
Modul Kuliah Pemrograman Dasar

Oleh :

dwi sakethi

PENGRAJIN SISTEM INFORMASI

0816-403-432 ATAU 0811-725-1106

dwijim@fmip.unila.ac.id

www.staff.unila.ac.id/dwijim

www.dwijim.wordpress.com

tulisan niki dipun serat ngangge L^AT_EX

MBANDAR LAMPUNG

2019

Daftar Isi

I	Pengantar Pemrograman	v
1	Memahami Pemrograman?	1
1.1	Sistem Informasi Berbasis Komputer	2
1.2	Kemampuan Komputer	4
1.3	Pemrograman	5
1.4	Sulitkah <i>Programming</i> itu?	6
1.5	Ketrampilan yang Dibutuhkan	7
1.6	Tingkatan Bahasa Pemrograman	8
1.7	Pemrograman yang Baik	11
1.8	Contoh-Contoh Program	16
1.9	Games	17
2	Flowchart	19
2.1	Pengantar	19
2.2	Simbol	19
II	Bahasa Pemrograman	21
3	Pengantar	23
3.1	Apa dan Mengapa Bahasa C	23
3.2	Proses Pembuatan Program pada Sistem Operasi Debian GNU Linux	25
3.3	Proses Pembuatan Program pada Sistem Operasi Debian GNU Linux	25
3.4	Proses Pembuatan Program pada Sistem Operasi Microsoft Win- dows	26

3.4.1 Instalasi dan Konfigurasi	26
4 Struktur Bahasa C	37
III Kemampuan Dasar Komputer	39
5 Menyimpan Suatu Nilai	41
6 Melakukan Operasi Matematika	43
7 Membandingkan Dua Buah Nilai	45
7.1 Pengantar	45
8 Perulangan	47
8.1 Contoh Aplikasi	47
9 Counter dan Accumulator	51
9.1 Pengertian	51
IV Aplikasi	53
10 Bilangan Terbesar	55
10.1 Pengertian	55
10.2 Contoh Implementasi	55
11 Metode Newton	57
11.1 Pengertian	57
11.2 Contoh Implementasi	57
12 Bilangan Prima	59
12.1 Pengertian	59
12.2 Contoh Implementasi	59
13 Bubble Short	61
13.1 Pengertian	61
13.2 Contoh Implementasi	61

Kata Pengantar

Catatan kuliah ini merupakan catatan yang baru mulai ditulis. Pada semester-semester sebelumnya sebenarnya sudah ada bahan akan tetapi mulai tahun semester genap 2008/2009 ini ada perubahan-perubahan terutama setelah adanya pelatihan Struktur Data. Juga mengikuti ungkapan yang menyatakan bahwa untuk menjadi lebih baik perlu perubahan walau perubahan tidak selalu membawa ke arah yang lebih baik.

Satu hal lain yang berbeda dengan semester-semester sebelumnya adalah penghayatan bahwa filosofi **mengajar dengan cinta** merupakan suatu keniscayaan. Selain itu entah mengapa juga, apakah ini suatu *misteri* ataukah merupakan kesadaran baru bahwa *resource sharing* sebagai lahan berbuat kebaikan berpacu dengan ritme jalan menuju perumahan abadi (kuburan).

Dokumen ini dapat ditelusuri di **www.dwijim.wordpress.com**. Semoga bermanfaat ... Tulisan ini kupersembahkan untuk semua mahasiswaku tanpa kecuali dan untuk mereka yang ingin belajar Pemrograman Web.

Bagian I

Pengantar Pemrograman

Bab 1

Memahami Pemrograman?

Good programmer write code for
machine, **great programmer**
write code for other
programmer.

- web artisan -

Sumber: <https://id-laravel.com/>

1.1 Sistem Informasi Berbasis Komputer

Sebagian besar dari pembaca tentu sudah sering mendengar istilah sistem informasi berbasis komputer. Ada empat komponen di dalam sistem berbasis komputer yaitu ([Englander, 2014]):

1. Data

Data adalah gambaran mentah dari fakta dan observasi. Data ini kemudian akan diproses oleh sistem komputer menjadi informasi. Hal ini menjadi alasan utama keberadaan komputer sampai dengan saat ini. Data dapat dinyatakan dalam berbagai macam bentuk seperti: bilangan, tulisan, gambar, suara. Namun semua itu kemudian akan dikonversi menjadi bilangan di dalam sistem komputer.

2. Perangkat keras (*hardware*)

Perangkat keras memproses data dengan menjalankan perintah-perintah, menyimpan data, memindah data dan informasi di antara berbagai peralatan masukan dan keluaran, dimana dengan hal ini memungkinkan pemakai untuk mengakses sistem dan informasi.

3. Perangkat lunak (*software*)

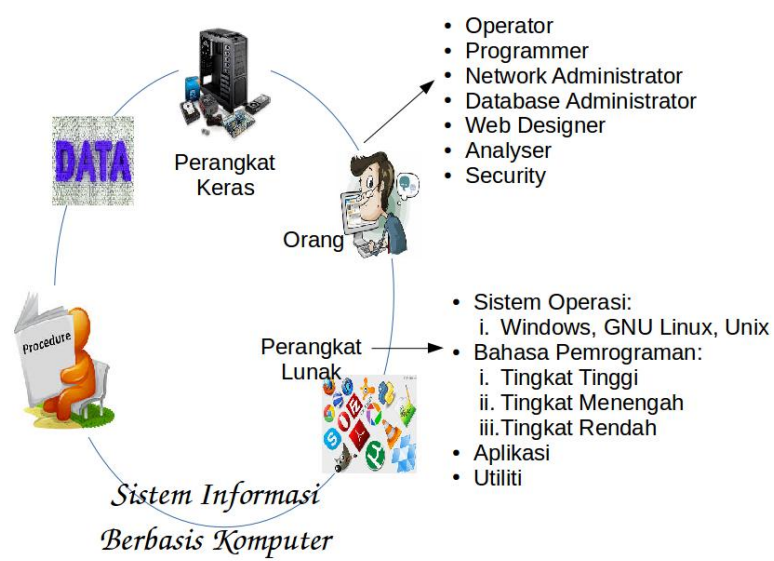
Perangkat lunak mencakup sistem dan program aplikasi yang mengandung instruksi yang akan dijalankan oleh perangkat keras. Perangkat lunak menentukan pekerjaan yang harus dilakukan dan mengendalikan beroperasinya sistem.

4. Komponen komunikasi

Sistem informasi pada komputer modern tergantung kepada kemampuannya untuk berbagi proses operasi dan data di antara komputer yang berbeda-beda, demikian juga dengan pemakainya, dengan lokasi yang berdekatan ataupun berjauhan. Komunikasi data menyediakan sarana ini.

Akan tetapi tinjauan ini menitikberatkan kepada arsitektur sistem komputer sehingga berbeda dengan pendapat lain.

Ada juga yang berpendapat bahwa di dalam sistem berbasis komputer terdapat lima komponen seperti pada Gambar 1.1:



Gambar 1.1: Sistem informasi berbasis komputer

1.2 Kemampuan Komputer

Komputer memiliki beberapa kemampuan dasar, yaitu ([Englander, 2014]) kemampuan operasi dasar dan bahasa tingkat tinggi dasar. Kemampuan operasi dasar (Basic Operation) meliputi:

1. Input/output

Komputer dapat menerima masukan data melalui alat masukan dan mencetak atau menampilkan sesuatu pada alat keluaran. Jika dalam kehidupan sehari-hari, dapat diumpamakan dengan pertanyaan-pertanyaan: siapa namamu, berapa nomor telponmu, berapa ukuran lebar, berapa ukuran panjang. Data-data ini akan disimpan dengan baik di dalam ingatan komputer.

2. Basic arithmetic and logical calculations

Komputer juga dapat melakukan perhitungan aritmetika sederhana dan proses logika. Seperti menjumlahkan, mengalikan, mengurangi, logika and, or dan lain sebagainya.

3. Data transformation or translation

Komputer mampu mengubah suatu nilai ke bentuk lain, misalnya nilai yang bersifat bilangan, diubah ke dalam nilai yang bersifat tulisan. Selain itu, komputer juga mampu menterjemahkan, dalam hal ini menterjemahkan perintah dalam bahasa pemrograman ke dalam bahasa yang dimengerti oleh komputer.

4. Data sorting

Di dalam perangkat lunak tertentu, komputer melakukan proses pengurutan. Namun dalam kondisi lain, pengurutan harus dilakukan secara detail, tidak cukup dengan satu perintah.

