengecek nilai dari status untuk menampilkan nilai yang
     didapatkan */
  printf("\nAkar-akar persamaan yang didapatkan:\n");
  if (status) {
    printf("x1 = %.2f\n", x1);
    printf("x2 = %.2f\n", x2);
  } else {
    printf("x1 dan x2 imaginer");
  }
  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Masukkan nilai a : 1
Masukkan nilai b : -4
Masukkan nilai c : 4

Akar-akar persamaan yang didapatkan:
x1 = 2.00
x2 = 2.00
```

Dari hasil di atas, tampak jelas bahwa nilai D = $(-4)^2$ – (4.1.4) = 0. Hal ini tentu akan menyebabkan akar-akar persamaan kuadrat yang dihasilkan adalah sama.

4.2.2. Statemen `switch`

Statemen `switch` digunakan untuk melakukan pemilihan terhadap ekspresi atau kondisi yang memiliki nilai-nilai konstan. Oleh karena itu, ekspresi yang didefinisikan harus menghasilkan nilai yang bertipe bilangan bulat atau karakter. Untuk mendefinisikan nilai-nilai konstan tersebut adalah dengan menggunakan kata kunci `case`. Hal yang perlu Anda perhatikan juga dalam melakukan pemilihan dengan menggunakan statamen `switch` ini adalah kita harus menambahkan statemen `break` pada setiap nilai yang kita definisikan.

Untuk lebih memahaminya, coba Anda perhatikan bentuk umum dari statemen `switch` di bawah ini.

```
switch (ekspresi) {
  case nilai_konstan1:
  {
    Statemen_yang_akan_dieksekusi;
    …
    break;
  }
  case nilai_konstan2:
  {
    Statemen_yang_akan_dieksekusi;
    …
    break;
  }
  …
  default:
  {
    Statemen_alternatif;    /* apabila semua nilai di
                               atas tidak terpenuhi */
  }
}
```

Kata kunci `default` di atas berguna untuk menyimpan statemen alternatif, yang akan dieksekusi apabila semua nilai yang didefinisikan tidak ada yang sesuai dengan ekspresi ada.

Berikut ini contoh program yang akan menunjukkan penggunaan statemen `switch` untuk melakukan suatu pemilihan nilai.

```
#include <stdio.h>

int main(void) {
  /* Mendeklarasikan variabel untuk menampung nomor hari */
  int nohari;
  /* Meminta user untuk memasukkan nomor hari */
  printf("Masukkan nomor hari (1-7): "); scanf("%d", &nohari);

  switch (nohari) {
    case 1: printf("Hari ke-%d adalah hari Minggu", nohari);
            break;
    case 2: printf("Hari ke-%d adalah hari Senin", nohari);
            break;
    case 3: printf("Hari ke-%d adalah hari Selasa", nohari);
            break;
    case 4: printf("Hari ke-%d adalah hari Rabu", nohari);
            break;
    case 5: printf("Hari ke-%d adalah hari Kamis", nohari);
            break;
    case 6: printf("Hari ke-%d adalah hari Jumat", nohari);
            break;
    case 7: printf("Hari ke-%d adalah hari Sabtu", nohari);
            break;
```

```
    default: printf(“Nomor hari yang dimasukkan salah”);
  }

  return 0;
}
```

Program di atas berguna untuk menentukan nama hari dari nomor hari yang dimasukkan. Adapun contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Masukkan nomor hari : 6
Hari ke-6 adalah hari Jumat
```

Sebagai contoh lain agar Anda dapat lebih memahaminya, perhatikan juga program di bawah ini yang akan mengecek nilai bertipe karakter. Adapun progam yang akan kita buat di sini adalah program yang akan menentukan nilai dari suatu operasi aritmetika.

```
#include <stdio.h>

int main(void) {
  int operand1, operand2;
  char optr;          /* variabel untuk menampung operator yang
                         akan digunakan */
  printf(“Masukkan operator yang diinginkan \t: ”);
  scanf(“%c”, &optr);
  printf(“Masukkan nilai untuk operand ke-1 \t: ”);
  scanf(“%d”, &operand1);
  printf(“Masukkan nilai untuk operand ke-2 \t: ”);
  scanf(“%d”, &operand2);

  switch (optr) {
    case ‘+’:
    {
      printf(“%d + %d = %d”,
             operand1, operand2, (operand1 + operand2));
      break;
    }
    case ‘-’:
    {
      printf(“%d - %d = %d”,
             operand1, operand2, (operand1 - operand2));
      break;
    }
    case ‘*’:
    {
      printf(“%d * %d = %d”,
             operand1, operand2, (operand1 * operand2));
      break;
    }
```

```
    case ‘/’:
    {
      printf(“%d / %d = %d”,
             operand1, operand2, (operand1 / operand2));
      break;
    }
    case ‘%’:
    {
      printf(“%d %% %d = %d”,
             operand1, operand2, (operand1 % operand2));
      break;
    }
  }  /* akhir switch */

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Masukkan operator yang diinginkan  : %
Masukkan nilai untuk operand ke-1   : 10
Masukkan nilai untuk operand ke-2   : 8
10 % 8 = 2
```

Apabila kita ingin mendefinisikan satu blok statemen yang dapat digunakan untuk beberapa nilai konstan, maka kita dapat menuliskan sintaknya seperti di bawah ini.

```
switch (ekspresi) {
  case nilai1:
  case nilai2:
  case nilai3:
  {
    /* Statemen yang berlaku untuk nilai1, nilai2 dan nilai3 */
    break;
  }
  case nilai4:
  case nilai5;
  {
    /* Statemen yang berlaku untuk nilai4 dan nilai5 */
    break;
  }
  ...
}
```

Untuk lebih jelasnya, di sini ini kita akan membuat program untuk menentukan bulan tertentu masuk ke dalam caturwulan ke berapa. Sebelumnya kita asumsikan bahwa bulan 1-4 (Januari sampai April) termasuk ke dalam caturwulan 1, bulan 5-8 (Mei sampai Agustus) termasuk ke dalam caturwulan 2 dan bulan 9-12 (September sampai

Desember) masuk ke dalam caturwulan 3. Adapun contoh sintak untuk mengimplementasikan kasus ini adalah seperti di bawah ini.

```
#include <stdio.h>

/* Mendeklarasikan array konstan untuk nama bulan */
const char namabulan[][12] =
             {“Januari”,”Februari”,”Maret”,”April”,
              “Mei”,”Juni”,”Juli”,”Agustus”,
              “September”,”Oktober”,”November”,”Desember”};

int main(void) {
  int nobulan;
  printf(“Masukkan nomor bulan (1-12) : ”);
  scanf(“%d”, &nobulan);

  switch (nobulan) {
    case 1: case 2: case 3: case 4:
      printf(“Bulan %s termasuk ke dalam caturwulan 1”,
             namabulan[nobulan-1]);
      break;
    case 5: case 6: case 7: case 8:
      printf(“Bulan %s termasuk ke dalam caturwulan 2”,
             namabulan[nobulan-1]);
      break;
    case 9: case 10: case 11: case 12:
      printf(“Bulan %s termasuk ke dalam caturwulan 3”,
             namabulan[nobulan-1]);
      braek;
    default: printf(“Nomor bulan yang dimasukkan salah”);
  }

  return 0;
}
```

Apabila program di atas dijalankan dan kita memasukkan nomor bulan dengan nilai 1, 2, 3 atau 4 maka statemen yang akan dieksekusi adalah statamen yang didefinisikan untuk blok nilai-nilai tersebut. Begitu juga dengan nilai 5, 6, 7 dan 8 serta nilai 9, 10, 11 dan 12. Berikut ini contoh hasil yang akan diberikan dari program di atas.

```
Masukkan nomor bulan : 10
Bulan Oktober termasuk ke dalam caturwulan 3
```

4.3. Pengulangan

Dalam pembuatan program, terkadang kita harus melakukan pengulangan suatu aksi, misalnya untuk melakukan perhitungan berulang dengan menggunakan formula yang sama. Sebagai contoh, misalnya kita ingin membuat program yang dapat menampilkan teks `'Saya sedang belajar bahasa C'` sebanyak 10 kali, maka kita tidak perlu untuk menuliskan 10 buah statemen melainkan kita hanya tinggal menempatkan satu buah statemen ke dalam suatu struktur pengulangan. Dengan demikian program kita akan lebih efesien.

Sebagai gambaran bagi Anda untuk dapat lebih menyerap konsep pengulangan, coba Anda perhatikan terlebih dahulu contoh program di bawah ini.

```
#include <stdio.h>

int main(void) {

  /* Mencetak teks ke layar sebanyak 10 kali */
  printf("Saya sedang belajar bahasa C");
  printf("Saya sedang belajar bahasa C");
  printf("Saya sedang belajar bahasa C");
  printf("Saya sedang belajar bahasa C");
  printf("Saya sedang belajar bahasa C");
  printf("Saya sedang belajar bahasa C");
  printf("Saya sedang belajar bahasa C");
  printf("Saya sedang belajar bahasa C");
  printf("Saya sedang belajar bahasa C");
  printf("Saya sedang belajar bahasa C");

  return 0;
}
```

Program di atas akan memberikan hasil sebagai berikut.

```
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
```

Apabila ditinjau dari hasilnya, program di atas memang memberikan hasil seperti yang diinginkan, namun apakah penulisan program tersebut dapat dikatakan efesien?

Jawabnya tentu tidak. Penulisan program seperti yang dilakukan di atas akan menyebabkan besarnya ukuran file kode program yang dihasilkan dan merupakan suatu pemborosan waktu dan tenaga. Mungkin untuk menuliskan 10 buah statemen di atas, kita masih dapat melakukkannya dengan cepat dan mudah, namun bagaimana apabila ternyata kita harus menuliskan statemen tersebut sebanyak 100 kali atau bahkan 500 kali. Hal tersebut tentu akan merupakan sebuah hal yang membosankan. Kasarnya, program di atas dapat kita katakan sebagai program yang salah.

Dalam bahasa C, terdapat tiga buah struktur pengulangan yang akan digunakan sebagai kontrol dalam melakukan pengulangan proses, yaitu struktur `for`, `while` dan `do-while`.

Setiap struktur yang ada masing-masing memiliki aturan tersendiri dan digunakan dalam konteks yang berbeda. Kita harus bisa menentukan kapan sebaiknya kita harus menggunakan struktur `for`, struktur `while` ataupun `do-while`. Untuk itu, pada bagian ini kita akan membahas secara detil apa perbedaan dan penggunaan dari masing-masing struktur tersebut. Hal ini juga bertujuan agar Anda dapat mengimplementasikannya dengan benar ke dalam kasus-kasus program yang Anda hadapi.

4.3.1. Struktur `for`

Struktur `for` ini digunakan untuk menuliskan jenis pengulangan yang banyaknya sudah pasti atau telah diketahui sebelumnya. Oleh karena itu, di sini kita harus melakukan inisialisai nilai untuk kondisi awal pengulangan dan juga harus menuliskan kondisi untuk menghentikan proses pengulangan. Adapun bentuk umum dari pendefinisian struktur `for` (untuk sebuah statemen) dalam bahasa C adalah sebagai berikut.

```
for (ekpresi1; ekspresi2; ekspresi3)
  Statemen_yang_akan_diulang;
```

Sedangkan untuk dua statemen atau lebih.

```
for (ekpresi1; ekspresi2; ekspresi3) {
  Statemen_yang_akan_diulang1;
  Statemen_yang_akan_diulang2;
  …
}
```

`Ekspresi1` di atas digunakan sebagai proses inisialisasi variabel yang akan dijadikan sebagai pencacah (*counter*) dari proses pengulangan, dengan kata lain ekspresi ini akan dijadikan sebagai kondisi awal.

`Ekspresi2` digunakan sebagai kondisi akhir, yaitu kondisi dimana proses pengulangan harus dihentikan. Perlu untuk diketahui bahwa pengulangan masih akan dilakukan selagi kondisi akhir bernilai benar.

`Ekspresi3` digunakan untuk menaikkan (*increment*) atau menurunkan (*decrement*) nilai variabel yang digunakan sebagai pencacah. Apabila pengulangan yang kita

lakukan bersifat menaik, maka kita akan menggunakan statemen *increment*, sedangkan apabila pengulangan yang akan kita lakukan bersifat menurun maka kita harus menggunakan statemen *decrement*.

Berikut ini contoh untuk mengilustrasikan struktur pengulangan `for` yang telah diterangkan di atas.

```
for (int j=0; j<10; j++) {
  /* Statemen yang akan diulang */
  ...
}
```

Pada sintak di atas, mula-mula kita menginisialisasi variabel `j` dengan nilai 0, kemudian karena ekspresi `(0 < 10)` bernilai benar maka program akan melakukan statemen untuk pertama kalinya. Setelah itu variabel `j` akan dinaikkan nilainya sebesar 1 melalui statemen *increment* `j++` sehingga nilai `j` sekarang menjadi 1. Sampai di sini program akan mengecek ekspresi `(j < 10)`. Oleh karena ekspresi `(2 < 10)` bernilai benar, maka program akan melakukan statemen yang kedua kalinya. Begitu seterusnya sampai nilai `j` bernilai 9. Namun pada saat variabel `j` telah bernilai 10 maka program akan keluar dari proses pengulangan. Hal ini disebabkan oleh karena ekspresi `(10 < 10)` bernilai salah.

Untuk membuktikan hal tersebut, perhatikan contoh program di bawah ini dimana kita akan menampilkan teks `‘Saya sedang belajar bahasa C’` sebanyak 10 kali.

```
#include <stdio.h>

int main(void) {
  for (int j=0; j<10; j++) {
    printf(“Saya sedang belajar bahasa C\n”);
  }

  return 0;
}
```

Hasil yang akan diberikan oleh program di atas adalah sebagai berikut:

```
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
```

```
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
```

Pengulangan yang dilakukan pada program di atas bersifat menaik sehingga kita menggunakan *increment*. Kita juga dapat melakukan pengulangan tersebut secara menurun, yaitu dengan sintak di bawah ini.

```
#include <stdio.h>

int main(void) {

  for (int j=10; j>0; j--) {
    printf("Saya sedang belajar bahasa C\n");
  }

  return 0;
}
```

Pada sintak di atas, mula-mula variabel `j` bernilai 10 dan setiap pengulangan dilakukan menyebabkan variabel tersebut dikurangi satu. Hal ini disebabkan karena statemen *decrement* `j--` di atas. Dalam program tersebut, pengulangan baru akan dihentikan ketika variabel `j` bernilai 0. Apabila dijalankan program di atas akan memberikan hasil yang sama dengan program sebelumnya.

Indeks dari variabel yang digunakan sebagai nilai awal dalam struktur `for` tidak selalu harus bernilai 0, artinya kita dapat memanipulasinya sesuai dengan keinginaan kita (misalnya dengan nilai 1, 2, 3 ataupun lainnya). Misalnya apabila kita akan melakukan suatu statemen sebanyak 5 kali, maka kita dapat menuliskannya sebagai berikut.

```
for (int j=1; j<=5; j++) {
  /* Statemen yang akan diulang */
  ...
}
```

atau bisa juga seperti di bawah ini

```
for (int j=10; j>=5; j--) {
  /* Statemen yang akan diulang */
  ...
}
```

Selain tipe `int`, kita juga dapat menggunakan variabel yang bertipe `char` sebagai pencacah dalam proses pengulangan. Sebagai contoh apabila kita akan melakukan pengulangan sebanyak 3 kali, maka kita dapat menuliskannya sebagai berikut.

```
for (char j='a'; j<='c'; j++) {
  /* Statemen yang akan diulang */
  ...
}
```

Berikut ini contoh program yang akan menunjukkan hal tersebut.

```
#include <stdio.h>

int main(void) {
  for (char j='A'; j<='E'; j++) {
    printf("%c = %d\n", j, j);
  }
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah seperti yang tampak di bawah ini.

```
A = 65
B = 66
C = 67
D = 68
E = 69
```

4.3.1.1. Melakukan Beberapa Inisialisasi dalam Struktur `for`

Dalam bahasa C, kita diizinkan untuk melakukan banyak inisialisasi di dalam struktur `for`. Hal ini tentu akan dapat mengurangi banyaknya baris kode program. Adapun cara untuk melakukan hal ini, yaitu dengan menambahkan tanda koma (,) di dalam bagian inisialisasi, seperti yang tertulis di bawah ini.

```
int a, b;
for (a=0, b=0; a<5; a++, b +=5) {
  ...
}
```

Sintak di atas sama seperti penulisan sintak berikut.

```
int a, b;
b = 0;
for (a=0; a<5; a++) {
```

```
  ...
  b += 5;  /* dapat ditulis dengan b = b + 5 */
}
```

Untuk membuktikannya, perhatikan dua buah contoh program di bawah ini.

a. Menggunakan dua inisialisasi

```
#include <stdio.h>

int main(void) {

  for (int a=0, int b=0; a < 5; a++, b += 5) {
    printf("Baris ke-%d : a = %d,   b = %2d\n", a+1, a, b);
  }

  return 0;
}
```

b. Menggunakan satu inisialisasi

```
#include <stdio.h>

int main(void) {
  int b=0;
  for (int a=0, a < 5; a++) {
    printf("Baris ke-%d : a = %d,   b = %2d\n", a+1, a, b);
    b += 5;
  }

  return 0;
}
```

Apabila dijalankan, kedua program di atas akan memberikan hasil yang sama, yaitu seperti yang tampak di bawah ini.

```
Baris ke-1 : a = 0, b = 0
Baris ke-2 : a = 1; b = 5
Baris ke-3 : a = 2; b = 10
Baris ke-4 : a = 3; b = 15
Baris ke-5 : a = 4; b = 20
```

4.3.1.2. Struktur `for` Bersarang (Nested for)

Dalam pemrograman sering kali kita dituntut untuk melakukan proses pengulangan di dalam strukur pengulangan yang sedang dilakukan. Begitupun dalam bahasa C, kita juga dapat melakukkannya dengan menggunakan struktur `for`. Sebagai contoh di sini kita akan mendemonstrasikannya dengan sebuah program yang dapat menampilkan tabel perkalian dari 1 sampai 10. Adapun sintaknya adalah sebagai berikut:

```
#include <stdio.h>

int main(void) {
  int baris, kolom;

  /* Proses pengulangan pertama */
  for (baris=1; baris<=10; baris++) {
    /* Proses pengulangan kedua yanag terdapat dalam pengulangan
       pertama */
    for (kolom=1; kolom<=10; kolom++) {
      printf("%3d ", baris*kolom);
    }
    printf("\n");
  }
  return 0;
}
```

Adapun hasil yang akan diberikan dari program di atas adalah seperti yang tampak di bawah ini.

```
  1   2   3   4   5   6   7   8   9  10
  2   4   6   8  10  12  14  16  18  20
  3   6   9  12  15  18  21  24  27  30
  4   8  12  16  20  24  28  32  36  40
  5  10  15  20  25  30  35  40  45  50
  6  12  18  24  30  36  42  48  54  60
  7  14  21  28  35  42  49  56  63  70
  8  16  24  32  40  48  56  64  72  80
  9  18  27  36  45  54  63  72  81  90
 10  20  30  40  50  60  70  80  90 100
```

4.3.2. Struktur `while`

Setelah Anda mengetahui struktur pengulangan jenis pertama, sekarang kita akan mempelajari struktur pengulangan berikutnya, yaitu struktur `while`. Pada struktur pengulangan jenis ini kondisi akan diperiksa di bagian awal. Hal ini tentu menyebabkan kemungkinan bahwa apabila ternyata kondisi yang kita definisikan tidak terpenuhi (bernilai salah), maka proses pengulangan pun tidak akan pernah dilakukan. Adapun bentuk umum dari struktur `while` adalah seperti yang tampak di bawah ini.

```
while (ekspresi) {
    Statemen_yang_akan_diulang1;
    Statemen_yang_akan_diulang2;
    …
}
```

Sama seperti pada struktur `for`, struktur pengulangan jenis ini juga memerlukan suatu inisialisasi nilai pada variabel yang akan digunakan sebagai pencacah, yaitu dengan menuliskannya di atas blok pengulangan. Selain itu kita juga harus melakukan penambahan ataupun pengurangan terhadap nilai dari variabel pencacah di dalam blok pengulangan tersebut. Hal ini bertujuan untuk menghentikan pengulangan sesuai dengan kondisi yang didefinisikan. Sebagai contoh apabila kita ingin melakukan pengulangan proses sebanyak 5 kali, maka kita akan menuliskannya sebagai berikut.

```
int j = 0; /* Melakukan inisialisasi terhadap variabel j dengan
              nilai 0 */
while (j<5) {
  /* Statemen yang akan diulang */
  ...
  j++;     /* Melakukan increment terhadap variabel j */
}
```

Untuk menunjukkan bagaimana struktur pengulangan `while` ini bekerja, perhatikan contoh program untuk menghitung jumlah 5 buah bilangan positif pertama ini.

```
#include <stdio.h>

int main(void) {
  int j = 1;          /* Mendeklarasikan variabel j sebagai
                         pencacah pengulangan */
  jumlah = 0;         /* Mendeklarasikan variabel jumlah untuk
                         menampung jumlah */
  while (j <= 5) {
    jumlah += j;
    j++;
  }
  printf("Jumlah = %d", jumlah);
  return 0;
}
```

Apabila program tersebut dijalankan maka hasil yang akan diberikan adalah sebagai berikut.

```
Jumlah = 15
```

Sebagai pembanding dengan struktur pengulangan `for` di atas, maka di sini dituliskan kembali program yang akan menampilkan teks `'Saya sedang belajar bahasa C'` dengan menggunakan struktur `while`. Adapun sintaknya adalah sebagai berikut.

```
#include <stdio.h>

int main(void) {
  /* Mendeklarasikan variabel j dan menginisialisasinya dengan
     nilai 0 */
  int j=0;

  /* Melakukan pengulangan proses */
  while (j<10) {
    printf("Saya sedang belajar bahasa C\n");
    j++;
  }
  return 0;
}
```

Program di atas akan memberikan hasil yang sama dengan program yang menggunakan struktur `for` di atas, yaitu seperti di bawah ini.

```
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
```

Selain menggunakan struktur `for`, kita juga dapat melakukan pengulangan bersarang dengan menggunakan struktur `while`. Caranya sama, yaitu dengan mendefinisikan struktur `while` di dalam pengulangan yang sedang dilakukan. Berikut ini contoh program yang akan menunjukkan pengulangan bersarang dengan menggunakan struktur `while`.

```
#include <stdio.h>

int main(void) {

  int j=1; /* Mendeklarasikan variabel j sebagai pencacah
              pengulangan ke-1 */
  int k;   /* Mendeklarasikan variabel k sebagai pencacah
```

```
                pengulangan ke-2 */

  /* Melakukan pengulangan ke-1 */
  while (j <= 10) {
    k = 1;
    /* Melakukan pengulangan ke-2 */
    while(k <=10 ) {
      printf("%3d ", j*k);
      k++;
    }
    printf("\n");
    j++;
  }

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
 1  2  3  4  5  6  7  8  9  10
 2  4  6  8 10 12 14 16 18  20
 3  6  9 12 15 18 21 24 27  30
 4  8 12 16 20 24 28 32 36  40
 5 10 15 20 25 30 35 40 45  50
 6 12 18 24 30 36 42 48 54  60
 7 14 21 28 35 42 49 56 63  70
 8 16 24 32 40 48 56 64 72  80
 9 18 27 36 45 54 63 72 81  90
10 20 30 40 50 60 70 80 90 100
```

Kita dapat juga mengkombinasikan struktur `for` dan struktur `while` untuk melakukan pengulangan bersarang. Berikut ini contoh program yang akan menunjukkan hal tersebut.

```
#include <stdio.h>

int main(void) {

  int j;   /* Mendeklarasikan variabel j sebagai pencacah
              pengulangan ke-1 */
  int k;   /* Mendeklarasikan variabel k sebagai pencacah
              pengulangan ke-2 */

  for (j=1; j<=5; j++) {
    k = 1;
    while(k<=j ) {
      printf("%2d ", j*k);
      k++;
```

```
    }
    printf("\n");
  }
  j = 4;
  while (j>=1) {
    for (k=1; k<=j; k++ ) {
      printf("%2d ", j*k);
    }
    printf("\n");
    j--;
  }

  return 0;
}
```

Hasil yang akan diberikan oleh program di atas adalah sebagai berikut.

```
1
2  4
3  6  9
4  8 12 16
5 10 15 20 25
4  8 12 16
3  6  9
2  4
1
```

4.3.3. Struktur `do-while`

Berbeda dengan struktur `while` dimana kondisinya terletak di awal blok pengulangan, pada struktur `do-while` kondisi diletakkan di akhir blok pengulangan. Hal ini menyebabkan bahwa statemen yang terdapat di dalam blok pengulangan ini pasti akan dieksekusi minimal satu kali, walaupun kondisinya bernilai salah sekalipun. Maka dari itu struktur `do-while` ini banyak digunakan untuk kasus-kasus pengulangan yang tidak mempedulikan benar atau salahnya kondisi pada saat memulai proses pengulangan.

Adapun bentuk umum dari struktur pengulangan `do-while` adalah seperti yang tertulis di bawah ini.

```
do {
  Statemen_yang_akan_diulang;
  …
} while (ekspresi);        /* Ingat tanda semicolon (;) */
```

Mungkin bagi Anda yang merupakan programmer pemula akan merasa bingung untuk membedakan struktur pengulangan `while` dan `do-while`. Maka dari itu, perhatikan dulu dua buah contoh program berikut ini.

a. Menggunakan struktur `while`

```
#include <stdio.h>

int main(void) {
  int j;
  printf("Statemen sebelum blok pengulangan\n");
  j = 10;

  while (j < 5) {
    printf("Statemen di dalam blok pengulangan\n");
    j++;
  }
  printf("Statemen setelah blok pengulangan\n");
  return 0;
}
```

Hasil yang akan diberikan oleh program di atas adalah sebagai berikut.

```
Statemen sebelum blok pengulangan
Statemen setelah blok pengulangan
```

Hal ini disebabkan karena variabel `j` bernilai 10 sehingga kondisi (j < 5) bernilai salah. Kemudian karena kondisi tidak terpenuhi, maka program tidak akan mengeksesusi statemen yang terdapat dalam blok pengulangan.

b. Menggunakan struktur `do-while`

```
#include <stdio.h>

int main(void) {
  int j;
  printf("Statemen sebelum blok pengulangan\n");
  j = 10;
  do {
    printf("Statemen di dalam blok pengulangan\n");
    j++;
  } while (j < 5);
  printf("Statemen setelah blok pengulangan\n");

  return 0;
}
```

Program di atas akan memberikan hasil sebagai berikut.

```
Statemen sebelum blok pengulangan
Statemen di dalam blok pengulangan
Statemen setelah blok pengulangan
```

Mungkin Anda akan bertanya kenapa statemen yang terdapat pada blok pengulangan tersebut ikut dieksekusi padahal kondisi yang didefinisikan tersebut tidak terpenuhi? Berbeda dengan sebelumnya, kali ini program akan langsung mengeksekusi statemen yang terdapat dalam blok pengulangan. Setelah itu, program baru akan mengecek ekspresi `(j < 5)`. Oleh karena variabel `j` bernilai 10 maka ekspresi tersebut bernilai salah dan hal ini tentu akan menyebabkan program menghentikan proses pengulangan.

Untuk lebih memahaminya, berikut ini disajikan contoh program yang menunjukkan penggunaan struktur pengulangan `do-while`. Pada program ini kita akan menenetukan kelipatan persekutuan terkecil (KPK) dari dua buah bilangan bulat. Sebagai contoh KPK dari bilangan 8 dan 12 adalah **24**. Untuk lebih jelasnya perhatikan tabel kelipatan di bawah ini.

8	16	**24**	32	40	48	…
12	**24**	36	48	60	72	…

Adapun sintak program yang dimaksud adalah sebagai berikut.

```
#include <stdio.h>

int main(void) {

  /* Mendeklarasikan variabel X, Y dan hasil */
  int X, Y, hasil = 0;

  printf("Masukkan bilangan ke-1 : "); scanf("%d", &X);
  printf("Masukkan bilangan ke-2 : "); scanf("%d", &Y);

  /* Melakukan pertukaran nilai */
  if (X < Y) {
    int temp = Y;
    Y = X;
    X = temp;
  }

  /* Melakukan proses pengulangan */
  do {
    hasil += X;
  } while (hasil % Y != 0);

  printf("\n\nKPK : %d", hasil);
  return 0;
}
```

Contoh hasil yang akan diberikan oleh program di atas adalah seperti yang tampak di bawah ini.

```
Masukkan bilangan ke-1 : 12
Masukkan bilangan ke-2 : 8

KPK : 24
```

Sebagai contoh lain untuk menunjukkan penggunaan struktur pengulangan do-while adalah dengan membuat program yang berfungsi untuk menentukkan faktor persekutuan terbesar atau pembagi bersama terbesar (PBT) dari dua buah bilangan bulat. Misalnya bilangan 12 dan 8, maka faktor persekutuan terbesarnya adalah **4**. Untuk lebih jelasnya, perhatikan tabel di bawah ini.

Bilangan	Faktor
8	1, 2, **4**, 8
12	1, 2, 3, **4**, 6, 12

Adapun sintak program yang dapat menentukan nilai tersebut adalah sebagai berikut.

```
#include <stdio.h>

int main(void) {
  int X, Y, sisa; /* Mendeklarasikan variabel X, Y dan sisa */

  printf("Masukkan bilangan ke-1 : "); scanf("%d", &X);
  printf("Masukkan bilangan ke-2 : "); scanf("%d", &Y);

  /* Melakukan pertukaran nilai */
  if (X < Y) {
    int temp = Y;
    Y = X;
    X = temp;
  }

  /* Melakukan proses pengulangan */
  do {
    sisa = X % Y;
    X = Y;
    Y = sisa;
  } while (sisa != 0);

  printf("\n\nPBT : %d", X);

  return 0;
}
```

Adapun contoh hasil yang akan diberikan dari program di atas adalah seperti yang tampak di bawah ini.

```
Masukkan bilangan ke-1 : 8
Masukkan bilangan ke-2 : 12

PBT : 4
```

4.4. Statemen Peloncatan

Statemen peloncatan pada umumnya digunakan dalam sebuah proses pengulangan, yang menentukan apakah pengulangan akan diteruskan, dihentikan atau dipindahkan ke statemen lain di dalam program. Dalam bahasa C, terdapat tiga buah kata kunci yang digunakan untuk melakukan proses peloncatan tersebut, yaitu `break`, `continue` dan `goto`. Namun terdapat juga sebuah fungsi yang digunakan untuk melakukan proses peloncatan, yaitu fungsi `exit()` yang disimpan dalam file header `<stdlib.h>`.

4.4.1. Menggunakan Kata Kunci `break`

Statemen break digunakan untuk menghentikan sebuah pengulangan dan program akan langsung meloncat ke statemen yang berada di bawah blok pengulangan. Ini biasanya dilakukan karena alasan efesiensi program, yaitu untuk menghindari proses pengulangan yang sebenarnya sudah tidak diperlukan lagi. Sebagai contoh, di sini kita akan menuliskan program untuk menentukan suatu bilangan apakah termasuk ke dalam bilangan prima atau tidak. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>

/* Mendefinisikan fungsi untuk mengecek bilangan prima atau
   bukan */
int CekPrima(int x) {
  int prima = 1;/* mula-mula variabel prima bernilai 1 (true) */
  int i;        /* variabel untuk indeks pengulangan */

  if (x <= 1) {
    prima = 0;  /* apabila x ≤ 1 maka prima bernilai 0
                   (false) */
  } else {
    for (i=2; i<=(x/2); i++) {
      if (x % i == 0) {
        prima = 0;    /* prima bernilai false */
        break;        /* menghentikan proses pengulangan */
      }
    }
  }

  return prima;
}
```

```
int main(void) {
  int a;

  printf(“Masukkan sebuah bilangan bulat “ \
         “yang akan diperiksa : ”);
  scanf(%d“, &a);

  /* Mengecek a merupakan bilangan prima atau bukan */
  if (CekPrima(a)) {
    printf(“%d merupakan bilangan prima”);
  } else {
    printf(“%d bukan merupakan bilangan prima”);
  }
  return 0;
}
```

Pada program di atas, apabila dalam proses pengulangan kita telah mengeset nilai variabel `prima` menjadi 0 (`false`), maka kita tidak perlu lagi untuk melanjutkan proses pengulangan, karena bilangan yang diperiksa sudah pasti bukan merupakan bilangan prima. Dengan demikian apabila kita masih melanjutkan proses pengulangan maka hal tersebut dapat dikatakan sebagai hal yang sia-sia. Oleh karena itu, pada kasus ini kita harus menambahkan statemen `break` untuk menghentikan pengulangan tersebut.

Mungkin sebagian dari Anda ada yang berfikir bagaimana apabila statemen `break` digunakan dalam pengulangan yang bersarang? Jawabnya mudah, statemen `break` hanya berlaku untuk satu buah blok pengulangan, artinya apabila di luar blok pengulangan terdapat suatu blok pengulangan lagi, maka yang akan dihentikan adalah pengulangan yang mengandung statemen `break` saja. Sedangkan blok pengulangan di luarnya akan tetap dilanjutkan sesuai dengan kondisi yang didefinisikan. Untuk lebih jelasnya perhatikan contoh program sederhana di bawah ini.

```
#include <stdio.h>

int main(void) {
  int i, j;

  /* Pengulangan luar */
  for (i=1; i<=10; i++) {
    /* Pengulangan dalam */
    for (j=1; j<=5; j++) {
      if (j > 3) {
        break;   /* hanya akan menghentikan pengulangan dalam */
      }
      printf(“%d\t”, i*j);
    }
    printf(“\n”);
  }
  return 0;
}
```

Hasil yang akan diberikan dari program tersebut adalah sebagai berikut.

```
1       2       3
2       4       6
3       6       9
4       8       12
5       10      15
6       12      18
7       14      21
8       16      24
9       18      27
10      20      30
```

Coba Anda amati proses pengulangan kedua diatas (pengulangan dalam). Di situ tampak bahwa apabila variabel `j` telah bernilai 4 (yang berarti `j` > 3) maka pengulangan tersebut akan dihentikan. Sedangkan proses pengulangan yang pertama (pengulangan luar) akan tetap dilakukan selama kondisi yang didefinisikan pada blok tersebut masih terpenuhi.

4.4.2. Menggunakan Kata Kunci `continue`

Berbeda dengan statemen `break` di atas yang berguna untuk menghentikan suatu proses pengulangan, statemen `continue` justru digunakan untuk melanjutkan proses pengulangan. Sebagai contoh apabila kita akan membuat program untuk melakukan pembagian dua buah bilangan, maka kita harus menjaga agar bilangan pembagi (penyebut) harus tidak sama dengan nol. Untuk kasus ini, kita akan membuat sebuah pengulangan untuk melakukan input sampai bilangan pembagi yang dimasukkan tidak sama dengan nol. Berikut ini contoh sintak program yang dimaksud.

```
#include <stdio.h>

#define TRUE    1
#define FALSE   0

int main(void) {
  double a = 1; /* Menginisialisasi bilangan yang akan di bagi
                   (pembilang) */
  double b;    /* Variabel penampung nilai pembagi (penyebut) */

  /* Memaksa proses pengulangan */
  while (TRUE) {
    printf("Masukkan bilangan pembagi : "); scanf("%lf", &b);
    if (b == 0) {
      continue; /* Apabila pembagi 0,
                   maka lanjutkan pengulangan */
    }
    printf("1/%d = %.2lf", a, a/b);
```

```
    break;    /* Apabila proses perhitungan selesai maka
                 pengulangan dihentikan */
  }

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Masukkan bilangan pembagi : 0
Masukkan bilangan pembagi : 0
Masukkan bilangan pembagi : 2
1/2 = 0.50
```

Tampak di atas bahwa selama bilangan pembagi yang dimasukkan oleh user (pengguna program) masih bernilai nol, maka proses pengulangan akan terus dilanjutkan. Hal yang perlu diperhatikan di sini adalah bahwa apabila program mengeksekusi statemen `continue` maka program akan langsung kembali meloncat ke statemen 'awal' pada blok pengulangan. Dengan kata lain, statemen-statemen yang terdapat di bawah statemen `continue` akan diabaikan oleh program.

4.4.3. Menggunakan Kata Kunci `goto`

Selain cara-cara yang telah dijelaskan di atas, bahasa C juga menyediakan kata kunci `goto` yang digunakan agar program dapat meloncat ke baris tertentu yang kita pilih. Adapun untuk menentukan baris tersebut kita harus membuat suatu label, yaitu dengan menempatkan tanda *colon* atau titik dua (**:**) di belakangnya, misalnya `LBL:`, `LABEL:`, `mylabel:` atau nama-nama lain yang Anda kehendaki. Berbeda dengan statemen `break` dan `continue` yang umumnya digunakan untuk proses pengulangan, statemen `goto` dapat ditempatkan di mana saja sesuai dengan kebutuhan program. Berikut ini contoh program yang akan menunjukkan penggunaan statemen `goto` di dalam proses pengulangan.

```
#include <stdio.h>

#define TRUE          1
#define FALSE   0

int main(void) {
  int counter = 0;    /* Variabel untuk indeks pengulangan */
  while (TRUE) {
    counter++;
    if (counter > 10) {
      goto LBL;
    }
    printf("Baris ke-%d\n", counter);
  }
```

```
  LBL:      /* Membuat label dengan nama LBL */

  printf("Statemen yang terdapat di luar blok pengulangan");

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Baris ke-1
Baris ke-2
Baris ke-3
Baris ke-4
Baris ke-5
Baris ke-6
Baris ke-7
Baris ke-8
Baris ke-9
Baris ke-10
Statemen yang terdapat di luar blok pengulangan
```

Pada program di atas terlihat jelas bahwa penggunaan statemen `goto` akan menyebabkan eksekusi program akan langsung berpindah atau meloncat ke label yang telah didefinisikan. Dalam hal ini, karena label didefinisikan di luar blok pengulangan maka proses peloncatan tersebut secara otomatis akan menyebabkan terhentinya proses pengulangan yang sedang berlangsung. Agar Anda lebih memahami penggunaan statemen goto, perhatikan kembali contoh program di bawah ini. (*catatan: nomor baris yang terdapat di bagian kiri program hanya digunanan untuk mempermudah pembacaan program*)

```
1:   #include <stdio.h>
2:
3:   int main(void) {
4:     printf("Ini adalah baris sebelum statemen goto\n");
5:
6:     goto LBL; /* untuk meloncat ke label LBL */
7:
8:     printf("Ini adalah baris setelah statemen goto " \
9:            "dan sebelum label LBL\n");
10:
11:    LBL:        /* Mendefinisikan label dengan nama LBL */
12:
13:    printf("Ini adalah baris setelah label LBL");
14:    return 0;
15:  }
```

Apabila dijalankan program di atas akan memberikan hasil sebagai berikut.

```
Ini adalah baris sebelum statemen goto
Ini adalah baris setelah label LBL
```

Dari hasil tersebut tampak jelas bahwa ketika program mengeksekusi statemen `goto` yang terdapat pada baris ke-6, maka program akan langsung meloncat menuju label dengan nama `LBL` yang didefinisikan (dalam hal baris ke-11). Hal ini berarti statemen yang terdapat pada baris ke-8 dan ke-9 akan diabaikan atau tidak ikut dieksekusi oleh program.

4.4.4. Menggunakan Fungsi `exit()`

Berbeda dengan statemen break yang berguna untuk menghentikan atau keluar dari proses pengulangan, fungsi `exit()` berguna untuk keluar dari program. Dalam bahasa C, terdapat dua buah nilai yang dikembalikan ke dalam sistem operasi yang menunjukkan bahwa program berjalan dengan baik (tanpa kesalahan) atau tidak. Nilai-nilai tersebut adalah 0 (`EXIT_SUCCESS`) dan 1 (`EXIT_FAILURE`). Hal yang perlu diperhatikan sebelum Anda menggunakan fungsi `exit()` adalah Anda harus mendaftarkan file header `<stdlib.h>` dimana fungsi tersebut dideklarasikan. Berikut ini contoh program yang akan menunjukkan penggunaan fungsi `exit()`.

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {

  /* Mendeklarasikan pointer ke struktur FILE */
  FILE *fp;

  /* Proses untuk membuka file COBA.TXT */
  if ((fp = fopen("COBA.TXT","r+")) == NULL) {
    printf("File COBA.TXT tidak dapat dibuka");
    exit(1); /* Keluar dari aplikasi dengan adanya kesalahan */
  }

  return 0;
}
```

Apabila tidak perlu cemas dengan kehadiran struktur `FILE` yang terdapat pada deklarasi di atas karena semua itu akan kita belajari dalam buku ini secara terpisah, tepatnya pada bab 10 – *Operasi File*. Namun untuk saat ini yang perlu Anda perhatikan hanyalah penggunaan fungsi `exit()` di atas. Pada program di atas fungsi tersebut akan menghentikan program secara tidak normal apabila file COBA.TXT yang kita buka tidak terdapat dalam direktori aktif.

Bab

5

Fungsi

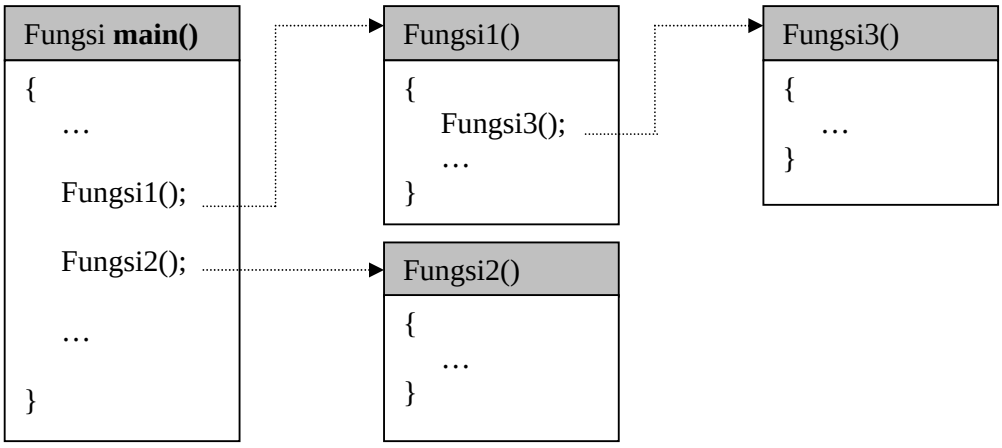
5.1. Pendahuluan

Dalam bahasa C, sebuah program terdiri atas fungsi-fungsi, baik yang didefinisikan secara langsung di dalam program maupun yang disimpan di dalam file lain (misalnya file *header*). Satu fungsi yang pasti terdapat dalam program yang ditulis menggunakan bahasa C adalah fungsi **`main()`**. Fungsi tersebut merupakan fungsi utama dan merupakan fungsi yang akan dieksekusi pertama kali.

Menurut definisinya, fungsi adalah suatu blok program yang digunakan untuk melakukan proses-proses tertentu. Sebuah fungsi dibutuhkan untuk menjadikan program yang akan kita buat menjadi lebih modular dan mudah untuk dipahami alurnya. Dengan adanya fungsi, maka kita dapat mengurangi duplikasi kode program sehingga performa dari program yang kita buat pun akan meningkat.

Dalam bahasa C, fungsi terbagi menjadi dua macam, yaitu fungsi yang mengembalikan nilai (*return value*) dan fungsi yang tidak mengembalikan nilai. Fungsi yang tidak mengembalikan nilai tersebut dinamakan dengan *void function*. Bagi Anda yang sebelumnya pernah belajar bahasa Pascal, *void function* ini serupa dengan *procedure* yang terdapat di dalam bahasa Pascal.

Sebelum melangkah lebih jauh ke dalam pembentukan fungsi, di sini akan diterangkan bagaimana kompilator C membaca fungsi-fungsi yang didefiniskan di dalam program secara berurutan sesuai dengan waktu pemanggilannya. Untuk itu, perhatikanlah gambar ilustrasi berikut.



Gambar 5.1. Pemanggilan fungsi dalam bahasa C

Mula-mula fungsi `main()` akan dieksekusi oleh kompilator. Oleh karena fungsi `main()` tersebut memanggil `Fungsi1()`, maka kompilator akan mengeksekusi Fungsi1(). Dalam `Fungsi1()` juga terdapat pemanggilan `Fungsi3()`, maka kompilator akan mengeksekusi `Fungsi3()` dan kembali lagi mengeksekusi baris selanjutnya yang terdapat pada `Fungsi1()` (jika ada). Setelah `Fungsi1()` selesai dieksekusi, kompilator akan kembali mengeksekusi baris berikutnya pada fungsi `main()`, yaitu dengan mengeksekusi kode-kode yang terdapat pada `Fungsi2()`. Setelah selesai, maka kompilator akan melanjutkan pengeksekusian kode pada baris-baris selanjutnya dalam fungsi `main()`. Apabila ternyata dalam fungsi `main()` tersebut kembali terdapat pemanggilan fungsi lain, maka kompilator akan meloncat ke fungsi lain tersebut, begitu seterusnya sampai semua baris kode dalam fungsi `main()` selesai dieksekusi.

Bagi Anda yang merupakan para pemula di dunia pemrograman, sebisa mungkin biasakanlah untuk menggunakan fungsi daripada Anda harus menuliskan kode secara langsung pada fungsi `main()`. Oleh karena itu, pada bab ini penulis akan membahas secara detil mengenai bagaimana cara pembuatan fungsi di dalam bahasa C sehingga Anda akan mudah dan terbiasa dalam mengimplementasikannya ke dalam kasus-kasus program yang Anda hadapi.

5.2. Apa Nilai yang dikembalikan Oleh Fungsi `main()`?

Pada bab-bab sebelumnya kita telah banyak menggunakan fungsi `main()` di dalam program yang kita buat. Mungkin sekarang Anda akan bertanya apa sebenarnya nilai yang dikembalikan oleh fungsi `main()` tersebut? Jawabnya adalah nilai 0 dan 1. Apabila fungsi `main()` mengembalikan nilai 0 ke sistem operasi, maka sistem operasi akan mengetahui bahwa program yang kita buat tesebut telah dieksekusi dengan benar tanpa adanya kesalahan. Sedangkan apabila nilai yang dikembalikan ke sistem operasi adalah nilai 1, maka sistem operasi akan mengetahui bahwa program telah dihentikan secara tidak normal (terdapat kesalahan). Dengan demikian, seharusnya fungsi `main()` tidaklah mengembalikan tipe `void`, melainkan tipe data `int`, seperti yang terlihat di bawah ini.

```
int main(void) {
  ...
  return 0;           /* Mengembalikan nilai 0 */
}
```

Namun, apabila Anda ingin mendefinisikan nilai kembalian tersebut dengan tipe `void`, maka seharusnya Anda menggunakan fungsi `exit()` yang dapat berguna untuk mengembalikan nilai ke sistem operasi (sama seperti halnya `return`). Adapun parameter yang dilewatkan ke fungsi `exit()` ini ada dua, yaitu:

1. Nilai 0 (`EXIT_SUCCESS`), yaitu menghentikan program secara normal
2. Nilai 1 (`EXIT_FAILURE`), yaitu menghentikan program secara tidak normal

Berikut ini contoh penggunaannya di dalam fungsi `main()`.

```
void  main(void) {
  ...
  exit(0);      /* dapat ditulis exit(EXIT_SUCCESS) */
}
```

5.3. Fungsi Tanpa Nilai Balik

Pada umumnya fungsi tanpa nilai balik (*return value*) ini digunakan untuk melakukan proses-proses yang tidak menghasilkan nilai, seperti melakukan pengulangan, proses pengesetan nilai ataupun yang lainnya. Dalam bahasa C, fungsi semacam ini tipe kembaliannya akan diisi dengan nilai `void`. Adapun bentuk umum dari pendefinisian fungsi tanpa nilai balik adalah sebagai berikut:

```
void nama_fungsi(parameter1, parameter2,…) {
  Statemen_yang_akan_dieksekusi;
  ...
}
```

Berikut ini contoh dari pembuatan fungsi tanpa nilai balik.

```
void Tulis10Kali(void) {
  int j;
  for (j=0; j<10; j++) {
    printf(“Saya sedang belajar bahasa C”);
  }
}
```

Adapun contoh program lengkap yang akan menggunakan fungsi tersebut adalah seperti yang tertulis di bawah ini.

```
#include <stdio.h>

/* Mendefinisikan sebuah fungsi dengan nama Tulis10Kali */
void Tulis10Kali(void) {
  int j;

  for (j=0; j<10; j++) {
    printf(“Saya sedang belajar bahasa C”);
  }
}

int main(void) {
```

```
  Tulis10Kali();      /* Memanggil fungsi Tulis10Kali() */
  return 0;
}
```

Apabila dijalankan, program tersebut akan memberikan hasil sebagai berikut.

```
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
Saya sedang belajar bahasa C
```

Apabila Anda perhatikan secara teliti, fungsi `Tulis10Kali()` dalam program di atas tidak menghasilkan nilai, melainkan melakukan proses, yaitu penulisan teks `'Saya sedang belajar bahasa C'` sebanyak 10 kali. Fungsi semacam inilah yang disebut dengan fungsi tanpa nilai balik (*void function*).

Sebenarnya sebuah fungsi tanpa nilai balik juga dapat digunakan untuk menghasilkan nilai dari suatu proses tertentu, yaitu dengan menampungnya ke dalam suatu parameter. Namun hal ini baru akan kita bahas pada sub bab selanjutnya dalam bab ini (lihat sub bab *Fungsi dengan Parameter*).

5.4. Fungsi dengan Nilai Balik

Berbeda dengan fungsi di atas yang hanya mengandung proses tanpa adanya nilai kembalian, di sini kita akan membahas mengenai fungsi yang digunakan untuk melakukan proses-proses yang berhubungan dengan nilai. Adapun cara pendefinisiannya adalah dengan menuliskan tipe data dari nilai yang akan dikembalikan di depan nama fungsi, berikut ini bentuk umum dari pendefinisian fungsi dengan nilai balik di dalam bahasa C.

```
tipe_data nama_fungsi(parameter1, parameter2,...) {
  Statemen_yang_akan_dieksekusi;
  ...
  return nilai_balik;
}
```

Sebagai contoh, di sini kita akan membuat fungsi sederhana yang berguna untuk menghitung nilai luas bujursangkar. Adapun sintak untuk pendefinisiannya adalah sebagai berikut.

```
int HitungLuasBujurSangkar(int sisi) {
  int L;  /* mendeklarasikan variabel L untuk menampung nilai
             luas */
  L = sisi * sisi;    /* memasukkan nilai sesuai dengan rumus
                         yang berlaku */
 return L;            /* mengembalikan nilai yang didapat dari
                         hasil proses */
}
```

Sedangkan untuk menggunakan fungsi tersebut, Anda harus menuliskan program lengkap seperti di bawah ini.

```
#include <stdio.h>

int HitungLuasBujurSangkar(int sisi) {
  int L;   /* mendeklarasikan variabel L untuk menampung nilai
              luas */
  L = sisi * sisi;    /* memasukkan nilai sesuai dengan rumus
                         yang berlaku */
  return L;           /* mengembalikan nilai yang didapat dari
                         hasil proses */
}

int main(void) {
  int S, Luas;

  /* Mengeset nilai variabel S dengan nilai 10*/
  S = 10;

  /* Memanggil fungsi HitungLuasBujurSangkar
     dan menampung nilainya ke variabel Luas
  */
  Luas = HitungLuasBujurSangkar(S);

  /* Mencetak hasil perhitungan ke layar monitor */
  printf("Luas bujur sangkar dengan sisi %d adalah %d", S,
          Luas);

  return 0;
}
```

Apabila Anda masih merasa bingung dengan kehadiran parameter di dalam fungsi di atas, Anda tidak perlu cemas karena akan dibahas pada sub bab di bawah ini.

5.5. Fungsi dengan Parameter

Parameter adalah suatu variabel yang berfungsi untuk menampung nilai yang akan dikirimkan ke dalam fungsi. Dengan adanya parameter, sebuah fungsi dapat bersifat dinamis. Parameter itu sendiri terbagi menjadi dua macam, yaitu *parameter formal* dan

parameter aktual. Parameter formal adalah parameter yang terdapat pada pendefinisian fungsi, sedangkan parameter aktual adalah parameter yang terdapat pada saat pemanggilan fungsi. Untuk lebih memahaminya, perhatikan contoh pendefinisian fungsi di bawah ini.

```
int TambahSatu(int x) {
     return ++x;
}
```

Pada sintak di atas, variabel `x` dinamakan sebagai *parameter formal*. Sekarang perhatikan sintak berikut.

```
int main(void) {
  int a = 10, hasil;
  hasil = TambahSatu(a);
  return 0;
}
```

Pada saat pemanggilan fungsi `TambahSatu()` di atas, variabel `a` dinamakan dengan *parameter aktual*.

Namun sebelum Anda mempelajari bagaimana cara melewatkan parameter di dalam sebuah fungsi, Anda harus mengetahui terlebih dahulu jenis-jenis dari parameter tersebut.

4.1.1. Jenis Parameter

Dalam dunia pemrograman dikenal tiga jenis parameter, yaitu parameter masukan, keluaran dan masukan/keluaran. Untuk memahami perbedaan dari setiap jenis parameter, di sini kita akan membahasnya satu per satu.

a. Parameter Masukan

Parameter masukan adalah parameter yang digunakan untuk menampung nilai data yang akan dijadikan sebagai masukan (*input*) ke dalam fungsi. Artinya, sebuah fungsi dapat menghasilkan nilai yang berbeda tergantung dari nilai parameter yang dimasukkan pada saat pemanggilan fungsi tersebut. Berikut ini contoh program yang akan menunjukkan kegunaan dari parameter masukan.

```
#include <stdio.h>

#define PI 3.14159

/* Mendefinisikan suatu fungsi dengan parameter berjenis
   masukan */
double HitungKelilingLingkaran(int radius) {
```

```
  double K;
  K = 2 * PI * radius;
  return K;
}

/* Fungsi Utama */
int main(void) {
  int R;
  printf("Masukkan nilai jari-jari lingkaran : ");
  scanf("%d", &R);
  double Keliling = HitungKelilingLingkaran(R);
  printf("Keliling lingkaran dengan jari-jari %d : %f", R,
         Keliling);
  return 0;
}
```

Contoh hasil yang akan diberikan apabila program di atas dijalankan adalah sebagai berikut.

```
Masukkan nilai jari-jari lingkaran : 4
Keliling lingkaran dengan jari-jari 4 : 25.132720
```

Pada sintak program di atas, variabel `R` merupakan parameter aktual yang berfungsi sebagai parameter masukan karena variabel tersebut digunakan untuk menampung nilai yang akan menjadi masukan (*input*) untuk proses perhitungan di dalam fungsi `HitungKelilingLingkaran()`.

Sekarang perhatikan apabila kita memasukkan nilai parameter di atas dengan nilai 5, maka program akan memberikan hasil yang berbeda, yaitu seperti yang terlihat di bawah ini.

```
Masukkan nilai jari-jari lingkaran : 5
Keliling lingkaran dengan jari-jari 5 : 31.415900
```

Hal ini membuktikan bahwa fungsi `HitungLuasLingkaran()` akan menghasilkan nilai yang berbeda sesuai dengan nilai parameter yang dimasukkan.

b. Parameter Keluaran

Kebalikan dari parameter masukan, parameter keluaran adalah parameter yang digunakan untuk menampung nilai kembalian / nilai keluaran (*output*) dari suatu proses. Umumnya parameter jenis ini digunakan di dalam fungsi yang tidak mempunyai nilai balik. Untuk lebih memahaminya, perhatikan contoh program di bawah ini yang merupakan modifikasi dari program sebelumnya.

```
#include <stdio.h>
#define PI 3.14159

/* Mendefinisikan fungsi yang mengandung parameter keluaran */
void HitungKelilingLingkaran(int radius, double *K) {
  *K = 2 * PI * radius;
}

/* Fungsi Utama */
int main(void) {
  int R;
  double Keliling;

  printf("Masukkan nilai jari-jari lingkaran : ");
  scanf("%d", &R);
  HitungKelilingLingkaran(R, Keliling);
  printf("Keliling lingkaran dengan jari-jari %d : %f", R,
         Keliling);

  return 0;
}
```

Pada sintak program di atas, variabel `Keliling` berfungsi sebagai parameter keluaran karena variabel tersebut digunakan untuk menampung nilai hasil dari proses yang terdapat di dalam fungsi. Sedangkan variabel `R` adalah variabel yang bersifat sebagai parameter masukan dimana nilainya digunakan untuk menampung nilai yang akan dilewatkan ke dalam fungsi. Adapun contoh hasil yang akan diberikan dari program di atas adalah seperti yang tertera di bawah ini.

```
Masukkan nilai jari-jari lingkaran : 5
Keliling lingkaran dengan jari-jari 5 : 31.415900
```

c. Parameter Masukan/Keluaran

Selain parameter masukan dan keluaran, terdapat parameter jenis lain, yaitu parameter masukan/keluaran dimana parameter tersebut mempunyai dua buah kegunaan, yaitu sebagai berikut:

- Pertama parameter ini akan bertindak sebagai parameter yang menampung nilai masukan
- Setelah itu, parameter ini akan bertindak sebagai parameter yang menampung nilai keluaran

Untuk lebih memahaminya, berikut ini diberikan contoh program dimana di dalamnya terdapat sebuah parameter yang berperan sebagai parameter masukan/keluaran. Adapun sintak programnya adalah seperti yang terlihat di bawah ini.

```
#include <stdio.h>
#define PI 3.14159

/* Mendefinisikan fungsi dengan parameter masukan/keluaran */
void HitungKelilingLingkaran(double *X) {
  *X = 2 * PI * (*X);
}

/* Fungsi Utama */
int main(void) {
  int R;
  double param;
  printf("Masukkan nilai jari-jari lingkaran : ");
  scanf("%d", &int);

  /* Melakukan typecast dari tipe int ke tipe double */
  param = (double) R;

  HitungKelilingLingkaran(&param);
  printf("Keliling lingkaran dengan jari-jari %d : %f", R,
          param);
  return 0;
}
```

Apabila dijalankan program di atas akan memberikan hasil sebagai berikut:

```
Masukkan nilai jari-jari lingkaran : 4
Keliling lingkaran dengan jari-jari 4 : 25.132720
```

4.1.2. Melewatkan Parameter Berdasarkan Nilai (*Pass By Value*)

Terdapat dua buah cara untuk melewatkan parameter ke dalam sebuah fungsi, yaitu dengan cara melewatkan berdasarkan nilainya (*pass by value*) dan berdasarkan alamatnya (*pass by reference*). Namun pada bagian ini hanya akan dibahas mengenai pelewatan parameter berdasarkan nilai saja, sedangkan untuk pelewatan parameter berdasarkan alamat akan kita bahas pada sub bab berikutnya.

Pada pelewatan parameter berdasarkan nilai, terjadi proses penyalinan (*copy*) nilai dari parameter formal ke parameter aktual. Hal ini akan meyebabkan nilai variabel yang dihasilkan oleh proses di dalam fungsi tidak akan berpengaruh terhadap nilai variabel yang terdapat di luar fungsi. Untuk lebih memahaminya, perhatikan contoh program berikut dimana di dalamnya terdapat sebuah parameter yang dilewatkan dengan cara *pass by value*.

