RESIST THE URGE TO CODE

AAA

CCC Monday 1/11/2016

I 1. Clearly define what the program is to do.
2. Visualize the program running on the computer. I

II 3. Use design tools such as a hierarchy chart, flowcharts, or pseudocode to create a model of the program.
4. Check the model for logical errors. II

III 5. Type the code, save it, and compile it. III

IV 6. Correct any errors found during compilation. Repeat Steps 5 and 6 as many times as necessary.
7. Run the program with test data for input.
8. Correct any errors found while running the program. Repeat Steps 5 through 8 as many times as necessary.

V 9. Validate the results of the program.

page 19

Figure 1.9

I Understand it – focus on WHAT ① Define ② Visualize

II PLAN it – design the HOW ③ pseudocode ④ check design model

III Code it – C++ – ⑤

IV Debug it – ⑥, ⑦, ⑧

BBB

I Understand it – Venkman WHAT

I 1. Clearly define what the program is to do.
2. Visualize the program running on the computer.

II PLAN IT – develop a plan (algorithm) of HOW to solve it.

II 3. Use design tools such as a hierarchy chart, flowcharts, or pseudocode to create a model of the program.
4. Check the model for logical errors.

Dr Raymond Stantz

III Code it < on paper first – then type it into computer –
write it on paper!! Code it on PAPER!
Translate the PLAN into C++ before you type it.

III 5. Type the code, save it, and compile it.

RESIST the URGE to CODE.

IV DEBUG it.

IV 6. Correct any errors found during compilation. Repeat Steps 5 and 6 as many times as necessary.
7. Run the program with test data for input.
8. Correct any errors found while running the program. Repeat Steps 5 through 8 as many times as necessary.

V Check to see if results and output are correct.

V 9. Validate the results of the program.

CCC

I Understand it – Venkman (WHAT)

WHAT is Input? WHAT is Output? (I)

I 1. Clearly define what the program is to do.
2. Visualize the program running on the computer.

algorithm

II PLAN IT – develop a plan of (HOW) to solve it.

HOW to get from I to O (II)

II 3. Use design tools such as a hierarchy chart, flowcharts, or pseudocode to create a model of the program.
4. Check the model for logical errors.

Dr Raymond Stuntz

The steps listed in Figure 1-9 emphasize the importance of planning. Just as there are good ways and bad ways to paint a house, there are good ways and bad ways to create a program. A good program always begins with planning.

With the pay-calculating program as our example, let's look at each of the steps in more detail.

I 1. Clearly define what the program is to do. (I 1) (WHAT) p19

p19 This step requires that you identify the purpose of the program, the information that is to be input, the processing that is to take place, and the desired output. Let's examine each of these requirements for the example program:

Purpose	To calculate the user's gross pay.
I *Input*	Number of hours worked, hourly pay rate.
Process (P)	Multiply number of hours worked by hourly pay rate. The result is the user's gross pay. (P) is for step II, 3. and 4.
O *Output*	Display a message indicating the user's gross pay.

II 2. Visualize the program running on the computer. (I 2)

p19 Before you create a program on the computer, you should first create it in your mind. Step 2 is the visualization of the program. Try to imagine what the computer screen looks like while the program is running. If it helps, draw pictures of the screen, with sample input and output, at various points in the program. For instance, here is the screen produced by the pay-calculating program:

```
How many hours did you work? 10
How much do you get paid per hour? 15
You have earned $150
```

I (WHAT) O

In this step, you must put yourself in the shoes of the user. What messages should the program display? What questions should it ask? By addressing these concerns, you will have already determined most of the program's output.

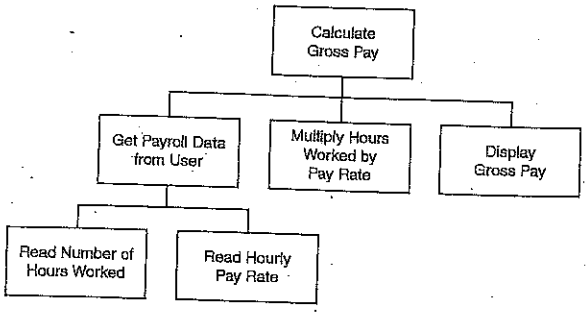
(II) 3. Use design tools such as a hierarchy chart, flowcharts, or pseudocode to create a model of the program.

p20-21

DDD

While planning a program, the programmer uses one or more design tools to create a model of the program. Three common design tools are hierarchy charts, flowcharts, and pseudocode. A *hierarchy chart* is a diagram that graphically depicts the structure of a program. It has boxes that represent each step in the program. The boxes are connected in a way that illustrates their relationship to one another. Figure 1-10 shows a hierarchy chart for the pay-calculating program.

