Method & Tools for Program Analysis & Design

TMB208 – Pemrograman Teknik
Kredit: 3 (2-3)

Proses Pengembangan Program

Activities

Analisis ➔ Desain ➔ Pemrograman ➔ Uji Coba

Diagram Terstruktur (Structured Chart), Mathematical Model, Pseudocode

Programming Language, Editor, Kompiler, Debugger

Debugger, Linker Program Executor

Tools (alat Bantu)

Programming Logic and Design, Introductory, Fourth Edition
Programming Methods

• Based on structures of programming logic
• Top Down Method
• Bottom-Up Method
• Used structured charts to describe the program logic

Structured Programming

• **Structured programming**: A technique for organizing and coding computer programs in which a hierarchy of modules is used, each having a single entry and a single exit point, and in which control is passed downward through the structure without unconditional branches to higher levels of the structure. Three types of control flow are used: sequential, test, and iteration.

• **Structure**: is a basic unit of programming logic
Top Down Design

• Top Down Design is a method of designing something that starts with the complete item then breaks it down into smaller and smaller components. In programming it means breaking a difficult task down (divide and conquer) and solving pieces independently until every step can easily be implemented (successive refinement).

Advantages of Top-Down Design

• Breaking the problem into parts helps us to clarify what needs to be done.
• At each step of refinement, the new parts become less complicated and, therefore, easier to figure out.
• Parts of the solution may turn out to be reusable.
• Breaking the problem into parts allows more than one person to work on the solution.
Structured Programs

• We will use top-down design for all remaining programming projects.
• This is the standard way of writing programs.
• Programs produced using this method and using only the three kinds of control structures, sequential, selection and repetition, are called **structured programs**.
• Structured programs are easier to test, modify, and are also easier for other programmers to understand.

Another Example

• **Problem:** Write a program that draws this picture of a house.
The Top Level

- Draw the outline of the house
- Draw the chimney
- Draw the door
- Draw the windows

Pseudocode

- A natural language for specific purpose with limited & specific grammar & vocabulary
Pseudocode for Main

Call Draw Outline
Call Draw Chimney
Call Draw Door
Call Draw Windows

Observation

• The door has both a frame and knob. We could break this into two steps.
Pseudocode for Draw Door

Call Draw Door Frame
Call Draw Knob

Another Observation

• There are three windows to be drawn.
One Last Observation

• But don’t the windows look the same? They just have different locations.

• So, we can reuse the code that draws a window.
  – Simply copy the code three times and edit it to place the window in the correct location, or
  – Use the code three times, “sending it” the correct location each time (we will see how to do this later).

• This is an example of code reuse.

Reusing the Window Code
Pseudocode for Draw Windows

Call Draw a Window, sending in Location 1
Call Draw a Window, sending in Location 2
Call Draw a Window, sending in Location 3

Another Level?

- Should any of these steps be broken down further? Possibly.
- How do I know? Ask yourself whether or not you could easily write the algorithm for the step. If not, break it down again.
- When you are comfortable with the breakdown, write the pseudocode for each of the steps (modules) in the hierarchy.
- Typically, each module will be coded as a separate function.
Bottom-up

- Produces smaller and more flexible programs
- Promotes code re-use
- Permits assessment of sub-modules
- Best suited when problem is ill-defined or missing

Tools for Program Analysis & Design

- **Structured Charts**: using graphical diagrams/charts to express programming logic
- **Pseudocode**: using narrative symbols to express programming logic
A Process View of An Algorithm

- A process is depicted as a circle and has a name
- A process is representing an algorithm that transforms a set of inputs into a set of outputs

Understanding the Three Basic Structures of Programming Logic

- Any program can be constructed from only three basic types of structures
  1. Sequence
  2. Selection
  3. Loop
Understanding the Three Basic Structures (continued)

- **Sequence structure**
  - A set of instructions, performed sequentially with no branching

![Sequence Structure Diagram]

Calculate the area below

![Flowchart Diagram]

