5. If only... (Selection)

"Is all good structure in a winding stair?"
George Herbert, *Jordan* (1633).

Sequencing is not the only form of plumbing used in programming: it is not the only way instructions can be ordered. Another form of plumbing is known as **selection** or **branch statements**. It allows a choice of actions to be performed depending on the situation. By choice here, we do not mean free choice, where you are just given two options and could do either. A branch statement tells you the two alternatives, but it also tells you precisely in which situation you would do each alternative.

Branch Statements
Return to the example of emergency instructions on an airliner, one of the things you are told is what to do if the cabin depressurises. You are told that oxygen masks will drop down from the ceiling. The basic instruction is simple:

1. Place mask over your nose and mouth.
2. Secure with the elastic behind your head.

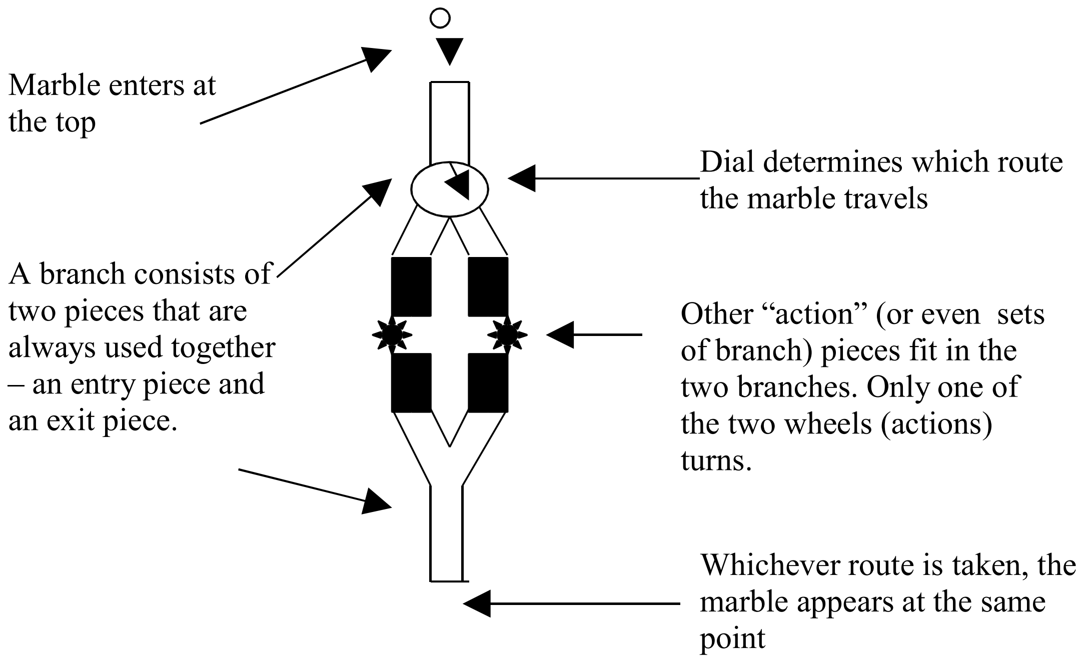
However, the instructions given also consider other situations than the basic one. "if you are travelling with a child then secure your mask first, then secure the mask of your child." Here the air-hostess is giving a second situation from the basic one and telling you exactly what to do if that situation is so. In English, we most commonly use the word "if" when there is a branching situation. We describe the particular situation and say what should be done in that situation.

if **you are travelling with a child** then ...

Here the situation that determines what we do is whether you are travelling with a child. This part of the instruction is not an action statement telling us to do something. Instead it is a descriptive statement. It asks a question about the state of the world. It is splitting the world in to two alternatives. Either you are travelling with a child or you are not. If you are you will need to follow one set of instructions. If you are not you will need to follow a different set of instructions. Both sets of instructions need to be listed. Thus a basic branch statement consists of three things: a question to ask yourself, and two sets of instructions to follow depending on the answer:

```
if you are travelling with a child
then
            secure your own mask
            secure the mask of the child
    else
            1.  secure your mask
```

Branch statements are the equivalent to having another part (actually a pair of parts that are always used together) in our marble run. The part has on pipe in, but two out of the bottom. A marble put in the top could appear out of either bottom pipe. Which one it appears out of depends on a two-position switch in the middle. When the switch is switched to the left, the marble goes one way. When it is switched to the right, the marble goes the other way. The part is always used with another part that connects the two pipes back to a single pipe.