```
#include <stdio.h>
```

```
/* Mendefinisikan fungsi dengan melewatkan parameter berdasarkan
nilai */
void TambahSatu(int X) {
  X++;
  /* Menampilkan nilai yang terdapat di dalam fungsi */
  printf("Nilai di dalam fungsi : %d\n", X);
}

/* Fungsi Utama */
int main(void) {
  int Bilangan;
  printf("Masukkan sebuah bilangan bulat : ");
  scanf("%d", &Bilangan);

  /* Menampilkan nilai awal */
  prinf("\nNilai awal : %d\n", Bilangan);

  /* Memanggil fungsi TambahSatu */
  TambahSatu(Bilangan);

  /* Menampilkan nilai akhir */
  prinf("Nilai akhir : %d\n", Bilangan);

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
Masukkan sebuah bilangan bulat : 10

Nilai awal : 10
Nilai di dalam fungsi : 11
Nilai akhir : 10
```

Seperti yang kita lihat pada hasil program di atas bahwa nilai dari variabel Bilangan tetap bernilai 10 walaupun kita telah memanggil fungsi `TambahSatu()`. Hal ini disebabkan karena variabel `Bilangan` dan variabel `X` merupakan dua variabel yang tidak saling berhubungan dan menempati alamat memori yang berbeda sehingga yang terjadi hanyalah proses penyalinan (peng-*copy*-an) nilai dari variabel `Bilangan` ke variabel `X`. Dengan demikian, perubahan nilai variabel `X` tentu tidak akan mempengaruhi nilai variabel `Bilangan`.

4.1.3. Melewatkan Parameter Berdasarkan Alamat (*Pass By Reference*)

Di sini, parameter yang dilewatkan ke dalam fungsi bukanlah berupa nilai, melainkan suatu alamat memori. Pada saat kita melewatkan parameter berdasarkan alamat, terjadi proses referensial antara variabel yang terdapat pada parameter formal dengan variabel yang terdapat parameter aktual. Hal tersebut menyebabkan kedua variabel tersebut akan berada pada satu alamat di memori yang sama sehingga apabila terdapat perubahan nilai

terhadap salah satu dari variabel tersebut, maka nilai variabel satunya juga akan ikut berubah. Untuk melakukan hal itu, parameter fungsi tersebut harus kita jadikan sebagai pointer (untuk informasi lebih detil mengenai pointer, lihat bab 7 - *Pointer*).

Agar dapat lebih memahami materi ini, di sini kita akan menuliskan kembali kasus di atas ke dalam sebuah program. Namun, sekarang kita akan melakukannya dengan menggunakan cara *pass by reference*. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>

/* Mendefinisikan fungsi dengan melewatkan parameter berdasarkan
alamat */
void TambahSatu(int *X) {
  (*X)++;
  /* Menampilkan nilai yang terdapat di dalam fungsi */
  printf("Nilai di dalam fungsi : %d\n", *X);
}

/* Fungsi Utama */
int main(void) {
  int Bilangan;
  printf("Masukkan sebuah bilangan bulat : ");
  scanf("%d", &Bilangan);

  /* Menampilkan nilai awal */
  prinf("\nNilai awal : %d\n", Bilangan);

  /* Memanggil fungsi TambahSatu dengan mengirimkan
     alamat variabel Bilangan */
  TambahSatu(&Bilangan);

  /* Menampilkan nilai akhir */
  prinf("Nilai akhir : %d\n", Bilangan);

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
Masukkan sebuah bilangan bulat : 10

Nilai awal : 10
Nilai di dalam fungsi : 11
Nilai akhir : 11
```

Seperti yang kita lihat di atas bahwa pada proses *pass by reference*, perubahan nilai di dalam fungsi akan mempengaruhi nilai di luar fungsi.

5.6. Melewatkan Parameter `argc` dan `argv` ke dalam Fungsi `main()`

Kita dapat melewatkan parameter `argc` dan `argv` ke dalam fungsi main() untuk mengetahui berapa banyak dan apa saja parameter yang dikirimkan oleh sistem operasi ke dalam program. Parameter **`argc`** (*argument for count*) merupakan parameter bertipe `int` dan berfungsi untuk menunjukkan banyaknya parameter yang digunakan dalam eksekusi program, sedangkan parameter **`argv`** (*argument for vector*) merupakan pointer ke string yang akan menyimpan parameter-parameter apa saja yang digunakan dalam eksekusi program. Parameter `argc` tidak dapat bernilai negatif, artinya apabila tidak terdapat parameter yang dilewatkan sekalipun maka `argc` ini akan bernilai 1 dan `argv[0]` selalu menunjuk ke nama program yang dieksekusi. Apabila `argc` bernilai lebih besar dari satu, maka parameter `argv` akan menampung parameter yang dilewatkan tersebut ke dalam `argv[1]` sampai `argv[argc-1]`. Adapun bentuk umum untuk melewatkan parameter `argc` dan `argv` ke dalam fungsi `main()` adalah sebagai berikut.

```
int main(int argv, char *argv[]) {
  ...
}
```

Untuk menunjukkan kegunaan parameter `argc` dan `argv`, di sini kita akan membuat contoh program sederhana. Adapun sintaknya adalah sebagai berikut.

```
int main(int argc, char *argv[]) {
  /* Menampilkan banyaknya parameter yang ditampung di dalam
     variabel argc */
  printf("Banyaknya parameter : %d\n\n", argc);

  /* Menampilkan nilai dari argv[0] */
  if (argc > 0) {
    printf("Nama program (Parameter ke-0) : %s\n\n", argv[0]);
  }

  /* Menampilkan nilai dari argv[1] sampai argv[argc-1] */
  if (argc > 1) {
    for (int j=0; j<argc; j++) {
      printf("Parameter ke-%d : %s\n", j, argv[j]);
    }
  }
  return 0;
}
```

Simpanlah sintak program di atas ke dalam file 'info.c' (misalnya terdapat pada direktori C:\Program). Selanjutnya compile kode tersebut dan jalankan dengan perintah berikut.

```
C:\Program>info param1 param2 param3
```

Maka hasil yang akan diberikan oleh program adalah seperti yanag terlihat di bawah ini.

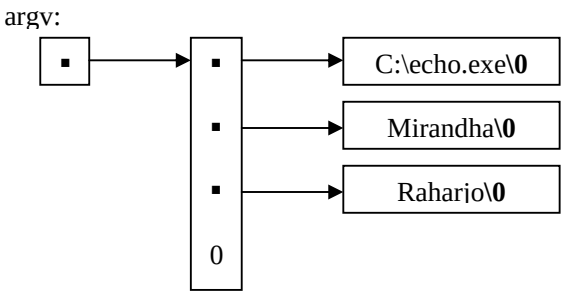
```
Banyaknya parameter : 4

Nama program (Parameter ke-0) : C:\Program\info.exe

Parameter ke-1 : param1
Parameter ke-2 : param2
Parameter ke-3 : param3
```

Sebagai contoh lain adalah program **echo** (dalam sistem operasi DOS) yang tentunya sudah Anda kenal dengan baik. Apabila kita menuliskan eksekusi program dengan perintah

```
C:>echo Mirandha Raharjo
```

Maka argc akan bernilai 3, `argv[0]` bernilai 'C:\echo.exe', `argv[1]` bernilai 'Mirandha' dan `argv[2]` bernilai 'Raharjo'.

Kejadian tersebut dapat kita ilustrasikan dengan gambar di bawah ini.



5.7. Membuat Prototipe Fungsi

Secara *default*, pendefinisian fungsi-fungsi di dalam program akan dilakukan sebelum fungsi `main()`. Namun bahasa C telah mengizinkan kita untuk dapat mendefinisikan fungsi-fungsi lain tersebut setelah melakukan penulisan fungsi `main()` asalkan kita menuliskan prototipe dari fungsi-fungsi tersebut sebelum fungsi `main()`.

Berikut ini contoh yang akan menunjukkan pembuatan program dengan melakukan prototipe fungsi. Di sini kita akan membuat program yang dapat melakukan konversi suhu dari Celcius ke Fahrenheit, yaitu dengan menggunakan persamaan berikut.

```
(F - 32) / C = 9 / 5
```

Adapun sintak program yang dimaksud tersebut adalah seperti yang tertulis di bawah ini.

```
#include <stdio.h>

/* Membuat prototipe dari fungsi CelciusKeFahrenheit() */
float CelciusKeFahrenheit(float suhu);

int main(void) {
  float C, F;
  printf("Masukkan suhu yang akan dikonversi : ");
  scanf("%f", &C);

  /* Memanggil fungsi CelciusKeFahrenheit() dan menampungnya ke
     variabel F*/
  F = CelciusKeFahrenheit(C);

  /* Menampilkan nilai hasil konversi */
  printf("%.2f C = %.2f F");

  return 0;
}

/* Implementasi fungsi CelciusKeFahrenheit */
float CelciusKeFahrenheit(float suhu) {
  float hasil;
  hasil = ((9 * suhu) / 5) + 32;
  return hasil;
}
```

Contoh hasil yang akan diperoleh dari program di atas adalah sebagai berikut.

```
Masukkan suhu yang akan dikonversi : 100
100.00 C = 212.00 F
```

5.8. Rekursi

Rekursi adalah proses pemanggilan fungsi oleh dirinya sendiri secara berulang. Istilah 'rekursi' sebenarnya berasal dari bahasa Latin 'recursus', yang berarti 'menjalankan ke belakang'. Rekursi digunakan untuk penyederhanaan algoritma dari suatu proses sehingga program yang dihasilkan menjadi lebih efesien. Pada bagian ini kita akan mempelajarinya langsung melalui contoh-contoh program.

5.8.1. Menentukan Nilai Faktorial

Pada bagian ini kita akan membuat sebuah fungsi rekursif untuk menentukan nilai faktorial dengan memasukkan nilai yang akan dihitung sebagai parameter fungsi ini. Sebagai contoh apabila parameter yang kita masukkan adalah 5, maka hasilnya adalah

5! = 5 x 4 x 3 x 2 x 1 = 120

Proses tersebut dapat kita sederhanakan melalui fungsi matematis sebagai berikut.

F! (N) = N * F! (N-1)

Namun yang harus kita perhatikan di sini adalah F! (0) = 1, ini adalah suatu tetapan numerik yang tidak dapat diubah. Berikut ini contoh implementasi kasus tersebut ke dalam sebuah program.

```
#include <stdio.h>

/* Mendefinisikan fungsi untuk menghitung nilai faktorial */
int Faktorial(int N) {
  if (N == 0) {
    return 1;
  }else {
    return N * Faktorial(N-1);
  }
}

int main(void) {
  int bilangan;
  printf("Masukkan bilangan yang akan dihitung : ");
  scanf("%d", &bilangan);
  printf("%d! = %d", bilangan, Faktorial(bilangan));
  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Masukkan bilangan yang akan dihitung : 5
5! = 120
```

Konsep dari proses di atas sebenarnya sederhana, yaitu dengan melakukan pemanggilan fungsi `Faktorial()` secara berulang. Untuk kasus ini, proses yang dilakukan adalah sebagai berikut.

Faktorial(5) = 5 * Faktorial(4)
    Faktorial(4) = 4 * Faktorial(3)
        Faktorial(3) = 3 * Faktorial(2)

Faktorial(2) = 2 * Faktorial(1)
Faktorial(1) = 1 * Faktorial(0)
Faktorial(0) = 1
Faktorial(1) = 1 * 1
Faktorial(2) = 2 * 1
Faktorial(3) = 3 * 2
Faktorial(4) = 4 * 6
Faktorial(5) = 5 * 24
= **120**

5.8.2. Menentukan Nilai Perpangkatan

Sekarang kita akan melakukan rekursi untuk menghitung nilai B^N, dimana B adalah bilangan basis dan N adalah nilai eksponen. Kita dapat merumuskan fungsi tersebut seperti di bawah ini.

$B^N = B * B^{N-1}$

Dengan demikian apabila kita implementasikan ke dalam program, maka sintaknya kurang lebih sebagai berikut.

```
#include <stdio.h>

/* Mendefinisikan fungsi untuk menghitung nilai eksponensial */
int Pangkat(int basis, int e) {
  if (e == 0) {
    return 1;
  }else {
    return basis * Pangkat(basis, e-1);
  }
}

int main(void) {
  int B, N;
  printf("Masukkan bilangan basis : "); scanf("%d", &B);
  printf("Masukkan bilangan eksponen : "); scanf("%d", &N);
  printf("%d^%d = %d", B, N, Pangkat(B, N));
  return 0;
}
```

Contoh hasil yang akan diberikan adalah seperti yang terlihat di bawah ini.

```
Masukkan bilangan basis : 2
Masukkan bilangan eksponen : 5
2^5 = 32
```

Proses yang terdapat di atas adalah sebagai berikut.

```
Pangkat(2, 5) = 2 * Pangkat(2, 4)
    Pangkat(2, 4) = 2 * Pangkat(2, 3)
        Pangkat (2, 3) = 2 * Pangkat(2, 2)
            Pangkat(2, 2) = 2 * Pangkat(2, 1)
                Pangkat(2, 1) = 2 * Pangkat(2, 0)
                    Pangkat(2, 0) = 1
                Pangkat(2, 1) = 2 * 1
            Pangkat(2, 2) = 2 * 2
        Pangkat(2, 3) = 2 * 4
    Pangkat(2, 4) = 2 * 8
Pangkat(2, 5) = 2 * 16
              = 32
```

5.8.3. Konversi Bilangan Desimal ke Bilangan Biner

Kali ini kita akan membuat sebuah fungsi rekursif tanpa nilai balik yang digunakan untuk melakukan konversi bilangan desimal ke bilangan biner. Untuk informasi lebih detil mengenai bilangan biner dan heksadesimal, Anda dapat melihat lampiran B – *Bit dan Byte* di bagian akhir buku ini. Artinya, di sini kita tidak akan membahas bagaimana proses pengkonversian tersebut, melainkan kita lebih berkonsentrasi ke pembahasan mengenai rekursi.

Adapun sintak program untuk melakukan hal tersebut adalah sebagai berikut.

```
#include <stdio.h>

void DesimalKeBiner(int n) {
  if (n>1) {
    DesimalKeBiner(n/2);
  }
  printf("%d", n%2);
}

int main(void)
{
  int a;
  printf("Masukkan bilangan desimal yang akan dikonversi : ");
  scanf("%d",&a);
  printf(“%d dalam biner : ”, a)
  DesimalKeBiner(a);

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Masukkan bilangan desimal yang akan dikonversi : 129
129 dalam biner : 10000001
```

5.8.4. Konversi Bilangan Desimal ke Bilangan Heksadesimal

Pada sub bab sebelumnya Anda telah mempelajari bagaimana melakukan rekursi dari konversi bilangan desimal ke bilangan biner. Sekarang kita akan membahas juga bagaimana melakukan rekursi dari konversi bilangan desimal ke bilangan heksadesimal. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>

void DesimalKeHeksa(int n) {
  char *daftarheksa[] =
          {"0","1","2","3","4","5","6","7","8","9",
           "A","B","C","D","E","F"};
  if (n>15) {
    DesimalKeHeksa(n/16);
  }
  printf("%s", daftarheksa[n%16]);
}

int main(void)
{
  int a;
  printf("Masukkan bilangan yang akan dikonversi : ");
  scanf("%d",&a);
  printf("%d dalam heksadesimal : ",a);
  DesimalKeHeksa(a);

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Masukkan bilangan yang akan dikonversi : 1000
1000 dalam heksadesimal : 3E8
```

Bab

6

Array dan String

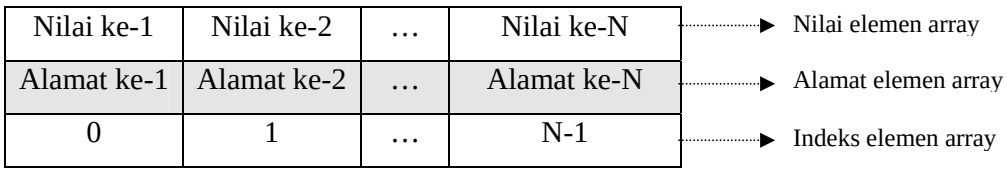
6.1. Pendahuluan

Efisiensi merupakan faktor utama yang perlu diperhatikan dalam pembuatan sebuah program, baik dari segi kecepatan, minimalisasi penggunaan memori maupun ketepatan algoritma. Maka dari itu, seorang programmer haruslah mengetahui semua hal dan teknik yang diperlukan. Salah satu materi yang akan kita bahas pada bab ini adalah penggunaan array.

Dalam buku ini, kita akan membahas array dan string secara bersamaan, alasannya adalah karena pemrosesan string dalam bahasa C selalu berhubungan dengan array. Memang kebanyakan dari buku C yang beredar, pembahasan array biasanya disatukan dengan pembahasan mengenai pointer. Namun, dalam buku ini pembahasan pointer akan diterangkan dalam bab tersendiri. Hal ini bertujuan agar Anda dapat lebih mudah untuk memahami secara detil dari materi-materi yang akan disampaikan dalam buku ini.

6.2. Apa Itu Array?

Menurut definisinya, array (larik) adalah suatu variabel yang merepresentasikan daftar (*list*) atau kumpulan data yang memiliki tipe data sama. Setiap data yang terdapat dalam array tersebut menempati alamat memori yang berbeda serta disebut dengan elemen array. Selanjutnya untuk mengakses nilai dari suatu elemen array, kita akan menggunakan indeks dari array tersebut. Perlu sekali untuk diperhatikan bahwa dalam bahasa C, indeks array selalu dimulai dari angka 0, *bukan 1*. Hal ini berbeda dengan bahasa pemrograman lainnya (misalnya bahasa Pascal) dimana indeks awal array dapat ditentukan sendiri sesuai dengan keinginan kita. Berikut ini gambar yang akan mengilustrasikan sebuah array dalam bahasa C.



Gambar 6.1. Array

Untuk mendeklarasikan suatu array satu dimensi dalam bahasa C adalah dengan menggunakan tanda `[]` (*bracket*). Adapun bentuk umum dari pendeklarasian tersebut adalah sebagai berikut.

```
tipe_data nama_array [banyak_elemen];
```

Sebagai contoh apabila kita ingin mendeklarasikan array dengan nama `A` sebanyak 100 elemen data yang bertipe `int`, maka kita akan menuliskan sintaknya sebagai berikut.

```
int A[100];
```

Ruang memori yang dibutuhkan untuk array tersebut adalah 400 byte, yang berasal dari 100 x 4 byte, dimana 4 byte di sini merupakan ukuran dari tipe data `int`. Begitu pun apabila kita akan mendeklarasikan suatu array yang bertipe `char` sebanyak 10 karakter, maka memori yang dibutuhkan adalah 10 byte, yaitu dari 10 x 1 byte (1 byte adalah ukuran dari tipe data `char`). Sekarang kita kembali ke array `A` yang telah kita deklarasikan di atas. Di sini kita dapat merepresentasikannnya melalui gambar berikut.

Gambar 6.2. Array dengan 10 elemen bertipe `int`

`A[0]` di atas menunjukkan nilai yang dikandung oleh elemen pertama array `A`. `A[1]` menunjukkan elemen kedua, dan begitu seterusnya sampai `A[99]` yang menunjukkan elemen ke-100.

Adapun cara untuk memasukkan nilai ke dalam setiap elemen array adalah sama seperti memasukkan nilai terhadap suatu variabel biasa. Sebagai contoh.

```
A[0] = 1;
A[1] = 2;
A[2] = 3;
A[3] = 4;
A[4] = 5;
A[5] = 6;
...
A[99] = 100;
```

Namun, untuk efesiensi program, proses memasukkan nilai ke dalam elemen array umumnya dilakukan dengan menggunakan pengulangan. Berikut ini contoh program yang akan menunjukkan cara memasukkan sekaligus menampilkan nilai dari suatu array.

```
#include <stdio.h>

#define MAX 5

int main() {
  int A[MAX];

  int j;
```

```
  /* Memasukkan nilai ke dalam elemen array*/
  printf("Memasukkan nilai:\n");
  for (j=0; j<MAX; j++) {
    printf("A[%d] = ", j); scanf("%d", &A[j]);
  }

  /* Menampilkan nilai dari elemen array*/
  printf("\nMenampilkan nilai:\n");
  for (j=0; j<MAX; j++) {
    printf("A[%d] = %d\n", j, A[j]);
  }

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Memasukkan nilai:
A[0] = 10
A[1] = 20
A[2] = 30
A[3] = 40
A[4] = 50

Menampilkan nilai:
A[0] = 10
A[1] = 20
A[2] = 30
A[3] = 40
A[4] = 50
```

6.3. Mengapa Harus Menggunakan Array?

Untuk mengetahui mengapa kita harus menggunakan array, asumsikan bahwa kita mempunyai lima buah variabel `x1`, `x2`, `x3`, `x4` dan `x5` yang semuanya bertipe `float`. Selanjutnya kita ingin mengetahui nilai rata-rata dari variabel tersebut. Apabila kita tidak menggunakan array untuk kasus ini, maka kita tentu akan menuliskan program seperti di bawah ini.

```
#include <stdio.h>

int main() {

  /* Mendeklarasikan variabel x1, x2, x3, x4 dan x5 yang
     semuanya bertipe float */
  float x1, x2, x3, x4, x5;
```

```
  float rata_rata;    /* Variabel untuk menampung hasil
                         perhitungan */

  /* Memasukkan nilai untuk variabel x1, x2, x3, x4 dan x5 */
  printf(“Memasukkan nilai:\n”)
  printf(“Nilai ke-1 = ”); scanf(“%f”, &x1);
  printf(“Nilai ke-2 = ”); scanf(“%f”, &x2);
  printf(“Nilai ke-3 = ”); scanf(“%f”, &x3);
  printf(“Nilai ke-4 = ”); scanf(“%f”, &x4);
  printf(“Nilai ke-5 = ”); scanf(“%f”, &x5);

  /* Melakukan perhitungan untuk menghasilkan rata-rata */
  rata_rata = (x1 + x2 + x3 + x4 + x5) / 5;

  /* Menampilkan hasil perhitungan */
  printf(“\nNilai rata-rata = %.2f”, rata_rata);

  return 0;
}
```

Program di atas memang pendek karena data yang kita miliki hanya 5 buah. Namun bagaimana apabila kita ingin menghitung rata-rata dari 100 atau bahkan 1000 data? Kita tentu tidak akan menggunakan cara di atas bukan? Oleh sebab itu, untuk menyederhanakan program dalam kasus ini, seharusnya kita menggunakan array. Berikut ini program yang merupakan perbaikan dari program sebelumnya.

```
#include <stdio.h>

#define MAX 5

int main() {
  float A[MAX], jumlah=0, rata_rata;
  int j;

  /* Memasukkan nilai ke dalam elemen array*/
  printf("Memasukkan nilai:\n");
  for (j=0; j<MAX; j++) {
    printf("A[%d] = ", j); scanf("%f", &A[j]);
    jumlah += A[j];
  }
  /* Melakukan proses perhitungan */
  rata_rata = jumlah / MAX;
  /* Menampilkan hasil perhitungan */
  printf(“\nNilai rata-rata = %.2f”, rata_rata);

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Memasukkan nilai:
A[0] = 12.42
A[1] = 20.35
A[2] = 18.55
A[3] = 21.86
A[4] = 15.95

Nilai rata-rata = 17.83
```

6.4. Inisialisasi Array

Pada saat array dideklarasikan, apabila kita tidak melakukan inisialisasi nilai secara ekplisit terhadap elemen-elemen di dalamnya, maka kompilator C akan secara otomatis mengeset nilai dari setiap elemen tersebut ke nol.

Pada kasus-kasus tertentu, kadang kala kita juga dituntut untuk melakukan inisialisasi nilai terhadap elemen-elemen array. Nilai yang kita definisikan dalam proses inisialisasi tersebut akan menjadi nilai default bagi elemen bersangkutan. Artinya apabila dalam program kita tidak mengganti nilai pada suatu elemen tertentu, maka yang akan digunakan adalah nilai default yang telah didefinisikan.

Berikut ini cara untuk melakukan inisialisasi nilai terhadap elemen array. Sebagai contoh apabila kita memiliki tiga buah array dengan nama A, B dan C yang semuanya bertipe int dan kita akan melakukan inisialisasi terhadap elemen-elemennya, maka kita dapat menuliskannya dengan cara sebagai berikut.

```
int A[5] = { 10, 20, 30, 40, 50 };
int B[5] = {10};
int C[5] = { 10, 0, 30 };
```

Pada baris pertama (array A), kita melakukan inisialisasi terhadap kelima elemen array, yaitu dengan nilai 10, 20, 30, 40 dan 50 sehingga array A dapat digambarkan sebagai berikut.

10	20	30	40	50
A[0]	A[1]	A[2]	A[3]	A[4]

Sedangkan pada baris kedua (array B), kita hanya melakukan inisialisasi terhadap elemen pertama saja (yaitu dengan nilai 10), artinya elemen lainnya masih kosong sehingga dapat digambarkan sebagai berikut.

10	0	0	0	0
B[0]	B[1]	B[2]	B[3]	B[4]

Baris terakhir (array C), kita melakukan inisialisasi terhadap elemen ke-1 dan ke-3, yaitu dengan nilai 10 dan 30. Namun perlu diperhatikan di sini bahwa kita juga harus mengisikan nilai 0 untuk elemen ke-2. Apabila kita hanya menuliskan

```
int C[5] = { 10, 20 };
```

maka *kompilator* akan menganggap kita melakukan inisialisasi terhadap elemen ke-1 dan ke-2 (bukan elemen ke-3), sehingga kita harus menuliskannya sebagai berikut.

```
int C[5] = { 10, 0, 30 };
```

Hal ini akan menyebabkan nilai dari array C dapat digambarkan seperti di bawah ini.

10	0	**30**	0	0
C[0]	C[1]	**C[2]**	C[3]	C[4]

Untuk membuktikan hal di atas, marilah kita implementasikan pernyataan di atas ke dalam sebuah program. Adapun sintaknya adalah sebagai berikut.

```
#include <stdio.h>

int main() {
  int A[5] = {10, 20, 30, 40, 50};
  int B[5] = {10};
  int C[5] = {10, 0, 30};
  int j;

  /* Menampilkan nilai dari elemen array */
  for (j=0; j<5; j++) {
    printf("A[%d] = %2d, B[%d] = %2d, C[%d] = %2d\n", j, A[j],
           j, B[j], j, C[j]);
  }

  return 0;
}
```

Hasil yang akan diberikan oleh program di atas adalah sebagai berikut.

```
A[0] = 10, B[0] = 10, C[0] = 10
A[1] = 20, B[1] =  0, C[1] =  0
A[2] = 30, B[2] =  0, C[2] = 30
A[3] = 40, B[3] =  0, C[3] =  0
A[4] = 50, B[4] =  0, C[4] =  0
```

Berikut ini contoh lain dari proses inisialisasi array di dalam bahasa C.

```
int digit[]     = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
char huruf[]    = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
                   'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q',
                   'r', 's', 't', 'u', 'v', 'w', 'x',
                   'y', 'z'};
char vokal[]    = {'a','i','u','e','o'};
```

Array `digit` di atas akan menghasilkan 10 elemen dengan ukuran 40 byte (10 x 4 byte), array `huruf` menghasilkan 26 elemen dengan ukuran 26 byte (26 x 1 byte), sedangkan array `vokal` menghasilkan 5 elemen dengan ukuran 5 byte (5 x 1 byte).

6.5. Array Konstan

Sama seperti pada variabel biasa, elemen array juga dapat dibuat menjadi konstan, artinya nilanya tidak dapat diubah selama program berjalan. Caranya sederhana, yaitu dengan menambahkan kata kunci `const` pada saat pendeklarasian array serta mengisikan nilai-nilai yang akan didefinisikan. Berikut ini bentuk umumnya.

```
const tipe_data nama_array[N] = { nilai1, nilai2, …, nilaiN
```

Sebagai contoh apabila kita memiliki array dengan nama `A` dengan tipe data `int`, dimana nilai dari setiap elemennya akan kita buat menjadi konstan, maka kita akan mendeklarasikannya sebagai berikut.

```
const int A[3] = { 10, 20, 30 };
```

Kita tidak dapat mengubah nilai dari elemen-elemen tersebut, perhatikan potongan sintak program berikut.

```
A[0] = 100;     /* SALAH, karena nilai A[0] akan selalu
                   bernilai 10 */
A[1] = 150;     /* SALAH, karena nilai A[1] akan selalu
                   bernilai 20 */
A[2] = 200;     /* SALAH, karena nilai A[2] akan selalu
                   bernilai 30 */
```

Dengan kata lain, nilai-nilai yang terdapat pada array konstan hanya digunakan untuk kepentingan akses program. Misalnya nilai tersebut akan kita gunakan untuk melakukan proses perkalian seperti yang terdapat pada sintak berikut.

```
for (int j=0; j<3; j++) {
  printf(“%d x %d = %d\n”, A[j], j, (A[j] * j)) ;
}
```

6.6. Array sebagai Parameter

Dalam kasus-kasus tertentu, kita dituntut untuk menggunakan array sebagai parameter dalam suatu fungsi. Hal ini biasanya ditemukan pada saat kita ingin membuat fungsi untuk pencarian data dalam array. Sebagai contoh, di sini kita akan membuat fungsi yang akan digunakan untuk mengetahui nilai rata-rata dari elemen-elemen array yang bertipe `float`.

Untuk menyelesaikan kasus ini, sebelumnya kita akan membuat sebuah fungsi yang berguna untuk memasukkan nilai ke dalam elemen array. Berikut ini definisi fungsi tersebut.

```
void InputArray(float A[], int N) {
  for (int j=0; j<N; j++) {
    printf(“A[%d] = ”, j); scanf(“%d”, &A[j]);
  }
}
```

Pada fungsi di atas, kita melewatkan array dengan nama `A` yang bertipe `float` sebagai parameter. Sedangkan `N` di sini digunakan untuk menentukan banyaknya elemen array yang akan dimasukkan.

Setelah itu kita akan membuat fungsi untuk menjumlahkan nilai-nilai dari suatu array. Adapun definisi fungsinya adalah sebagai berikut.

```
int HitungRataRata(float A[], int N) {
  int jumlah = 0;
  for (int j=0; j<N; j++) {
    jumlah += A[j];
  }
  return (jumlah / N);
}
```

Parameter `A` di atas menunjukkan array yang elemen-elemennya akan diproses untuk dihitung nilai rata-ratanya. Sedangkan parameter `N` merupakan banyaknya elemen yang terdapat pada array `A`.

Agar Anda dapat melihat hasilnya, berikut ini program yang akan menggunakan fungsi-fungsi di atas. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>

#define MAX 100

/* Fungsi untuk memasukkan nilai ke dalam elemen array */
void InputArray(float A[], int N) {
  for (int j=0; j<N; j++) {
    printf("A[%d] = ", j); scanf("%d", &A[j]);
  }
}

/* Fungsi untuk menghitung nilai rata-rata */
float HitungRataRata(float A[], int N) {
  float jumlah = 0;
  for (int j=0; j<N; j++) {
    jumlah += A[j];
  }
  return (jumlah / N);
}

int main() {
  float Arr[MAX];
  int n;  /* Variabel untuk menampung banyak elemen array */
  float rata_rata; /* Variabel untuk menampung nilai rata-rata*/
  printf("Masukkan banyaknya elemen yang diinginkan : ");
  scanf("%d", &n);

  /* Memanggil fungsi InputArray */
  InputArray(Arr, n);

  /* Memanggil fungsi HitungRataRata dan
     menampilkannya ke layar */
  printf("\n\nNilai rata-rata = %.2f", HitungRataRata(Arr, n));

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah seperti yang tampak di bawah ini.

```
Masukkan banyaknya elemen yang diinginkan : 5
A[0] = 12.50
A[1] = 14.13
A[2] = 10.74
A[3] = 15.88
A[4] = 16.35

Nilai rata-rata = 13.92
```

6.7. Array sebagai Tipe Data Bentukan

Seperti yang telah dikatakan pada sebelumnya bahwa array juga dapat digunakan sebagai tipe data bentukan. Artinya, array ini akan digunakan untuk mendeklarasikan variabel lain. Untuk melakukan hal ini, bahasa C telah menyediakan kata kunci `typedef`. Sebagai contoh, di sini kita akan membuat tipe data bentukan yang bertipe array (misalnya dengan nama `LARIK`). Adapun cara untuk melakukan hal tersebut adalah dengan mendeklarasikannya sebagai berikut.

```
typedef int LARIK[100];
```

Sampai di sini kita telah mempunyai tipe data `LARIK` yang dapat digunakan untuk mendeklarasikan variabel lain. Perhatikan contoh penggunaan tipe data `LARIK` di bawah ini.

```
LARIK X;
```

Sintak di atas akan memerintahkan kompilator untuk mendeklarasikan array `X` dengan 100 buah elemen bertipe `int`. Untuk memudahkan pemahaman, sintak tersebut dapat dituliskan seperti di bawah ini.

```
int X[100];
```

Sebagai bukti pernyataan di atas, perhatikan contoh program berikut.

```
#include <stdio.h>

typedef int LARIK[100];

/* Mendefinisikan fungsi untuk menghitung jumlah (sum) dari
elemen array */
int JumlahArray(int A[], int N) {
  int jumlah=0;
  for(int j=0; j<N; j++) {
    jumlah += A[j];
  }
  return jumlah;
}

int main() {

  /* Mendeklarasikan variabel X dengan tipe LARIK dengan
     inisialisasi 5 elemen */
  LARIK X = { 10, 20, 30, 40, 50 };
```

```
  /* Menampilkan nilai yang terdapat pada elemen ke-1 sampai
     ke-5 array X */
  for (int j=0; j<5; j++) {
    printf("A[%d] = %d\n", j, X[j]);
  }

  /* Menampilkan jumlah elemen array X dengan memanggil fungsi
     JumlahArray */
  printf("\n\nJumlah = %d", JumlahArray(X, 5));

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
A[0] = 10
A[1] = 20
A[2] = 30
A[3] = 40
A[4] = 50

Jumlah = 150
```

6.8. Pencarian pada Elemen Array

Pencarian data di dalam array merupakan suatu hal dasar yang sering dijumpai dalam kasus-kasus pemrograman. Sebagai salah satu contoh nyata adalah apabila kita akan menghapus nilai elemen dari suatu array tertentu, maka secara tidak langsung kita harus melakukan pencarian terlebih dahulu. Apabila data yang dicari ditemukan, maka data tersebut baru akan dihapus. Contoh lainnya adalah pada saat kita ingin menentukan nilai maksimal atau minimal dari suatu kumpulan nilai tertentu, di sini kita juga akan melakukannya melalui suatu pencarian.

Karena alasan tersebut, seorang programmer haruslah memiliki kemampuan untuk melakukan pencarian data secara cepat dan tepat. Maka dari itu, pada bab ini kita akan membahas secara sekilas bagaimana cara melakukan pencarian elemen array di dalam bahasa C melalui contoh-contoh program.

Menurut teori algoritma dan pemrogaman, terdapat beberapa buah metode untuk melakukan pencarian dalam elemen array. Namun, di sini kita tidak akan membahas lebih detil mengenai metode-metode tersebut karena materi yang akan dibahas di sini lebih ditekankan kepada bagaimana bahasa C melakukan pencarian elemen di dalam array.

Untuk dapat lebih memahaminya, di sini kita akan membuat program yang akan menentukan apakan suatu nilai terdapat dalam array atau tidak. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>

#define MAX_ELEMEN 100

int main() {
  int BIL[MAX_ELEMEN];
  int N, X;

  printf("Masukkan banyaknya elemen yang diinginkan : ");
  scanf("%d",&N);

  /* Mengisikan elemen array */
  for (int j=0; j<N; j++) {
    printf("BIL[%d] = ", j); scanf("%d", &BIL[j]);
  }
  printf("Masukkan nilai yang akan dicari : "); scanf("%d", &X);

  /* Melakukan pencarian elemen array */
  int k=0;
  while ((k<N) && (BIL[k] != X)) {
    k++;
  }

  /* Menyimpulkan hasil pencarian */
  if (BIL[k] == X) {
    printf("%d ditemukan dalam array, yaitu pada indeks ke-%d",
           X, k);
  } else {
    printf("%d tidak ditemukan dalam array", X);
  }
  return 0;
}
```

Contoh hasil yang akan diberikan oleh program di atas adalah sebagai berikut.

```
Masukkan banyaknya elemen array yang diinginkan : 5
BIL[0] = 13
BIL[1] = 20
BIL[2] = 10
BIL[3] = 18
BIL[4] = 21
Masukkan nilai yang akan dicari : 18
18 ditemukan dalam array, yaitu pada indeks ke-3
```

Sebagai contoh lain yang banyak dijumpai adalah penentuan nilai maksimal dan minimal dari suatu array. Di sini kita akan membuat program untuk melakukan hal tersebut. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>
#include <conio.h>

int main() {

  /* Mendeklarasikan array dan menginisialiasi nilai
     ke dalamnya */
  int BIL[10] = { 13, 12, 14, 11, 10, 16, 15, 18, 19, 20 };

  int max=BIL[0], min=BIL[0];
  for (int j=0; j<10; j++) {

    /* Mencari nilai maksimal */
    if (BIL[j] > max) max = BIL[j];

    /* Mencari nilai minimal */
    if (BIL[j] < min) min = BIL[j];
  }

  /* Menampilkan nilai maksimal dan minimal */
  printf("Nilai maksimal : %d\n", max);
  printf("Nilai minimal  : %d", min);

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah seperti yang terlihat di bawah ini.

```
Nilai maksimal : 20
Nilai minimal  : 10
```

6.9. Pengurutan pada Elemen Array

Setelah mengetahui bagaimana melakukan pencarian terhadap elemen array, terdapat satu hal lagi yang tidak kalah pentingnya, yaitu melakukan pengurutan terhadap elemen-elemen array. Salah satu faktor dari lambatnya proses pencarian yang terapat dalam suatu kumpulan data adalah karena data tersebut tidak dalam keadaan terurut. Dengan kata lain, pengurutan sangatlah diperlukan sebelum kita melakukan pencarian data. Terdapat banyak metode pengurutan data yang terdapat dalam teori algoritma dan pemrograman, diantaranya metode gelembung (*bubble sort*), sisipan (*insertion sort*), maksimum-minimum (*maximum-minimum sort*), *quick sort* dan banyak lagi yang lainnnya.

Namun, dari banyak metode yang ada tersebut, di sini kita hanya akan menerangkan sekilas dari beberapa metode saja serta mengimplementasikannya langsung ke dalam sebuah program.

6.9.1. Menggunakan Metode Gelembung (*Bubble Sort*)

Menurut sumber yang ada, ide pembentukan metode ini diinspirasi oleh adanya pengapungan gelembung sabun yang terdapat pada permukaan air. Kita semua tahu bahwa berat jenis dari gelembung sabun lebih kecil daripada berat jenis air sehingga menyebabkan gelembung sabun tersebut selalu terapung ke atas permukaan. Sekarang perhatikan sebuah batu yang tenggelam di dalam air, hal ini tentu disebabkan oleh karena berat jenis batulebih besar daripada berat jenis air. Fenomena tersebut menunjukkan bahwa zat yang lebih ringan akan dilempar ke atas (diapungkan), sedangkan benda yang lebih berat akan dilempar ke bawah (ditenggelamkan).

Ide 'pengapungan' di atas kemudian digunakan untuk membentuk algoritma pengurutan data, yaitu dengan melempar elemen array terkecil ke bagian ujung kiri array (dijadikan sebagai elemen pertama). Proses tersebut tentunya dilakukan dengan melakukan pertukaran antara elemen array.

Sebagai contoh, apabila kita mempunyai sebuah array (misalnya dengan nama `A`) dengan 5 buah elemen sebagai berikut.

40	4	30	8	7
A[0]	A[1]	A[2]	A[3]	A[4]

Dalam metode ini, terdapat beberapa tahapan untuk membuat elemen-elemen di atas menjadi terurut secara menaik. Adapun tahapan tersebut adalah sebagai berikut.

Tahapan-1

Mulai dari elemen terakhir (`A[4]`) sampai elemen kedua (`A[1]`), lakukan perbandingan nilai antara `A[x]` dengan `A[x-1]` (dimana `x` mewakili indeks yang sedang aktif). Apabila `A[x]` lebih kecil maka lakukan pertukaran dengan `A[x-1]`. Untuk kasus di atas, elemen array akan berubah menjadi seperti di bawah ini.

4	40	7	30	8
A[0]	A[1]	A[2]	A[3]	A[4]

Tahapan-2

Mulai dari `A[4]` sampai `A[2]`, lakukan kembali perbandingan nilai seperti pada tahapan-1 sehingga sekarang array di atas akan terlihat seperti di bawah ini.

4	7	40	8	30
A[0]	A[1]	A[2]	A[3]	A[4]

Tahapan-3

Mulai dari `A[4]` sampai `A[3]`, lakukan kembali perbandingan nilai seperti pada tahapan-1. Proses pada tahapan ini akan memberikan hasil seperti di bawah ini.

4	7	8	40	30
A[0]	A[1]	A[2]	A[3]	A[4]

Tahapan-4

Tahapan ini adalah tahapan terakhir dimana kita akan melakukan perbandingan elemen terakhir (`A[4]`) dengan elemen terakhir-1 (`A[3]`). Apabila `A[4]` lebih kecil, maka

tukarkan nilainya dengan A[3] sehingga sampai di sini array akan terurut secara menaik seperti yang terlihat di bawah ini.

4	7	8	30	40
A[0]	A[1]	A[2]	A[3]	A[4]

Pada proses di atas, dapat kita lihat bahwa ada 4 tahapan yang harus dilalui untuk melakukan proses pengurutan 5 buah elemen array. Secara umum, apabila proses di atas akan kita terjemahkan ke dalam kode program dalam bahasa C, maka kita akan menuliskannya sebagai berikut.

```
int N;     /* Mendeklarasikan variabel untuk menampung banyak
              elemen array */
int j, k;  /* Mendeklarasikan variabel sebagai indeks
              pengulangan */
int temp;  /* Mendeklarasikan variabel sebagai variabel
              temporari */
for (j=0; j<N-1; j++) {
  for (k=N-1; k>=(j+1); k--) {
    if (A[k] < A[k-1]) {
      /* Melakukan proses pertukaran nilai antara A[k]
          dengan A[k-1] */
      temp = A[k];
      A[k] = A[k-1];
      A[k-1] = temp;
    }
  }
}
```

Untuk membuktikan hal tersebut, perhatikan program lengkap di bawah ini yang akan mengurutkan elemen-elemen yang terdapat pada array A di atas.

```
#include <stdio.h>

#define MAX 5

/* Mendefinisikan fungsi untuk menampilkan elemen-elemen dari
suatu array */
void TampilkanArray(int A[], int n) {
  for (int j=0; j<n; j++) {
    printf("A[%d] = %d\n", j, A[j]);
  }
}

int main() {
  /* Mendeklarasikan array dengan melakukan inisialisasi nilai
     ke dalamnya */
  int A[MAX] = { 40, 4, 30, 8, 7};
  int j, k;     /* Mendeklarasikan variabel sebagai indeks
                   pengulangan */
```

```
  int temp;     /* Mendeklarasikan variabel sebagai variabel
                   temporari */

  /* Menampilkan array sebelum diurutkan */
  printf("Sebelum pengurutan:\n");
  TampilkanArray(A, MAX);

  /* Melakukan proses pengurutan elemen array */
  for (j=0; j<MAX-1; j++) {
    for (k=MAX-1; k>=(j+1); k--) {
      if (A[k] < A[k-1]) {
        temp = A[k];
        A[k] = A[k-1];
        A[k-1] = temp;
      }
    }
  }
  printf("\n");

  /* Menampilkan array setelah diurutkan */
  printf("Setelah pengurutan:\n");
  TampilkanArray(A, MAX);
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah seperti yang terlihat di bawah ini.

```
Sebelum pengurutan:
A[0] = 40
A[1] = 4
A[2] = 30
A[3] = 8
A[4] = 7

Setelah pengurutan:
A[0] = 4
A[1] = 7
A[2] = 8
A[3] = 30
A[4] = 40
```

6.9.2. Menggunakan Metode Maksimum/Minimum (*Maximum/Minimum Sort*)

Metode ini merupakan metode yang relatif mudah untuk dipahami dan banyak digunakan. Konsep dari metode ini adalah menyimpan nilai maksimum ataupun nilai minimun ke bagian ujung array (elemen pertama ataupun terakhir). Setelah itu elemen tersebut akan ‘diikat’ dan tidak diikutkan lagi dalam proses selanjutnya.

Sebagai contoh untuk menerangkan metode ini, coba Anda perhatikan kembali array A yang terdapat pada sub bab sebelumnya.

40	4	30	8	7
A[0]	A[1]	A[2]	A[3]	A[4]

Di sini kita hanya akan melakukan pengurutan array di atas dengan menggunakan metode maksimum, yaitu dengan melemparkan nilai maksimal ke dalam elemen terakhir (paling kanan). Sedangkan metode minimum tidak akan kita bahas karena pada dasarnya konsepnya sama dengan metode maksimum. Adapun tahapan-tahapan yang akan dilakukan untuk melakukan pengurutan dengan metode maksimum adalah sebagai berikut.

Tahapan-1

Mulai dari A[0] sampai A[4], cari nilai maksimal dan tukarkan dengan elemen terakhir (A[4]). Untuk kasus ini, elemen array A akan menjadi seperti di bawah ini.

7	4	30	8	40
A[0]	A[1]	A[2]	A[3]	A[4]

Sampai di sini, A[4] akan diisolasi dan tidak akan diikutkan kembali pada tahapan berikutnya.

Tahapan-2

Mulai dari A[0] sampai A[3], cari nilai maksimal dan tukarkan dengan elemen terakhir (A[3]) sehingga sekarang array akan terlihat seperti di bawah ini.

7	4	8	30	40
A[0]	A[1]	A[2]	A[3]	A[4]

Tahapan-3

Mulai dari A[0] sampai A[2], cari nilai maksimal dan tukarkan dengan elemen terakhir (A[2]). Untuk kasus ini, nilai maksimal telah berada pada elemen A[2] sehingga tidak terjadi pertukaran nilai sehingga array masih dalam keadaan semula, yaitu sebagai berikut.

7	4	8	30	40
A[0]	A[1]	A[2]	A[3]	A[4]

Tahapan-4

Terakhir, cari nilai maksimal yang terdapat di dalam A[0] dan A[1] dan tukarkan nilainya dengan A[1]. Dalam kasus ini, karena nilai maksimal ditemukan pada elemen pertama (A[0]), maka di sini masih terjadi pertukaran nilai antara A[0] dan A[1]. Sampai di sini, proses selesai dan array di atas telah terurut secara menaik, yaitu seperti yang terlihat di bawah ini.

4	7	8	30	40
A[0]	A[1]	A[2]	A[3]	A[4]

Apabila proses di atas kita terjemahkan ke dalam kode program, maka sintaknya adalah sebagai berikut.

```
int N, /* Mendeklarasikan variabel untuk menyimpan banyak
          elemen array */
    X, /* Mendeklarasikan variabel untuk menyimpan banyak elemen
          array yang belum terurut */
    j,k, /* Mendeklarasikan variabel untuk indeks pengulangan */
    maks,  /* Mendeklarasikan variabel untuk menyimpan nilai
              maksimal */
    imaks, /* Mendeklarasikan variabel untuk menyimpan indeks
              dari elemen yang menyimpan nilai maksimal */
    temp;  /* Mendeklarasikan variabel sebagai
              variabel temporari */
X = N;
for (j=0; j<N-1; j++) {
  maks = A[0];
  imaks = 0;
  for (k=1; k<X; k++) {
    if (A[k] > maks) {
      maks = A[k];
      imaks = k;
    }
  }
  /* Melakukan pertukaran nilai dengan nilai maks */
  temp = A[X-1];
  A[X-1] = A[imaks];
  A[imaks] = temp;
  /* Melakukan decrement terhadap nilai X */
  X--;
}
```

Berikut ini program lengkap yang akan menunjukkan pengurutan elemen array dengan menggunakan metode pengurutan maksimum.

```
#include <stdio.h>

#define MAX 5

void TampilkanArray(int A[], int n) {
  for (int j=0; j<n; j++) {
    printf("A[%d] = %d\n", j, A[j]);
  }
}

int main() {
  /* Mendeklarasikan array dan melakukan inisialisasi nilai
     ke dalamnya */
  int A[MAX] = {40,4,30,8,7};

  int X,   /* Mendeklarasikan variabel untuk menyimpan banyak
```

```
              elemen array yang belum terurut */
      j, k, /* Mendeklarasikan variabel untuk indeks
               pengulangan */
      maks,     /* Mendeklarasikan variabel untuk menyimpan
                   nilai maksimal */
      imaks,    /* Mendeklarasikan variabel untuk menyimpan
                   indeks dari elemen yang menyimpan
                   nilai maksimal */
      temp;     /* Mendeklarasikan variabel sebagai variabel
                   temporari */

  /* Menampilkan array sebelum diurutkan */
  printf("Sebelum pengurutan:\n");
  TampilkanArray(A, MAX);

  /* Melakukan proses pengurutan dengan metode maksimum */
  X = MAX;
  for (j=0; j<MAX-1; j++) {
    maks = A[0];
    imaks = 0;
    for (k=1; k<X; k++) {
      if (A[k] > maks) {
        maks = A[k];
        imaks = k;
      }
    }
    /* Melakukan pertukaran nilai dengan nilai maks */
    temp = A[X-1];
    A[X-1] = A[imaks];
    A[imaks] = temp;
    /* Melakukan decrement terhadap nilai X */
    X--;
  }
  printf("\n");

  /* Menampilkan array setelah diurutkan */
  printf("Setelah pengurutan:\n");
  TampilkanArray(A, MAX);

  return 0;
}
```

Hasil yang diberikan dari program di atas akan sama persis dengan program yang terdapat pada sub bab sebelumnya, yaitu yang ditulis dengan menggunakan pengurutan gelembung.

Sedangkan untuk metode pengurutan minimum, proses yang dilakukan di dalamnya sebenarnya sama dengan metode maksimum, hanya saja yang dicari untuk ditukarkan adalah nilai minimalnya (bukan nilai maksimalnya).

6.10. Array Multidimensi

Untuk memudahkan pemahaman materi ini, di sini kita akan memecah pembahasan ke dalam dua bagian, yaitu array dua dimensi dan array tiga dimensi.

6.10.1. Array Dua Dimensi

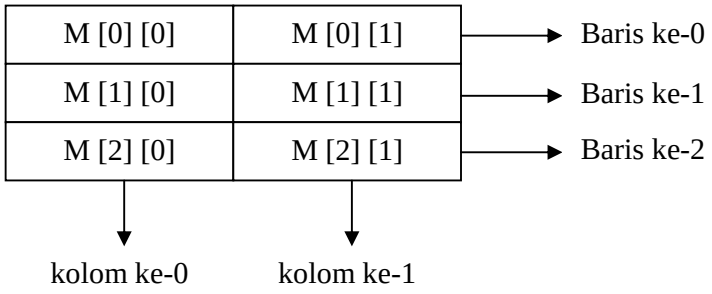
Array dua dimensi adalah array yang memiliki dua buah subskrip, yaitu baris dan kolom. Untuk mendeklarasikan array dua dimensi, kita akan menggunakan bentuk umum berikut.

```
tipe_data nama_array [banyak_baris] [banyak_kolom];
```

Sebagai contoh apabila kita akan membuat deklarasi matrik berordo 3x2, maka kita akan menuliskan sintak untuk deklarasinya sebagai berikut.

```
int M [3] [2];
```

Untuk mempermudah memahaminya, bentuk matrik di atas dapat kita tuliskan dalam bentuk tabel berikut.



Gambar 6.3. Array dua dimensi

Berikut ini contoh program lengkap yang akan menunjukkan penggunaan array dua dimensi. Di sini kita akan melakukan penjumlahan dua buah matrik berordo 3x2. Adapun sintak programnya adalah seperti di bawah ini.

```
#include <stdio.h>

int main(void) {
  /* Membuat tipe data bentukan untuk merepresentasikan matrik
     ordo 3x2 */
  typedef int Matrik32 [3][2];

  /* Mendeklarasikan variabel A,B dan C yang bertipe Matrik32 */
  Matrik32 A, B, C;

  /*Mendeklarasikan variabel j dan k untuk indeks pengulangan */
  int j, k;

  /* Mengisikan nilai ke dalam elemen-elemen matrik A */
```

```
  printf("Mengisikan elemen matrik A:\n");
  for (j=0; j<3; j++) {
    for (k=0; k<2; k++) {
      printf("A[%d][%d] = ", j, k); scanf("%d", &A[j][k]);
    }
  }

  /* Mengisikan nilai ke dalam elemen-elemen matrik B */
  printf("\nMengisikan elemen matrik B:\n");
  for (j=0; j<3; j++) {
    for (k=0; k<2; k++) {
      printf("B[%d][%d] = ", j, k); scanf("%d", &B[j][k]);
    }
  }

  /* Melakukan penjumlahan matrik A dan B dan
     menyimpannya ke dalam matrik C */
  for (j=0; j<3; j++) {
    for (k=0; k<2; k++) {
      C[j][k] = A[j][k] + B[j][k];
    }
  }

  /* Menampilkan hasil penjumlahan matrik di atas */
  printf("\nHasil penjumlahan matrik A dan B:\n");
  for (j=0; j<3; j++) {
    for (k=0; k<2; k++) {
      printf("C[%d][%d] = %d\n", j, k, C[j][k]);
    }
  }
  return 0;
}
```

Hasil yang akan diberkan dari program di atas adalah sebagai berikut.

```
Mengisikan elemen matrik A:
A[0][0] = 10
A[0][1] = 20
A[1][0] = 30
A[1][1] = 40
A[2][0] = 50
A[3][1] = 60

Mengisikan elemen matrik B:
B[0][0] = 10
B[0][1] = 20
B[1][0] = 30
B[1][1] = 40
B[2][0] = 50
B[3][1] = 60
```

```
Hasil penjumlahan matrik A dan B:
B[0][0] = 20
B[0][1] = 40
B[1][0] = 60
B[1][1] = 80
B[2][0] = 100
B[3][1] = 120
```

Apabila dituliskan dalam bentuk matematik, maka hasil dari program di atas dapat dituliskan sebagai berikut.

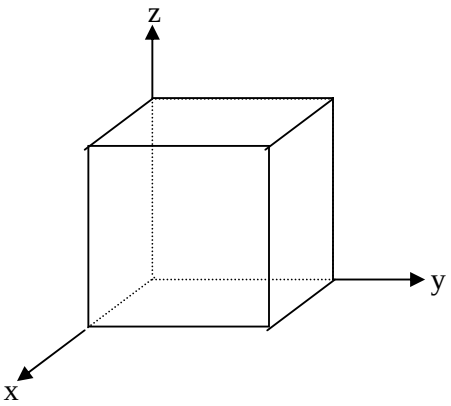
```
| 10   20 |          | 10   20 |          | 10   20  |
| 30   40 |    +     | 30   40 |    =     | 60   80  |
| 50   60 |          | 50   60 |          | 100  120 |
```

Setelah Anda mengetahui konsep dasar array dua dimensi di atas, satu hal lagi yang perlu Anda ketahui adalah cara melakukan inisialisasi nilai terhadap array dua dimensi. Sebagai contoh apabila kita ingin mendeklarasikan matrik `D` yang berordo 3x2 dan kita akan melakukan inisialisasi nilai ke dalamnya, maka sintak yang akan dituliskan adalah sebagai berikut.

```
int D [3] [2] = { 10, 20,  /* Inisialisasi nilai pada baris
                              pertama */
                  30, 40,  /* Inisialisasi nilai pada baris
                              kedua */
                  50, 60 };/* Inisialisasi nilai pada baris
                              ketiga */
```

6.10.2. Array Tiga Dimensi

Berbeda dengan array dua dimensi yang hanya memiliki dua buah subskrip, pada array tiga dimensi subskrip yang dimiliki ada tiga. Untuk lebih mempermudah pembahasan, bayangkanlah sebuah kubus atau balok dimana bangun tersebut selain memiliki panjang dan lebar, juga memiliki tinggi. Secara matematis, hal tersebut dapat kita representasikan ke dalam tiga buah sumbu, yaitu sumbu `x`, `y` dan `z`. Berikut ini ilustrasi dari suatu bangun dalam tiga dimensi.



Gambar 6.4. Contoh bentuk array 3 dimensi

Adapun bentuk umum dari pendeklarasian array tiga dimensi adalah seperti yang tampak di bawah ini.

```
tipe_data nama_array [banyaknya_elemen_pada_sumbu_x]
                     [banyaknya_elemen_pada_sumbu_y]
                     [banyaknya_elemen_pada_sumbu_z];
```

Perhatikan contoh pendeklarasian array di bawah ini.

```
int Arr3D [3] [4] [6];
```

Ini artinya array `Arr3D` diatas memiliki 3 buah elemen bertipe int yang masing-masing elemennya terdiri dari 4 buah elemen lain dimana keempat buah elemen tersebut juga masing-masing terdiri dari 6 buah elemen.

Berikut ini contoh program sederhana yang akan menunjukkan bagaimana cara melakukan inisialisasi dan pengaksesan nilai dari elemen-elemen dalam array tiga dimensi.

```
#include <stdio.h>

int main(void) {
  /* Mendeklarasikan array tiga dimensi dengan nama Arr3D dan
     melakukan inisialisasi nilai ke dalam setiap elemennya */
  int Arr3D [2][2][2] = {1,2,3,4, 5,6,7,8 };
  /* Mendeklarasikan variabel sebagai indeks pengulangan */
  int i, j, k;

  /* Mengakses nilai yang terdapat pada setiap elemen array
     Arr3D */
  for (i=0; i<2; i++) {
    for (j=0; j<2; j++) {
      for (k=0; k<2; k++) {
```

```
        printf("Arr3D[%d][%d][%d] = %d\t", i, j, k,
               Arr3D[i][j][k]);
      }
    }
    printf("\n\n");
  }

  return 0;
}
```

Hasil yang akan diperoleh dari program di atas adalah sebagai berikut.

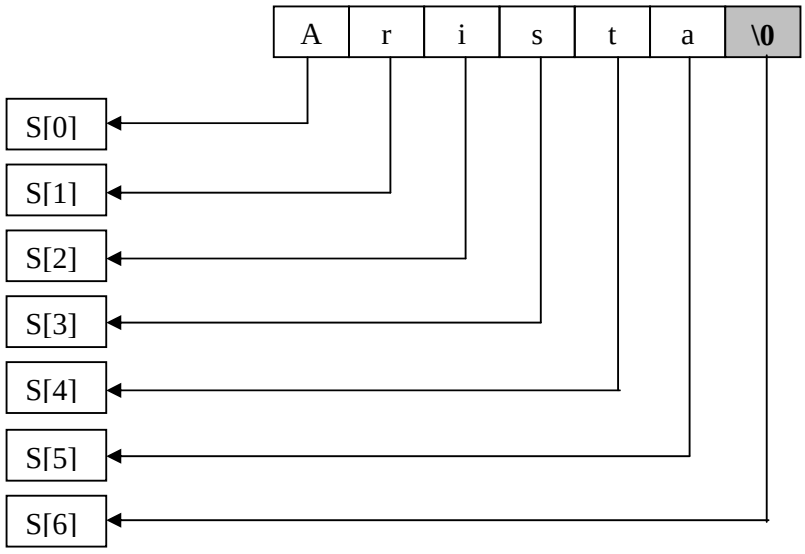
```
Arr3D[0][0][0] = 1    Arr3D[0][0][1] = 2    Arr3D[0][1][0] = 3    Arr3D[0][1][1]= 4

Arr3D[1][0][0] = 5    Arr3D[1][0][1] = 6    Arr3D[1][1][0] = 7    Arr3D[1][1][1]= 8
```

6.11. String (Array dari Karakter)

Dalam permrograman, string merupakan suatu kumpulan karakter yang terangkai secara bersamaan. Sedangkan dalam bahasa C, string merupakan suatu pointer ke tipe `char` sehingga tipe string ini direpresentasikan dengan tipe `char*` atau dengan menggunakan array dari tipe `char`. Hal ini disebabkan karena variabel yang bertipe `char` hanya dapat menampung satu buah karakter saja. Adapun dalam buku ini, pointer baru akan kita bahas pada bab selanjutnya setelah bab ini. Dalam bahasa C, string akan diapit oleh tanda petik ganda (“”). Jadi, perlu diperhatikan bahwa ‘B’ akan berbeda arti dengan “B”. ‘B’ oleh bahasa C akan dianggap sebagai karakter, namun “B” akan dianggap sebagai suatu string. Berikut ini beberapa cara pendeklarasian suatu variabel bertipe string di dalam bahasa C.

```
/* Melakukan inisialisasi string */
char string[] = "Ini adalah string konstan";
/* Membatasi string dengan 10 karakter */
char string[10];
/* Mendeklarasikan pointer ke tipe char */
char *string;
```

Perlu diingat bahwa dalam bahasa C, suatu string selalu diakhiri dengan karakter *null* (‘`\0`’, bukan berarti nol). Dalam kode ASCII, karakter ‘`\0`’ memiliki nilai 0. Untuk itu, setiap pendeklarasian string kita juga harus mengalokasikan ruang untuk menempatkan karakter *null* tersebut. Sebagai contoh apabila kita akan mendeklarasikan suatu variabel bertipe string yang akan menyimpan teks `"Arista"`, maka kita membutuhkan suatu array dengan 7 buah elemen. Hal ini artinya kita akan menempatkan 7 buah karakter, yaitu 6 karakter untuk teks `"Arista"` dan 1 karakter untuk karakter *null*. Berikut ini gambar yang akan merepresentasikan hal tersebut.



Gambar 6.5. String di dalam bahasa C

Pada gambar di atas, elemen `S[6]` berisi karakter *null* yang menunjukkan bahwa string tersebut telah berakhir.

6.11.1. Menampilkan String ke Layar

Untuk menampilkan suatu string ke suatu alat keluaran (misalnya monitor), kita dapat menggunakan fungsi `printf()` yang diikuti dengan format `%s` atau dengan menggunakan fungsi `puts()` maupun `cputs()`. Berikut ini contoh program yang akan menunjukkan hal tersebut.

```
#include <stdio.h>

int main(void) {
  /* Mendeklarasikan string dan melakukan inisialisasi ke
     dalamnya */
  char *S1 = “Menampilkan string dengan fungsi printf()”;
  char *S2 = “Menampilkan string dengan fungsi puts()”;
  char *S3 = “Menampilkan string dengan fungsi cputs()”;

  /* Menampilkan string dengan fungsi printf(), puts() dan
     cputs() */
  printf(“%s\n”, S1);
  puts(S2);
  cputs(S3);

  return 0;
}
```

Hasil yang akan diperoleh dari program di atas adalah sebagai berikut.