5. Searching for data matches

Komputer dapat melakukan pencarian kecocokan suatu data dengan data lain. Tingkat kemampuan ini berbeda-beda tergantung bahasa program atau perangkat lunak yang digunakan.

6. Data storage and retrieval

Komputer dapat menyimpan suatu data dan mengambilnya kembali dengan cepat dan tepat.

7. Data movement

Komputer juga memiliki kemampuan untuk memindahkan data, baik dalam bentuk teks atau pun khusus berkas data. Seperti misalkan dalam penyisipan dan atau penambahan data.

Sedangkan kemampuan dasar bahasa tingkat tinggi (Basic High-Level Language Constructs):

1. Input/output (including file storage and retrieval)
2. Arithmetic and logical assignment statements
3. True/false decision branching (IF-THEN-ELSE or IF-GOTO)
4. Loops and/or unconditional branching (WHILE-DO, REPEAT-UNTIL, FOR, GOTO)

1.3 Pemrograman

Beberapa hasil pencarian tentang definisi pemrograman menggunakan [geevv.com](http://www.geevv.com):

1. <http://interactivepython.org>
Programming is the process of taking an algorithm and encoding it into a notation, a programming language, so that it can be executed by a computer.
2. <http://www.businessdictionary.com>
The process of developing and implementing various sets of instructions to enable a computer to do a certain task.

3. <https://www.techopedia.com>

A programming language is a computer language engineered to create a standard form of commands. These commands can be interpreted into a code understood by a machine. Programs are created through programming languages to control the behavior and output of a machine through accurate algorithms, similar to the human communication process.

4. Menurut [Wang, 2007]: Programming is nothing more than writing step-by-step instructions telling the computer exactly what you want it to do. Because **computers are stupid**, they require exact instructions, and this limitation is what makes programming so time consuming.

Program komputer akan mengolah data yang dimasukkan ke dalam sistem menjadi suatu informasi atau keluaran. Seperti contoh pada Gambar 1.2:

Type of Program	Input	What the Program Does	Output
Word processor	Characters you type from the keyboard	Formats the text; corrects spelling	Displays and prints neatly organized text
Game	Keystrokes or joystick movements	Calculates how fast and far to move a cartoon figure on-screen	Moves a cartoon figure on-screen
Stock-market predictor	Current and past prices for stocks	Tries to recognize trends in a stock's price fluctuations	Predicts the future price of a stock
Missile guidance program	Current location of the missile and the target	Calculates how to make the missile's location and the target's location coincide	Corrects the trajectory so that it stays aimed at the target
Optical character recognition (OCR)	Text from a scanner	Recognizes shapes of characters	Converts scanned text into a text file that a word processor can edit
Web browser	HyperText Markup Language (HTML) codes on other computers	Converts the HTML codes into text and graphics	Displays Web pages on-screen

Gambar 1.2: Masukan-Proses-Keluaran

1.4 Sulitkah *Programming* itu?

Ketika ada mahasiswa jurusan komputer, jika ditanyakan kepadanya, "Apakah Anda suka membuat program?". Mungkin ada atau malahan banyak yang merasa tidak bisa dan kemudian tidak suka, takut untuk membuat program. Jika

memungkinkan ingin rasanya menghindari untuk membuat program (*coding*). Perhatikan petikan berikut [Wang, 2007]:

Anyone can learn to program a computer. Computer programming doesn't require a high IQ or an innate proficiency in advanced mathematics. Computer programming just requires a desire to learn and the patience never to give up.

Tentu saja kemudian menjadi hal yang sangat ajaib ketika ada orang yang merasa 'tidak bisa membuat program'.

1.5 Ketrampilan yang Dibutuhkan

Menulis atau membuat program bukanlah pekerjaan yang mudah, akan tetapi juga tidak cukup alasan untuk mengatakannya sebagai sesuatu yang sulit. Dalam menulis program, tidak membutuhkan bakat alam atau ketrampilan bawaan, seperti misalnya menyanyi atau melukis. Dalam menyanyi atau melukis, seseorang yang tidak memiliki bakat bawaan akan terasa sulit untuk melakukannya. Namun berbeda dengan menulis program komputer. Ada 4 ketrampilan yang dibutuhkan di sini, yaitu :

1. Perhatian terhadap hal-hal yang rinci. Komputer itu barang yang bodoh tidak dapat dipercaya (*incredibly stupid*). Pemrogram tidak dapat hanya menjelaskan 3/4 bagian dari proses dan kemudian berkata, "Kamu ngerti kan dengan apa yang aku mauin ?". Jika dalam penulisan mesti ada ; maka harus ditulis. Karena variabel harus dideklarasikan, maka variabel pun harus dideklarasikan.
2. Bersikap bodoh. Sebagaimana tadi sudah disebutkan bahwa komputer bodoh tidak dapat dipercaya. Komputer hanya mengerjakan apa yang betul-betul diperintahkan kepada mereka, tidak lebih dan tidak kurang. Komputer tidak bisa diberikan mie rebus dan kemudian disuruh membaca petunjuk yang ada dibungkusnya, maka kemudian komputer akan membuat mie rebus.
3. Ingatan yang baik. Banyak hal yang harus diingat ketika membuat program. Mulai dari aturan penulisan, fungsi dan perintah yang digunakan,

parameter untuk suatu fungsi yang sudah dibuat, nama-nama variabel yang digunakan dan lain sebagainya.

4. ability to abstract, think on several levels. This is probably the most important skill in programming. Computers are some of the most complex systems we've ever built, and if while programming you had to keep in mind every aspect of the functioning of the computer at all levels, it would be a Herculean task to write even a simple program.

Apakah ada orang yang berfikir bahwa dia tidak memiliki bakat untuk berenang? Membuat program itu seperti berenang, menari dan bermain sulap [Wang, 2007], serta menuliskan bahwa untuk mempelajari *programming* yang dibutuhkan adalah:

1. Hasrat yang besar (*desire*).

Dalam membuat program, akan banyak mendapatkan masalah. Terkadang masalahnya sederhana namun perlu waktu yang lama untuk mendapatkan penyelesaiannya. Jika tidak ada keinginan/hasrat yang besar, bisa jadi *programming* kemudian akan ditinggalkan begitu saja dan menyerah.

2. Keseriusan (*curiosity*).

Meskipun *programming* dapat juga dikatakan menjadi suatu hal yang *funny* tapi perlu keseriusan.

3. Imajinasi (*imagination*).

Program komputer yang akan dieksekusi menggunakan komputer, dapat diperkirakan terlebih dulu hasilnya sebelum proses eksekusi. Tentu saja ini membutuhkan data imajinasi.

1.6 Tingkatan Bahasa Pemrograman

Sebagai sebuah tradisi, bahasa pemrograman telah banyak dikembangkan. Bahasa pemrograman dibagi tingkatannya sesuai dengan kedekatan kepada bahasa manusia. Bahasa pemrograman ini dibagi menjadi tiga tingkatan [Gookin, 2013]:

1. *High-level languages.*

Bahasa pemrograman tingkat tinggi (*High-level languages*) merupakan bahasa pemrograman yang paling mudah dibaca, perintah-perintahnya menggunakan kata-kata dan phrasa yang biasa digunakan oleh manusia (khususnya dalam bahasa Inggris). Bahasa pemrograman ini dapat dipelajari dengan cepat namun sering ada kendala dalam hal fleksibilitas.

Bahasa pemrograman ini sangat banyak, di antaranya: C, C++, Java, PHP, Perl, Python, Go. Bahasa yang sudah agak jarang pemakainya: Fortran, Cobol, Pascal, Clipper.

