

Seni dan Ilmu Pemrograman:
Upaya untuk menghasilkan programmer yang
handal dan program yang bermutu

Budi Rahardjo

Agustus 2005

Daftar Isi

Daftar Isi	1
1 Pendahuluan	5
2 Seni Pemrograman	9
2.1 Elegan dalam tampilan	10
2.2 Tetap elegan di belakang layar program	12
2.3 Seni dalam source code	13
2.4 Puisi dalam kode program	14
2.5 Easter Egg	14
2.6 Belajar dari para Maestro	15
3 Seni Komputer Lainnya	17
3.1 Nama Rekursif	17
3.2 Silicon Art	18

2

DAFTAR ISI

4 Ilmu Pemrograman

19

Bibliografi

21

Kata Pengantar

Bagian “Kata Pengantar” akan saya tulis di kemudian hari. Untuk sementara, silahkan langsung nikmati tulisan saya ini. Tentu saja saya mengucapkan terima kasih kepada beberapa pihak yang menjadi ide dari buku ini, dan pihak-pihak yang memberikan saran.

Komentar, koreksi, dan saran silahkan dikirimkan ke saya melalui email di br@paume.itb.ac.id.

Bandung, Agustus 2005

Budi Rahardjo

Bab 1

Pendahuluan

Aplikasi dari komputer sudah menjadi bagian dari kehidupan kita sehari-hari. Komputer yang dapat diprogram muncul dalam berbagai bentuk, mulai dari komputer pribadi (personal computer) sampai ke mesin cuci yang bisa kita “program”. Secara tidak sadar, “pemrograman” sudah menjadi bagian dari kehidupan kita.

Secara umum, aplikasi dari pemrograman ada banyak, seperti antara lain:

1. Aplikasi program komputer di perkantoran, toko, perusahaan;
2. Aplikasi di *embedded system*, seperti di handphone, kendali (*controller*) di pabrik;
3. Aplikasi di permainan (*games*) dan musik;
4. Aplikasi khusus di kedokteran, visualisasi data, penelitian.

Ketika kita berbicara tentang “pemrograman,” seberapa besar langsung mengasosiasikannya dengan ilmu komputer. Dengan kata lain pemrograman harus dipelajari secara formal. Di sisi lain, banyak juga orang yang menganggap bahwa pemrograman tidak membutuhkan ilmu khusus. Buktinya banyak programmer yang tidak memiliki latar belakang formal ilmu komputer. Mana yang benar? Perlukah sekolah komputer?

Ketika komputer pertama kali dikembangkan, belum ada jurusan ilmu komputer. Belum ada orang yang memiliki latar belakang pendidikan ilmu komputer. Yang ada adalah orang-orang yang senang bermain-main dengan

komputer, atau “hacker.” Di kemudian hari memang muncul ilmu komputer secara formal.

Jadi, apakah pemrograman itu ilmu atau seni? Jawaban dari pertanyaan ini adalah keduanya. Bahkan pakar komputer Charles Simonyi mengatakan bahwa *programming* adalah ilmu (*science*), seni (*art*), dan ketrampilan (*skill, trade*). Pengetahuan mengenai algoritma-algoritma adalah ilmu. Kemampuan membayangkan (*imagining*) hasil atau tampilan adalah seni. Kemampuan membuat kode adalah ketrampilan.

Pendidikan formal ilmu komputer mengajari siswa tentang cara-cara membuat program. Siswa diajari logika, mengurutkan langkah, dan menuliskan program. Dua aspek yang diajarkan dengan metoda ini, yaitu aspek keilmuan dan ketrampilan dari pemrograman. Namun sayangnya, terlupakan dalam pendidikan ini adalah aspek seni dari pemrograman. Hasilnya adalah siswa yang mampu membuat program akan tetapi program yang dihasilkan tidak memiliki nuansa seni.

Membuat program mengandung aspek desain. Di sinilah aspek seni muncul. Faktor manusia sangat kental dalam desain. Itulah sebabnya komputer atau robot masih belum dapat membuat program sendiri. Saya tidak tahu apakah Isac Asimov¹ setuju dengan pendapat ini.