Marble enters at the top
Dial determines which route the marble travels
A branch consists of two pieces that are always used together – an entry piece and an exit piece.
Other "action" (or even sets of branch) pieces fit in the two branches. Only one of the two wheels (actions) turns.
Whichever route is taken, the marble appears at the same point

This selection part is not used on its own but with the basic wheel parts. Wheel parts or sequences of them can be connected on each branch. If the marble goes one way it will turn one set of wheels, but if the switch is switched the other way, the marble will turn the wheels on the other branch. Eventually, whichever branch the marble passed down it gets to the join piece and drops out of the same slot at the bottom. In terms of a program, the flow of control passes down one of the branches, and only the actions on that branch are executed. Either way, the flow of control ends up in the same place at the bottom. Of course other wheel parts could be put on the bottom of our construction allowing further wheels to be turned (whichever branch the marble passed down.

We have seen selection already when we were looking at the properties of algorithms.

```
1. if the shop has Croissants
       then
               buy Croissants
       else
               buy Muffins.

2. Leave shop.
```

One branch of this selection is the action (wheel) "buy Croissants". The other branch is the action "buy Muffins". The selection plumbing in this example is indicated by the words if...then...else. The switch is the question "Does the shop have croissants?" If the answer to this question is yes (the switch is one way) then we move to the "Buy Croissant" action. If the answer is no (the switch is the other way) we move to the "Buy Muffins" action. In either case we drop off the bottom of the selection and do the next action in sequence – here "Leave Shop". We have combined a selection with a sequence. We will either "Buy Croissants then Leave shop" or we will "Buy Muffins then Leave shop".

Notice that the question is a boolean question. It is really "True or false: the shop has croissants?". There are two alternative answers and two possible actions. This is the situation where boolean calculations are used in programs – to determine which branch to take in a selection statement.

We discussed sequencing – doing actions one after the other – in terms of a relay race. Sequencing is a relay race where on each leg only one person is waiting to take the baton for the next leg. Selection is a relay race where there are two people waiting for the next leg. The baton is only passed to one of them who must then run the leg. The other person (and the other action) waits for another race (i.e. another run of the program) where they may or may not be passed the baton when it comes to their leg. The decision of who to pass the baton to is decided by the answer to the true/false question. If the answer to the question is true at that time then the baton goes to one person. If it is false it goes to the other runner. At the end of the leg, however, the baton gets to the same place whichever person ran the leg.

In the relay race version of computation, compiling a program corresponds to putting all the runners in the correct start positions and ensuring the questions that must be asked are available in the correct places. The race is not being run while this is happening – it is just preparation for the race. Running a program is like starting the race. The actions in the program are runners running legs. The end of execution of the program is when the last runner gets to the finish line. At this point the baton is given up. The race and the program are over.

The Imps handle branch statements by having an instruction that gives a choice of who to pass the baton to next. However they do not have a free choice. In fact the instructions tell them which of the two possible other instruction Imps to hand the baton to. The way this is done is by a boolean instruction.

For example, suppose the Imp computer was to follow the following instructions:

```
if (Bridget's value equals Alf's value)
then Alf gets the number 13
else Alf gets  the number 42.

Return.
```

Three instruction Imps will be allocated to this program – one for each line. The special instruction Imp that must do something more than we have so far seen is the first one. He has as instruction to find out if Bridget is holding the same value as Alf. Notice that this is a boolean expression which will have a true or false answer. When the instruction Imp for this instruction gets the baton, they get copies of the values of Bridget and Alf and pass them to the Equality Imp (who happens to be called Eric). Eric specialises in telling if two things are equal or not. Eric checks if the two things he has been given are equal and if so announces that the answer is true. If they are different he announces that the answer is false. The instruction Imp waits for this pronouncement eagerly because it determines what he does with the baton. If the pronouncement is "true" he passes the baton to the instruction Imp in charge of the "then" instruction who will give 13 to Alf. If the pronouncement is false however, he passes it to the "else" instruction Imp that gives 42 to Alf. Only one of these two instruction Imps actually takes part. The other can remain asleep. The baton is not passed to them. That does not mean that on some later day when the same set of instructions are run, the other branch will not be taken if the values held by Bridget and Alf have changed. Whichever way the baton is passed, it always ends up at the same point. Both the "then" Imp and the "else" Imp on finishing their instruction will pass the baton on to the Instruction Imp whose instruction follows the if-then-else. Here it is a Return statement, but it could be any instruction.