Contoh potongan program menggunakan bahasa Java:

```
// mendeklarasikan variabel bilangan dengan tipe string
String bilangan;

// mendeklarasikan variabel lain
String kiri, tengah, kanan;

Scanner input_program = new Scanner (System.in);
System.out.print("Masukan suatu bilangan : ");
bilangan = input_program.nextLine();
System.out.println("Bilangan Anda : " + bilangan);

// cara yang digunakan di sini masih menggunakan
// cara praktis namun kurang efisien

// mengambil karakter paling kiri
kiri = bilangan.substring(0,1);

// mengambil karakter yang di tengah
tengah = bilangan.substring(1,2);

// mengambil karakter paling kanan
kanan = bilangan.substring(2,3);
```



```
System.out.println("Hasil membalik bilangan");

// mencetaknya dibalik: kanan, tengah, kiri
System.out.println(kanan+tengah+kiri);
```

2. *Low-level languages.*

Bahasa pemrograman tingkat rendah (*Low-level languages*) sangat samar-samar, umumnya sangat sedikit mengandung kata-kata yang mudah dibaca dan dipahami oleh manusia. Dengan bahasa ini, pengguna dapat melakukan akses langsung ke perangkat keras komputer dan karenanya proses ini dapat dilakukan dengan sangat cepat. Kekurangannya adalah waktu pengembangan suatu sistem dengan bahasa ini sangat lama karena segala sesuatunya harus dikerjakan dari nol (*done from scratch*). Contoh bahasa assembly. Berikut ini contoh program yang ditulis menggunakan bahasa tingkat rendah. Contoh diambil dari sumber di <https://enginedesigns.net/retroassembler>:

```
sei           //Disable interrupts by setting a flag.

lda #$7f      //Turn off Timer interrupts on both CIA chips.
sta $dc0d
sta $dd0d

lda $dc0d     //Clear queued CIA IRQs by reading the registers
lda $dd0d
asl $d019     //Clear VIC interrupts too, just in case.

//Now that the system's old interrupts are cleared, we can
//set up our own interrupt, that will execute a small chunk
//of code on every screen refresh.

lda #$01      //Request for a Raster Interrupt that we need.
sta $d01a
```

```

lda #<irq    //Point the IRQ Vector to our custom IRQ routine
ldx #>irq    //that will be executed on every screen refresh
sta $0314    //IRQ is short for Interrupt ReQuest.
stx $0315

```

3. *Midlevel languages.*

Bahasa pemrograman tingkat menengah (*Mid-level languages*) merupakan gabungan antara high- and low-level languages. Dengan demikian, bahasa pemrograman ini memiliki fungsi yang sangat serba guna dan program dapat didisain untuk mengerjakan apa saja. Bahasa C merupakan contoh dari bahasa pemrograman tingkat menengah.

1.7 Pemrograman yang Baik

Seperti apakah kriteria pemrograman yang baik ? Salah satu bahan acuan tentang ini, diambil dari tulisan Prof. Finkel (lagi-lagi) yang bisa diakses di : <http://www.cs.uky.edu/~raphael/checklist.html>

Checklist for good programming This checklist should help you write high-quality programs. Raphael Finkel, 8/17/2005

1. Identifiers : Make sure all your identifiers are meaningful.
 - (a) One-letter identifiers are almost never meaningful.
 - (b) Names like `flag` and `temp` are seldom meaningful. Instead of `flag`, consider naming the Boolean condition it checks for, such as `value-Found`.
 - (c) Consider multi-word identifiers, like `nameIndex`. Long identifiers (within reason) tend to be very readable.
2. Bare literals : Avoid numbers other than 0 and 1 and strings other than `""` in your program except when you define constants.

- (a) Don't use a literal integer as an array bound.
 - (b) Don't use a literal integer as a run parameter, such as a timeout or port number.
 - (c) Don't use literal integers to select menu entries.
 - (d) Don't use a literal integer to measure the size of a string or some data; use `sizeof()` and `strlen()` in C and C++ and `.length()` and `.size` in Java.
 - (e) Don't use a literal string for a file name. You may output literal strings, though.
 - (f) Don't use a literal integer to index into an array containing heterogeneous data.
 - (g) Don't declare an identifier with a name denoting a literal, such as "thirty".
3. Modularization : A program is built out of interacting components.
- (a) Don't put all your code into the `main()` routine.
 - (b) In fact, don't make any routine do too much work. If it's longer than about 50 lines, it is maybe too long.
 - (c) If you duplicate code several times, consider whether a loop would work better, or perhaps a subroutine.
 - (d) If you find you are indenting very deeply, you most likely aren't using subroutines when you should.
 - (e) Don't reinvent library routines (unless your assignment requires it). Look in the manuals to learn about `sprintf()` and `atoi()`, for instance.
 - (f) Use header files in C and C++ (header files have names ending `.h`) to define all constants needed by multiple files and declare all subroutines exported between files. But don't put the body of subroutines in header files (with the rare exception of inline subroutines).
4. Formatting : Your program should be easy to read.
- (a) Look at <http://geosoft.no/development/javastyle.html> for clear suggestions on formatting and other presentation issues. This reference

is specifically directed at Java, but it has value for other languages, too.

- (b) Try to restrict all your lines to 80 characters; many people view code in 80-column windows for historical reasons.
 - (c) Don't use both tabs and spaces for indentation, because not all text editors treat tabs as exactly 8 spaces.
 - (d) Do follow a consistent indentation pattern that reflects the program's control structure.
 - (e) Don't put lots of blank lines in your program. One blank line between subroutines is enough.
 - (f) Different operating systems terminate lines different ways. If you move between Win32 (which uses `/r/n`), Unix (which uses `/n`), and MacOS (which uses `/r`), reformat your file to use a consistent termination method.
 - (g) Don't set the executable bit (Unix) on your source files.
5. Coding : You want your coding to be clear, maintainable, and efficient, in that order. Some of the rules here are very specific; others are more general.
- (a) Don't use a sequence of if statements that have no else if only one can match; use else if.
 - (b) When you want to categorize text input, don't enumerate the possible first characters.
 - (c) Use shift operators instead of multiplication for constructing bit patterns.
 - (d) In a switch statement, always check for the default case. Likewise, in a sequence of if-then-else statements, use a final else.
 - (e) All system calls can fail. Always check the return code, and use `perror()` to report the failure.
 - (f) Booleans should always use the boolean type in Java, `bool` in C++, and `0/1` integers in C. Don't use characters `t` and `f`, and don't use `-1` and `1`.

- (g) Use loops to initialize data structures if possible.
- (h) Use each variable and each field of a structure for exactly one purpose. Don't overload them unless there is an excellent reason to do so.
- (i) Don't use the same identifier for both a type, a variable, and a file name, even if you change the capitalization. It's too confusing.
- (j) If you are modifying data with `htonl()` or a similar routine before network transmission, don't modify the data in place. Build a second data structure.
- (k) Try not to use global or nonlocal variables. Declare each variable in the smallest scope you can. There are legitimate uses of nonlocal variables, but make sure you really need them.
- (l) Shell and Perl programs should have their `# !` line as the first line of the file; otherwise, the line is just a comment.
- (m) Try to avoid coding special cases. You can often use pseudo-data or other data-structure methods that allow you to fold special cases into the regular cases.

6. Compilers : Let them help you find mistakes

- (a) Always invoke compilers with all warnings enabled. For C and C++, use the `-Wall` flag; for Java, use `-Xlint:all -deprecation`.
- (b) All Perl programs should run with the `-w` flag and should have `use strict`. All Perl cgi-bin scripts should have the `-T` flag, too.

7. The make utility : Use it, and use it well.

- (a) A makefile should always have a "clean" recipe, which should remove all files that can be reconstructed by other recipes in the makefile, including object and executable files.
- (b) If your project has multiple source files, the makefile should generate object (`.o`) files as needed and link them together.
- (c) The makefile should be written so that if you run `make` twice in a row, the second run does no recompilation.

- (d) Every recipe should create the file specified in its target.
 - (e) Every recipe should use every file specified in its prerequisite list.
 - (f) Learn to use rules for targets like `.c.o` to avoid repetitious makefiles.
 - (g) If you have just one C or C++ source file, the executable file should have the same name (without the extension `.c` or `.cpp`).
 - (h) Make sure you list all `.h` files as prerequisites where they are needed. Consider using `makedepend` to generate the prerequisite list for you.
8. Documentation : It's not just for the grader. It helps you as you write the program, too!
- (a) Add documentation as you write the program. You can always modify it as your design changes.
 - (b) Include external documentation : How does one compile and run the program, and what is it meant to do ? The external documentation could be in a separate file; for small projects, it can be a comment in the single source file.
 - (c) Include internal documentation : What algorithms and data structures are you using? An overview can be in a separate file, but usually internal documentation is placed on the specific routines, declarations, and steps that it describes.
 - (d) Check your whole program and documentation for spelling mistakes. It is impolite to turn in misspelled work.
 - (e) Check all your documentation (and output messages) for grammar mistakes.
 - (f) Programs are much more readable if you put a short comment on closing braces. For instance, the brace closing a conditional can have a comment like "if value looks good". A brace closing a loop can have a comment like "for each input line". A brace closing a procedure can have a comment just naming the procedure. A brace closing a class can have a comment saying "class" and then the name of the class.