```
Menampilkan string dengan fungsi printf()
Menampilkan string dengan fungsi puts()
Menampilkan string dengan fungsi cputs()
```

6.11.2. Membaca String dari Keyboard

Pada umumnya para programmer pemula biasanya akan mendapatkan masalah ketika ia menginginkan untuk melakukan pembacaan (*input*) string yang mengandung spasi. Pasalnya, apabila kita menggunakan fungsi `scanf()` untuk melakukan hal tersebut, maka karakter spasi (‘ ‘) yang ditemukan pertama kali akan dianggap sebagai karakter *null*. Hal ini tentu akan menyebabkan hasil yang berbeda dengan apa yang diinginkan. Berikut ini contoh program yang menggunakan fungsi `scanf()` dalam membaca suatu string

```
#include  <stdio.h>

int main(void) {
  /* Mendeklarasikan variabel S yang bertipe string */
  char *S;

  /* Melakukan input string dengan fungsi scanf() */
  printf(“Masukkan nama lengkap Anda : ”); scanf(“%s”, &S);

  /* Menampilkan string yang telah disimpan ke dalam
     variabel S */
  printf(“Hai %s, apa kabarmu?”, S);

  return 0;
}
```

Contoh hasil yang akan diberikan adalah sebagai berikut.

```
Masukkan nama lengkap Anda : Camellia Panatarani
Hai Camellia, apa kabarmu?
```

Apa yang dapat Anda lihat? Di atas kita memasukkan string `“Camellia Panatarani”`, namun ketika ditampilkan ternyata string yang disimpan ke dalam variabel `S` hanya berupa string `“Camellia”`. Kita tentu tidak menginginkan hal tersebut terjadi bukan? Untuk itu, di sini Anda akan dikenalkan dengan fungsi `gets()` yang dapat melakukan hal di atas dengan benar. Perhatikan penggunaannya dalam program di bawah ini.

```
#include <stdio.h>

int main(void) {
  /* Mendeklarasikan variabel S yang bertipe string */
  char *S;

  /* Melakukan input string dengan fungsi gets() */
  printf("Masukkan nama lengkap Anda : "); gets(S);

  /* Menampilkan string yang telah disimpan ke dalam
     variabel S */
  printf("Hai %s, apa kabarmu?", S);

  return 0;
}
```

Program di atas akan memberikan hasil sebagai berikut.

```
Masukkan nama lengkap Anda : Camellia Panatarani
Hai Camellia Panatarani, apa kabarmu?
```

Berbeda dengan program sebelumnya, kali ini string yang ditampilkan sebagai hasil adalah sesuai dengan string yang dimasukkan.

6.11.3. Manipulasi String

Hal yang sering sekali ditemui dalam mengembangkan sebuah program adalah proses manipulasi string, misalnya proses pencarian, penyalinan, pembandingan yang dilakukan terhadap string dan banyak lagi proses lainnya. Untuk itu, pada bagian ini kita akan membahas secara detil mengenai fungsi-fungsi yang terdapat pada file header `<string.h>`. Sedangkan untuk mempermudah pembahasan, maka fungsi-fungsi tersebut akan diterangkan berdasarkan kategori proses tertentu.

6.11.3.1. Menggabungkan String

Bahasa C telah menyediakan dua buah fungsi yang berguna untuk melakukan penggabungan string, yaitu fungsi `strcat()` dan `strncat()`. Berikut ini penjelasan masing-masing fungsi tersebut.

a. Fungsi `strcat()`

Prototipe dari fungsi ini adalah sebagai berikut.

```
char *strcat(char *str1, char* str2);
```

Fungsi `strcat()` akan menambahkan salinan dari string `str2` ke bagian akhir string `str1`. Di sini karakter null yang terdapat pada string `str1` akan dipindahkan ke bagian akhir dari string baru hasil penggabungan. Berikut ini contoh penggunaan dari fungsi tersebut.

```
#include <stdio.h>
#include <string.h>

int main(void) {
  char s1[50] = “Pemrograman ”;
  char s2[21] = “Menggunakan Bahasa C”;
  strcat(s1, s2);
  printf(“%s”, s1);
  return 0;
}
```

Hasil yang akan didapatkan dari program di atas adalah sebagai berikut.

```
Pemrograman Menggunakan Bahasa C
```

b. Fungsi `strncat()`

Prototipe dari fungsi ini adalah sebagai berikut.

```
char *strncat(char *str1, char* str2, size_t n);
```

Fungsi ini juga berguna untuk menambahkan string dari string `str2` ke dalam `str1`. Namun di sini kita diizinkan untuk menentukan berapa banyak karakter (n) dari `str2` yang akan digabungkan ke `str1`. Berikut ini program yang akan menunjukkan hal tersebut.

```
#include <stdio.h>
#include <string.h>

int main(void) {
  char s1[50] = “Pemrograman ”;
  char *s2[21] = “Menggunakan Bahasa C”;
  strncat(s1, s2, 11);  /* Menambahkan 11 karakter dari str2 ke
                           str1 */
  printf(“%s”, s1);
  return 0;
}
```

Hasil yang akan didapatkan adalah sebagai berikut.

```
Pemrograman Menggunakan
```

6.11.3.2. Menentukan Panjang String

Bahasa C telah menyediakan fungsi `strlen()` yang berguna untuk mengembalikan panjang dari sebuah string (tidak termasuk karakter *null*). Adapun prototipe dari fungsi tersebut adalah sebagai berikut.

```
size_t strlen(char *str);
```

Mungkin Anda bingung dengan tipe `size_t` yang merupakan tipe kembalian dari fungsi di atas. `size_t` ini merupakan tipe yang didefinisikan di dalam file header `<string.h>` untuk menampung nilai-nilai bilangan bulat (`unsigned integer`). Berikut ini contoh program yang akan menunjukkan penggunaan fungsi `strlen()`.

```
#include <stdio.h>
#include <string.h>

int main(void) {
  char *str1 = “Panjang string”;
  char str2[9] = “Bahasa C”;

  /* Menampilkan panjang string dari str1 dan str2 */
  printf(“Panjang str1 : %d\n”, strlen(str1));
  printf(“Panjang str2 : %d”, strlen(str2));

  return 0;
}
```

Hasil yang akan diberikan dari program tersebut adalah sebagai berikut.

```
Panjang str1 : 14
Panjang str2 : 8
```

6.11.3.3. Menyalin String

Untuk melakukan penyalinan string, bahasa C telah menyediakan tiga buah fungsi, yaitu `strcpy()`, `strncpy()` dan `strdup()`. Berikut ini penjelasan dari masing-masing fungsi tersebut.

a. Fungsi `strcpy()`

Prototipe dari fungsi ini adalah sebagai berikut.

```
char *strcpy(char *str1, char *str2);
```

Fungsi ini akan melakukan penyalinan (*copy*) dari string `str2` ke `str1` (termasuk juga karakter *null*). Namun hal yang perlu diperhatikan di sini adalah keduanya merupakan pointer sehingga pada saat proses penyalinan string, kita harus melakukan pengalokasian memori terlebih dahulu untuk pointer `str1`. Apabila pengalokasian memori tersebut tidak dilakukan maka akan menimbulkan masalah dengan memori, yaitu berupa penimpaan (*overwrite*) sebanyak `strlen(str2)` byte. Berikut ini contoh program yang akan mengilustrasikan penggunaan fungsi `strcpy()`.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>   /* untuk menggunakan fungsi malloc() */
int main(void) {
  char str_sumber[] = “Ini adalah string yang akan disalin”;

  /* Mendeklarasikan variabel untuk menampung string hasil
     penyalinan */
  char str_tujuan1[100];   /* mengalokasikan 100 byte untuk 100
                              karakter */
  char *str_tujuan2;       /* pointer ke char namun belum
                              dialokasikan */
  char *str_tujuan3;       /* pointer ke char namun belum
                              dialokasikan */

  /* Melakukan penyalinan dari str_sumber ke variabel
     str_tujuan1 */
  strcpy(str_tujuan1, str_sumber);

  /* Melakukan penyalinan dari str_sumber ke variabel
     str_tujuan2 */
  /* Di sini kita harus melakukan alokasi memori terlebih
     dahulu */
  str_tujuan2 =
    (char *) malloc(sizeof(char) * (strlen(str_sumber) + 1));
  strcpy(str_tujuan2, str_sumber);

  /* Menampilkan string hasil penyalinan */
  printf(“str_tujuan1 : %s\n”, str_tujuan1);
  printf(“str_tujuan2 : %s\n”, str_tujuan2);

  /* Berikut ini contoh penggunaan strcpy yang SALAH */
  /* Melakukan penyalinan dari str_sumber ke str_tujuan3 tanpa
     alokasi str_tujuan3 */

  /* strcpy(str_tujuan3, str_sumber); */         /* SALAH */
```

```
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
str_tujuan1 : Ini adalah string yang akan disalin
str_tujuan2 : Ini adalah string yang akan disalin
```

b. Fungsi `strncpy()`

Prototipe dari fungsi ini adalah sebagai berikut.

```
char *strncpy(char *str1, char *str2, size_t n);
```

Sebenarnya fungsi ini hampir mirip dengan fungsi `strcpy()`, namun di sini kita diizinkan untuk menentukan berapa jumlah karakter (`n`) dari `str2` yang akan disalin ke `str1`. Adapun contoh penggunaannya dalam program adalah seperti di bawah ini.

```
#include <stdio.h>
#include <string.h>

int main(void) {
  char str1[12];
  char str2[] = “Pemrograman Menggunakan Bahasa C”;
  strncpy(str1, str2, 11);
  printf(“Hasil penyalinan : %s”, str1);
  return 0;
}
```

Program di atas akan memberikan hasil seperti di bawah ini.

```
Hasil penyalinan : Pemrograman
```

c. Fungsi `strdup()`

Satu lagi fungsi yang berguna untuk melakukan penyalinan string adalah `strdup()`. Fungsi ini sebenarnya sama dengan fungsi `strcpy()`, hanya di sini pengalokasian memori dari string tujuan akan dilakukan secara otomatis. Adapun prototipe dari fungsi ini adalah sebagai berikut.

```
char *strdup(char *str);
```

Fungsi ini akan mengembalikan pointer ke string dimana ruangnya telah dipesan secara otomatis degan menggunakan fungsi `malloc()`. Apabila terdapat kegagalan dalam pemesanan memori maka yang akan dikembalikan adalah pointer `NULL`. Untuk lebih memahaminya, perhatikan contoh program di bawah ini yang akan mengilustrasikan perbedaan fungsi `strdup()` dengan `strcpy()`.

```
#include <stdio.h>
#include <string.h>

int main(void) {
  char str_sumber[] = "Pemrograman Menggunakan Bahasa C";
  char *str_tujuan;        /* pointer ke char yang belum
                              dialokasi */
  if ((str_tujuan = strdup(str_sumber)) != NULL) {
    printf("%s", str_tujuan);
  }
  return 0;
}
```

Perhatikan sintak di atas, di sana tampak pointer `str_tujuan` tidak perlu dialokasikan terlebih dahulu untuk melakukan proses penyalinan string. Hal ini sangat berbeda dengan fungsi `strcpy()` yang menuntut kita mengalokasikan ruang memori terlebih dahulu untuk menyimpan string hasil penyalinan. Adapun hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Pemrograman Menggunakan Bahasa C
```

6.11.3.4. Membandingkan String

Dalam bahasa C standar, terdapat dua buah fungsi yang dapat melakukan pembandingan terhadap dua buah string, yaitu fungsi `strcmp()` dan `strncmp()`. Namun kebanyakan kompilator C juga telah menyediakan fungsi-fungsi lainnya yang spesifik untuk keperluan ini Berikut ini penjelasan dari masing-masing fungsi tersebut.

a. Fungsi `strcmp()`

Prototipe dari fungsi ini adalah sebagai berikut.

```
int strcmp(char *str1, char *str2);
```

Fungsi ini akan mengembalikan nilai bilangan bulat (*integer*) sebagai hasil perbandingan dua buah string, yaitu `str1` dan `str2`. Adapun nilai yang akan dihasilkan dari perbandingan dua string di atas adalah seperti yang terdapat pada tabel di bawah ini.

Nilai	Arti
< 0 (negatif)	`str1` lebih kecil dari `str2`
0	`str1` sama dengan `str2`
> 0 (positif)	`str1` lebih besar dari `str2`

Untuk lebih memahaminya, perhatikan contoh program di bawah ini yang akan menunjukkan penggunaan fungsi `strcmp()`.

```
#include <stdio.h>
#include <string.h>

int main(void) {
  char *s1 = “Bahasa C”;
  char *s2 = “Bahasa C++”;
  char *s3 = “Bahasa C”;

  /* Melakukan perbandingan terhadap dua string dan menampilkan
     nilainya */
  printf(“Nilai yang dikembalikan : %d\n”, strcmp(s1, s2));
  printf(“Nilai yang dikembalikan : %d\n”, strcmp(s1, s3));
  printf(“Nilai yang dikembalikan : %d\n”, strcmp(s2, s1));

  return 0;
}
```

Program di atas akan memberikan hasil sebagai berikut.

```
Nilai yang dikembalikan : -1
Nilai yang dikembalikan : 0
Nilai yang dikembalikan : 1
```

b. Fungsi `strncmp()`

Prototipe dari fungsi ini adalah sebagaia berikut.

```
int strncmp(char *str1, char *str2, size_t n);
```

Fungsi ini akan membandingkan n buah karakter string `str2` dengan string `str1`. Adapun nilai yang dihasilkan dari fungsi ini sama seperti pada saat kita menggunakan

fungsi `strcmp()`. Untuk lebih memahaminya, berikut ini contoh yang akan menunjukkan penggunaan fungsi `strncmp()`.

```
#include <stdio.h>
#include <string.h>

int main(void) {
  char str1[] = “String yang pertama”;
  char str2[] = “String yang kedua”;

  /* Membandingkan str1 dan beberapa karakter dari str2 */

  /* semua karakter */
  printf(“Nilai kembalian = %d\n”,
         strncmp(str1,str2,strlen(str2)));
  /* 6 buah karakter */
  printf(“Nilai kembalian = %d”, strncmp(str1, str2, 6));

  return 0;
}
```

Hasil yang akan diperoleh dari program di atas adalah sebagai berikut.

```
Nilai Kembalian = 1
Nilai Kembalian = 0
```

c. Fungsi Pembanding String Lainnya

Beberapa kompilator C yang beredar saat ini telah menyediakan beberapa fungsi yang bergna untuk melakukan komparasi atau perbandingan terhadap dua buah string yang tidak bersifat *case-sensitive*. Artinya perbandingan tersebut akan mengabaikan perbedaan penulisan huruf yang terdapat pada string tersebut. Sebagai contoh, string `“Bahasa C”` akan sama dengan string `“bahasa c”`, `“BAHASA C”`, `”bAHaSa c”` dan yang lainnya. Untuk mengetahui fungsi-fungsi tersebut tentunya Anda harus membaca manual dari masing-masing kompilator C atau C++ bersangkutan yang Anda gunakan. Namun sebagai referensi bagi Anda, Borland telah mempunyai dua buah fungsi untuk melakukan proses ini, yaitu `strcmpi()` dan `stricmp()`. Sedangkan Microsoft menggunakan fungsi `_stricmp()` dan Symantec menggunakan fungsi `strcmpl()`.

6.11.3.5. Melakukan Pencarian String

Hal yang paling sering dilakukan dalam proses manipulasi string adalah suatu pencarian, dimana kita melakukan pencarian terhadap suatu string ataupun karakter tertentu apakah terdapat dalam string lain atau tidak. Untuk melakukan hal ini, bahasa C telah menyediakan enam buah fungsi khusus yang juga terdapat dalam file header `<string.h>`, yaitu fungsi `strchr()`, `strrchr()`, `strspn()`, `strcspn()`, `strpbrk()` dan `strstr()`. Berikut ini penjelasan dari masing-masing fungsi tersebut.

a. Fungsi `strchr()`

Prototipe dari fungsi ini adalah sebagai berikut.

```
char *strchr(char *str, int ch);
```

Fungsi ini akan melakukan pencarian karakter `ch` terhadap string `str` mulai dari kiri sampai ke kanan. Pointer ke karakter akan langsung dikembalikan sebagai hasil apabila karakter tersebut ditemukan untuk pertama kalinya. Sedangkan apabila tidak ditemukan maka fungsi ini akan mengembalikan nilai `NULL`. Sebagai contoh, apabila kita akan melakukan pencarian karakter `'S'` dari string `"ARISTA"`. Di sini fungsi `strchr()` akan menemukan karakter tersebut pada indeks ke-3 (ingat indeks dalam bahasa C dimulai dari 0), sehingga nilai yang akan dikembalikan oleh fungsi ini adalah string `"STA"`. Berikut ini contoh program yang akan membuktikan hal tersebut.

```
#include <stdio.h>
#include <string.h>

int main(void) {
  char str[100] = "ARISTA";
  char karakter = 'S';
  char *hasil;
  hasil = strchr(str, karakter);
  printf("Nilai kembalian : %s\n", hasil);
  printf("Karakter %c ditemukan pada indeks ke-%d",
          karakter, (hasil – str));
  return 0;
}
```

Apabila dijalankan, maka program di atas akan memberikan hasil seperti di bawah ini.

```
Nilai kembalian : STA
Karakter S ditemukan pada indeks ke-3
```

b. Fungsi `strrchr()`

Prototipe dan cara kerja dari fungsi ini sama dengan fungsi `strchr()`, hanya di sini yang akan ditemukan adalah karakter bersangkutan yang berada pada bagian akhir dari string. Sebagai contoh apabila kita akan melakuka pencarian karakter `'a'` terhadap string `"Pemrograman"`. Apabila kita menggunakan fungsi `strchr()`, kita akan mendapatkan hasil `"aman"`. Sedangkan hasil yang akan diperoleh apabila kita

menggunakan fungsi `strrchr()` adalah string `"an"`. Hal ini disebabkan karena karakter `'a'` yang ditemukan oleh fungsi `strrchr()` adalah karakter `'a'` yang terdapat pada bagian akhir dari string tersebut. Untuk membuktikan hal ini, coba Anda lakukan modifikasi sendiri terhadap program sebelumnya.

c. Fungsi `strspn()`

Prototipe dari fungsi ini adalah sebagai berikut.

```
size_t strspn(char *str1, char *str2);
```

Fungsi ini akan mengembalikan jumlah karakter yang sama dari string `str1` dan `str2`. Artinya, di sini akan terjadi proses pencocokan setiap karakter yang terdapat pada `str2` dengan karakter yang terdapat pada `str1`. Proses pencocokan akan dimulai dari awal karakter pertama dari `str1`. Apabila karakter pertama `str1` tidak sama dengan karakter pertama `str2`, maka fungsi akan mengembalikan nilai 0 dan proses pencocokan akan dihentikan. Namun apabila sama, maka fungsi ini akan melakukan pencocokan untuk karakter berikutnya. Untuk dapat lebih memahaminya, perhatikan contoh program berikut ini.

```
#include <stdio.h>
#include <string.h>

int main(void) {
  char str[100] = "ARISTA";
  printf("Lebar karakter yang ditemukan : %d\n",
         strspn(str,"ARIS"));
  printf("Lebar karakter yang ditemukan : %d\n",
         strspn(str,"ARTIS"));
  printf("Lebar karakter yang ditemukan : %d\n",
         strspn(str,"STRING"));
  return 0;
}
```

Hasil yang akan diberikan oleh program di atas adalah sebagai berikut.

```
Lebar karakter yang ditemukan : 4
Lebar karakter yang ditemukan : 2
Lebar karakter yang ditemukan : 0
```

d. Fungsi `strcspn()`

Prototipe dari fungsi ini adalah sebagai berikut.

```
size_t strcspn(char *str1, char *str2);
```

Fungsi ini akan mengembalikan jumlah karakter yang terdapat di bagian awal dari string `str1`, dimana karakter-karakter tersebut **tidak** sama dengan karakter-karakter yang terdapat pada `str2`. Berikut ini contoh program yang akan menunjukkan penggunaan fungsi `strcspn()`.

```
#include <stdio.h>
#include <string.h>

int main(void) {
  printf("Lebar karakter yang ditemukan : %d\n",
         strcspn("xxARISTA","ARIS"));
  printf("Lebar karakter yang ditemukan : %d\n",
         strcspn("xxxARISTA","ARIS"));
  printf("Lebar karakter yang ditemukan : %d\n",
         strcspn("ARISTA","XX"));
  return 0;
}
```

Hasil yang akan diberikan oleh program di atas adalah seperti berikut.

```
Lebar karakter yang ditemukan : 2
Lebar karakter yang ditemukan : 3
Lebar karakter yang ditemukan : 6
```

e. Fungsi `strpbrk()`

Prototipe dari fungsi ini adalah sebagai berikut.

```
char *strpbrk(char *str1, char *str2);
```

Fungsi ini akan mengembalikan string dari `str1` yang merupakan susunan karakter-karkter yang terdapat pada `str2`. Namun apabila karakter-karakter yang terdapat pada `str2` tidak ditemukan di dalam `str1`, maka fungsi akan mengembalikan nilai `NULL`. Sebagai contoh apabila kita akan melakukan pencarian string `“ST”` pada string `“ARISTA”`, maka fungsi akan mengembalikan string `“STA”`. Begitupun apabila kita mencari string `“TS”`, yang dihasilkan pun tetap `“STA”`. Untuk lebih memahaminya, perhatikan contoh program di bawah ini yang akan menunjukkan penggunaan fungsi `strpbrk()`.

```
#include <stdio.h>
#include <string.h>

int main(void) {
  printf("Hasil = %s\n", strpbrk("xxARISTA","ARIS"));
  printf("Hasil = %s\n", strpbrk("xxxARISTA","ST"));
  printf("Hasil = %s\n", strpbrk("xxxARISTA","TS"));
  printf("Hasil = %s\n", strpbrk("xxxARISTA","IS"));
  printf("Hasil = %s\n", strpbrk("ARISTA","XX"));
  return 0;
}
```

Adapun hasil yang akan diberikan dari program di atas adalah seperti yang tampak di bawah ini.

```
Hasil = ARISTA
Hasil = STA
Hasil = STA
Hasil = ISTA
Hasil = (null)
```

f. Fungsi `strstr()`

Fungsi terakhir yang digunakan untuk mencari string adalah fungsi `strstr()`, yaitu fungsi yang akan mengembalikan string yang cocok. Di sini rangkaian karakter yang terdapat pada `str2` harus sama persis dengan rangkaian karakter yang terdapat pada `str1`. Ini tentu berbeda dengan fungsi pencarian yang lain yang dapat melibatkan kehadiran karakter saja (tanpa susunan atau rangkaian yang jelas). Sebagai contoh apabila terdapat string `"Pemrograman Menggunakan Bahasa C"`, maka apabila kita mencari string `"nam"`, maka hasilnya adalah `NULL` walaupun karakter `'n'`, `'a'` dan `'m'` masing-masing terdapat dalam string tersebut. Namun apabila kita melakukan pencarian string `"man"`, maka fungsi akan mengembalikan nilai `"man Menggunakan Bahasa C"`. Berikut ini contoh program yang akan membuktikan hal tersebut.

```
#include <stdio.h>
#include <string.h>

int main(void) {
  char str[100] = "Pemrograman Menggunakan Bahasa C";

  printf("Hasil = %s\n", strstr(str,"nam"));
  printf("Hasil = %s\n", strstr(str,"man"));
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Hasil = (null)
Hasil = man Menggunakan Bahasa C
```

6.11.3.6. Melakukan Konversi String

Meskipun bukan termasuk ke dalam fungsi-fungsi standar ANSI, namun beberapa kompilator C yang beredar saat ini kebanyakan telah menyediakan fungsi untuk melakukan konversi string, yaitu fungsi `strlwr()` dan `strupr()`.

a. Fungsi `strlwr()`

Prototipe dari fungsi ini adalah sebagai berikut.

```
char *strlwr(char *str);
```

Fungsi ini akan menyebabkan string `str` berubah menjadi huruf kecil. Sebagai contoh apabila kita menggunakan `strlwr()` untuk string `“Pemrograman Menggunakan Bahasa C”`, maka nilai yang akan dikembalikan adalah string `“pemrograman menggunakan bahasa c”`. Untuk membuktikan hal tersebut, coba Anda perhatikan contoh program di bawah ini.

```
#include <stdio.h>
#include <string.h>

#define MAX 100

int main(void) {
  char *str;
  str = malloc (char *) (MAX * sizeof(char));
  str = strlwr(“PEMROGRAMAN MENGGUNAKAN BAHASA C”);
  printf(“%s”, str);
  free(str);
  return 0;
}
```

Hasil yang akan diberikan oleh program di atas adalah sebagai berikut.

```
pemrograman menggunakan bahasa c
```

b. Fungsi `strupr()`

Fungsi ini merupakan kebalikan dari fungsi `strlwr()`, dimana fungsinya adalah untuk mengubah suatu string ke dalam bentuk huruf besar (kapital). Adapun prototipe dari fungsi `strupr()` ini adalah sebagai berikut.

```
char *strupr(char *str);
```

Perhatikan contoh penggunaan fungsi tersebut di dalam progam di bawah ini.

```
#include <stdio.h>
#include <string.h>

#define MAX 100

int main(void) {
  char *str;
  str = malloc (char *) (MAX * sizeof(char));
  str = strupr(“pemrograman menggunakan bahasa c”);
  printf(“%s”, str);
  free(str);
  return 0;
}
```

Apabila dijalankan, program tersebut akan memberikan hasil sebagai berikut.

```
PEMROGRAMAN MENGGUNAKAN BAHASA C
```

Bab

7

7.1. Pendahuluan

Bahasa C adalah bahasa pemrograman yang syarat dengan pointer, artinya kehadiran pointer di dalam program yang ditulis dengan bahasa C merupakan hal yang sangat mutlak. Apabila Anda berfikir bahwa pointer itu susah untuk dipelajari serta menganggap materi ini dapat Anda lewati dan tinggalkan, maka berhentilah berfikir untuk menjadi seorang programmer C. Memang harus diakui bahwa pointer merupakan materi yang cukup membingungkan untuk dipahami, apalagi bagi para programmer pemula. Maka dari itu pada bab ini kita akan membahas secara detail mengenai bagaimana cara mendeklarasikan dan mengimplementasikan pointer di dalam kode program yang ditulis dalam bahasa C.

Pointer merupakan fitur andalan yang dimiliki oleh bahasa C, namun pointer juga dapat dikatakan sebagai fitur yang berbahaya dan menjadi malapetaka apabila digunakan oleh para programmer yang belum benar-benar memiliki konsep yang kuat dan berpengalaman dalam kasus-kasus tertentu yang berhubungan dengan pointer. Kesalahan dalam penggunaan pointer di dalam program yang kita tulis akan menyebabkan terdapatnya *bug* (kesalahan atau lebih tepatnya kecacatan program) yang akan sulit untuk dilacak atau ditemukan di mana letak kesalahannya.

Pointer dikatakan sebagai suatu kelebihan bahasa C karena dengan menggunakan pointer, kita dapat mengalokasikan alamat memori secara dinamis, artinya kita dapat mengatur keberadaan suatu nilai dalam memori komputer sesuai dengan paruh waktu yang kita inginkan. Dengan demikian ruang memori dapat kita atur sesuai kebutuhan program sehingga tidak akan terdapat adanya pemborosan memori. Hal ini tentu akan meningkatkan performa dari program yang kita buat.

7.2. Apa Itu Memori Komputer?

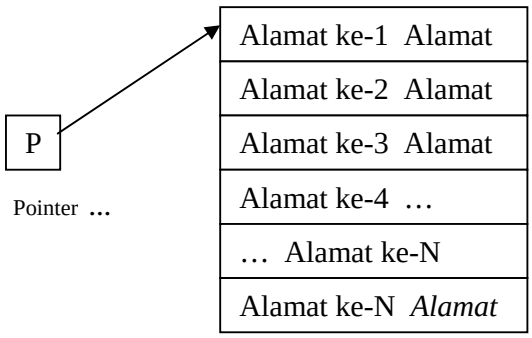
Setiap RAM (*Random Access Memory*) yang dimiliki oleh PC (*Personal Computer*) terdiri dari ribuan runtunan blok-blok lokasi dimana setiap blok tersebut diidentifikasi dengan alamat yang bersifat unik. Adapun rentang dari alamat-alamat memori tersebut adalah dari 0 sampai nilai maksimum (tergantung dari besarnya memori yang terpasang di komputer kita).

Pada saat kita menggunakan komputer, sistem operasi akan mengambil sejumlah memori untuk kebutuhan sistem. Begitu juga ketika kita menjalankan suatu program aplikasi, kode-kode (instruksi mesin untuk melakukan proses tertentu) dan data-data yang terdapat di dalam program tersebut, keduanya akan mengambil sejumlah memori.

Ketika kita mendeklarasikan suatu variabel di dalam bahasa C, maka kompiler akan memesan ruang memori (dengan alamat tertentu yang bersifat unik) untuk menampung variabel tersebut. Dengan kata lain, kompiler akan mengasosiasikan alamat yang dimaksud melalui nama variabel yang telah dideklarasikan. Hal ini berarti bahwa apabila kita menggunakan variabel tersebut, maka sebenarnya secara tidak langsung kita juga mengakses alamat memori yang bersangkutan. Kita tidak perlu mempermasalahkan bagaimana cara kompiler mengakses alamat tersebut, karena semua kerumitan dari proses tersebut telah disembunyikan.

7.3. Apa Itu Pointer?

Menurut definisinya, pointer adalah sebuah variabel yang berisikan alamat memori (bukan nilai) atau dengan kata lain dapat dikatakan bahwa pointer adalah suatu variabel penunjuk ke alamat memori tertentu. Secara umum, pointer dapat digambarkan sebagai berikut.



Gambar 7.1. Ilustrasi Pointer

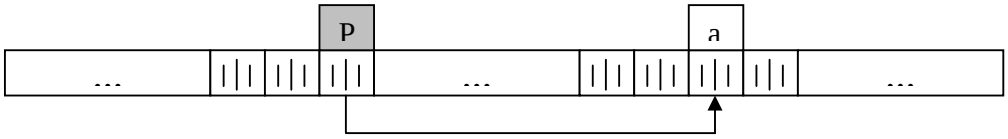
Dari gambar di atas dapat kita lihat bahwa pointer `P` sedang menunjuk ke alamat ke-1, ini berarti bahwa nilai dari pointer `P` adalah *`alamat ke-1`*. Hal ini tentu berbeda dengan definisi variabel biasa yang hanya dapat menyimpan nilai dengan tipe data tertentu. Perhatikan contoh pendeklarasian variabel berikut ini.

```
int X;
X = 10;
```

Pada pendeklarasian di atas nilai `X` bukan berupa alamat, melainkan berupa nilai yang memiliki tipe data `int` (yaitu nilai 10).

Untuk dapat lebih memahami apa sebenarnya pengertian pointer, marilah kita melihat kembali pengertian memori dan alamat memori komputer. Memori komputer dapat dikatakan sebagai kumpulan atau larik dari alamat memori yang masing-masing bersifat unik. Hal yang perlu sekali untuk diperhatikan adalah bahwa pointer juga merupakan isi memori. Pointer merupakan ‘alamat khusus’ yang telah dipesan oleh kompiler dan *linker* untuk mencatat atau menunjuk alamat memori dari variabel lain. Oleh sebab itu, pointer juga sering dinamakan dengan alamat memori, walaupun sebenarnya lebih tepat

bila disebut dengan pencatat atau penunjuk alamat memori. Apabila kita memiliki pointer `P` yang akan menunjuk ke alamat dari variabel `a` yang bertipe `char` (berukuran 1 byte), maka situasi ini dapat kita representasikan melalui gambar berikut.



Gambar 7.2. Pointer ke tipe karakter

Adapun cara mendeklarasikan pointer di dalam bahasa C adalah dengan menambahkan tanda asterisk (**`*`**) di depan nama pointer yang akan dibuat. Berikut ini bentuk umumnya.

```
tipe data *nama pointer;
```

`tipe_data` di sini menunjukkan bahwa pointer tersebut akan menunjuk ke suatu alamat dimana alamat tersebut ditempati oleh nilai yang memiliki tipe data tertentu. Sebagai contoh, coba Anda perhatikan pendeklarasian pointer `P` berikut ini.

```
/* Mendeklarasikan pointer P yang akan menunjuk ke alamat yang
   ditempati nilai dengan tipe data int */

int *P;
```

Pendeklarasian di atas akan menyebabkan pointer `P` hanya dapat menunjuk ke alamat yang menampung nilai dengan tipe data `int` saja. Artinya, pointer `P` tidak dapat menunjuk ke alamat yang ditempati oleh nilai-nilai dengan tipe data selain `int` (misalnya tipe `double`, `char` atau yang lainnya).

Sama seperti pada pendeklarasian variabel biasa, kita juga dapat mendeklarasikan beberapa buah pointer dengan tipe sama hanya dengan menuliskan satu baris kode. Adapun contohnya dapat Anda lihat di bawah ini.

```
/* MendeklarasikanpointerP1 dan P2  yang akan menunjuk ke tipe
data int */
int *P1, * P2;

/* Mendeklarasikan pointer ptr1 dan ptr2 yang akan menunjuk ke
tipe data double */
double *ptr1, *ptr2;

/* Mendeklarasikan pointer P3 yang akan menunjuk ke tipe data
double dan mendeklarasikan variabel var1 dengan tipe double */
double *P1,  var1;
```

Seperti telah dikemukakan sebelumnya bahwa pointer merupakan variabel yang berisi alamat, maka dari itu kita juga harus mengetahui bagaimana cara untuk mendapatkan

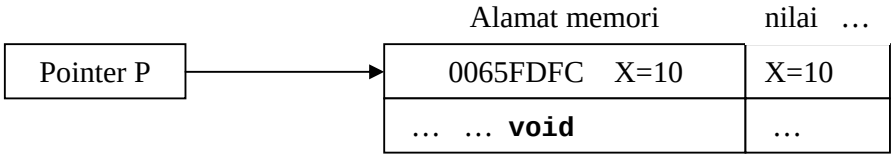
alamat dari suatu variabel di dalam bahasa C. Caranya adalah dengan menggunakan operator `&` di depan nama variabel tersebut. Sebagai contoh, apabila kita memiliki variabel `X` yang bertipe `int` dan pointer `P` yang akan menujuk ke tipe data `int`, maka sintak program yang akan digunakan untuk mendapatkan alamat dari variabel `X` tersebut dan memasukkannya ke dalam pointer `P` adalah sebagai berikut.

```
int X;     /* Mendeklarasikan variabel X dengan tipe int */
int *P;    /* Mendeklarasikan pointer P yang akan menunjuk
              ke tipe int */

P = &X;    /* Mendapatkan alamat dari variabel X dengan cara
              menuliskan &X dan
              menyimpan alamat tersebut ke dalam pointer P */
```

Setelah mengetahui hal di atas, mungkin Anda akan berfikir bagaimana cara untuk mendapatkan nilai apabila yang diketahui hanya alamat yang ditempatinya saja? Jawabnya adalah dengan melakukan *dereference pointer*. Menurut definisinya, *dereference pointer* adalah proses pengambilan nilai dari suatu alamat memori melalui sebuah pointer. Adapun cara untuk mengambil nilai tersebut adalah dengan menambahkan tanda asterisk (*) di depan nama pointer tersebut. Sebagai contoh apabila kita memiliki pointer `P` yang telah menunjuk ke alamat tertentu, kemudian kita ingin mengambil nilai yang terdapat di dalam alamat tersebut untuk dimasukkan ke dalam variabel `Y`, maka sintak yang harus dituliskan adalah sebagai berikut.

```
Y = *P;
```

Agar Anda dapat lebih memahami tentang konsep pointer, perhatikan gambar di bawah ini yang akan menunjukkan hubungan pointer dengan sebuah variabel.



Gambar 7.3. Pointer yang menunjuk ke alamat memori tertentu

Pada gambar di atas, terdapat pointer yang sedang menunjuk ke suatu alamat (misalnya `0065FDFC`), dimana alamat tersebut ditempati oleh variabel `X` yang bernilai 10, maka untuk mendapatkan nilai 10 tersebut kita dapat menggunakan *dereference pointer*, yaitu dengan menuliskan **`*P`**. Dengan demikian, kita dapat menyimpulkan bahwa apabila

```
P = &X;    /* Keduanya menyimpan alamat (yaitu 0065FDFC) */
```

maka

```
*P = X;    /* Keduanya menyimpan nilai (yaitu 10) */
```

Untuk membuktikan hal tersebut di sini kita akan membuat sebuah program sederhana yang menggunakan pointer ke tipe data `int`. Adapun sintaknya adalah sebagai berikut.

```
#include <stdio.h>

int main(void) {

  int *P;  /* Mendeklarasikan pointer P */
  int X;   /* Mendeklarasikan variabel X */
  /* Mengisikan nilai 10  ke dalam variabel X */
  X = 10;
  /*  Mengisikan alamat dari variabel X  ke dalam pointer P */
  P = &X;

  /* Menampilkan nilai */
  printf("Nilai X     : %d\n", X);
  printf("Nilai &X    : %p\n", &X);
  printf("Nilai *P    : %d\n", *P);
  printf("Nilai P     : %p\n\n", P);

  /* Mengisikan nilai 20  ke dalam *P */
  *P = 20;

  /* Menampilkan nilai */
  printf("Nilai X     : %d\n", X);
  printf("Nilai &X    : %p\n", &X);
  printf("Nilai *P    : %d\n", *P);
  printf("Nilai P     : %p\n", P);

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Nilai X     : 10
Nilai &X    : 0074FDE0
Nilai *P    : 10
Nilai P     : 0074FDE0

Nilai X     : 20
Nilai &X    : 0074FDE0
Nilai *P    : 20
Nilai P     : 0074FDE0
```

Dari hasil yang diberikan di atas, di situ terlihat bahwa nilai dari `P` akan sama dengan nilai `&X`, sedangkan nilai `X` akan sama dengan nilai `*P`. Tampak bahwa apabila nilai `X` akan selalu sama dengan nilai `*P`, begitu juga sebaliknya. Hal ini disebabkan oleh karena keduanya merupakan variabel yang tersimpan dalam alamat yang sama.

7.4. Mendeklarasikan Pointer Tanpa Tipe

Sebelumnya Anda telah mempelajari bahwa pointer yang dideklarsikan dengan tipe data tertentu (misalnya tipe `int`), hanya dapat menunjuk alamat yang berisi variabel dengan tipe data yang sesuai saja (dalam hal ini tipe `int`). Dengan kata lain pointer tersebut tidak dapat digunakan untuk menunjuk ke alamat yang berisi variabel dari tipe data lain seperti `float`, `double`, `char` maupun lainnya.

Dalam bahasa C terdapat cara khusus untuk membuat pointer tersebut dapat menunjuk ke alamat-alamat yang berisi variabel dari tipe apapun, yaitu dengan menggunakan kata kunci `void` pada saat pendeklarasiannya. Maka dari itu, pointer seperti ini sering disebut dengan istilah pointer tanpa tipe (*void pointer*). Berikut ini bentuk umum untuk mendeklarasikan suatu pointer tanpa tipe.

```
void *nama_pointer;
```

Untuk membuktikan hal tersebut, perhatikan program di bawah ini.

```
#include <stdio.h>

int main(void) {

  void *P;       /* Mendeklarasikan pointer tanpa tipe dengan
                    nama P*/
  int X = 21;    /* Mendeklarasikan variabel X dengan tipe int */
  double Y = 5.42;    /* Mendeklarasikan variabel Y dengan tipe
                         double */

  /* Memesan ruang memori untuk mengalokasikan nilai dengan tipe
     data int */
  P = (int *) malloc (sizeof(X));

  /* Mengisikan alamat dari variabel X  yang bertipe int */
  P = &X;

  /* Menampilkan nilai */
  printf("Nilai X \t: %d\n", X);
  printf("Nilai &X \t: %p\n", &X);
  printf("Nilai *P \t: %d\n", *((int *) P));
  printf("Nilai P \t: %p\n\n", P);
```

```
  /* Membebaskan memori yang telah dipesan untuk tipe int */
  free(P);


  /* Memesan ruang memori untuk mengalokasikan nilai dengan tipe
     data int */
  P = (double *) malloc (sizeof(Y));

  /* Mengisikan alamat dari variabel Y  yang bertipe double */
  P = &Y;

  /* Menampilkan nilai */
  printf("Nilai Y \t: %3.2fd\n", Y);
  printf("Nilai &Y \t: %p\n", &Y);
  printf("Nilai *P \t: %3.2f\n", *((double *) P));
  printf("Nilai P \t: %p\n", P);

  /* Membebaskan memori yang telah dipesan untuk tipe double */
  free(P);

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Nilai X       : 21
Nilai &X      : 0073FDE0
Nilai *P      : 21
Nilai P       : 0073FDE0

Nilai Y       : 5.42
Nilai &Y      : 0073FDD8
Nilai *P      : 5.42
Nilai P       : 0073FDD8
```

Apabila Anda masih merasa bingung dengan kehadiran fungsi `malloc()` dan `free()` yang terdapat pada sintak di atas, Anda tidak perlu cemas karena materi tersebut akan kita bahas pada sub bab selanjutnya dalam bab ini.

Dari hasil di atas terlihat jelas bahwa pointer `P` yang bertipe `void`, dapat kita gunakan untuk menunjuk alamat yang berisi tipe data yang lain (dalam contoh ini adalah tipe `int` dan `double`).

Satu hal yang perlu diperhatikan dari sintak di atas adalah pada saat kita menuliskan *dereference pointer* untuk tipe `int` dan `double`, digunakan sintak seperti berikut.

```
*( (int *) P)        /* dan */       *( (double *) P)
```

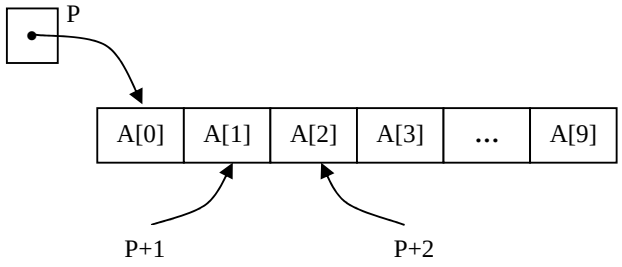
Ini berarti sebelum kita melakukan *dereference pointer*, kita harus menganggap pointer `P` sebagai pointer yang menunjuk ke tipe `int` ataupun `double` terlebih dahulu. Jadi kesimpulannya kita tidak dapat melakukan *dereference pointer* terhadap pointer yang bertipe `void`.

7.5. Pointer dan Array

Sebagai programmer C, Anda harus mengetahui bahwa pointer dan array adalah dua buah hal yang saling berhubungan. Untuk menerangkan hal ini, anggaplah kita memiliki array `A` yang terdiri dari 10 buah elemen yang bertipe int serta pointer `P` yang akan menunjuk ke tipe `int`. Sekarang apabila kita mengeset pointer tersebut untuk menunjuk ke elemen pertama array, maka kita dapat mengakses elemen-elemen dari array A tersebut dengan menggunakan pointer `P`. Perhatikan sintak berikut ini.

```
/* Mendeklarasikan array A dengan 10 buah elemen bertipe int */
int A[10];

/* Mendeklarasikan pointer P dan mengesetnya untuk menunjuk ke
   alamat dari A[0] */
int *P = &A[0]; /* bisa ditulis dengan int *P = A */
```

Melalui sintak di atas kita dapat melakukan pengaksesan elemen array dengan menggunakan `A[0]`, `A[1]`, `A[2]`, …, `A[9]` atau dengan cara `*P`, `*(P+1)`, `*(P+2)`, …, `*(P+9)`. Situasi ini dapat direpresentasikan melalui cara seperti berikut.



Gambar 7.4. Hubungan Pointer dan Array

Dari gambar tersebut dapat kita simpulkan bahwa apabila `P = &A[0]`, maka `P+1 = &A[1]`, `P+2 = &A[2]` dan seterusnya. Di sini kita melakukan operasi aritmetika terhadap pointer `P` (dalam hal ini operasi penjumlahan), proses seperti ini dinamakan dengan aritmetika pointer.

Untuk membuktikan hal tersebut, berikut ini contoh program yang akan menunjukkan hubungan antara pointer dan array. Adapun sintaknya adalah seperti berikut.

```
#include <stdio.h>

int main(void) {

  /* Mendeklarasikan array A dan melakukan inisialisasi nilai ke
     dalamnya */
  int A[10] = {10,20,30,40,50,60,70,80,90,100};

  /* Mendeklarasikan variabel untuk indeks pengulangan */
  int j;

  /* Mendeklarasikan pointer P dan mengesetnya untuk menunjuk
     alamat dari A[0] */
  int *P = &A[0];

  /* Menampilkan elemen array dengan menggunakan akses array
     biasa dan pointer */
  printf("Menggunakan array: \t\tMenggunakan pointer:\n");

  for (j=0; j<10; j++, P++) {
    printf("%d\t\t\t\t%d\n", A[j], *P);
  }

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah seperti yang tampak di bawah ini.

```
Menggunakan array:      Menggunakan pointer:
10                      10
20                      20
30                      30
40                      40
50                      50
60                      60
70                      70
80                      80
90                      90
100                     100
```

Dari hasil yang didapatkan di atas terlihat bahwa pengaksesan elemen array dengan cara biasa dan dengan menggunakan pointer akan memberikan hasil yang sama. Di atas kita melakukan aritmetika pointer dengan melakukan *increment* terhadap pointer `P`, yaitu pada statemen `P++`.

Perlu ditekankan di sini bahwa operasi aritmetika yang dapat dilakukan terhadap pointer hanyalah operasi penjumlahan dan pengurangan saja. Dengan kata lain kita tidak

diizinkan untuk melakukan operasi perkalian dan pembagian. Perhatikan potongan sintak di bawah ini.

```
int A[10];
int *P1 = &A[0];
int *P2 = &A[3];
```

Maka statemen `P2-P1` akan menghasilkan nilai 3. Kenapa 3? Ini merupakan selisih letak alamat memori antara yang ditunjuk oleh pointer `P2` dan `P1`. Untuk membuktikan hal tersebut, di bawah ini kita akan mengimplementasikannya ke dalam sebuah program. Adapun sintaknya adalah sebagai berikut.

```
#include <stdio.h>

int main(void) {
  int A[10] = {10,20,30,40,50,60,70,80,90,100};
  int j, selisih;

  /* Mendeklarasikan pointer P1 dan mengesetnya untuk menunjuk
     alamat dari A[0] */
  int *P1 = &A[0];

  /* Mendeklarasikan pointer P2 dan mengesetnya untuk menunjuk
     alamat dari A[3] */
  int *P2 = &A[3];

  /* Menampilkan semua elemen array */
  for (j=0; j<10; j++) {
    printf("A[%d] = %d\n ", j, A[j]);
  }
  printf("\n");

  /* Melakukan operasi pengurangan terhadap dua buah pointer */
  selisih = P2 – P1;

  /* Menampilkan nilai serta alamat yang ditunjuk oleh
     pointer P1 dan P2 */
  printf("*P1 = %d \tP1 = %p\n", *P1, P1);
  printf("*P2 = %d \tP2 = %p\n", *P2, P2);

  printf("\nP2-P1 = %d\n", selisih);

  /* Melakukan operasi penjumlahan terhadap pointer P1 dan P2 */
  P1 = P1 + 1;  /* Menyebabkan P1 menunjuk ke alamat
                   dari A[1] */
  P2 = P2 + 2;  /* Menyebabkan P2 menunjuk ke alamat
                   dari A[5] */

  /* Menampilkan kembali nilai serta alamat yang ditunjuk oleh
     pointer P1 dan P2 */
  printf("\nSetelah operasi penjumlahan:\n");
```

```
  printf("*P1 = %d \tP1 = %p\n", *P1, P1);
  printf("*P2 = %d \tP2 = %p\n", *P2, P2);

  return 0;
}
```

Apabila dijalankan, program di atas akan memberikan hasil seperti di bawah ini.

```
A[0] = 10
A[1] = 20
A[2] = 30
A[3] = 40
A[4] = 50
A[5] = 60
A[6] = 70
A[7] = 80
A[8] = 90
A[9] = 100

*P1 = 10        P1 = 0073FDB0
*P2 = 40        P2 = 0073FDBC

P2 - P1 = 3

Setelah operasi penjumlahan:
*P1 = 20        P1 = 0073FDB4
*P2 = 60        P2 = 0073FDC4
```

7.6. Pointer dan Fungsi

Pada bagian ini kita akan membahas bagaimana hubungan yang terdapat antara fungsi dan pointer. Dalam bahasa C, pointer dapat digunakan sebagai argumen atau parameter dari suatu fungsi. Selain itu, bahasa C juga mengizinkan kita untuk dapat mendeklarasikan pointer yang akan menunjuk ke sebuah fungsi. Berikut ini penjelasan lebih detil mengenai kedua materi tersebut.

7.6.1. Pointer sebagai Parameter Fungsi

Pada bab sebelumnya kita telah membahas bahwa terdapat dua buah cara untuk melewatkan parameter ke dalam suatu fungsi, yaitu dengan menggunakan *pass by value* (melewatkan berdasarkan nilai) dan *pass by reference* (melewatkan berdasarkan alamat). Alamat yang dimaksud di sini tidak lain adalah pointer. Dalam bab ini kita akan menengok kembali apa sebenarnya yang terjadi pada saat proses pelewatan parameter yang bertipe pointer.

Sebagai contoh untuk menerangkan materi ini, kita akan membuat fungsi yang berguna untuk menukarkan dua buah bilangan (proses *swapping*). Di sini kita akan melewatkan

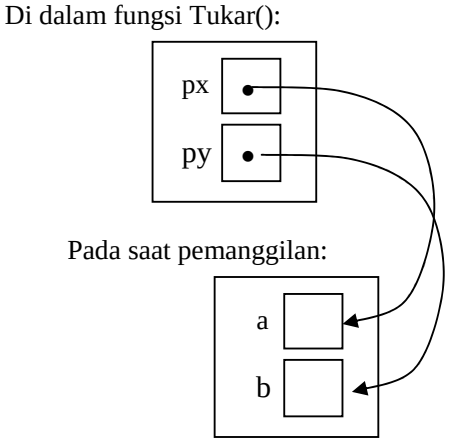
dua buah pointer sebagai parameternya. Adapun sintak pendefinisannya adalah sebagai berikut.

```
void Tukar(int *px, int *py) {
  int temp = *px;
  *px = *py;
  *py = temp;
}
```

Adapun contoh sintak yang melakukan pemanggilan fungsi tersebut adalah sebagai berikut.

```
int a, b;  /* variabel untuk menyimpan nilai yang akan
              ditukar */

Tukar(&a, &b);
```

Sintak di atas akan menukarkan nilai yang terdapat di dalam dua buah alamat memori, yaitu alamat dari variabel `a` dan variabel `b`. Di sini pointer `px` akan menunjuk ke alamat dari variabel `a` (`&a`, sebut saja dengan **`alamat_A`**), sedangkan pointer `py` menunjuk ke alamat dari variabel `b` (`&b`, sebut saja dengan **`alamat_B`**). Setelah itu, nilai yang terdapat pada `alamat_A` akan ditukarkan dengan nilai yang terdapat pada `alamat_B` melalui variabel bantu, yaitu variabel `temp`. Hal inilah yang menyebabkan setelah pemanggilan fungsi `Tukar()`, maka nilai `a` dan `b` akan tertukar. Situasi ini dapat kita representasikan dengan gambar berikut.



Gambar 7.5. Menukarkan dua buah nilai dengan menggunakan pointer

Berikut ini contoh program yang akan membuktikan hal di atas. Adapun sintaknya adalah sebagai berikut.

```
#include <stdio.h>
```

```
void Tukar(int *px, int *py) {
  int temp = *px;
  *px = *py;
  *py = temp;
}

int main(void) {
  int a=10, b=50;

  /* Menampilkan nilai a dan b sebelum ditukar */
  printf("Sebelum pertukaran nilai:\n");
  printf("Nilai a = %d\n", a);
  printf("Nilai b = %d\n", b);

  /* Menukarkan bilangan a dan b dengan cara memanggil fungsi
     Tukar() */
  Tukar(&a, &b);/* Melewatkan alamat dari variabel a dan b */

  /* Menampilkan nilai a dan b sebelum ditukar */
  printf("\nSetelah pertukaran nilai:\n");
  printf("Nilai a = %d\n", a);
  printf("Nilai b = %d\n", b);

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Sebelum pertukaran nilai:
Nilai a = 10
Nilai b = 50

Setelah pertukaran nilai:
Nilai a = 50
Nilai b = 10
```

7.6.2. Pointer ke Fungsi

Suatu pointer dapat digunakan untuk menunjuk alamat dari fungsi yang telah didefinisikan sebelumnya. Hal ini biasanya dilakukan untuk membuat fungsi *callback*. Adapun cara untuk melakukan hal tersebut adalah dengan membuat deklarasi pointer yang menunjuk ke fungsi bersangkutan. Untuk lebih memahaminya, di sini kita akan lagsung membuat contoh program di mana di dalamnya kita mendeklarasikan pointer yang menunjuk ke alamat dari sebuah fungsi. Berikut ini sintak dari program yang dimaksud di atas.

```
#include <stdio.h>

/* Mendeklarasikan pointer pF untuk menunjuk ke alamat fungsi
   yang tidak memiliki parameter */
int (*pF) (int);

/* Mendefinisikan fungsi untuk menghitung nilai faktorial dari
   sebuah bilangan bulat */
int Faktorial(int x) {
  if (x == 0) {
    return 1;
  } else {
    return x * Faktorial(x-1);
  }
}

int main(void) {
  /* Memerintahkan pointer pF untuk menunjuk ke alamat dari
     fungsi Faktorial() */
  pF = &Faktorial;    /* dapat ditulis dengan pF = Faktorial; */

  int n;
  printf("Masukkan nilai yang akan dihitung : ");
  scanf("%d", &n);

  /* Menampilkan nilai faktorial melalui pointer pF */
  printf("%d! = %d", n, pF(n));

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Masukkan nilai yang akan dihitung : 5
5! = 120
```

Perhatikan kembali sintak di atas secara teliti, kita dapat menunjuk alamat dari fungsi `Faktorial()` dengan pointer `pF`, sehingga kita dapat melakukan proses yang terdapat dalam fungsi `Faktorial()` melalui pointer `pF`. Dalam contoh di atas, kita melakukan pemanggilan fungsi tersebut dengan cara menuliskan statemen `pF(5)`.

Sebagai bahan perbandingan dengan contoh sebelumnya, di sini kita akan menuliskan program lainnya yang sama-sama menggunakan pointer untuk mengakses sebuah fungsi. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>
```

```
/* Mendeklarasikan pointer pT untuk menunjuk alamat dari fungsi
yang memiliki dua buah parameter bertipe pointer ke tipe int */
void (*pT) (int*, int*);

/* Mendefinisikan fungsi untuk menukarkan dua buah bilangan */
void Tukar(int *px, int *py) {
  int temp = *px;
  *px = *py;
  *py = temp;
}

int main(void) {
  /* Mendeklarasikan dua buah variabel yang nilainya akan
     ditukar */
  int a=10, b=50;

  pT = Tukar;

  /* Menampilkan nilai a dan b sebelum ditukar */
  printf("Sebelum pertukaran nilai:\n");
  printf("Nilai a : %d\n", a);
  printf("Nilai b : %d\n", b);

  /* Menukarkan nilai a dan b dengan cara menggunakan
     pointer pT */
  pT(&a, &b);

  /* Menampilkan nilai a dan b setelah ditukar */
  printf("\nSetelah pertukaran nilai:\n");
  printf("Nilai a : %d\n", a);
  printf("Nilai b : %d\n", b);

  return 0;
}
```

Hasil yang akan diberikan dari program tersebut adalah sebagai berikut.

```
Sebelum pertukaran nilai:
Nilai a : 10
Nilai b : 50

Setelah pertukaran nilai:
Nilai a : 50
Nilai b : 10
```

7.7. Pointer `NULL`

Dalam mendeklarasikan pointer, sebaiknya kita mengeset pointer tersebut untuk tidak menunjuk ke lokasi acak di memori. Untuk melakukan hal ini, bahasa C telah menyediakan pointer `NULL`, yaitu pointer khusus yang tidak menunjuk ke alamat

manapun. Dalam bahasa C, nilai yang digunakan untuk merepresentasikan nilai null ini adalah dengan menggunakan konstanta `NULL`, yaitu konstanta yang telah didefinisikan di dalam beberapa file header (yaitu `<stdio.h>`, `<stddef.h>` dan `<string.h>`). Dengan demikian, apabila kita ingin menginisialisasi suatu pointer ke nilai null, maka kita dapat menuliskannya sebagai berikut.

```
int *P = NULL;
```

Selain konstanta `NULL` di atas, kita juga dapat menggunakan nilai **`0`** untuk merepresentasikan nilai null, sehingga kita mengganti sintak di atas dengan sintak di bawah ini.

```
int *P = 0;
```

Nilai `NULL` dan `0` adalah sama karena nilai `NULL` sebenarnya merupakan konstanta yang berupa makro, dimana kehadirannya akan digantikan dengan 0.

Apabila kita ingin melakukan pengecekan terhadap suatu pointer, apakah bernilai null atau tidak, maka kita dapat melakukannya dengan cara berikut.

```
if (P != NULL) {
  /* Statemen yang akan dilakukan */
}

atau

if (P != 0) {
  /* Statemen yang akan dilakukan */
}
```

Sebagai contoh, di sini kita akan membuat fungsi untuk mencari teks di dalam teks lain, seperti halnya fungsi `strstr()`, dimana fungsi ini akan mengembalikan nilai `NULL` apabila teks yang dicari tersebut tidak ditemukan. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>
#include <string.h>   /* untuk menggunakan fungsi strcpy() */

/* Mendefinisikan fungsi untuk pencarian teks */
char *CariTeks(char teks[], char cari[]) {
  char *Pteks, *P1, *P2;

  for (Pteks = &teks[0]; *Pteks != '\0'; Pteks++) {
```

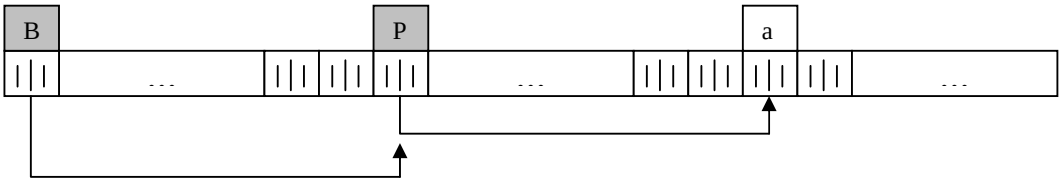
```
    P1 = cari;
    P2 = Pteks;
    while (P1 != '\0') {
      /* Apabila karakter tidak sama, maka keluar dari
         pengulangan */
      if (*P1 != *P2) {
        break;
      }
      P1++;
      P2++;
    }
    /* Apabila ditemukan */
    if (*P1 == '\0') {
      return Pteks;
    }
  }
  return NULL;
}

int main(void) {
  char *hasil;

  strcpy(hasil, CariTeks("Saya sedang belajar bahasa C",
         "belajar"));
  printf("%s", hasil);

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
belajar bahasa C
```

7.8. Pointer Ke Pointer

Bagi Anda yang merupakan programmer pemula pasti akan mengalami sedikit kebingungan mengenai materi ini. Mungkin Anda akan bertanya pointer adalah sebuah variabel yang bernilai alamat dari variabel lain, maka apa isi dari pointer yang menunjuk ke pointer lain? Jawabnya adalah alamat yang ditempati oleh pointer yang sedang menunjuk ke variabel lain tersebut. Anggaplah kita memiliki pointer `P` yang menunjuk ke alamat dari variabel `a` (bertipe `char`). Kemudian kita mendeklarasikan pointer `B` yang akan menunjuk ke alamat dari pointer `P`, maka situasi ini dapat kita gambarkan dengan cara berikut.