Write the program code
Understanding the Three Basic Structures (continued)

• **Selection structure**
  
  – Asks a question, then takes one of two possible courses of action based on the answer
  – Also called a **decision structure** or an **if-then-else**

![Selection Structure Diagram]

Understanding the Three Basic Structures (continued)

• **Dual-alternative if**: contains two alternatives

    ```
    if hoursWorked is more than 40 then
        calculate regularPay and overtimePay
    else
        calculate regularPay
    ```
Understanding the Three Basic Structures (continued)

- Single-alternative if: contains one alternative

![Figure 2-5: Single-alternative decision structure]

Understanding the Three Basic Structures (continued)

- Single-alternative if

```plaintext
if employee belongs to dentalPlan then
deduct $40 from employeeGrossPay
```

- Else clause is not required
- Null case: situation where nothing is done
Understanding the Three Basic Structures (continued)

- **Loop structure**
  - Repeats a set of actions based on the answer to a question
  - Also called *repetition* or *iteration*
  - Question is asked first in the most common form of loop

```
while testCondition continues to be true
do someProcess

while you continue to beHungry
take anotherBiteOfFood
```
Understanding the Three Basic Structures (continued)

- All logic problems can be solved using only these three structures
- Structures can be combined in an infinite number of ways
- **Stacking**: attaching structures end-to-end
- End-structure statements
  - Indicate the end of a structure
  - **endif**: ends an if-then-else structure
  - **endwhile**: ends a loop structure

---

**Figure 2-7: Structured Flowchart and Pseudocode**

```
do stepA
  do stepB
  if conditionC is true then
    do stepD
  else
    do stepE
  endif
  while conditionF is true
  do stepG
endwhile
```
Understanding the Three Basic Structures (continued)

- Any individual task or step in a structure can be replaced by a structure
- **Nesting**: placing one structure within another
- Indent the nested structure’s statements
- **Block**: group of statements that execute as a single unit

![Figure 2-8: Flowchart and Pseudocode showing a sequence nested within a selection](image)

```plaintext
if conditionA is true then
  do stepE
else
  do stepB
  do stepC
  do stepD
endif
```
Understanding the Three Basic Structures (continued)

**FIGURE 2-9: SELECTION IN A SEQUENCE WITHIN A SELECTION**

```
if conditionA is true then
  do stepE
else
  do stepG
  if conditionF is true then
    do stepI
  else
    do stepG
  endif
  do step0
endif
```

**FIGURE 2-10: FLOWCHART AND PSEUDOCODE FOR LOOP WITHIN SELECTION WITHIN SEQUENCE WITHIN SELECTION**

```
if conditionA is true then
  do stepE
else
  do stepG
  if conditionF is true then
    do stepI
    endwhile
  else
    do stepG
  endif
  do step0
endif
```
Understanding the Three Basic Structures (continued)

- Each structure has one entry and one exit point
- Structures attach to others only at entry or exit points

**Flowchart Symbols**

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Use in Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oval</td>
<td>![Oval Symbol]</td>
<td>Denotes the beginning or end of the program</td>
</tr>
<tr>
<td>Parallelogram</td>
<td>![Parallelogram Symbol]</td>
<td>Denotes an input operation</td>
</tr>
<tr>
<td>Rectangle</td>
<td>![Rectangle Symbol]</td>
<td>Denotes a process to be carried out e.g. addition, subtraction, division etc.</td>
</tr>
<tr>
<td>Diamond</td>
<td>![Diamond Symbol]</td>
<td>Denotes a decision (or branch) to be made. The program should continue along one of two routes. (e.g. IF/THEN/ELSE)</td>
</tr>
<tr>
<td>Hybrid</td>
<td>![Hybrid Symbol]</td>
<td>Denotes an output operation</td>
</tr>
<tr>
<td>Flow line</td>
<td>![Flow line Symbol]</td>
<td>Denotes the direction of logic flow in the program</td>
</tr>
</tbody>
</table>
Example

