# LOGICAL CONTROL STRUCTURES
(Showing Flow of Control)

For many years program analysts have been using an approach to program design called "structured design" in which programs are assembled like building blocks from simple "structures" that define the flow of control to be followed by a processor when executing the program's commands. Three simple structures have been identified that suffice to build a solution to any complex task. These structures are known as: Sequence, Selection, and Repetition. Basic forms of each are shown in the illustration below using standard flowcharting notation.
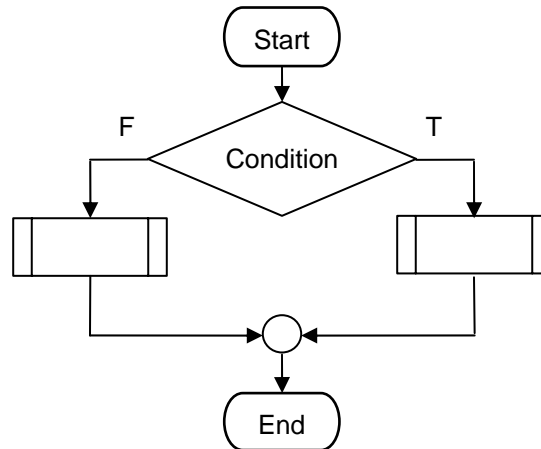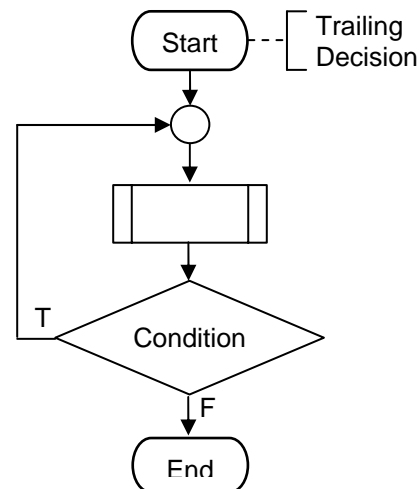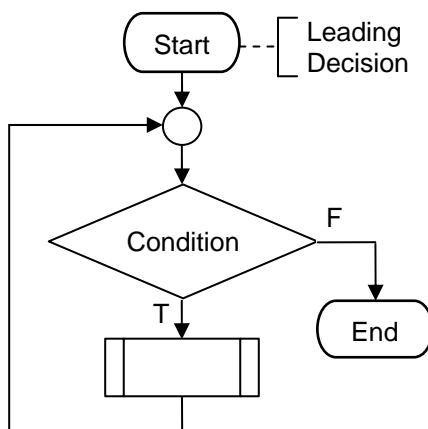
SEQUENCE
(No Branching)
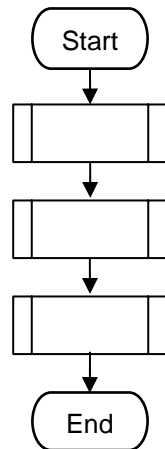
SELECTION
(Branching Forwards)

REPETITION (Two Different Looping Styles)
(Branching Backwards)
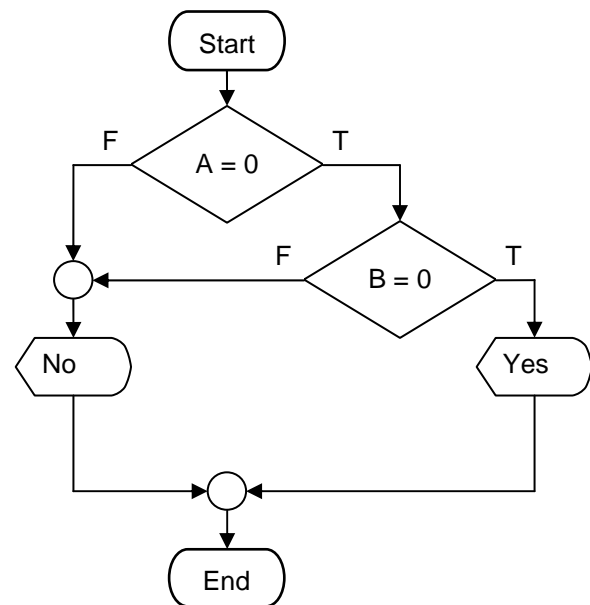
Structured vs. Unstructured Flow of Control

Using flowcharts (graphic algorithms), any process can be diagrammed to show the flow of control from one step to another. But some designs are easier to implement than others. Consider the following example. Given two integer variables named A and B have already been properly defined and stored, consider the two flowchart segments below that graphically illustrate the following algorithm step:

> If the values stored in variables A and B are both zero, then display the word "Yes", otherwise display the word "No".
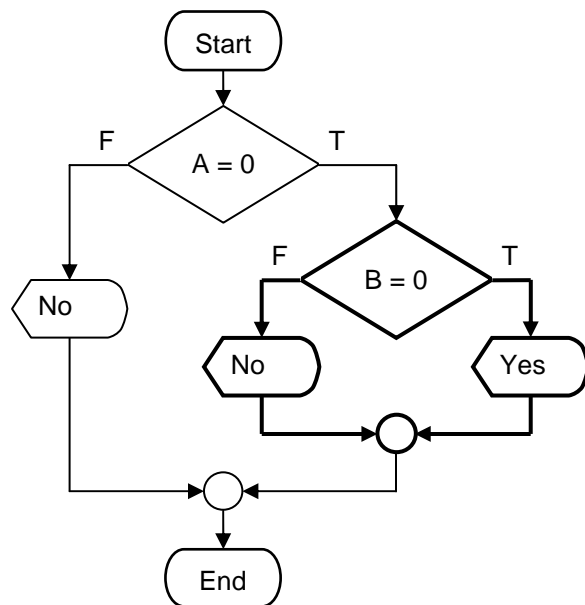
Both flowcharts properly define steps necessary to accomplish the objective.