Our Imp computer can thus now do different things depending on the results of calculations or the values of data supplied. It is still however done without the Imps needing to know anything about what the overall task is. They just follow their instructions blindly, each knowing and being able to accomplish one small well-defined task.

Let us consider a more useful example. Suppose we wished our Imp computer to advise students on their exam results – telling them whether they have passed or not. An A is more than 80%, a B is more than 60% and a C is anything else. The student tells the Imp computer their mark and the Imp computer will tell them their grade. Alf will be used to store the mark and Bridget the corresponding grade. The instructions the Imps must follow is as follows:

```
Alf gets the score from the student.
If Alf's value is greater than 80
then Bridget gets the grade A
else if Alf's value is greater than 60
    then Bridget gets the grade B
    else Bridget gets the grade C.
Bridget's value is written on the screen (a large blackboard)
Return.
```

This will need 8 instruction Imps (one for each line) and an expert in determining when a value is greater than another, in addition to Bridget and Alf. What happens when we run this program will depend on the value given to Alf at the start. Let us watch some sample runs.

The first instruction is executed – the student provides a mark (84) and it is put in Alf's box. The baton passes on to the next Imp. Who gets a copy of Alf's value (84) and passes it and the number 80 to the "greater-than" expert. The expertt pronounces that 84 is indeed greater than 80, so the baton is passed to the first then Imp who holds the instruction "Bridget gets the grade A". Bridget is given the letter 'A' to store and the baton is passed to the Instruction Imp for the instruction immediately after the if-then-else statement. That is the instruction to write Bridget's value on the screen. This is done (by another specialist Imp who abseils down the wall of the computer with some chalk and writes 'A' (Bridget's value) on the screen. The baton then passes to the Return and on to M who announces the program terminated.

Another student arrives. This time they give the mark 69 to Alf. Now the answer to the first question is false. The greater than Imp pronounces "false: that Alf's value is **not** greater than 80 this time so instead of the baton passing to the "then" Instruction Imp, it passes to the "else" Imp whose instruction is actually another branch. He must determine the answer to the question "Is Alf's value greater than 60?" The greater than Imp pronounces "true" this time. The baton goes to the second then Imp who gives Bridget the value B. The baton passes on to the instruction after the if-then-else which again results in the abseiling Imp writing on the board. This time however he writes 'B' as that is the value in Bridget's box on this run of the program.

A final student has a mark of 2 (they got their name right). The first "greater than" Imp pronounces true leading to the second greater than Imp being questioned as above. This time however he answers "false". Bridget is given the value C which is written on the board.

Thus the Imps have executed the same program three times, with three different results. In one an A was written on the board, in the second a B was written on the board and in the third a C was written up. In each case the letter written was the correct grade for the mark presented. Branch statements thus give us a way of having the program make decisions based on the data input.

When writing computer programs we have to turn all such decisions into this more confined form. First you must recognise that there is a choice of actions in a situation, then work out what the question is, and finally decide what actions are done when the answer to the question is yes and what actions are done when the answer is no.

Sometimes the way instructions are phrased in English the question appears to be missed altogether. It is still there lurking however. For example, a recipe might say: Add the Pecorino (or Parmesan) cheese.

This really means

```
        If you could find the exceedingly hard to find Pecorino cheese
                (that I am only mentioning to impress you)
        then Add the Pecorino
        else Add Parmesan
```

"For a simpler ice cream replace the coconut milk with milk." (Abensur, 1996) can be rephrased as

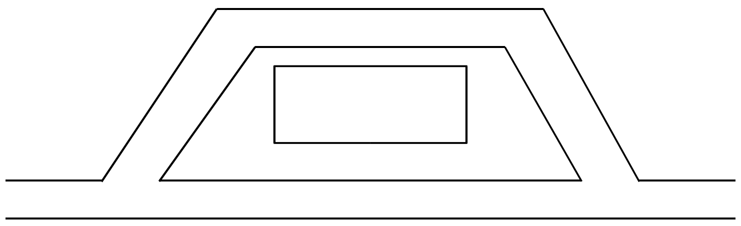
```
If you wish a simple ice-cream
then add milk
else add coconut milk
```

