7. Bigger and Better (Comments, Functions and Procedures)

Most of the sets of rules an instructions we come across in everyday life are small. For example, the instructions on the side of a packet of pasta about how to cook it are usually a few lines long. Individual recipes are also usually only a dozen or so instructions long. With such simple sets of instructions, the way they are organised is not too critical. However, as the number of instructions involved increases it becomes more and more important that there be some structure. The instructions on the side of a clothes handwash are usually just written as unstructured sentences:

> "Add 50ml of hand wash to 10 litres of water. Allow it to dissolve completely. Wash clothes then rinse well in water. Rinse and dry hands thoroughly after use."

Computer programs can be sets of instructions that are millions of lines long. Even the most trivial programs are likely to be hundreds of lines long. With such large sets of instructions we need a way of making them manageable. How is this done with large sets of instructions in real life? Recipes are generally longer than clothes wash instructions and they are normally structured as a consequence. They often have numbered steps to make them easier to follow – the unstructured paragraph becomes a list. The recipe is given a title, and some description is given about it. An ingredients list is also often given separately. Ingredients lists correspond to type declarations as discussed earlier. We will look at each of these other structuring techniques and how they relate to programming in the subsequent sections.

> ***Fiorentina Pizza***
> This is a vegetarian pizza that makes a really good meal served with a Broschetta starter. Serves 4.
>
> **Ingredients**
> Yeast
> Water
> Flour
> Tomato Puree
> Cheese
> Spinach
> 1 egg
>
> 1. Mix yeast and water, add flour and stir
> 2. Knead for 10 minutes.
> 3. Leave to rise for 30 minutes.
> 4. Roll out the dough.
> 5. Place in a large round pizza tin.
> 6. Spread with tomato puree, and cheese and spinach.
> 7. Crack an egg into the middle.
> 8. Bake in oven for 25 minutes.

Craft books, such as one I have on how to make animated wooden toys (Holland 1995) have a near identical structure. They are divided into separate parts each giving an algorithm for achieving a separate task (i.e. a different toy: make a Flapping Goose

or a Tug-of-war Toy). A contents page lists them all at the start. Each has a title and a short passage giving general information at the start:

> "**The Shirt**
> Who will win: the goat or the farmer's wife? This parallelogram stick toy is at a large scale for added impact..." (Holland 1995)

This is followed by a list of materials and tools needed and then the actual instructions:

> "Copy the shapes from the pattern on page 45 on to the plywood. ..."

The book of card games I have had since a boy (Harbin, 1972) is similar. Each card game is described in a separate titled section, followed by what is required for the game (for example, two packs of cards, pencil and paper. This is followed by some general introduction: "This variation of rummy is included because it is widely played ... and is an ideal game for two players" (Harbin, 1972). Finally the actual rules of the game are given.

Comments

Recipes normally have with them passages that are not instructions to be followed, but are instead information about the recipe. They give information about how and when to use the recipe: "Good as a light snack", "Serves 6". Sometimes they give general information about the background of the recipe or how it has been adapted from some earlier recipe: "This is a traditional Indian meal"; "This recipe is adapted from one I came across when visiting Vienna. I replaced the Bavarian Cheese with Smoked Applewood to give it a more tangy flavour". They may also give some indication about whether the recipe is a quick or slow one: "Cooking Time: 20 minutes". One of the aims of this information is that a person trying to pick a recipe should be able to make that decision without having to look at the actual instructions. The information should answer the question: "Is this the recipe I am after?"

In programming terms this kind of information that is not an instruction to be followed is known as a **comment**. All programs should have comments to help others understand the situations in which the program should be used. Students learning to program often include comments that just repeat the instructions using different words. Such a comment is pointless as it gives no new information. As in a recipe, comments should give different kinds of information – the history of modification of the program, what situations the program should and should not be used and give any limitations on how well the program works.

In recipes the comments are normally placed up at the top of the recipe, so that they can easily be read before looking at the details of the recipe. Similarly comments are put at the top of a program where they are easily found. Other comments might be put within the recipe or program itself, giving extra information about a particular step to help understand what it is for:

> "Heat the paste for 2 minutes. (This helps ensure the sauce does not have lumps)."

Note that this comment (the sentence in brackets) does not just repeat the instruction, but instead gives a different sort of information.

Comments can also be used to give a shorter description of what a whole group of instructions do (the instructions themselves say how this is done). The comments then

give a less detailed description of the algorithm. For example the following instructions are based on those for putting together our travel cot (Mothercare 2000):

Opening the travel cot ← **Comment**

1. Unzip the carry bag.
2. Remove the contents.
3. Undo the tabs that secure the mattress.
4. Put the mattress aside.
5. Place the cot upright.