Gambar 7.6. Pointer ke pointer

Kejadian seperti ini sering dinamakan dengan istilah *multiple indirection*. Adapun cara untuk mendeklarasikan sebuah pointer ke pointer adalah dengan menambahkan tanda asterisk (*) sebanyak dua kali di depan nama pointer yang akan dideklarasikan. Berikut ini bentuk umumnya.

```
tipe_data **nama_pointer;
```

Untuk lebih memperjelas pembahasan mengenai materi ini, di sini kita akan membuat program yang didalamnya terdapat pendeklarasian sebuah pointer ke pointer yang sedang menunjuk ke alamat dari variabel lain. Adapun contoh sintaknya adalah sebagai berikut.

```
#include <stdio.h>

int main() {

  /* Mendeklarasikan variabel a bertipe char dan pointer P yang

     akan menunjuk ke tipe char */

  char a, *P;

  /* Mendeklarasikan pointer B untuk menunjuk ke alamat dari ke
     pointer P */
  char **B;

  a = 'A';
  P = &a;
  B = &P;

  /* Menampilkan nilai yang dikandung di dalam variabel a,
     pointer P dan B */
  printf("Nilai a \t\t = %c\n", a);
  printf("Alamat a \t\t = %p\n\n", &a);

  printf("Alamat yang ditunjuk P \t = %p\n", P);
  printf("Alamat P \t\t = %p\n", &P);
  printf("Nilai *P \t\t = %c\n\n", *P);

  printf("Alamat yang ditunjuk B \t = %p\n", B);
  printf("Alamat B \t\t = %p\n", &B);
  printf("Nilai *B \t\t = %p\n", *B);
```

```
  printf("Nilai **B \t\t = %c\n", **B);
  return 0;
}
```

Program di atas akan memberikan hasil sebagai berikut.

```
Nilai a                 = A
Alamat a                = 0073FDE7

Alamat yang ditunjuk P  = 0073FDE7
Alamat P                = 0073FDE0
Nilai *P                = A

Alamat yang ditunjuk B  = 0073FDE0
Alamat B                = 0073FDDC
Nilai *B                = 0073FDE7
Nilai **B               = A
```

Apabila Anda perhatikan secara teliti hasil dari program di atas, maka Anda dapat menyimpulkan bahwa nilai dari variabel `a` (yaitu karakter `'A'`) dapat kita akses melalui pointer `P` dengan cara menuliskan `*P` dan dengan menggunakan pointer `B` dengan cara menuliskan `**B`. Sedangkan alamat dari variabel `a` (yaitu `0073FDE7`), dapat kita akses melalui perintah `&a`, `P` ataupun `*B`. Hal ini menunjukkan bahwa pointer `B` bukanlah pointer yang menunjuk ke alamat dari sebuah variabel dengan tipe `char`, melainkan pointer yang menunjuk ke alamat dari pointer `P`. Sebagai bukti dari pernyataan tersebut adalah pada saat kita melakukan *dereference pointer* pada pointer `B` (yaitu dengan menuliskan `*B`), maka nilai yang didapatkan bukanlah sebuah nilai bertipe karakter, melainkan sebuah alamat memori (yaitu alamat yang ditunjuk oleh pointer `P` atau alamat dari variabel `a`).

Apabila kita amati, contoh yang kita ambil di atas sebenarnya dapat dikatakan sebagai pointer ke string. Pasalnya kita telah mendeklarasikan pointer ke tipe `char*` (string). Sebagai bahan tambahan bagi Anda untuk dapat benar-benar memahami konsep dari pointer ke pointer, berikut ini akan disajikan sebuah contoh program lagi dimana di dalamnya terdapat deklarasi pointer ke pointer (dalam hal ini pointer ke string). Sebelumnya ingat kembali bahwa string adalah array dari karakter, sedangkan array sebenarnya adalah pointer, maka dari itu kita dapat menuliskan sintak seperti berikut.

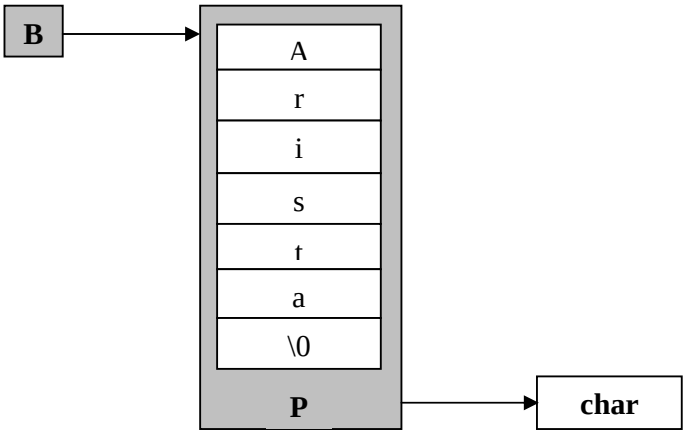
```
#include <stdio.h>

int main(void) {

  /* Mendeklarasaikan array dari karakter (disebut dengan
     string) */
  char S[] = "Arista";

  /* Mendeklarasikan pointer ke tipe char */
  char *P = S;     /* Pointer P menunjuk ke array S[0], dapat
```

```
                        dikatakan bahwa P = S */

  /* Mendeklarasikan pointer ke pointer dan menunjuk ke alamat
     pointer P */
  char **B = &P;
  /* Menampilkan nilai yang dikandung oleh pointer P dan B */
  printf("Alamat yang ditunjuk P \t= %p\n", P);/* P dalam
                                                  bentuk pointer */
  printf("Nilai P \t\t= %s\n", P); /* P dalam bentuk string */
  printf("Nilai *P \t\t= %c\n", *P);
  printf("Alamat P \t\t= %p\n\n", &P);

  printf("Alamat yang ditunjuk B \t= %p\n", B);
  printf("Nilai *B \t\t= %p\n", *B);    /* *B dalam bentuk
                                          pointer */
  printf("Nilai *B \t\t= %s\n", *B);    /* *B dalam bentuk
                                          string */
  printf("Nilai **B \t\t= %c\n", **B);
  printf("Alamat B \t\t= %p\n", &B);

  return 0;
}
```

Hasil yang akan diberikan oleh program di atas adalah sebagai berikut.

```
Alamat yang ditunjuk P      = 0073FDD0
Nilai P                     = Arista
Nilai *P                    = A
Alamat P                    = 0073FDCC

Alamat yang ditunjuk B      = 0073FDCC
Nilai *B                    = 0073FDD0
Nilai *B                    = Arista
Nilai **B                   = A
Alamat B                    = 0073FDC8
```

Untuk lebih mempermudah, kasus ini dapat kita representasikan dengan gambar di bawah ini.



Gambar 7.7. Pointer ke string

Ini artinya pointer `B` menunjuk ke pointer `P`, sedangkan pointer `P` sendiri adalah pointer yang menunjuk ke tipe `char`. Pointer `P` inilah yang disebut dengan string, sehingga situasi di atas sering dinamakan sebagai pointer ke string.

7.9. Konstanta pada Pointer

Pointer dapat bersifat konstan, artinya pointer tersebut hanya dapat menunjuk dari alamat tertentu saja. Untuk melakukan hal ini kita harus menggunakan kata kunci `const` yang telah disediakan dalam bahasa C. Namun, terdapat hal yang harus diperhatikan dalam menggunakan kata kunci tersebut, yaitu masalah penempatan dalam penulisannya pada saat pendeklarasian pointer. Kata kunci `const` dapat ditempatkan sebelum tipe data, setelah tipe data maupun keduanya (sebelum dan sesudah tipe data). Penempatan penulisan yang salah tentu akan menyebabkan perbedaan arti pula.

7.9.1. Kata Kunci `const` Sebelum Tipe Data

Bentuk umum dari penempatan kata kunci `const` sebelum tipe data adalah sebagai berikut.

const *tipe_data* * nama_pointer;

Pada bentuk ini, pointer akan menunjuk ke alamat yang ditempati oleh sebuah nilai yang bersifat konstan. Artinya kita tidak dapat melakukan perubahan terhadap nilai tersebut. Berikut ini contoh program yang akan menunjukkan hal di atas.

```
#include <stdio.h>

int main(void) {

  /* Mendeklarasikan pointer P untuk menunjuk ke alamat yang
     ditempati oleh nilai yang konstan (dengan tipe int) */
  const int *P;
```

```
  int x, y;

  x = 10;
  y = 12;

  P = &x;

  *P = 5;  /* SALAH, karena nilai dari alamat yang ditunjuk
              bersifat konstan */
  x = 30;  /* SALAH, karena nilai x besifat konstan */

  P = &y;  /* BENAR, hal ini diizinkan karena pointer P tidak
              bersifat konstan */

  return 0;
}
```

Pada program di atas tampak statemen `const int *P` yang menyatakan bahwa kita melakukan deklarasi konstanta terhadap nilai yang alamatnya akan ditunjuk oleh pointer `P`. Dengan demikian, apabila pointer `P` menunjuk ke alamat dari variabel `x`, maka nilai `x` ataupun `*P` tidak dapat diubah. Sedangkan pointer `P` di atas tidak bersifat konstan sehingga kita masih dapat memerintahkan pointer tersebut untuk menunjuk ke alamat dari variabel lain (dalam hal ini `y`).

7.9.2. Kata Kunci `const` Setelah Tipe Data

Bentuk umum dari penempatan kata kunci `const` setelah tipe data adalah sebagai berikut.

```
tipe_data * const nama_pointer;
```

Berbeda dengan sebelumnya, di sini kita mendeklarasikan pointer yang bersita konstan. Artinya pointer tersebut hanya dapat menunjuk ke satu buah alamat tertentu saja. Berikut ini contoh program yang akan menunjukkan hal tersebut.

```
#include <stdio.h>

int main(void) {

  /* Mendeklarasikan pointer P yang bersifat konstan */
  int *const P;

  int x, y;

  x = 10;
  y = 12;

  P = &x;

  *P = 5;    /* BENAR, karena nilai dari alamat yang ditunjuk
```

```
                tidak bersifat konstan */
  x = 30;    /* BENAR, karena nilai x tidak besifat konstan */

  P = &y;   /* SALAH, karena pointer P bersifat konstan */

  return 0;
}
```

Kali ini kita mendeklarasikan konstanta pointer, yaitu dengan nama `P`, dimana pointer tersebut tidak dapat menunjuk ke alamat dari variabel lain (dalam hal ini variabel `y`). Pada kasus di atas pointer `P` hanya dapat menunjuk ke alamat dari variabel `x` saja. Namun di sini kita masih diperbolehkan untuk mengubah nilai dari variabel `x` maupun `*P`.

7.9.3. Kata Kunci `const` Sebelum dan Setelah Tipe Data

Bentuk umum dari penempatan kata kunci `const` sebelum dan setelah tipe data adalah sebagai berikut.

```
const tipe_data * const nama_pointer;
```

Bentuk yang ketiga ini merupakan gabungan dari bentuk pertama dan kedua. Hal ini berarti bahwa di sini kita melakukan deklarasi konstanta terhadap nilai dari variabel dan juga nilai dari pointernya. Dengan melakukan hal ini, maka kita tidak diizinkan untuk mengubah nilai dari variabel maupun mengubah alamat yang telah ditunjuk oleh pointer tersebut. Berikut ini contoh program yang akan menunjukkan konsep di atas.

```
#include <stdio.h>

int main(void) {

  /* Mendeklarasikan konstanta terhadap variabel dan juga
     pointer P */
  const int *const P;

  int x, y;

  x = 10;
  y = 12;

  P = &x;

  *P = 5;    /* SALAH, karena nilai dari alamat yang ditunjuk
                bersifat konstan */
  x = 30;    /* SALAH, karena nilai x besifat konstan */

  P = &y;   /* SALAH, karena pointer P bersifat konstan */

  return 0;
```

```
}
```

Pada program di atas, pointer `P` hanya dapat menunjuk ke alamat dari variabel `x` saja. Selain itu kita juga tidak diizinkan untuk melakukan perubahan nilai terhadap variabel dari alamat yang ditunjuk oleh pointer `P`.

7.10. Mengalokasikan Memori

Dalam dunia pemrograman kita diizinkan untuk mengalokasikan ruang memori secara dinamis pada saat program berjalan sesuai dengan yang kita butuhkan, proses seperti ini dinamakan dengan alokasi memori dinamis (*dynamic memory allocation*). Sedangkan pada saat kita mendeklarasikan variabel (baik global maupun lokal) di dalam kode program yang kita tulis, kompiler secara otomatis akan memesan sejumlah ruang memori untuk menampung variabel tersebut, proses seperti ini disebut dengan alokasi memori statis (*static memory allocation*).

Untuk melakukan alokasi memori secara dinamis kita membutuhkan pointer untuk mencatat alamat-alamat baru yang dipesan. Bahasa C telah menyediakan fungsi-fungsi yang digunakan untuk proses pemesanan memori secara dinamis, yaitu fungsi `malloc()`, `calloc()` dan `realloc()`. Adapun penjelasan dari masing-masing fungsi tersebut dapat Anda lihat pada sub bab di bawah ini.

7.10.1. Menggunakan Fungsi `malloc()`

Fungsi ini mempunyai bentuk prototipe sebagai berikut.

```
void *malloc(size_t n);
```

Fungsi `malloc()` akan mengembalikan pointer ke sejumlah *n* byte ruang memori yang belum diinisialisasi. Apabila tidak terpenuhi, maka fungsi ini akan mengembalikan nilai `NULL`. Hal yang perlu diperhatikan dalam menggunakan fungsi ini adalah kita harus melakukan *typecasting* terhadap pointer yang akan dipesan sesuai dengan tipe data yang diinginkan. Perhatikan contoh penggunaan fungsi `malloc()` di bawah ini.

```
int array[10];
int *P;

P = (int*) malloc (10 * sizeof(int));
```

Sintak di atas artinya kita memesan atau mengalokasikan ruang memori sebanyak 40 byte (berasal dari 10x4) untuk menyimpan data yang bertipe int dan mencatat ruang tersebut ke dalam pointer P. Adapun nilai 4 di sini berasal dari ukuran tipe data `int`, yang dituliskan dengan `sizeof(int)`. Jadi apabila kita ingin mengalokasikan ruang memori untuk tipe data `double`, sedangkan kita telah mengetahui bahwa tipe data `double` berukuran 8 byte, maka kita dapat menuliskannya sebagai berikut.

```
double *P;

P = (double*) malloc(8); /* atau dapat juga ditulis dengan
                         P = (double*) malloc(sizeof(double)); */
```

Walaupun cara di atas diizinkan oleh kompiler serta program juga dapat berjalan dengan benar, namun kebanyakan dari programmer C pada umumnya lebih memilih untuk menggunakan operator `sizeof` untuk mendapatkan ukuran dari tipe data tertentu.

7.10.2. Menggunakan Fungsi `calloc()`

Fungsi ini mempunyai bentuk prototipe sebagai berikut.

```
void *calloc(size_t n, size_t size);
```

Fungsi `calloc()` akan mengembalikan pointer ke sebuah array yang terdiri dari *n* elemen data dengan *size* (ukuran) yang ditentukan. Apabila tidak terpenuhi, maka fungsi ini akan mengembalikan nilai `NULL`. Berbeda dengan fungsi `malloc()` yang tidak melakukan inisialiasi, pada fungsi `calloc()` ini ruang yang dialokasikan akan diinisialisasi dengan nilai **nol**. Untuk dapat lebih memahaminya, coba Anda perhataikan contoh penggunaan fungsi `calloc()` berikut.

```
int *P;

P = (int*) calloc(20, sizeof(int));
```

Sintak di atas berarti mengalokasikan 80 byte ruang memori (yang berasal dari 20x4). Dengan kata lain, terdapat 20 elemen array yang masing-masing elemennya berukuran 4 byte. Adapun nilai 4 di sini merupakan ukuran dari tipe data `int`.

7.10.3. Menggunakan Fungsi `realloc()`

Fungsi terakhir yang dapat digunakan untuk mengalokasikan memori di dalam bahasa C adalah fungsi `realloc()`.Berikut ini bentuk prototipe dari fungsi tersebut.

```
void *realloc(void *p, size_t size);
```

Perlu ditekankan bahwa fungsi `realloc()` ini sebenarnya digunakan untuk melakukan perubahan terhadap alokasi memori yang sebelumnya telah dilakukan dengan menggunakan fungsi `malloc()` atau `calloc()`. Dengan kata lain, fungsi ini hanya digunakan untuk alokasi ulang apabila ternyata ruang yang dialokasikan oleh fungsi `malloc()` atau `calloc()` kurang besar. Fungsi `realloc()` ini akan mengembalikan pointer ke ruang baru yang ditambahkan, atau mengembalikan nilai `NULL` apabila

permintaan ruang tersebut tidak dipenuhi. Berikut ini contoh penggunaan fungsi `realloc()` untuk menambahkan ruang memori yang telah dialokasikan.

```
int *P;
P = (int*) calloc(10, sizeof(int)); /* Memesan ruang 40 byte */
realloc((int*) P, 80); /* Memesan ruang sebanyak 40 byte lagi */
```

7.11. Mendealokasikan Memori

Untuk menghindari adanya pemborosan memori ataupun terjadinya *memory leak* (kebocoran memori), maka sebaiknya kita melakukan dealokasi terhadap ruang-ruang memori yang sebelumnya telah dialokasikan melalui fungsi `malloc()`, `calloc()` maupun `realloc()`. Dalam bahasa C, proses ini akan dilakukan dengan menggunakan fungsi `free()` yang memiliki parameter berupa pointer. Berikut ini prototype dari fungsi free().

```
void *free(void *p);
```

p di sini haruslah berupa pointer yang sebelumnya dialokasikan dengan menggunakan fungsi `malloc()`, `calloc()` ataupun `realloc()`. Berikut ini contoh penggunaan fungsi `free()`.

```
#include <stdio.h>

int main(void) {
  int *P;

  P = (int*) malloc(sizeof(int));

  /* Kode yang akan dilakukan */
  ...

  /* Mendealokasikan alamat yang telah selesai digunakan */
  free(P);

  return 0;
}
```

7.12. Memory Leak

Bagi kebanyakan programmer pemula, penggunaan pointer biasanya akan menyebabkan *memory leak* (kebocoran memori), yaitu peristiwa dimana terdapat adanya ruang memori yang terbuang secara sia-sia karena ruang memori tersebut sudah tidak dapat di akses untuk didealokasikan. *Memory leak* ini dapat mengakibatkan program ataupun

sistem operasi kita menjadi rusak ataupun mengalami *hang*. Berikut ini contoh sintak yang akan mengakibatkan terjadinya *memory leak*.

```
#include <stdio.h>

int main(void) {

  /* Mendeklarasikan pointer P yang akan menunjuk ke tipe data
     int */
  void *P;

  int x = 10;
  double y = 15.3;

  /* Memesan ruang memori untuk menempatkan tipe int */
  P = (int *) malloc(sizeof(int));

  /* Memerintahkan pointer P untuk menunjuk ke alamat dari
     variabel x */
  P = &x;

  /* Menampilkan nilai yang terkandung dalam pointer P */
  printf("Nilai P \t=  %p\n", P);
  printf("Nilai *P \t=  %d\n\n", *((int *) P));

  /* Memesan ruang memori untuk menempatkan tipe double */
  P = (double *) malloc(sizeof(double));

  /* Memerintahkan pointer P untuk menunjuk ke alamat dari
     variabel y */
  P = &y;

  /* Menampilkan nilai yang terkandung dalam pointer P */
  printf("Nilai P \t=  %p\n", P);
  printf("Nilai *P \t=  %.1f\n", *((double *) P));

  return 0;
}
```

Sepintas program di atas seperti benar dan apabila dijalankan juga akan memberikan hasil seperti berikut.

```
Nilai P         = 0073FDE0
Nilai *P        = 10

Nilai P         = 0073FDD8
Nilai *P        = 15.3
```

Pada program di atas, mula-mula kita memesan sejumlah ruang (alamat memori) untuk menempatkan nilai yang bertipe `int` (yaitu pada alamat `0073FDE0`) kemudian mencatatnya ke dalam pointer `P`. Setelah melakukan proses terhadap pointer tersebut, di atas kita memesan ruang kembali untuk menampung nilai dengan tipe `double` (yaitu pada alamat `0073FDD8`) dan mencatatnya kembali ke pointer `P`. Hal ini mengakibatkan pointer `P` yang tadinya menunjuk alamat `0073FDE0` berpindah untuk menunjuk ke alamat `0073FDD8`, sehingga alamat `0073FDE0` tidak dapat diakses lagi yang akan mengakibatkan terbuangnya ruang memori secara sia-sia. Kejadian semacam inilah yang dinamakan dengan *memory leak*.

Untuk menghindari hal tersebut, seharusnya kita mendealokasikan alamat `0073FDE0` terlebih dahulu sebelum memerintahkan pointer `P` untuk menunjuk ke alamat yang baru, yaitu dengan menggunakan fungsi `free()`. Jadi, sintak program tersebut seharusnya dituliskan sebagai berikut.

```
#include <stdio.h>

int main(void) {
  void *P;

  int x = 10;
  double y = 15.3;

  P = (int *) malloc(sizeof(int));
  P = &x;
  printf("Nilai P \t=  %p\n", P);
  printf("Nilai *P \t=  %d\n\n", *((int *) P));

  /* Mendealokasikan pointer P dengan fungsi free() */
  free(P);

  P = (double *) malloc(sizeof(double));
  P = &y;
  printf("Nilai P \t=  %p\n", P);
  printf("Nilai *P \t=  %.1f\n", *((double *) P));

  return 0;
}
```

Bab

8

Struktur dan Union

8.1. Pendahuluan

Struktur merupakan sekumpulan variabel yang mungkin terdiri dari beberapa tipe data berbeda dan dikelompokkan dalam satu nama untuk kemudian diakses oleh program. Tipe data yang dimaksud di sini meliputi tipe data dasar dan tipe data bentukan seperti array, pointer dan juga struktur lain yang telah didefinisikan sebelumnya. Dalam beberapa bahasa pemrograman lain (misalnya Pascal), struktur sering disebut dengan istilah rekaman (*record*). Kehadiran struktur akan sangat membantu untuk menyederhanakan masalah dalam pengaturan data yang relatif kompleks, karena dalam struktur kita diizinkan untuk mengelompokkan data-data yang saling berhubungan tersebut ke dalam satu entitas yang terpisah. Dalam program-program besar, pendefinisian struktur biasanya dilakukan dalam unit atau file tersendiri secara terpisah, hal ini tentu akan membuat program yang kita kembangkan dapat lebih modular.

Bayangkan apabila kita akan mengembangkan sebuah program besar yang berhubungan banyak data, misalnya untuk sistem informasi nasabah bank. Di sini, setiap nasabah juga mempunyai data detil tertentu seperti nomor rekening, nama, alamat, jenis kelamin dan lain-lain. Pada kasus ini kita tidak mungkin untuk mendeklarasikan variabel untuk setiap data-data dari nasabah tersebut, karena cara tersebut tentu tidak akan efesien. Untuk itu kita dapat menyelesaikannya dengan cara memasukkan informasi mengenai nasabah-nasabah tersebut ke dalam sebuah struktur.

8.2. Dasar-Dasar Struktur

Sebelum melangkah lebih jauh mengenai pembahasan struktur, pada bagian ini akan diterangkan terlebih dahulu konsep-konsep dasar dari sebuah struktur, yaitu yang meliputi cara pendefinisiannya, ukuran yang dimilikinya, inisialisasi serta pendefinisian struktur yang berisi struktur lainnya.

8.2.1. Mendefinisikan Struktur

Untuk mendefinisikan sebuah struktur di dalam bahasa C kita harus menggunakan kata kunci `struct` yang diikuti dengan nama struktur, kemudian diikuti dengan blok (yang diapit dengan tanda `{ }`) dimana isinya adalah variabel-variabel (disebut *member* atau *field*) yang akan dideklarasikan di dalam struktur tersebut. Hal yang perlu untuk diingat adalah pendefinisian struktur harus selalu diakhiri oleh tanda titik koma (*semicolon*), Untuk lebih jelasnya, berikut ini bentuk umum dari pendefinisian struktur dalam bahasa C.

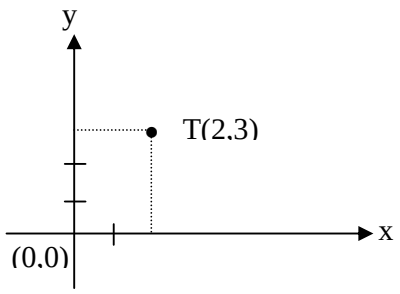
```
struct nama_struktur {
      tipe_data field1;
      tipe_data field2;
      …
} var1, var2, …; /* ingat tanda titik koma */
```

Field1, *Field2*, … di atas menunjukkan anggota dari struktur yang didefinisikan, sedangkan *var1*, *var2*, … menunjukkan variabel yang dideklarasikan dengan tipe struktur yang bersangkutan.

Sekarang mungkin Anda bertanya bagaimana cara mengakses nilai-nilai *field* yang terdapat di dalam struktur tersebut? Jawabnya adalah dengan menggunakan operator unary **.** (operator titik). Berikut ini bentuk umumnya.

```
nama_struktur.nama_field
```

Sebagai contoh untuk pendefinisian struktur, kita tahu bahwa suatu titik (koordinat) tertentu di dalam dimensi dua pasti akan memiliki absis (sumbu `X`) dan ordinat (sumbu `Y`). Di sini kita akan mendefinisikan sebuah struktur dengan nama `TITIK` dimana anggotanya adalah `x` dan `y` yang masing-masing betipe `int`. Perhatikan gambar di bawah ini.



Gambar 8.1. Titik T dalam diagram kartesian

Pada gambar di atas, dapat diartikan bahwa `T` adalah variabel yang bertipe struktur `TITIK` dan menempati koordinat (2,3).

Adapun sintak yang diperlukan untuk melakukan hal tersebut adalah sebagai berikut.

```
struct TITIK {
  int x;
  int y;
};
```

Sekarang apabila kita ingin mendeklarasikan variabel `T` sebagai sebuah titik, maka kita dapat menuliskannya sebagai berikut.

```
/* Mendeklarasikannya langsung pada saat mendefinisikan
   struktur */
struct TITIK {
      int x;
      int y;
} T;

/* atau dengan cara di bawah ini (dengan cacatan struktur TITIK
   sudah terdefinisi) */

struct TITIK T;
```

Untuk lebih memahaminya, berikut ini contoh program yang akan menunjukkan pendefinisian dan penggunaan sebuah struktur.

```
#include <stdio.h>

/* Mendefinisikan struktur TITIK sekaligus mendeklarasikan
   variabel T1 */
struct TITIK {
  int x;
  int y;
} T1;      / * struktur T1 merupakan variabel global */

int main(void) {

  /* Mendeklarasikan struktur T2 */
  struct TITIK T2;      /* struktur T2 merupakan variabel lokal
                           dalam fungsi main() */

  /* Membaca nilai x dan y untuk struktur T1 */
  printf("Menentukan koordinat T1:\n");
  printf("Nilai x = "); scanf("%d", &T1.x);
  printf("Nilai y = "); scanf("%d", &T1.y);

  /* Membaca nilai x dan y untuk struktur T2 */
  printf("\nMenentukan koordinat T2:\n");
  printf("Nilai x = "); scanf("%d", &T2.x);
  printf("Nilai y = "); scanf("%d", &T2.y);

  /* Menampilkan nilai yang terdapat pada struktur */
  printf("\nT1(%d,%d)\n", T1.x, T1.y);
  printf("\nT2(%d,%d)\n", T2.x, T2.y);

  return 0;
}
```

Contoh hasil yang akan diberikan oleh program di atas adalah seperti terlihat di bawah ini.

```
Menentukan koordinat T1:
Nilai x = 2
Nilai y = 3

Menentukan koordinat T2:
Nilai x = 4
Nilai y = 5

T1(2,3)
T2(4,5)
```

Kita dapat juga memasukkan nilai yang terdapat pada struktur tertentu ke dalam struktur lain yang sejenis. Perhatikan potongan sintak di bawah ini.

```
/* Mendefinisikan mendeklarasikan struktur */
struct TITIK {
  int x;
  int y;
} T1, T2;

/* Mengisikan nilai X ke dalam struktur T1 */
T1.x = 10;
T1.y = 20;
```

Sekarang apabila kita ingin memasukkan nilai yang terdapat pada struktur `T1` tersebut ke dalam struktur `T2`, maka kita dapat melakukkannya dengan menuliskan sintak di bawah ini.

```
T2 = T1;
```

Sintak tersebut sama artinya dengan sintak berikut.

```
T2.x = T1.x;
T2.y = T1.y;
```

8.2.2. Inisialisasi Struktur

Sama seperti halnya pada array, kita juga dapat melakukan inisialisasi nilai terhadap field yang terdapat dalam struktur. Berikut ini contoh program yang akan memperlihatkan hal tersebut.

```
#include <stdio.h>

struct TANGGAL {
  char *tanggal;
  char *bulan;
  char *tahun;
} tgl_lahir = {"21","03","1978"};

int main(void) {
  printf("Tanggal lahir : %s-%s-%s",
         tgl_lahir.tanggal,
         tgl_lahir.bulan,
         tgl_lahir.tahun);
  return 0;
}
```

8.2.3. Ukuran Struktur dalam Memori

Hal yang perlu diketahui oleh para programmer dalam bekerja dengan struktur adalah masalah ruang (ukuran) memori yang dibutuhkan untuk mendefinisikan sebuah struktur. Ukuran memori yang dibutuhkan tentunya berbeda-beda tergantung dari tipe data dan banyaknya *field* yang terdapat dalam struktur yang bersangkutan. Kita dapat mendapatkan ukuran dari sebuah struktur dengan menggunakan kata kunci `sizeof`.

Perhatikan pendefinisan struktur di bawah ini.

```
struct STRUKTUR1 {
  char a;
  char b;
};
```

Kita tahu bahwa ukuran dari tipe data `char` adalah 1 byte, maka tipe dari struktur `STRUKTUR1` di atas adalah 2 byte, karena di dalamnya terdapat pendeklarasian dua buah variabel yang bertipe `char`. Namun perhatikan kembali pendefinisan struktur di bawah ini.

```
struct STRUKTUR2 {
  int a;
  char b;
};
```

Ukuran dari struktur `STRUKTUR2` di atas bukan 5 byte (tipe int 4 byte ditambah tipe char 1 byte), melainkan 8 byte (tipe int 4 byte dikali 2). Hal ini disebabkan oleh adanya proses perentangan (*alignment*) terhadap tipe data char yang disesuaikan dengan tipe

data int. Proses tersebut akan ditentukan secara otomatis oleh kompiler melalui operator `sizeof` dengan cara mengambil nilai yang sesuai dengan banyak dan tipe data dari *field* yang terdapat dalam struktur.

Untuk membuktikan hal tersebut, Anda dapat menuliskan program berikut.

```
#include <stdio.h>

struct STRUKTUR1 {
  char a;
  char b;
};
struct STRUKTUR2 {
  char a;
  int b;
};

int main(void) {

  /* Mendeklarasikan variabel S1 bertipe STRUKTUR1 */
  struct STRUKTUR1 S1;
  /* Mendeklarasikan variabel S2 bertipe STRUKTUR2 */
  struct STRUKTUR2 S2;

  /* Menampilkan ukuran dari S1 dan S2 */
  printf("sizeof(S1) = %d\n", sizeof(S1));
  printf("sizeof(S2) = %d\n", sizeof(S2));

  return 0;
}
```

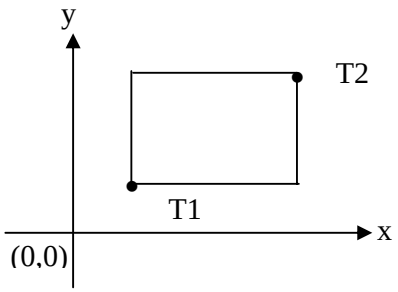
Hasil yang akan diberikan adalah sebagai berikut.

```
sizeof(S1) = 2
sizeof(S2) = 8
```

8.2.4. Mendefinisikan Struktur yang Berisi Struktur Lain

Seperti yang telah disinggung sebelumnya bahwa struktur adalah kumpulan variabel yang dapat terdiri dari berbagai tipe data, bahkan tipe struktur sekalipun. Dengan kata lain, bahasa C mengizinkan kita untuk membuat sebuah struktur dimana anggota atau field dari struktur tersebut juga bertipe struktur.

Sebagai contoh untuk menunjukkan hal ini, di sini kita akan membuat struktur dengan nama `SEGIEMPAT` dimana yang terdiri dari dua buah titik bertipe `TITIK`. Ingatlah bahwa sebuah segiempat dapat terbentuk hanya dengan menentukan dua titik yang berada di sudut kiri bawah dan sudut kanan atas saja. Perhatikan gambar di bawah ini.



Gambar 8.2. Titik T1 dan T2 dalam diagram kartesian

Adapun cara pendefinisiannya dapat Anda lihat pada sintak di bawah ini.

```
/* Mendefinisikan struktur TITIK */
struct TITIK {
      int x;
      int y;
};

/* Mendefinisikan struktur SEGIEMPAT */
struct SEGIEMPAT {
      struct TITIK T1;
      struct TITIK T2;
} S;
```

Sedangkan untuk mengisikan nilai ke dalam field yang terdapat pada struktur `S`, kita akan menuliskannya seperti berikut.

```
/* Mengisikan nilai ke dalam field ke-1 struktur S */
S.T1.x = 2
S.T1.y = 3

/* Mengisikan nilai ke dalam field ke-2 struktur S */
S.T2.x = 4
S.T2.y = 6
```

Berikut ini contoh program yang akan menentukan luas dari segiempat yang terbentuk dari dua buah titik di atas.

```
#include <stdio.h>

struct TITIK {
      int x;
      int y;
};
```

```
struct SEGIEMPAT {
  struct TITIK T1;
  struct TITIK T2;
};

int main(void) {

  struct SEGIEMPAT S;
  int panjang, lebar, luas;

  /* Mengisikan nilai ke struktur SEGIEMPAT */
  printf("Titik ke-1:\n");
  printf("Nilai x = "); scanf("%d", &S.T1.x);
  printf("Nilai y = "); scanf("%d", &S.T1.y);
  printf("\nTitik ke-2:\n");
  printf("Nilai x = "); scanf("%d", &S.T2.x);
  printf("Nilai y = "); scanf("%d", &S.T2.y);

  /* Menggunakan fungsi abs untuk mendapatkan nilai absolut */
  panjang = abs(S.T2.x - S.T1.x);
  lebar = abs(S.T2.y - S.T1.y);

  /* Menghitung luas segiempat */
  luas = panjang * lebar;

  /* Menampilkan hasil perhitungan */
  printf("\nLuas segi empat = %d", luas);
  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah seperti yang terlihat di bawah ini.

```
Titik ke-1:
Nilai x = 2
Nilai y = 3

Titik ke-2:
Nilai x = 6
Nilai y = 5

Luas segiempat = 8
```

8.3. Struktur sebagai Tipe Data Bentukan

Seperti yang telah disinggung sebelumnya bahwa struktur juga dapat digunakan sebagai tipe data bentukan, yaitu dengan cara menggunakan kata kunci `typedef`. Dengan demikian, kita tidak perlu lagi untuk menuliskan kata kunci `struct` dalam

mendeklarasikan suatu struktur, sama halnya seperti pendeklarasian variabel dengan tipe data dasar.

Untuk lebih memahaminya, perhatikan sintak program berikut di bawah ini dimana kita akan mendefinisikan struktur sebagai tipe data bentukan.

```
#include <stdio.h>
#include <string.h>   /* untuk menggunakan strcpy() */

typedef struct {
  char NIM[8];
  char nama[25];
  char alamat[40];
  int usia;
} SISWA;

int main(void) {

  /* Mendeklarasikan variabel S dengan tipe SISWA */
  SISWA S;

  /* Mengisikan nilai ke dalam S */
  strcpy(S.NIM, “D0D98021”);
  strcpy(S.nama, “Arista Destriana”);
  strcpy(S.alamat, “Jl. Kopo 46, Bandung”);
  S.usia = 23;

  /* Menampilkan nilai yang dikandung di dalam S */
  printf(“NIM\t: %.8s\n”, S.NIM);
  printf(“Nama\t: %s\n”, S.nama);
  printf(“Alamat\t: %s\n”, S.alamat);
  printf(“Nilai\t: %d\n”, S.nilai);

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
NIM       : D0D98021
Nama      : Arista Destriana
Alamat    : Jl. Kopo 46, Bandung
Usia      : 23
```

8.4. Struktur dan Fungsi

Dalam bahasa C, kita diizinkan untuk menggunakan struktur sebagai parameter dan juga sebagai nilai kembalian dari sebuah fungsi. Dua hal ini akan kita bahas secara tersendiri dalam sub bab di bawah ini.

8.4.1. Struktur sebagai Parameter Fungsi

Pada bab sebelumnya kita telah banyak membahas tentang bagaimana cara melewatkan parameter atau argumen ke dalam sebuah fungsi. Parameter dari fungsi tersebut tentunya dapat bertipe apa saja, baik tipe data dasar maupun tipe bentukan. Di sini, kita akan membahas bagaimana cara untuk melewatkan parameter yang bertipe struktur ke dalam sebuah fungsi.

Sebagai contoh, perhatikan kembali struktur `TITIK` di atas. Di sini kita akan membuat program yang dapat menentukan titik tengah dari dua buah koordinat, yaitu dengan cara melewatkan kedua koordinat tersebut sebagai parameter ke dalam sebuah fungsi. Berikut ini sintak yang dimaskudkan.

```
#include <stdio.h>

/* Membuat tipe bentukan bertipe struktur dengan nama TITIK */
typedef struct {
  int x;
  int y;
} TITIK;

/* Membuat fungsi untuk menentukan titik tengah */
void TitikTengah(TITIK T1, TITIK T2, TITIK *TT) {
  (*TT).x = (T1.x + T2.x) / 2;
  (*TT).y = (T1.y + T2.y) / 2;
}

int main(void) {
  /* Mendeklarasikan variabel-variabel yang dibutuhkan */
  TITIK A1, A2, titik_tengah;

  /* Memasukan nilai ke dalam A1 dan A2 */
  printf("Titik ke-1:\n");
  printf("Nilai x = "); scanf("%d", &A1.x);
  printf("Nilai y = "); scanf("%d", &A1.y);
  printf("\nTitik ke-2:\n");
  printf("Nilai x = "); scanf("%d", &A2.x);
  printf("Nilai y = "); scanf("%d", &A2.y);

  /* Memanggil fungsi TitikTengah() */
  TitikTengah(A1, A2, &titik_tengah);

  /* Menampilkan hasil */
  printf("\nTitik tengah berada pada koordinat (%d,%d)",
         titik_tengah.x, titik_tengah.y);

  return 0;
}
```

Berikut ini contoh hasil yang akan diberikan oleh program di atas.

```
Titik ke-1:
Nilai x = 2
Nilai y = 3

Titik ke-2:
Nilai x = 6
Nilai y = 3

Titik tengah berada pada koordinat (4,3)
```

Pada sintak di atas kita membuat fungsi dengan nama `TitikTengah()` yang memiliki tiga buah parameter bertipe `TITIK`. Parameter pertama dan kedua digunakan untuk menyimpan koordinat-koordinat yang akan dihitung, sedangkan parameter ketiga adalah parameter yang dilewatkan berdasarkan alamatnya dan berguna untuk menampung nilai hasil perhitungan.

Hal yang perlu Anda perhatikan dari sintak di atas adalah pada saat kita mengakses field dari pointer yang menunjuk ke struktur `TT`, yaitu sebagai berikut.

```
(*TT).x dan (*TT).y
```

Di sini, Anda harus menyertakan tanda kurung. Hal ini disebabkan karena operator titik (.) prioritasnya lebih tinggi daripada operator `*`. Dengan demikian apabila kita menuliskannya seperti berikut:

```
*TT.x dan *TT.y
```

maka kompiler akan menganggapnya sebagai pointer ke `TT.x` dan ke `TT.y` (bukan pointer ke struktur `TT`). Sintak tersebut sama dengan sintak berikut.

```
*(TT.x) dan *(TT.y)
```

Hal ini tentu berbeda dengan apa yang kita inginkan sebelumnya dan akan menyebabkan hasil yang salah.

8.4.2. Struktur sebagai Nilai Kembalian Fungsi

Selain sebagai parameter, struktur juga dapat digunakan untuk menyimpan nilai kembalian yang terdapat di dalam fungsi. Berikut ini contoh program yang akan

menunjukkan hal tersebut dimana kita akan melewatkan dua buah variabel yang bertipe `int` dan memasukkannya ke dalam struktur `TITIK`. Adapun sintaknya adalah sebagai berikut.

```
#include <stdio.h>

/* Membuat tipe data bentukan bertipe struktur dengan nama TITIK
*/
typedef struct {
      int x;
      int y;
} TITIK;

/* Membuat fungsi untuk memasukkan nilai ke struktur TITIK */
TITIK BuatTitik(int a, int b) {
  TITIK temp; /* membuat variabel temporari
                 yang bertipe TITIK */
  /* Memasukkan nilai a ke dalam x dan b ke dalam y */
  temp.x = a;
  temp.y = b;

  return temp;  /* nilai kembalian bertipe struktur */
};

int main(void) {
  TITIK A1, A2, A3;

  /* Memasukkan nilai ke dalam A1, A2 dan A3 */
  A1 = BuatTitik(3,2);
  A2 = BuatTitik(5,4);
  A3 = BuatTitik(7,2);

  /* Menampilkan nilai yang terdapat pada A1, A2 dan A3 */
  printf("A1(%d,%d)\n", A1.x, A1.y);
  printf("A2(%d,%d)\n", A2.x, A2.y);
  printf("A3(%d,%d)\n", A3.x, A3.y);

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
A1(3,2)
A2(5,4)
A3(7,2)
```

8.5. Struktur dan Array

Pada umumnya para programmer baru merasa bingung dengan istilah struktur dari array (*structure of array*) dan array dari struktur (*array of structure*). Untuk itu, pada bagian ini kita akan membahas perbedaan antara keduanya.

8.5.1. Struktur dari Array

Struktur dari array adalah suatu struktur yang salah satu atau semua anggotanya bertipe array. Sebagai contoh, perhatikan definisi struktur berikut.

```
struct SISWA {
  char NIM[8];
  char nama[25];
  float nilai[2];
  char nilai_indeks;
};
```

Pada sintak di atas terlihat bahwa struktur `SISWA` terdiri dari empat buah anggota. Anggota pertama dan kedua merupakan anggota yang bertipe array dari karakter (string). Anggota ketiga merupakan array dari tipe `float` yang berguna untuk menampung dua buah nilai, yaitu nilai UTS dan UAS. Sedangkan anggota terakhir bertipe char yang akan digunakan untuk menampung nilai indeks (A, B, C, D atau E). Untuk lebih memahaminya, berikut ini implementasi dari struktur di atas ke dalam sebuah program.

```
#include <stdio.h>

typedef struct {
  char NIM[8];
  char nama[25];
  float nilai[2];
  char nilai_indeks;
} SISWA;

/* Membuat fungsi untuk menentukan nilai indeks */
char TentukanIndeks(float nilai) {
  char indeks;
  if (nilai > 80) {
    indeks = 'A';
  } else {
    if ((nilai > 70) && (nilai <= 80)) {
      indeks = 'B';
    } else {
      if ((nilai > 60) && (nilai <= 70)) {
        indeks = 'C';
      } else {
        if ((nilai > 50) && (nilai <= 60)) {
          indeks = 'D';
```

```
        } else {
          indeks = 'E';
        }
      }
    }
  }
  return indeks;
}

int main(void) {

  /* Mendeklarasikan variabel S yang bertipe SISWA */
  SISWA S;
  float nilai;  /* Variabel untuk menampung nilai akhir */

  /* Memasukkan nilai ke dalam S */
  printf("INPUT:\n");
  printf("NIM \t: "); scanf("%s", &S.NIM);
  printf("Nama \t: "); scanf("%s", &S.nama);
  printf("Nilai\n");
  printf("  UTS \t: "); scanf("%f", &S.nilai[0]);
  printf("  UAS \t: "); scanf("%f", &S.nilai[1]);

  /* Menampilkan hasil */
  printf("\nOUTPUT:\n");
  printf("NIM \t: %.8s\n", S.NIM);
  printf("Nama \t: %.25s\n", S.nama);

  /* Menghitung nilai akhir */
  nilai = (0.4 * S.nilai[0]) + (0.6 * S.nilai[1]);

  /* Memanggil fungsi TentukanIndeks() */
  S.nilai_indeks = TentukanIndeks(nilai);

  printf("Indeks \t: %c\n", S.nilai_indeks);

  return 0;
}
```

Adapun contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
INPUT:
NIM    : D0D99021
Nama   : Mira
Nilai
  UTS  : 75
  UAS  : 80

OUTPUT:
NIM    : D0D99021
Nama   : Mira
```

```
Indeks : B
```

8.5.2. Array dari Struktur

Setelah mengetahui struktur yang berisi array, di sini kita akan mempelajari array dari struktur. Konsepnya sederhana, array dari struktur merupakan array yang setiap elemennya bertipe struktur, sama halnya seperti tipe dasar yang lain. Dalam pemrograman kita sering dihadapkan dengan masalah-masalah seperti ini. Misalnya untuk mencatat data sekumpulan siswa dalam satu kelas, para nasabah bank maupun contoh-contoh lainnya yang serupa.

Apabila kita memiliki sebuah struktur (misalnya dengan nama `SISWA`), yang didefinisikan sebagai berikut.

```
struct SISWA {
  char NIM[8];
  char nama[25];
  int nilai[2];
  char nilai_indeks;
};
```

Maka kita dapat mendeklarasikan array yang terdiri dari 100 elemen (misalnya dengan nama `S`) dengan menuliskannya sebagai berikut.

```
struct SISWA S[100];
```

Namun apabila struktur tersebut sudah dijadikan tipe data bentukan, maka kita cukup menuliskannya seperti berikut.

```
SISWA S[100];
```

Untuk lebih mudah dalam memahaminya, berikut ini dituliskan contoh program yang akan menunjukkan penggunaan array dari struktur.

```
#include <stdio.h>

/* Membuat konstanta untuk menentukan banyak elemen maksimal */
#define MAX 100

/* Membuat tipe data bentukan bertipe struktur
   dengan nama SISWA */
typedef struct {
  char NIM[8];
  char nama[25];
  float nilai[2];
```

```
  char nilai_indeks;
} SISWA;

/* Membuat fungsi untuk menentukan nilai indeks */
char TentukanIndeks(float nilai) {
  char indeks;
  if (nilai > 80) {
    indeks = 'A';
  } else {
    if ((nilai > 70) && (nilai <= 80)) {
      indeks = 'B';
    } else {
      if ((nilai > 60) && (nilai <= 70)) {
        indeks = 'C';
      } else {
        if ((nilai > 50) && (nilai <= 60)) {
          indeks = 'D';
        } else {
          indeks = 'E';
        }
      }
    }
  }
  return indeks;
}

int main(void) {
  /* Mendeklarasikan array S yang elemennya bertipe SISWA */
  SISWA S[MAX];
  int N;  /* Variabel untuk menampung jumlah siswa */
  float nilai;  /* Variabel untuk menampung nilai akhir */
  int j;        /* Variabel untuk indeks pengulangan */

  /* Memasukkan banyaknya siswa yang ada */
  printf("Banyaknya siswa : "); scanf("%d", &N);

  /* Memasukkan nilai ke dalam S */
  for(j=0; j<N; j++) {
    printf("\nSiswa ke-%d:\n", j+1);
    printf("NIM \t: "); scanf("%s", &S[j].NIM);
    printf("Nama \t: "); scanf("%s", &S[j].nama);
    printf("Nilai:\n");
    printf("  UTS \t: "); scanf("%f", &S[j].nilai[0]);
    printf("  UAS \t: "); scanf("%f", &S[j].nilai[1]);
  }
  /* Menampilkan hasil */
  printf("\nOutput:\n");
  for(j=0; j<N; j++) {
    /* Menghitung nilai akhir */
    nilai = (0.4 * S[j].nilai[0]) + (0.6 * S[j].nilai[1]);
    /* Memanggil fungsi Tentukan Indeks */
    S[j].nilai_indeks = TentukanIndeks(nilai);
    printf("%d. NIM : %.8s\t Nama : %.25s\t Indeks : %c\n",
           j+1,
           S[j].NIM,
```

```
            S[j].nama,
            S[j].nilai_indeks
      );
    }
    return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah seperti terlihat di bawah ini.

```
Banyaknya siswa : 5

Siswa ke-1:
NIM    : DXX99001
Nama   : MIRA
Nilai:
  UTS  : 90
  UAS  : 95

Siswa ke-2:
NIM    : DXX99002
Nama   : HERI
Nilai:
  UTS  : 85
  UAS  : 95

Siswa ke-3:
NIM    : DXX99003
Nama   : PUJI
Nilai:
  UTS  : 60
  UAS  : 60

Siswa ke-4:
NIM    : DXX99004
Nama   : RANDY
Nilai:
  UTS  : 50
  UAS  : 55

Siswa ke-5:
NIM    : DXX99005
Nama   : ALEX
Nilai:
  UTS  : 70
  UAS  : 60

1. NIM : DXX001      Nama : MIRA          Indeks : A
```

```
2. NIM : DXX002     Nama : HERI       Indeks : A
3. NIM : DXX003     Nama : PUJI       Indeks : D
4. NIM : DXX004     Nama : RANDY      Indeks : D
5. NIM : DXX005     Nama : ALEX       Indeks : C
```

8.6. Struktur dan Pointer

Pointer merupakan fitur andalan yang terdapat di dalam bahasa C, kehadirannya dapat ada di mana-mana. Kita dapat mendeklarasikan pointer sebagai anggota dari struktur dan juga dapat mendeklarasikan pointer ke tipe struktur. Anda tidak perlu bingung akan perbedaan antara keduanya karena topik ini akan kita bahas secara terpisah dalam sub bab berikut.

8.6.1. Struktur yang Berisi Pointer

Kita dapat melengkapi fleksibilitas pointer dengan menempatkannya sebagai anggota dari suatu struktur. Pendeklarasian pointer sebagai anggota struktur tidaklah berbeda dengan pendeklarasian pointer yang telah diterangkan pada bab sebelumnya, yaitu dengan menempatkan operator unary `*` di depan nama pointer yang ingin dideklarasikan. Berikut ini merupakan contoh pendefinisian struktur yang di dalamnya mengandung anggota yang bertipe pointer.

```
struct TITIK {
  int *x;  /* x adalah anggota yang merupakan pointer
              ke tipe int */
  int *y;  /* y adalah anggota yang merupakan pointer
              ke tipe int */
} T;
```

Sekarang apabila kita memiliki variabel `a` dan `b` yang keduanya bertipe `int`, maka kita dapat memasukkan alamat memori dari variabel `a` dan `b` ke dalam pointer `x` dan `y` yang terdapat pada struktur `T`, dengan cara seperti di bawah ini.

```
T.x = &a;       /* T.x menunjuk ke alamat dari variabel a */
T.y = &b;       /* T.y menunjuk ke alamat dari variabel b */
```

Tampak di atas bahwa `T.x` dan `T.y` berisi alamat (bukan nilai). Dengan demikian berarti untuk mengambil nilai-nilai yang terdapat pada struktur `T` di atas kita akan menuliskannya seperti berikut.

```
*(T.x)     /* akan bernilai sama dengan nilai dari variabel a */
*(T.y)     /* akan bernilai sama dengan nilai dari variabel b */
```

Untuk dapat lebih memahaminya, coba Anda perhatikan contoh program berikut ini dimana di dalamnya terdapat pendefinisian struktur yang anggotanya bertipe pointer.

```
#include <stdio.h>

struct INFO_BUKU {
  char *judul;        /* pointer ke tipe char */
  char *pengarang;    /* pointer ke tipe char */
  char *penerbit;     /* pointer ke tipe char */
  long *tahun;        /* pointer ke tipe long */
} IB;

int main(void) {

  long thn = 2006;

  /* Memasukkan nilai ke dalam IB */
  IB.judul = "Pemrograman Menggunakan Bahasa C";
  IB.pengarang = "Budi Raharjo & I Made Joni";
  IB.penerbit = "INFORMATIKA Bandung";
  IB.tahun = (long *) malloc(sizeof(long));
  IB.tahun = &thn;

  /* Menampilkan isi yang terdapata pada struktur IB */
  printf("Judul \t\t: %s\n", IB.judul);
  printf("Pengarang \t: %s\n", IB.pengarang);
  printf("Penerbit \t: %s\n", IB.penerbit);
  printf("Tahun \t\t: %ld", *(IB.tahun));

  free(IB.tahun);

  return 0;
}
```

Hasil yang akan didapatkan dari program di atas adalah seperti yang terlihat di bawah ini.

```
Judul       : Pemrograman Menggunakan Bahasa C
Pengarang   : Budi Raharjo & I Made Joni
Penerbit    : INFORMATIKA Bandung
Tahun       : 2006
```

8.6.2. Pointer ke Struktur

Pada sub bab sebelumnya kita telah mempelajari bagaimana cara mendeklarasikan pointer di dalam sebuah struktur. Kali ini kita akan membahas materi yang masih terkait dengan pointer dan struktur, yaitu cara mendeklarasikan pointer yang akan menunjuk ke tipe struktur. Berikut ini potongan sintak yang akan menunjukkan cara pendeklarasian pointer ke tipe struktur.

```
/* Mendefinisikan struktur dengan nama SISWA */
struct SISWA {
  char NIM[8];
  char nama[25];
  char alamat[40];
};

/* Mendeklarasikan pointer P ke struktur SISWA */
struct SISWA *P;
```

Sebenarnya konsep dari pointer ke struktur sama saja dengan pointer yang menunjuk ke tipe-tipe data lainnya. Walaupun demikian, terdapat sedikit perbedaan yang perlu sekali untuk diperhatikan, yaitu operator yang digunakan pada saat pengaksesan nilai dari anggota-anggota yang terkandung dalam struktur.

Apabila kita memiliki variabel yang bertipe struktur, maka untuk mengakses nilai dari anggota-anggota yang terdapat pada struktur tersebut kita akan menggunakan operator titik (.). Namun apabila kita mendeklarasikan pointer ke struktur, maka operator yang digunakan untuk mengakses nilai dari anggota struktur tersebut adalah operator `->`. Berikut ini contoh sintak yang akan menunjukkan perbedaan antara keduanya.

```
/* Mendeklarasikan variabel S bertipe struktur */
struct SISWA S;

/* Untuk mengakses anggota S digunakan operator titik (.) */
S.NIM = “D0D99021”;
S.nama = “Arista Destriana”;
...

/* Mendeklarasikan pointer P ke tipe struktur */
struct SISWA *P;

/* Alokasi memori */
P = (SISWA*) malloc(sizeof(SISWA));

/* Memerintahkan P untuk menunjuk ke alamat S */
P = &S;

/* Untuk mengakses anggota dengan pointer digunakan
   operator -> */
printf(“NIM \t: %s\n”, P->NIM);
printf(“Nama \t: %s\n”, P->nama);
...

/* Mendealokasikan memori */
free(P);
```

Untuk membuktikan hal tersebut, Anda dapat menuliskan program lengkap di bawah ini.

```
#include <stdio.h>
#include <string.h>

typedef struct {
  char NIM[8];
  char nama[25];
  char alamat[40];
} SISWA;

int main(void) {
  SISWA S, *P;

  P = (SISWA *) malloc(sizeof(SISWA));
  P = &S;

  strcpy(S.NIM, “D0D99021”);
  strcpy(S.nama, “Arista Destriana”);
  strcpy(S.alamat, “Jl. Kopo 46, Bandung”);

  printf(“NIM \t: %.8s\n”, P->NIM);
  printf(“Nama \t: %.25s\n”, P->nama);
  printf(“Alamat \t: %.45s\n”, P->alamat);

  free(P);
  return 0;
}
```

Hasil dari program di atas adalah sebagai berikut.

```
NIM    : D0D99021
Nama   : Arista Destriana
Alamat: Jl. Kopo 46, Bandung
```

8.7. Union

Union sebenarnya adalah suatu struktur dan cara kerjanya pun sama dengan sebuah struktur. Namun perbedaannya hanya terletak pada ruang memori yang dialokasikan untuk setiap field-nya. Pada struktur setiap field-nya mempunyai ruang memori tersendiri sedangkan pada union semua field-nya mengacu ke satu buah alamat yang sama. Begitu juga dengan ukuran memorinya, pada struktur setiap field memiliki ukuran memori tersendiri sedangkan pada union ukuran yang ada akan dipakai oleh semua field yang terdapat di dalamnya, tentunya secara bergantian. Untuk mendefinisikan suatu union, kita harus menambahkan kata kunci `union` di depan nama union yang akan dedefinisikan tersebut. Berikut ini bentuk umumnya.

```
union nama_struktur {
  tipe_data field1;
  tipe_data field2;
  ...
} var1, var2, ...;    /* ingat tanda titik koma */
```

Agar Anda dapat lebih memahami perbedaan yang terdapat antara union dan struktur, coba Anda perhatikan contoh program berikut dimana akan ditunjukkan bahwa setiap field dari union akan mengacu ke alamat yang sama.

```
#include <stdio.h>

/* Mendefinisikan sebuah struktur */
struct STRUKTUR {
  int x;
  char y;
} S;
/* Mendefinisikan sebuah union */
union UNION {
  int x;
  char y;
} U;

/* Fungsi utama */
int main(void) {

  /* Menampilkan alamat dari field x dan y yang terdapat dalam
     struktur S */
  printf("Pada Struktur S:\n");
  printf("Alamat x \t\t: %p\n", &(S.x));
  printf("Alamat y \t\t: %p\n", &(S.y));
  printf("Ukuran struktur \t: %d byte\n", sizeof(S));

  /* Menampilkan alamat dari field x dan y yang terdapat dalam
     union U */
  printf("Pada Union U:\n");
  printf("Alamat x \t\t: %p\n", &(U.x));
  printf("Alamat y \t\t: %p\n", &(U.y));
  printf("Ukuran union \t\t: %d byte\n", sizeof(U));

  return 0;
}
```

Hasil yang akan diperoleh dari program di atas adalah sebagai berikut.

```
Pada Struktur S:
```

```
Alamat x          : 00405050
Alamat y          : 00405054
Ukuran struktur   : 8 byte

Pada Union U:
Alamat x          : 00405040
Alamat y          : 00405040
Ukuran struktur   : 4 byte
```

Dari hasil tersebut dapat Anda lihat bahwa field yang terdapat pada struktur (dalam hal ini `x` dan `y`) menempati alamat yang berbeda. Dengan kata lain masing-masing field tersebut memiliki alamat tersendiri. Sedangkan pada union, field `x` dan `y` menempati satu alamat yang sama. Begitu juga dengan ukuran memori yang dibutuhkan untuk pendefinisiannya. Untuk menyimpan field `x` (bertipe `int`) dan `y` (bertipe `char`), sebuah struktur memerlukan ruang 8 byte, sedangkan union hanya membutuhkan 4 byte.

Bab

9

Linked List

9.1. Pendahuluan

Dalam dunia komputer, linked list secara ekstensif banyak digunakan di dalam sistem manajemen database, manajemen proses, sistem operasi, editor dan lain sebagainya. Mungkin buku lain ada yang meng-istilah-kan linked list ini dengan istilah ‘senarai berantai’, ‘daftar berantai’ ataupun yang lainnya. Namun, yang jelas di sini kita tidak akan mempermasalahkan hal tersebut, karena itu hanya sebuah penamaan saja. Sebaliknya, yang perlu Anda pahami dalam bab ini adalah bagaimana cara menjalin keterkaitan antar struktur (yang sejenis) yang dirangkai dengan menggunakan bantuan pointer.

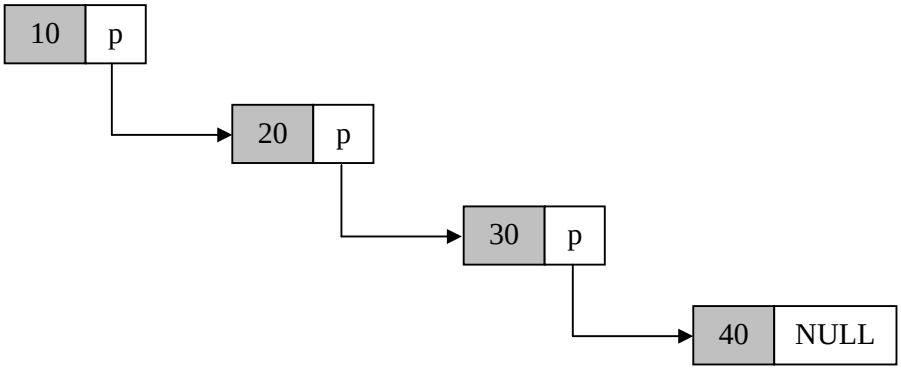
Apabila Anda menggunakan array untuk menempatkan beberapa buah struktur sejenis (*array of structure*) di memori, maka Anda akan mengalokasikannya dengan cara yang statis. Artinya apabila Anda mendeklarasikan array tersebut dengan 10 buah elemen, maka ukuran memori yang akan dialokasikan di memori adalah 10 dikalikan ukuran struktur yang didefinisikan sebelumnya. Hal yang perlu ditekankan di sini adalah apabila ternyata Anda hanya melakukan pengisian terhadap array tersebut sebanyak 5 elemen, maka memori yang dialokasikan tetap untuk 10 buah elemen. Ini jelas merupakan suatu pemborosan memori yang harus kita hindari dalam pembuatan program. Oleh karena itu, untuk membuat pekerjaan ini menjadi dinamis, para programmer pada umumnya menggunakan linked list, sehingga pengalokasian memori dapat sesuai dengan kebutuhan. Selain itu, dengan menggunakan cara ini kita juga dapat menambahkan atau menghapus elemen-elemen yang terdapat linked list tersebut.

Materi ini memang merupakan materi yang cukup kompleks dan membutuhkan bidang kajian tersendiri untuk memahaminya secara lebih mendalam. Namun di sini, penulis merasa perlu untuk menyampaikan konsep dasar dan cara kerja yang terdapat di dalam linked list, khususnya bentuk pengimplementasiannya di dalam bahasa C.

Menurut definisinya, linked list adalah sebuah rangkaian struktur sejenis (bertipe sama) yang dihubungkan dengan menggunakan salah satu (beberapa) field yang bertipe pointer. Untuk dapat lebih memahaminya, perhatikan pendefinisian struktur di bawah ini.