Single-branched if
Selection requires us to work out three things: the question that corresponds to the decision being made; the action(s) if the answer is yes; and the actions if the answer is no. Sometimes in one of the cases nothing is to be done until the branches meet again. This is known as a **single-branched if statement**. We can assume that nothing is done when the answer is yes, as we can always turn the question round to make this so.
The example we saw earlier of determining who goes through to the next round of the Euro football competition:

> *England will qualify for the quarter-finals with Portugal if they win or draw with Romania.*

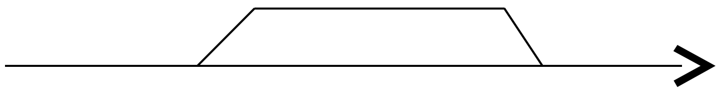
Here the true/false questions is "True or false: England won or drew with Romania". If this is true, this rule states that the action is "England qualify". This rule gives no action if the answer to the question is false. You must move on to the next rule in either case.

A motorway with a service station is a little like this. A driver, drives down the motorway. On seeing the sign for the Service Station, she asks herself the question "Do I need a break". If the answer is yes, she pulls in and has a coffee (the action). On finishing the coffee she pulls back onto the motorway, using the exit slip road. If the answer is no, she just drives on and performs no action. In either case she arrives at the same point to continue her journey. There was no other way out of the service station other than back onto the motorway, a little further along.



Now imagine the journey is a long one (from London to Edinburgh perhaps) and the driver makes the journey several times a year. Each time she makes the drive she will pass many service stations. On different trips she might need a break at different points, so on each time will take a different route. Occasionally she might make the whole journey without pulling in to any service station. If she had diarhea she might stop at every one. This is like a program consisting of a series of if statements placed end to end being executed. The road is the control structure (the plumbing), the service stations are actions (drinking coffee, going to the toilet) and the driver represents the flow of control. Each journey is an execution of the program, and on each different actions are performed depending on the particular situation as it effects the questions asked at each service station.

Suppose a person is walking down the street and comes to a ladder. Some people are superstitious and walk into the road to get round the ladder. An action stepping out into the road is needed. Others (like me) are perverse and make a point of walking under the ladder. Whichever you do once on the other side of the ladder, both people end up at the same point and continue down the road in the same way. No person walks both routes on one journey however (though on different journeys they may take different paths).



As with selection we have two possible routes. The switch in this case is the question "Is the person feeling superstitious". If the answer is yes, they will step out into the road. If the answer is no, they will continue along the pavement performing no action.

In its written form a singly-branched if statement might appear something like the following:

1) **if** the shop has Croissants

```
        then buy Croissants
    2) Leave shop.
```

The question here is does the shop have Croissants. If it does then we buy some if not we are not instructed to do anything by the if statement. In both situations we then Leave the shop – the next instruction after the if statement.

This kind of decision making often appears in instructions but the structure is often hidden within the language. A recipe might include an instruction “Season to taste”. This is just a singly-branched if statement in disguise. It means

```
If you like salty food
        then add the amount of salt you prefer
```

Notice if you do not like salty food, you do nothing for this step.

Multiple If Statements
Often there are more than just two alternatives but a whole range. To write instructions that cope with this we can combine together several branch statements. For example, the following is a joke that appeals to my sense of humour spoofing the emergency instructions on airlines. It was supposedly actually announced on a real flight. Here we write it as an algorithm.

```
1. if you are travelling alone
       then
           1.1 secure your mask
2. if you are travelling with a child
       then
           2.1 secure your mask
           2.2 secure the child’s mask
3. if you are travelling with two children
       then
           3.1 secure your mask
           3.2 decide which child you love the most
           3.3 secure the mask of that child
       3.4 secure the mask of the other child
```

This algorithm consists of three main instructions (numbered 1, 2, 3) that are performed in sequence. Each is an if-statement. They describe a situation and then say what to do if that situation holds. All three questions will be asked. However, the actions of only those that are relevant will actually be followed. In this case as only one of the questions can be true in a given situation, then only one will be followed. When putting branch instructions in sequence in this way, care has to be taken that all possible cases are covered, otherwise nothing will be done. For example, if following the above instructions, a father travelling with three instructions would not have any instructions to follow as the instructions only deal with the situations of none, one and two children. This is the issue of algorithms being **complete** that we discussed previously. You must cover all the cases.