Step 1: Input M1, M2, M3, M4
Step 2: GRADE ← (M1 + M2 + M3 + M4) / 4
Step 3: if (GRADE < 50) then
    Print “FAIL”
else
    Print “PASS”
endif

Understanding the Reasons for Structure

• Advantages of structure:
  – Provides clarity
  – Professionalism
  – Efficiency
  – Ease of maintenance
  – Supports modularity
Understanding the Reasons for Structure (continued)

![Flowchart and Pseudocode of Structured College Admission Program](image)

```plaintext
start
    read testScore, classRank
    if testScore >= 90 then
        if classRank >= 25 then
            print "Accept"
        else
            print "Reject"
        endif
    else
        if testScore >= 80 then
            if classRank >= 50 then
                print "Accept"
            else
                print "Reject"
            endif
        else
            if testScore >= 70 then
                if classRank >= 75 then
                    print "Accept"
                else
                    print "Reject"
                endif
            else
                print "Reject"
            endif
        endif
    endif
stop
```

Understanding the Reasons for Structure (continued)

![Pseudocode for the Rock, Paper, Scissors Game](image)
Recognizing Structure

• Any set of instructions can be expressed in structured format
• Is this flowchart structured?

![Flowchart Example 2](image1)

Recognizing Structure (continued)

• Is this flowchart structured?

![Flowchart Example 3](image2)
Recognizing Structure (continued)

- To make it structured, pull each symbol out and rework

```
Figure 2-24: Untangling Example 3, First Step

A
```

- B begins a selection structure

```
Figure 2-25: Untangling Example 3, Second Step

A
B?
```

Recognizing Structure (continued)

- Pull up on the flowline from the left side of B

```
Figure 2-26: Untangling Example 3, Third Step

A

No

End

C
```
Recognizing Structure (continued)

• Next, pull up the flowline on the right side of B

![Flowchart](image1)

Recognizing Structure (continued)

• Pull up the flowline on the left side of D and untangle it from the B selection by repeating C

![Flowchart](image2)
Recognizing Structure (continued)

• Now pull up the flowline on the right side of D

\[\text{Figure 2.29: Untangling Example 3, Sixth Step}\]

Recognizing Structure (continued)

• Bring together the loose ends of D and of B

\[\text{Figure 2.30: Finished Flowchart and Pseudocode for Untangling Example 3}\]
Nested

• Using nested **if-then-else** for multiple alternatives

Three Special Structures – **Case**, **Do While**, and **Do Until**

• Many languages allow three additional structures:
  - **case** structure
  - **do-while** structure
  - **do-until** structure

• **Case** Structure:
  - Decisions with more than two alternatives
  - Tests a variable against a series of values and takes action based on a match
  - Nested **if-then-else** statements will do what a **case** structure does
Three Special Structures – Case

• Using a case structure for multiple alternatives

![Flowchart and PseudoCode of Case Structure](image)

Three Special Structures – Do While, and Do Until

• do-while and do-until loops
  – Question is asked at the end of the loop structure
  – Ensures that the loop statements are always used at least once

![Structure of a Do-While or Do-Until (Posttest) Loop](image)
Three Special Structures – **Do While**, and **Do Until**

- **do-while** loop executes as long as the question’s answer is Yes or True
- **do-until** loop executes as long as the question’s answer is No or False (until it becomes Yes or True)

```plaintext
do
  pay bills
while more bills remain to be paid
```

```plaintext
do
  pay bills
until all bills are paid
```

Three Special Structures – **While** (Pretest & Posttest Loop)

- **while** loop with question at beginning is called a **pretest loop**
- **while** loop with question at end are called **posttest loops**
Pretest Loop Using **While**

**FIGURE 2-36**: FLOWCHART AND PSEUDOCODE FOR SEQUENCE FOLLOWED BY WHILE LOOP

Posttest Loop Using **While**

**FIGURE 2-35**: FLOWCHART AND PSEUDOCODE FOR DO-WHILE LOOP