```
struct node {
  int data;
  struct node *p;
};
```

Di atas kita memiliki sebuah struktur dengan nama `node`, dimana di dalamnya terdapat dua buah field, yaitu *`data`* (bertipe `int`) dan *`p`* (bertipe pointer ke struktur `node`). Field *`data`* akan digunakan untuk menyimpan nilai, sedangkan pointer *`p`* akan digunakan untuk menyimpan alamat dari struktur `node` lainnya yang terdapat dalam rangkaian. Berikut ini contoh gambar yang akan mengilustrasikan kasus tersebut.



Gambar 9.1. Rangkaian linked list

Pada gambar di atas kita memiliki 4 buah struktur `node`, yang masing-masing menyimpan `data` 10, 20, 30 dan 40. Pointer `p` yang terdapat pada struktur `node` ke-1 berisi alamat dari struktur `node` ke-2, `p` yang terdapat pada struktur `node` ke-2 akan berisi alamat dari struktur `node` yang ke-3. Begitupun pointer `p` yang terdapat pada struktur `node` ke-3, dia akan menunjuk atau berisikan alamat dari struktur `node` ke-4. Sedangkan pointer `p` yang terdapat di dalam struktur `node` ke-4 akan berisi NULL, hal ini menunjukkan bahwa struktur tersebut merupakan struktur `node` yang berada pada akhir rangkaian. Hal semacam inilah yang dinamakan dengan struktur terkait atau yang lebih dikenal dengan sebutan linked list.

Menurut cara pengisian dan pengambilan elemennya, linked list dibedakan menjadi dua macam, yaitu *stack* (tumpukan) dan *queue* (antrian). Adapun penjelasan selengkapnya dari kedua topik tersebut dapat Anda lihat pada sub bab di bawah ini.

9.2. *Stack* (Tumpukan)

Stack adalah jenis linked list yang menerapkan konsep LIFO (*Last-In-Fist-Out*), artinya elemen struktur yang dimasukkan ke dalam rangkaian terakhir kali apabila ditampilkan kembali maka akan muncul pertama kali. Untuk memahami materi ini, coba Anda bayangkan sebuah tumpukan piring yang telah selesai dicuci. Di sini piring yang diletakkan pertama kali ke dalam tumpukan tersebut akan diambil terakhir kali, sedangkan piring yang diletakkan terakhir kali justru akan diambil pertama kali. Cara kerja seperti inilah yang menyebabkan linked list ini dinamakan dengan *stack*

(tumpukan). Sebagai contoh dari cara kerja stack adalah apabila kita memasukkan data secara berurutan seperti di bawah ini.

10
20
30
40

Apabila data tersebut ditampilkan kembali, maka hasil yang akan diberikan adalah sebagai berikut.

40
30
20
10

Untuk membuktikan hal tersebut perhatikan contoh program berikut ini.

```
#include <stdio.h>
#include <stdlib.h>

/* Mendefinisikan struktur node */
struct node {
  int data;
  struct node *p;
};

/* Mendefinisikan fungsi untuk meletakkan atau mengisikan
   struktur node ke dalam stack */
void push(struct node **s, int nilai) {
  struct node *temp;
  temp = (struct node*) malloc (sizeof(struct node));
  temp->data = nilai;     /* Mengisikan field data pada
                             struktur node */
  temp->p = *s;           /* Mengisikan field p pada
                             struktur node */
  *s = temp;
}

/* Mendefinisikan fungsi untuk mengambil struktur node dari
dalam stack */
void pop(struct node **s) {
  struct node *temp;
  int nilai;
  if (*s != NULL) {
    temp = *s;
    nilai = temp->data;
    *s = temp->p;
    free(temp);
    return nilai;
  } else {
    return 1;   /* Apabila stack kosong maka program
```

```
                    dihentikan */
      }
}

int main(void) {
  struct node *paling_atas; /* pointer ini akan selalu menunjuk
                               ke elemen paling atas */

  /* Melakukan inisialisasi terhadap pointer paling_atas dengan
     nilai NULL */
  paling_atas = NULL;       /* Mula-mula stack masih dalam
                               keadaan kosong */

  /* Mengisikan struktur node ke dalam stack menggunakan
     fungsi push() */
  push(&paling_atas, 10);
  push(&paling_atas, 20);
  push(&paling_atas, 30);
  push(&paling_atas, 40);

  /* Menampilkan struktur node yang terdapat di dalam stack
     menggunakan fungsi pop() */
  while (paling_atas != NULL) {
    printf("%d\n", pop(&paling_atas));
    paling_atas = paling_atas->p;    /* menunjuk ke elemen
                                        di bawahnya */
  }

  return 0;
}
```

Hasil yang akan diberikan oleh program di atas adalah sebagai berikut.

```
40
30
20
10
```

Coba Anda perhatikan kembali fungsi `push()` pada sintak program di atas.

```
void push(struct node **s, int nilai) {
  struct node *temp;
  temp = (struct node*) malloc (sizeof(struct node));
  temp->data = nilai;     /* Mengisikan field data pada
                             struktur node */
  temp->p = *s;           /* Mengisikan field p pada
                             struktur node */
  *s = temp;
}
```

Di sini kita melewatkan dua buah parameter, yaitu pointer `*s` (pointer ke struktur `node`) dan `nilai`. Namun dalam fungsi ini, kita akan melewatkan pointer `*s` tersebut berdasarkan alamatnya (*passsing by reference*) sehingga kita akan menuliskannya ke dalam parameter dengan sintak berikut.

```
struct node **s
```

Dengan kata lain `s` merupakan pointer ke pointer yang sedang menunjuk ke tipe struktur `node`. Apabila Anda masih merasa bingung dengan keadaan ini, cobalah untuk melihat kembali bab sebelumnya yang menerangkan tentang pointer ke pointer dan juga tentang pengiriman parameter berdasarkan alamat.

Dalam kasus ini, pointer `*s` akan digunakan untuk menyimpan alamat dari struktur `node` yang akan dimasukkan ke dalam stack. Sebagai bantuan untuk memudahkan proses dalam fungsi ini, kita mendeklarasikan pointer `temp`. Selanjutnya pointer `temp` tersebut digunakan untuk mencatat alamat dari struktur yang dialokasikan menggunakan fungsi `malloc()`. Field `data` dari pointer `temp` akan diisi dengan parameter *`nilai`* sedangkan field `p` diisi dengan parameter *`*s`*. Kemudian pointer `*s` itu sendiri sekarang akan diisi dengan pointer `temp`.

Oleh karena pointer `*s` dilewatkan berdasarkan alamat, maka pada saat pemanggilan pun kita harus mendefinisikannya dengan sebuah alamat dari pointer ke tipe struktur `node`. Pada program di atas, kita memiliki pointer `paling_atas` yang menunjuk ke tipe struktur `node`, sehingga kita dapat melakukan pemanggilan fungsi `push()` tersebut dengan cara di bawah ini.

```
push(&paling_atas, 10);
```

Ini artinya mula-mula kita akan menyimpan alamat yang ditunjuk oleh pointer `paling_atas` ke dalam field `p` dari struktur baru yang dialokasikan. Sedangkan field `data` dari struktur baru tersebut diisi dengan nilai 10. Setelah itu pointer `paling_atas` sekarang diperintahkan untuk menunjuk ke alamat lain, yaitu alamat dari struktur yang baru saja dialokasikan. Apabila ditambahkan elemen struktur baru lagi ke dalam stack, maka pointer `paling_atas` juga akan berpindah menunjuk ke alamat dari struktur yang ditambahkan, begitu seterusnya.

9.3. *Queue* (Antrian)

Queue merupakan kebalikan dari stack, yaitu jenis linked list yang menerapkan konsep FIFO (*First-In-First-Out*). Artinya elemen yang dimasukkan pertama kali apabila ditampilkan akan muncul pertama kali juga. Dengan kata lain, urutan dari keluaran (*output*) yang dihasilkan akan sesuai dengan urutan masukan (*input*) yang dilakukan. Untuk mempermudah pembahasan, coba Anda bayangkan sebuah antrian yang ada di sebuah loket, di situ orang yang datang pertama kali untuk mengantri akan dilayani terlebih dahulu dan yang datang terakhir juga akan dilayani terakhir kali.

Berikut ini contoh program yang mengimplementasikan sebuah pembentukan queue. Adapun sintak programnya adalah seperti berikut.

```
#include <stdio.h>
#include <stdlib.h>

/* Mendefinisikan struktur yang akan dirangkai sebagai queue */
struct node {
  int data;
  struct node *p;
};

/* Mendefinisikan fungsi untuk menambahkan elemen ke dalam queue
   serta menempatkannya pada bagian akhir queue */
void TambahElemen(struct node **d, struct node **b, int nilai) {
  struct node *temp;

  /* Membuat struktur node baru */
  temp = (struct node*) malloc (sizeof(struct node));

  /* Mengisikan field yang terdapat pada strukrur node yang baru
     dialokasikan */
  temp->data = nilai;   /* Mengeset field data dengan
                           parameter nilai */
  temp->p = NULL;       /* Mengeset field p dengan nilai NULL */

  /* Apabila queue kosong */
  if (*d == NULL) {
    *d = temp;          /* Mengeset pointer *d untuk
                           menunjuk struktur baru */
  } else {      /* Apabila queue sudah ada isinya */
    /* Mengeset field p dari elemen terakhir untuk menunjuk
       struktur baru */
    (*b)->p = temp;
  }

  *b = temp;          /* Mengeset pointer *b untuk menunjuk ke
                         struktur baru */
}

/* Mendefinisikan fungsi untuk menghapus elemen yang terdapat
   pada bagian depan dari queue */
void HapusElemen(struct node **d, struct node **b) {
  struct node *temp;

  /* Apabila queue kosong */
  if (*d == NULL) {
    printf("Tidak terdapat elemen dalam queue");
  } else { /* Apabila queue sudah ada isinya */
    /* Hapus elemen terdepan dan ambil nilainya */
    temp = *d;
    *d = temp->p;     /* Mengeset pointer *d untuk menunjuk ke
                         elemen di belakangnya */
    free(temp);       /* Menghapus elemen terdepan dari queue */
```

```
    /* Apabila penghapusan menyebabkan queue menjadi kosong */
    if (*d == NULL) {
      *b = NULL;
    }
  }
}

/* Mendefinisikan fungsi untuk menampilkan nilai-nilai
   di dalam queue */
void TampilkanNilai(struct node *d) {
  int nilai;
  /* Selama pointer d masih menunjuk ke alamat
     dari elemen tertentu */
  while (d != NULL) {
    nilai = d->data;
    printf("%d\n", nilai);
    d = d->p;   /* Pointer d menunjuk ke alamat dari
                   elemen selanjutnya */
  }
}

/* Fungsi utama */
int main(void) {

  /* Mendeklarasikan pointer untuk menunjuk elemen paling depan
     dan paling belakang */
  struct node *depan, *belakang;

  /* Mula-mula queue kosong, maka kita mengeset pointer depan
     dan belakang dengan nilai NULL */
  depan = belakang = NULL;

  /* Menambahkan elemen ke dalam queue */
  TambahElemen(&depan, &belakang, 10);
  TambahElemen(&depan, &belakang, 20);
  TambahElemen(&depan, &belakang, 30);
  TambahElemen(&depan, &belakang, 40);

  /* Menampilkan nilai-nilai yang terdapat di dalam queue */
  TampilkanNilai(depan);

  /* Menghapus satu elemen pertama dari queue */
  printf("Nilai-nilai yang terdapat di dalam queue:\n");
  HapusElemen(&depan, &belakang);

  /* Menampilkan kembali nilai-nilai yang terdapat
     di dalam queue */
  printf("\nNilai-nilai di dalam queue " \
         "setelah penghapusan elemen pertama:\n");
  TampilkanNilai(depan);

  return 0;
}
```

Hasil yang akan diperoleh dari program di atas adalah sebagai berikut.

```
Nilai-nilai yang terdapat di dalam queue:
10
20
30
40

Nilai-nilai di dalam queue setelah penghapusan elemen pertama:
20
30
40
```

Sekarang mari kita teliti jalannya program di atas. Mula-mula tidak memiliki satu buah elemen pun di dalam queue atau dengan kata lain queue masih dalam keadaan kosong. Untuk itu, pointer `depan` (pointer yang akan digunakan untuk menunjuk ke elemen pertama) dan pointer `belakang` (pointer yang akan digunakan untuk menunjuk ke elemen terakhir) keduanya masih menunjuk nilai `NULL` (tidak menunjuk kemana-mana).

Pada saat penambahan elemen yang pertama kali ke dalam queue, maka pointer `depan` akan menunjuk ke alamat dari struktur atau elemen yang baru saja ditambahkan tersebut. Karena jumlah dari elemen di dalam queue hanya satu maka pointer `belakang` juga menunjuk ke alamat dari struktur tersebut. Namun pada saat penambahan elemen yang kedua kalinya, maka pointer depan tetap menunjuk ke elemen pertama, sedangkan pointer belakang berpindah menunjuk ke alamat dari elemen kedua. Perlu diperhatikan juga bahwa field *p* yang terdapat pada elemen pertama juga akan menunjuk ke alamat dari elemen yang baru saja ditambahkan. Sedangkan field *p* yang terdapat pada elemen baru menunjuk ke nilai `NULL`. Hal ini berarti elemen tersebut merupakan elemen terakhir dalam queue. Pada penambahan elemen yang ketiga, pointer `belakang` akan menunjuk ke alamat elemen yang ketiga serta field *p* yang terdapat pada elemen kedua juga akan menunjuk ke alamat elemen baru tersebut, begitu seterusnya sehingga akan terbentuk sebuah rangkaian yang dihubungkan oleh pointer `p` di dalam struktur `node`.

9.4. Linked List dengan Dua Buah Pointer

Sejauh ini kita baru membahas mengenai keterkaitan struktur yang menggunakan sebuah pointer. Hal ini membuat linked list tersebut dinamakan dengan *Singly Linked List*. Namun, pada kenyataannya untuk dapat lebih memudahkan pengaksesan dari linked list, terkadang kita dituntut untuk menggunakan kaitan yang melibatkan dua buah pointer. Linked list jenis ini sering dinamakan dengan linked list dua arah atau *Doubly Linked List*.

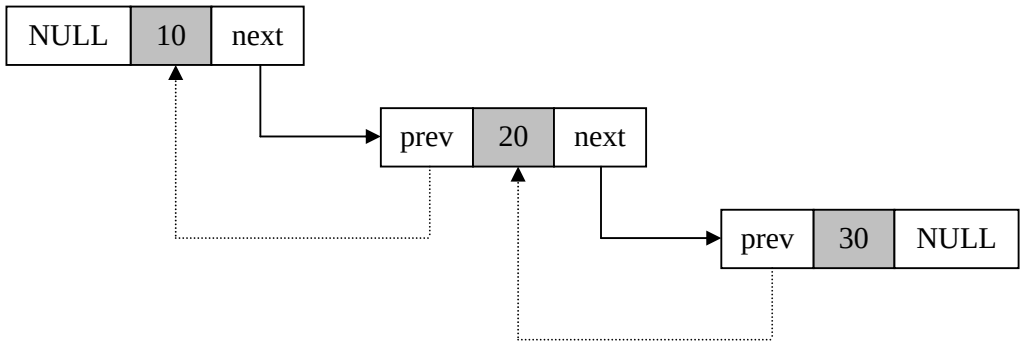
Untuk memudahkan pembahasan, perhatikan kembali struktur `node` pada sub bab di atas, yaitu dengan definisi sebagai berikut.

```
struct node {
  int data;
  struct node *p;
};
```

Di sini, pointer `p` akan menunjuk ke alamat dari struktur selanjutnya saja, artinya kita tidak memiliki pointer untuk menunjuk ke alamat dari struktur sebelumnya. Hal ini tentu berbeda dengan linked list yang menggunakan dua buah pointer. Dengan dua buah pointer, maka kita dapat menggunakan pointer pertama untuk menunjuk ke alamat dari struktur sebelumnya, sedangkan pointer satunya lagi akan digunakan untuk menunjuk ke alamat dari struktur selanjutnya. Berikut ini contoh pendefinisian struktur yang akan digunakan untuk membentuk rangkaian struktur dua arah.

```
struct node {
  struct node *prev;
  int data;
  struct node *next;
};
```

Untuk memudahkan, coba Anda perhatikan contoh gambar di bawah ini yang akan merepresentasikan keadaan linked list dengan dua arah.



Gambar 9.2. Linked list dengan dua buah pointer

Pada gambar di atas terdapat tiga buah struktur yang dikaitkan dengan dua buah pointer, yaitu pointer `prev` dan `next`. Pointer `prev` digunakan untuk menunjuk ke alamat dari struktur sebelumnya dan pointer `next` digunakan untuk menunjuk ke alamat dari struktur berikutnya. Pada struktur pertama di atas pointer `prev` menunjuk ke nilai `NULL`, ini berarti bahwa tidak terdapat lagi struktur di depannya (sebelumnya). Sedangkan pointer `next` akan menunjuk ke alamat dari struktur kedua (berikutnya). Pada struktur kedua dari gambar di atas (yang bernilai 20), pointer `prev` akan menunjuk ke alamat dari struktur pertama (sebelumnya) dan pointer `next` akan menunjuk ke alamat dari struktur ketiga (berikutnya). Terakhir, pada struktur yang ketiga (yang bernilai 30), pointer `prev` akan menunjuk ke alamat dari struktur kedua (sebelumnya) dan pointer `next` akan menunjuk ke nilai `NULL` yang berarti setelah struktur tersebut sudah tidak terdapat lagi struktur berikutnya.

Berikut ini contoh program yang merupakan implementasi dari linked list dengan dua buah pointer.

```
#include <stdio.h>
#include <stdlib.h>

/* Mendefinisikan struktur untuk membentuk suatu linked list
   dua arah */
struct node {
  struct node *prev;
  int data;
  struct node *next;
};

/* Mendefinisikan fungsi untuk menambahkan elemen pada bagian
   awal linked list */
void TambahDiAwal(struct node **s, int nilai) {
  struct node *temp;
  /* Membuat struktur node baru */
  temp = (struct node *) malloc (sizeof(struct node));
  /* Memasukkannya ke elemen pertama */
  temp->prev = NULL;
  temp->data = nilai;
  temp->next = *s;
  (*s)->prev = temp;
  *s = temp;
}

/* Mendefinisikan fungsi untuk menambahkan elemen pada bagian
   akhir linked list */
void TambahDiAkhir(struct node **s, int nilai) {
  struct node *temp, *bantu;

  if (*s != NULL) {
    bantu = *s;

    /* Mendapatkan elemen terakhir sebelum linked list ditambah
       elemen baru */
    while (bantu->next != NULL) {
      bantu = bantu->next;
    }

    /* Memasukkan elemen ke bagian akhir linked list */
    temp = (struct node*) malloc (sizeof(struct node));
    temp->prev = bantu;
    temp->data = nilai;
    temp->next = NULL;

    /* pointer next dari elemen akhir menunjuk ke alamat dari
       elemen baru */
    bantu->next = temp;
  } else {
    *s = (struct node *) malloc (sizeof(struct node));
    (*s)->prev = NULL;
```

```
    (*s)->data = nilai;
    (*s)->next = NULL;
  }
}

/* Mendefinisikan fungsi untuk menampilkan nilai yang terdapat
dalam linked list */
void TampilkanNilai(struct node *s) {
  while (s != NULL) {
    printf("%d\n", s->data);
    s = s->next;
  }
}

/* Fungsi utama */
int main(void) {
  struct node *ptr;

  ptr = NULL;   /* Mula-mula linked list dalam keadaan kosong */

  /* Menambahkan elemen ke dalam linked list */
  TambahDiAkhir(&ptr, 10);
  TambahDiAkhir(&ptr, 20);
  TambahDiAkhir(&ptr, 30);

  /* Menampilkan nilai-nilai yang terdapat dalam linked list */
  printf("Keadaan semula:\n");
  TampilkanNilai(ptr);

  /* Memasukkan elemen ke awal rangkaian */
  TambahDiAwal(&ptr, 300);

  /* Menampilkan kembali nilai-nilai yang terdapat dalam
     linked list */
  printf("\nSetelah ditambah di awal:\n");
  TampilkanNilai(ptr);
  return 0;
}
```

Hasil yang diperoleh dari program di atas adalah sebagai berikut.

```
Keadaan semula:
10
20
30

Setelah ditambah diawal:
300
10
20
30
```

9.5. Mengimplementasikan Linked List dengan Rekursi

Dalam bahasa C, kita juga diizinkan untuk menggunakan rekursi dalam pembentukan sebuah linked list. Misalnya untuk proses penambahan, penyalinan maupun menampilkan elemen-elemen yang terdapat di dalamnya. Walaupun demikian, terdapat batasan yang harus diperhatikan karena apabila terdapat banyak pemanggilan fungsi yang sama secara rekursif, maka akan mempengaruhi kecepatan jalannya program. Maka dari itu, hati-hati dalam penggunaannya karena apabila pemanggilan fungsi terlalu banyak maka hal ini dapat menyebabkan *stack overflow*.

Berikut ini contoh program sederhana yang akan menunjukkan penggunaan rekursi dalam linked list.

```
#include <stdio.h>
#include <stdlib.h>

/* Mendefinisikan struktur yang akan digunakan dalam
   linked list */
struct node {
  int data;
  struct node *p;
};

/* Mendefinisikan fungsi untuk menambahkan elemen pada bagian
   akhir dari linked list */
void TambahElemen(struct node **s, int nilai) {
  if (*s == NULL) {
    *s = (struct node *) malloc (sizeof(struct node));
    (*s)->data = nilai;
    (*s)->p = NULL;
  } else {
    TambahElemen(&((*s)->p), nilai);
  }
}

/* Mendefinisikan fungsi untuk menampilkan nilai-nilai yang
   terdapat linked list */
void TampilkanNilai(struct node *s) {
  if (s != NULL) {
    printf("%d\n", s->data);
    TampilkanNilai(s->p);
  }
}

/* Fungsi utama */
int main(void) {
  struct node *ptr;

  ptr = NULL;   /* Mula-mula dalam keadaan kosong */

  /* Menambahkan elemen ke dalam linked list */
  TambahElemen(&ptr, 10);
  TambahElemen(&ptr, 20);
```

```
  TambahElemen(&ptr, 30);
  TambahElemen(&ptr, 40);

  /* Menampilkan nilai-nilai yang terdapat pada setiap elemen

     linked list */

  TampilkanNilai(ptr);

  return 0;
}
```

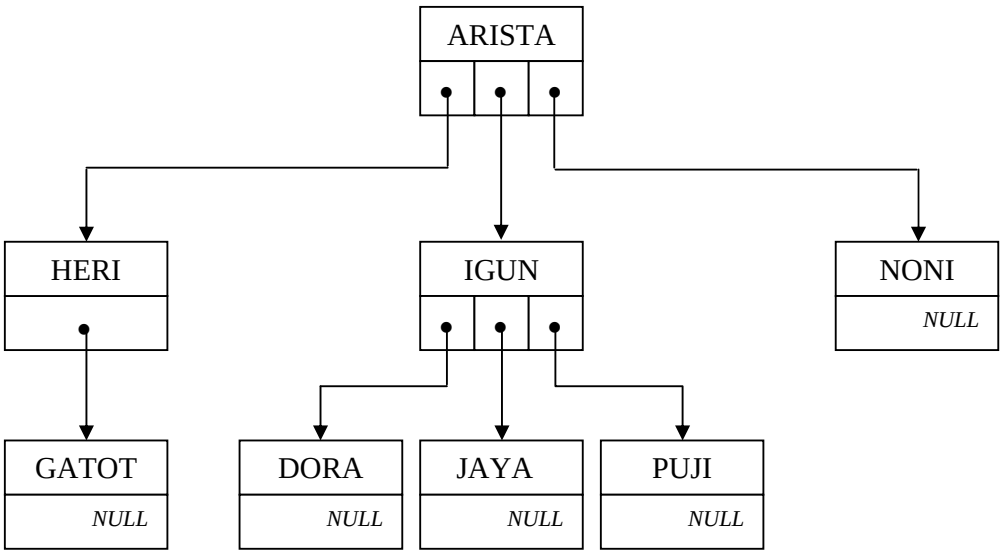
Hasil yang akan diperolah dari program di atas adalah sebagai berikut.

```
10
20
30
40
```

Tampak di atas bahwa fungsi `TambahElemen()` dan `TampilkanNilai()` merupakan fungsi rekursif, dimana di dalamnya terdapat pemanggilan pada dirinya sendiri.

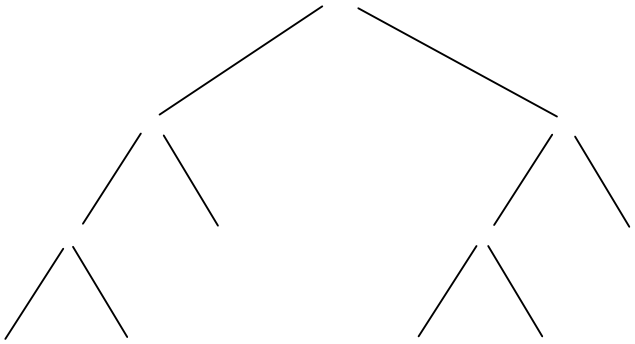
9.6. *Binary Tree* (Pohon Biner)

Tree adalah struktur yang memiliki nol atau lebih cabang yang terorganisasi dalam suatu hirarki. Sedangkan kata 'binary' berarti dua. Dengan demikian *binary tree* adalah rangkaian suatu node (direpresentasikan dengan sebuah struktur) yang maksimal akan memiliki dua buah cabang (cabang kiri dan cabang kanan). Berikut ini contoh gambar hirarki yang dapat mengilustrasikan sebuah tree secara umum.



Perhatikan contoh hirarki di atas, masing-masing elemen disebut dengan `node`. Adapun `node` yang tidak memiliki parent (dalam hal ini `ARISTA`) di sebut dengan *root*. Dalam contoh ini `ARISTA` mempunyai tiga orang anak, yaitu `HERI`, `IGUN` dan `NONI`. `HERI` sendiri memiliki seorang anak yang bernama `GATOT`. Begitu juga dengan `IGUN`, dirinya memiliki tiga orang anak, yaitu `DORA`, `JAYA` dan `PUJI`. Sedangkan `GATOT`, `DORA`, `JAYA`, `PUJI` dan `NONI` masing-masing tidak memiliki anak.

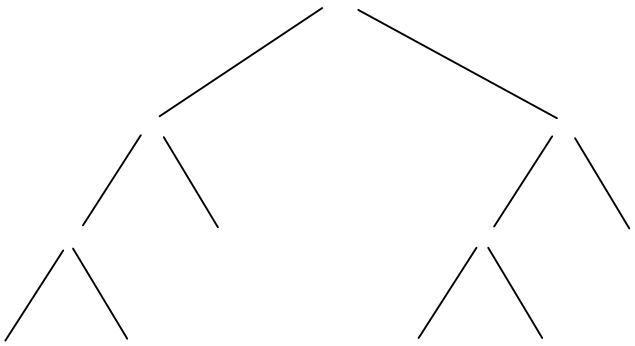
Dengan meninjau contoh tersebut diharapkan Anda dapat memahami konsep tree secara umum. Sedangkan dalam pemrograman, khususnya untuk bahasa C, tree adalah suatu struktur yang dikaitkan melalui bantuan pointer sehingga terjadi rangkaian yang berupa cabang-cabang seperti halnya ranting pohon. Dengan demikian, apabila `node` tidak memiliki cabang (baik cabang kiri maupun kanan), maka pointer yang terdapat dalam `node` tersebut akan berisi nilai `NULL`. Adapun cara pembacaan sebuah tree adalah mulai dari bagian awal elemen menuju bagian akhir elemen. Untuk lebih jelasnya, perhatikan kembali gambar binary tree di bawah ini. Sebelumnya, asumsikan kita telah memiliki sekumpulan data bertipe karakter sebagai berikut: { I,N,F,O,R,M,A,T,I,K,A }. Apabila data tersebut akan dimasukkan ke dalam binary tree, maka hasil yang akan didapatkan adalah sebagai berikut.



Para matematikawan biasanya menggunakan binary tree ini untuk menuliskan suatu ekspresi. Sebagai contoh apabila kita memiliki ekspresi berikut.

(((5 - 2) * 3) + ((6 + 4) * 2))

Maka kita dapat menuliskannya ke dalam binary tree seperti di bawah ini.



Setelah Anda mendapatkan penjelasan singkat mengenai *binary tree* di atas, sekarang coba Anda perhatikan contoh implementasinya ke dalam sebuah program. Program yang akan kita buat di bawah ini merupakan program untuk melakukan pengurutan elemen data yang bertipe karakter yang dimasukkan secara acak (tidak dalam keadaan terurut). Adapun sintaknya adalah sebagai berikut.

```
#include <stdio.h>

/* Mendefinisikan tipe data bentukan dengan nama boolean
   yang berupa enumerasi */
typedef enum {false=0, true=1} boolean;

/* Mendefinisikan struktur yang akan dijadikan binary tree */
struct btree {
  struct btree *kiri;      /* untuk menunjuk alamat node sebelah
                              kiri */
  char data;               /* untuk menampung data karakter yang
                              dimasukkan */
  struct btree *kanan;     /* untuk menunjuk alamat node sebelah
                              kanan */
};

/* Mendefinisikan fungsi untuk menambahkan data ke dalam
   binary tree */
boolean TambahData(struct btree **bt, char nilai) {
  /* Apabila binary tree masih kosong */
  if (*bt == NULL) {
    *bt = (struct btree*) malloc (sizeof(struct btree));

    if (*bt == NULL) return false;    /* Apabila pengalokasian
                                         gagal */

    /* Apabila pengalokasian berhasil */
    (*bt)->kiri = NULL;
    (*bt)->data = nilai;
    (*bt)->kanan = NULL;
    return true;
  } else {      /* Apabila binary tree sudah mengandung data */
    if (nilai < ((*bt)->data)) {
      /* Masukkan data di bagian kiri */
      return TambahData(&((*bt)->kiri), nilai);
    } else {
      /* Masukkan data di bagian kanan */
      return TambahData(&((*bt)->kanan), nilai);
    }
  }
}

/* Mendefinisikan fungsi untuk mencetak data dari
   binary tree yang terbentuk */
void CetakData(struct btree *bt) {
  /* Meletakkan data bagian kiri */
  if (bt->kiri != NULL) {
    CetakData(bt->kiri);
```

```
  }
  /* Meletakkan data aktif
    (data yang disimpan dalam field data) */
  printf("%c", bt->data);

  /* Meletakkan data bagian kiri */
  if (bt->kanan != NULL) {
    CetakData(bt->kanan);
  }
}

/* Fungsi utama */
int main(void) {
  struct btree *ptr;
  char karakter;
  boolean berhasil;        /* untuk mengetahui status pada saat
                              data dimasukkan */

  /* Mula-mula binary tree belum mengandung data */
  ptr = NULL;

  /* Menampilkan teks sebagai informasi untuk user
    (pengguna program) */
  printf("Masukkan sebuah kata:\n");

  /* Pengulangan untuk mencatat setiap karakter yang dimasukkan
     oleh user */
  do {
    karakter = getchar();
    berhasil = TambahData(&ptr, karakter);

    /* Apabila fungsi TambahData() mengembalikan nilai false */
    if (!berhasil) {        /* Apabila data gagal dimasukkan ke
                               dalam binary tree */
      printf("Data gagal dimasukkan ke dalam binary tree");
      return 0;       /* keluar program */
    }
  } while (karakter != '\n');    /* selama karakter yang
                                    dimasukkan bukan
                                    karakter baris baru */

  /* Mencetak data setelah diurutkan ke layar monitor */
  printf("\nData setelah diurutkan: ");
  CetakData(ptr);

  return 0;
}
```

Berikut ini contoh hasil yang akan diberikan oleh program di atas.

```
Masukkan sebuah kata:
KOMPUTER
```

```
Data setelah diurutkan:
EKMOPRTU
```

Dari hasil tersebut tampak jelas bahwa karakter yang kita masukkan secara acak (belum terurut) akan diurutkan oleh program dengan menggunakan binary tree sesuai dengan urutan abjad yang berlaku.

Bab

10

Operasi File

10.1. Pendahuluan

Sejauh ini data-data yang Anda gunakan di dalam program masih disimpan dalam memori komputer sehingga ketika program terhenti maka data tersebut juga akan hilang dari memori. Untuk melakukan penyimpanan terhadap data-data tersebut tentunya kita memerlukan suatu alat bantu. Dalam dunia komputer alat bantu seperti ini disebut dengan file. File bukan di simpan di dalam memori melainkan dalam suatu disk (pita magnetik), misalnya hardisk maupun disket. Data-data yang disimpan di dalam file bersifat semi permanen, artinya kita dapat menambahkan, menghapus ataupun mengubah isi dari data-data tersebut. Berdasarkan isi datanya, file terbagi ke dalam dua jenis, yaitu file teks dan file biner.

Sebagai programmer, kita harus mengetahui bagaimana cara untuk menciptakan/ membuat maupun menghapus suatu file ke atau dari dalam disk. Selain itu kita juga harus dapat melakukan manipulasi terhadap isi dari file tersebut dari dalam program yang kita buat. Untuk itu, pada bagian ini akan dibahas mengenai operasi-operasi apa saja yang dapat dilakukan terhadap suatu file serta bagaimana cara melakukannya di dalam bahasa C.

Bahasa C telah menyediakan sebuah struktur yang digunakan untuk kebutuhan dalam pengaksesan sebuah file, yaitu struktur `FILE` yang dideklarasikan di dalam file header `<stdio.h>`. Struktur tersebut akan menyimpan informasi-informasi dari file yang akan diakses, misalnya seperti lokasi (*path*), ukuran (dalam satuan *byte*), data-data yang terkandung di dalamnya serta informasi lainnya. Adapun cara pertama yang harus dilakukan untuk mendapatkan informasi dari struktur `FILE` tersebut adalah mendeklarasikan sebuah pointer yang akan menunjuk ke struktur `FILE`. Pointer semacam ini dinamakan dengan **pointer file**. Berikut ini bentuk umum dari pendeklarasian pointer ke file yang dimaksud di atas.

```
FILE *nama_pointer_ke_file;
```

Sebagai contoh kita akan mendeklarasikan pointer dengan nama `pf` untuk menunjuk ke struktur `FILE`, maka sintak penulisannya adalah sebagai berikut.

```
FILE *pf;
```

Dengan demikian kita dapat menggunakan pointer `pf` untuk mendapatkan informasi-informasi dari file yang akan kita akses di dalam program.

10.2. Membuka File

Sebelum menggunakan file di dalam suatu program, kita harus menghubungkannya terlebih dahulu dengan cara membuka file tersebut. Adapun fungsi yang digunakan untuk melakukan hal tersebut adalah fungsi `fopen()`, yaitu fungsi yang terdapat di dalam file header `<stdio.h>`. Berikut ini prototipe dari fungsi `fopen()`.

```
FILE *fopen(char *namafile, char *mode);
```

Fungsi ini akan mengembalikan pointer ke file apabila proses yang dilakukan berjalan dengan benar. Sebaliknya, apabila gagal, maka fungsi ini akan mengembalikan nilai `NULL`. Berikut ini beberapa faktor yang sering menyebabkan terjadinya kesalahan dalam proses pembukaan file.

- Nama file yang diisikan tidak absah (*valid*), misalnya nama file yang mengandung spasi ataupun tanda lain.
- Membuka file dari disk yang belum disiapkan, misalnya disk belum dimasukkan atau disk belum terformat.
- Membuka file yang tidak terdapat dalam direktori yang didefinisikan.
- Membuka file yang belum terbuat untuk proses pembacaan (mode **`"r"`**).

Parameter *`namafile`* di atas digunakan untuk menunjukkan nama file yang akan dibuka di dalam program. Sedangkan parameter *`mode`* digunakan untuk menentukan mode atau aksi yang akan dilakukan setelah file dibuka. Adapun mode-mode yang telah didefinisikan untuk fungsi `fopen()` adalah seperti yang tertera pada tabel di bawah ini.

Mode	Keterangan
`r`	File dibuka untuk proses pembacaan (*reading*). Apabila file belum ada maka proses gagal dan fungsi `fopen()` akan mengembalikan nilai NULL.
`w`	File dibuka untuk proses penulisan (*writing*). Apabila file belum ada maka file akan dibuat. Sebaliknya, apabila file yang didefinisikan sudah ada, maka file tersebut akan dihapus tanpa adanya peringatan terlebih dahulu, selanjutnya akan dibuat file baru (dengan data kosong).
`a`	File dibuka untuk proses penambahan data (*appending*). Apabila file belum ada maka file akan dibuat. Sebaliknya, apabila file telah ada maka data yang dimasukkan ke dalam file akan ditambahkan pada bagian akhir dari data lama (*end of file*).

r+	File dibuka untuk proses pembacaan dan penulisan. Apabila file belum ada maka file akan dibuat. Sebaliknya, apabila file telah ada, maka data baru akan ditambahkan pada bagian awal, yaitu dengan melakukan penimpaan (*overwriting*) terhadap data lama yang sebelumnya telah ada.
w+	File dibuka untuk proses penulisan dan pembacaan. Apabila file belum ada maka file akan dibuat. Sebaliknya, apabila file telah ada maka file tersebut akan ditimpa (*overwritten*) dengan file baru.
a+	File dibuka untuk proses pembacaan dan penambahan data. Apabila file belum ada maka file akan dibuat. Sebaliknya, apabila file telah ada maka data yang dimasukkan ke dalam file akan ditambahkan pada bagian akhir dari data lama (*end of file*).

Untuk lebih memahaminya, coba Anda perhatikan contoh program di bawah ini dimana didalamnya akan dilakukan proses pembacaan data dari suatu file dan menampilkannya ke layar monitor (sebuah terminal). Namun sebelumnya asumsikan bahwa kita telah memiliki file teks (misalnya COBA.TXT) yang disimpan di dalam direktori D di dalam komputer kita. Sebagai contoh data yang terdapat di dalam file teks tersebut adalah seperti di bawah ini.

```
Pemrograman Menggunakan Bahasa C
Budi Raharjo & I Made Joni
Penerbit INFORMATIKA Bandung
```

Adapun sintak program untuk melakukan hal tersebut adalah sebagai berikut.

```
#include <stdio.h>
#include <stdlib.h>   /* untuk menggunakan fungsi exit() */

int main(void) {

  /* Mendeklarasikan pointer ke file dengan nama pf */
  FILE *pf;

  char karakter;     /* Variabel bantu untuk menampung data
                        dari file */

  /* Membuka file dan menampungnya ke dalam pointer pf */
  pf = fopen("D:/COBA.TXT", "r");

  /* Mengecek apakah file tersebut dapat dibuka atau tidak */
  if (pf != NULL) {        /* apabila berhasil */
    while ((karakter = getc(pf)) != EOF) {  /* selama data masih
                                               ada */
      printf("%c", karakter);
    }
  } else {           /* apabila gagal */
    printf("Kesalahan: File COBA.TXT tidak dapat dibuka");
    exit(EXIT_FAILURE);    /* keluar program dengan nilai 1 */
```

```
  }

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Pemrograman Menggunakan Bahasa C
Budi Raharjo & I Made Joni
Penerbit INFORMATIKA Bandung
```

10.3. Menutup File

Setelah kita melakukan pengaksesan terhadap suatu file melalui fungsi `fopen()`, maka sebaiknya kita memutuskan koneksi dari file pointer ke file eksternal yang bersangkutan, yaitu dengan cara menutupnya menggunakan fungsi `fclose()`. Hal ini dilakukan untuk menjaga keamanan karena dalam sistem operasi bisa saja suatu file digunakan atau diakses secara simultan oleh beberapa program lain yang sedang dijalankan. Jadi, sebaiknya kita sesegera mungkin untuk menutup koneksi terhadap suatu file apabila file tersebut sudah tidak gunakan lagi di dalam program.

Fungsi `fclose()` ini merupakan kebalikan dari fungsi `fopen()` dan memiliki prototipe seperti berikut.

```
int fclose(FILE *fp);
```

Parameter *fp* di atas merupakan pointer file yang sedang menunjuk ke file yang telah dibuka. Fungsi ini akan mengembalikan nilai 0 apabila proses berjalan dengan lancar dan akan mengembalikan nilai 1 apabila terdapat kesalahan.

Dengan menutup suatu file maka informasi-informasi yang terdapat pada buffer pointer file akan dibuang sehingga pointer file dapat kita gunakan kembali untuk mengakses file lain yang kita butuhkan di dalam program. Apabila kita tidak menyertakan fungsi `fclose()` pada pointer file yang masih menunjuk ke file, maka fungsi `fclose()` ini akan dipanggil secara otomatis ketika program dihentikan. Dengan kata lain, apabila kita menggunakan beberapa pointer file di dalam program, maka ketika program dihentikan semua pointer ke file tersebut akan ditutup dan koneksi pun akan terputus.

Berikut ini contoh program yang akan menunjukkan penggunaan fungsi `fclose()`. Di sini kita akan membuat program yang dapat menghitung banyaknya baris yang terdapat dalam suatu file teks. Sebagai contoh untuk kasus ini, kita akan menggunakan kembali file COBA.TXT, yaitu file yang kita gunakan untuk program pada sub bab di atas.

```
#include <stdio.h>
#include <stdlib.h>   /* Untuk menggunakan fungsi exit() */

int main(void) {

  /* Mendeklarasikan pointer file */
  FILE *pf;
  int counter;     /* variabel bantu untuk menampung banyaknya
                      baris dalam file */
  char string[256]; /* variabel bantu untuk menampung string di
                       setiap baris */

  pf = fopen("D:/COBA.TXT", "r");

  if (pf != NULL) {
    counter = 0;      /* Mula-mula banyak baris sama dengan 0 */
    printf("Data yang terdapat dalam file:\n");
    printf("-----------------------------------");
    while( (fgets(string, 256, pf)) != EOF) {
      printf("%s", string);    /* Mencetak setiap baris ke
                                  layar monitor */
      counter++;      /* banyak baris bertambah satu */
    }
  } else {
    printf("Kesalahan:  File COBA.TXT tidak dapat dibuka");
            exit(EXIT_FAILURE);  /* sama dengan exit(1) */
  }

  /* Menampilkan banyak baris yang didapatkan */
  printf("\n-----------------------------------\n");
  printf("\nBanyaknya baris yang ditemukan: %d", counter);

  fclose(pf);         /* Menutup file */

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Data yang terdapat dalam file:
-----------------------------------------------
Pemrograman Menggunakan Bahasa C
Budi Raharjo & I Made Joni
Penerbit INFORMATIKA Bandung
-----------------------------------------------

Banyaknya baris yang ditemukan : 3
```

10.4. Membaca Data dari File

Pada program sebelumnya Anda melihat bahwa kita menggunakan fungsi `getc()` (pada sub bab 10.2) dan fungsi `fgets()` (pada sub bab 10.3). Fungsi-fungsi tersebut adalah fungsi yang digunakan untuk membaca data dari file. Artinya, data yang terdapat di dalam file kita tampung ke dalam suatu variabel dan variabel tersebut nantinya dapat kita gunakan untuk kebutuhan program. Selain kedua fungsi di atas, dalam bahasa C juga terdapat fungsi lain yang juga dapat digunakan untuk proses pembacaan data, yaitu `fscanf()`. Untuk lebih lengkapnya, perhatikan penjelasan dari masing-masing fungsi berikut.

10.4.1. Fungsi `getc()`

Fungsi `getc()` digunakan untuk mengambil atau membaca sebuah karakter yang terdapat di dalam file.Adapun prototipe dari fungsi ini adalah sebagai berikut.

```
int getc(FILE *fp);
```

Anda tidak perlu bingung antara perbedaan yang terdapat pada fungsi `getc()` dan `getchar()`. Fungsi `getchar()` digunakan untuk membaca karakter dari keyboard yang dituliskan sebagai input/masukan di dalam program, sedangkan fungsi `getc()` digunakan untuk membaca secara langsung karakter yang terdapat di dalam file. Dengan kata lain, fungsi `getchar()` akan digunakan di dalam program-program yang tidak berhubungan dengan file.

Berikut ini contoh program yang akan menunjukkan kembali penggunaan fungsi `getc()`. Di sini kita akan membuat program untuk mengetahui banyaknya huruf vokal (‘a’, ’i’, ’u’, ’e’, ’o’, ’A’, ’I’, ’U’, ’E’, ’O’) yang terdapat dalam suatu file teks. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  char karakter;     /* untuk menampung karakter yang terdapat
                        dalam file */
  /* Mendeklarasikan 5 buah counter untuk vokal A, I, U, E, O */
  int counter[5] = {0,0,0,0,0};  /* mula-mula setiap counter
                                    bernilai 0 */

  FILE *pf;
  char namafile[256]; /* untuk menampung nama file yang akan
                         dibuka */

  /* Meminta user untuk memasukkan nama file
     yang akan diproses */
  printf(“Masukkan nama file (beserta lokasinya) : ”);
  gets(namafile);
```

```
  if ((pf = fopen(namafile, "r")) != NULL) {
    printf("\nData dalam file (%s):\n", namafile);
    printf("-------------------------------------------------" \
           "------------\n");

    while ((karakter = getc(pf)) != EOF) {
      /* Mencetak karakter ke layar monitor */
      printf("%c", karakter);

      /* Melakukan perhitungan vokal */
      switch (karakter) {
        case 'a': case 'A': counter[0]++; break; /* vokal A */
        case 'i': case 'I': counter[1]++; break; /* vokal I */
        case 'u': case 'U': counter[2]++; break; /* vokal U */
        case 'e': case 'E': counter[3]++; break; /* vokal E */
        case 'o': case 'O': counter[4]++; break; /* vokal O */
        default: { }  /* tidak melakukan apa-apa */
      }
    }  /* akhir struktur while */
    printf("-------------------------------------------------" \
           "------------\n");
  } else {
    printf("Kesalahan: File %s tidak dapat dibuka", namafile);
           exit(EXIT_FAILURE);
  }

  /* Menampilkan hasil perhitungan */
  printf("\n\nDaftar huruf vokal yang ditemukan dalam file:\n");
  printf("Vokal A \t: %d\n", counter[0]);
  printf("Vokal I \t: %d\n", counter[1]);
  printf("Vokal U \t: %d\n", counter[2]);
  printf("Vokal E \t: %d\n", counter[3]);
  printf("Vokal O \t: %d\n", counter[4]);

  fclose(pf);         /* Menutup file */

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Masukkan nama file (beserta lokasinya) : D:/COBA.TXT

Data dalam file (D:/COBA.TXT):
-----------------------------------------------------------------
Pemrograman Menggunakan Bahasa C
Budi Raharjo & I Made Joni
Penerbit INFORMATIKA Bandung
-----------------------------------------------------------------
```

```
Daftar huruf vokal yang ditemukan dalam file:
Vokal A      : 12
Vokal I      : 4
Vokal U      : 3
Vokal E      : 4
Vokal O      : 3
```

10.4.2. Fungsi `fgets()`

Fungsi ini digunakan untuk membaca satu baris data yang terdapat di dalam file. Adapun prototipenya adalah sebagai berikut.

```
char *fgets(char *str, int n, FILE *fp);
```

Parameter *str* di atas berguna untuk menyimpan string yang merupakan nilai kembalian dari fungsi tersebut, *n* berguna untuk menentukan panjang string maksimal yang akan dimasukkan ke string *str*. Sedangkan parameter *fp* merupakan pointer ke struktur `FILE`.

Berikut ini contoh program yang akan menunjukkan kembali penggunaan fungsi `fgets()`. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 256

int main(void) {

  FILE *pf;
  char str[256];   /* variabel untuk menampung string
                      setiap baris */

  pf = fopen("D:/COBA.TXT", "r");

  printf("Membaca data dari file (D:/COBA.TXT):\n");
  if (pf != NULL) {
    /* Membaca 2 baris pertama dari data yang terdapat
       dalam file */
    for (int i=0; i<2; i++) {
      fgets(str, MAX, pf);
      /* Menampilkan variabel str ke layar */
      printf("Baris ke-%d : %s", i+1, str);
    }
  } else {
    printf("Kesalahan: File COBA.TXT tidak dapat dibuka");
    exit(EXIT_FAILURE);
  }
```

```
  fclose(pf);

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Membaca data dari file (D:/COBA.TXT):
Baris ke-1 : Pemrograman Menggunakan Bahasa C
Baris ke-2 : Budi Raharjo & I Made Joni
```

10.4.3. Fungsi `fscanf()`

Konsep dari fungsi `fscanf()` adalah sama dengan fungsi `scanf()`, yaitu untuk mengambil data dengan format tertentu. Perbedaannya, fungsi `fscanf()` operasinya dilakukan terhadap file, sedangkan `scanf()` dilakukan terhadap data yang dimasukkan atau dibaca dari keybord. Adapun prototipe dari fungsi `fscanf()` adalah sebagai berikut.

```
int fscanf(FILE *fp, char *format, ...);
```

Parameter *fp* merupakan pointer ke file yang akan dibaca datanya, sedangkan parameter *format* menunjukkan format data yang akan dibaca. Hal ini tentu disesuaikan dengan tipe datanya. Tanda **...** di atas berarti kita dapat mendefinisikan alamat yang akan digunakan untuk mengisikan data-data hasil proses pembacaan, yang disesuaikan dengan format yang didefinisikan.

Untuk lebih memahaminya, coba Anda perhatikan contoh program di bawah ini. Di sini kita akan membaca data yang terdapat pada file COBA.TXT dengan menggunakan fungsi `fscanf()`.

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  FILE *pf;
  char karakter;  /* Variabel bantu untuk menampung data
                     dari file */

  pf = fopen("D:/COBA.TXT", "r");

  if (pf != NULL) {
    while ( (fscanf(pf, "%c", &karakter)) != EOF) {
      printf("%c", karakter);
```

```
    }
  } else {
    printf("Kesalahan: File COBA.TXT tidak dapat dibuka");
    exit(EXIT_FAILURE);
  }

  fclose(pf);

  return 0;
}
```

Hasil yang diperoleh dari program di atas akan sama persis seperti pada saat kita menggunakan fungsi `getc()` untuk kasus yang sama.

10.5. Menulis Data ke dalam File

Setelah mengetahui bagaimana cara membaca data dari suatu file, kini saatnya Anda mengetahui bagaimana suatu data ditulis atau dimasukkan ke dalam file. Hal yang perlu diingat di sini adalah dalam membuka file kita harus menggunakan mode-mode untuk proses penulisan data, yaitu selain mode **"r"** (*reading*). Sama seperti proses pembacaan data, untuk menulis data ke file juga disediakan tiga buah fungsi, yaitu `putc()`, `fputs()` dan `fprintf()`. Berikut ini penjelasan dari masing-masing fungsi tersebut.

10.5.1. Fungsi `putc()`

Fungsi `putc()` merupakan kebalikan dari fungsi `getc()`, yaitu digunakan untuk menuliskan sebuah karakter ke dalam file. Prototipe fungsi ini adalah sebagai berikut.

```
int putc(int c, FILE *fp);
```

Parameter *c* di atas digunakan untuk menampung karakter yang akan ditulis atau dimasukkan ke dalam file, sedangkan *fp* merupakan pointer ke file tujuan.

Sebagai contoh penggunaan fungsi `putc()`, di sini kita akan membuat program untuk melakukan penyalinan (*copy*) file. Adapun sintak programnya adalah seperti berikut.

```
#include <stdio.h>
#include <stdlib.h>

/* Mendefinisikan fungsi untuk proses copy file */
int CopyFile(FILE *filesumber, FILE *filetujuan) {
  char karakter;
  /* Membaca data dari file sumber dan menuliskannya ke file
     tujuan */
  while ( (karakter = getc(filesumber)) != EOF ) {
    putc(karakter, filetujuan);
  }
```

```
}

int main(void) {
  FILE *pfsumber, *pftujuan;
  char namafilesumber[256], namafiletujuan[256];

  /* Meminta user untuk memasukkan nama file sumber
     dan tujuan */
  printf("Melakukan penyalinan (copy) file:\n");
  printf("Nama file sumber \t: "); gets(namafilesumber);
  printf("Nama file tujuan \t: "); gets(namafiletujuan);

  /* Membuka file sumber untuk proses pembacaan data */
  pfsumber = fopen(namafilesumber, "r");

  if (pfsumber == NULL) {
    printf("Kesalahan: File %s tidak dapat dibuka");
    exit(EXIT_FAILURE);
  }

  /* Membuka file tujuan untuk proses penulisan data */
  pftujuan = fopen(namafiletujuan, "w+");

  /* Melakukan copy file */
  if (CopyFile(pfsumber, pftujuan)) {
    printf("\nProses copy file berhasil");
  } else {
    printf("\nProses copy file gagal");
  }

  /* Menutup file */
  fclose(pftujuan);
  fclose(pfsumber);

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Melakukan penyalinan (copy) file:
Nama file sumber      : D:/COBA.TXT
Nama file tujuan      : D:/BARU.TXT

Proses copy file berhasil
```

Sekarang coba Anda lihat di drive D, di situ akan terbentuk file baru dengan nama BARU.TXT yang isinya sama persis dengan file COBA.TXT.

10.5.2. Fungsi `fputs()`

Fungsi `fputs()` merupakan kebalikan dari fungsi `fgets()`, yaitu digunakan untuk melakukan penulisan atau memasukkan satu buah baris data ke dalam file. Adapun prototipe dari fungsi ini adalah sebagai berikut.

```
char fputs(char *str, FILE *fp)
```

Parameter *`str`* digunakan untuk menampung string yang akan dimasukkan atau dituliskan ke dalam file, sedangkan *`fp`* merupakan pointer yang menunjuk ke file tujuan. Fungsi ini akan mengembalikan nilai non-negatif apabila proses berhasil, sebaliknya apabila gagal, maka nilai yang dikembalikan adalah `EOF` (*End Of File*).

Berikut ini contoh program yang akan menunjukkan proses penulisan data ke dalam file dengan menggunakan fungsi `fputs()`. Adapun sintak programnya adalah seperti yang tertulis di bawah ini.

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  FILE *pf;

  /* Membuat file baru dengan nama PUTS.TXT untuk diisi data */
  pf = fopen("D:/PUTS.TXT", "w+");

  if (pf != NULL) {
    /* Menulis data ke file */
    fputs("Toko Buku BI-OBSES", pf);
    fputs("Bandung", pf);
  } else {
    printf("Kesalahan: File PUTS.TXT tidak dapat dibuka");
    exit(EXIT_FAILURE);
  }

  fclose(pf);

  return 0;
}
```

Jalankan program di atas, maka Anda akan mendapatkan file baru yang terdapat di drive D dengan nama PUTS.TXT. Adapun isi dari file tersebut adalah sebagai berikut.

```
Toko Buku BI-OBSES
Bandung
```

10.5.3. Fungsi `fprintf()`

Fungsi `fprintf()` merupakan kebalikan dari fungsi `fscanf()`. Fungsi ini digunakan untuk menuliskan data dengan format tertentu ke dalam file. Prototipenya adalah sebagai berikut.

```
int fprintf(FILE *fp, char *format, ...);
```

Parameter *`fp`* merupakan pointer ke file yang akan diproses, sedangkan parameter *`format`* menunjukkan format data yang akan dituliskan ke dalam file.

Sebagai contoh dari penggunaan fungsi `fprintf()`, coba Anda perhatikan program di bawah ini.

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  FILE *pf;
  char karakter;
  int data[5], i;

  pf = fopen("D:/PRINTF.TXT", "w+");

  printf("Masukkan data (berupa bilangan bulat)\n");
  for (i=0; i<5; i++) {
    printf("Data ke-%d : ", i+1); scanf("%d", &data[i]);
  }

  if (pf != NULL) {
    for (i=0; i<5; i++) {
      /* Menulis data ke dalam file dengan fungsi fprintf() */
      fprintf(pf, "Data ke-%d \t: %d\n", i+1, data[i]);
    }
  } else {
    printf("Kesalahan: File PUTS.TXT tidak dapat dibuka");
    exit(EXIT_FAILURE);
  }

  fclose(pf);

  return 0;
}
```

Pada program di atas kita akan diminta untuk mengisikan lima buah data bertipe int dan data tersebut kemudian akan disimpan ke dalam file PRINTF.TXT.

10.6. Mendeteksi `EOF` (*End Of File*)

Pada program di atas kita telah sering menggunakan makro `EOF` untuk mendeteksi akhir dari suatu file. Makro `EOF` didefinisikan di dalam file header <stdio.h>. Namun makro `EOF` tersebut hanya digunakan untuk file teks, apabila kita akan melakukan pembacaan data dari file biner maka fungsi yang harus kita gunakan untuk mendeteksi akhir dari suatu file adalah `feof()`. Adapun prototipe dari fungsi tersebut adalah sebagai berikut.

```
int feof(FILE *fp);
```

Fungsi ini akan mengembalikan nilai 0 selama akhir file belum ditemukan dan nilai selain 0 apabila akhir file telah ditemukan.

Sebagai tambahan bagi Anda bahwa untuk membaca suatu karakter dari file biner kita menggunakan file `fgetc()`, dan untuk menuliskan ke file kita menggunakan fungsi `fputc()`. Parameter dari fungsi `fgetc()` ini sama persis dengan parameter fungsi `getc()`, sedangkan parameter `fputc()` juga sama dengan parameter fungsi `putc()`.

10.7. Manajemen File

Manajemen file merupakan suatu hal yang perlu dilakukan apabila kita ingin menempatkan file-file sesuai dengan tempat dan direktori-nya masing-masing. Adapun manajemen yang dimaksud di sini tidak lain adalah suatu penyalinan (*copy*) file, pengubahan nama file dan juga penghapusan file.

10.7.1. Menyalin File

Untuk dapat melakukan penyalinan dari satu file ke file lain dari dalam program, baik itu berupa pembentukan file baru maupun penimpaan file yang telah ada, kita harus mendefinisikan fungsi sendiri karena di dalam bahasa C tidak disediakan fungsi standar untuk melakukan hal tersebut. Hal yang perlu diperhatikan pada saat kita mendefinisikan fungsi tersebut adalah kita harus mengeset mode dari file yang akan dibuka menjadi mode biner (**`"rb"`** atau **`"wb"`**). Ini bertujuan agar fungsi tersebut juga dapat digunakan untuk melakukan penyalinan file biner.

Berikut ini contoh sederhana yang akan menunjukkan pendefinisian sebuah fungsi untuk proses penyalinan di dalam program yang kita buat. Program ini merupakan modifikasi dari program di atas yang sudah pernah kita buat (*lihat sub bab 10.5.1*) Selanjutnya fungsi tersebut akan digunakan untuk melakukan penyalinan file COBA.TXT ke file baru yang menjadi tujuan, yaitu SALIN.TXT.

```
#include <stdio.h>

/* Mendefinisikan fungsi untuk proses copy file */
int CopyFile(char *filesumber, char *filetujuan) {
```

```
  FILE *pfsumber, *pftujuan;
  int c;   /* variabel bantu untuk menampung data per byte */

  /* Membuka file sumber dalam mode pembacaan
     file biner (“rb”) */
  pfsumber = fopen(filesumber, “rb”);
  if (pfsumber == NULL) {
    printf(“Terdapat kesalahan pada saat membuka file %s”,
           filesumber);
    return -1;
  }

  /* Membuka file sumber dalam mode penulisan
     file biner (“wb”) */
  pftujuan = fopen(filetujuan, “wb”);
  if (pftujuan == NULL) {
    fclose(pfsumber); /* Menutup file sumber */
    return -1;  /* Mengembalikan nilai –1
                   apabila proses gagal */
  }

  /* Membaca data per byte dari file sumber dan menuliskannya ke
     file tujuan */
  while (1) {
    c = fgetc(pfsumber);
    if (!feof(pfsumber)) {
      fputc(c, pftujuan);
    } else {
      break;
    }
  }
  /* Menutup semua file */
  fclose(pftujuan);
  fclose(pfsumber);

  return 0; /* mengembalikan nilai 0 apabila proses berhasil */
}

/* Fungsi utama */
int main(void) {

  char namafilesumber[256], namafiletujuan[256];

  /* Meminta user untuk memasukkan nama file sumber
     dan tujuan */
  printf(“Melakukan penyalinan (copy) file:\n”);
  printf(“Nama file sumber \t: ”); gets(namafilesumber);
  printf(“Nama file tujuan \t: ”); gets(namafiletujuan);

  /* Melakukan copy file */
  if (CopyFile(namafilesumber, namafiletujuan) == 0) {
    printf(“\nProses copy file berhasil”);
  } else {
    printf(“\nProses copy file gagal”);
  }
```

```
  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah seperti berikut.