The flowchart on the left uses "structured design". This means that it is based on the use of a limited set of control structures that are assembled like building blocks to accomplish the desired task. For example, the test for B=0 starts a selection structure (shown **bolder**) that closes at the first circle. That structure is "nested" inside the True leg of another selection structure (shown lighter) that starts with the test of the A=0 condition and ends at the bottom circle. The the bolder structure that tests B would be referred to as the "inner selection" and the lighter one that tests A is referred to as the "outer selection". The thing that makes the left solution "structured" is the fact that it could be assembled from some combination of the three basic control structures shown on the previous page, each of which have only one entry point (start) and one exit point (end). Notice that the inner (bolder) selection is completely self-contained on just the True leg of the outer selection. The benefit is that the inner (bold) selection could be extracted and written as an independent function and then called at that point in the diagram.

The C Language source code for the structured flowchart above would be:

```
if (A==0)
   if (B==0) printf ("Yes");
   else printf ("No");
else printf ("No");
```

Remember that indentation means nothing to the compiler in C, although it may help programmers to read the code and visualize the structure better. Each *else* statement is paired with the most recently unpaired *if* statement.

The flowchart on the right (above) uses "*un*structured design" because it is *not* based on the use of the basic control structures. The test for B=0 starts a branching process that resides on the True leg of the outer structure, but it's False leg branches across into the False leg of the outer structure. The inner selection is not a structure that could be extracted because it has no clearly defined single exit point.

Initially, the unstructured approach might seem better because it does not require the extra (redundant) output step to display "No" that we see in the structured approach. The difficulty is that the C language was written based on the use of the three basic control structures. C offers statements to perform the actions defined in those structures. So if we design algorithms based on those structures (or assembled from combinations of them) then these algorithms can be easily coded in C. We can write each small structure as a function and then call each one where needed in other structures. The disadvantage is that we sometimes pay the price of redundancy to obtain this ease of coding.

If we code the bolder inner selection as a function (shown on the right below and named "Inner"), then the flowchart for the outer selection would be simplified to the one shown on the left:.
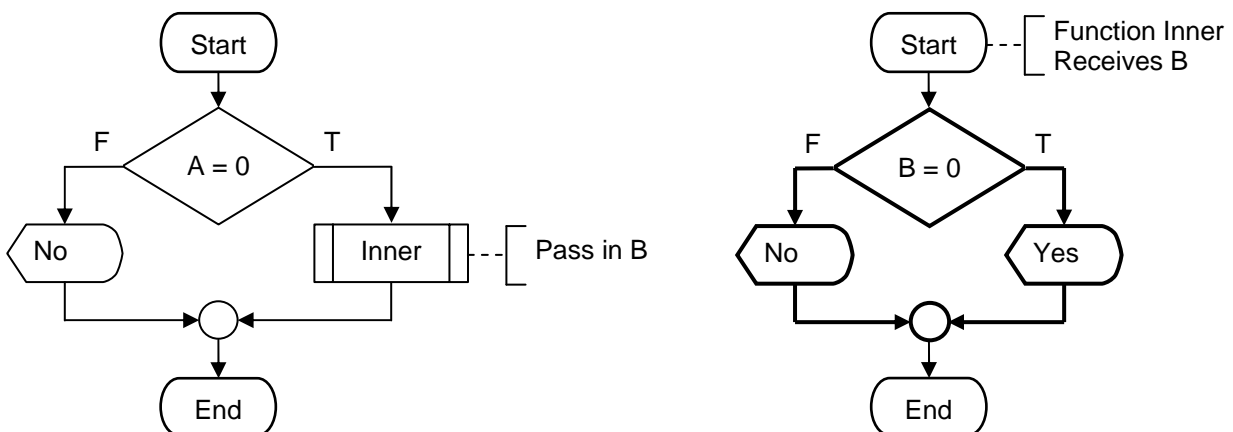
Notice the use of comments to show the formal passage of the value of B. The task has been broken into two uses of the simple selection structure, in which the parent function (on the left) calls the child function (named "inner" in this example). This is an example of procedural abstraction discussed in your textbook. The workings of Inner might be unknown or defined later, but the parent function can still be written. The code for Inner would be:

```
void Inner (int B)
{
   if (B==0) printf ("Yes");
   else printf ("No");
}
```

The calling statement for Inner would appear inside of an if statement in the parent function:

```
if (A==0) Inner (B);
else printf ("No");
```

For more information about logical control structures, see the [Example Links for Chapters 4 & 5](#) for our course textbook on the course web site.