Ada program yang dapat membantu kita dalam menghasilkan program, seperti *compiler* atau “yacc” (*yet another compiler compiler*). Dapatkah mereka disebut robot yang menghasilkan robot? Menurut pendapat saya, tidak! Program tersebut tidak melakukan pemikiran tentang desain (membuat *design decision*), akan tetapi hanya menghasilkan keluaran secara mekanistik seperti halnya sebuah baut dihasilkan oleh mesin bubut yang berbasis CNC.

Buku ini ingin membahas kedua aspek dari pemrograman, seni dan ilmu. Akan tetapi saya ingin memberi bahasan yang lebih ke aspek seninya. Sudah banyak buku yang membahas aspek teknis atau ilmunya. Tentu saja saya tidak ingin menganggap kecil sisi keilmuan dari pemrograman. Itulah sebabnya akan saya tampilkan juga sisi ilmu dari pemrograman ini.

Saya bukan orang pertama yang tertarik untuk membahas masalah seni dalam pemrograman. Donald Knuth² merupakan seorang pakar komput-

¹Isac Asimov banyak menulis cerita (novel) tentang robot. Robot, dalam ceritanya, memiliki perasaan seperti manusia. Salah satu karyanya yang terkenal adalah “*I, robot*” yang akhirnya dibuatkan filmnya, dengan Will Smith sebagai bintang utamanya.

²Informasi mengenai Donald Knuth dapat diperoleh melalui situs web pribadinya yang berada di <http://www-cs-faculty.stanford.edu/~knuth/>

er yang sangat serius dalam hal seni pemrograman. Dia membuat buku “*The Art of Computer Programming*” [Knu98] yang sangat terkenal tersebut. Bahkan keprofesoran dari Donald Knuth adalah di bidang *art of computer programming*.

Jika anda ingin belajar pemrograman, dengan bahasa tertentu, anda akan kecewa dengan buku ini karena buku ini lebih banyak membahas aspek filosofis dari pemrograman. Jika anda senang kepada hal-hal yang filosofis, mengawang-awang dan tidak membumi, maka buku ini cocok untuk anda. Kita perlu sedikit menghayal dan bermimpi.

Contoh-contoh yang saya pakai di sini menggunakan beragam bahasa pemrograman. Hal ini sebetulnya kurang baik. Ini seperti mengajari anak berbicara, akan tetapi bahasa yang digunakan berubah-ubah setiap harinya. Anak tersebut akan bingung³. Dalam konteks buku ini, anda mungkin bingung. Tidak apalah. Sekali-sekali (berkali-kali?) saya perlu membuat anda bingung.

³Ibu Inge dari Informatika ITB memberikan analogi seperti itu ketika saya ingin mengajarkan pemrograman dengan bahasa yang berbeda-beda.

Bab 2

Seni Pemrograman

*Programming is the ultimate field
for someone who likes to tinker.*

(Ray Ozzie - pengembang Lotus Symphony)

Mengapa saya bersikeras untuk secara khusus menulis mengenai seni pemrograman? Bab ini akan menjelaskan alasan-alasan saya.

Saya banyak melihat program buatan mahasiswa dan juga programmer “profesional” di Indonesia. Untuk sementara ini saya tidak membahas masalah fungsional dari program-program tersebut. Ini akan menjadi bahasan terpisah (mungkin di buku lain). Kita asumsikan saja bahwa program-program tersebut berjalan sesuai dengan fungsinya. Yang saya keluhkan adalah program tersebut tidak memiliki jiwa seni. Bahkan banyak program yang dibuat asal jalan dan “jorok.”

Saya memang menuntut program yang anda buat memiliki aspek seni. Mengapa demikian? Bukankah yang penting adalah fungsi dari program tersebut? Selama program tersebut bisa berjalan secara fungsional apa salahnya?

Baiklah, saya ambil beberapa analogi. Apa yang membedakan mobil BMW dengan angkot? Keduanya mobil bukan? Keduanya bisa membawa anda dari satu tempat ke tempat lain. Fungsi keduanya sama. Akan tetapi, angkot tentu saja bukan BMW. Desain dari mobil BMW sangat elegan.