1.8 Contoh-Contoh Program

Coba bayangkan Anda membuat program untuk mencetak hasil seperti berikut ini (cukup angkanya saja, tidak perlu dengan kotak-kotaknya): Program untuk

1	2	3	4	5	6	7
---	---	---	---	---	---	---

Tabel 1.1: Hasil keluaran

hasil seperti tersebut bisa dilihat pada contoh berikut :

```

/* -----
 * program untuk mencetak angka 1-7
 * dibuat hari ahad, 12 maret 2011
 * sambilan aja ...
 * nama file : cetak-angka-biasa.c
 * ----- */

#include <stdio.h>
// karena ada perintah printf maka ... include ini

#define AkhirLooping 7
// untuk mencetak angka 1-7 sesuai banyaknya hari
// karena tidak boleh ada magic number,
// maka dibuat definisi nilai ini

int main (void)
// program bahasa c minimal terdiri dari satu
// fungsi yaitu fungsi main()
{
    int bilangan;
    // deklarasi variabel bilangan

    for (bilangan=1;bilangan<=AkhirLooping;bilangan++)
    // membuat looping dari 1-7

    {
        printf("%d ",bilangan);
        // mencetak angka 1 2 3 4 5 6 7
    }
    printf("\n");
} // akhir fungsi main

```

Meningkat ke masalah berikutnya, yaitu bagaimana mencetak angka 1 pada suatu posisi tertentu dari posisi 1-7 seperti pada hasil keluaran tersebut. Perhatikan contoh berikut :

1	2	3	4	5	6	7
			1			

Tabel 1.2: Hasil keluaran

1.9 Games

1. Coba buatlah perintah-perintah yang mengarahkan seseorang pulang ke rumah Anda ...
2. Menukar isi dua buah gelas. Gelas berisi cairan, bagaimana supaya isinya bisa bertukar. Misal gelas A berisi cairan berwarna hitam dan gelas B berisi cairan bening.



Gambar 1.3: Menukar Isi Gelas

Bab 2

Flowchart

2.1 Pengantar










Flowchart atau diagram alir adalah bentuk grafis dari sebuah proses, seperti sebuah operasi pabrikan atau operasi komputer menandakan 10 langkah berbeda yang diambil sesuai langkah produksi sepanjang langkah produksi atau sepanjang langkah komputer [Boillot et al., 1997].

Flowchart adalah sebuah representasi bergambar yg menerangkan urutan langkah-langkah dalam sebuah program, orang dalam sebuah organisasi, atau lembar-lembaran dalam sebuah presentasi.

Langkah-langkah dalam suatu program, jika digambarkan dalam bentuk *flowchart* ini, akan lebih mudah untuk dipahami.

2.2 Simbol

Flowchart digambarkan dalam simbol-simbol yang sudah ditentukan dan masing-masing mempunyai arti yang sudah ditentukan juga.

SIMBOL	NAMA	FUNGSI
	TERMINATOR	Permulaan/akhir program
	GARIS ALIR (FLOW LINE)	Arah aliran program
	PREPARATION	Proses inisialisasi/ pemberian harga awal
	PROSES	Proses perhitungan/ proses pengolahan data
	INPUT/OUTPUT DATA	Proses input/output data, parameter, informasi
	PREDEFINED PROCESS (SUB PROGRAM)	Permulaan sub program/ proses menjalankan sub program
	DECISION	Perbandingan pernyataan, penyeleksian data yang memberikan pilihan untuk langkah selanjutnya
	ON PAGE CONNECTOR	Penghubung bagian-bagian flowchart yang berada pada satu halaman
	OFF PAGE CONNECTOR	Penghubung bagian-bagian flowchart yang berada pada halaman berbeda

Gambar 2.1: Simbol Flowchart

Sumber: [Anharku, 2009]

Bagian II

Bahasa Pemrograman

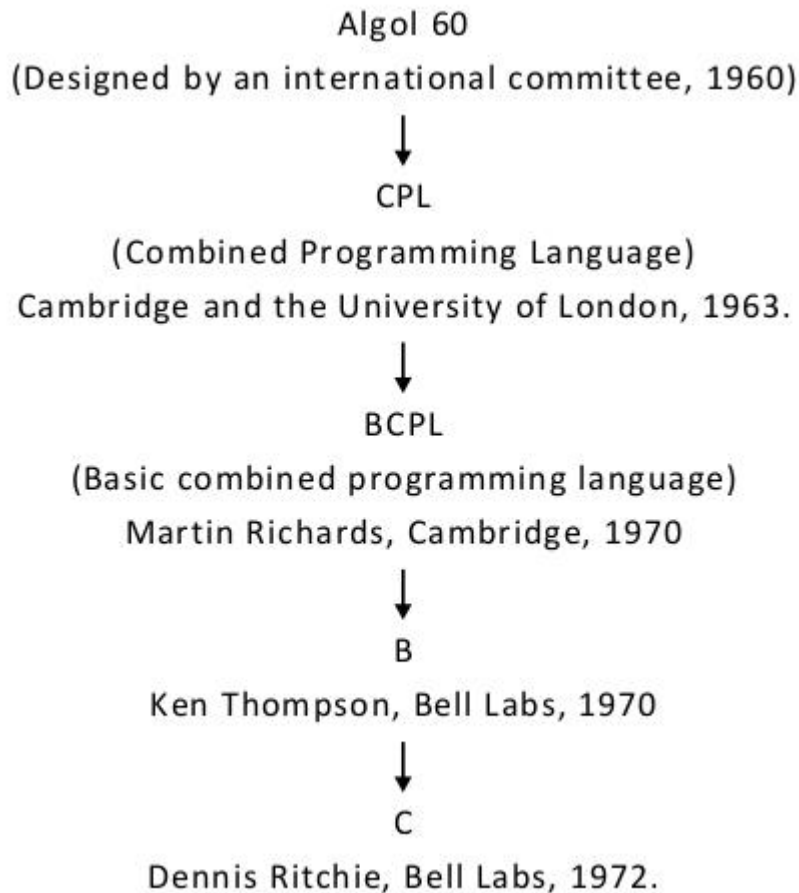
Bab 3

Pengantar

Di dalam materi ini, akan digunakan dua bahasa pemrograman, yaitu bahasa C dan Java. Dalam proses pembuatan program ini, yang paling penting adalah cara berfikir atau logika dari masalah yang ada, sehingga bahasa pemrograman yang digunakan, hanya sekedar perbedaan tata cara penulisan. Bahasa Java digunakan dengan tujuan sedikit membiasakan pengguna dengan model Pemrograman Berorientasi Obyek (*Object Oriented Programming*), meskipun tidak akan dibahas.

3.1 Apa dan Mengapa Bahasa C

Bahasa C dikembangkan pada sekitar tahun 1970-an oleh Dennis Ritchie dari Bell Laboratories [Gulati and Gulati, 2001] atau lebih tepatnya tahun 1972 [Gookin, 2013]. Bahasa C ini merupakan pengembangan dari pendahulunya yaitu B [Johnsonbaugh and Kalin, 1997]. Lantas, apakah kemudian berarti bahwa pendahulu dari bahasa B adalah bahasa A? Ternyata bukan A. Pendahulu dari bahasa B adalah BCPL (Basic Combined Programming Language). Silsilah bahasa C dapat dilihat pada gambar berikut [Gulati and Gulati, 2001].



Gambar 3.1: Silsilah Bahasa C

Bahasa C termasuk ke dalam bahasa pemrograman tingkat tinggi (*high-level programming language*) [Johnsonbaugh and Kalin, 1997]. Selain C, terdapat juga bahasa Fortran, Cobol, Pascal dan Lisp. Kemudian juga terdapat bahasa Basic, Clipper, PHP, Java, Python dan sebagainya. Bahasa Basic, Clipper dan Pascal menjadi bahasa pemrograman yang sudah cukup jarang lagi digunakan pada saat ini.

Bahasa C merupakan bahasa prosedural [Gulati and Gulati, 2001], sebagaimana disebutkan:

C is a procedural programming language, which means that you design and code programs as procedural modules. The procedural modules in a C program are called functions. Every C program

3.2. PROSES PEMBUATAN PROGRAM PADA SISTEM OPERASI DEBIAN GNU LINUX2

begins execution in a function named `main` and terminates when the main function returns.