```
Melakukan proses penyalinan (copy) file:
Nama file sumber      : D:/COBA.TXT
Nama file tujuan      : D:/SALIN.TXT

Proses copy file berhasil
```

10.7.2. Mengubah Nama File

Untuk mengubah suatu nama file dari dalam program, maka kita dapat menggunakan fungsi standar yang telah disediakan oleh bahasa C. Adapun fungsi yang dimaksud tersebut adalah fungsi `rename()` yang memiliki prototipe berikut.

```
int rename(const char *oldname, const char *newname);
```

Parameter *`oldname`* merupakan nama file lama yang akan diubah menjadi nama file baru, yaitu yang didefinisikan di dalam parameter *`newname`*. Fungsi ini akan mengembalikan nilai 0 apabila proses berhasil atau nilai –1 apabila proses gagal.

Berikut ini contoh program sederhana yang akan menunjukkan penggunaan fungsi `rename()`. Di sini kita akan mengubah nama file COBA.TXT menjadi GANTI.TXT. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>

int main(void) {
  int status;   /* untuk mengetahui status berhasil tidaknya
                   proses yang dilakukan */

  /* Mengubah nama file COBA.TXT menjadi GANTI.TXT */
  status = rename("D:/COBA.TXT", "D:/GANTI.TXT");

  if (status == 0) {
    printf("File COBA.TXT telah diganti dengan nama GANTI.TXT");
  } else {
    printf("proses perubahan nama file gagal");
  }
  return 0;
}
```

10.7.3. Menghapus File

Selain melakukan penyalinan atau perubahan nama file, mungkin kita juga membutuhkan untuk menghapus suatu file dari dalam program yang kita buat. Untuk melakukan hal tersebut, bahasa C telah menyediakan fungsi `remove()`. Adapun prototipe dari fungsi ini adalah sebagai berikut.

```
int remove(const char *filename);
```

Parameter *`filename`* di atas menunjukkan nama file yang akan dihapus. Fungsi ini akan mengembalikan nilai 0 apabila proses berhasil dan nilai –1 apabila proses gagal.

Untuk lebih jelasnya, coba Anda perhatikan contoh program di bawah ini. Di sini kita akan menghapus file TEST.TXT yang terdapat pada direktori PROG dalam drive D.

```
#include <stdio.h>

int main(void) {
  int status;

  status = remove("D:/PROG/TEST.TXT");

  if (status == 0) {
    printf("File telah terhapus");
  } else {
    printf("Proses penghapusan file gagal");
  }

  return 0;
}
```

10.8. File Temporari

Dalam kasus-kasus tertentu mungkin program kita perlu untuk menggunakan file temporari. Adapun yang disebut dengan file temporari di sini adalah file yang dibuat oleh program pada saat program dijalankan dan setelah file tersebut sudah tidak digunakan lagi atau sebelum program dihentikan, maka file akan dihapus. Dengan kata lain file temporari hanya digunakan pada saat program dijalankan. Ingatlah untuk menghapus file temporari setelah selesai digunakan karena hal ini tidak dilakukan secara otomatis oleh program.

Bahasa C telah menyediakan fungsi standar untuk melakukan proses pembentukan file temporari, yaitu fungsi `tmpnam()`. Prototipenya adalah sebagai berikut.

```
char *tmpnam(char *s)
```

Parameter *s* digunakan untuk menyimpan nama file temporari yang akan dibuat.

Bab 11

Directive

11.1. Pendahuluan

Preprocessor C adalah sebuah macro processor yang digunakan secara otomatis oleh kompiler C untuk mentransformasikan program yang kita buat sebelum melakukan proses kompilasi sebenarnya. Disebut "macro processor" karena di sini kita diizinkan untuk mendefinisikan suatu makro yang dapat memperpendek konstruksi dari sebuah program. Sebagai contoh, kita dapat menggunakan *directive* `#include` untuk melakukan *linking* terhadap file header yang akan kita sertakan dalam program.

Secara umum, kita dapat mengatakan bahwa preprocessor C telah menyediakan empat buah fasilitas yang dapat kita gunakan untuk keperluan-keperluan tertentu dalam pembuatan program, yaitu sebagai berikut.

- Menyertakan file header. Di sini artinya file-file header yang kita tuliskan (deklarasikan) di bagian atas program akan digantikan oleh kode-kode program (bagian implementasi) yang terdapat dalam file header tersebut.

- Mendefinisikan makro yang dapat mempersingkat penulisan kode program.

- Melakukan pengkondisian pada proses kompilasi.

- Melakukan kontrol baris pada program. Sebagai contoh apabila program yang kita tulis terpisah dalam dua buah file dan kita akan menggabungkannya ke dalam file gabungan, maka kita dapat menggunakan preprocessor directive untuk menginformasikan dimana letak baris-baris tertentu.

Dalam bahasa C, preprocessor directive adalah bagian program yang didepannya ditandai dengan tanda # (*pound*). Adapun yang termasuk ke dalam preprocessor directive dalam bahasa C adalah seperti yang terlihat di bawah ini.

```
#include        #if             #ifdef
#define         #elif           #ifndef
#pragma         #else           #undef
#error          #endif          #line
```

11.2. Directive `#include`

Directive `#include` digunakan untuk melakukan deklarasi file header yang akan digunakan dalam program. Apabila kita akan melakukan pemanggilan fungsi yang terdapat dalam file header tertentu, maka kita juga harus mendaftarkan atau

mendeklarasikan file header tersebut ke dalam program melalui directive `#include`. Sebagai contoh, bila kita akan menggunakan file `exit()`, sedangkan file tersebut terdapat dalam file `<stdlib.h>`, maka kita harus menuliskan sintak seperti di bawah ini.

```
#include <stdlib.h>
```

Directive `#include` juga sering dituliskan dengan menggunakan kutip ganda, seperti contoh berikut.

```
#include "my_header.h"
```

Dalam bab 1 (*Pengenalan Bahasa C*) telah disinggung perbedaan antara keduanya, namun di sini marilah kita bahas kembali secara lebih rinci agar Anda dapat lebih benar-benar memahaminya.

Tanda <> digunakan untuk mengapit file header yang lokasinya (*path*-nya) telah diset secara otomatis oleh kompiler. Sedangkan tanda "" digunakan untuk mengapit file header yang lokasinya kita dapat kita tentukan sendiri. Namun apabila kita tidak menyertakan nama direktori dari file tersebut, maka kompiler akan menganggap bahwa file header yang kita gunakan terdapat dalam direktori yang sedang aktif. Adapun cara untuk menyertakan file header yang tersimpan dalam suatu direktori tertentu adalah sebagai berikut.

```
#include "nama_dir/nama_file_header"
```

Berikut ini contoh pendeklarasiannya.

```
#include "dir/my_header.h"
```

Sintak di atas artunya kita menyertakan file header my_header.h yang terdapat di dalam direkrori dir. Konsep kerjanya sederhana, yaitu kompiler akan menggantikan deklarasi file header tersebut dengan kode-kode program yang merupakan bagian implementasinya.

11.3. Directive `#define`

Kita dapat mendefiniskan sebuah makro dengan menggunakan directive `#define`. Dalam bahasa C, directive `#define` ini juga biasanya digunakan untuk membuat sebuah konstanta nilai di dalam program. Sebuah makro tidak akan menempati ruang di memori komputer, cara kerjanya hanya menggantikan sebuah makro bersangkutan dengan teks yang terdapat di belakangnya. Sebagai contoh, apabila kita memiliki makro

(misalnya dengan nama MAKRO) yang diikuti dengan nilai 20, maka kita dapat menuliskannya sebagai berikut.

```
#define MAKRO 20      /* ingat, tidak diakhiri dengan tanda
                         titik koma (;) */
```

Hal ini akan menyebabkan setiap terdapat teks MAKRO di dalam kode program, maka teks tersebut akan diganti dengan teks 20. Sebagai contoh, perhatikan program di bawah ini.

```
#include <stdio.h>

#define MAKRO 20

int main() {
  int x;   /* Mendeklarasikan variable x */
  x = MAKRO * MAKRO;
  printf("Nilai x = %d", x);
  return 0;
}
```

Sebelum dilakukan proses kompilasi, maka kompiler akan mengganti sintak di atas menjadi sebagai berikut.

```
#include <stdio.h>

#define MAKRO 20

int main() {
  int x;   /* Mendeklarasikan variable x */
  x = 20 * 20;
  printf("Nilai x = %d", x);
  return 0;
}
```

Apabila dijalankan, program tersebut akan memberikan hasil seperti di bawah ini.

```
Nilai x = 400
```

Untuk dapat lebih membuktikan hal di atas, di sini kita akan menuliskan contoh program lain yang akan menggantikan makro dengan sebuah string. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>

#define TEKS “Saya sedang belajar bahasa C”

int main() {
  printf(“%s”, TEKS);

  return 0;
}
```

Cara kerjanya sama seperti pada program sebelumnya, di sini makro dengan nama `TEKS` akan digantikan dengan string `"Saya sedang belajar bahasa C"`. Maka dari itu, hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Saya sedang belajar bahasa C
```

Makro juga dapat mengandung parameter seperti halnya sebuah fungsi sehingga makro seperti ini sering pula disebut dengan makro fungsi. Berikut ini contoh pendefinisian makro yang memiliki parameter.

```
/* makro untuk fungsi F(x) = x2 + x + 3 */
#define F(x) ((x*x) + (x) + 3)

/* makro untuk menentukan nilai absolut */
#define ABS(x) (x < 0) ? (-x) : (x)

/* makro untuk menentukan nilai maksimal */
#define MAX(x, y) (x  > y) ? x : y
```

Adapun contoh penggunaannya di dalam program adalah sebagai berikut.

```
#include <stdio.h>

/* Mendefinisikan makro */
#define F(x)  ((x*x) + (x) + 3)
#define ABS(x) (x < 0) ? (-x) : (x)
#define MAX(x, y) (x  > y) ? x : y

int main() {
  int a, b;

  printf(“Masukkan nilai a : ”); scanf(“%d”, &a);
  printf(“Masukkan nilai b : ”); scanf(“%d”, &b);

  /* Menggunakan makro */
```

```
  printf("\nNilai F(%d) \t\t = %d\n", a, F(a));
  printf("Nilai ABS(%d) \t\t = %d\n", b, ABS(b));
  printf("Nilai MAX(%d, %d) \t = %d\n", a, b, MAX(a,b));

  return 0;
}
```

Program di atas akan memberikan hasil seperti di bawah ini.

```
Masukkan nilai a : 4
Masukkan nilai b : -15

Nilai F(4)          = 23
Nilai ABS(-15)      = 15
Nilai MAX(4, -15)   = 4
```

Hal yang perlu diperhatikan dalam membuat makro fungsi adalah penggunaan tanda kurung. Apabila kita salah dalam penulisan makro maka hasil yang diinginkan pun akan berbeda. Berikut ini contoh program yang melakukan pendefinisian makro dengan cara yang salah.

```
#include <stdio.h>

/* Mendefinisikan makro untuk menghitung nilai kuadrat */
#define KUADRAT(x) x * x

int main() {
  /* Menggunakan makro */
  printf("KUADRAT(4) = %d", KUADRAT(4));

  return 0;
}
```

Hasil yang akan diberikan oleh program di atas adalah sebagai berikut.

```
KUADRAT(4) = 16
```

Apa yang dapat Anda simpulkan dari pendefinisian makro dalam program di atas? Pada contoh di atas hasil yang diberikan memang benar, namun bagaimana apabila kita menggunakan makro tersebut dengan sintak di bawah ini.

```
KUADRAT(2+2)
```

Oleh karena cara kerja makro adalah dengan menggantikan teks dalam program, maka sintak tersebut akan diterjemahkan sebagai berikut.

2+2 * 2+2

Kita tahu prioritas dari operator perkalian (*) lebih tinggi dibandingkan operator penjumlahan (+) sehingga operasi yang akan dilakukan pertama kali dari sintak di atas akan operasi perkalian. Hal ini tentu menghasilkan nilai di bawah ini.

2 + **4** + 2 */* akan menghasilkan nilai 8 */*

Dengan demikian, dapat kita simpulkan bahwa pendefinisian makro di atas adalah salah. Hal ini disebabkan oleh karena hal berikut.

KUADRAT(4) = 16

sedangkan

KUADRAT(2+2) = 8

Sekarang mungkin Anda akan bertanya bagaimana cara mendefinisikan makro tersebut supaya hasilnya sesuai dengan yang diinginkan? Jawabnya adalah dengan menambahkan tanda kurung yang sesuai dengan kebutuhan. Dalam kasus ini, seharusnya kita mendefinisikan makro tersebut dengan sintak seperti berikut.

```
#define KUADRAT (x) (x) * (x)

/* atau lebih lengkap lagi bila ditulis seperti di bawah ini */

#define KUADRAT (x) ((x) * (x))
```

Dengan cara seperti ini, apabila kita melakukan pemanggilan makro tersebut dengan menuliskan sintak berikut.

```
KUADRAT(2+2)
```

Maka nilai yang akan dihasilkan tetaplah 16 (bukan 8). Hal ini disebabkan karena makro tersebut akan diganti dengan sintak berikut.

(2+2) * (2+2) */* sama dengan 4 * 4, menghasilkan nilai 16 */*

Satu hal lagi yang perlu diketahui dalam penulisan makro adalah apabila kita ingin menuliskannya lebih dari satu baris, maka kita harus menggunakan tanda *backslash* (\) untuk menggabungnya. Perhatikan contoh pendefinisian makro di bawah ini.

```
#define TEKS “Saya sedang belajar bahasa C”

#define F(x) (2*x*x) + (3*x) + 5
```

Makro tersebut dapat ditulis dengan cara berikut.

```
#define TEKS “Saya sedang belajar ” \
             “bahasa C”

#define F(x) (2*x*x) + \
             (3*x) + \
              5
```

11.4. Pengkodisian dalam proses Kompilasi

Dalam proses kompilasi kita juga dapat melakukan pengecekan kondisi terhadap kompiler, yaitu dengan menggunakan directive `#if`, `#ifdef` ataupun `#ifdef`. Masing-masing penggunaan dari directive tersebut akan diakhiri oleh directive `#endif`.

11.4.1. Directive `#if` dan `#endif`

Directive ini digunakan untuk melakukan pengecekan terhadap suatu nilai tertentu. Artinya apabila kondisi terpenuhi maka statemen yang terdapat pada blok pengecekan akan dieksekusi sedangkan apabila kondisi bernilai salah (tidak terpenuhi) maka statemen tersebut akan diabaikan. Berikut ini bentuk umum dari penggunaan directive `#if`.

```
#if kondisi
  statemen_yang_akan_dieksekusi;
  …
#endif
```

Berikut ini contoh program yang akan menunjukkan penggunaan directive `#if`.

```
#include <stdio.h>

#define MAX 10

int main() {
  #if MAX > 5
    prinf(“Nilai MAX lebih besar dari 5”);
  #endif
  return 0;
}
```

Hasil yang akan diberikan oleh program di atas adalah sebagai berikut.

```
Nilai MAX lebih besar dari 5
```

Contoh lain penggunaan directive `#if` adalah untuk melakukan pengecekan terhadap suatu makro, apakah sudah didefinisikan atau belum. Di sini kita akan menggunakan operator `defined` untuk melakukan pengecekan tersebut. Berikut ini contoh sintak yang dimaksud di atas.

```
#include <stdio.h>

#if defined (__WIN32__)
  #define OS “Windows”
#endif

int main() {

  printf(“Sistem Operasi : %s”, OS);

  return 0;
}
```

Apabila Anda menggunakan sistem operasi Windows, maka program di atas akan memberikan hasil sebagai berikut.

```
Sistem Operasi : Windows
```

11.4.2. Directive `#else` dan `#elif`

Directive `#else` dan `#elsif` ditujukan untuk melakukan pengecekan kondisi yang banyaknya lebih dari satu. Berikut ini bentuk umum dari penggunaan directive `#else` dan `#elif`.

```
#if kondisi1
  statemen_yang_akan_dieksekusi;
#elif kondisi2
  statemen_yang_akan_dieksekusi;
…
  …
#else
  statemen_yang_akan_dieksekusi;
#endif
```

Sebagai contoh pertama, di sini kita akan membuat contoh program yang akan menggunakan directive `#else`. Adapun sintaknya adalah sebagai berikut.

```
#include <stdio.h>

#define MAX 3

int main() {
  #if MAX > 5
    printf("Nilai MAX lebih besar dari 5");
  #else
    printf("Nilai MAX lebih kecil dari 5");
  #endif

  return 0;
}
```

Hasil yang akan diberikan oleh program di atas adalah sebagai berikut.

```
Nilai MAX lebih kecil dari 5
```

Contoh kedua, yaitu program yang akan menunjukkan penggunaan directive `#elif`. Berikut ini sintak programnya.

```
#include <stdio.h>

#define MAX 5

int main() {
  #if MAX > 5
    printf("Nilai MAX lebih besar dari 5");
  #elif MAX = 5
    printf("Nilai MAX sama dengan 5");
  #else
    printf("Nilai MAX lebih kecil dari 5");
  #endif
```

```
  return 0;
}
```

Apabila dijalankan, program tersebut akan memberikan hasil seperti di bawah ini.

```
Nilai MAX sama dengan 5
```

Adapun contoh lain dari penggunaan directive `#else` dan `#elif` adalah sebagai berikut.

```
#include <stdio.h>

#if defined (__WIN32__)
  #define OS “Windows”
#elif defined (__LINUX__)
  #define OS “Linux”
#else
  #define OS “Tidak diketahui”
#endif

int main() {

  printf(“Sistem Operasi : %s”, OS);
  return 0;
}
```

11.4.3. Directive `#ifdef`

Pada sub bab sebelumnya kita mengecek suatu makro apakah sudah didefinisikan atau belum dengan menggunakan directive `#if` dan operator unary `defined`. Namun, dalam bahasa C sebenarnya telah tersedia directive yang fungsinya sama, yaitu directive `#ifdef`. Dengan demikian, maka kita dapat menuliskan sintaknya sebagai berikut.

```
#if defined (__WIN32__)
  ...
#endif

/* dapat ditulis dengan sintak di bawah ini */

#ifdef __WIN32__
  ...
#endif
```

11.4.4. Directive #ifndef

Directive ini merupakan kebalikan dari directive #ifdef. Artinya apabila sebuah makro yang diperiksa belum didefinisikan di dalam program maka bagian yang terdapat di dalam blok pengecekan akan dieksekusi. Sebaliknya, apabila makro tersebut sudah terdefinisi, maka statemen dalam blok pengecekan akan diabaikan. Directive ini biasanya dituliskan apabila kita akan mendefinisikan suatu makro, hal ini berguna untuk menghindari adanya duplikasi nama makro di dalam program. Berikut ini contoh dari penggunaan directive #ifndef.

```
#ifndef MAX
  #define MAX 10
#else
  ...
#endif

/* atau dapat ditulis juga dengan sintak di bawah ini */

#if !defined (MAX)
  #define MAX 10
#else
  ...
#endif
```

11.5. Directive #undef

Directive ini digunakan untuk melakukan penghapusan terhadap makro tertentu. Hal ini dilakukan apabila suatu makro sudah tidak diperlukan lagi kehadirannya di dalam sebuah program. Berikut ini contoh program yang akan menunjukkan penggunaan directive #undef.

```
#include <stdio.h>

#ifndef X
  #define X 10  /* mendefinisikan makro X */
#endif

int main() {

  #ifdef X
    printf("Makro X terdefinisi\n");
    printf("Nilai X = %d\n");          /* Menggunakan makro X */
    #undef X                    /* Menghapus makro X */
  #endif

  #ifndef X
    printf("Makro X sudah tidak terdefinisi");
  #else
    printf("Makro X masih terdefinisi");
```

```
  #endif;

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Makro X terdefinisi
Nilai X = 10
Makro X sudah tidak terdefinisi
```

Dari hasil tersebut dapat kita simpulkan bahwa setelah penggunaan directive `#undef` terhadap suatu makro maka makro tersebut akan tidak dikenali lagi oleh program. Apabila kita membutuhkannya lagi, kita harus mendefinisikannya lagi dengan meggunakan directive `#define`.

11.6. Directive `#pragma`

Directive `#pragma` mengizinkan kita untuk mendefinisikan suatu directive tertentu sesuai dengan kebutuhan. Pengaruhnya adalah berupa implementasi dari directive yang telah didefinisikan tersebut. Namun apabila kompiler yang kita gunakan tidak mendukung `#pragma`, maka directive ini akan diabaikan. Berikut ini contoh penggunaan directive `#pragma`.

```
#pragma inline
#pragma noinline
```

11.7. Directive `#line`

Kompiler akan terus menjaga pencatatan terhadap nomor baris dari kode-kode program selama proses kompilasi. Kita juga dapat mengontrol nomor baris tersebut dengan cara mengubahnya sesuai kebutuhan yang kita inginkan. Untuk melakukan hal tersebut kita dapat menggunakan directive `#line`. Directive ini akan mengubah nomor baris berikutnya menjadi nomor baris yang didefinisikan. Untuk memahaminya, coba anda perhatikan contoh sintak di bawah ini. (*Catatan: nomor yang terdapat pada bagian kiri program hanya untuk menunjukkan nomor baris saja dan tidak ikut diproses*)

```
1: #inlcude <stdio.h>
2:
3: int main() {
4:   #line 20
5:   /* Ini baris yang akan diganti dengan nomor baris 20 */
6:   printf("Baris ini terdapat ada baris ke-%d", __LINE__);
7:   return 0;
```

```
8: }
```

Anda bingung dengan kehadiran makro __LINE__ yang digunakan di atas? Itu wajar, namun Anda tidak perlu cemas karena materi tersebut akan kita bahas pada sub bab selanjutnya dalam bab ini (*Lihat sub bab 11.9 – Makro yang Telah Didefinisikan dalam Bahasa C*). Yang perlu Anda perhatikan di sini adalah penulisan directive `#line` yang terdapat pada baris ke-4 dan nilai yang didefinisikan adalah nilai 20. Maksud dari sintak tersebut adalah menggantikan nomor baris berikutnya (yaitu baris ke-5) menjadi baris ke-20. Dengan demikian, apabila program di atas dijalankan, maka hasil yang akan diberikan dalah seperti berikut.

```
Baris ini terdapat pada baris ke-21
```

Kenapa 21? Bukan 6? Jawabnya adalah karena baris ke-5 dari kode program di atas telah dianggap sebagai baris ke–20 sehingga baris ke-6 pun akan dianggap sebagai baris ke-21.

11.8. Directive `#error`

Directive ini digunakan untuk menampilkan pesan kesalahan yang terdapat pada saat proses kompilasi. Berikut ini bentuk umum dari penggunaan directive `#error`.

```
#error pesan_kesalahan
```

Perlu diperhatikan bahwa dalam menuliskan pesan kesalahan kita tidak perlu menuliskan tanda kutip ganda. Untuk lebih memahaminya, perhatikan contoh program di bawah ini yang akan menunjukkan penggunaan directive `#error`.

```
#include <stdio.h>

#define MAX 200

int main() {
  #if MAX > 100
    #error Nilai maksimal yang diizinkan adalah 100
  #endif
  return 0;
}
```

Pada saat proses kompilasi dari program di atas, maka kompiler akan menampilkan pesan kesalahan, yaitu dengan teks “`Nilai maksimal yang diizinkan adalah 100`”.

11.9. Makro yang Telah Didefinisikan dalam Bahasa C

Dalam bahasa C telah disediakan makro-makro yang siap pakai. Makro ini sering dinamakan dengan *predefined macro*. Makro-makro tersebut adalah *__FILE__*, *__LINE__*, *__DATE__*, *__TIME__* dan *__STDC__*.

11.9.1. Makro __FILE__

Makro ini berfungsi untuk mendapatkan nilai berupa string yang merupakan nama file dari kode program yang kita tulis. Sebagai contoh, kita memiliki kode program yang disimpan ke dalam file coba.c. Adapun sintak dari program tersebut adalah sebagai berikut.

```
/* Nama file : coba.c */

#include <stdio.h>

int main() {

  /* Menggunakan makro __FILE__ */
  printf("Nama file dari kode program ini adalah : %s",
         __FILE__);

  return 0;
}
```

Hasil dari program di atas adalah sebagai berikut.

```
Nama file dari kode program ini adalah : coba.c
```

Bagi beberapa kompiler, makro __FILE__ ini akan menampilkan nama file beserta direktorinya, misalnya **C:\budi\C\coba.c**.

11.9.2. Makro __LINE__

Makro ini digunakan untuk mengambil nilai berupa bilangan bulat (integer) yang merupakan nomor baris tertentu dari kode program yang kita tulis. Sebagai contoh perhatikan kode program berikut. (*catatan: no urut yang terdapat pada bagian kiri sintak program hanyalah sebagai nomor baris dan tidak ikut dikompilasi*)

```
1: #include <stdio.h>
2:
3: int main() {
4:
5:   printf("Nomor baris dari baris ini adalah : %d", __LINE__);
6:
```

```
7:   return 0;
8: }
```

Program di atas akan memberikan hasil sebagai berikut.

```
Nomor baris dari baris ini adalah : 5
```

Mungkin Anda akan bertanya, kenapa 5? Jawabnya adalah karena makro `__LINE__` dituliskan pada baris ke-5. Sebagai bukti dari pernyataan tersebut, marilah kita modifikasi program di atas dengan sintak berikut.

```
1: #include <stdio.h>
2:
3: int main() {
4:   int x = __LINE__;
5:   printf("Nomor baris dari baris ini adalah : %d", x);
6:
7:   return 0;
8: }
```

Sekarang hasil yang akan diberikan oleh program menjadi seperti di bawah ini.

```
Nomor baris dari baris ini adalah : 4
```

Hal ini disebabkan karena makro `__LINE__` dituliskan pada baris ke-4.

11.9.3. Makro `__DATE__`

Makro ini digunakan untuk mendapatkan tanggal yang sedang aktif dalam sistem komputer kita dengan menampilkannya ke dalam nilai bertipe string dan dengan format tanggal yang sudah ditentukan, yaitu "**Mmm dd yyyy**". Berikut ini contoh program yang akan menunjukkan penggunaan makro `__DATE__`.

```
#include <stdio.h>

int main() {

  printf("Sekarang tanggal : %s", __DATE__);

  return 0;
}
```

Hasil dari program di atas adalah sebagai berikut.

```
Sekarang tanggal : May 29 2004
```

11.9.4. Makro __TIME__

Makro ini berguna untuk mendapatkan waktu yang sedang aktif pada sistem komputer kita dan menampilkannya dalam bentuk string. Adapun format waktu yang akan ditampilkan tersebut adalah hh:mm:ss. Berikut ini contoh program yang menunjukkan penggunaan makro __TIME__.

```
#include <stdio.h>

int main() {

  printf("Sekarang jam : %s", __TIME__);

  return 0;
}
```

Hasil dari program di atas adalah sebagai berikut.

```
Sekarang jam : 18:17:59
```

11.9.5. Makro __STDC__

Kita dapat mendapatkan informasi tentang kompiler C yang kita gunakan, apakah merupakan kompiler standar atau bukan. Makro __STDC__ ini akan mengembalikan nilai berupa nilai konstan 1 apabila kompiler yang kita gunakan merupakan kompiler C standar. Berikut ini contoh program yang menunjukkan penggunaan makro __STDC__. (*catatan : penulis menggunakan kompiler MinGW (untuk Windows) serta cc dan gcc (untuk Linux)*).

```
#include <stdio.h>

#define STANDAR 1

int main() {

  if (__STDC__ == STANDAR) {
    printf("Kompiler C standar");
  } else {
    printf("Belum merupakan kompiler C standar");
  }
```

```
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Kompiler C standar
```

Bab

12

Implementasi C untuk

Pemrograman Paralel dan Serial

Pendahuluan

Jurang kesenjangan sering terjadi antara penguasaan pemrograman dan pengetahuan tentang piranti elektronik yang dihubungkan ke komputer. Hal ini mengakibatkan terjadinya kesulitan dalam menggunakan teknik pemrograman untuk mengakses perangkat keras. Pemrograman port (*port programming*) adalah suatu perintah program yang digunakan untuk membaca suatu register yang ditugaskan secara spesifik untuk membaca piranti elektronik tertentu yang dihubungkan ke personal komputer (PC), dalam hal ini akan diacu menggunakan IBM-PC. Kendala yang sering terjadi adalah sedikitnya pengalaman langsung dalam pendesainan dan pembuatan piranti elektronika sederhana sebagai media awal yang akan digunakan sebagai alat bantu dalam melatih kemampuan pemrograman port. Agar Anda dapat menguasai teknik pemrograman port ini, Anda perlu pengalaman praktis dalam bentuk proyek kecil pembuatan perangkat lunak untuk mengakses port.

Tumbuh pesatnya perkembangan device-device yang mampu dihubungkan ke komputer mengakibatkan sebagian orang melihatnya sebagai suatu kerumitan. Perusahaan pemroduksi device biasanya sudah menyediakan driver atau suatu paket software yang mampu mendeteksi dan memfasilitasi pembacaan perangkat keras yang dipasangkan ke PC yang ditawarkannya baik untuk multimedia, aplikasi kontrol industri maupun untuk alat pengukuran saintifik. Akan tetapi tidak semua pemroduksi device tersebut menyediakan driver, kadang kita hanya diberi tahu tentang standar komunikasi yang digunakannya saja. Oleh kerena itu, sebagai seorang programmer, pemahaman tentang pemrograman port sangatlah diperlukan. Aplikasi pemrograman port sangat luas seperti pada robotik, industri, instrumentasi dan sistem kontrol. Demikian luasnya perkembangan pemanfaatan PC dan juga interaksi dengan lingkungan di luar PC membuat motivasi dan ketertarikan tersendiri untuk menekuninya.

Berdasarkan pertimbangan inilah penulisan buku ini akan menyertakan bahasan mengenai pemrograman port, mulai dari pemberian teknik pengalamatan port (sebagai pemahaman dasar yang perlu diketahui) dan dilanjutkan dengan pembuatan proyek sederhana (seperti penampilan LED) hingga yang agak rumit (seperti proses pengiriman karakter ke LCD). Pekerjaan-pekerjaan sederhana semacam ini pula telah diuji-cobakan di laboratorium Instrumentasi Elektronika Jurusan Fisika Universitas Padjadjaran dalam rangka mengembangkan ketertertarikan awal terhadap bidang ini.

Pembahasan bab ini akan diawali dengan pengantar teori tentang komunikasi paralel sesuai dengan standar komunikasi data IEEE 1284. Selanjutnya kita akan mempelajari juga bagaimana medesain modul baik hardware maupun pemrograman port yang akan dibahas dengan dua penekanan kategori proses, yaitu: pegiriman data melalui I/O ke luar komputer dan pengambilan data dari luar ke komputer. Untuk pengiriman data dari komputer ke luar akan dimulai dari yang paling sederhana seperti menyalakan lampu LED melalui port LPT, menggunakan port untuk membuat musik, mengatur gerak motor DC & Stepper. Kemudian kita juga akan membahas mengenai sistem komunikasi dua arah pada pemrograman port, yaitu pengiriman data ke LCD matrik 2x16.

Input/Output (I/O) PC

Kita perlu untuk berkomunikasi dengan komputer dan sebaliknya komputer juga perlu berkomunikasi dengan kita dan apapun yang dihubungkan ke komputer. Secara konsep, perpindahan sinyal listrik ke perangkat komputer disebut input dan pergerakan sinyal dari komputer ke perangkat luar disebut output. Kedua proses ini biasanya dibahas secara bersamaan yang secara sederhananya disebut sebagai input/output (I/O). Perangkat elektronik yang ada di dalam komputer dan berkomunikasi dengan perangkat lain yang masih ada di dalam komputer harus menggunakan sistem I/O. Sistem I/O juga digunakan jika Anda berkomunikasi dengan komputer melalui keyboard dan mouse atau komputer berkomunikasi dengan kita melalui monitor dan printer.

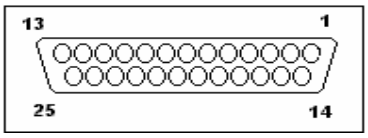
Menghubungkan dua perangkat elektronik artinya memindahkan sinyal listrik antara dua perangkat tersebut. Walaupun kelihatannya cukup sederhana, hanya dengan menghubungkan dua perangkat secara elektronik, namun tidaklah cukup untuk melaksanakan proses I/O. Untuk itu perangkat yang dihubungkan hendaknya mempunyai kecocokan dalam hal bagaimana cara sinyal listrik tersebut dikomunikasikan, diproses dan digunakan. Bayangkan seandainya Anda ingin menekan huruf "M" pada keyboard ternyata komputer berfikir sinyal ini untuk "L". Kesalahan seperti ini tidak boleh terjadi. Oleh karena itu proses I/O adalah hal yang sangat penting dalam teknologi komputer. Hal lain yang juga menjadi perhatian dalam I/O adalah kecepatan dalam mentransfer data. Sebagai contoh lain yang lebih sederhana adalah port yang digunakan untuk pemasangan keyboard dan mouse. Secara fisik, port tersebut sama, namun kita harus memasangnya secara benar sehingga kedua alat tersebut dapat beroperasi sesuai dengan fungsinya. Inilah yang dimaksud dengan kecocokan dalam komunikasi sinyal listrik.

Mengenal Transfer Data pada Port Paralel

Untuk memudahkan Anda dalam memahami proses transfer data pada port paralel, maka di sini kita akan mengambil sebuah contoh, yaitu cara kerja printer.

Printer dihubungkan ke komputer melalui kabel 25 pin yang dihubungkan dengan konektor tipe betina (*female*) DB-25 (sering juga disebut sebagai metode pengkabelan centronics) yang dapat dilihat pada Gambar 12.1 dan Tabel 12.1. Kabel koneksi ini mempunyai 8 pin data, 8 pin ground, 5 pin status dan 9 pin kontrol. Sekarang mari kita pelajari secara sederhana bagimana komputer berkomunikasi dengan printer. Salah satu kanal yaitu pin 13 membawa sinyal dari printer yang mengatakan kepada komputer bahwa printer online dan siap untuk menerima data. Jika komputer telah siap dengan pekerjaan yang akan dicetak, komputer mengirimkan data melalui pin 2 sampai dengan

pin 9 dalam bentuk tegangan. Tegangan 5 Volt merepresentasikan biner 1 dan apabila tegangannya rendah mendekati nol merepresentasikan biner 0. Setelah data siap, selanjutnya komputer mengirimkan sinyal strobe melalui pin 1 untuk memberitahu printer membaca tegangan yang ada di pin 2 sampai dengan 9. Sinyal strobe ini biasanya berakhir dalam 1 mikrodetik. Jika proses berjalan lancar, maka printer membaca tegangan-tegangan pada pin 2-9 dan menterjemahkannya kembali menjadi bit-bit. Printer selanjutnya memberi informasi ke komputer melalui pin 10 bahwa data telah diterima dan siap untuk pengiriman data selanjutnya. Proses ini dapat berulang ratusan bahkan ribuan kali per detik. Jika kesalahan pengiriman terjadi maka printer akan memberitahu komputer dengan mengirimkan sinyal error melalui pin 15.



Gambar 12.1 DB25 sebagai konektor port LPT parallel.

Tabel 12.1. Pin-pin pada konektor DB-25 dan Fungsi Registernya.

No Pin DB-25	No Pin Mode Centronic	Fungsi Register
1	1	Kontrol
2	2	Data
3	3	Data
4	4	Data
5	5	Data
6	6	Data
7	7	Data
8	8	Data
9	9	Data
10	10	Status
11	11	Status
12	12	Status
13	13	Status
14	14	Kontrol
15	32	Status
16	31	Kontrol
17	36	Kontrol
18-25	19-30	(Ground)

Ukuran data dari motherboard yang dipindahkan menuju printer adalah 32 bit, sedangkan jalur data yang tersedia pada port adalah 8 bit. Sebagai contoh, pada saat kita akan melakukan pencetakan naskah atau dokumen dengan ukuran tertentu, maka prosesor akan mengirimkan data tersebut dengan ukuran 32 bit. Sedangkan kita tahu bahwa jalur data yang tersedia pada port adalah 8 bit sehingga data-data yang dikirimkan dengan ukuran 32 bit itu juga perlu dipecah lagi sesuai dengan jalur data yang ada, yaitu menjadi empat bagian data yang masing-masing berukuran 8 bit. Proses pemecahan seperti ini dilakukan oleh port paralel.

Port paralel ini juga bertangung jawab terhadap urutan data yang dipecah dan menjadikannya sebagai sinyal listrik yang kemudian ditransmisikan ke kabel paralel. Selain itu, port paralel juga akan menerima dan mengirimkan sinyal yang diberikan oleh printer. Misalnya jika printer kehabisan kertas, maka printer akan mengirimkan sinyal melalui pin 12 dan port paralel yang memantau pin 12 ini akan bereaksi dengan menghentikan pengiriman data dan melaporkan terjadinya kondisi error ini kepada user.

Awalnya printer I/O didesain dalam satu arah yang artinya data selalu dikirim ke printer dan printer tidak pernah mengirimkan informasi ke komputer. Dengan berkembangnya teknologi printer, scanner dan piranti lain yang terhubung melalui port paralel, dimana dibutuhkan suatu komunikasi dua arah antara komputer dan piranti tersebut, maka komputer harus memfasilitasi penerimaan data dari piranti luar. Untuk menangani operasi I/0 ini dibuatlah transfer data port paralel dwi-arah. Artinya, teknologi pendesainan printer juga harus memungkinkan adanya transfer data ke komputer dan komputer juga harus mampu membaca data ini yang dikirim melalui kanal tertentu.

Agar terdapat standarisasi dalam proses tranfer data, maka pada tahun 1991 diadakanlah suatu pertemuan antara pembuat printer dan komputer, seperti Texas Instrument, IBM, Lexmark dan juga yang lainnya. Hasil dari pertemuan ini adalah adanya kesepakatan untuk membentuk Network Printing Allience (NPA). NPA ini kemudian bekerja dalam menetapkan aturan-aturan yang harus diikuti dalam pembuatan perangkat keras agar tidak terjadi ketidak-kompatibelan antara berbagai peralatan yag berbeda. Agar lebih diakui masyarakat luas dalam penggunaan aturan-aturan ini, NPA mengajukannya ke Institue of Electric and Electronic Engineer (IEEE) yang akhirnya disetujui sebagai keputusan IEEE 1284 pada tahun 1994. Pada IEEE 1284 ini terdapat metode pensinyalan standar antarmuka paralel dwi-arah. Standar IEEE 1284 ini masih kompatibel dengan port paralel sebelumnya. Standar ini mendefinisikan lima mode operasi antara lain: *Compatibility Mode*, *Nibble Mode*, *Byte Mode*, *EPP Mode*, dan *ECP Mode*. Pada buku ini hanya akan disinggung sedikit mengenai mekanisme pengalamatan port paralel. Jika Anda membutuhkan keterangan lebih lengkap tentang port paralel ini silahkan mengunjungi situs http://www.senet.com.au/~cpeacock yang membahas tentang *Interfacing The Standard Parallel Port*. Selain menambahkan komunikasi dwi-arah, port yang ditetapkan pada standar ini juga mampu menangani transfer data hingga 1 Mega Bit per detik dan memungkinkan panjang kabel hingga maksimum 10 meter.

Dengan berkembangnya teknologi perangkat lunak yang memudahkan pemrosesan, pengolahan dan pembantu analisa data merangsang setiap orang untuk menghubungkan perangkat kerasnya ke komputer. Oleh karena itu, port paralel tidak hanya digunakan untuk menghubungkan komputer ke printer, tetapi juga biasa digunakan untuk menghubungkan atau sering disebut mengantarmuka suatu alat atau perangkat keras lain ke komputer. Perangkat keras ini biasanya berupa desain sendiri dengan tujuan desain dan sistem tersendiri ataupun alat yang dibeli dari pabrik yang telah memiliki spesifikasi standar IEEE 1284. Tentu saja port paralel bukannya satu-satunya cara berkomunikasi dengan komputer. Masih terdapat cara lain untuk melakukan komunikasi tersebut, yaitu dengan melalui komunikasi serial seperti RS232 atau USB.

- **Standar Komunikasi Paralel menurut IEEE 1284**

Seperti yang telah dikemukakan sebelumnya, keputusan IEEE 1284 mengeluarkan lima mode operasi yaitu *Compatibility Mode*, *Nibble Mode*, *Byte Mode*, *EPP Mode*, dan *ECP Mode*. Pembuatan standar ini bertujuan untuk mendesain driver dan device baru yang kompatibel satu dengan yang lainnya sesuai dengan *Standard Parallel Port* (SPP). Mode Compatibility, Nibble & Byte merupakan standar suatu perangkat keras yang aslinya tersedia dalam bentuk kartu port paralel, sedangkan mode EPP dan ECP membutuhkan perangkat keras tambahan yang memungkinkan untuk melakukan pekerjaannya lebih cepat. Transfer data dengan cara *Compatibility Mode* (sering juga disebut mode Centronics) hanya dapat mengirimkan dengan kecepatan sekitar 50 kilobyte per detik dan bisa juga sampai maksimum dengan kecepatan 150 kilobyte per detik. Jika kita ingin komputer mampu menerima data maka modenya harus diubah ke mode Nibble ataupun Byte. Mode Nibble dapat diberi masukan satu nibble (4 bit) dari suatu sistem rangkaian di luar komputer. Mode Byte menggunakan fitur paralel dwi-arah untuk memasukkan data satu byte (8 bit) data ke komputer.

Extended Parallel Ports (EPP) dan *Enhanced Parallel Ports* (ECP) menggunakan tambahan perangkat keras untuk membuat dan mengatur *handshaking*, yaitu istilah untuk menjelaskan proses komunikasi antara komputer dan perangkat lain.

Berikut ini akan diberikan contoh penggunaan mode-mode di atas pada alat yang sering kita gunakan setiap hari yaitu printer seperti yang dicontohkan pada bagian awal dari pembahasan ini. Langkah-langkah atau algoritma pemrograman menggunakan *Compatibility Mode* untuk mengirim atau mengeluarkan data satu byte ke printer yang bisa mengandung informasi teks, gambar atau apa saja adalah sebagai berikut:

2. Menulis atau mengirim satu byte ke port data

3. Memeriksa apakah printer sedang sibuk melalui pin 11. Jika printer sedang sibuk maka printer tersebut tidak akan menerima data sehingga data apapun yang dikirimkan akan hilang

4. Memberi nibble rendah (Strobe) ke pin 1 untuk memberitahu printer bahwa ada data yang valid pada pin 2-9

5. Setelah menunggu 5 mikrodetik dari saat memberi strobe nibel rendah, beri strobe atau pin 1 tinggi

Langkah-langkah perulangan semacam ini akan membatasi kecepatan kerja port yang semestinya, sehingga dapat kita anggap sebagai suatu kelemahan. Untuk mengatasi kelemahan tersebut maka diperkenalkanlah penggunaan mode port EPP dan ECP. Port EPP dan ECP mengatasi hal ini dengan membiarkan hardware memeriksa apakah printer sibuk atau tidak, kemudian mengirim suatu *strobe* atau *handshaking* yang sesuai dengan keadaannya. Artinya hanya satu instruksi I/O yang dibutuhkan untuk melakukan pengiriman data tersebut dan ini akan meningkatkan kecepatan kerja port. Port-port ini dapat mengeluarkan data sekitar 1-2 megabyte per detik. Disamping itu keuntungan lain penggunaan port ECP adalah penggunaan pin-pin DMA (*Direct Memory Access*) dan buffer FIFO (*First In First Out*) yang memungkinkan data dipindahkan tanpa menggunakan instruksi I/O. Peran ECP semacam ini tidak akan dibahas dalam buku ini. Untuk itu, apabila Anda membutuhkan informasi lebih lengkap mengenai mode EPP dan ECP, silahkan kunjungi situs yang telah diberikan di atas.

Dalam sistem komputer IBM-PC terdapat port-port standar yang dapat digunakan sebagai fasilitas I/O, dimana port tersebut digunakan untuk menghubungkan komputer yang dalam bahasan ini akan menekankan pemakaian IBM-PC dengan piranti luar. Port paralel LPT adalah port standar, umumnya terdiri dari 25 pin yang digunakan untuk proses komunikasi data antara IBM-PC dengan piranti luar. Port ini memungkinkan kita memiliki masukan hingga 8 bit atau keluaran hingga 12 bit pada saat yang bersamaan, dengan hanya membutuhkan rangkaian eksternal sederhana untuk melakukan suatu tugas tertentu.

- **Pengalamatan Port Paralel LPT**

Pada umumnya port paralel memiliki tiga alamat fisik yang bisa digunakan, sebagaimana ditunjukkan pada Tabel 12.2. Alamat fisik yang digunakan untuk keperluan ini disebut juga register. Register yang ada pada port paralel sendiri terdiri atas Register Data, Register Status dan Register Kontrol. Bagi para programer hardware, pengertian register ini sangat penting karena sebenarnya hanya dengan bermain pada data di register tertentu, programmer tersebut sudah dapat membuat program-program yang digunakan untuk mengakses piranti luar. Artinya, programmer tersebut sebenarnya tidak perlu memahami secara detail mengenai spesifikasi dari piranti luar yang dia gunakan, tapi yang perlu dia ketahui hanyalah alamat fisik dari port paralel yang digunakan.

Alamat register 3BCH pertama kali diperkenalkan sebagai alamat port paralel pada kartu-kartu video (*video card*) lama, tapi kemudian alamat ini sempat menghilang ketika port paralel dicabut dari kartu-kartu video. LPT1 biasanya memiliki alamat dasar 378h, sedangkan LPT2 pada 278h.

Pada port parallel terdapat 17 pin yang digunakan sebagai jalur I/O. Komputer memiliki register dengan 8 bit data per alamat. Untuk menyesuaikan dengan kebutuhan komunikasi atau *handshaking* maka dibutuhkan 3 alamat fisik untuk setiap alamat port paralel. Kebutuhan register yang dimaksud pada proses handshaking adalah satu Register Data (untuk proses pengiriman dan penerimaan data), satu Register Status (untuk mengetahui keadaan piranti luar), dan satu Register Kontrol (untuk melakukan aksi terhadap status tertentu pada piranti luar). Bit-bit pada register itu tidak semuanya dipakai, jumlah register yang dipakai Register Data (8 bit), Register Status (5 bit) dan Register Kontrol (4 bit). Alamat register tersebut berurutan, yakni jika alamat dasar untuk LPT1 adalah 0378h, maka alamat 0378 untuk Register Data, 0379 untuk Register Status dan 037A untuk Register Kontrol. Alamat demikian disebut juga offset yaitu jarak dari alamat awal. Register Status hanya memakai 5 bit dari 8 bit yang ada, register kontrol hanya menggunakan 4 bit dari 8 bit, sedangkan register data menggunakan semua bit yang ada. Selengkapnya pemakaian alamat 8 bit untuk masing-masing register terdapat ada Tabel 12.3-12.5.

Tabel 12.2. Table Alamat Port

Alamat	Keterangan
38Ch-3BFh	Port pilihan yang dikontrol melalui BIOS. Port ini tidak mendukung alamat ECP

378h-37Fh	Alamat untuk LPT1
278h-27Fh	Alamat Untuk LPT2

Berikutnya kita akan mempelajari bagimana cara pengalamatan port ini. Hal yang paling penting untuk diketahui sebelum melakukan pembuatan pemrograman port adalah penentuan alamat port dasar yang dipilih sesuai dengan piranti luar yang akan kita hubungkan. Alamat port dasar tersebut dapat dipilih dari salah satu alamat yang tertera pada Table 12.2. Sebagai contoh, apabila kita menggunakan piranti luar pada LPT1 maka alamat port yang harus digunakan adalah 378h-37Fh. Berdasarkan alamat port tersebut kita baru dapat menentukan alamat ketiga tipe register, yaitu: Register Kontrol, Register Data dan Register Status seperti yang ditunjukkan pada Tabel 12.3 sampai 12.5. Selain itu, Anda juga bisa mengetahui alokasi bit-bit untuk masing-masing register yang ditabulasikan pada Tabel 12.3-12.5.

Tabel 12.3 Pemetaan Bit-bit pada Register Data.

Offset	Nama	Read/write	Bit ke	Properti
Alamat dasar +0	Port Data	Write *)	Bit 7	Data 7
			Bit 6	Data 6
			Bit 5	Data 5
			Bit 4	Data 4
			Bit 3	Data 3
			Bit 2	Data 2
			Bit 1	Data 1
			Bit 0	Data 0

Tabel 12.4 Pemetaan Bit-bit pada Register Status

Offset	Nama	Read/write	Bit ke	Properti
Alamat dasar +1	Port Status	Read Only	Bit 7	Busy
			Bit 6	Ack
			Bit 5	Paper Out
			Bit 4	Selct In
			Bit 3	Error
			Bit 2	IRQ
			Bit 1	-
			Bit 0	-

Tabel 12.5 Pemetaan Bit-bit pada Register Kontrol

Offset	Nama	Read/write	Bit ke	Properti

Alamat dasar +2	Port Kontrol	Read/write	Bit 7	-
			Bit 6	-
			Bit 5	Enable Bi-di
			Bit 4	Enable IRQ
			Bit 3	Select
			Bit 2	Initial
			Bit 1	Auto
			Bit 0	Strobe

Berdasarkan pemahaman terhadap Port Parallel dan prinsip dasar pengalamatan port, Anda dapat memulai suatu proyek kecil dengan menggunakan rangkaian eksternal sederhana yang akan dihubungkan ke port parallel PC. Rangkaian sederhana itu seperti yang ditunjukkan pada Gambar 12.2 dengan menggunakan Port Paralel LPT1 terhubung melalui DB-25. Rangakain ini digunakan untuk melatih penggunaan pemrograman port untuk mengeluarkan data dari PC ke piranti eksternal.

Untuk pembahasan selanjutnya, kita akan selalu menganggap bahwa kita menggunakan LPT1 dengan alamat dasar 378h pada Standard Parellel Port (SPP). Bedasarkan teknik pengalamatan pemrograman port yang telah dijelaskan di atas, alamat port LPT yang bersangkutan menjadi seperti pada Tabel 12.6-12.8 yaitu Register Data dengan alamat dasar +0 yaitu pada alamat 0378h , Register Status dengan alamat dasar +1 yaitu pada alamat 0379h dan Register Kontrol dengan alamat dasar +2 yaitu pada alamat 037Ah.

Tabel 12.6 Register Data dengan alamat dasar +0 yaitu pada alamat 0378h.

Offset	Nama	Read/write	Bit ke	Properti
Alamat dasar+0 Atau 0378h	Port Data	Write *)	Bit 7	Data 7
			Bit 6	Data 6
			Bit 5	Data 5
			Bit 4	Data 4
			Bit 3	Data 3
			Bit 2	Data 2
			Bit 1	Data 1
			Bit 0	Data 0

Tabel 12.7. Register Status dengan alamat dasar +1 yaitu pada alamat 0379h.

Offset	Nama	Read/write	Bit ke	Properti
Alamat dasar +1 Atau 0379h	Port Status	Read Only	Bit 7	Busy
			Bit 6	Ack
			Bit 5	Paper Out
			Bit 4	Selct In
			Bit 3	Error
			Bit 2	IRQ
			Bit 1	-
			Bit 0	-

Tabel 12.8. Register Kontrol dengan alamat dasar +2 yaitu pada alamat 037Ah.

Offset	Nama	Read/write	Bit ke	Properti
Alamat dasar+2	Port Kontrol	Read/write	Bit 7	-

			Bit 6	-
			Bit 5	Enable Bi-di
			Bit 4	Enable IRQ
			Bit 3	Select
			Bit 2	Initial
			Bit 1	Auto Linefeed
			Bit 0	Strobe

Sekarang Anda telah siap membuat program port dengan mengetahui dan menentukan register mana yang dipakai, bit mana yang harus dibaca dan kontrol apa yang harus dilakukan untuk piranti luar. Berikut ini adalah cara mengirimkan data ke port paralel yang telah Anda tentukan. Bentuk umum perintah pemrograman port adalah seperti berikut:

```
outportb(alamat port, nilai);
```

dengan *alamat port* adalah alamat yang ingin dituju dan *nilai* adalah data yang ingin dikirimkan. Sedangkan bentuk umum untuk membaca masukan dari alamat port tertentu adalah seperti berikut:

```
inportb(alamat port);
```

Sebagai contoh, apabila kita menggunakan LPT1 artinya Register Data terdapat pada alamat 378h, Register Status pada alamat 379h dan Register Kontrol pada alamat 037Ah. Misalnya, sekarang kita ingin mengirimkan data 10h melalui Register Data ke piranti luar (misal: LED), maka perintahnya adalah:

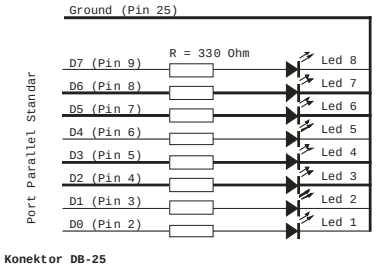
```
outportb(0x378,0x10)
```

Sebaliknya apabila kita ingin mengambil masukan dari piranti luar melalui Register Status, maka perintahnya adalah

```
inportb(0x379)
```

❑ Aplikasi Pemrograman Port C untuk Tampilan 8 buah LED

Pada aplikasi ini sengaja dibuat rangkaian eksternal yang paling sederhana untuk memahami kerja dan cara membuat pemrograman port LPT. Rangkaian eksternal ini dapat dibuat hanya dengan menggunakan 8 buah LED dan juga dipasang 8 buah resistor 330 Ohm secara seri dengan LED agar arus tidak terlalu besar ke LED. Ground untuk rangkaian ini diambil dari LPT pada pin 25. Desain rangkaian yang dibuat menggunakan LED katoda dan dihubungkan ke Ground sering disebut common katoda.



Gambar 12.2 Tampilan LED dengan konfigurasi Common Katoda

Berikut ini adalah contoh program yang digunakan untuk menyalakan LED.

```
/* ---------------------------------------------------------- */
/*    Modul Pemrograman Port Untuk LED                        */
/*    D0-D7 dihubungkan ke Pin2-Pin9 pada konektor DB25       */
/*    Ground dihubungkan ke Ground pada Pin 25                */
/*    konektor DB25                                           */
/*---------------------------------------------------------- */


#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <dos.h>

int main(void) {
  clrscr();

  unsigned int far *h;

  outportb(0x378,0x10);

  return 0;
}
```

Perintah `outportb(0x378,0x20)` artinya adalah kita mengirimkan data heksa 10 atau (0000 1010)biner ke alamat register 378h yang diteruskan ke pin 2-9 sehinga LED yang menyala adalah LED 2 dan LED 4.

❑ Aplikasi Pemrograman Port C untuk Musik

IBM PC biasanya mengunakan speaker untuk bunyi sebagai tanda kondisi error. Lebih dari itu, kita juga dapat menggunakan speaker ini untuk menghasilkan irama musik. Metode ini menghasilkan timbre yang sangat menarik, apalagi bila digabungkan dengan efek grafik suatu animasi kita akan mendapatkan sensasi yang menarik.

Untuk menghasilkan irama, kita harus membuat pulsa dan mengirimkannya ke speaker. Jika arus diberikan ke koil speaker, secara fisik membran speaker bergetar (mengeluarkan bunyi) dan jika arusnya dimatikan maka speaker akan diam (tidak berbunyi). Seandainya kita memberikan sederetan sinyal on atau off ke speaker misalnya 100 kali per detik, artinya kita telah membuat irama dengan frekuensi 100 Hz.

Untuk menghidupkan speaker pada IBM PC, kita dapat melakukannya dengan mengirimkan nilai 2 ke port yang beralamat 61h. Sedangkan untuk mematikan speaker kita dapat mengirimkan nilai 0 ke port 61h. Berikut ini kode yang diperlukan untuk melakukan hal tersebut.

```
outport(0x61h, 0x02h);      /* untuk menghidupkan */
outport(0x61h, 0x0h);       /* untuk mematikan */
```

Di sini, kita mencoba program yang mengkonversi apa yang anda ketik di keyboard menjadi tone musik dengan berbagai variasi pitch (pola nada). Kita juga dengan mudah dapat mengganti fungsi `play()` untuk memasukkan durasi pitch yang diinginkan. Silahkan Anda bermain tone dengan keyboard.

```
/* ----------------------------------------------------------*/
/*   Modul Pemrograman Port Untuk Musik Menggunakan Speaker */

/*   Mengaktifkan speaker dengan alamat 61h                  */
/*   Data 2 untuk on dan 0 untuk off                         */
/*---------------------------------------------------------- */

#include <stdio.h>
#include <string.h>
#include <dos.h>

int main (void)
{
  int note, durasi;
  durasi=10;
  do {
    note=getchar();
    play(note, durasi);
  } while (note!='q');
}

play(int note,int d) {
  int t, tone;
  d = d+1000/note;
  for (;d;d--){
    tone = note;
    outport(0x61,2);   /* menghidupkan speaker*/
    outport(0x61,0);   /* mematikan speaker*/
    for (;tone;--tone);
```

```
  }
}
```

❑ Aplikasi Pemrograman Port C untuk Menggerakkan Motor DC

Motor DC (*Direct Current*) adalah motor yang menggunakan sumber tegangan searah. Terdapat beberapa jenis motor DC yang tersedia, diantaranya adalah motor DC dengan koil medan dan motor DC dengan magnet permanen. Motor DC dengan koil medan membutuhkan arus yang lebih besar dari motor magnet permanen untuk mengatur medan magnet pada koil magnet. Oleh karena itu, koil medan mempunyai rentang kecepatan yang lebih luas dan torsi yang lebih besar. Motor DC magnet permanen tidak dapat mengubah besarnya medan magnet sehingga rentang torsi dan kecepatannya kecil.

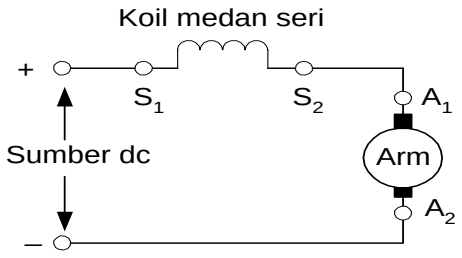
❑ Motor DC dengan Koil Medan

Berdasarkan cara merangkainya, Motor DC dengan koil medan ini dapat dibedakan menjadi tiga, yaitu: jenis seri, jenis shunt, dan jenis gabungan.

12.5.1.1. Motor DC Jenis Seri

Motor DC jenis seri terdiri dari medan seri yang diidentifikasi dengan S1 dan S2 yang terbuat dari sedikit lilitan kawat besar yang dihubungkan seri dengan jangkar yang diidentifikasi dengan A1 dan A2, seperti yang ditunjukkan pada Gambar 12.3. Jenis motor DC ini mempunyai karakteristik torsi awal dan variasi kecepatan yang tinggi. Ini berarti bahwa motor dapat memulai menggerakkan beban yang sangat berat, tetapi kecepatan akan bertambah bila beban turun. Motor DC seri dapat membangkitkan torsi awal yang besar karena arus yang sama yang melewati jangkar juga melewati medan. Jadi, jangkar memerlukan arus lebih banyak (untuk membangkitkan torsi lebih besar) dan arus ini juga melewati medan (untuk menambah kekuatan medan). Oleh karena itu, motor seri berputar cepat dengan beban ringan dan berputar lambat pada saat beban ditambahkan. Sifat istimewa terpenting dari motor DC seri adalah kemampuannya untuk memulai pergerakan pada beban yang sangat berat. Karena alasan itu, motor jenis ini sering digunakan pada elevator. Untuk membalik arah putaran motor DC seri, baliklah arah arus pada kumparan seri atau kumparan jangkar.

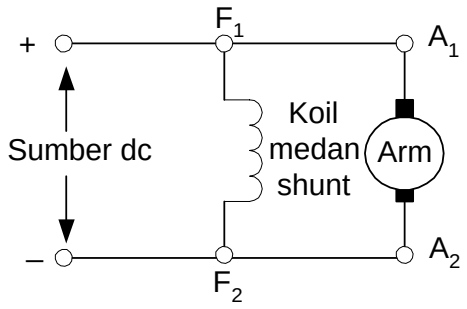
Jenis motor DC ini juga disebut motor universal karena dapat dioperasikan baik dengan arus searah maupun dengan arus bolak-balik. Motor DC akan terus berputar pada arah yang sama bila arus yang mengalir pada jangkar dan medan, dibalik secara bersamaan.



