DEVELOPING AN ALGORITHM

Pertemuan 3

Objectives

- To introduce methods of analysing a problem and developing a solution
- To develop simple algorithms using the sequence control structure
- To introduce methods of manually checking the developed solution

Outline

3.1 Defining the problem
3.2 Designing a solution algorithm
3.3 Checking the solution algorithm

Defining the problem

Langkah pertama, dan salah satu yang paling penting, adalah mendefinisikan masalah. Ini melibatkan membaca dan membaca ulang masalah dengan cermat sampai Anda benar-benar memahami apa yang diperlukan. Cukup sering, informasi tambahan perlu dicari untuk membantu menyelesaikan setiap ambiguitas atau kekurangan dalam spesifikasi masalah. Untuk membantu analisis awal ini, masalah harus dibagi menjadi tiga komponen terpisah:

1. Input: daftar sumber data yang disediakan untuk masalah.

2. Output: daftar output yang dibutuhkan.

3. Pemrosesan: daftar tindakan yang diperlukan untuk menghasilkan keluaran yang diperlukan.

Saat membaca rumusan masalah, komponen input dan output mudah diidentifikasi, karena menggunakan kata-kata deskriptif seperti kata benda dan kata sifat. Komponen pemrosesan juga diidentifikasi dengan mudah. Pernyataan masalah biasanya menggambarkan langkah-langkah pemrosesan sebagai tindakan, menggunakan kata kerja dan kata keterangan.

Saat membagi masalah menjadi tiga komponen yang berbeda, analisis kata-kata aktual yang digunakan dalam spesifikasi, dan bagi menjadi kata-kata yang deskriptif dan yang menyiratkan tindakan. Mungkin membantu untuk menggarisbawahi kata benda, kata sifat dan kata kerja yang digunakan dalam spesifikasi.

Dalam beberapa masalah pemrograman, input, proses, dan output mungkin tidak didefinisikan dengan jelas. Dalam kasus seperti itu, yang terbaik adalah berkonsentrasi pada output yang diperlukan. Melakukan hal ini kemudian akan menentukan input, dan cara akan ditetapkan untuk menentukan langkah-langkah pemrosesan yang diperlukan untuk menghasilkan output yang diinginkan. Pada tahap ini, bagian pemrosesan harus menjadi daftar tindakan apa yang perlu dilakukan, bukan bagaimana tindakan itu akan diselesaikan.

Jangan mencoba untuk menemukan solusi sebelum masalah sepenuhnya didefinisikan.

Mari kita lihat contoh sederhana. Sebuah program diperlukan untuk membaca tiga angka, menjumlahkannya dan mencetak totalnya.

Dengan melihat kata benda dan kata sifat yang digarisbawahi, mudah untuk melihat bahwa input untuk masalah ini adalah tiga angka dan outputnya adalah total. Akan sangat membantu untuk menuliskan dua komponen pertama ini dalam diagram sederhana, yang disebut diagram pendefinisian.

Input	Processing	Output
Angka1 Angka2 Angka3		total

Kedua, garis bawahi kata kerja dan kata keterangan yang digunakan dalam spesifikasi. Ini untuk menetapkan tindakan yang diperlukan.

Contoh selanjutnya sekarang akan terlihat seperti ini:

Sebuah program diperlukan untuk membaca tiga angka, menjumlahkannya dan mencetak totalnya.

Dengan melihat kata-kata yang digarisbawahi, dapat dilihat bahwa kata kerja pemrosesan adalah 'baca', 'jumlah' dan 'cetak'. Langkah-langkah ini sekarang dapat ditambahkan ke diagram pendefinisian kita untuk membuatnya lengkap. Saat menuliskan setiap kata kerja pemrosesan, sertakan juga objek atau kata benda yang terkait dengan setiap kata kerja. Diagram yang mendefinisikan sekarang menjadi:

Input	Processing	Output
Angka1 Angka2 Angka3	**Baca** 3 Angka three numbers **Jumlahkan** ketiga angka tersebut **Cetak** angka total	total

Input untuk masalah, **Output** yang akan dihasilkan, dan langkah-langkah **Processing** yang diperlukan untuk mengubah input menjadi output.