1-10



PLAN IT
(II)
(3) Design tools
Develop HOW
(4) Check designs
logic

A hierarchy chart begins with the overall task, and then refines it into smaller subtasks. Each of the subtasks is then refined into even smaller sets of subtasks, until each is small enough to be easily performed. For instance, in Figure 1-10, the overall task "Calculate Gross Pay" is listed in the top-level box. That task is broken into three subtasks. The first subtask, "Get Payroll Data from User," is broken further into two subtasks. This process of "divide and conquer" is known as *top-down design*.

VideoNote **Introduction to Flowcharting**

A *flowchart* is a diagram that shows the logical flow of a program. It is a useful tool for planning each operation a program performs, and the order in which the operations are to occur. For more information see Appendix D, Introduction to Flowcharting.

Pseudocode is a cross between human language and a programming language. Although the computer can't understand pseudocode, programmers often find it helpful to write an algorithm in a language that's "almost" a programming language, but still very similar to natural language. For example, here is pseudocode that describes the pay-calculating program:

VIdeoNote **Designing a Program with Pseudocode**

```
Get payroll data.
Calculate gross pay.
Display gross pay.
```

Although the pseudocode above gives a broad view of the program, it doesn't reveal all the program's details. A more detailed version of the pseudocode follows.

```
Display "How many hours did you work?".
Input hours.
Display "How much do you get paid per hour?".
Input rate.
Store the value of hours times rate in the pay variable.
Display the value in the pay variable.
```

Notice the pseudocode contains statements that look more like commands than the English statements that describe the algorithm in Section 1.4 (What Is a Program Made of?). The pseudocode even names variables and describes mathematical operations.

(II) 4. Check the model for logical errors.

Logical errors are mistakes that cause the program to produce erroneous results. Once a hierarchy chart, flowchart, or pseudocode model of the program is assembled, it should be checked for these errors. The programmer should trace through the charts or pseudocode, checking the logic of each step. If an error is found, the model can be corrected before the next step is attempted.

algorithms - develop a plan of HOW to solve it.

PLAN IT
Use design tools such as a hierarchy chart, flowcharts, or pseudocode to create a model of the program.
Check the model for logical errors.

(II) 3. 4. (II)

Dr. Raymond Stultz

I 1. Clearly define what the program is to do.
2. Visualize the program running on the computer.

II 3. Use design tools such as a hierarchy chart, flowcharts, or pseudocode to create a model of the program.
4. Check the model for logical errors.

III 5. Type the code, save it, and compile it.

IV 6. Correct any errors found during compilation. Repeat Steps 5 and 6 as many times as necessary.
7. Run the program with test data for input.
8. Correct any errors found while running the program.
Repeat Steps 5 through 8 as many times as necessary.

V 9. Validate the results of the program.

I Understand it.
II PLAN it.
Code it.
III
IV Debug it

EFE

Recall in developing an algorithm or "how" (step 2) to solve the "what" (step 1), it is best to develop several plans as to how and then choose which approach seems best.

Do not just go with the first thing that pops into your head.

I step 1 Understand what the problem is. Absorb its language and become familiar with its terms and its data and output.

Ask: What is given? What is the data?

Ask: What is the goal? What would the output look like?

II step 2 Develop a top-level plan, a step by step algorithm to solve the problem you a fairly deep understanding of from step 1 efforts.

Before going on to step 3, develop another possible approach to solving the problem. Develop at least a 2nd approach. Restate your 1st plan to improve it. Do these two differently stated algorithms suggest a 3rd.

CHOOSE the better algorithm.

Before going on to step 3 with your chosen plan (algorithm), ask:

a. is this plan simple enough to comprehend and work with. ------

b. does this plan capture the whole problem, is this plan adequate, does it solve the problem as stated? --------

Criteria: Is it a simple and an adequate plan?

RESIST THE URGE TO CODE it.

III Step 3: Translate the step 2 plan into the programming language of your choice. Code it in Fortran or Pascal or C or LOGO.

IV Step 4: Test and debug your program.