Pada *jaman now* seperti sekarang ini, apakah masih relevan belajar bahasa C yang merupakan bahasa *jaman old*? Bahasa yang penuh variasi. Bahasa C ibarat bahasa Latin bagi bahasa pemrograman komputer. *Keywords* dan *functions* yang ada di dalam bahasa C, sebagian besar terdapat pula di dalam bahasa pemrograman lain. Sehingga seseorang yang sudah menguasai bahasa C, akan lebih mudah ketika belajar bahasa pemrograman lain. Bahkan terkadang dalam banyak buku yang mengajarkan pemrograman, penguasaan bahasa C menjadi salah satu persyaratan [Gookin, 2013]. Selain itu juga, bahwa pemrograman *microcontroller*, sistem operasi dan sebagian besar paket program, masih menggunakan bahasa C.

3.2 Proses Pembuatan Program pada Sistem Operasi Debian GNU Linux

3.3 Proses Pembuatan Program pada Sistem Operasi Debian GNU Linux

Untuk dapat menulis program dengan bahasa C dibutuhkan C Compiler demikian juga dengan C++. C Compiler di Linux salah satunya gcc (Debian 6.3.0-18+deb9u1) 6.3.0 20170516. Sedangkan untuk C++ ada g++ (Debian 6.3.0-18+deb9u1) 6.3.0 20170516.

3.4 Proses Pembuatan Program pada Sistem Operasi Microsoft Windows

3.4.1 Instalasi dan Konfigurasi

Salah satu alternatif perangkat lunak yang dapat digunakan untuk pembelajaran bahasa pemrograman C adalah Code::Blocks yang dapat diunduh di:

<https://www.fosshub.com/Code-Blocks.html?dwl=codeblocks-20.03-setup.exe>

Sebaiknya bagian yang diunduh adalah bagian yang di dalamnya terdapat: GCC/G++/GFortran compiler and GDB debugger

Biasanya hasil instalasi Code::Blocks ada di direktori dasar yaitu

C : \ProgramFiles\CodeBlocks

kemudian untuk *compiler* ada di

C : \ProgramFiles\CodeBlocks\MinGW\bin.

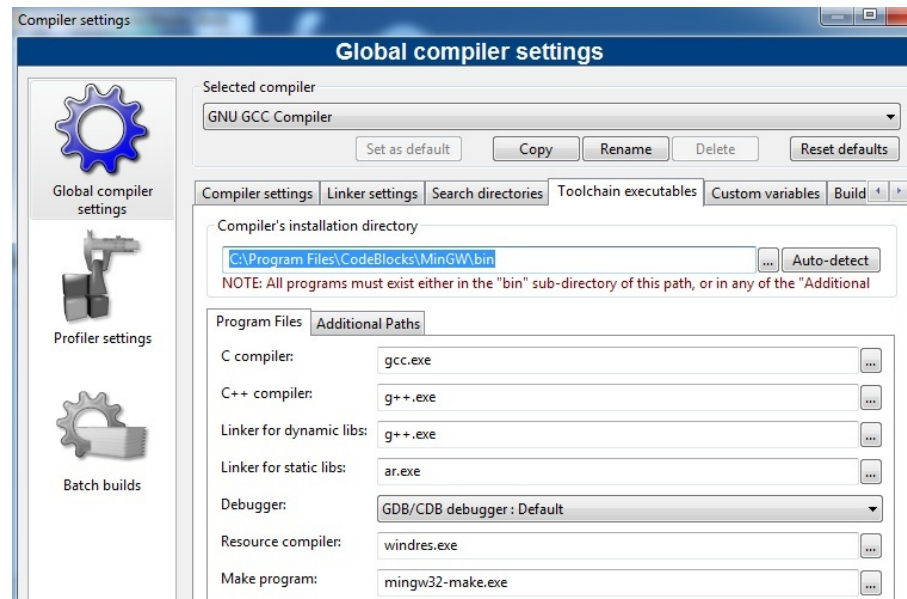
Sebelum digunakan, terlebih dahulu harus dilakukan konfigurasi, yaitu mengatur Compiler yang akan digunakan. Setelah Code::Blocks diinstal, perlu diingat dengan baik, direktori tempat menginstal program Code::Blocks ini. Untuk mengatur *compiler* pada Code::Blocks ini,

masuklah ke dalam program Code::Blocks, kemudian pilihlah menu Settings-Compiler-Toolchain executables. Kemudian pada bagian Compiler's installation directory, isi dengan

C : \ProgramFiles\CodeBlocks\MinGW\bin.

seperti pada Gambar 3.2.

3.4. PROSES PEMBUATAN PROGRAM PADA SISTEM OPERASI MICROSOFT WINDOW



Gambar 3.2: Pengaturan Direktori *Compiler*

Karena di dalam proses pembuatan program, dibutuhkan *compiler* ini, maka harus ditentukan letaknya ada di mana. Ini adalah direktori *default*. Direktori ini tentu saja disesuaikan dengan letak sebenarnya dari instalasi Code::Blocks-nya.

Karena proses pembuatan program akan dilakukan dengan dua macam cara, maka setelah pengaturan cara pertama tadi, diperlukan pengaturan model kedua. Cara pembuatan program yang kedua ini, merupakan cara yang lebih disarankan. Mengapa? Supaya prosesnya lebih terasa. Adapun pengaturan yang diperlukan ada pembuatan PATH untuk pemakai. Path itu apa? Path di sini adalah direktori-direktori yang akan dicari jika ada suatu program dijalankan sehingga program tersebut dapat ditemukan dan kemudian dijalankan.

Compiler yang digunakan, nama berkasnya adalah gcc.exe. Program ini dapat dijalankan dari layar *Console*. Pertama-tama, klik Start-Run kemudian isi dengan cmd. Atau dapat juga Start-cmd. Kemudian di posisi ini, ketikkan perintah gcc. Misalkan ketika di sistem komputer belum dilakukan pengaturan yang dibutuhkan, maka ketika dijalankan *compiler* GCC hasilnya akan tampak seperti pada Gambar 3.3.

Dengan asumsi program Code::Blocks sudah dinstal, maka akar masalahnya adanya karena belum adanya pengaturan PATH untuk menentukan letak

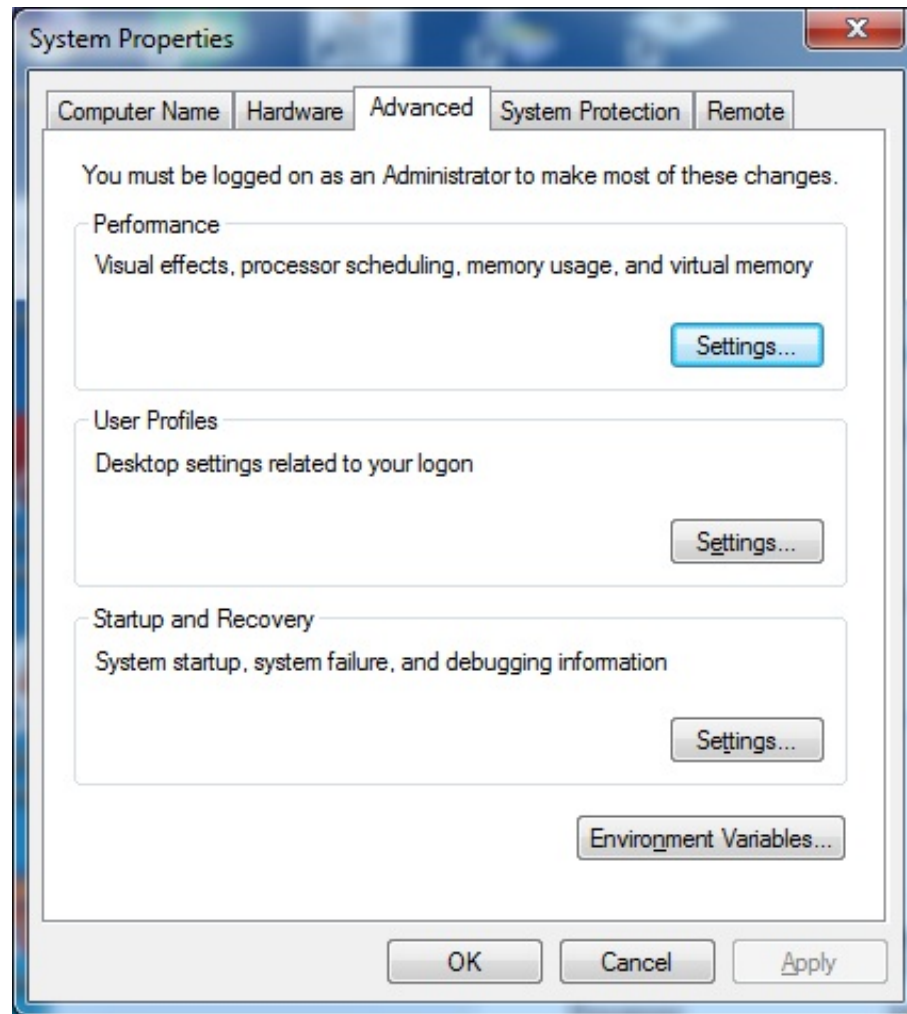


Gambar 3.3: Belum Ada Pengaturan Direktori *Compiler*

direktori dari program gcc.exe.