Ketika menulis langkah-langkah pemrosesan dalam suatu algoritma, gunakan kata-kata yang menggambarkan pekerjaan yang harus dilakukan dalam hal tugas atau fungsi tertentu. Sebagai contoh:

Read three numbers
add numbers together
Print total number

Ada pola dalam kata-kata yang dipilih untuk menggambarkan langkah-langkah ini. Setiap tindakan digambarkan sebagai kata kerja tunggal diikuti oleh objek dua kata. Penelitian telah menunjukkan bahwa jika Anda mengikuti konvensi ini untuk menggambarkan langkah pemrosesan, dua manfaat akan dihasilkan. Pertama, Anda menggunakan pendekatan disiplin untuk mendefinisikan masalah dan, kedua, pemrosesan dibagi menjadi tugas atau fungsi yang terpisah.

Contoh:

Sebuah program diperlukan untuk meminta operator terminal untuk pembacaan suhu maksimum dan minimum pada hari tertentu, menerima pembacaan tersebut sebagai bilangan bulat, dan menghitung dan menampilkan ke layar suhu rata-rata, dihitung dengan (suhu maksimum + suhu minimum)/2.

Pertama, menetapkan komponen input dan output dengan menggarisbawahi kata benda dan kata sifat dalam pernyataan masalah. Sebuah program diperlukan untuk meminta operator terminal untuk maksimum dan minimum suhu pada hari tertentu, terima pembacaan tersebut sebagai bilangan bulat, dan hitung serta tampilkan ke layar suhu rata-rata, dihitung dengan (suhu maksimum + suhu minimum)/2.

Komponen input adalah pembacaan suhu maksimum dan minimum, dan output adalah suhu rata-rata.

Menggunakan nama yang bermakna, komponen algoritma ini dapat diatur dalam diagram pendefinisian sebagai berikut:

Input	Processing	Output
max_temp min_temp		avg_temp

Sekarang buat langkah-langkah pemrosesan dengan menggarisbawahi kata kerja dalam pernyataan masalah.

Sebuah program diperlukan untuk meminta operator terminal untuk pembacaan suhu maksimum dan minimum pada hari tertentu, menerima pembacaan tersebut sebagai bilangan bulat, dan menghitung dan menampilkan ke layar suhu rata-rata, dihitung dengan (suhu maksimum + suhu minimum)/2.

Kata kerja pemrosesan adalah 'prompt', get', 'calculate' dan 'display'. Dengan menemukan objek terkait dari kata kerja ini, diagram pendefinisian sekarang dapat diselesaikan, sebagai berikut:

Input	Processing	Output
max_temp min_temp	Prompt suhu Get suhu Calculate average suhu Display average suhu	avg_temp

Designing a solution algorithm

Merancang algoritma solusi adalah tugas yang paling menantang dalam siklus hidup suatu program. Setelah masalah didefinisikan dengan benar, Anda dapat mulai menguraikan solusi. Upaya pertama dalam merancang algoritma solusi biasanya tidak menghasilkan produk jadi. Langkah-langkah solusi mungkin nantinya dapat diubah atau dihapus. Pseudocode berguna dalam proses coba-coba ini, karena relatif mudah untuk menambah, menghapus, atau mengubah instruksi. Jangan ragu untuk mengubah algoritma, atau bahkan membuangnya dan memulai lagi, jika Anda tidak sepenuhnya puas dengannya atau jika algoritma tidak benar, program tidak akan pernah benar.

Ada beberapa argumen bahwa pekerjaan seorang programmer berakhir dengan desain algoritma. Setelah itu, seorang pembuat kode atau programmer dapat mengkodekan algoritma solusi ke dalam bahasa pemrograman tertentu. Dalam praktiknya, ini biasanya tidak terjadi. Namun, penting bahwa Anda tidak memulai pengkodean sampai langkah-langkah yang diperlukan untuk mendefinisikan masalah dan rancangan algoritma solusi selesai.

Berikut adalah algoritma solusi untuk tiga contoh sebelumnya. Semua melibatkan struktur kontrol urutan saja; tidak ada keputusan atau loop, sehingga algoritma solusinya relatif sederhana. Contoh, Sebuah program diperlukan untuk membaca tiga angka, menjumlahkannya dan mencetak totalnya.

