HOW TO PROGRAM IT

UNDERSTANDING THE PROBLEM

What are the inputs (or arguments)? What are the outputs (or results)? What is the specification of the problem? Can the specification be satisfied? Is it insufficient? or redundant? or contradictory? What special conditions are there on the inputs and outputs?

Does the problem break into parts? It can help to draw diagrams and to write things down in pseudo-code or plain English.

DESIGNING THE PROGRAM

Have you seen the problem before? In a slightly different form?

Do you know a related problem? Do you know any programs or functions which could be useful?

Look at the specification. Try to find a familiar problem with the same or similar specification.

Here is a problem related to yours and solved before. Could you use it? Could you use its results? Could you use its methods? Should you introduce some auxiliary parts to the program?

If you cannot solve the proposed problem try to solve a related one. Can you imagine a more accessible related one? A more general one? A more special one? An analogous problem?

Can you solve part of the problem? Can you get something useful from the inputs? Can you think of information which would help you to calculate the outputs? How could you change the inputs/outputs so that they were ‘closer’ to each other?

Did you use all the inputs? Did you use the special conditions on the inputs? Have you taken into account all that the specification requires?

WRITING YOUR PROGRAM

In writing your program, make sure that you check each step of the design. Can you see clearly that each step does what it should?

You can write the program in stages. Think about the different cases into which the problem divides; in particular think about the different cases for the inputs.

You can also think about computing parts of the result separately, and how to put the parts together to get the final results.

You can think of solving the problem by solving it for a ‘smaller’ input and using the result to get your result — this is recursion.

Your design may call on you to solve a more general or more specific problem. Write the solutions to these; they may guide how you write the solution itself, or may indeed be used in that solution.

You should also draw on other programs you have written. Can they be used? Can they be modified? Can they guide how to build the solution?

LOOKING BACK

Can you test that the program works, on a variety of arguments?

Can you think of how you might write the program differently if you had to start again?

Can you see how you might use the program or its method to build another program?

Writing the program means taking your design into a particular programming language.

Think about how you can build programs in the language. How do you deal with different cases? With doing things in sequence? With doing things repeatedly or recursively?

You also need to know the programs you have already written, and the functions built into the language or library.

Examine your solution: how can it be improved?

© Simon Thompson, 1996. Acknowledgements to How To Solve It by G. Polya upon which this table is based.