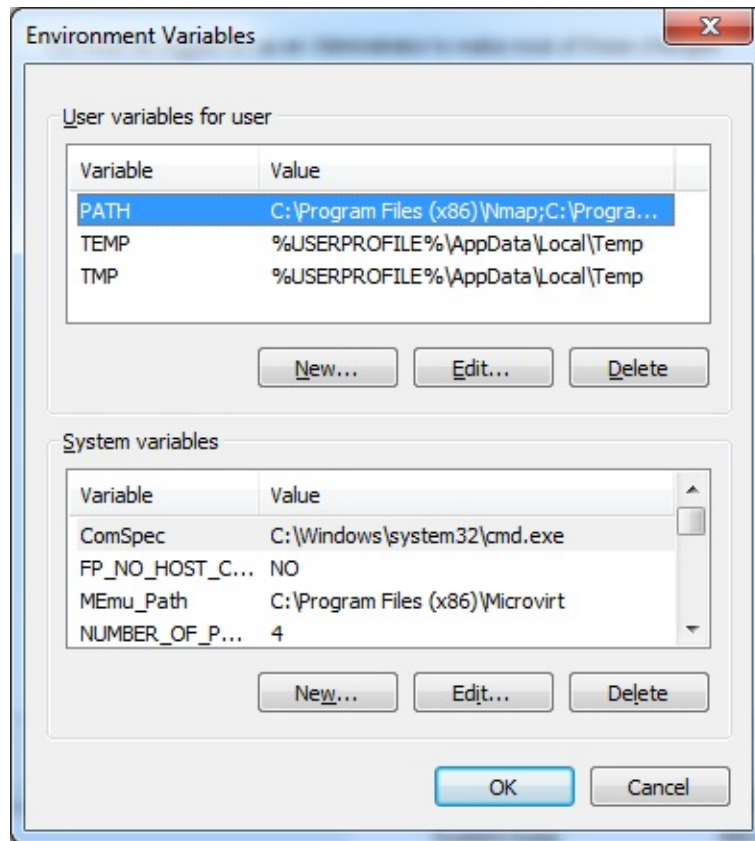
Maka perlu dilakukan pengaturan dengan memilih Start-Computer, kemudian klik kanan dan pilih Properties. Selanjutnya pilih Advanced system settings. Hasilnya seperti pada Gambar 3.4.

3.4. PROSES PEMBUATAN PROGRAM PADA SISTEM OPERASI MICROSOFT WINDOW

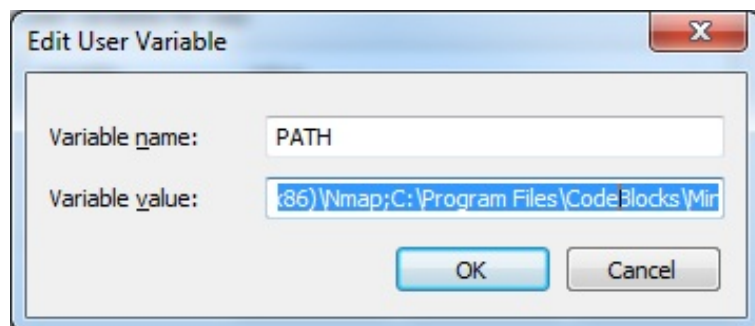


Gambar 3.4: Menu Pengaturan *Environment Variables*

Gambar 3.4 menunjukkan pengaturan yang dapat dilakukan. Dari sini, pilihlah *Environment Variables*. Kemudian pada bagian *Environment Variables* dipilih. Maka akan ada pilihan *Variables* apa yang akan diatur. Karena akan mengatur nilai *PATH*, maka pilihlah *Path* dan kemudian klik *Edit*.

Gambar 3.5: Pengaturan *Environment Variables*

Path ini sudah ada nilainya, nilai yang ada sebaiknya tidak dihapus, namun tinggal ditambahkan. Pada bagian paling kanan, ketikkan tanda ';' dan kemudian tuliskan letak dari gcc.exe. Menu ini dapat dilihat pada Gambar 3.6.



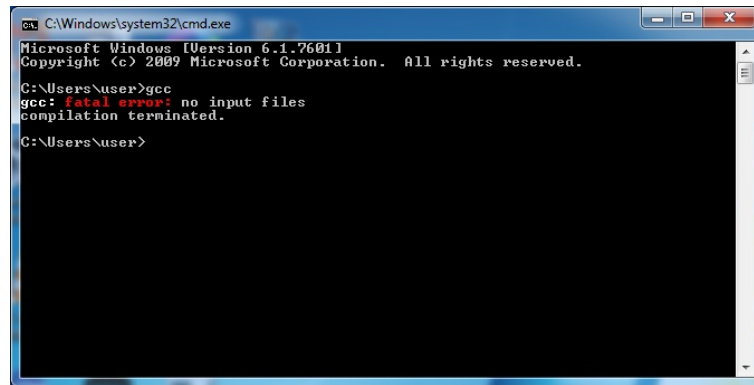
Gambar 3.6: Pengaturan Path

3.4. PROSES PEMBUATAN PROGRAM PADA SISTEM OPERASI MICROSOFT WINDOWS

Pengaturan user path contohnya dapat dilihat seperti ini:

C:\Program Files (x86)\Nmap;C:\Program Files\CodeBlocks\MinGW\bin

Pada Gambar 3.3 menunjukkan ada pesan kesalahan. Bagaimana dengan sekarang?



Gambar 3.7: Kompilasi di Console

Pada Gambar 3.7 terlihat pesan yang berbeda. Tampilan ini menunjukkan bahwa program gcc.exe sudah dapat dijalankan untuk melakukan kompilasi program.

Pembuatan Program

Proses yang dilakukan dalam pembuatan program meliputi [Gookin, 2013]:

1. Write the source code.

Penulisan program dapat dilakukan dengan menggunakan perangkat lunak *text editor*, seperti misalnya: Notepad++, Kate, Atom, vim, neovim, Sublime Text Editor, Visual Studio Code, dan sebagainya.

2. Compile the source code into object code.

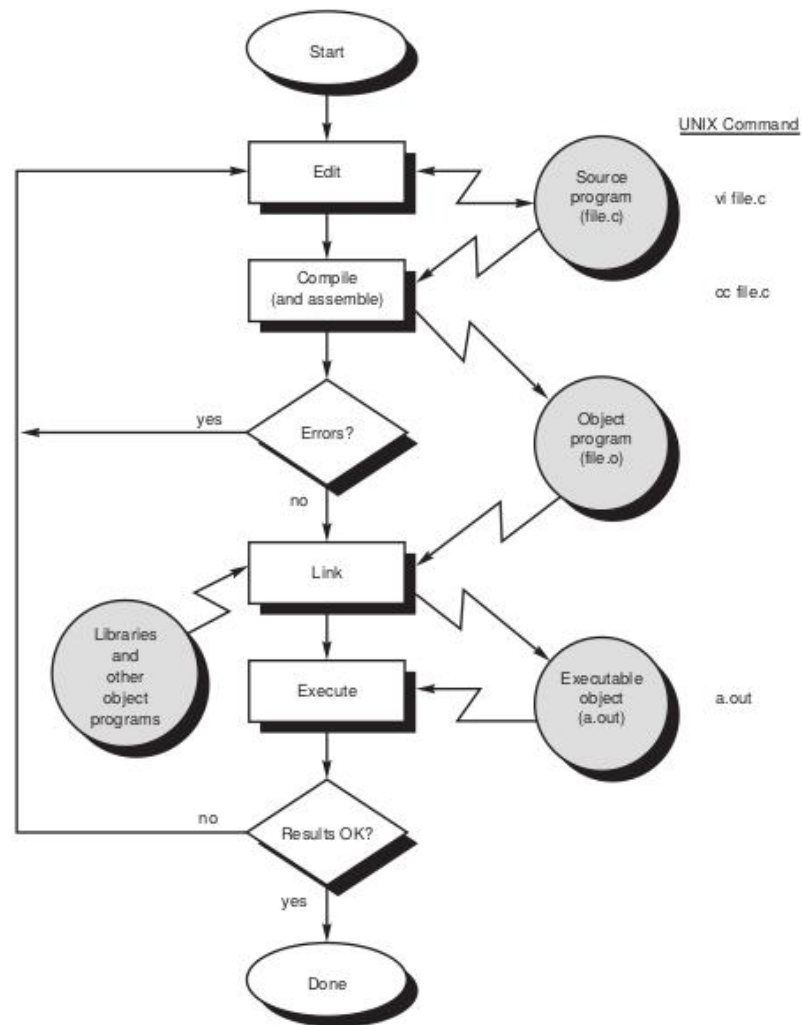
3. Fix errors and repeat Steps 1 and 2.

