Approach to procedural programming tasks

If you are faced with a more complex programming task, it is best to proceed using the top-down approach. A crucial step is to divide the generalized problem into individual partial steps. These subtasks, in turn, can be successively refined into further individual subtasks.

The time spent on the program development and debugging can be minimized by taking a thoughtful and structured approach to the program development. At the same time, a carefully considered programming concept increases the quality and readability of the program code if you consider a few simple and basic principles:

1. Produce a clear and unambiguous description of the requirements/problem.
2. Clearly describe which data is to be input and which data is to be output.
3. Create a rough draft.
4. Now divide and structure the generalized problem into individual partial steps. To do so, divide the overall problem into ever finer partial steps, which can, successively, be refined even more. ("From the general to the detailed = top-down-design") (In this step, you can use data flow charts or pseudo code).
5. Only now convert your solution approach into program code. If necessary, implement individual, independent program parts as subroutines.
6. Repeat the debugging and elimination cycle described above as often as necessary.
7. Pass through every test case and check whether the selected data types are suitable, whether the program can be further optimized (Is there redundancy?) and whether the task is completely solved.

Benefits of functions

1. ***Independent development and testing of the subtasks***

   Each individual subtask can be converted into code and compiled as an independent part of the program. Tests of individual program units can take part independently of other program units. This reduces the likelihood of unwanted interference between individual program parts.

   → **Program development - and the program - become more clearly organized and structured and less prone to errors.**

2. ***Reusable code***

   The logically independent functions can usually be reused again and again. In order to rewrite the same algorithm over and over again it is defined in a function only once. This prevents redundancy within the program code.

   → **The development and debugging times as well as the program code become shorter.**

3. ***Clear Structure***

   The passing of values between the individual program units is clearly regulated. Local variables are only valid in the individual statement block in which they were declared. The local variables can also only be used and modified within their program unit. This reduces the likelihood of unintended consequences elsewhere in the program as a result of program changes at another position in the code. Therefore, avoid global variables and use them only if they are really necessary.

   → **The risk of "programming accidents" is reduced.**

   → **Debugging is made easier.**

Example: program temperature

During the year, you have recorded the average temperature of the individual months. You now want to write a program that helps you to output this data. Write a program that outputs a menu. The following options should be available:

- *When you are entering a month, the corresponding average temperature should be displayed.*
    - *The input should be possible as text input (e. g. "August")*
    - *and as a selection (e. g. selection: 8 → temperature of August)*
- *Calculate the average temperature of all months.*
- *The hottest month of the year is to be output.*
- *All months should be output in an ascending order with their temperatures.*
- *If you enter 0, a goodbye message should appear and the program terminates.*
- *In case the user selection in the menu is non-existent, a message is to be output.*

You can use fictitious figures for the temperatures of the individual months.

Approach:

On the basis of this example we would like to answer the following questions in turn:

- **What needs to be done?**
  First of all, you should be aware of the requirements and the problems that may occur. You should therefore create a rough draft with the different types of instructions.

- **What should the rough structure of the program look like?**
  From the task above (*program temperature*) we can deduce that a selection has to be made. So we notice that we may be using a switch-case for this.
  Since entering 0 terminates the program, we can conclude that a loop around the initial menu is required.
  Based on the selection made in the menu different calculations and outputs follow, which are generated on the basis of the temperatures. The different case scenarios are therefore *functions* that perform the different options that are available in the menu.

- **How do I proceed?**
  Now that we know how the program is structured, we can create a more detailed design. We first write down which inputs (variables) and outputs we need. It is obvious that we need the following variables:
    - Array containing temperature values
    - Array containing the months
    - Input month
    - Menu selection

  Based on the plan and the individual functions, we can also write down the outputs of the program:
    - Menu with options
    - Average temperature
        - one month
        - all months
        - all months in an order
        - hottest month
    - Goodbye message
    - Message: wrong input

  Once we have defined the input/output, the program can finally be broken down into its individual parts. The first task is to implement the switch-case by using a for loop, since this only generates one output.
  The cases are then used to process the options that are available in the menu one after another. This is realized either by a flowchart or a pseudo-code. We then implement this in our programme. We write, test and improve our code until there are no more errors. If there is no more error, we will create the next function.
  If possible, functions should call each other or should be used several times. Since in the first requirement of the task the average temperature is to be displayed in two different ways, we can assume that these two functions are identical.
  If you are finished with the programming part, check whether the program meets your requirements, whether your data types are correct, whether you can optimize your code (Is there redundancy?) and go through all test scenarios in your program and check for logical errors. This can best be done by other people testing your program and giving you feedback.