Chapter 5 : Desain Pemrograman

Oleh : Fiftin Noviyanto

A. What Is Top-Down Design?

- Top-down design bekerja dengan beberapa cara. Pertama kita memiliki satu pekerjaan yang besar, selanjutnya pekerjaan tersebut kita pecah menjadi beberapa bagian
- Analogi : Google Earth, untuk mencari kampus UAD
- Pada kasus membuat program biasanya kita dihadapkan pada masalah yang umum. Namun dari kasus tersebut dapat dibagi menjadi beberapa bagian kecil sehingga memudahkan kita untuk menyelesaikannya.
- Kita harus memulainya dengan membuat kerangka permasalahan. Selanjutnya membagi dalam beberapa bagian proses.

- Top-Down Desain<> Buttom Up Desain
- Bottom-up design adalah desain dengan memfokuskan pada permasalahan-permasalahan kecil sebelum melihat permasalahan yang lebih besar.
- Top-down design memungkinkan programmer menyelesaikan masalah dimulai dengan perencanaan global kemudian mengisi detailnya pada setiap level sampai lengkap

B. Real-World Problems and Design

Contoh 1 :

- Andi mengoleksi Video VHS selama beberapa tahun. Dan saat ini ingin dipindah ke format DVD. Dia telah membeli beberapa film dalam format DVD dan telah diduplikasikan ke dalam format VHS. I decide to take inventory and get rid of the old duplicate VHS movies. How should I approach this problem?

Urutan Desain Penyelesaian

1. Make a list of all movies. Get the title of each movie and then alphabetize those movies.
2. Next to each movie write DVD, VHS, or Both.
3. Separate the movies with the tag Both from those without that tag.
4. Throw out the videos of the Both collection. Put them into boxes and deliver to a Goodwill location. Change all words Both to DVD.
5. Alphabetize the entire collection.

- Notice how steps 1 and 4 became more detailed in the second list. We refined those steps by giving them more detail. This process is known as stepwise refinement. Stepwise refinement is useful because it helps the programmer outline the code he must write.
- In step 1, I could write a separate function called getTitle and another called alphabetizeList. When I set out to write this code, I will have a clear idea of what I need to accomplish. If I don't refine the steps, I have less of an idea of what I need to do, since step 1 initially only says, "Make a list of all movies."
- Stepwise refinement is important because it trains the programmer to move step by step, the way a computer works. Every time you refine a task by making it more specific, you are getting closer to being able to write these instructions in a programming language. As a beginning programmer, you need to learn how to organize solutions in a systematic way so that they can be processed. If your tasks are too vague, then chances are you will have a tough time writing them in a language.

Contoh Kasus (1)

- Here is another example of taking a large task and breaking it into smaller tasks using a top-down approach. Consider what you would do if you were buying a car. Most of us would not just go to the store and buy it without doing a few separate tasks. Each major task must be broken into subtasks that give more
- Buying a car is really a set of other smaller tasks, like finding out how much money you can spend and making choices about the car. You would need to choose a car's make, its type, and its color before deciding whether the car was the one you wanted to buy. You also have to get the money together for the car. Are you going to need a bank check written instead of a personal check? Most likely!

Table 5.1 Buying a Car

Task	Steps to Complete Task (Subtask)
How much money do I have?	Check first bank account. Check second bank account. Add all accounts
Choose a car.	Select make. Select type (sedan, coup, SUV, etc.) Select color.
Pay $$.	Get bank check ready. Bring check to dealer.

- As you can see, the tasks on the left have been broken down on the right to make each task more clearly defined.

C. Computer Problems and Design

- Now we will examine some practical examples that you can program. Each example involves a main task that requires some refinement. As you read each example, try to identify which task will be refined and how it will be refined.

Contoh Kasus 1

- In this example, we will consider a math problem involving prime numbers. We need to make a list of all the primes less than or equal to the number 500. This sounds like a big task, since there are 500 numbers to check, and we have to keep a list of all those that are prime.
- Let's start by looking at the primes less than 10. There are four primes less than the number 10 (1 is not considered prime)—2, 3, 5, and 7. Now we have to find out which numbers in the number set 1, 2, . . . , 499, 500 are prime. Let's look at this algorithm:

1. Start a number counter called myNum at 2, since 2 is the first number we will check.
2. Check to see whether myNum is prime.
3. If myNum is prime, write it down.
4. Increase myNum by 1 so that you can check the next number.
5. Go back to step 2 until you hit 501.

- Most of these steps are clear, but we need to elaborate on how we would determine whether a number is prime. So step 2 needs to be refined. In this step we would have to articulate how a prime number is recognized. We would have to think about what we could write in programming terms that would allow the computer to recognize a prime number. In fact, it is not that difficult a task, but it is good that we have drawn attention to it by separating that work from the other steps in the algorithm.

Contoh Kasus 2

- In this example, we will find the youngest person in a group of 50 people. At first, this problem can seem difficult to program. However, I have developed a list of steps that you can follow to complete this task. Again, as with the previous example, consider what step needs to be clarified.

1. Find the first person's birth date.
2. Find the next person's birth date. Retain the more recent birth date.
3. Go back to step 2 if there are more people.
4. Stop when there are no more people to check.

- The step that needs the most refinement is step 2, where we have to retain the more recent birth date. That would involve some subtasks of comparing years of birth, and if those were the same, we would compare months and then days, if necessary.

- Find the first person's birth date. Get the year, the month, and the day.
- Find the next person's birth date. Get the year, the month, and the day.
  - Retain the more recent birth date. Compare years, then months, if necessary, and days, if necessary.
- Go back to step 2 if there are more people. Check the list of people.
- Stop when there are no more people to check.

Both of these are examples of top-down design. As you continue to learn about programming and encounter more elaborate programs, you will find this perspective useful. The process of developing separate blocks of codes, to accomplish separate tasks called modules or functions

Summary

- Each time you break a big problem into other smaller problems, you are using a process of top-down design, which we also call stepwise refinement. Each set of tasks is refined by adding more steps to indicate how the big task is accomplished. By trying to break your tasks into other smaller tasks, you are starting to develop a modular approach to a program.