How to analyze a complex problem

With illustrations from the COMP 40 assignment on arithmetic

Norman Ramsey

Tufts University

nr@cs.tufts.edu

This handout explains how to work through a complex assignment like the image compressor you build in the `arith` assignment. If you don't know how to manage such a complex assignment, the handout may give you some helpful ideas. It is not something to be handed in for a grade.

1. Start by getting your class notebook.
2. Write these headings:

   Things I need to complete or create

   Tasks Modules Data Types Algorithms Functions

3. On a separate piece of paper, write these headings:

   Things that are provided for me

   Modules Data Types Algorithms Functions Code

Now read the rest of the assignment paragraph by paragraph. For each paragraph, ask yourself these questions:

4. Does this paragraph describe a *task* you are to accomplish? If so, list it in your notebook under tasks. For nontrivial tasks, start a fresh sheet of paper for the task, and use it to keep track of modules, data types, algorithms, functions, and advice related to the task.

   For example, the last paragraph in the introduction to Part B in the `arith` *assignment gives you the task of implementing the* `Bitpack` *interface. The rest of Section 2.2 in that assignment gives the prototypes of the functions of the interface—the functions that you are tasked with implementing. These declarations are spread out over chunks called* ⟨*exported macros, functions, and values*⟩.

5. Does this paragraph describe a module, data type, algorithm, or function that is provided for you? If so, what is it? Does it solve a problem? Does it represent a subtask to be accomplished? Will it help solve a subtask you already know about? In that case, make a note of it on the sheet of paper for that task.

   For example, in the section on **Field-update functions** *in the* `arith` *assignment, the code fragment* ⟨*code to be used in file* `bitpack.c`⟩ *gives you the code you need to define the exception* `Bitpack_Overflow`. *It solves the tiny problem of defining that part of the* `Bitpack` *interface.*

   As another example, item 4(b) of Part A in Section 2.1 of the `arith` *assignment describes the function called* `Arith40_index_of_chroma`, *which is provided for you. It solves the subtask of converting a floating-point* P_B *or* P_R *into a small integer index. This subtask is used to help you solve the task posed in item 4(e) in that same section.*

6. Does this paragraph introduce a new type of *data*? If so,
   - How will that data be represented?
   - Will it be an *abstract* data type or will its *representation be exposed* in a header file?
   - What *functions* should be *packaged with the data type* in the same *module*?
   - Does it relate to a task? If so, note it on the sheet for that task.

   For example, the table in item 4(e) of Part A of the `arith` *assignment describes the format of a code word used to represent a 2-by-2 block of pixels. Although the code word is represented as a 32-bit integer, it really is a new data type.*[1] *The same paragraph also poses a task: given values* a, b, c, $\overline{P_B}$, *and* $\overline{P_R}$, *compute the value of the code word. You should write in your notebook the data type* code word *and note that this task is related to it. When you get to item 9 of Part A, you will find another task related to the same data type. Your notes will tell you that these two tasks, which are related to the same data type, belong in the same module.*

7. Does this paragraph suggest a module, algorithm, or function that you will need to develop? If so, what subproblem will it solve? What tasks does it relate to? How will it connect with other elements of the system in order to form a harmonious, useful whole? Will it need new data types?

   For example, item 4(c) in Part A almost certainly suggests that you develop a function from a quadruple of Y *values*

[1] One way to identify "code word" as a new type is that the usual operations on integers, like addition or multiplication, make no sense on code words.

to a set of cosine coefficients. How will you represent the cosine coefficients?

As another example, items 1–5 in Part A collectively suggest the development of at least one function. In order that your design be composed of general-purpose parts with clean interfaces, you might scrutinize this paragraph closely and decide whether these items should be implemented in a single function or should be spread out over more than one function. What design leads to the most harmonious combinations?

8. If this paragraph suggests a programming technique or a problem to watch out for, to which tasks, modules, data types, algorithms, or functions is it likely to be relevant?

   For example, the warning about quantization error in Section 3.1 of the `arith` *assignment ("traps and pitfalls") almost certainly applies to your average chroma values* $\overline{P_B}$ *and* $\overline{P_R}$*. You should make a mental connection to the output of the* `Arith40_chroma_of_index` *function and be aware that wherever you use that output, it is likely to include a significant error term.*

As you answer these questions, *organize the answers in writing* using the two sets of headers I've suggested for you, as well as a separate sheet for each major task.

Once you've analyzed the whole assignment, you can start tackling the design. Here are some more questions to ask:

9. What are my most important data types?
10. For each data type, should it reside in a module of its own, or should it be grouped in a single module with other data types?
11. For each data type, are there functions related to it that should reside in the same module with the data type? What are they?
12. Are there functions that are not necessarily related to any data type but are related to each other, so they should be grouped in the same module?
13. Which modules should be clients of which other modules?

At this point you are probably ready to design your compressor and decompressor. I encourage you to follow Wirth's method of *stepwise refinement*.[2] This method encourages you to decompose problems into subproblems. About any possible decomposition you can ask:

- Do all of the subproblems appear easier to solve (or better specified) than the original problem?
- Can I solve some of the subproblems with library functions?
- Are some of the subproblems so simple I know how to write functions for them?
- Can I test the solutions to subproblems independently of the whole problem?

During the process of refinement, you may *invent new modules, data types, algorithms, and functions* that you believe you need to help solve your problem. *Add them to your lists.*

[2] The lengthy exegesis on the first two pages of Section 2.1 of the `arith` assignment is really a first step in refining the problem "convert a PPM file to a compressed form." In this first refinement I made many design decisions; for example, I indicated techniques and algorithms to use, and I specified the form of a compressed image.