Straightening the top rails ← **Comment**

6. Lift the lock in the centre of the rail.
7. Rotate the lock.
8. Repeat the procedure for the other three sides

Without comments, it can be very hard to understand what a set of instructions are really for. Even if it was you who wrote the instructions it can be hard to understand if it is several months since you wrote them. I often write instructions to myself on post-it notes or on the back of my hand. Often by the next day I have no idea what the instructions were for. This is only a problem for humans – computers just follow instructions blindly without caring what they are for – but humans have to instruct the computer to follow a given set of instructions so there has to be something to tell those humans what the instructions do.

To give you an idea of how hard it can be to work out why comments are so useful, look at the following algorithm that was one of my favourites as a child.

1. Take an A4 piece of paper 30cm long.
2. Mark points every 3cm along its long edge to give 11 marks including those at each corner.
3. Draw a line parallel to the long edge of the sheet, 2.6cm from the edge.
4. Make a mark on this line 1.5 cm from the short edge of the paper.
5. Cut down the line, giving a thin strip of paper.
6. Starting at the mark just made, make marks at 3cm intervals along the line.
7. Draw two lines 3cm long from each of the 10 marks along the long line to the two marks nearest to it on the edge of the paper creating 19 triangles.
8. Cut off the two ends that do not make equilateral triangles.
9. Write the number 1 in the first triangle, 2 in the next, 3 in the next, 1 in the next and so on, so that 6 triangles have the number 1, 6 the number 2 and 6 the number 3.
10. Leave the last triangle blank.
11. Turn the strip over.
12. Draw on triangles on this side of the strip in the same way.
13. Leave the first triangle blank, then write 4 in the next two, 5 in the two after that, 6 in the two after that, 4 in the next two and so on until all but the first triangle has a number.
14. Fold a crease along each of the edges of the triangles.
15. Take the strip and fold the first two “4” triangles together.
16. Fold the first two “5”s on to each other.
17. Fold the first two “6”s on to each other.

18. Fold the next two “4”s on to each other and so on until the last two “6”s are folded together giving a shorter strip with only the “1”s “2”s and “3”s showing.
19. Now do a similar thing, folding each pair of “3”s onto each other to give a hexagon with a triangular tab that is blank on one side.
20. Fold the tab down and glue it to the other blank triangle giving a hexagon with six “1” s on one side and six “2”s on the other.

Do you have any idea what the above instructions do? I’ve told you in the instructions you will have a hexagon but why? In what circumstances would you follow these instructions? Those are the kinds of things comments would tell you. Even if you followed them, you might not have any idea what you had made or what to do with it. The instructions are virtually useless to a human without something else – comments. Good comments for these rules should answer all these questions. If you cannot answer them, think what it would be like to try and understand an algorithm containing a million instructions – programs are often that long.

Here are some comments that could have been given before the instructions above that should help you understand the instructions a little more. Even this may not be enough for you to understand – more documentation still is needed!

> **Instructions for Making a Hexaflexagon**
> A hexaflexagon is a puzzle. It is a six-sided piece of paper, of which only two sides can be seen at any one time. By “flexing” the flexagon each of the sides can be revealed. To flex a hexaflexagon you pinch together two adjacent triangles, and push the opposite two sides flat. Open up the flexagon from the centre to reveal a new side. Sides 1, 2 and 3 are quite easy to find. Sides 4, 5, and 6 are much harder to find. It takes at most 9 flexes to bring all sides to the surface.

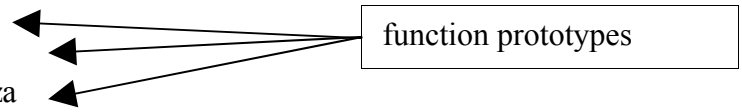
You can find out more about flexagons in the book *Mathematical Puzzles and Diversions* (Gardener, 1965) which is where I first came across them.

If you have trouble following the instructions, then you have probably learnt something about the ambiguity of English – you have thought the instructions meant something different to that which I intended in writing them.