Gambar 12.3. Motor DC Jenis Seri

12.5.1.2. Motor DC Jenis Shunt

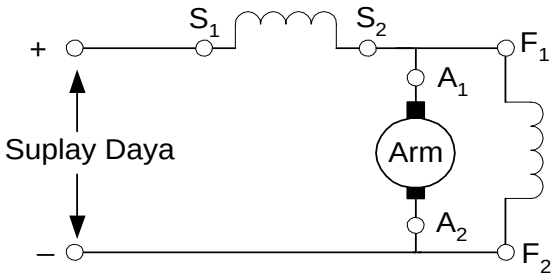
Pada motor DC jenis shunt, kumparan medan shunt (diidentifikasikan dengan F1 dan F2) dibuat dengan banyak lilitan kawat kecil, sehingga akan memiliki tahanan yang tinggi. Motor shunt mempunyai rangkaian jangkar dan medan yang dihubungkan secara paralel yang akan memberikan kekuatan medan dan kecepatan motor yang sangat konstan. Motor shunt digunakan pada suatu sistem yang memerlukan pengaturan kecepatan yang baik (konstan). Dengan menambah *rheostat* yang dipasang seri dengan rangkain medan shunt, kecepatan motor dapat dikontrol di atas kecepatan dasar. Kecepatan motor akan berbanding terbalik dengan dengan arus medan. Ini berarti motor shunt berputar cepat dengan arus medan rendah, dan bertambah pada saat arus ditambahkan. Motor shunt dapat melaju pada kecepatan tinggi yang berbahaya jika arus kumparan medan hilang.



Gambar 12.4. Motor DC Jenis Shunt

12.5.1.3. Motor DC jenis Gabungan

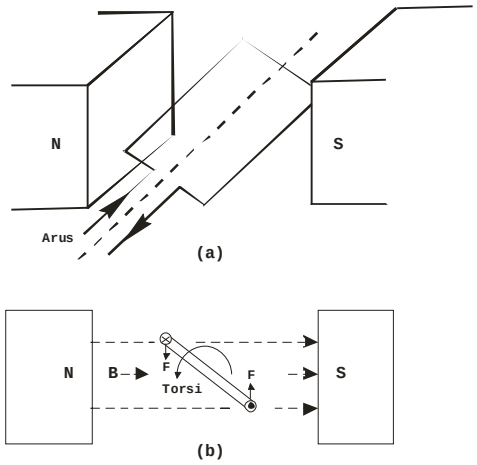
Motor DC jenis gabungan (*compound*) menggunakan lilitan seri dan lilitan shunt, yang umumnya dihubungkan dengan medan-medannya bertambah secara kumulatif. Hubungan dua lilitan ini menghasilkan karakteristik pada motor medan shunt dan motor medan seri. Kecepatan motor tersebut bervariasi lebih sedikit dibandingkan motor shunt, tetapi tidak sebanyak motor seri. Motor DC jenis gabungan juga mempunyai torsi awal yang agak besar – jauh lebih besar dibandingkan dengan motor shunt, tetapi sedikit lebih kecil dibandingkan motor seri. Keistimewaan dari cara penggabungan ini membuat motor jenis ini akan memberikan variasi penggunaan yang luas.



Gambar 12.5. Motor DC Jenis Shunt

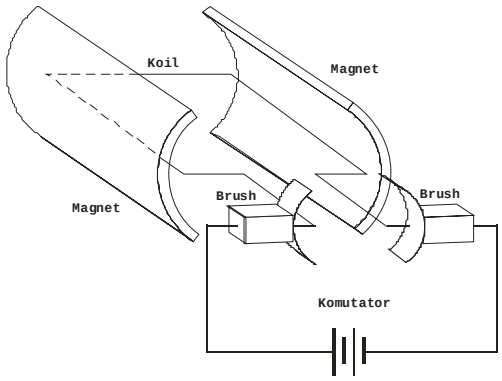
- **Motor DC Magnet Permanen**

Motor DC magnet permanen memiliki dua bagian, yaitu stator dan rotor. Stator adalah bagian yang diam sedangkan rotor adalah bagian yang ikut berputar. Perputaran rotor ini disebabkan karena adanya torka yang dibangkitkan oleh konduktor yang dialiri arus dalam medan magnet seperti terlihat dalam Gambar 12.6



Gambar.12.6. Torka yang dibangkitkan oleh arus yang mengalir dalam rotor

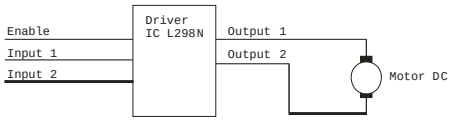
Berdasarkan gambar di atas, tampak bahwa motor akan berputar bolak balik karena setelah lengan berputar 180^o atau tegak lurus dengan arah medan magnet arah torsi berbalik. Untuk menghindari gerakan bolak balik ini maka arus yang diberikan harus dibalik pada saat lengan tegak lurus dengan medan magnet. Proses ini disebut *commutation* (penukaran arah arus listrik). Untuk mengubah arah arus tersebut dapat dilakukan dengan menggunakan sikat atau komutator. Gambar 12.7 berikut memperlihatkan cara kerja commutator dari motor DC magnet permanen.



Gambar 12.7. Cara kerja komutator

- **Pengaturan Gerak Motor DC**

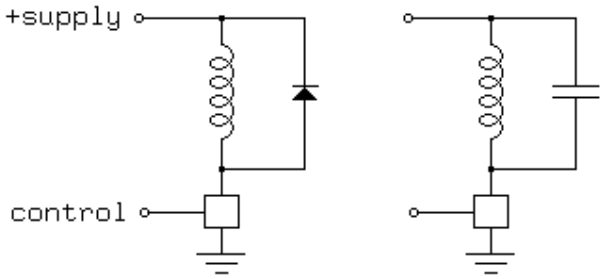
Salah satu diagram Blok untuk menggerakakan motor DC dengan driver IC L298N adalah seperti pada Gambar 12.8.



Gambar 12.8 Diagram Blok Pengaturan motor DC dengan Drriver IC L298N

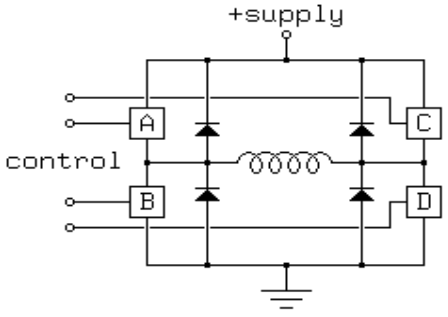
12.5.3.1. Driver Motor DC

Driver motor digunakan untuk mengontrol lamanya putaran motor dan arah putaran motor. Prinsip kerja dari driver ini hanya dengan melewatkan arus pada motor dan menghentikan arus yang melewati motor serta mengatur arah arusnya dengan menggunakan switch. Kumparan motor merupakan beban induktif, sehingga arus yang melewati kumparan motor tidak dapat dinyalakan dan dimatikan dengan segera. Ketika switch pengontrol kumparan motor terhubung maka arus akan naik dengan perlahan. Sedangkan pada saat switch pengontrol ini dilepas arus dari kumparan turun perlahan, akibatnya terjadi *Voltage spike* yang dapat merusak switch. Untuk menghindari voltage spike ini digunakan dioda atau kapasitor seperti pada gambar berikut:



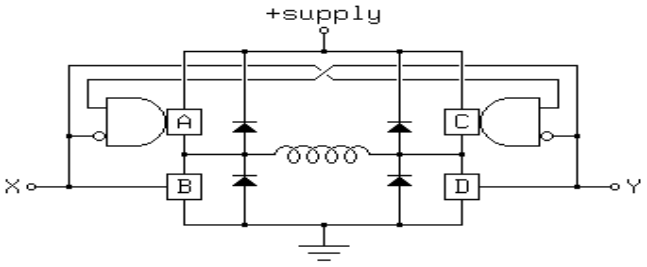
Gambar.12.9. Driver motor dengan pengaman

Driver pada Gambar 12.9 merupakan driver untuk mengontrol lamanya perputaran motor. Dari gambar tersebut switch digerakkan oleh suatu pulsa yang dibangkitkan oleh PC. Untuk mengontrol arah putaran motor dapat digunakan jembatan H driver, yang dapat dibuat seperti rangkaian pada Gambar 12.10.



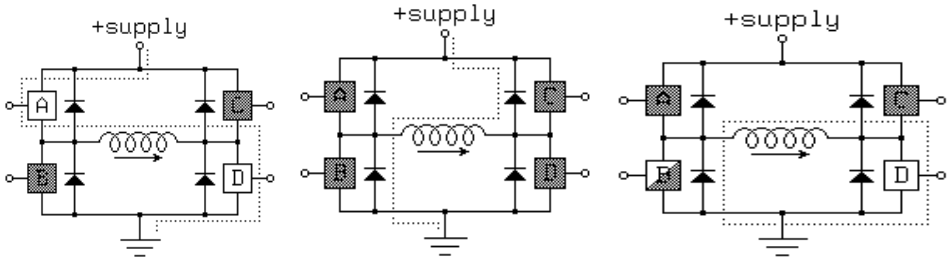
Gambar.12.10. Driver jembatan H.

Dengan menggunakan 4 switch, jembatan H akan memberikan 16 kemungkinan dengan 7 kemungkinan mengakibatkan hubungan singkat. Untuk mengatasi hubungan singkat ini maka jembatan H seharusnya dibuat seperti berikut:



Gambar 12.11. Jembatan H untuk menghindari hubung singkat.

Dari rangkaian di atas hanya terdapat 4 kemungkinan. Kemungkinan pertama yaitu pada saat X = 0 dan Y = 1 keadaan ABCD = 1001, motor akan berputar ke depan seperti pada Gambar 12.12(a). Sedangkan pada saat X = 1 dan Y = 0 keadaan switch ABCD menjadi 0110, yang berarti motor akan berbalik arah. Pada keadaan X = 0 dan Y = 0, semua switch akan berada dalam keadaan terbuka dan semua arus yang berasal dari kumparan motor akan cepat meluruh, sehingga motor berhenti secara perlahan – seperti yang ditunjukkan oleh Gambar 12.12(b). Keadaan X = 1 dan Y = 1 mengakibatkan switch B dan D tertutup, sehingga apabila motor sedang bergerak kemudian diberi pulsa seperti ini, maka arus kumparan motor akan meluruh dengan lambat, akibatnya motor akan mengerem – seperti yang ditunjukkan oleh gambar 12.12(c). Prinsip dasar jembatan inilah yang ada pada driver IC L298N.

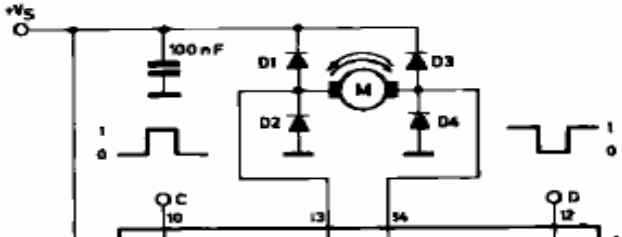


(a) berputar (b) tergelincir (c) mengerem

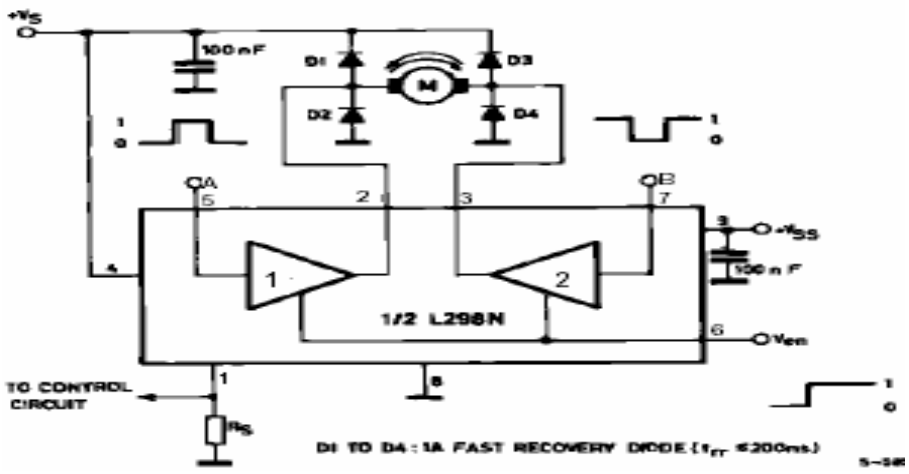
Gambar 12.12. Keadaan Jembatan H

Pada umumnya penggunaan motor DC tidak digerakkan dalam satu arah melainkan dapat digerakkan secara bolak-balik. Untuk mengatur perubahan arah gerak motor ini digunakan driver. Driver yang digunakan adalah IC L298N yang merupakan jenis jembatan H (H-Bridge) yang terdiri dari dua buah jembatan.

Masing-masing jembatan terdiri dari tiga masukan, yaitu dua masukkan In dan sebuah masukan En. Masukan En digunakan untuk mengktifkan jembatan H sedangkan masukan In untuk mengatur putaran motor seperti terlihat pada Tabel 12.9 logika putaran motor. Seperti pada Gambar 12.13 dua buah jembatan H yang terdapat dalam satu IC digunakan untuk mengatur dua motor.



(a)



(b)

Gambar 12.13. Rangkaian Driver motor L298

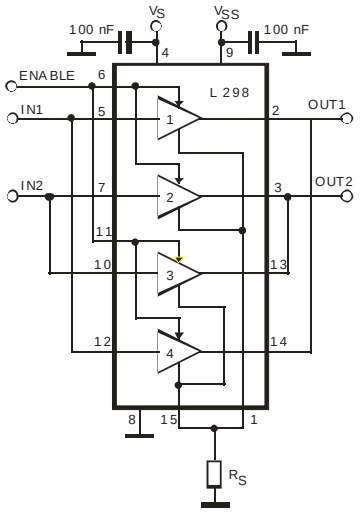
Tabel 12.9. Logika perputaran motor

Masukan		Fungsi
En = H	In1 = H;In2 = L	Berputar ke kanan
	In1 = L;In2 = H	Berputar ke kiri
	In1 = In2	Mengerem
En = L	In1 = X;In2 = X	Berhenti perlahan-lahan

Keterangan : H = tinggi, L = rendah, X = tinggi atau rendah

Dioda D1 sampai D4 digunakan untuk melewatkan arus yang dihasilkan dari putaran motor saat switch dimatikan sehingga tidak akan merusak IC. Tegangan output sense dapat digunakan untuk mengontrol arus dengan Rs digunakan untuk mendeteksi

intensitas arus. Rangkaian driver di atas digunakan untuk menggerakkan motor dengan arus kurang dari 2 A, sedangkan untuk menggerakkan motor dengan arus lebih dari 2 A dapat digunakan rangkaian dengan konfigurasi pararel seperti pada Gambar 12.14.

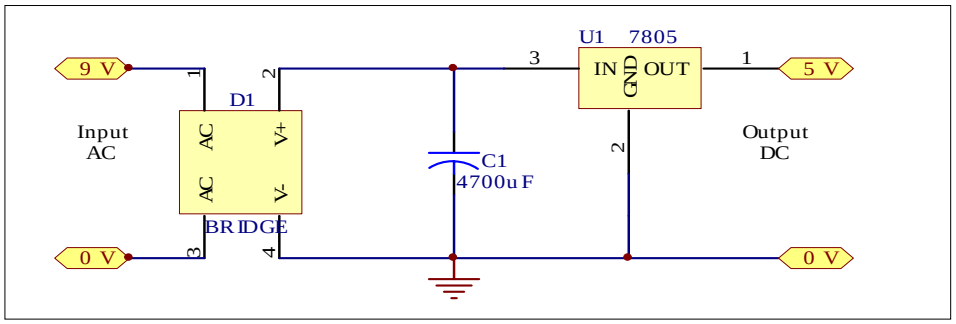


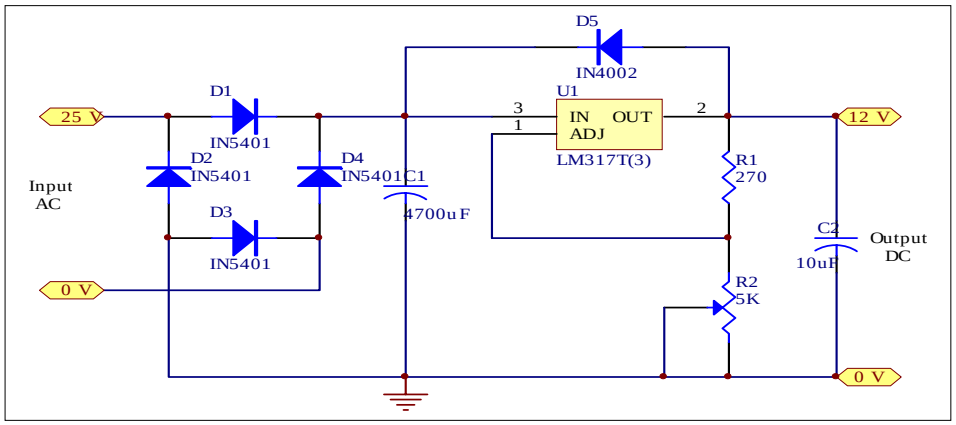
Gambar.12.14 Rangkaian Driver motor untuk I > 2A

Motor yang digunakan dalam perancangan ini ada dua jenis yaitu dua buah motor gearbox dengan arus kurang dari 2 A dan sebuah motor dengan arus 3 A tanpa gearbox. Penggunaan gearbox ini mengakibatkan kecepatan motor menjadi rendah dan torsi motor meningkat hal ini perlu diberikan pada motor dengan arus yang kecil sedangkan motor dengan arus 3 A memiliki kecepatan dan torsi yang tinggi. Penggunaan torsi yang tinggi dikarenakan motor harus dapat menggerakkan tempat pakan yang memiliki beban yang besar.

12.5.3.2. Sumber Tegangan

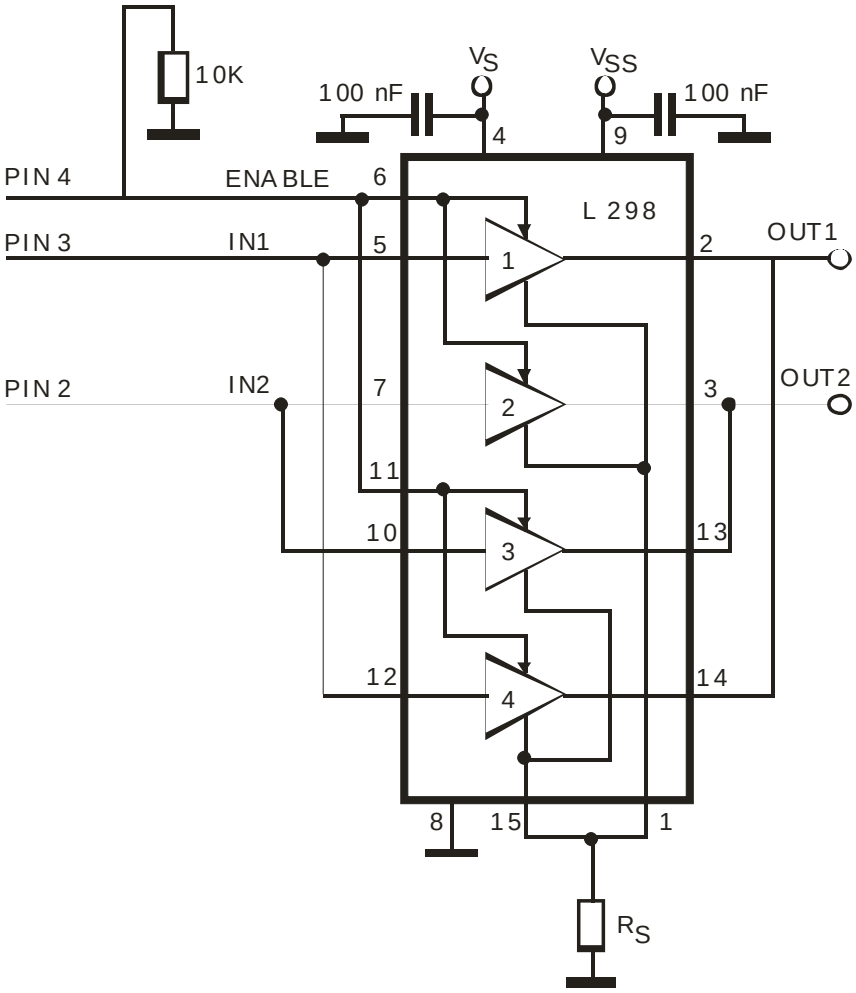
Sumber tegangan yang dibutuhkan untuk motor DC ini adalah 5 V dan 12 V. Tegangan 5 V digunakan untuk mengaktifkan driver, sedangkan tegangan 12 V digunakan untuk menggerakkan motor DC. Sumber tegangan yang dibuat menggunakan transformator 3 A, hal ini disesuaikan dengan kebutuhan motor DC yang membutuhkan arus 3 A.





Gambar 12.15. Power supply

12.5.3.3. Pemrograman Port Paralel Untuk Pengaturan motor DC



Gambar 12.16. Hubungan dari LPT ke Motor DC

Rangakain ke komputer atau port paralel ini sangat sederhana seperti yang ditunjukkan Gambar 12.16 yaitu dengan menghubungkan motor DC pada Enable ke Port Kontrol LPT. Umumnya port parallel mempunyai resitor untuk internal pull-up, tetapi beberapa mungkin tidak dilengkapi dengan resistor ini. Anda disarankan lebih baik melengkapi saja Port Kontrol ini dengan sebuah resitor 10K untuk eksternal pull up agar rangkaian dapat lebih portable untuk berbagai jenis komputer yang mungkin tidak memiliki resistor sebagai internal pull up.

Untuk menggerakkan motor DC ini dari komputer dibutuhkan tiga bit data dengan kondisi-kondisi seperti pada Tabel 12.9 1 bit untuk enable yang membuat motor bergerak atau berhenti perlahan dan 2 bit untuk arah gerakan kiri, gerak kanan atau mengerem. Alokasi port paralel untuk mengerkakkan motor DC ini adalah pada Register Data digunakan pin 2 dan 3 untuk arah gerak pada In 1 dan in 2 motor DC dan Register Kontrol pada pin 4 untuk enable pada motor DC.

Instruksi untuk bergerak atau berhenti dikirimkan ke motor DC yang dikontrol oleh Register Select pada Pin 4. Ketika pin 4 rendah instruksi Register Kontrol diambil diambil dan ketika tinggi Register Data harus diambil. Instruksi ini kita hubungkan ke

Port Parallel Select Printer yang kadang diinversi oleh hardware, sehingga Anda perlu menuliskan logika '1' ke bit 3 pada Register Kontrol agar Select Printer menjadi rendah.

Agar Anda dapat mengirimkan instruksi ke motor DC, maka Register Select harus dibuat rendah. Hal ini dapat dilakukan dengan mengeset bit 3 dari Register Kontrol menjadi '1'. Tentu saja kita tidak ingin register ini diset ulang pada saat pengirimin bit lain pada Port Register Kontrol. Cara menanggulanginya adalah dengan membaca Port kontrol dan melakukan operasi OR dengan 0x80, berikut contoh programnya .

```
outportb(KONTROL, inportb(KONTROL) | 0x08);
```

Operasi ini hanya akan mengeset bit 3 pada Register Kontrol. Setelah itu Anda menempatkan data byte pada baris data untuk menentukan arah gerak motor DC. Data tersebut kemudian dikirim ke motor DC dengan pewaktuan (timing) dari transisi tinggi ke rendah. Strobe diinversi secara hardware yaitu dengan mengeset bit 0 pada Register Kontrol sehingga Anda mendapatkan transisi tinggi ke rendah pada baris Strobe. Kemudian kita menunggu delay dan kembali ke level tinggi untuk pengiriman byte berikutnya. Byte ini sebenarnya dibutuhkan hanya 2 bit yaitu bit 0 dan bit 1 pada register data. Jika anda ingin memutar ke arah kanan Anda harus mengirimkan byte 0001 atau 1h dan jika anda ingin berputar ke arah kiri anda harus mengirimkan byte 0010 atau 2h. Jika ingin motor direm maka Anda harus memberikan kedu bit tersebut rendah atau tinggi artinya Anda bisa mengirm 0000 atau 0h atau 0011 atau 3h.

```
outportb(Data, 0x01);   /* untuk arah gerak kanan */
outportb(Data, 0x02);   /* untuk arah gerak kiri */
outportb(Data, 0x03);   /* untuk mengeremi */
```

Setelah Anda mengirimkan data arah putaran motor DC yang dikirim melalui Register Data ke Input 1 dan Input 2 motor DC, Anda perlu menjadikan bit 3 nol. Untuk itu kita bisa menggunkan perintah

```
outportb(KONTROL, inportb(KONTROL) & 0xF7);
```

Langkah berikutnya adalah melakukan perulangan sesui dengan keinginan Anda mengatur gerakan motor DC berikutnya. Perintah perulangan dapat dilakukan dengan menggunakan struktur `for`.

Delay yang Anda berikan harus cocok cukup untuk motor DC merespon perintah dari komputer. Jika ternyata motor DC tidak melakukan gerakan maka Anda dapat menaikkan delay. Agar lebih mudah, kita masih menggunakan LPT1 dengan Standard Parellel Port (SPP) pada alamat register dasar 378H. Jadi kita menggunakan Register Data dengan alamat dasar +0 yaitu pada alamat 0378H dan Register Kontrol dengan alamat dasar +2 yaitu pada alamat 037AH. Dalam hal ini kita tidak menggunakan Register Status dengan alamat dasar +1 yaitu pada alamat 0379H.

❑ Aplikasi Pemrograman Port C untuk Menggerakkan Motor Stepper

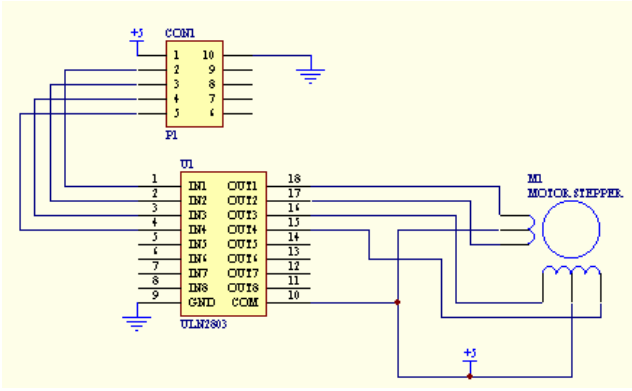
Motor stepper memiliki kontrol posisi dan kecepatan yang tepat serta memiliki torsi yang besar pada kecepatan rendah. Tegangan masukan stepper motor berupa sinyal pulsa yang akan menggerakan rotor dalam posisi diskrit. Untuk menggerakan motor stepper dapat dilakukan dengan mengirimkan data digital 4 bit yang digeser secara berurutan. Data yang dikirm untuk mengerakkan motor stepper dapat dapat dilihat pada Tabel 12.17.

Gambar 12.17 Bentuk fisik Motor Stepper

Tabel 12.10. Data yang diberikan untuk menggerakan motor stepper

Data Binner	Data Decimal	Langkah motor stepper
0001	1	1
0010	2	2
0100	4	3
1000	8	4

Bila sudah di langkah ke-4 ingin dilanjutkan ke langkah ke-5, maka dapat dilakukan dengan mengulang kembali mengirimkan data 0001 atau 1 desimal dan ke langkah ke-6 juga dengan mingirim data 0010 atau 2 desimal dan begitu seterusnya.



Gambar 12.18. Rangkaian Interface Motor Stepper

Data digital output dari LPT memiliki arus yang kecil sehingga tidak dapat digunakan untuk menggerakan motor stepper, sehingga perlu ditambahkan rangkaian driver untuk menguatkan arus sehingga dapat digunakan untuk menggerak motor stepper. Salah satu

driver yang dapat digunakan adalah IC ULN2803 yang berfungsi untuk menguatkan arus. Skematik rangkaian driver motor stepper dapat dilihat seperti pada Gambar 12.18.

```
/*  Pemrograman Port untuk  pengaturan motor stepper   */

#include <dos.h>
#include <string.h>

/* Memasukkan alamat dasar port LPT yang dipakai */
#define alamatportdasar 0x378

#define DATA alamatportdasar+0
#define STATUS alamatportdasar+1
#define KONTROL alamatportdasar+2

void main(void) {
  clrscr();
  int cacahan;
  a=1;
  for (cacahan = 0; cacahan <= 7; cacahan++) {
    a =data;
    outportb(DATA, a);
    a = a > 1;
  }
}
```

❑ Aplikasi Pemrograman Port untuk Tampilan LCD Dua Baris 16 Karakter

12.7.1. Gambaran Umum Pengoperasian LCD

LCD merupakan salah satu perangkat display yang bisa menampilkan gambar atau karakter yang diinginkan. Data yang ingin ditampilkan dalam LCD dapat dikirimkan dari komputer atau mikrokontroler dapat berupa data ASCII. Untuk mengirimkan data melalui komputer, kita harus membuat program tersendiri.

LCD mempunyai display dot matrix, yang sudah dilengkapi dengan panel dan rangkaian controller/driver. LCD ini bisa menampilkan 1 baris, atau 2 baris dan tiap baris memiliki 16 karakter. Untuk keperluan display, LCD biasanya mempunyai rangkaian pengontrol, data RAM, dan ROM pembangkit karakter.

Untuk pengiriman dan penerimaan data dari/ke LCD, dibutuhkan pin-pin kontrol. Tabel berikut ini menjelaskan hal tersebut.

Tabel 12.11. Fungsi Sinyal Kontrol

SINYAL KONTROL	FUNGSI
E	Menyebabkan kondisi data/kontrol di-latch Transisi Naik : me-latch kondisi kontrol (RS dan R_W) Transisi Turun : me-latch data
RS	Register Select Kontrol 1 = LCD pada mode data 0 = LCD pada mode command
R_W	Read/Write Control 1 = LCD menulis data 0 = LCD membaca data

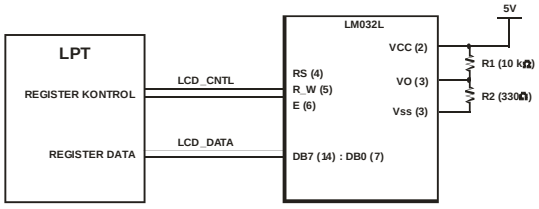
LCD ini dapat dioperasikan menjadi beberapa mode atau cara. LCD jenis LM032L dapat dioperasikan menjadi 2 cara. Cara 1 interface data 4 bit, dan yang kedua interface data 8 bit. Jika pengoperasiannya menggunakan data 4 bit maka dibutuhkan 2 kali pengiriman data per karakter, sedangkan menggunakan pengiriman data 8 bit relatif lebih mudah, karena tidak menghabiskan memori program tapi membutuhkan 4 tambahan jalur I/O.

Dalam implementasinya, secara umum ada 3 cara yang sering digunakan:

- Interface 8 bit
- Interface data 4 bit, dengan pengiriman data high nibble pada port
- Interface data 4 bit, dengan pengiriman data low nibble pada port

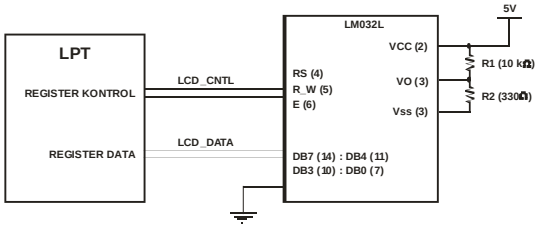
Ketiga cara pengoperasian data pada LCD seperti yang telah ditulis di atas, dapat dilihat pada skema/gambar berikut :

1. Interface Data 8 bit



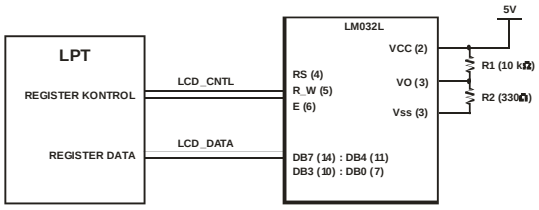
Gambar 12.19 Interface data 8 bit

2. Interface data 4 bit, pengiriman data high nibble pada port



Gambar 12.20 Interface data 4 bit, pengiriman data high nibble pada port

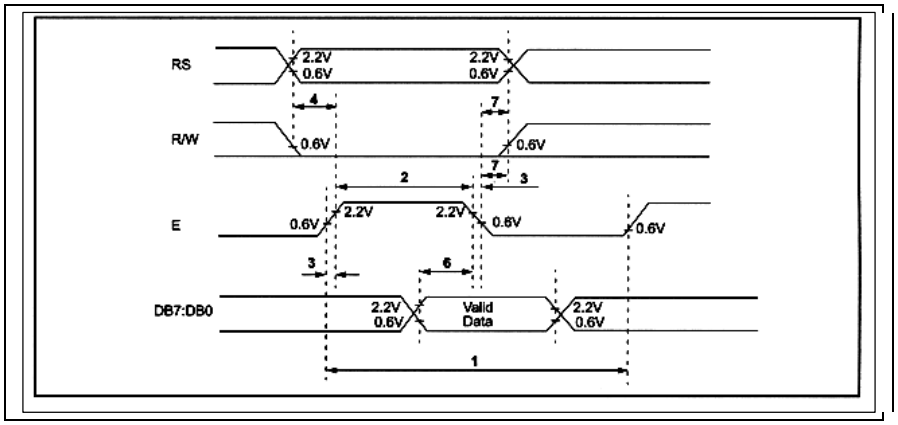
3. Interface data 4 bit, pengiriman data low nibble pada port



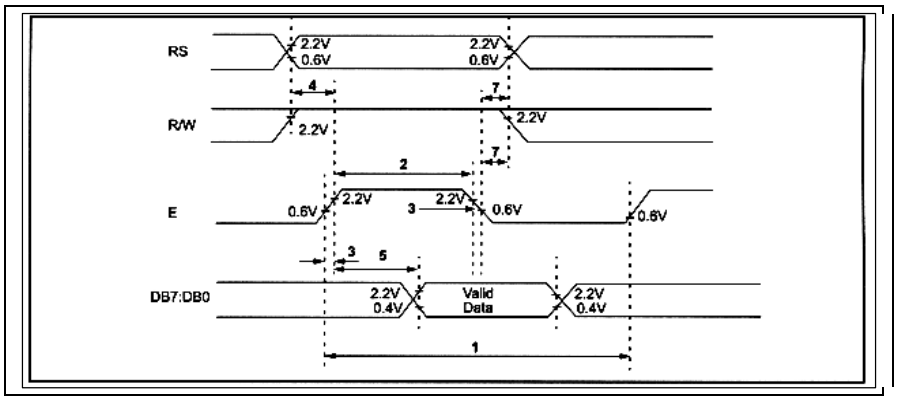
Gambar 12.21. Interface data 4 bit, pengiriman data low nibble pada port

LCD ini juga mempunyai tiga sinyal kontrol, diantaranya: Enable (E), Read/Write (R_W), dan register select (RS). Untuk menampilkan suatu huruf atau angka, data yang dikirim harus merupakan kode ASCII dari huruf dan angka tersebut.

Bagan pewaktu interface untuk penulisan dan pembacaan data dapat dilihat seperti pada Gambar 12.22-12.23.



Gambar 12.22. Pewaktu Interface Penulisan Data



Gambar 12.23. Pewaktu Interface Pembacaan Data

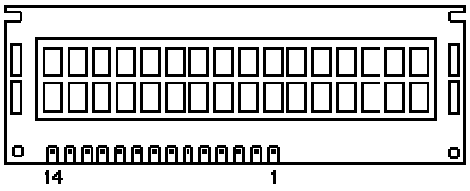
Daftar perintah untuk pengoperasian LCD dapat dilihat pada Tabel 12.12.

Tabel 12.12. Daftar Perintah untuk LCD

KODE HEX	INSTRUKSI PERINTAH LCD
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
5	Increment cursor (shift cursor to right)
6	Shift display right
7	Shift display left
8	Display OFF, Cursor OFF
A	Display OFF, Cursor ON
C	Display ON, Cursor OFF
E	Display OFF, Cursor blinking
F	Display ON, Cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to left
1C	Shift the entire display to right
80	Force cursor to beginning of 1st line
C0	Force cursor to beginning of 2nd line
38	2 lines and 5 x 7 matrix

12.7.2. Inisialisasi LCD

Tampak depan dari panel LCD yang mempunyai 14 pin dengan nomor pin secara berurutan dimulai dari sebelah kiri ditunjukkan oleh Gambar 11.24.



Gambar 12.24. Pin pada panel LCD

Sebelum mengirimkan data yang ingin ditampilkan, kita harus melakukan inisialilasi dengan prosedur sebagai berikut:

Proses inisalisasi LCD dengan mengirim beberapa perintah. Sebelum mengirimkan perintah-perintah tersebut kita harus melakukan pengecekan apakah LCD sudah siap menerima data atau belum. Cara pengecekannya dilakukan dengan rutin program siap()

Gambar 12.25.(a) yaitu dengan mengirim bit-bit control R/W logika satu dan RS logika nol. Selanjutnya, kita menunggu jawaban dari LCD yang akan dikirimkannya melalui bit P0.7.

Bit RS mempunyai dua keadaan yaitu nol dan satu. Jika kita ingin memberikan perintah atau kontrol maka bit RS ini harus diberi logika nol dan jika ingin mengirimkan data maka harus diberi logika satu.

Bit R/W juga mempunyai dua keadaan yaitu logika nol dan satu. Jika kita ingin membaca kondisi LCD pada bit BF (busy flag), maka kita harus mengirimkan logika 1 dan jawabannya akan dikrimkan melalui bit ke-7. Bila ingin menampilkan/menuliskan data ke LCD maka bit R/W ini harus diberi logika nol.

Apabila pada bit ke-7 ada pada logika satu, hal ini menandakan bahwa LCD sedang dalam keadaan sibuk sehingga tidak dapat menerima data atau perintah. Tetapi apabila bit ke-7 berlogika nol, ini berarti LCD dalam kondisi siap menerima data atau perintah.

Setelah selesai proses pengecekan bit ke-7, dan LCD telah memberi tanda sudah siap menerima data atau perintah maka langkah selanjutanya adalah menjalankan rutin program `perintah()` yang dapat dilihat pada Gambar 12.5.(b). Pengiriman perintah menggunakan jalur data D0-D7, sehingga seluruh data-data inisialisasi dikirim melalui jalur ini.

```
void siap()
{
 Each = 0;
 RS = 0;
 RatauW = 1;
do
  {
  Each = 0;
  Each = 1;
  }
while (antos == 1);
  Each = 0;
}
```

(a)

```
void perintah()
{
siap();
P0 = UNPAD;
RS = 0;
RatauW = 0;
Each = 1;
Each = 0;
}
```

(b)

Gambar 12.25.
a. Rutin pengecekan kesiapan LCD Matrik untuk menerima data.
b. Rutin untuk mengirimkan perintah ke LCD

```
void inisialisasi()
{
unsigned char init[4] = {0x38,0xC,0x6,0x1};
char x;
for (x = 0 ; x < 4 ; x ++)
 {
 UNPAD = init[x];
```

```
 perintah();
 }
}
```

Gambar 12.26. Rutin pengiriman data-data Inisialisasi LCD

Setelah pengecekan kesiapan LCD dilakukan, selanjutnya Anda melakukan pengiriman data-data inisilisasi yaitu pengiriman data 38h yang merupakan perintah untuk menjadikan LCD ini bekerja dengan matrik 5 x 7 dot per karakter. Kemudian dilakukan pengecekan kembali kesiapan LCD untuk menerima data insialisasi berikutnya. Data berikutnya adalah data fh yang merupakan perintah untuk menampikan kursor pada LCD dan membuat kursor berkedip. Data inisialisasi ketiga adalah data 6h yang merupakan perintah untuk membuat kursor bergeser dari kiri ke kanan. Sedangkan data lainnya adalah data 1h yang merupakan perintah untuk membersihkan layar LCD dan menempatkan kursor di alamat memori paling awal (80h). Rutin program pengiriman data-data inisialilasi LCD ini ditulis dalam rutin program `inisialisai()` Gambar 12.26.

12.7.3. Pengiriman Alamat Tujuan Penulisan Karakter

LCD yang digunakan dalam contoh ini adalah LCD Matrik 2 baris x 16 kolom yang setiap kolomnya memiliki alamat masing-masing yang sudah ditentukan oleh pabrik pembuatnya dengan pengalamatannya dapat dilihat pada Tabel 12.13.

Tabel 12.13. Pengalamatan LCD Matrik 2 baris x 16 kolom

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Baris ke-1	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
Baris ke-2	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF

Alamat tujuan penulisan karakter yang akan Anda kirim ke LCD harus dikirim sebagai rutin perintah bukan sebagai data ASCII. Bila pada pengiriman karakter yang banyak dan penempatannya berada pada alamat memori yang berurutan dari kiri ke kanan, maka pengiriman alamat tujuan penulisan karakter cukup satu kali saja yaitu dikirim alamat yang paling kiri. Jadi ketika karakter berikutnya dikirim maka LCD langsung menaikkan satu alamat tujuan penulisan karakternya.

12.7.4. Mengirim karakter ASCII

Data yang akan Anda tampilkan di LCD ini adalah data dalam bentuk karakter ASCII. Pengiriman data ini harus menggunakan rutin program tersendiri yang berbeda dengan rutin program perintah. Rutin program untuk mengirim data ASCII dapat Anda lihat pada rutin program `tampil()` pada Gambar 12.27.

```
void tampil()
{
siap();
RS = 1;
RatauW = 0;
Each = 1;
Each = 0;
}
```

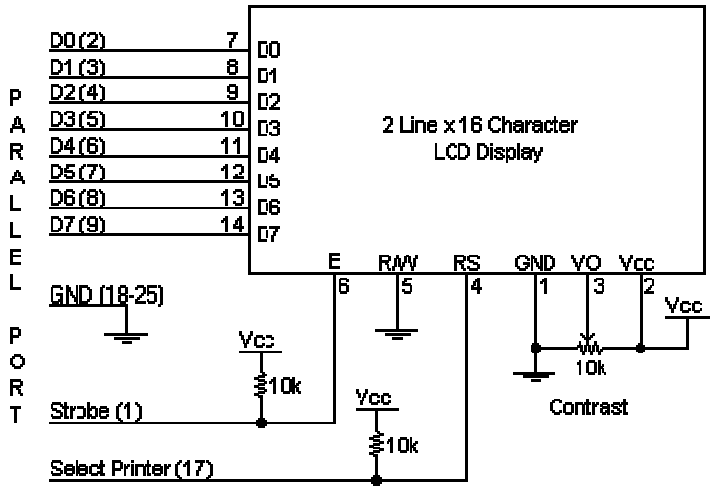
Gambar 11.27. Rutin program untuk mengirim data ASCII

Apabila kita akan melakukan pengiriman 16 karater sekaligus maka sebaiknya datanya disimpan pada Array baru dengan menggunakan perintah for untuk melakukan perulangan atau looping pengiriman ke-16 data tersebut.

Bila kita akan melakukan penggantian tampilan pada LCD dengan tampilan yang berbeda dengan tampilan sebelumnya, maka sebaiknya sebelum datanya dikirim diawali dengan perintah untuk membersihkan LCD dan memindahkan kursor ke alamat tujuan karakter kiri atas (80h).

12.7.5. Antarmuka LCD ke Port Paralel LPT PC

Setelah kita mengenal arsitekturnya, aturan inisialisasi dan aturan pengiriman data ke LCD, sekarang kita akan diperkenalkan dengan cara membuat program port yang dihubungkan ke LPT. LCD 2 baris x 16 karakter ini sangat banyak digunakan orang saat ini karana cukup sederhana cara kerjannya dan relatif murah harganya. Port LPT yang akan digunakan untuk mengantarmuka LCD ini adalah Standard Parallel Port (SPP).



Gambar 12.28. Skematik rangkaian antarmuka dengan port parallel

Rangakain antarmuka LCD ini sangat sederhana yaitu dengan menghubungkan panel kontrol LCD ke Enable dan Register Select ke Port Kontrol LPT, dan data pin 7-pin 14 ke Port Data. Umumnya Port parallel mempunyai resitor untuk internal pull-up, tetapi beberapa mungkin tidak dilengkapi dengan resistor ini. Anda disarankan lebih baik melengkapi saja dengan dua buah resitor 10K untuk eksternal pull up agar rangkain dapat lebih portable untuk berbagai jenis komputer yang mungkin tidak memiliki resistor sebagai internal pull up.
R/W pada panel LCD kita hubungkan pada write mode. Cara ini tidak akan mengakibatkan konflik bus pada baris data. Akan tetapi kita tidak bisa membaca balik internal LCD Busy Flag yang memberitahukan kita apakah LCD telah menerima dan menyelesaikan proses eksekusi instruksi terakhir. Hal ini bisa diatasi dengan memberikan delay pada program yang dibuat. Berikut ini adalah contoh lengkap pemrograman port untuk menggunakan LCD.

```
/*************************************************************
Pemrograman Port LCD
Register select harus dihubungkan ke select Printer pada Pin 17
Enable harus dihubungkan ke Strobe pada (PIN1)
DATA 0:7 dihubungkan ke  DATA 0:7 pada pin 2- pin 9
*************************************************************/

#include <dos.h>
#include <string.h>

/* Memasukkan alamat dasar port LPT yang dipakai */
#define alamatportdasar 0x378

#define DATA alamatportdasar+0
#define STATUS alamatportdasar+1
#define KONTROL alamatportdasar+2

void main(void)
```

```
{
  char string[] = {" Inst. Elektronika FI UNPAD    "};
  char init[20];
  int cacahan;
  int len;
  init[0] = 0x0F;     /* Inisialisasi  Tampilan  */
  init[1] = 0x01;     /* Membersihkan Tampilan  */
  init[2] = 0x38;     /* Dual Line / 8 Bits */

  /* Reset Control Port */
  outportb(KONTROL, inportb(KONTROL) & 0xDF);

  /* Set Register Select */
  outportb(KONTROL, inportb(KONTROL) | 0x08);

  for (cacahan = 0; cacahan <= 2; cacahan++) {
    outportb(DATA, init[count]);
    /* Set Strobe (Enable)*/
    outportb(KONTROL,inportb(KONTROL) | 0x01);
    delay(20); /* Penggunaan delay
                  untuk menunggu proses oleh LCD */
    /* Reset Strobe (Enable)*/
    outportb(KONTROL,inportb(KONTROL) & 0xFE);
    delay(20); /* Penggunaan delay
                  untuk menunggu proses oleh LCD */
  }

  /* Reset Register Select */
  outportb(KONTROL, inportb(KONTROL) & 0xF7);
  len = strlen(string);

  for (cacahan = 0; cacahan < len; cacahan++) {
    outportb(DATA, string[count]);
    /* Set Strobe */
    outportb(KONTROL,inportb(KONTROL) | 0x01);
    delay(2);
    /* Reset Strobe */
    outportb(KONTROL,inportb(KONTROL) & 0xFE);
    delay(2);
  }
}
```

Seperti telah dijelaskan pada tahapan-tahapan insialisasi LCD sebelumnya, untuk dapat menampilkan karakter pada LCD kita harus mengirim beberpa instruksi. Hal ini dikerjakan pada *for* pertama. Instruksi ini dikirimkan ke LCD pada Instruction Register yang dikontrol oleh Register Select pada Pin 4. Ketika pin 4 rendah instruksi register diambil dan ketika tinggi register data harus diambil. Instruksi ini kita hubungkan ke Port Parallel Select Printer yang kadang diinversi oleh hardware, sehingga kita perlu menuliskan logika '1' ke bit 3 pada Control Register agar Select Printer menjadi rendah.

Agar kita dapat mengirimkan instruksi ke modul LCD, maka Register Select harus dibuat rendah. Hal ini dapat dilakukan dengan mengeset bit 3 dari Register Kontrol menjadi '1'. Tentu saja kita tidak ingin register ini diset ulang pada saat pengirimin bit

lain pada Port Register Kontrol. Cara menaggulanginya adalah dengan membaca Port kontrol dan melakukan operasi OR dengan 0x80, berikut contoh programnnya

```
outportb(KONTROL, inportb(KONTROL) | 0x08);
```

Operasi ini hanya akan mengset bit 3 pada Register Kontrol. Setelah Anda menempatkan data byte pada baris data, Anda harus mengirimkan sinyal enable agar modul LCD membaca data. Data tersebut kemudian dikirm ke modul LCD dengan pewaktuan (timing) dari transisi tinggi ke rendah. Strobe diinversi secara hardware yaitu dengan mengeset bit 0 pada Register Kontrol sehingga Anda mendapatkan transisi tinggi ke rendah pada baris Strobe. Kemudian kita menunggu delay dan kembali ke level tinggi untuk pengiriman byte berikutnya.

Setelah Anda menginisilisasi dan mengirimkan text ke modul LCD dengan karakter yang dikirim melalui Data Port pada LCD, kita perlu menjadikan bit 3 nol. Untuk itu kita bisa menggunkan perintah

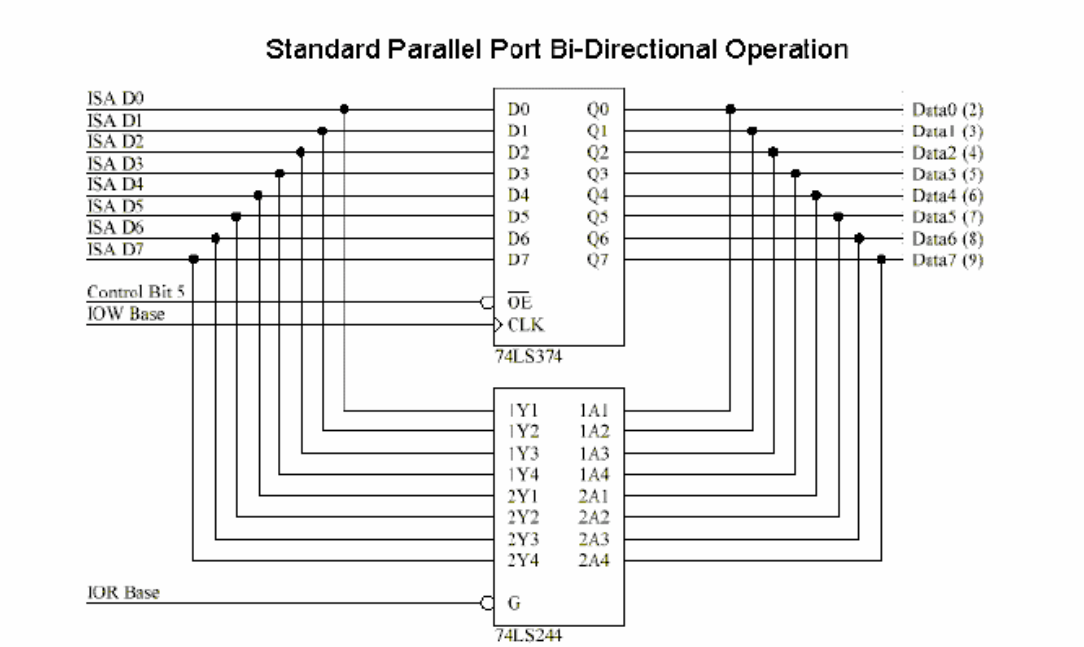
```
outportb(CONTROL, inportb(CONTROL) & 0xF7);
```

Langkah berikutnya adalah mengatur satu loop for untuk membaca byte dari string dan mengirimnya ke panel LCD. Langkah ini diulang sepanjang string.

Delay yang kita berikan harus cocok dengan LCD yang digunakan. Jika ternyata panel tidak melakukan inisialisasi maka kita dapat menaikkan delay. Sebaliknya jika panel melompatkan karakter atau mengulang-ulang karakter mungkin kita gagal dalam koneksi Enable. Kalau hal ini terjadi, kita bisa cek koneksi Enable ke Strobe.

❑ Port Paralel Dwi-Arah

Skematik pada Gambar 12.29 menunjukkan penyederhanaan Register Data dari Port parallel. Awalnya memang port Parallel digunakan dalam bentuk kartu dengan menggunakan komponen digital 74LS. Saat ini semuannya dibuat dalam satu IC, tetapi teori pengoperasiannya tetap sama.

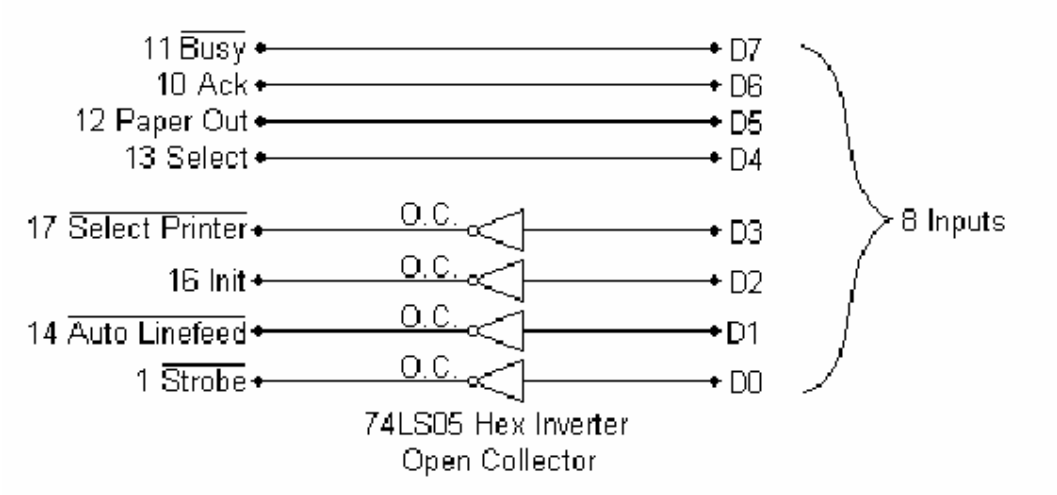


Gambar 12.29. Rangkaian Port dwi-arah

Port dwi-arah dibuat dengan keluaran enable 74LS374 dipaksa secara permanent dalam logika rendah, sehingga port data selalu hanya berupa keluaran. Jika kita membaca register data pada port parallel, datanya berasal dari 74LS374 yang juga terhubung dengan pin-pin data. Seandainya kita mampu mendrive(memicu) 74LS374, kita akan dapat mengunakannya sebagai port dwi-arah atau hanya sebagai input saja. Untuk mendrive ini kita dapat melakukannya dengan menambahkan transistor pada setiap pin input.

12.8.1. Mengunakan Port Paralel untuk Input 8 Bit

Ketika kita menggunakan suatu perangkat konversi data dari analog ke digital atau ADC untuk mengakuisisi suatu data maka kita sangat membutuhkan validitas data dengan mode port dwi-arah. Akan tetapi jika port parallel yang ada tidak mendukung untuk mode dwi-arah, kita jangan kawatir. Kita dapat maksimum memasukkan 9 bit data. Untuk melakukan hal ini guanakan 5 kanal input dari Port Status dan 4 kanal input (open collector) port Control.



Gambar 12.30. Rangkaian Port 8 input

Kita harus memilih Input port parallel sedemikan rupa sehingga memudahkan dalam pemrograman. Penempatan pin-pinnya seperti pada Gambar 11.26. Berdasarkan desain koneksi seperti di atas, sekarang kita membuat programnya. Hal pertama yang harus dilakukan adalah mengirimkan xxxx0100 ke Port Kontrol. Hal ini akan menyiapkan pin Port Kontrol dalam logika tinggi, sehingga pin-pin ini siap digunakan sebagai penginput data. Berikut ini contoh perintahnya:

```
outportb(CONTROL, inportb(CONTROL) & 0xF0 | 0x04);
```

Selanjutnya kita dapat membaca data most significant nibble dari port status. Karena kita menginginkan hanya data MSnibble, manupulasi logika digital yang dapat kita lakukan adalah dengan melakukan operasi AND pada hasilnya dengan 0xF0, dengan demikian LSnibble-nya nol.

Pin Busy bersifat terinversi secara hardware, tetapi tidak perlu dipermasalahkan karena kita sudah melakukan operasi AND di atas. Berikut contoh perintahnya:

```
a = (inportb(STATUS) & 0xF0);    /* Read MSnibble */
```

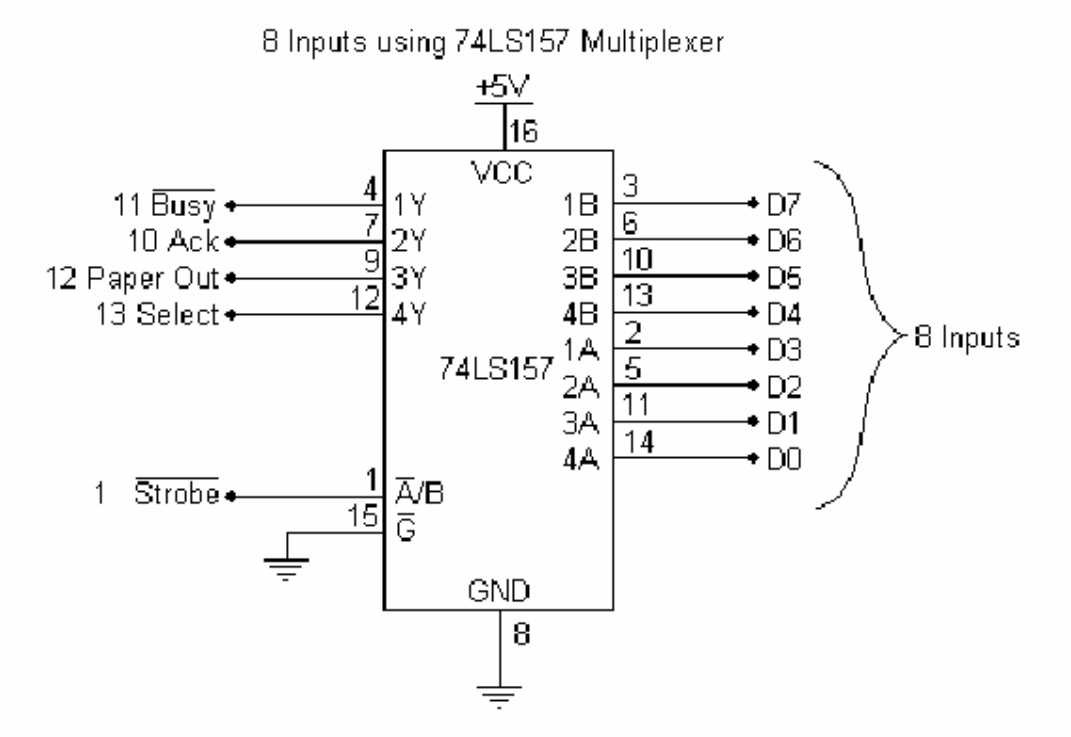
Sekarang kita dapat membaca LSnibble yang berada pada port control. Karena kita tidak memerlukan Msnibblenya,maka kita melakukan operasi AND pada hasilnya dengan 0x0F untuk membuat nol pada MSnibble. Selanjutnya adalah melakukan penggabungan kedua byte ini untuk mendapatkan 8 bit input. Hal ini dapat dilakukan dengan melakukan operasi OR terhadap kedua byte tersebut. Apakah data sudah valid? Karena Bits 2 dan 7 adalah inversi, jadi data belum valid. Untuk itu kita dapat melakukan operasi OR lagi pada byte itu dengan 0x84, yang menjadikan kedua bit itu togel (terbalik).

```
a = a |(inportb(CONTROL) & 0x0F);     /* Read LSnibble */
a = a ^ 0x84;                         /* Toggle Bit 2 & 7 */
```

12.8.2. Mode Nibble

Mode Nibble digunakan untuk membaca data input 8 bit tanpa menggunakan port yang berada dalam keadaan mode inverse, melainkan menggunakan kanal data. Mode Nibble menggunakan multiflekser 2 kanal ke 1 kanal untuk membaca satu nibble data pada satu pewaktuan. Kemudian menjadi berubah ke nibel yang lainnya, sehingga kita menjadi lebih mudah membacanya secara program. Program hanya perlu merekonstruksi kembali nible-nible itu menjadi data 8 bit atau 1 byte. Kekurangan desain perangkat keras seperti ini adalah sedikit agak lambat.

Sekarang kita hanya membutuhkan beberapa isntruksi I/O untuk membaca data itu, tetapi sebagai konsekwensinya kita harus menambah biaya untuk membeli IC eksternal berupa multiplekser.



Gambar 12.31. Rangkaian Port 8 input menggunakan 74LS157

Cara kerja multiplekser 74LS157 kanal 2 ke kanal 1 cukup sederhana. IC ini berperilaku seperti empat buah switch atau saklar. Ketika input A/B rendah, input A yang terpilih , misalnya 1A dilewatkan ke 1Y, 2A dilewatkan ke 2Y dan seterusnya. Ketika A/B tinggi, input B yang diambil.
Output-output Y dihubungkan ke port status pada port parallel, dengan desain seperti ini, port Y merepresentasikan MSnibble dari register status.

