Conditional control structures give your programs the ability to look at data and then decide what to do next. Yes, your programs finally have the ability to "think"!

These "If Statements" are found in practically all programming languages (at least the ones that count) and are used whenever the computer needs to make a decision.

**Part 1 – Modules**

A module is an independent “subprogram” that can be “called” by your main program to carry out a specific task. Modules make a programmer’s life much easier, especially when dealing with large amounts of code. It makes your code easier to understand, easier to maintain and, ultimately, results in less code! Modules are created “outside” of the main program and can be called from main or another module.

Let's add a nice introduction to your flowchart – and future flowcharts!

1. Start a new flowchart

2. Start by selected "Add a Function" from the Program Menu or clicking on the add icon on the main toolbar. Flowgorithm uses the term "function" rather than "module". This is also the case with most languages.

3. A new window will open.

4. In the name box, type: *introduction*.

5. For the "Return Type", select "none". You'll get to use that feature in the future (not this week).

6. Click OK. A new flowchart will appear. Notice that there is a dropdown the toolbar with the text "introduction". You can use this dropdown to switch between any modules you create.

7. Add a couple output shapes to your Introduction module. Print your name and your section. If you want to add more output shapes (to make your introduction look cool), please do!

8. Now, return to your main module (using the dropdown menu).

9. Add a Call Shape to your flowchart.

10. Double-click on it and enter *introduction* in box.
Part 2 – Using Parameters

Modules can do much, much more than just print text to the screen. A module has two basic components: a **name** and **parameters**. Look at the following pseudocode (in the book's format).

```plaintext
Module showSum(real x, real y)
    Display x + y;
End Module
```

So, this module has the following attributes: the name is `showSum` and it has the parameters `x` and `y`. These parameters act like local variables which are assigned the values you pass in.

1. Click on the "Add Function" again.
2. For the name, enter `showSum`.
3. Add some parameters. Make both real with the names `left` and `right`.
4. For this part, we are just going to output the sum of the two numbers. So create the flowchart below.
5. Go back to Main (using the dropdown box).

6. Create the flowchart below – the first 4 shapes: two declares and two inputs.

7. Next, create a “Call” block of your module and double click on it. Your arguments are the values you pass from your calling program to your module. These can be numbers or variables.

Look carefully at your variable names in main. They are named first and last. The parameters inside the showSum are called left and right. The name of the parameters and the arguments do not have to match. You are passing in values; not names.

8. Execute it. Now, the same function can be called multiple times with different arguments, giving different results each time.
Part 3 – Conditional Logic

The most common decision structure is the If Statement. The if statement begins with a condition, followed by a set of statements that will be executed only when the condition is true, and a set of statements that will be executed when the condition is false.

In lecture, you have learned to use pseudocode to represent If / Then / Else. The same control structure is represented in flowcharts. The diamond shape represents the condition. From this diamond, there are two branches – one if the condition is true and another if it is false.

Let’s look at the following pseudocode (written in the book’s format). It’s not a full program.

```
Input operator
If operator == "+" Then
  Call showSum(first, last)
End If
```

It will call the `showSum` module if the operator variable is equal to "+".

1. Add a variable declaration for `operator`. Make it a string – so the user can enter text.
2. Add an Input shape for the operator (with a nice caption)
3. Now add an If Shape. You will notice that there is a branch for `true` and a second branch for `false`.
4. You now need to move your Call Shape to the true branch.
5. Execute for flowchart. Try entering a plus sign for the operator and some other symbol.

Notice that your module is not called if you entered a value other than a plus sign.

Part 4 – A Calculator

Now that we have a basic version of our calculator flowchart working, let's make it a tad fancier. The program only displays the sum. That is, if the user enters a plus sign.

1. Don’t delete the flowchart. You are going to add more modules to it.

2. Create modules for that will display the result of addition, subtraction and division. They are going to look a lot like your showSum module.

3. Your If Shape only executes showSum if the operator is +. Add three more If Shapes for the operators: -, /, and *. Make sure that each If Shape calls the correct module.

4. Execute it. Test it for all 4 operators.

Did your output look something like this?

If not, keep working on your flowchart until it works properly.
EXPLORE: Flowgorithm's Code Viewer

Did your flowchart match the pseudocode? Well, Flowgorithm has the ability to convert your flowchart to traditional text-based programs.

1. Select the "Source Code Viewer" from the Tools menu. A new window will open. It will show, at first, your program in Java – which is a very popular language.

2. On the dropdown box, select "Gaddis Pseudocode". Yes, this is the same pseudocode used by our textbook.

3. Check to see how closely your program resembles the pseudocode above

Part 4 – A Better Calculator

So, your flowchart now can perform addition, subtract, multiplication, and division. But, what would happen if you enter 0 (zero) as the second number and try to divide?

1. Try it. Enter 12 as the first number, 0 as the second number and / for the operator.

Did you get following window?

Uhh ohhh. You just destroyed the Universe by dividing by zero! Well, not really. This is a "runtime error". Your program performed an illegal operation and essentially crashed. Fortunately, there is a solution!

2. Open your showDivision module (or whatever you called it).

Notice that you divide by zero in every case? Well, you need to prevent that by using an If Shape. In many cases, you will write programs that avoid run-time errors by preventing them from happening. If the denominator (the right operand) is zero, you should display a message such as "Divide by Zero".

3. Add an If Shape. Study what needs to happen. This time you are going to use both the true and false branches. You will have to copy-n-paste some shapes.

4. Execute and test your flowchart.

Did you get an error message again or did you solve the problem? If you got an error message, take a close look at what happened.