Another common way to combine branch statements is to **nest** them. What this means is put one *inside* the other, rather than one after the other. For example we could rewrite the above instructions about putting on oxygen masks as follows:

```
1.      if you are travelling alone
    then
            A.      secure your mask
    else
            B.      if you are travelling with a child
                    then
                            secure your mask
                            secure the child's mask
                    else
                            if you are travelling with 2 children
                            then
                                    secure your mask
                                    decide which child you love the most
                                    secure the mask of that child
                                    secure the mask of the other child
```

This only has one main instruction: an if statement with two options.

```
1.      if you are travelling alone
    then
            A ...
    else
            B ...
```

If you are travelling alone then you do instruction A – you just secure your mask. All the rest of the algorithm is the something different that you do if you are not travelling alone – instruction B. Since you are travelling alone you can ignore it. It is not the instructions that are meant for you this time. You ask one question, take the action and you are finished.

However, suppose you are travelling with a child. You ignore the first part of the instruction as that is only for people travelling alone and instead follow the instructions for people for which this does not apply: you follow instruction B. But here B is actually another branch statement. B is the statement:

We used the example of the instructions for opening an emergency door on an airliner, earlier. On a different plane there were slightly different instructions: a different algorithm. It involved my making an observation (a test) and doing something different depending on the result. Before opening the door, I was instructed to check if there was a fire outside as if so I should use a different door. Only **if** the access was free should I open the door. I should in that case follow instructions similar to the original ones. Finally I needed to make another observation – I was only to use the door if the escape slide was there. This is an if statement inside another one – I only have to worry about the slide if their was no fire – if there was a fire the slides status is immaterial as I am not going to open the door in the first place. We use a nested if statement. The algorithm is as follows:

```
1. If there is a fire outside the door
   then
      1. Go to another exit.
   else
      1. Remove the flap covering the handle.
      2. Turn the handle.
      3. Pull the door into the plane
      4. Throw it out the doorway, as far away as possible
      5. If the slide is out
         then
            1. Climb out onto the wing.
            2. Slide down the slide.
         else
            1. Go to another exit.
```

Notice how the second if statement is part of the first. After each then and else is a mini-algorithm. The observation made as part of the test decides which mini-algorithm to follow, the other one being ignored. Either you go to another exit or you start to open the door – depending on what you see. The instruction to check if the slide is out is in the mini-algorithm that is followed only if there was no fire. This example also shows the distinction between the test and the actions that are carried out. Tests are the observations that appear after the word "if" like "the slide is out". Me looking does not make the slide go out, all I am doing is checking whether the slide is out. In general the act of making an observation does not change the state of the plane or people on it. The test observes what the state is. On the other hand the actions like "Pull the door into the aeroplane" change the plane – it changes the state of the plane from being a plane with a closed door to being a plane with an open doorway. It has an actual effect on the plane itself. "Climb onto the wing" is similar but this time it has an effect on the person – After doing this I am in a different place. The actions change the state, whereas the tests check what the state of something is. An if statement combines them – first making an observation, then based on that observation does one thing or another. Nested if statements require you to make one observation, then depending on the result possibly make other observations to decide what to do next.

The phone systems from hell that some companies use instead of employing real people are giving you instructions in this form. You know the ones where you press a different button depending on what service you want. The message will say something like this:

1. **if** you wish to buy a ticket **then** press 1.
2. **if** you wish to listen to some horrible tinny music **then** press 2.
3. **if** you wish to listen to recorded information that is of no use to you **then** press 3
4. **if** you wish to speak to an operator **then** press 4

This is a sequence of if statements, each with a different question. In each case the action to be performed is to press the button. You only press the button if the answer to the corresponding question is yes. The last possibility is there for completeness – if none of the other if statements covers your situation then this one will – you can talk to a real person! (Real phone systems from hell – do not have this as an option leaving you with nothing to do but hang up). Notice also that because the if statements are followed in sequence you still have to check (ie listen to) all the questions before the one that applies to you. The person who just wants to talk to a person still has to work through all the other questions before finding which button they need to press.

Often in real life the equivalent of this kind of information is given in terms of tables of information. We will see later how tables can be dealt with directly in a more general way. For now we will look at how they can be turned into if statements that have the same effect. Most shop doors have a sign on the door that looks like this.