Apa yang membedakan produk dari Apple Computer dengan komputer dan sistem operasi buatan perusahaan komputer lainnya? Apple Computer memperhatikan masalah desain. Produk mereka memiliki estetika. Elegan. Saya masih ingat apa yang dikatakan oleh Steve Jobs - pendiri Apple

Computer - dalam sebuah wawancara. Dia ditanya mengenai pendapatnya tentang Microsoft. Steve Jobs mengatakan bahwa dia tidak permasalahan kesuksesan Microsoft, hanya ...

"They (Microsoft) don't have taste!"

Nah, itu dia!

Software atau program komputer merupakan sebuah produk yang digunakan oleh manusia. Dia merupakan sebuah karya desain.

2.1 Elegan dalam tampilan

Ketika kita berbicara seni dari komputer atau pemrograman, biasanya kita mengasosiasikan dengan tampilan. Itulah sebabnya ada “perang agama” antara Microsoft, Apple Macintosh, dan UNIX (termasuk di dalamnya Linux, FreeBSD, dan berbagai varian lainnya) yang menggunakan X window.

Sebetulnya ketika kita berbicara mengenai tampilan, kita tidak hanya berbicara mengenai *Graphical User Interface* (GUI) saja. Jaman awal komputer tanpa GUI pun kita sudah berbicara mengenai desain tampilan. Mari kita ambil sebuah contoh program yang ditulis dalam bahasa C++.

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

int main()
{
    int x, y, z;
    cout << "Masukkan x\n";
    cin >> x;
    cout << "Masukkan y\n";
    cin >> y;
    z = x * y;
    cout << "z=" << z;
}
```

Program di atas meminta pengguna untuk memasukkan dua buah bilangan, mengalikan kedua bilangan tersebut dan memperagakan hasilnya.

Berikut ini contoh keluaran atau tampilan dari program tersebut. Perhatikan bahwa kata “unix\$” merupakan *prompt* dari *console* yang saya gunakan dan bukan bagian dari program.

```
unix$ ./a.out
Masukkan x
7
Masukkan y
5
z=35unix$
```

Program di atas terlihat berjalan dengan baik. Saya memasukkan nilai 7 dan 5. Hasil perkaliannya adalah 35. Kelihatannya program berjalan seperti yang diharapkan. Apa komentar anda terhadap tampilan dari program di atas?

Komentar saya adalah sebagai berikut. Ketika kita diminta untuk memasukkan nilai x , kursor turun ke bawah. Ini disebabkan adanya *newline* pada akhir *cout*. Apakah tidak lebih elegan apabila tampilannya seperti berikut ini?

```
Masukkan bilangan integer x: []
```

Selain memberikan informasi yang lebih jelas (bahwa yang harus dimasukkan adalah bilangan integer), pada tampilan tersebut, kursor berada di sebelah kanan dari tanda titik dua (:) dan dibatasi dengan sebuah spasi.

Demikian pula tampilan hasil perkalian kurang elegan. Kita tidak tahu apa itu variabel z . Untuk saat ini karena kita baru saja membaca kode sumber dari program itu dan kodenya sangat sederhana, maka kita masih ingat bahwa z merupakan perkalian dari dua bilangan x dan y yang kita masukkan. Akan tetapi jika kodenya kompleks dan sudah lama kita tidak membuka kode sumber dari program tersebut, kita akan dibuat bingung dengan keluaran seperti itu. Kemudian setelah menampilkan hasilnya (dalam hal ini adalah angka 35), tidak ada *newline* sehingga prompt menempel di sebelah kanan dari hasil perkalian.

Mari kita perbaiki kode program tersebut sehingga memiliki tampilan keluaran yang lebih baik. Berikut ini adalah salah satu alternatif perbaikan dari kode tersebut.

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

int main()
{
    int x, y, z;
    cout << "Masukkan bilangan integer x: ";
    cin >> x;
    cout << "Masukkan bilangan integer y: ";
    cin >> y;
    z = x * y;
    cout << "Hasil perkalian keduanya : " << z << "\n";
}
```

Keluaran dari program di atas menjadi seperti ini:

```
Masukkan bilangan integer x: 7
Masukkan bilangan integer y: 5
Hasil perkalian keduanya : 35
```

Bandingkan tampilan di atas dengan tampilan sebelumnya. Meskipun program memiliki fungsi yang sama, akan tetapi tampilan berbeda. Ini yang saya maksud dengan tampilan yang elegan.

