

Example of Designing a Solution

Contents

1	Overview of Design Procedure	1
2	What the Problem <i>IS</i>	2
3	How to Solve the Problem	2
4	Designing an Algorithm	4
4.1	Step 1: Creating and Initializing my variables	4
4.2	Step 2: Getting <code>numNumbers</code> from the user	4
4.3	Step 3: The Main Loop	5
4.4	Steps 4 & 5: Calculate Average & Display Results	5
4.5	Putting it all together	5
5	Implementing the Algorithm	6
6	Some Comments	8

1 Overview of Design Procedure

- On Monday, we covered how to design a *solution* to a *problem*, in detail (in the **DesignOverview.pdf** document on the course website)
- Today, I'd like to go through an example of this problem solving methodology, specifically regarding the last lab assignment (Lab 3)
- First, an overview of the method we discussed on Friday...
 1. Understand what *exactly* the problem is. Identify any vague or ambiguous parts, and remove them.
 2. Figure out how to solve the problem by yourself, with pencil and paper. Start with an intuitive understanding of how to solve the problem “at 30,000 feet”. Once you have that, go into more detail.
 3. Next, layout an algorithm (again on paper) for solving the problem. Start with a top-down design, and drill-down each part to a most basic *step-by-step* procedure for that specific part. Afterwards, *put it all together*
 4. Finally, implement the algorithm in a programming language (C++ in our case)

-
- The problem statement, as it appears on the Lab 3 site, is...
 - *You are to write a program that does the following...*
 1. Prompt the user for a number (positive integer) that will indicate how many numbers will then be input from the user.
 2. Once the user has input such a number, say *n*, you will then prompt the user for *n* numbers.
 3. After these *n* numbers have been input from the user, you are to display the sum of the *n* numbers input, the maximum and minimum of the *n* numbers, and the average.
-

2 What the Problem *IS*

- In general, *we have to write a program that reads in some ordered list of numbers from the user, calculate the minimum, maximum, and average, and then output those values*
 - **Ambiguity:** What if the user enters 0, or some negative number, for the number of numbers which they are supposed to input?
If they enter 0, should we not do anything?
 - **Solution:** Keep prompting the user until they enter a value greater than 0
-

- With that, I think I have a complete understanding of the problem
 - In *some* detail, the program should
 1. Prompt the user for a number, called `numNumbers`
 2. If `numNumbers` \leq 0, alert the user that they must enter a number \geq 1, and keep re-prompting them
 3. Once we have `numNumbers`, do the following...
 4. Read in `numNumbers` from the user
 5. Once done, output the sum, average, minimum, and maximum of the `numNumbers` input
-

- **NOTE:** At this point, I'm not concerned *at all* about actually solving the problem...
 - I just need to make sure that *I* understand what the problem is
 - Does anyone see anything else?
-

3 How to Solve the Problem

- Can I solve such a problem?
 - If I've had enough sleep or enough coffee, I'll say "yes"
 - To solve such a problem, I could proceed as follows...
-
- Keep prompting the user for `numNumbers` until they enter a correct value
 - *Once I have a correct value, I can proceed onto the "guts" of the program...*
 - I'll loop `numNumbers` times, and each time I'll do the following...
 1. Prompt the user for a number, store it in `currentNumber`
 2. Add `currentNumber` to an ongoing sum, call that variable `sum`
 3. Check to see if `currentNumber` is bigger than what I think is currently the maximum, which I'll store in some other variable (call it `maximum`), and update appropriately
 4. Check to see if `currentNumber` is smaller than what I think is currently the minimum, which I'll also store in some other variable (call it `minimum`), and update appropriately.
 - Then I'll calculate the average, and store it in some variable, call it `average`

- Finally, I'll report the values of `sum`, `average`, `maximum`, and `minimum`
-

- So, I need the following variables...

1. `numNumbers`: Stores how many numbers to read in
 2. `currentNumber`: A holding space for each number I read from the user in the "guts"
 3. `sum`: To store the sum of all the numbers
 4. `average`: To store the average of all the numbers
 5. `maximum/minimum`: To store what I think is the maximum/minimum of the numbers input
-

- Do any of these variables need special initialization?