Functions and Procedures

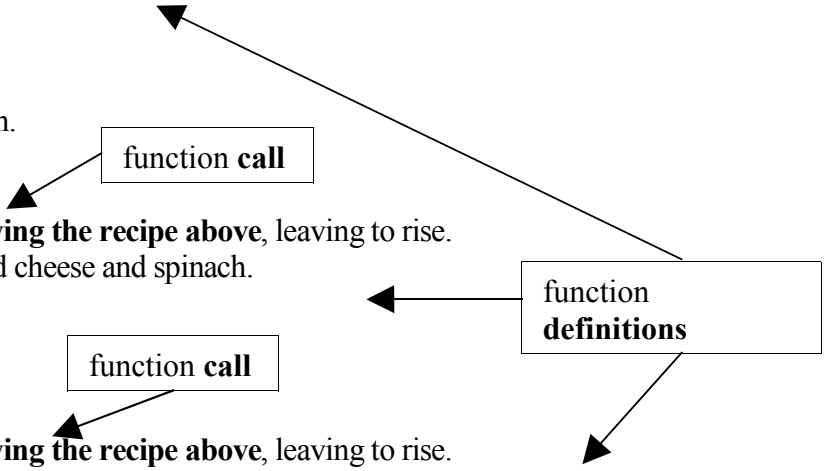
As the number of instructions gets larger new measures are taken. Recipes are generally combined into recipe books that contain in all thousands of instructions. The general structure of a computer program is very similar to a recipe book, though precise details vary depending on the programming language used. This is a difference to most written human languages: programming languages are very finicky about not only the grammar and spelling but also the format used. It is as though all written English had to be written following the format of a Delia Smith cookery book. Even following the format of a different author’s of cookery book would lead to the computer not being able to read the recipes. Let us consider a short recipe book for Pizza. It might start with a contents page, that list all the recipes inside. It then has a series of recipes, each having a name, followed by a set of instructions, to make that recipe. Suppose you wish to make Fiorentina Pizza, you would check where to find it in the contents, and then follow its instructions. In doing so you might, part way through be referred to another recipe, for example, to make the pizza dough. A similar structure exists in many programming languages.

Contents



1. Pizza Dough

1. Mix yeast and water, add flour and stir
2. Knead for 10 minutes.
3. Leave to rise for 30 minutes.
4. Roll out the dough.
5. Place in a large round pizza tin.



2. Fiorentina Pizza

1. **Make the *pizza dough* following the recipe above**, leaving to rise.
2. Spread with tomato puree, and cheese and spinach.
3. Crack an egg into the middle.
4. Bake in oven for 25 minutes.

3. Tomato and Chilli Pizza

1. **Make the *pizza dough* following the recipe above**, leaving to rise.
2. Fry the chilli pepper for 3 minutes, add onions and garlic and fry for a further 5 minutes.
3. Add tomatoes and spread over the base.
4. Bake in oven for 25 minutes.

In computer program terms, each separate recipe is a **function definition**. It is a self-contained set of instructions to do something that can be referred to in several other places in the program. Whenever, that thing is to be done, the person following the instructions is referred to the appropriate recipe/function definition. The point where one recipe refers to another is known as a **function call** when used in a program. To follow an instruction that is a function call, the person temporarily stops following the current recipe and jumps to the other instructions. When those instructions have been completed, the person returns to where they were in the original recipe and continues. Similarly when a computer executes a function call, it stops executing the current function, jumps to the other set of functions. On finishing following them it returns to the original instructions and continues where it left off.

This way of structuring a recipe book or program has several advantages. It means that the same instructions do not need to be written out over and over again. This saves space, but also reduces the chances of the different versions being different. It also makes it easy to substitute one recipe to do a particular thing for another. Perhaps we came across a pizza dough recipe that taster nicer, or was easier to make. We can change the instructions in just one place in the book and all the pizza recipes that use it now refer to the new version. If the dough instructions had been written out in every single recipe then we would need to make the changes in all those sets of instructions. Splitting up recipes or programs in this way can also make them easier to understand. An instruction such as “make the dough as on ...” makes it clear to someone who has made dough before, what they are doing. Reading the series of instructions

1. Mix yeast and water, add flour and stir
2. Knead for 10 minutes.

etc.

does not make it immediately clear what you are doing in the same way.

Similar issues arise from programs split into functions. It makes programs easier to write and understand, and helps ensure run-time errors do not creep in. It also makes programs easier to change. We will look at some of these issues in more detail later.

Each separate set of instructions in a recipe book is a recipe in its own right: it is an algorithm that does something distinct and worthwhile in its own right. Similarly functions in a program are individual algorithms that should do something coherent in their own right. You do not just take a random set of instructions from a recipe, and split them off as a separate recipe. Similarly you do not just take a random set of instructions from a program and turn them into a separate function.

Parameter Passing

When we cook dinner at home, my wife and I usually split up the tasks. I do the frying and stirring whilst my wife does the chopping. When it is my turn to be in charge I rarely cook to a recipe but tend to make up things as I go along (often ending up with a red or brown mosh). I rummage through the fridge and see what we have. "Chop this" I might say to Margaret on finding an onion. In essence what I have just done is execute a function call. Margaret is the chopping function – she specialises in chopping. I ask her to chop something and she chops it. The rules she knows about chopping vegetables are followed and on finishing she stops and waits for the next thing to chop. A difference to a function call is that with a traditional computer, only one thing can be done at once – so it is as though when I ask her to chop I stop and wait for her to finish before I go on to the next thing. However, something that is the same to a computer's function call is the way we pass the onion about. A computer program executing in a computer is more like the situation when my wife is away and I am cooking on my own – then I have to stop what I was doing and take her role chopping the onion. When I have finished I go back to what I was doing. I can only do one thing at once. Similarly when a function call is made when a computer is executing a program, the computer halts what it is doing to execute the instructions in the function.