2.2 Tetap elegan di belakang layar program

Jika demikian, maka kita buat program yang memiliki tampilan (*graphical user interface*) (GUI) yang bagus saja. Kode di belakangnya tidak perlu kita perhatikan. Ketika saya menuntut adanya desain yang elegan, saya tidak membatasi hal ini dalam tampilan saja. Saya justru ingin menekankan pentingnya seni pemrograman yang terjadi di belakang layar.

Steve Wozniak - salah seorang pendiri Apple Computer juga - disebut orang sebagai Mozart-nya desain rangkaian digital. *The Mozart of digital design*. Mengapa demikian? Steve Wozniak adalah desainer dari hardware komputer Apple []. Jika anda memiliki akses terhadap komputer Apple [] tersebut, coba perhatikan desain rangkaiannya. Buka kap penutup komputer tersebut dan perhatikan rangkaiannya. *Floppy disk controller* buatan Steve Wozniak menggabungkan beberapa komponen (chip) menjadi satu.

Desainer rangkaian sebelumnya memperkirakan hal tersebut tidak mungkin dilakukan. Rangkaianya harusnya kompleks dan memiliki ukuran yang besar sehingga tidak cocok untuk sebuah komputer pribadi. Steve Wozniak tidak saja dapat membuat desain yang kompak, akan tetapi juga membuatnya dengan style! Padahal, siapa *sih* yang mau melihat rangkaian dari komputer Apple][tersebut? Toh, asal komputer bisa jalan, orang tidak begitu peduli. Salah! Banyak orang yang peduli. Ah, mungkin tidak banyak, tapi ada.

Mari kita ambil contoh kode program perkalian dua bilangan integer yang sudah kita baca pada bagian sebelumnya. Perhatikan kode di bawah ini.

```
#include <iostream>
using namespace std;
int main() { int x, y, z;
cout << "Masukkan bilangan integer x: "; cin >> x;
    cout << "Masukkan bilangan integer y: "; cin >> y;
z = x * y; cout << "Hasil perkalian keduanya    : " << z << "\n"; }
```

Meskipun kode di atas dapat dijalankan dan hasil tampilannya juga bagus, akan tetapi kode tersebut susah dibaca. Tidak elegan sama sekali! Kacau! Mengapa kode tidak disusun dengan elegan?

Jika malas membuat kode yang elegan, ada beberapa tools yang dapat digunakan untuk membuat kode lebih mudah dibaca. Nama program tersebut biasanya seputar *source code pretty printing* atau *source code beautifier*.

2.3 Seni dalam source code

Program yang anda buat harus tetap elegan meskipun tidak banyak orang yang akan melihat *source code* dari program anda. Di jaman *open source* ini, dimana kode dari program disediakan secara terbuka dan boleh dibaca oleh banyak orang, akan makin banyak orang yang melihat kode buatan anda. Anda akan dinilai dari kode buatan anda. Jika anda “jorok” dalam membuat program, maka anda akan menjadi bahan tertawaan.

Adanya paradigma *free software* dan *open source* membuka harta karun pemrograman. Saya sering menelusuri kode yang dibuat oleh berbagai programmer. Ternyata banyak kode-kode yang lucu-lucu. Sungguh, kode yang

membuat kita tertawa. Saya akan tampilkan beberapa contoh kode tersebut.

Beberapa komentar tentang kode program (*source code*) antara lain bisa dilihat di daftar di bawah ini.