1. `numNumbers`: No, it's read from the user
 2. `currentNumber`: No, it's read from the user
 3. `sum`: Should initialize this to 0.
 4. `average`: No, it's value is computed from `sum` and `numNumbers`
 5. `maximum/minimum`: *Hmmm...* I think I see a problem...
-

- It's tempting to say that `maximum` and `minimum` should be initialized to something like 0
 - But then my program will calculate the maximum of the user inputs *and zero!*
 - Suppose, for example, that the user only enters negative numbers
 - In this case, my program would report that 0 was the maximum because it is larger than all of the user inputs!
 - Similar problem with minimum
-

- *How can I solve this?*

- As stated above, I can't simply initialize `maximum` to some value, say x , because my program would then calculate the maximum of the user inputs *and x*
 - I need to make sure *only* the user inputs are considered when calculating the maximum
 - (this is all similar, of course, for `minimum`)
 - I have to do *something else...*
-

- Idea: The *first* time through the loop, I'll set `maximum` and `minimum` to whatever the user enters (stored in `currentNumber`)
 - In all *subsequent* passes through the loop, I'll only update those variables if what the user inputs is larger/smaller than the current values of those variables
 - This way, *only the user inputs* are taken into account when calculating the maximum and minimum
-

- At this point, I think that's it...
 - That is, *I think I could solve this problem myself*
 - Let's see if I can now design a detailed algorithm in pseudo-code to actually solve this problem
-

4 Designing an Algorithm

- Now that I have a complete understanding of the problem...
- AND I have some intuitive method for solving it...
- I design a detailed algorithm
- The one I come up with is the following, which I build using *top-down* design

A solution at 30,000 feet...

1. Create my variables and initialize them as described (only `sum` needs to be initialized)
2. Prompt the user to enter `numNumbers`, and make sure it's valid
3. For `numNumbers` times, do the following...
 - (a) Prompt the user to enter another number, stored in `currentNumber`
 - (b) Add `currentNumber` to `sum`
 - (c) Update `maximum`
 - (d) Update `minimum`
4. Calculate `average`
5. Display all of my computed values

4.1 Step 1: Creating and Initializing my variables

- All of my variables can be stored safely as numbers of type integer except for `average`... that needs to have precision
- The only Initialization needed is setting `sum` to 0

```
sum = 0;
```

- *That's it for this step*

4.2 Step 2: Getting `numNumbers` from the user

- For this step, I come up with the following algorithm
 1. Display message to user asking for a number (include in message what the number represent)
 2. Read in value and store in `numNumbers`
 3. If `numNumbers` is less than 1, go back to step 1
- *That's it for this step*

4.3 Step 3: The Main Loop

- This is really the “guts” of the program. My algorithm is as follows
 1. Count from 1 to `numNumbers` and each time do the following
 - (a) Display message to user asking for a number
 - (b) Read in a value from the user and store in `currentNumber`
 - (c) Update `sum` by `sum = sum + currentNumber`.
 - (d) If this is the first time through the loop (if count is 1)
 - Set `maximum = minimum = currentNumber`
 - (e) Otherwise, this is a subsequent time through the loop, and we update `maximum` and `minimum` as follows...
 - If `currentNumber > maximum` then set `maximum = currentNumber`
 - If `currentNumber < minimum` then set `minimum = currentNumber`
-

4.4 Steps 4 & 5: Calculate Average & Display Results

- This step is rather easy, and can be done by the following
 1. Calculate average as `average = sum / numNumbers`
 2. Display `sum`, `average`, `maximum`, and `minimum`
 - *That's it*
-

4.5 Putting it all together

- I've employed top-down design, starting by designing the solution to the problem as a whole and then drilling into each steps for more detail
 - Now I need to put all the steps together into a single algorithm
 - NOTE: In the slides-version of this lectures, the following is broken into multiple slides (this could not be helped). See the handout version.
-