A Defining diagram

Input	Processing	Output
Angka1 Angka2 Angka3	**Read** 3 Angka three numbers Jumlahkan ketiga angka tersebut **Print** angka total	total

B Solution algorithm

Diagram yang menentukan menunjukkan apa yang diperlukan, dan perhitungan sederhana akan menentukan caranya. Menggunakan pseudocode dan struktur kontrol urutan membangun algoritma solusi sebagai berikut:

```
Add_three_numbers
   Read Angka1, Angka2, Angka3
   total = Angka1+ Angka2+ Angka3
   Print total
END
```


Ada beberapa hal yang perlu dipertimbangkan dalam algoritma solusi ini:

1 Sebuah nama telah diberikan untuk algoritma, yaitu Add_three_numbers.

Nama algoritma harus menjelaskan secara singkat fungsi dari algoritma,dan biasanya dinyatakan sebagai kata kerja tunggal diikuti oleh objek dua kata.

2 Pernyataan END di akhir algoritma menunjukkan bahwa algoritma selesai.

3 Semua langkah pemrosesan antara nama algoritme dan pernyataan END telah diindentasi agar mudah dibaca.

4 Setiap langkah pemrosesan dalam diagram pendefinisian berhubungan langsung dengan satu atau lebih pernyataan dalam algoritma. Misalnya, ‘baca tiga angka' dalam diagram pendefinisian menjadi Read angka1, angka2, angka3' dalam algoritma; dan 'Menjumlahkan angka' menjadi 'total = angka1 + angka2 + angka3'.

Sekarang setelah algoritma selesai, desk checking solusi dan kemudian menerjemahkannya ke dalam Bahasa pemrograman

Checking the solution algorithm

Setelah algoritma solusi telah ditetapkan, itu harus diuji kebenarannya. Langkah ini diperlukan karena sebagian besar kesalahan logika utama terjadi selama pengembangan algoritma, dan jika tidak terdeteksi, kesalahan ini dapat diteruskan ke program.

Jauh lebih mudah untuk mendeteksi kesalahan dalam pseudocode daripada dalam kode program yang sebenarnya. Ini karena begitu pemrograman dimulai, biasanya diasumsikan bahwa logika algoritmanya benar. Kemudian, ketika kesalahan terdeteksi, perhatian Anda terfokus pada baris kode individu untuk mengidentifikasi masalah, bukan pada logika yang dinyatakan dalam algoritma. Seringkali terlalu sulit untuk mundur dan menganalisis program secara keseluruhan. Akibatnya, banyak jam yang membuat frustrasi dana terbuang sia-sia selama pengujian, yang sebenarnya hal ini dapat dihindari dengan menghabiskan beberapa menit saja dengan desk checking algoritma solusi.

Desk checking melibatkan penelusuran melalui logika algoritma dengan beberapa data uji yang dipilih. Artinya, ‘bekerja' melalui logika algoritma persis seperti yang dilakukan komputer, melacak semua nilai variabel utama pada selembar kertas. 'Memainkan logika seperti komputer' ini tidak hanya membantu mendeteksi kesalahan sejak dini, tetapi juga membantu Anda menjadi terbiasa dengan cara program berjalan. Semakin dekat Anda dengan eksekusi program, semakin mudah untuk mendeteksi kesalahan.

Selecting test data

Saat memilih data uji untuk memeriksa algoritma, lihat spesifikasi program dan pilih kasus uji sederhana yang didasarkan pada persyaratan spesifikasi, bukan algoritma. Dengan melakukan ini, Anda masih dapat berkonsentrasi pada apa yang seharusnya dilakukan oleh program. Untuk memeriksa algoritma, Anda hanya memerlukan beberapa kasus uji sederhana yang akan mengikuti jalur utama logika algoritma. Tes yang jauh lebih komprehensif akan dilakukan setelah algoritma dikodekan ke dalam bahasa pemrograman.

Steps in desk checking an algorithm

Ada enam langkah sederhana yang harus diikuti saat melakukan desk checking suatu algoritma:

1 Pilih kasus uji input sederhana yang valid. Dua atau tiga kasus uji biasanya cukup.

2 Tetapkan hasil sederhana yang diharapkan untuk setiap kasus uji. Ini adalah salah satu alasan untuk memilih data uji sederhana: jauh lebih mudah untuk menentukan total 10, 20 dan 30 daripada 3,75, 2,89 dan 5,31!