- Source code yang nyaman dibaca sehingga bisa kita pelajari.
- Algoritma (trik, optimasi) yang digunakan juga sering kali menarik.
- Kadang-kadang ada guyonan atau humor dalam source code.
- Source code yang buruk sukar dimengerti dan membuat sakit mata.

2.4 Puisi dalam kode program

Temuan pertama saya dengan seni pemrograman adalah adanya puisi dalam kode program. Puisi ini tidak dibuat dalam bentuk komentar, akan tetapi betul-betul ditulis dengan sintaks dari bahasa pemrograman tersebut. Untuk masalah puisi, bahasa perl merupakan rajanya.

Larry Wall - pembuat bahasa perl - merupakan orang yang penuh humor. Anda bisa mengunjungi situs webnya, <http://wall.org/~larry> untuk membuktikan hal ini. Itulah sebabnya bahasa perl banyak lucunya¹. Lagi-lagi ini merupakan contoh bahwa spirit dari pengembang bisa tercermin dalam produknya.

Sharon Hopkins membuat sebuah makalah yang menarik dengan judul “*Camels And Needles: computer Poetry Meets The Perl Programming Language.*” Dalam makalahnya ini dia mengumpulkan puisi-puisi yang ditulis dalam bahasa perl.

2.5 Easter Egg

Easter egg merupakan istilah yang digunakan untuk program yang ditanam di program hanya untuk lucu-lucuan. Program, atau lebih tepatnya fungsi,

¹Saya akui bahwa humor itu sangat subyektif dan terkait erat dengan kultur. Sesuatu yang lucu bagi seseorang mungkin tidak lucu, atau bahkan memuakkan, bagi orang lain. Saya berpendapat jika seseorang sudah mengerti humor / lawakan / joke yang diceritakan dalam bahasa lain, maka orang tersebut sudah memahami bahasa tersebut.

dari easter egg ini diaktifkan dengan memberikan kode tertentu kepada program yang bersangkutan. Misalnya, dengan menekan tombol “ctrl” “alt” “shift” dan “enter” secara bersamaan akan muncul daftar pengembang program tersebut. Atau lebih jauh lagi, muncul sebuah games. Pengguna harus mencari kode untuk mengaktifkan easter egg tersebut, sama seperti anak-anak yang mencari telur paskah.

Ada masalah dengan easter egg. Umumnya perusahaan tidak menyukai adanya easter egg ini karena (1) menghabiskan resources dari program (semestinya kode menjadi lebih kecil), (2) programmer yang dibayar untuk mengembangkan kode malah sibuk membuat easter egg (meskipun sang programmer dapat beralasan bahwa easter egg tersebut dibuat di luar waktu kerja), (3) ketidak-jelasan dampak yang ditimbulkan oleh easter egg tersebut. Saya pribadi tidak menyukai adanya easter egg ini.

2.6 Belajar dari para Maestro

Untuk memahami mengenai seni pemrograman ini ada baiknya kita lihat apa yang dilakukan oleh para programmer maestro. Banyak hal yang mungkin tidak sesuai dengan teori pemrograman yang diajarkan di sekolah. Bagaimana cara mereka mendesain program? Bagaimana bisa muncul ide? Berapa lama mengimplementasikan ide tersebut? Hambatan apa saja yang dialami? Dan seterusnya. Itulah pertanyaan-pertanyaan yang ingin kita ajukan kepada para maestro itu untuk melihat sisi seni dari pemrograman.

Sayangnya tidak banyak buku yang menceritakan cerita di belakang layar ini. Salah satu buku yang menarik untuk dibaca adalah buku karangan Susan Lammers [Lam86].

Bab 3

Seni Komputer Lainnya

Selain pemrograman, ada juga beberapa seni lain yang terkait dengan komputer. Orang-orang yang hobby dengan komputer, setan komputer atau hacker, memiliki sifat senang bermain-main (*playful*). Sayangnya hacker saat ini lebih banyak menjurus ke hal yang negatif dan merugikan orang lain. Padahal hacker sesungguhnya tidak demikian. Mereka senang bermain-main, tapi tidak merugikan orang lain.