1. Before I do anything, I create variables for `numNumbers`, `currentNumber`, `sum`, `average`, `maximum`, and `minimum`
2. I initialize `sum` to be 0
3. Display message for user to enter `numNumbers`
4. Read in value from user and store in `numNumbers`
5. IF `numNumbers` is ≤ 0 THEN go back to step 3
6. FOR `count` = 1 to `numNumbers` times, do the following...
 - (a) Display message to the user asking for a number
 - (b) Read in value from user and store in `currentNumber`
 - (c) Update `sum` by `sum = sum + currentNumber`
 - (d) IF `count` equals 1 THEN...
 - Set `maximum` = `currentNumber`
 - Set `minimum` = `currentNumber`
 - (e) OTHERWISE this is not the first number I've asked for, and I do the following...
 - IF `currentNumber` > `maximum`
THEN update `maximum` = `currentNumber`
 - IF `currentNumber` < `minimum`
THEN update `minimum` = `currentNumber`
7. Calculate the average as `average = sum / numNumbers`
8. Display `sum`, `average`, `maximum`, and `minimum`

-
- *And that's it!*
 - Notice the above psuedo-code...everyone in this room should be able to take each line and write a C++ statement that could accomplish the task
 - *And now we implement the above...*
-

5 Implementing the Algorithm

- Now that we have a *detailed* algorithm for solving the problem, we are ready to start writing code
 - Notice that with the detail we went into above, we really just need to do a line-by-line translation into C++ code
-
- Note: It's impossible to put the side-by-side translation in the slides-version of this lectures... move to the handout-version for this
-

Algorithm	C++ Code
<p>1. Before I do anything, I create variables for numNumbers, currentNumber, sum, average, maximum, and minimum</p> <p>2. I initialize sum to be 0</p> <p>3. Display message for user to enter numNumbers</p> <p>4. Read in value from user and store in numNumbers</p> <p>5. IF numNumbers is ≤ 0 THEN go back to step 3</p> <p>6. FOR count = 1 to numNumbers times, do the following...</p> <p>(a) Display message to the user asking for a number</p> <p>(b) Read in value from user and store in currentNumber</p> <p>(c) Update sum by sum = sum + currentNumber</p> <p>(d) IF count equals 1 THEN...</p> <ul style="list-style-type: none"> • Set maximum = currentNumber • Set minimum = currentNumber <p>(e) OTHERWISE this is not the first number I've asked for, and I do the following...</p> <ul style="list-style-type: none"> • IF currentNumber > maximum THEN update maximum = currentNumber • IF currentNumber < minimum THEN update minimum = currentNumber <p>7. Calculate the average as average = sum / numNumbers</p> <p>8. Display sum, average, maximum, and minimum</p>	<pre> int maximum, minimum, sum, numNumbers, currentNumber; float average; sum = 0; do { cout >> "Enter the number of numbers you want to enter: "; cin >> numNumbers; } while (numNumbers <= 0); for (int count = 1; count <= numNumbers; count++) { cout << "Enter a number: "; cin >> currentNumber; sum += currentNumber; if (count == 0) maximum = minimum = currentNumber; else { if (currentNumber > maximum) maximum = currentNumber; if (currentNumber < minimum) minimum = currentNumber; } } average = sum / numNumbers; cout << " Sum is: " << sum << endl << " Average is: " << average << endl << " Maximum is: " << maximum << endl << " Minimum is: " << minimum << endl; </pre>

6 Some Comments

- Wake Up!
- Yes, the level of detail and work we just covered was probably boring for most of you
- *However, most of you had immense difficulty during Lab 3!*
- What we covered above was **not** a solution to Lab 3, but rather an example of the **process** of designing software
- Notice that this methodology, although it may seem *too verbose*, we were able to start with a problem description and work up to a complete solution in C++
- In each step, we worked only slightly closer to the end solution in C++
- The difference from step to step, although minor, allowed us to *easily* and *intuitively* create a complete solution