Monday	8am-8pm
Tuesday	8am-12pm
Wednesday	8am-8pm
Thursday	8am-8pm
Friday	8am-8pm
Saturday	9am-5pm
Sunday	10am-4pm

This notice is allowing you to work out what the opening times are. It says that if the day is Monday then we are open 8am-8pm, if the day is Tuesday then we are open 8am to 12pm, and so on. Writing this out as a sequence of if statements, we get.

if the day is Monday **then** the opening hours are 8am-8pm.
if the day is Tuesday **then** the opening hours are 8am-12pm.
if the day is Wednesday **then** the opening hours are 8am-8pm.
if the day is Thursday **then** the opening hours are 8am-8pm.
if the day is Friday **then** the opening hours are 8am-8pm.
if the day is Saturday **then** the opening hours are 9am-5pm.
if the day is Sunday **then** the opening hours are 10am-4pm.

This is not normally how it is done on doors, but the same information could have been done more compactly using the following table:

Tuesday	8am-12pm
Saturday	9am-5pm
Sunday	10am-4pm
Monday, Wednesday, Thursday, Friday	8am-8pm

This can be written in fewer instructions and is quicker to follow.

if the day is Tuesday **then** the opening hours are 8am-12pm.
if the day is Saturday **then** the opening hours are 9am-5pm.
if the day is Sunday **then** the opening hours are 10am-4pm.
if the day is Monday or the day is not Wednesday or the day is Thursday or the day is Friday **then** the opening hours are 8am-8pm.

We have grouped together cases that require the same action. This does not actually save us that much but with a slight change it can do. Notice that only one of the if statements will ever be true. The last one is really a catch all – it is saying "and any case I have not covered do the following". We really want to use an else statement.

However we do not want it to be an else to just one of the other rules: the following would be wrong

```
if the day is Tuesday         then the opening hours are 8am-12pm.
if the day is Saturday        then the opening hours are 9am-5pm.
if the day is Sunday          then the opening hours are 10am-4pm.
                              else the opening hours are 8am-8pm.
```

Can you see the problem? Suppose the day is Tuesday, then the first rule's question is true so we are told that the opening hours are 8am-12pm. However we then move on to the next instruction. Tuesday is not Saturday so we do nothing, but now move on to the third instruction. Tuesday is not Sunday, so the else case comes into action. We are now told that the opening hours are 8am-8pm. We want the else case to apply to all the other ifs as well. That can be done by nesting them.

```
if the day is Tuesday         then the opening hours are 8am-12pm.
else if the day is Saturday   then the opening hours are 9am-5pm.
else if the day is Sunday     then the opening hours are 10am-4pm.
                              else the opening hours are 8am-8pm.
```

Notice that we have added **else** before the second and third **if**. What that means is we ask the first question. If it is Tuesday, then we are told the opening hours and we ignore the else – which is everything else up to the final else. Only if it is not Tuesday do you check if is Saturday, Sunday or none of these.

Problem
Sad Joe's Cafe uses the following Breakfast Price List on the wall. Write an algorithm using if statements like that above that Joe must follow to Charge the correct price for a single meal.

Sausage, Egg and Chips	**£4**
Bacon and Eggs	**£3**
Omelette	**£3**
Blueberry Muffin	**£1**

Here is another algorithm that saves lives that uses a series of if-statements. This version is adapted from a medical encyclopedia (BMA, 1990) but the same algorithm is taught on first aid courses:

First Aid in case of Suffocation:
1. Remove any obstruction.
2. Move the victim into fresh air.
3. **If** the victim is conscious **then** offer reassurance.
4. **If** the victim is unconscious **and** breathing normally **then** place in the recovery position.
5. **If** the victim's breathing is difficult **or** has stopped **then** begin artificial respiration.

This is a sequence of 5 instructions to be performed in order, but where three of the instructions are if statements. The action in each case is only taken if some condition is true. There is only a point offering reassurance to victims that are conscious for example. Only having checked that the person is conscious do you move to the next check – are they unconscious and breathing normally. If so place in the recovery position and then go to the next instruction. Notice that in fact only one of the three conditions will ever be true so only one of the actions will be taken. This means nested if-then-else statements could have been used instead.

Problem
Rewrite the suffocation first-aid algorithm using nested if-then-else statements.

Arguably the order given to do the checks is wrong. If the victim is conscious then things are basically okay – you do not need to offer reassurance instantly. The most important thing to heck is actually the last one as if their breathing has stopped you need to apply artificial respiration immediately or they will die. It should be the first if statement.

Problem
Rewrite the suffocation first-aid algorithm to ensure that the checks are made in order of importance.