Langkah-langkah dalam pengambilan data pada port tersebut diawali dengan penginisialisasian multiplexer untuk mengubah salah satu input A /B. Selanjunya adalah membaca LSnibble pertama, kemudian kita menjadikan A/B rendah. Strobe mempunyai

sifat terinversi secara hardware, sehingga kita harus mengeset Bit 0 pada port control untuk mendapatkan logika rendah pada Pin 1. Berikut contoh perintahnya:

```
outportb(KONTROL, inportb(KONTROL) | 0x01);
```

Sesudah low nibble terpilih, kita membaca LSnibble dari Port Status. Kita perlu mengingat bahwa pin Busy Line bersifat inversi, sehingga kita perlu melakukan operasi AND dari hasilnya dengan 0xF0 untuk membuat nol pada Lsnibble. Berikut ini contoh perintahnya:

```
a = (inportb(STATUS) & 0xF0);          /* Read Low Nibble */
```

Sekarang kita hanya perlu untuk menggeser nibble yang baru dibaca ke LSnibble yang ada di variabel a,

```
a = a >> 4;           /* Menggeser 4 bit ke kanan */
```

Kita baru mendapatkan setengah dari data, sehinga selanjutnya kita perlu mendapatkan MSnibble. Untuk itu kita harus meswitch multiplexer untuk medapatkan input B. Kemudian kita dapat membaca MSnibble dan mengabungkan kedau nible tersebut menjadikannya 8 bit input atau satu byte, berikut contoh perintahnya:

```
/* memilih High Nibble (B)*/
outportb(CONTROL, inportb(CONTROL) & 0xFE);

/* Membaca High Nibble */
a = a |(inportb(STATUS) & 0xF0);

byte = byte ^ 0x88;
```

Dua kanal yang lain yang bersifat inversi dibaca pada pin Busy. Untuk itu kita perlu memberikan proses delay, jika terjadi kesalahan hasil pembacaan.

❑ Pemrograman Port Paralel dan Serial Menggunakan Fungsi DOS dan BIOS

12.9.1. Pemrograman Port Paralel Menggunakan Fungsi DOS dan BIOS

Kita juga dapat menggunakan perintah DOS untuk secara mudah mengirim karakter atau data ke port paralel. Perintah tersebut menggunakan fungsi `prints()`. Berikut adalah contoh rutin untuk melakukan hal tersebut:

```
```

```
# include <dos.h>

/* mengirim karakter ke port parallel atau printer* /
void prints(char *s) {
   while(*s) bdos(0x5,*s++,0);
}
```

Sebagai alternatif kita juga dapat menggunakan fungsi library dari BIOS dari pada menggunakan rutin DOS. Bentuk umum perintahnya adalah

```
int biosprint( int mode, int value, int port)
```

dengan *mode* merupakan penentuan bagaimana `biosprint()` bekerja. Jika mode adalah nol, `biosprint()` mengirim *value* ke port parallel, jika *mode* adalah 1, artinya menginisialisasi port parallel dan jika mode adalah 2, adalah kembali ke status printer.

Sedangkan port adalah spesifikasi port parallel yang digunakan yaitu 0 untuk LPT1 dan 1 untuk LPT2. Fungsi `biosprint()` ini ada pada header bios.h. Berikut ini adalah contoh perintahnya:

```
biosprint(1, 1000, 378h);
```

12.9.2. Pemrograman Port Serial Menggunakan Fungsi DOS dan BIOS

Untuk mengakses port serial kita dapat menggunakan fungsi DOS untuk membaca data dan mengirimkan data. Berikut ini contoh rutin untuk mengakses port serial:

```
# include <dos.h>

/* mengirim karakter ke port serial * /
void put_async(char ch) {
  bdos(0x4,ch,0);
}

/* membaca karakter dari port serial * /
void get_async() {
  return((char) bdos(0x3,0,0);
}
```

Fungsi ini secara otomatis menggunakan port serial default. Jika kita melakukan penggantian port pada MODE command, fungsi ini secara otomatis menggunakan port yang baru. Kita juga dapat menggunakan rutin dari BIOS yaitu fungsi `bioscom()` dengan bentuk umumnya

```

```

```
int bioscom(int mode, char val, int port)
```

dengan mode menujukkan operasi dari `bioscom()` sebagai berikut:

Mode	**Fungsi**
0	Inisalisasi Port
1	Mentransmisikan satu byte
2	Menerima satu byte
3	Mengembalikan status Port

Port serial yang akan diakses ditentukan oleh port dengan 0 menandakan COM1 dan 1 menandakan COM2 dan seterusnya. Fungsi ini ada di dalam bios.h.

Lampiran A

Standard Library

Pada lampiran ini kita akan sedikit membahas tentang beberapa pustaka standar (*standard library*) atau file header yang terdapat dalam bahasa C dan sering digunakan dalam pembuatan program. Sebagai catatan, kompilator yang Anda gunakan mungkin menyediakan pustaka-pustaka lain sebagai tambahan. Pembahasan mengenai pustaka standar ini dapat Anda pelajari lebih detil di dalam buku manual (*help*), tergantung dari kompilator yang Anda gunakan. Adapun pustaka-pustaka yang dimaksud di sini adalah sebagai berikut.

A.1. File `<assert.h>`

Header ini digunakan untuk melakukan diagnosa terhadap kesalahan program yang kita buat, yaitu dengan mengirimkannya ke dalam suatu file yang sering disebut dengan *standard error file*. Di dalam header ini hanya dua buah makro, yaitu `assert()` dan `NDEBUG`.

Berikut ini deklarasi dari makro `assert()`.

```
void assert(int ekspresi);
```

Apabila ekspresi bernilai 0 (salah atau *false*), maka ekspresi tersebut, nama file dari kode program dan nomor baris akan dikirimkan ke kompilator serta akan ditampilkan sebagai pesan kesalahan. Adapun pesan kesalahan yang pada umumnya di tampilkan adalah sebagai berikut.

Assertion failed: *ekspresi*, **file** *nama-file*, **line** *nomor-baris*

Namun, apabila kita menggunakan makro `NDEBUG` (kependekan dari "*No Debug*"), yaitu dengan mendefinisikannya menggunakan perintah `#define NDEBUG` maka makro `assert()` akan diabaikan oleh kompilator (tidak berpengaruh keberadaannya).

Untuk lebih memahaminya, coba Anda perhatikan contoh program di bawah ini yang akan menunjukkan implementasi dari makro `assert()` di atas.

```
#include <stdio.h>
#include <assert.h>
```

```
/* Mendefinisikan contoh fungsi untuk mengeset sebuah nama */
char *SetNama(char *s) {
  assert(s != NULL);
  return s;
}

int main(void) {
  char nama[25];

  /* Menggunakan fungsi SetNama dengan mengisikan
     parameter NULL */
  nama = SetNama( NULL );

  return 0;
}
```

A.2. File <ctype.h>

Header ini digunakan untuk melakukan pengecekan dan konversi terhadap sebuah karakter, *bukan* string. Fungsi-fungsi yang terdapat di dalam file header ini diawali dengan awalan `is...` dan `to...`. Adapun deklarasi dari fungsi-fungsi yang dimaksud di atas adalah sebagai berikut.

- Fungsi yang berawalan `is...` (untuk melakukan pengecekan karakter)

`int isalnum(int karakter);` untuk karakter (A..Z atau a..z) atau (0..9)
`int isalpha(int karakter);` untuk karakter A..Z atau a..z
`int iscntrl(int karakter);` untuk karakter 0x00..0x1F atau 0x7F
`int isdigit(int karakter);` untuk karakter 0..9
`int isgraph(int karakter);` untuk karakter 0x21..0x7E
`int islower(int karakter);` untuk karakter a..z
`int isprint(int karakter);` untuk karakter 0x20..0x7E
`int ispunct (int karakter);` untuk karakter 0x20..0x7E selain spasi atau isalnum()
`int isspace(int karakter);` untuk karakter spasi, tab, *carriage return*, *new line*, tab vertikal atau *formfeed*
`int isupper (int karakter);` untuk karakter A..Z
`int isxdigit(int karakter);` untuk karakter 0..9, A..F atau a..f

- Fungsi yang berawalan `to...` (untuk melakukan konversi karakter)

`int tolower(int karakter);` untuk mengkonversi suatu karakter menjadi huruf kecil
`int toupper(int karakter);` untuk mengkonversi suatu karakter menjadi huruf besar

Berikut ini contoh program yang akan menunjukkan penggunaan beberapa fungsi yang terdapat di dalam file header `<ctype.h>`

```
#include <stdio.h>
#include <ctype.h>

int main(void) {
  char indeks;

  /* Melakukan konversi karakter a..z menjadi A..Z */
  for (indeks = 'a'; indeks <= 'z'; indeks++) {
    printf("%c", toupper(indeks));
  }
  printf("\n");

  /* Melakukan konversi karakter A..Z menjadi a..z */
  for (indeks = 'A'; indeks <= 'Z'; indeks++) {
    printf("%c", tolower(indeks));
  }

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
```

A.3. File `<errno.h>`

Header ini digunakan untuk mengatasi kesalahan-kesalahan (*error*) yang muncul di dalam program. Di dalam header ini hanya terdapat dua buah makro, yaitu `EDOM` dan `ERANGE` serta sebuah variabel bertipe int, yaitu `errno`. Berikut ini bentuk pendefinisian dari makro-makro tersebut.

```
#define EDOM nilai_error
```

`nilai_error` di sini akan merepresentasikan nilai error yang dikembalikan oleh kompilator pada saat terjadi kesalahan penentuan *domain* dari perhitungan-perhitungan matematika yang kita lakukan di dalam program.

```
#define ERANGE nilai_error
```

Pada makro `ERANGE`, *`nilai_error`* di atas akan merepresentasikan nilai error yang dikembalikan oleh kompilator pada saat terjadi kesalahan penentuan rentang (*range*) dari perhitungan yang kita lakukan.

Sedangkan bentuk pendeklarasian dari variabel errno adalah sebagai berikut.

```
int errno;
```

Variabel ini selalu bernilai 0 pada saat permulaan program. Namun apabila terjadi error, maka variabel ini akan menampung nilai error tersebut.

Untuk lebih memahaminya, coba Anda perhatikan contoh program di bawah ini yang akan menunjukkan implementasi dari file header `<error.h>`.

```
#include <stdio.h>
#include <math.h>     /* untuk menggunakan fungsi pow() */
#include <errno.h>

int main(void) {
  /* Mendeklarasikan variabel hasil bertipe char untuk menampung
     hasil perpangkatan bertipe long */
  char hasil;
  /* Menampilkan nilai yang didefinisikan kompilator untuk EDOM
dan
     ERANGE */
  printf("EDOM \t: %d\n", EDOM);
  printf("ERANGE \t: %d\n\n", ERANGE);

  /* Menampilkan nilai dari variabel errno sebelum proses
     perhitungan dilakukan */
  printf("Nilai errno sebelum perhitungan \t: %d\n", errno);

  /* Melakukan perpangkatan menggunakan fungsi pow() */
  hasil = pow(32760, 234);

  /* Menampilkan nilai dari variabel errno setelah proses
     perhitungan */
  printf("Nilai errno setelah perhitungan \t: %d\n", errno);
  return 0;
}
```

Hasil yang akan diperoleh dari program di atas adalah seperti yang terlihat di bawah ini.

```
EDOM        : 33
ERANGE      : 34

Nilai errno sebelum perhitungan      : 0
Nilai errno setelah perhitungan      : 34
```

Apa yang dapat Anda simpulkan dari hasil tersebut? Sebenarnya apabila kita teliti sintak program di atas, maka program pasti akan mengalami kesalahan karena pada program tersebut kita mendeklarasikan variabel bertipe `char` sebagai nilai hasil dari

perhitungan yang melibatkan bilangan bertipe `long`. Ingat, tipe data `char` berada pada rentang –128 sampai +127, sedangkan kita melakukan perhitungan untuk bilangan yang berada di luar rentang tersebut (yaitu 32760), maka variabel hasil yang kita deklarasikan dengan tipe `char` di atas jelas tidak akan mampu untuk menampung nilai hasil perhitungan. Hal inilah yang menyebabkan terjadinya error pada penentuan rentang (*range*) nilai. Sebagai bukti dari pernyataan tersebut adalah nilai `errno` yang semula 0, maka setelah dilakukan proses perhitungan nilainya menjadi 34. Sedangkan kita tahu bahwa nilai 34 adalah nilai yang digunakan oleh makro `ERANGE` untuk merepresentasikan kesalahan dalam penentuan rentang nilai.

A.4. File `<float.h>`

Header ini digunakan untuk menyimpan nilai-nilai konstan yang didefinisikan untuk melakukan perhitungan-perhitungan pada bilangan riil. Adapun nilai-nilai tersebut adalah seperti yang tampak di bawah ini.

Nilai	Keterangan
`FLT_ROUNDS`	Mendefinisikan cara (*mode*) yang digunakan untuk melakukan pembulatan bilangan riil (*floating point*) -1, apabila nilai tidak bisa didapatkan/dibulatkan 0, apabila mendekati nol 1, apabila dibulatkan ke bilangan terdekat 2, apabila mendekati positif tak hingga 3, apabila mendekati negatif tak hingga
`FLT_RADIX 2`	Mendefinisikan basis untuk perpangkatan, (misalnya basis 2, basis 10 ataupun basis 16)

`FLT_MAN_DIG`
`DBL_MAN_DIG`
`LDBL_MAN_DIG`
	Mendefinisikan banyaknya digit dari suatu bilangan (dalam basis `FLT_RADIX`)

`FLT_DIG 6`
`DBL_DIG 10`
`LDBL_DIG 10`
	Maksimum jumlah digit dalam sebuah bilangan riil (basis 10) yang dapat direpresentasikan tanpa adanya perubahan setelah proses pembulatan

`FLT_MIN_EXP`
`DBL_MIN_EXP`
`LDBL_MIN_EXP`
	Nilai negatif minimum untuk perpangkatan menggunakan basis `FLT_RADIX`

`FLT_MIN_10_EXP –37`
`DBL_MIN_10_EXP –37`
`LDBL_MIN_10_EXP –37`
	Nilai negatif minimum untuk perpangkatan menggunakan basis 10

`FLT_MAX_EXP`
`DBL_MAX_EXP`
`LDBL_MAX_EXP`
	Nilai maksimum yang digunakan untuk perpangkatan menggunakan basis `FLT_RADIX`

`FLT_MIN_10_EXP +37`
`DBL_MIN_10_EXP +37`
`LDBL_MIN_10_EXP +37`
	Nilai maksimum yang digunakan untuk perpangkatan menggunakan basis 10

FLT_EPSILON 1E-5 DBL_EPSILON 1E-5 LDBL_EPSILON 1E-5	Digit signifikan yang dapat direpresentasikan
FLT_EPSILON 1E-37 DBL_EPSILON 1E-37 LDBL_EPSILON 1E-37	Nilai minimum untuk bilangan riil

Sebagai catatan bahwa `FLT` mewakili tipe `float`, `DBL` mewakili tipe `double` dan `LDBL` mewakili tipe `long double`.

A.5. File <limits.h>

Header ini digunakan untuk menyimpan makro yang merupakan nilai-nilai konstan yang siap pakai. Adapun makro-makro tersebut adalah sebagai berikut.

CHAR_BIT	8
CHAR_MAX	255
CHAR_MIN	-127
INT_MAX	+32767
INT_MIN	-32767
LONG_MAX	+2147483647
LONG_MIN	-2147483647
SCHAR_MAX	+127
SCHAR_MIN	-127
SHRT_MAX	+32767
SHRT_MIN	-32767
UCHAR_MAX	255
UINT_MAX	65535
ULONG_MAX	4294967295
USHRT_MAX	65535

A.6. File <math.h>

Header ini mendefinisikan fungsi-fungsi yang digunakan untuk melakukan perhitungan-perhitungan matematika. Adapun fungsi-fungsi yang termasuk ke dalamnya adalah sebagai berikut.

`double sin(double x);`	sinus dari x
`double cos(double x);`	cosinus dari x
`double tan(double x);`	tangen dari x
`double asin(double x);`	$\sin^{-1}(x)$ dalam rentang $[-\pi/2, \pi/2]$, $x \in [-1,1]$
`double acos(double x);`	$\cos^{-1}(x)$ dalam rentang $[0, \pi]$, $x \in [-1,1]$
`double atan(double x);`	$\tan^{-1}(x)$ dalam rentang $[-\pi/2, \pi/2]$

`double atan2(double y, double x);`	$\tan^{-1}(y/x)$ dalam rentang $[-\pi, \pi]$
`double sinh(double x);`	sinus hiperbola dari x
`double cosh(double x);`	cosinus hiperbola dari x
`double tanh(double x);`	tangen hiperbola dari x
`double exp(double x);`	Fungsi eksponensial e^x
`double log(double x);`	logaritma dari x, x > 0
`double log10(double x);`	logaritma (basis 10) dari x, x > 0
`double pow(double x, double y);`	x^y
`double sqrt(double x);`	$\sqrt{x}$, x ≥ 0
`double ceil(double x);`	Pembulatan ke bawah
`double floor(double x);`	Pembulatan ke atas
`double fabs(double x);`	Nilai absolut dari x yang bertipe double atau float
`double ldexp(double x, int n);`	$x.2^n$
`double frexp(double x, int *exp);`	Memecah x ke dalam dua bagian, bilangan yang dikalikan (misalnya a) dan eksponen. Sedangkan Bilangan yang digunakan untuk perpangkatan adalah 2, sehingga dapat dikatakan bahwa $x = a.2^{exp}$ Fungsi ini mengembalikan nilai a.
`double modf(double x, double *ip);`	Memecah x ke dalam dua bagian, bilangan bulat dan pecahan (yang berada dibelakang koma). Nilai bilangan bulat akan diset ke *ip* dan fungsi mengembalikan nilai pecahan.
`double fmod(double x, double y);`	Sisa bagi dari dari x/y

A.7. File `<signal.h>`

Header ini digunakan untuk mengetahui signal atau tanda-tanda yang perlu dilaporakan dalam eksekusi sebuah program. Dalam header ini telah didefinisikan sembilan buah makro, dua buah fungsi dan sebuah variabel. Berikut ini daftar dan penjelasan dari masing-masing item tersebut.

Adapun daftar makro yang telah didefinisikan tersebut adalah seperti yang tertera pada tabel di bawah ini.

`SIG_DFL`	*Default handler*
`SIG_ERR`	Merepresentasikan kesalahan signal
`SIG_IGN`	Mengabaikan signal
`SIGABRT`	Terminasi program secara tidak normal (dijalankan oleh fungsi `abort()`)
`SIGFPE`	Menandakan terjadinya *floating point error* disebabkan karena terjadi pembagian dengan nol dan juga operasi-operasi yang tidak diizinkan.
`SIGILL`	Menandakan terjadinya operasi-operasi yang tidak diperbolehkan

	(*illegal operation*)
`SIGINT`	Menandakan signal interaktif (seperti halnya ctrl-C)
`SIGSEGV`	Menandakan adanya kesalahan dalam pengalamatan memori.
`SIGTERM`	Permintaan terminasi program

Adapun fungsi-fungsi yang terdapat di dalam file header `<signal.h>` adalah sebagai berikut.

```
void (*signal(int sig, void (*funct) (int))) (int)
int raise(int sig);
```

Pada fungsi `signal()`, parameter *`sig`* merepresentasikan nomor signal yang kompatibel dengan makro-makro yang didefinisikan. Sedangkan parameter *`funct`* adalah sebuah fungsi yang akan dipanggil ketika signal terdeteksi. Fungsi `raise()` digunakan untuk membangkitkan atau menjalankan signal yang dilewatkan.

Berikut ini contoh program yang akan menunjukkan penggunaan file header `<signal.h>`.

```
#include <stdio.h>
#include <signal.h>

int main(void) {
  int x = 4, y = 0;

  if (y == 0) {
    raise(SIG_FPE);   /* membangkitkan signal SIG_FPE */
  }
  return 0;
}
```

A.8. File `<stdarg.h>`

Bahasa C mengizinkan kita untuk membentuk fungsi yang terdiri dari parameter (argumen) dinamis, artinya banyaknya parameter dapat berubah-ubah dalam setiap fungsi tersebut dipanggil. Adapun makro yang digunakan untuk melakukan hal tersebut adalah `va_arg`, `va_start`, `va_end` yang semuanya disimpan dalam file header `<stdarg.h>`.

Berikut ini deklarasi dari makro-makro tersebut.

```
void va_start (va_list ap, parameter_akhir);
tipe_data va_arg (va_list ap, tipe_data);
void va_end (va_list ap);
```

Parameter *`ap`* yang bertipe **`va_list`** di atas adalah sebuah array yang digunakan untuk menyimpan informasi yang dibutuhkan dalam penggunaan makro-makro tersebut. Sedangkan *`parameter_akhir`* merupakan parameter terakhir yang dicantumkan pada

saat pendefinisian fungsi. Terakhir, *tipe_data* di atas menunjukkan tipe data dari parameter yang akan dilewatkan selanjutnya ke dalam fungsi yang bersangkutan.

Makro `va_start`, `va_arg` dan `va_end` menyediakan cara yang portabel dalam mendapatkan atau mengakses daftar parameter yang dilewatkan secara dinamis ke dalam sebuah fungsi. Berikut ini penjelasan dari kegunaan tiga buah makro tersebut.

- `va_start`, digunakan untuk mengeset parameter *ap* agar menunjuk ke parameter pertama yang dilewatkan ke dalam fungsi.

- `va_arg`, digunakan untuk menunjuk dan mendapatkan nilai dari satu buah parameter selanjutnya yang dilewatkan di dalam fungsi. Ini dilakukan sampai dengan parameter terakhir.

- `va_end`, digunakan untuk membantu atau meyakinkan bahwa pemanggilan fungsi tersebut akan mengembalikan nilai secara normal (tidak terdapat error/kesalahan).

Untuk lebih jelasnya, perhatikan contoh program di bawah ini yang akan menunjukkan penggunaan dari ketiga makro di atas. Di sini kita akan membuat sebuah fungsi yang mencetak setiap parameter (berupa angka) yang dilewatkan ke dalam fungsi tersebut. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>
#include <stdarg.h>

/*  Mendefinisikan fungsi dimana banyaknya parameter bisa
dinamis, yaitu dengan cara mengisikan parameternya dengan tanda
titik sebanyak tiga kali (…) */
void cetak_angka(int argc, ...) {
  va_list ap;
  int arg;

  va_start(ap, argc);

  while (argc--) {
    printf("%d%c", va_arg(ap, int), (argc) ? '\t' : '\n');
  }
  va_end(ap);
}

int main(void) {

  cetak_angka(1, 100);          /* mencetak satu angka ke layar
*/
  cetak_angka(2, 100,200);      /* mencetak dua angka ke layar */
  cetak_angka(3, 100,200,300); /* mencetak tiga angka ke layar
*/
  cetak_angka(4, 100,200,300,400);    /* mencetak empat angka
                                         ke layar */

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
100
100     200
100     200     300
100     200     300     400
```

A.9. File <stddef.h>

Header ini menyimpan beberapa variabel dan makro standar. Beberapa diantaranya juga dapat Anda temui di dalam file header yang lain. Adapun deklarasi dari makro dan variabel tersebut adalah sebagai berikut.

- Makro

`NULL` nilai konstan yang diisikan ke dalam pointer null (pointer yang tidak menunjuk ke mana-mana)

`offsetof(`*`tipe_struktur`*`, `*`field`*`)`
menunjukkan offset dari salah satu field (dalam byte) dalam struktur.

- Variabel

`typedef ptrdiff_t`	hasil dari pengurangan dua buah pointer
`typedef size_t`	bilangan bulat tak bertanda (unsigned integer) sebagai hasil dari kata kunci `sizeof`
`typedef wchar_t`	bilangan integer sebagai ukuran dari tipe *wide character*

Berikut ini contoh program sederhana yang akan menunjukkan penggunaan makro `offsetof()` dan tipe `size_t` di atas.

```
#include <stdio.h>
#include <stddef.h>

/* Mendefinisikan sebuah struktur */
struct SISWA {
  char NIM[8];
  char nama[25];
  int usia;
};

/* Fungsi utama */
int main(void) {
```

```
  size_t ofs;
  ofs = offsetof(struct SISWA, usia);
  printf("Offset field usia dari struktur SISWA adalah : %d",
         ofs);
  return 0;
}
```

Contoh hasil yang akan diberikan dari program tersebut adalah sebagai berikut.

```
Offset field usia dari struktur SISWA adalah : 36
```

Sebagai catatan, terdapat beberapa kompilator C tidak mendefinisikan makro `offsetof()`.

A.10. File `<stdio.h>`

Header ini adalah header yang hampir digunakan dalam semua program C, yaitu untuk melakukan proses input-output. Adapun fungsi-fungsi yang terdapat di dalamnya adalah sebagai berikut.

- `FILE *fopen(const char *filename, const char *mode);`

Fungsi ini digunakan untuk membuka file *`filename`* dengan mode tertentu yang disimpan ke dalam parameter *`mode`*.

- `FILE *freopen(const char *filename, const char *mode, FILE *stream);`

Fungsi ini juga digunakan untuk membuka file dengan mode yang dispesifikasikan dan akan mengembalikan pointer stream bila berhasil. Apabila gagal maka nilai yang dikembalikan adalah `NULL`. Fungsi ini biasanya digunakan untuk mengubah file yang diasosiasikan dengan `stdin`, `stdout` maupun `stderr`.

- `int fflush(FILE *stream);`

Fungsi ini digunakan untuk membersihkan buffer keluaran dari *stream*. Apabila stream `NULL` maka semua buffer akan dibuang. Fungsi ini mengembalikan nilai 0 apabila berhasil dan `EOF` apabila gagal.

- `int fclose(FILE *stream);`

Fungsi ini digunakan untuk menutup file yang sebelumnya telah dibuka dengan menggunakan fungsi `fopen()`.

- `int remove(const char *filename);`

Fungsi ini digunakan untuk melakukan penghapusan file dimana nama file yang akan dihapus disimpan ke dalam parameter *`filename`*.

- `int rename(const char *oldname, const char *newname);`

Fungsi ini digunakan untuk mengubah nama file dari *`oldname`* menjadi *`newname`*.

- `FILE *tmpfile(void);`

Fungsi ini digunakan untuk membuat file temporari dengan mode `"wb+"` serta akan dibuang atau dihapus secara otomatis ketika program dihentikan secara normal. Apabila file temporari gagal terbuat maka nilai yang dikembalikan adalah `NULL`.

- `char *tmpnam(char s[L_tmpnam]);`

Fungsi ini digunakan untuk membuat file tempori namun penghapusan file tersebut harus dilakukan secara eksplisit. Artinya ketika program dihentikan file tersebut tidak secara otomatis dibuang.

- `int setvbuf(FILE *stream, char *buf, int mode, size_t size);`

Fungsi ini digunakan untuk melakukan kontrol pem-*buffer*-an terhadap *stream*. Fungsi `setvbuf()` harus dipanggil sebelum proses baca, tulis ataupun operasi lainnya. Adapun parameter mode di atas dapat bernilai `_IOFBF` (untuk *full buffering*), `_IOLBF` (untuk line buferring) dan `_IONBF` (untuk *no buferring*). Apabila proses berhasil maka buffer akan dialokasikan sebesar *`size`*.

- `void setbuf(FILE *stream, char *buf);`

Apabila buf bernilai `NULL`, maka buffer untuk stream juga akan dimatikan. Sebaliknya apabila buf tidak `NULL`, maka fungsi ini akan sama dengan `(void) setbuf(stream, buf, _IOFBF, BUFSIZ)`.

- `int fprintf(FILE *stream, const char *format, …);`

Fungsi ini digunakan untuk melakukan penulisan ke dalam *stream* dengan format tertentu.

- `int printf(const char * format, …);`

Fungsi ini digunakan untuk menampilkan data ke alat keluaran standar (misalnya monitor) dengan format-format tertentu.

- `int sprintf(char *s, const char *format, …);`

Fungsi `sprintf()` sama dengan `printf()`, hanya keluarannya terlebih dahulu disimpan ke dalam string *s*.

- `vprintf(const char *format, va_list arg);`

Fungsi ini digunakan untuk mengirimkan data ke `stdout` dengan menggunakan daftar argumen yang bertipe `va_list`.

- `vfprintf(FILE *stream, const char *format, va_list arg);`

Fungsi ini digunakan untuk mengirimkan data ke *stream* dengan menggunakan daftar argumen yang bertipe `va_list`.

- `vsprintf(char *s, const char *format, va_list arg);`

Fungsi ini digunakan untuk mengirimkan data ke string *s* dengan menggunakan daftar argumen yang bertipe `va_list`.

- `int fscanf(FILE *stream, const char *format, …);`

Fungsi ini digunakan untuk membaca data dari *stream* dengan format tertentu.

- `scanf(const char *format, …);`

Fungsi ini digunakan untuk membaca data dari `stdin`, sehingga sama apabila dituliskan dengan `fscanf(stdin, …)`.

- `int sscanf(char *s, const char *format, …);`

Fungsi ini digunakan untuk membaca data dengan format tertentu dari string *s*.

- `int fgetc(FILE *stream);`

Fungsi ini digunakan untuk mendapatkan karakter berikutnya dari sebuah *stream* (dianggap sebagai tipe `unsigned char`). Apabila berhasil maka fungsi ini akan mengembalikan karakter sampai ditemukan EOF (end of file). Sebaliknya apabila gagal maka fungsi akan langsung mengembalikan nilai EOF.

- `char *fgets(char *s, int n, FILE *stream);`

Fungsi ini digunakan untuk membaca baris dari *stream* yang dispesifikasikan dan menyimpannya ke string *s*.

- `int fputc(int c, FILE *stream);`

Fungsi ini digunakan untuk menuliskan karakter *c* (dianggap sebagai tipe `unsigned char`) ke dalam *stream*.

- `int fputs(const char *s, FILE *stream);`

Fungsi ini digunakan untuk menuliskan string *s* ke *stream*.

- `int getc(FILE *stream);`

Fungsi ini sebenarnya sama dengan fungsi `fgetc()`, hanya sebagai makro.

- `int getchar(void);`

Fungsi ini digunakan untuk mendapatkan sebuah karakter dari `stdin`. Apabila terjadi error maka fungsi akan mengembalikan nilai EOF. Ini sama seperti `getc(stdin)`.

- `char *gets(char *s);`

Fungsi ini digunkan untuk membaca data dari stdin dan menyimpannya ke string *s*.

- `int putc(int c, FILE *stream);`

Fungsi ini sebenarnya sama dengan fungsi `fputc()`, hanya sebagai makro.

- `int putchar(int c);`

Fungsi ini digunakan untuk menuliskan karakter *c*, sehingga sama dengan `putc(c, stdin)`.

- `int puts(const char *s);`

Fungsi ini digunakan untuk menuliskan string *str* ke `stdout` (tidak termasuk karakter null). Baris baru akan ditambahkan pada saat kita menggunakan fungsi ini.

- `int ungetc(int c, FILE *stream);`

Fungsi ini digunakan untuk mendorong karakter *c* (yang akan dikonversi ke tipe `unsigned char`) ke dalam *stream*, dimana karakter tersebut akan dikembalikan untuk proses pembacaan selanjutnya. Apabila terjadi kesalahan maka fungsi akan mengembalikan EOF.

- `size_t fread(void *ptr, size_t size, size_t nobj, FILE *stream);`

Fungsi ini digunakan untuk membaca data dari *stream* dan menyimpannya ke dalam array yang ditunjuk oleh *ptr*. Fungsi tersebut membaca *nobj* elemen sebesar *size* byte sehingga total byte yang dibaca adalah (`nobj` x `size`) byte.

- `size_t fwrite(const void *ptr, size_t size, size_t nobj, FILE *stream);`

Fungsi ini digunakan untuk menulis data dari *stream* dan menyimpannya ke dalam array yang ditunjuk oleh *ptr*. Fungsi tersebut membaca *nobj* elemen sebesar *size* byte sehingga total byte yang dibaca adalah (nobj x size) byte.

- `int fseek(FILE *stream, long offset, int origin);`

Fungsi ini digunakan untuk mengeset posisi file untuk *stream* yang dispesifikasikan. Untuk file biner posisi akan diset ke *offset* dari *origin*. Parameter *origin* dapat diisi nilai SEEK_SET (bagian awal), SEEK_CURR (posisi aktif) dan SEEK_END (bagian akhir atau *end of file*). Sedangkan untuk file teks, maka offset harus bernilai 0 atau nilai yang dikembalikan oleh fungsi ftell(), dimana parameter *origin* juga harus diset dengan nilai SEEK_SET. Fungsi ini akan mengembalikan nilai selain nol apabila terjadi kesalahan.

- `long ftell(FILE *stream);`

Fungsi ini digunakan untuk mengambil posisi aktif dari *stream*. Apabila terjadi kesalahan atau proses gagal maka nilai yang akan dikembalikan adalah –1L.

- `void rewind(FILE *stream);`

Fungsi ini digunakan untuk mengeset posisi file ke bagian awal dari *stream* yang diberikan. Sebagai contoh apabila kita menggunakan rewind(fp), maka sama artinya apabila kita menuliskan **fseek(fp, 0L, SEEK_SET); clearerror;**.

- `int fgetpos(FILE *stream, fpos_t *ptr);`

Fungsi ini digunakan untuk mengambil posisi file aktif dari *stream* yang dispesifikasikan dan merekamnya ke dalam *ptr*. Apabila proses berhasil nilai yang dikembalikan adalah 0. Apabila terjadi kesalahan maka fungsi akan mengembalikan nilai selain nol dan akan mengirimkannya ke dalam variabel errno.

- `int fsetpos(FILE *stream, const fpos_t *ptr);`

Fungsi ini digunakan untuk mengeset posisi yang direkam dengan menggunakan fungsi getpos() ke dalam *stream*. Apabila terjadi error, maka fungsi akan mengembalikan nilai selain 0.

- `void clearerr(FILE *stream);`

Fungsi ini digunakan untuk membersihkan EOF dan indikator kesalahan (*error indicator*) dari *stream*.

- `int fof(FILE *stream);`

Fungsi ini akan mendeteksi EOF dari *stream*.

- `int ferror(FILE *stream);`

Fungsi ini akan mengembalikan nilai selain nol apabila indikator kesalahan dari *stream* diset.

- `void perror(const char *s);`

Fungsi ini digunakan untuk menampilkan pesan kesalahan secara deskriptif. Mula-mula string `s` akan ditampilkan, diikuti tanda titik dua. Selanjutnya pesan kesalahan yang didasarkan pada variabel `errno` akan ditampilkan di belakangnya.

A.11. File `<stdlib.h>`

Header ini menyimpan fungsi-fungsi yang akan digunakan untuk melakukan pengkonversian bilangan, alokasi memori dan pekerjaan-pekerjaan pemrograman lainnya yang sejenis. Adapun fungsi-fungsi tersebut adalah sebagai berikut.

- `double atof(const char *s);`

Fungsi ini akan mengkonversi *s* ke tipe `double`.

- `int atoi(const char *s);`

Fungsi ini akan mengkonversi *s* ke tipe `int`.

- `long atol(const char *s);`

Fungsi ini akan mengkonversi *s* ke tipe `long`.

- `double strtod(const char *s, char **endp);`

Fungsi ini akan mengkonversi `s` ke tipe `double`, apabila terdapat *whitespace character* (spasi, *tab*, *carriage return*, *new line*, *vertical tab*, atau *formfeed*) maka karakter tersebut akan diabaikan. Namun apabila ditemukan karakter tidak dapat dikonversi maka alamat dari karakter tersebut akan dikirimkan ke pointer yang sedang ditunjuk oleh pointer *endp*.

- `long strtol(const char *s, char **endp, int base);`

Fungsi ini akan mengkonversi *s* ke tipe `double`, apabila terdapat *whitespace character* (spasi, *tab*, *carriage return*, *new line*, *vertical tab*, atau *formfeed*) maka karakter tersebut akan diabaikan. Namun apabila ditemukan karakter tidak dapat dikonversi maka alamat dari karakter tersebut akan dikirimkan ke pointer yang sedang ditunjuk oleh pointer *endp*. Apabila *base* bernilai antara 2 sampai 32, maka basis tersebut akan digunakan. Sedangkan apabila bernilai 0, maka apabila karakter pertama berupa karakter 1..9, basis yang digunakan adalah basis 10. Apabila karakter pertama berupa 0 maka basis yang digunakan adalah basis 8, sedangkan apabila diawali karakter 0 dan diikuti karakter `x` atau `X`, maka yang akan digunakan adalah basis 16.

- `unsigned long strtoul(const char *s, char **endp, int base);`

Fungsi ini sama dengan fungsi di atas `strtol()`, namun hasilnya berupa `unsigned long`.

- `int rand(void)`

Fungsi ini akan mengembalikan nilai acak antara rentang 0 sampai `RAND_MAX (32767)`.

- `void srand(unsigned int seed);`

Fungsi ini sebenarnya sama dengan fungsi `rand()`, hanya saja di sini kita menempatkan nilai yang akan diacak.

- `void *calloc(size_t nobj, size_t size);`

Fungsi ini akan mengalokasikan memori sebanyak *`nobj`* dengan ukuran *`size`* serta ruang yang dihasilkan dari pengalokasian tersebut akan diinisialisasi dengan nilai 0.

- `void *malloc(size_t size);`

Fungsi ini sama gunanya seperti fungsi `calloc()`, hanya ruang yang dihasilkan tidak diinisialisasi.

- `void *realloc(void *p, size_t size);`

Fungsi ini akan mengubah banyaknya objek yang ditunjuk oleh pointer *`p`* menjadi *`size`*.

- `void free(void *p);`

Fungsi ini digunakan untuk melakukan dealokasi memori yang sebelumnya telah dialokasikan menggunakan fungsi `calloc()`, `malloc()` maupun `realloc()`.

- `void abort(void);`

Fungsi ini akan menyebebkan program dihentikan secara tidak normal, sama seperti apabila kita melakukannya dengan `raise(SIGABRT)`.

- `void exit(int status);`

Fungsi ini akan menyebabkan program terhenti. Apabila kita mengisikan parameter *`status`* dengan nilai 0 maka program akan dihentikan secara normal. Sebaliknya apabila nilai yang dilewatkan adalah nilai 1 maka program akan dihentikan secara tidak normal. Kita juga dapat menggantikan 0 dan 1 dengan nilai `EXIT_SUCCESS` dan `EXIT_FAILURE`.

- `int atexit(void (*fcn) (void));`

Fungsi ini akan mendaftarkan fungsi *fcn* untuk dipanggil ketika program dihentikan secara normal. Nilai yang akan dikembalikan adalah nilai selain nol apabila pendaftaran fungsi tersebut tidak dapat dilakukan.

- `int system(const char *s);`

Fungsi ini akan mengirimkan perintah yang disimpan dalam string s ke sistem operasi tempat Anda melakukan eksekusi program. Misalnya apabila Anda menggunakan sistem operasi Linux atau Unix dan Anda menuliskan sintak `system("ls -la")`, maka sintak tersebut sama seperti kita menuliskan perintah `"ls -la"` pada *console*. Apabila `s` bernilai `NULL` maka fungsi ini akan mengembalikan nilai selain nol.

- `char *getenv(const char *name);`

Fungsi ini akan melakukan pencarian string di dalam lingkungan yang kita gunakan. Apabila ditemukan maka fungsi ini akan mengembalikan pointer yang berasosiasi dengan string *name*. Sebaliknya, apabila string tidak ditemukan maka nilai yang akan dikembalikan adalah `NULL`.

- ```
  void *bsearch(const void *key, const void *base, size_t n,
    size_t size,
    int (*cmp) (const void *keyval, const void *datum) );
  ```

Fungsi ini digunakan untuk melakukan *binary search* (pencarian bagi dua). Di sini terjadi proses pencarian dari elemen `base[0]` sampai `base[`*n*`-1]` yang sama dengan **key*, dimana *n* menunjukkan banyaknya elemen array *base* yang masing-masing elemennya berukuran *size*. Sedangkan `cmp` di sini berguna untuk melakukan perbandingan array sehingga nilainya adalah negatif apabila `key` < `base`, 0 apabila `key` = `base` dan positif apabila `key` > `base`. Sebagai catatan bahwa elemen-elemen yang terdapat dalam array harus sudah dalam keadaan terurut secara menaik (*ascending*).

Apabila proses pencarian berjalan dengan baik, maka fungsi `bsearch()` tersebut akan mengembalikan elemen yang ditemukan. Sebaliknya apabila tidak ditemukan maka yang akan dikembalikan adalah nilai `NULL`.

- ```
  void qsort(void *base, size_t n, size_t size,
              int (*cmp) (const void *, const void *) );
  ```

Fungsi ini digunakan untuk melakukan pengurutan cepat (*quick sort*) terhadap elemen-elemen yang terdapat dalam array *base*.

- `int abs(int n);`

Fungsi ini akan mengembalikan nila absolut (harga mutlak) dari bilangan *n* yang bertipe `int`.

- `int labs(long n);`

Fungsi ini akan mengembalikan nila absolut (harga mutlak) dari bilangan *n* yang bertipe `long`.

- `div_t div(int num, int denom);`

Fungsi ini digunakan untuk melakukan pembagian bilangan bulat bertipe `int` (*num* dibagi *denom*) dan menyimpan nilai-nilai yang dihasilkan ke dalam struktur `div_t`. Nilai hasil pembagian akan disimpan ke dalam field **`quot`** dan sisa baginya akan disimpan ke dalam field **`rem`** dari struktur `div_t` tersebut.

- `ldiv_t ldiv(long num, long denom);`

Fungsi ini cara kerjanya sama seperti fungsi `div()` di atas, hanya dilakukan untuk bilangan yang bertipe long dan dikembalikan ke dalam struktur `ldiv_t`. Adapun field yang terdapat pada struktur `ldiv_t` adalah sama persis dengan field yang terdapat pada struktur `div_t`.

A.12. File `<string.h>`

Header ini menyimpan fungsi-fungsi yang digunakan untuk melakukan pemanipulasian string. Untuk melihat penjelasan secara rinci, lihat kembali bab 6 – *Array dan String*. Meskipun demikian, terdapat beberapa fungsi yang belum sempat kita bahas dalam bab tersebut. Adapun prototipe dari fungsi-fungsi yang dimaksud tersebut adalah sebagai berikut.

`void *memcpy(s, ct, n);`	Melakukan penyalinan (*copy*) *n* karakter dari *ct* ke *s*
`void *memmove(s, ct, n);`	Sama seperti `memcpy()`, namun apabila terjadi overlap antara *s* dan *ct*, maka informasi akan dikirimkan ke *s* terlebih dahulu, selanjutnya baru ditulis ke *ct* sehingga proses *copy* berjalan dengan baik
`int *memcmp(cs, ct, n);`	Membandingkan *n* byte pertama dari *cs* dan *ct*.
`void *memchr(cs, c, n);`	Mencari karakter *c* pada n byte dari *cs*
`void *memset(s, c, n);`	Menyalin karakter *c* ke *n* karakter dari *s*

Sebagai catatan untuk prototipe-prototipe di atas, *s* bertipe `void*`, *cs* dan *ct* bertipe `const void*` serta *n* bertipe `size_t`.

A.13. File `<time.h>`

File header ini menyimpan fungsi-fungsi yang digunakan untuk melakukan pembacaan maupun pengkonversian terhadap waktu dan tanggal. Dalam header ini dideklarasikan beberapa tipe bentukan untuk keperluan-keperluan fungsi yang bersangkutan. Adapun tipe yang dimaksud tersebut adalah tipe `clock_t`, `size_t`, `time_t` dan struktur `tm`. Struktur `tm` sendiri memiliki field berikut (yang semuanya bertipe `int`).

`tm_sec`	Detik (0..61)
`tm_min`	Menit (0..59)
`tm_hour`	Jam (0..23)
`tm_mday`	Nomor hari atau tanggal (1..31)
`tm_mon`	Bulan (0..11)
`tm_year`	Tahun (mulai dari tahun 1900)
`tm_wday`	Nomor hari dalam seminggu (0..6 dimana 0 = minggu)
`tm_yday`	Nomor hari dalam setahun (0..365 dimana 0 = tanggal 1 Januari)
`tm_isdst`	*Daylight Saving Times*

Berikut ini daftar dan definisi dari fungsi-fungsi yang terdapat dalam file header `<time.h>` di atas.

- `asctime()`

Fungsi ini digunakan untuk melakukan konversi tanggal dan waktu menjadi string. Adapun prototipenya adalah sebagai berikut.

```
char *asctime(const struct tm *timeptr);
```

Parameter *`timeptr`* di atas merupakan pointer ke struktur `tm` dan digunakan untuk menyimpan nilai tanggal dan waktu yang akan dikonversi. Berikut ini format tanggal dan waktu yang akan dihasilkan oleh fungsi `asctime()`.

DDD MMM dd hh:mm:ss YYYY

dimana:

DDD	Nama hari dalam satu minggu (*Sun,Mon,Tue,Wed,Thu,Fri,Sat*)
MMM	Nama bulan dalam satu tahun (*Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sept, Oct, Nov, Dec*)
dd	Nomor hari dalam satu bulan (1..31)
hh	Jam (0..23)
mm	Menit (0..59)
ss	Detik (0..59)
YYYY	Tahun (empat digit)

- `clock()`

Fungsi ini digunakan untuk mendapatkan waktu/jam yang terdapat dalam prosesor. Kemudian hasilnya akan dibagi dengan makro `CLOCK_PER_SECOND` dan nilainya akan digunakan sebagai nilai kembalian fungsi. Sebaliknya, apabila proses gagal maka fungsi ini akan mengembalikan nilai –1. Berikut ini prototipe dari fungsi di atas.

```
clock_t clock(void);
```

- `ctime()`

Fungsi `ctime()` cara kerjanya sama dengan `asctime()`, hanya saja di dalam fungsi ini menggunkan parameter pointer yang akan menunjuk ke tipe `time_t`. Sedangkan

pada pada fungsi asctime(), parameter yang digunakan merupakan pointer ke struktur `tm`. Berikut ini prototipe dari fungsi `ctime()`.

```
char *ctime(const time_t *timeptr);
```

- `difftime()`

Fungsi ini digunakan untuk mengetahui selisih dari dua buah tanggal atau waktu yang disimpan di dalam parameter yang bertipe `time_t`. Adapun prototipe dari fungsi ini adalah sebagai berikut.

```
double difftime(time_t time1, time_t time2);
```

Apabila proses berjalan dengan baik maka fungsi ini akan mengembalikan perhitungan *`time1 - time2`*.

- `gmtime()`

Fungsi ini digunakan untuk melakukan konversi dari tipe `time_t` menjadi struktur `tm` dan diekspresikan ke dalam UTC (*Coordinated Universal Time*) atau yang sering juga dikenal dengan istilah GMT (*Greenwich Meant Time*). Berikut ini prototipe dari fungsi `gmtime()`.

```
struct tm *gmtime(const time_t *timer);
```

Apabila proses berjalan dengan baik maka nilai kembalian dari fungsi ini berupa pointer ke struktur `tm`. Sebaliknya, apabila proses gagal maka nilai yang dikembalikan adalah nilai `NULL`.

- `localtime()`

Fungsi ini digunakan untuk mendapatkan waktu lokal berdasarkan zona yang kita tempati. Adapun prototipenya adalah sebagai berikut.

```
struct tm *localtime(const time_t timer);
```

Sama seperti fungsi `gmtime()`, fungsi `localtime()` juga akan mengembalikan pointer ke struktur `tm` atau nilai `NULL` apabila proses gagal.

- `mktime()`

Fungsi ini digunakan untuk melakukan konversi dari waktu lokal yang terdapat dalam struktur `tm` menjadi tipe `time_t`. Berikut ini prototipe dari fungsi `maketime()`.

```
time_t  maketime(const struct tm *timeptr);
```

- `strftime()`

Fungsi ini digunakan untuk melakukan penulisan waktu yang disesuaikan dengan format tertentu. Berikut ini prototipe dari fungsi `strftime()`.

```
size_t strftime(char *str, size_t maxsize, const char *format,
       const struct tm *timeptr);
```

Apabila kita amati, cara kerja dari fungsi ini hampir mirip dengan fungsi `printf()` yang biasanya kita gunakan untuk menampilkan data ke layar monitor. Adapun bentuk format yang telah disediakan oleh bahasa C untuk waktu adalah sebagai berikut.

`%a`	Nama hari (disingkat)
`%A`	Nama hari (nama penuh)
`%b`	Nama bulan (disingkat)
`%B`	Nama bulan (nama penuh)
`%c`	Tanggal dan waktu
`%d`	Nomor hari dalam satu bulan (01..31)
`%H`	Jam (00..23)
`%I`	Jam (01..12)
`%j`	Nomor hari atau tanggal (001..366)
`%m`	Nomor bulan dalam satu tahun (01..12)
`%M`	Menit dalam satu jam (00..59)
`%p`	AM/PM
`%S`	Detik dalam satu menit (01..61)
`%U`	Nomor minggu dalam satu tahun (00..53, diawali hari Minggu)
`%w`	Nomor hari (0..6)
`%W`	Nomor minggu dalam satu tahun (00..53, diawali hari Senin)
`%x`	Tanggal
`%X`	Waktu
`%y`	Tahun tanpa penulisan abad (00-99)
`%Y`	Tahun dengan penulisan abad
`%Z`	zona waktu, jika ada
`%%`	%

- `time()`

Fungsi ini digunakan untuk melakukan mencatat waktu kalender yang sedang aktif dan menyimpannya ke dalam format `time_t`. Berikut ini prototipe dari fungsi ini.

```
time_t time(time_t *timer);
```

Apabila parameter *`timer`* di atas bernilai `NULL` maka fungsi `time()` ini akan mengembalikan nilai –1.

Lampiran B

Membuat File Header (.h)*

Dalam pemrograman tingkat lanjut yang melibatkan kasus-kasus yang bersifat kompleks, kita tentu akan memisahkan kode program yang kita buat ke dalam file-file pustaka (*library files*, dalam hal ini file *.h) sesuai dengan modul-modul yang bersangkutan. Untuk itu, di sini kita akan membahas bagaimana cara membuat file header tersebut sehingga Anda dapat memahami dan mampu mengimplementasikannya pada kasus-kasus program yang Anda hadapi.

Sebagai catatan di sini penulis akan menggunakan kompiler **MinGW** (untuk sistem operasi Windows) dan **cc** (untuk sistem operasi Linux). Hal ini bertujuan untuk melakukan perbandingan serta menunjukkan bahwa bahasa C adalah bahasa yang bersifat *portable*. Sebagai contoh, kita akan membuat file header dengan nama **contoh.h** yang berisi fungsi-fungsi perhitungan sederhana. Selanjutnya file tersebut akan kita gunakan di dalam program utama yang kita beri nama **utama.c**. Adapun langkah-langkah yang harus dilakukan adalah sebagai berikut.

Langkah 1

Buatlah file **contoh.h** dengan editor yang tersedia dan isikan sintak di bawah ini ke dalamnya.

```
extern int kuadrat(int x);
extern int pangkat(int basis, int e);
extern int faktorial(int n);
```

Kata kunci `extern` (yang berarti eksternal) di atas menunjukkan bahwa fungsi-fungsi tersebut implementasinya terdapat pada kode program yang terdapat di dalam file lain.

Langkah 2

Buatlah file **contoh.c** yang merupakan implementasi dari fungsi-fungsi yang telah dideklarasikan dalam file contoh.h di atas. Adapun sintak programnya adalah sebagai berikut.

```
int kuadrat(int x) {
  return x*x;
}
```

```
int pangkat(int basis, int e) {
  if (e == 0) return 1;
  else return basis * pangkat(basis, e-1);
}

int faktorial(int n) {
  if (n == 0) return 1;
  else  return n * faktorial(n-1);
}
```

Ingat, jangan lakukan kompilasi terhadap file tersebut terlebih dahulu. Satu hal lagi yang perlu diperhatikan adalah apabila Anda menggunakan fungsi-fungsi yang terdapat pada file header lain, maka Anda juga harus mendaftarkan file header tersebut. Sebagai contoh apabila dalam implementasi fungsi di atas Anda akan menggunakan fungsi `printf()`, maka Anda harus mendaftarkan file `<stdio.h>` ke dalam file di atas. Untuk kasus ini, kita tidak tidak menggunakan fungsi-fungsi yang terdapat pada file header lain sehingga kita tidak perlu menuliskannya ke dalam file tersebut.

Langkah 3

Buat file **utama.c** dimana di dalamnya akan menggunakan file contoh.h di atas. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>

#include "contoh.h"

int main(void) {

  /* Menggunakan fungsi-fungsi yang terdapat pada file header
     contoh.h */
  printf("Kuadrat 3 \t= %d\n", kuadrat(3));
  printf("2^5 \t\t= %d\n", pangkat(2,5));
  printf("5! \t\t= %d\n", faktorial(5));

  return 0;
}
```

Langkah 4

Lakukan kompilasi terhadap file-file di atas. Apabila Anda menggunakan kompiler untuk win32 untuk sistem operasi Windows seperti Microsoft Visual C, MinGW Visual Studio maupun kompiler lainnya yang sejenis (di sini penulis menggunakan MinGW Visual Studio), maka Anda tinggal membuat sebuah project baru (berupa *win32 console application*) dan tambahkan ketiga file tersebut (contoh.h, contoh.c dan utama.c) ke dalam project bersangkutan, selanjutnya pilih menu **Build**. Namun apabila Anda

menggunakan kompiler **cc** di dalam sistem operasi Linux, Anda dapat melakukan kompilasi file-file tersebut dengan cara menuliskan perintah berikut.

```
$ cc –c contoh.c –o contoh.o
$ cc –c utama.c –o utama.o
$ cc contoh.o utama.o –o eksekusi.o
```

–c di atas merupakan suatu option yang menunjukkan bahwa kita hanya melakukan kompilasi terhadap file tersebut, namun tidak melakukan *linking* terhadap file objek dari file lain. Untuk memeriksanya, Anda dapat mengeksekusi file hasil kompilasi tersebut dengan cara menuliskan perintah berikut.

```
$ ./eksekusi.o
```

Hasil yang akan diberikan oleh program di atas adalah sebagai berikut.

```
Kuadrat 3       = 9
2^5             = 32
5!              = 120
```

Kekurangan dari cara di atas adalah pada saat kita akan melakukan perubahan terhadap salah satu file di atas, kita tetap melakukan kompilasi ulang terhadap semua file. Bayangkan apabila kita mempunyai banyak file di dalam program, tentu hal ini akan memperlambat proses kompilasi. Untuk menghindari hal itu, para programmer C biasanya menggunakan file khusus yang digunakan untuk melakukan kompilasi terhadap file-file yang telah dilakukan perubahan saja. File khusus tersebut sering dinakaman dengan *makefile*.

Sebagai contoh, di sini kita akan membuat *makefile* (di dalam sistem operasi Linux).

Adapun langkah-langkahnya adalah sebagai berikut.

- ***Langkah a***

Buatlah file (misalnya dengan nama **eksekusi.mak**), dan isikan sintak seperti di bawah ini.

```
eksekusi: utama.o contoh.o
cc utama.o contoh.o –o eksekusi.o

utama.o: utama.c
```

```
cc –c utama.c

contoh.o: contoh.c
cc –c contoh.c
```

- ***Langkah b***

Untuk melakukan kompilasi terhadap makefile tersebut, tuliskan perintah di bawah ini.

```
$ make –i –m eksekusi
```

–i dan **–m** merupakan option yang tidak harus disertakan. **–i** adalah option yang berfungsi untuk mengabaikan pesan kesalahan (*error message*) yang mungkin akan dikeluarkan pada saat pemanggilan perintah **make**. Sedangkan **–m** adalah option yang akan mengabaikan perintah **make** (versi sebelumnya) yang terdapat di dalam sistem operasi Linux.

Apabila Anda ingin memeriksa hasil dari program di atas, caranya seperti biasa yaitu dengan menuliskan perintah berikut.

```
$ ./eksekusi.o
```

Daftar Pustaka

1. Brian W. Kernighan & Dennis M. Ritchie, *The C Programming Language*, Prentice-Hall, 1988.
2. Robert Lafore, *C Programming using TurboC++*, SAMS Publishing, 1993.
3. Mike Benahan, Declan Brady & Mark Doran, *The C Book*, Addison Wesley, 1991.
4. Jerry Jongerius, *Writing Bug-Free C Code*, Prentice-Hall, 1995.
5. Robert C. Hutchison & Steven B. Just, *Programming Using The C Language*, McGraw-Hill, 1988.
6. A. Dave Marshall, *Programming in C UNIX System Call and Subroutines using C*, 1999.
7. Steve Ouallin, *Practical C Programming*, O'Reilly & Associates.
8. Peter Aitken & Bradley L. Jones, *Teach Yourself C in 21 Days*, SAMS Publishing.
9. Erric Huss, *The C Library Reference Guide*, 1997.
10. Steve Holmes, *C Programming*, University of Strathclyde Computer Centre, 1995.
11. Gordon Dodrill, *Tutorial C*, Admin.NET, 1995.
12. University of Leicester, *Introduction to C Programming*, 2001.
13. P.J. Plauger & Jim Brodie, *Standard C*, 2001.
14. K. Joseph Wesley & R. Rajesh Jeba Anbiah, *A to Z of C*, 2004.
15. Herbert Schildt, *C++ : The Complete Reference*, McGraw-Hill, 1998.
16. Herbert Schildt, *Advance C*, McGraw-Hill, 1986.
17. Joseph S. Byrd & Roberto O, *Micro Computer System: Architecture and Programming*, Prentice Hall, 1993.
18. Joseph Williams, *An Introduction to Computing Infrastructure: Hardware and Operating System*, Que Education &Training, 1997.
19. James P. Cohoon & Jack W. Davidson, *C++ Programming Design*, McGraw-Hill, 1997.
20. Craig Arnush, *Teach Yourself Borland C++ 5 in 21 Days*, SAMS Publishing, 1996.
21. Hartono Partoharsodjo, *Tuntunan Praktis Pemrograman Bahasa C*, Elexmedia Komputindo Jakarta, 1989.
22. Abdul Kadir, *Pemrograman Dasar Turbo C*, Andi Offset Yogyakarta, 1991.

23. Budi Raharjo, *Pemrograman C++: Mudah dan Cepat Menjadi Master C++*, Informatika Bandung, 2006.

Cara Mudah *Mempelajari* Pemrograman C & Implementasinya

C merupakan bahasa pemrograman yang sudah tidak diragukan lagi kehandalannya dan banyak digunakan untuk membuat program-program dalam berbagai bidang, termasuk pembuatan kompilator (*compiler*) dan sistem operasi. Sampai saat ini, C masih tetap menjadi bahasa populer dan berwibawa dalam kancah pemrograman. Sejauh ini, C juga telah menjadi inspirasi bagi kelahiran bahasa-bahasa pemrograman baru, seperti C++, Java, dan juga yang lainnya; sehingga dari sisi sintak kontrol programnya, ketiga bahasa ini bisa dikatakan sama. Bahasa pemrograman C sangatlah fleksibel dan portabel, sehingga dapat ditempatkan dan dijalankan di dalam beragam sistem operasi. Pada umumnya, C banyak digunakan untuk melakukan interfacing antar perangkat keras (*hardware*) agar dapat berkomunikasi satu sama lainnya.

Buku ini akan mengantarkan Anda untuk menjadi seorang programmer C yang berkualitas dengan memahami esensi dan konsep yang terdapat di dalam bahasa C. Pembahasan dalam buku ini juga disertai dengan teori dan implementasi C ke dalam pemrograman port, baik paralel maupun serial; yaitu untuk *tampilan 8 buah LED*, *membangkitkan speaker PC*, *menggerakkan motor DC dan Stepper*, dan juga *tampilan LCD dua baris 16 karakter*. Penulis berharap buku ini dapat menjadi referensi yang memudahkan Anda dalam mempelajari pemrograman C secara cepat.

Tentang Penulis

I Made Joni

Penulis dilahirkan pada 1 Juni 1972 di Karangasem, Bali. Setelah menyelesaikan pendidikan sarjananya di jurusan Fisika Universitas Padjadjaran pada bidang instrumentasi elektronika, penulis melanjutkan pendidikan Master of Science di School of Physical Science Jawaharlal Nehru University, New Delhi, India; pada bidang Theoritical Physics. Sejak tahun 2001 penulis aktif menjabat sebagai dosen pada bidang instrumentasi elektronika di jurusan Fisika Universitas Padjadjaran. Saat ini, penulis sedang melakukan riset *Simulation on Chemical Control Process for nanoparticles Production* di Hiroshima University, Jepang; dalam rangka memperoleh gelar doktoral. Bidang keahlian yang ditekuni adalah sistem instrumentasi elektronika (Microcontroller, PLC, PC-Based Instrumentation) dan Simulation Process.

Budi Raharjo

Penulis adalah seorang profesional yang berkecimpung dalam bidang teknologi informasi. Penulis telah bertahun-tahun bekerja di PT. Sigma Delta Duta Nusantara (SDDN), Bandung; sebagai *software engineer*. Saat ini, penulis aktif membantu PT. Telekomunikasi Indonesia dalam pengembangan aplikasi CCIS (Corporate Customer Information System). Beberapa buku yang pernah penulis susun dan telah dipublikasikan oleh penerbit yang sama adalah *Memahami Konsep SQL dan PL/SQL Oracle*, *Pemrograman PL/SQL Oracle*, *Pemrograman C++Builder*, *Pemrograman C++*, dan *Teknik Pemrograman Pascal*. Sebagai pengembang aplikasi, penulis juga merupakan pemegang beberapa sertifikat bertaraf internasional, diantaranya: **C++ Programmer** (MASTER Level), **Computer Programmer**, **Java Programmer**, **Database Administrator (DBA)**, dll.