One of the reasons for splitting a set of instructions up into functions is so that a general purpose set of instructions can be used to do a range of related things. I could have passed Margaret a tomato or a carrot and she would have chopped those just as easily. She is not just a specific "chop onion" function. She is more flexible than that. I **pass** her something and she **returns** back to me the result of her labours. The result returned depends on the thing I pass (If I were to pass her a pepper I would not be given back a chopped courgette for example, but a chopped pepper). She is acting as a general chop function so she chops whatever she is given but that means I must give her the thing to chop. In doing so I am **passing a parameter** to her. A parameter is just a thing that is passed to a function that the function performs its operation on. In computers the thing is a piece of information – data – rather than a real object. I may need to pass my wife some information when she is chopping as well as an object. She will need to know whether to chop the vegetable into big or small pieces. Thus I might actually say "Chop this onion into small pieces", or "Chop this carrot into large pieces". I am passing two parameters to the chop function: the thing to be chopped and the way it should be chopped. Functions do not need to I could pass even more information

A function call thus consists of two parts. I must specify which function I wish to be performed (here "chop"). I must also specify the parameters – in this case the thing that is to be chopped: "this onion" and the way it is to be chopped "small". A written instruction (as in a program) to do this must thus give a way of indicating these two things: the name of the function and a list of parameters. An instruction in a recipe reflects this:

Chop the *potato* into *large* pieces

Any programming language that allows function call must also give a way of indicating the same pieces of information.

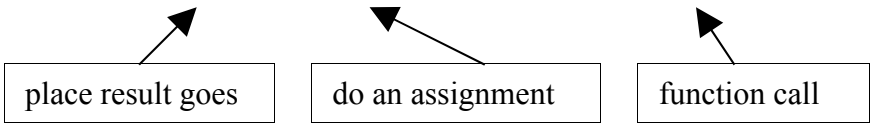
Once the chopping has been done, the result must be passed back. I did not just want the onion to be chopped for the sake of it, I wanted it chopped so that I could then do something further with the chopped onion: fry it for example (another function call, incidentally, this time to a fry function). The function must therefore **return** the result (chopped onions) back to the place where the operation was requested (me). A written instruction in the chop instruction must include something that indicates that the instruction is to return something and also indicate what it is that is to be returned.

We must also have some way of saying what is to be done with the result. In the case of chopped potatoes, we may wish to put them to be placed in a pan of water. Margaret returns the chopped potatoes to me and I put them in the pan. The kind of instruction we commonly use for this is an **assignment**. As we saw earlier, assignments are used when we want to specify that something is to moved from one place to another. We must specify what is to be moved and where it is to be moved to. The place they are to be moved to is the pan of water. The thing to be moved is whatever the result of the function call was which is usually indicated by the function call itself

"Chop the onion and put it in the pan"

Remember though that we said that in computer languages assignments usually are written the other way round – the place the thing is to go is placed first. It would be written more like:

"The pan gets the result of chopping the onion"



An alternative to doing an assignment with the result is to just pass it straight on to another function. On chopping the onion we then fry it. First we pass it to the chop function. The result returned is then passed to the fry function. In a recipe this might be written as "Fry the chopped onion putting the fried onion in the casserole dish". Again we need to know where the final result goes (the assignment here is to a casserole dish), but the result from the chop function is passed straight on to the fry function.

Procedures are just like functions in that they are a packaged up set of instructions for doing something specific. The difference is that a procedure is not intended to return a result. The point of a procedure is just to do something, not to produce something (the thing returned). To see what we mean by this distinction, suppose I had a servant Egor

who did whatever I said. One day I was thirsty so I gave Egor some money and told him “Get me a can of Lilt”. To ensure he did this properly, I wrote the instructions of how to get it on a piece of paper. He would follow the instructions, whilst I sat waiting under the shade of a tree waiting. Eventually he would **return** with a can of Lilt that he would give to me. We have executed a function call with a result (a can of Lilt) being returned to me.

On another day I noticed that Egor looked thirsty, so being a kind master I gave him some money and said “Buy yourself a Cola”. He would go away and follow my instructions (written on another piece of paper). Eventually he would return as before. This time however, I would not expect him to give me anything. He would not return to me until he had drunk the Cola. This time we executed a **procedure call**. We have followed a set of instructions, but the point was not for a drink to be returned, but just that some action be taken.