3 Buatlah tabel pada selembar kertas dari nama variabel yang relevan dalam algoritma.

4 Jalankan kasus uji pertama melalui algoritma, baris demi baris, simpan catatan langkah demi langkah dari isi setiap variabel dalam tabel saat data melewati logika.

5 Ulangi proses *walk-through* menggunakan kasus data uji lainnya, hingga algoritma telah mencapai akhir logisnya.

6 Periksa apakah hasil yang diharapkan yang ditetapkan pada Langkah 2 cocok dengan hasil aktual yang dikembangkan pada Langkah 5.

Dengan memeriksa algoritma di meja, Anda mencoba mendeteksi kesalahan lebih awal. Ini adalah ide yang baik untuk orang lain selain penulis algoritma solusi untuk merancang data uji untuk program, karena mereka tidak dipengaruhi oleh logika program.

Desk checking akan menghilangkan sebagian besar kesalahan, tetapi masih tidak dapat membuktikan bahwa algoritma tersebut 100% benar!. Contoh

A Solution algorithm

```
   Add_three_numbers
1      Read Angka1, Angka2, Angka3
2      total = Angka1+ Angka2+ Angka3
3      Print total
   END
```


B Desk checking

1. Pilih dua set data uji masukan. Tiga angka yang dipilih adalah 10, 20 dan 30 untuk kasus uji pertama dan 40, 41 dan 42 untuk yang kedua.

	First data set	Second data set
Angka1	10	40
Angka2	20	41
Angka3	30	42

1. Tetapkan hasil yang diharapkan untuk setiap kasus uji

	First data set	Second data set
total	60	123

3. Siapkan tabel nama variabel yang relevan, dan lewati setiap kumpulan data uji melalui algoritme solusi, pernyataan demi pernyataan. Nomor baris telah digunakan untuk mengidentifikasi setiap pernyataan dalam program.

Statement number	Angka 1	Angka 2	Angka3	total
First pass				
1	10	20	30	
2				60
3				print
Second pass				
1	40	41	42	
2				123
3				print

Periksa apakah hasil yang diharapkan (60 dan 123) cocok dengan hasil sebenarnya (total kolom dalam tabel)

Desk checking ini, yang seharusnya memakan waktu tidak lebih dari beberapa menit, Anda sekarang dapat melanjutkan untuk mengkodekan algoritma ke dalam bahasa pemrograman. Perhatikan bahwa jika, pada akhir desk checking, hasil aktual tidak sesuai dengan hasil yang diharapkan, algoritma mungkin berisi kesalahan logika. Dalam hal ini, perlu periksa kembali ke algoritma nya, lalu perbaiki kesalahan.

Simpulan

- Mendefinisikan masalah dengan membuat *defining diagram*
- membuat algoritma solusi menggunakan pseudocode
- desk check algoritma menggunakan dua kasus uji yang valid

Soal Latihan

1. Untuk membantu analisis awal, masalah harus dibagi menjadi … komponen terpisah

a. Satu
b. Dua
c. Tiga
d. Empat
e. Lima

2. Tugas yang paling menantang dalam siklus hidup suatu program adalah …

a. Definisi masalah
b. Merancang algoritma solusi
c. Pengujian algoritma
d. Memilih data uji
e. Mengetes algoritma

3. Pilih kasus uji input sederhana yang valid. Dua atau tiga kasus uji biasanya cukup, merupakan Langkah ke berapa saat melakukan desk checking suatu algoritma :

a. Satu
b. Dua
c. Tiga
d. Empat
e. Lima

4. Langkah-langkah **Processing** yang diperlukan untuk mengubah input menjadi …

a. Output
b. Proses
c. Algoritma
d. Masukan
e. Pseudocode

Soal Latihan

5. Upaya pertama dalam merancang algoritma solusi biasanya tidak menghasilkan …

a. Produk jadi
b. Produk solusi
c. Produk setengah jadi
d. produk pilihan
e. produk langsung