4. Link the object code with libraries to build the program.
5. Fix errors and repeat Steps 1 through 4.
6. Run and test the program.
7. Fix bugs by repeating the entire process.

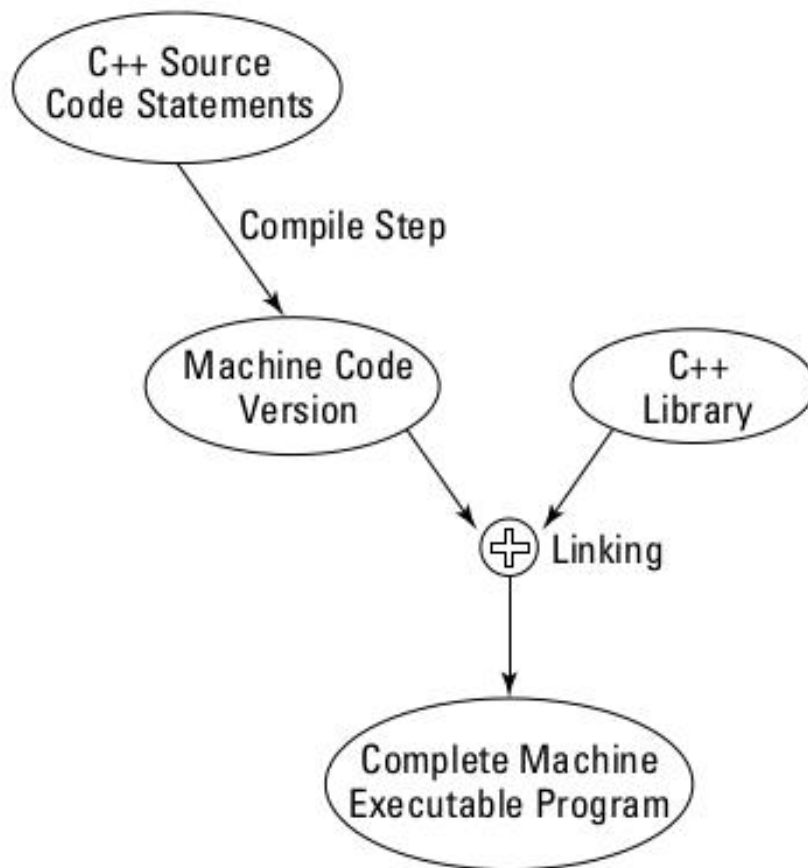
Akan tetapi, pada *compiler* modern, biasanya proses *compiling* dan *linking* sudah disatukan. Pada tulisan ini, tidak dibahas lagi proses pendahulu untuk mendapatkan *source code* yang akan ditulis. Proses dapat dilihat pada Gambar 3.9

Sedangkan menurut [Davis, 2015]

3.4. PROSES PEMBUATAN PROGRAM PADA SISTEM OPERASI MICROSOFT WINDOW



Gambar 3.8: Proses Pembuatan Program



Gambar 3.9: Proses Pembuatan Program

3.4. PROSES PEMBUATAN PROGRAM PADA SISTEM OPERASI MICROSOFT WINDOW

Misalkan ada pelanggan yang ingin menggunakan sistem untuk mencari bilangan terbesar. Berkas (*file*) apa yang akan diberikan kepada pelanggan tersebut? Apakah berkas dengan nama bilangan-terbesar.c? Apa masalahnya jika berkas ini yang diberikan?

Bab 4

Struktur Bahasa C

Bagian-bagian dalam bahasa pemrograman C [Gookin, 2013]:

1. *Keyword*.

Perintah dasar dalam bahasa C disebut dengan *keyword*. Tidak seperti bahasa manusia yang mengenal *nouns*, *verbs*, *adjectives*, and *adverbs*, bahasa C hanya mengenal *keyword*. Jumlahnya pun sedikit. Daftar lengkap dapat dilihat pada Gambar 4.1.

Table 3-1 C Language Keywords			
_Alignas	break	float	signed
_Alignof	case	for	sizeof
_Atomic	char	goto	static
_Bool	const	if	struct
_Complex	continue	inline	switch
_Generic	default	int	typedef
_Imaginary	do	long	union
_Noreturn	double	register	unsigned
_Static_assert	else	restrict	void
_Thread_local	enum	return	volatile
auto	extern	short	while

Gambar 4.1: *Keyword* Bahasa C

2. Fungsi.

3. Operator.
4. Variabel dan Nilai.
5. Pernyataan dan Struktur.
6. Komentar.

Bagian III

Kemampuan Dasar Komputer

Bab 5

Menyimpan Suatu Nilai

Bab 6

Melakukan Operasi Matematika

Kemampuan komputer lainnya adalah:

1. Melakukan Operasi Aritmatika.

Operasi aritmatika: tambah, kurang, bagi, kali, mod.

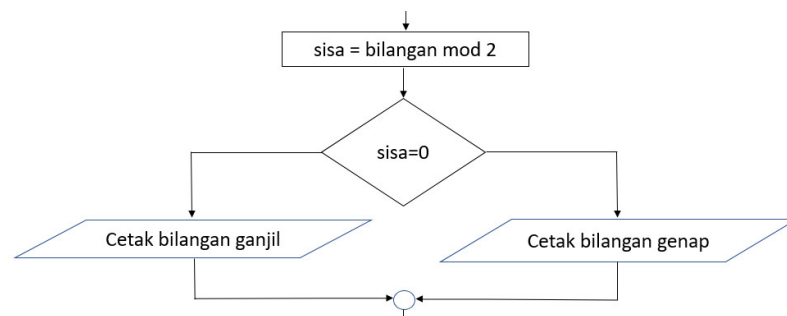
Bab 7

Membandingkan Dua Buah Nilai

7.1 Pengantar

Salah satu kemampuan komputer adalah melakukan perbandingan terhadap dua buah nilai, baik nilai yang sama-sama bertipe bilangan atau pun nilai yang sama-sama bertipe tulisan. Dengan perbandingan ini, akan dapat diketahui apakah kedua itu sama, mana yang lebih besar atau mana yang lebih kecil.

Perbandingan dua buah nilai dilakukan dengan perintah `if ... else ...`. Dalam bentuk diagram alir, proses perbandingan ini dapat digambar seperti pada Gambar 7.1 Jika proses ini diterjemahkan ke dalam bahasa pem-



Gambar 7.1: Bentuk if ... else

rograman akan menjadi seperti ini:

```
// menghitung sisa pembagian
```

```
sisapembagian = bilangan % 2;

// mencek apakah sisa pembagian sama dengan 0
if (sisapembagian==0)
{
    printf("Bilangan genap \n");
}
else
{
    printf("Bilangan ganjil \n");
}
```

Jika apa kondisi yang ada di dalam tanda kurung bernilai benar (*true*), maka nilai dua akan disimpan ke dalam variabel hasil. Sedangkan jika bernilai salah, maka nilai satu akan disimpan ke dalam variabel hasil.

Contoh lain misalkan dalam penentuan bilangan ganjil genap.

Bab 8

Perulangan

Salah satu kemampuan mendasar dari komputer adalah bahwa komputer dapat mengeksekusi sekumpulan perintah secara berulang-ulang [Kochan, 2005]. Dengan kemampuan ini, perintah yang mulanya sangat panjang jika ditulis, maka dengan menggunakan perulangan dapat dipersingkat.

Di dalam bahasa C, ada tiga cara untuk melakukan perulangan ini [Kochan, 2005].

1. The for Statement.

Untuk perulangan yang sudah jelas jumlah perulangannya, misalkan 30, 100, 1000 atau dapat juga berupa variabel misal n.

2. The while Statement.

Untuk perulangan yang belum jelas jumlah perulangannya, Perulangan akan berhenti tergantung kondisi.