3.1 Nama Rekursif

Salah satu kesukaan hacker adalah memilih nama yang aneh dan lucu untuk karyanya, dan juga untuk nama dirinya sendiri. Salah satu trik yang banyak digunakan dalam memilih nama adalah dengan menggunakan nama yang rekursif.

Ketika mencari nama untuk sistem operasi bebas (*free operating system*) yang akan dibuatnya, Richard Stallman¹ mencoba mencari nama yang lucu. Sistem operasi yang ingin dia buat merupakan sesuatu yang mirip dengan sistem operasi UNIX yang waktu itu sangat populer di perguruan tinggi. Akhirnya dia memilih “GNU” untuk nama sistem operasinya.

GNU merupakan contoh pemilihan nama yang rekursif karena kepanjangan dari “GNU” adalah “*GNU is Not Unix.*” Huruf “G” dalam kata

¹ Richard Stallman merupakan pengagas software bebas - free software yang sangat bersemangat dan konsisten. Cerita mengenai Stallman ini bisa menjadi buku tersendiri. Dia sudah beberapa kali datang ke Indonesia dan menceritakan mengenai filosofi dari Free Software.

“GNU” merupakan singkatan dari “GNU” lagi. Rekursif

Di dalam dunia *science* ada “*definitional circle*,” yaitu definisi yang rekursif. Contoh dari *definitional circle* adalah rumus Newton $f = ma$. Poincare, salah seorang pakar, berpandangan bahwa gaya f dan massa m didefinisikan secara sirkular dengan menggunakan term pasangannya. Jadi f didefinisikan dengan menggunakan m , sementara m didefinisikan dengan menggunakan f . Mbingungkan bukan?

Latihan.

Cari nama software atau program yang juga rekursif.

Sudah menemukan nama rekursif untuk produk anda?

Ada humor plesetan dari kawan-kawan tentang moto kota Bandung, yaitu “Bandung Bermartabat.” Kata “bermartabat” merupakan singkatan dari “Bersih, Makmur, Taat, dan Bermartabat”. Benarkah demikian? Tentu saja tidak. Seharusnya kepanjangannya adalah “Bersih, Makmur, Taat, dan Bersahabat.” Kalau kata “bermartabat” juga rekursif, nampaknya Pemda Bandung juga berisi hacker!

3.2 Silicon Art

Seni dan komputer tidak hanya terjadi di sisi software saja, akan tetapi juga terjadi di sisi hardware. Desain dari rangkaian terpadu (*Integrated Circuit Design*) merupakan bidang desain perangkat keras yang secara spesifik memiliki perlindungan HaKI. Hal ini menunjukkan bahwa desain IC memiliki kandungan desain yang sangat tinggi.

Dalam desain IC, desainer meletakkan komponen dan menyusun jalur-jalur yang menghubungkan komponen tersebut. Kedua hal tersebut dapat dilakukan dengan menggunakan program atau secara manual. Untuk rangkaian dengan jumlah komponen yang banyak dan memiliki struktur yang teratur, seperti RAM, software banyak membantu. Akan tetapi ada hal-hal tertentu yang lebih bagus jika dilakukan oleh manusia. Kesempatan ini digunakan oleh desainer untuk membuat *silicon art*.

Bab 4

Ilmu Pemrograman

Pada bab ini saya akan lebih serius dengan membahas mengenai ilmu dari pemrograman. Kata “serius” perlu mendapat pertanyaan karena belajar pemrograman tidak harus selalu serius dalam artian membosankan. Kita bisa belajar pemrograman dari games, misalnya. Berikut ini ada beberapa contoh pelajaran pemrograman yang mengambil pendekatan dengan menggunakan games:

- Game Maker: <http://www.gamemaker.nl/>
- Corewars
- Ant wars: <http://ant-wars.com/>
- ICFP contest: <http://www.cis.upenn.edu/proj/plclub/contest/index.php>

Bibliografi

- [Knu98] Donald Knuth. *The Art of Computer Programming*. Addison-Wesley, 1998.
- [Lam86] Susan Lammers. *Programmers at Work*. Tempus Books of Microsoft Press, 1986.