

Chapter 4

C: Branching & Looping

Now that I have discussed variables and numbers and the operations that can be applied to them, I will move on to program flow in terms of branching and looping. A program that, on power-up, performed one sequence of things and then stopped has only limited use. Much more can be accomplished if it performs different operations depending on information taken in from external inputs—*branching*. It might do the same thing multiple times—*looping*. Overall, your programs should never terminate but go on polling for new inputs until the power is shut off.

Decisions

A computer's power lies in making decisions and carrying out different operations based on the result. For example:

Based on the condition of a switch, turn on the water or not.

Check for the signal telling you that the A-D converter is done and, if so, collect the reading.

If this operation has been repeated 22 times, then go on and do the next operation.

All these are examples of a *decision* that a microcontroller would be routinely asked to make. Based on the result of a decision test, the program flow will *loop* (go back on itself) or *branch* (go in one of several possible directions).

Flowcharts

Flowchart: a pictorial view of a program or function

The basic structure of a program is revealed in a *flowchart*. I will illustrate basic parts of flowcharting as I go into the different types of program flow. My style may be different from flowcharts you have seen before—mine are intended to be only a *quick overview* of the solution method.

For the C constructs that follow in this chapter, I use a limited set of flowchart symbols. Computer science, particularly in the area of data processing, has developed a more extensive set, but most of them

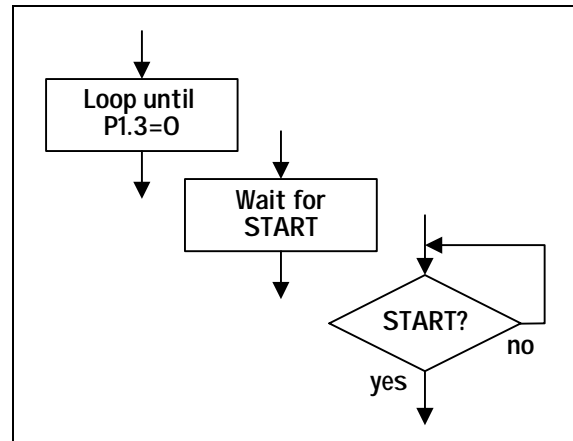


Figure 4.1: Descriptive Flowchart Progression

are unnecessary here. Iterative loops in particular may be charted in other ways.

One Page I argue that flowcharts should fit on a *single* page. I have occasionally seen multi-page flowcharts with paths leading to points several pages away. Although technically correct, such flowcharts do not help you understand what is going on in the program. When a program becomes too complex to fit on one page, simplify it (and the program it represents) by assigning pieces to functions (discussed in Chapter 5). The *details* transferred to those functions can be shown in *separate* flowcharts on other pages. As a result, the *main* flowchart will always give an overview of the entire program on one page. If you need to see more detail about a part of the solution, you go to the appropriate function and its flowchart.

Use Descriptive Words All flowcharts should talk about things by their purpose or function rather than by referring to specific variable names. For example, consider the flowcharts in Figure 4.1 showing a routine to wait for a button to be pushed. It is more descriptive to have a block like, *Wait for START*, than a block like, *Loop until P1.3=0*? Better yet might be to loop on *START?* as shown as the third choice.

P Code There is also a form of charting and planning based on *pseudo code* or *P code* where the actions are laid out in blocks of *words* with indentation