3. The do Statement.

8.1 Contoh Aplikasi

Salah satu contoh penerapa konsep perulangan adalah dalam masalah pembuatan penanggalan pada suatu bulan. Contoh program dapat dilihat pada program berikut ini.

```
/* -----  
* program untuk membuat kalender pada
```

```

* suatu bulan tertentu
* ----- */

#include <stdio.h>
#include <math.h>

#define akhir_bulan 30

int main (void)
{
    int bulan, // bulan ke berapa
        hari_ke // sekarang hari ke berapa
        ;
    int tanggal_satu_hari_apa; // tanggal 1 jatuh
                                // pada hari apa ?
    int sekarang_hari_ke;
        // dalam proses sekarang ada pada hari apa
        // atau hari ke berapa ?

    printf("kalender bulan [1, 2, 3, ..., 12] ");
    scanf("%d",&bulan);

    printf("tanggal 1 jatuh pada hari [1, 2, ..., 7] ");
    scanf("%d",&tanggal_satu_hari_apa);

    hari_ke = 1; // posisi hari dimulai hari ke-1

    printf("  M  S  S  R  K  J  S\n");

    // tulis kosong sampai ketemu tanggal satu jatuh
    // pada hari apa
    for(hari_ke=1;hari_ke<=tanggal_satu_hari_apa-1;hari_ke++)
    {
        printf("  -");
        sekarang_hari_ke = hari_ke;
    }
    sekarang_hari_ke++;

    // dari posisi tanggal 1 yang jatuh pada hari apa
    // cetak tanggal sampai akhir bulan
    for(hari_ke=1;hari_ke<=akhir_bulan;hari_ke++)
    {

```

```
printf ("%3d",hari_ke);
sekarang_hari_ke++;

// jika sudah lebih dari hari ke-7 (nilainya 8)
// maka menulis tanggalnya pindah baris
if (sekarang_hari_ke==8)
{
    sekarang_hari_ke = 1;
    printf("\n");
}
}
```


Bab 9

Counter dan Accumulator

9.1 Pengertian

Bagian IV

Aplikasi

Bab 10

Bilangan Terbesar

10.1 Pengertian

10.2 Contoh Implementasi

Bab 11

Metode Newton

11.1 Pengertian

Metode Newton adalah metode pencarian akar suatu fungsi $f(x)$ dengan pendekatan satu titik, dimana fungsi $f(x)$ mempunyai turunan.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

11.2 Contoh Implementasi

Bab 12

Bilangan Prima

12.1 Pengertian

Metode Newton adalah metode pencarian akar suatu fungsi $f(x)$ dengan pendekatan satu titik, dimana fungsi $f(x)$ mempunyai turunan.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

12.2 Contoh Implementasi

```
/* -----  
contoh ini telah di-compile pada sistem operasi Linux  
dengan compiler gcc. nama file prima.c  
perintah compile : gcc prima.c -o prima  
jika tidak ada kesalahan, program dapat dijalankan dengan  
memberikan perintah:  
  
./prima  
  
#dwi sakethi 26 agustus 2018  
----- */  
  
// menyertakan library untuk memasukkan data dan mencetak  
// data ke layar  
#include <stdio.h>  
  
// program bahasa C harus memiliki fungsi main
```

```
int main()
{
    int bilangan;
    int sisa_pembagian;
    int bukan_prima,proses;

    // mencetak judul
    printf("Program Bilangan Prima \n");

    // menanyakan suatu bilangan
    printf("Masukkan suatu bilangan: ");
    scanf("%d",&bilangan);

    // mencek apakah bilangan mempunyai pembagi
    // antara 2 sampai dengan bilangan-1
    bukan_prima=0;
    for (proses=2;proses<=bilangan-1;proses++)
    {
        // menghitung sisa pembagian
        sisa_pembagian = bilangan % proses;
        if (sisa_pembagian==0)
        {
            bukan_prima=1;
            break;
        }
    }

    if (bukan_prima==0)
        { printf("Bilangan prima \n");}
    else
        { printf("Bukan bilangan prima \n");}
    // karena int main, maka harus ada return
    return 0;
}
```

Bab 13

Bubble Short

13.1 Pengertian

Bubble Short merupakan salah satu metode untuk mengurutkan data.

13.2 Contoh Implementasi

```
/* -----
contoh ini telah di-compile pada sistem operasi Linux
dengan compiler gcc. nama file buble.c
perintah compile : gcc buble.c -o buble -lm
jika tidak ada kesalahan, program dapat dijalankan dengan
memberikan perintah:

./buble

#dwi sakethi 26 agustus 2018
----- */

// menyertakan library untuk mencetak data ke layar
#include <stdio.h>

// misalkan banyaknya data maksimal yang akan diolah ada 100 data
#define jumlah_data_maksimum 100

int main()
{
```

```

int banyak_data;
int data[jumlah_data_maksimum];
int data_ke,data_setelah,bilangan_sementara;

// menghapus layar
system("clear");

// menanyakan banyaknya data
printf("Banyaknya data: ");
scanf("%d",&banyak_data);

// looping pemasukan data
for (data_ke=0;data_ke<banyak_data;data_ke++)
{
    // memasukkan data ke-?
    printf("Masukkan data ke-%d : ",data_ke+1);
    scanf("%d",&data[data_ke]);
}

/* pengurutan data dengan metode bubble sort */

/* -----
pembandingan dilakukan dari data pertama
sampai dengan data terakhir-1
karena perbandingan terakhir adalah perbandingan
antara data terakhir-1 dengan data terakhir
0-1 0-2 0-3 0-4
    1-2 1-3 1-4
        2-3 2-4
            3-4
terlihat bahwa nilai yang sebelah kiri,
nilai terbesarnya adalah 3 (untuk contoh 5 data)
sehingga batas loopingnya banyak_data-2
karena array dimulai dari angka 0,
maka akhir looping harus dikurangi lagi
----- */

/* -----
langkah pertama adalah membuat looping dengan
hasil angka seperti berikut:
0-1 0-2 0-3 0-4

```

```

    1-2 1-3 1-4
      2-3 2-4
        3-4
    ----- */

for (data_ke=0;data_ke<=banyak_data-2;data_ke++)
    for (data_setelah=data_ke+1;
        data_setelah<=banyak_data-1;data_setelah++)

/* mengurutkan data dari kecil ke besar */
    if (data[data_setelah]<data[data_ke])

/* ditukar tempatnya, misal data[0] = 11 dan data[1]=6
   maka ditukar sehingga menjadi data[0]=6 data[1]=11 */
    {
        bilangan_sementara = data[data_ke];
        data[data_ke]      = data[data_setelah];
        data[data_setelah] = bilangan_sementara;
    }

/* cetak data hasil sorting */
    printf("Data hasil sorting dengan Bubble Short: \n");
    for (data_ke=0;data_ke<=banyak_data-1;data_ke++)
        printf(" %2d : %2d \n",data_ke+1,data[data_ke]);

// karena int main, maka harus ada return
return 0;
}

```


Daftar Pustaka

- [Anharku, 2009] Anharku (2009). *Flowchart*. IlmuKomputer.Com, <https://ilmukomputer.org>.
- [Boillot et al., 1997] Boillot, M., Gleason, G., and Horn, L. W. (1997). *Essentials of flowcharting*. William C. Brown Pub, USA, fifth edition.
- [Davis, 2015] Davis, S. R. (2015). *Beginning Programming with C++ For Dummies*. John Wiley & Sons, Inc., Hoboken, New Jersey, second edition.
- [Englander, 2014] Englander, I. (2014). *The architecture of computer hardware, systems software, & networking : an information technology approach*. John Wiley & Sons, Inc, 111 River Street, Hoboken, NJ 07030-5774, fifth edition.
- [Gookin, 2013] Gookin, D. (2013). *Beginning Programming with C For Dummies*. John Wiley & Sons, Inc., Hoboken, New Jersey, first edition.
- [Gulati and Gulati, 2001] Gulati, M. and Gulati, M. (2001). *Everything You Wanted to Know about C*. Silicon Media Press, Regd. Off. I-19, Lajpat Nagar - II, New Delhi, first edition.
- [Johnsonbaugh and Kalin, 1997] Johnsonbaugh, R. and Kalin, M. (1997). *C for Scientists and Engineers*. Prentice-Hall Inc., Upper Saddle River, New Jersey 07458.
- [Kochan, 2005] Kochan, S. G. (2005). *Programming in C*. Sams Publishing, 800 East 96th Street, Indianapolis, Indiana 46240, third edition.
- [Wang, 2007] Wang, W. (2007). *Beginning Programming for Dummies*. John Wiley & Sons, Wiley Publishing, Inc. 111 River Street Hoboken, NJ 07030-5774, 4th edition.