Structured Design

Using Flowcharts

C Code Implementation

Revision 2.0
December 2001

Andrew J. Blauch
Paul D. Johnson
Padnos School of Engineering

BACKGROUND

Simple programming exercises can often be solved by just sitting down and writing code to implement the desired problem. Complex problems, however, can be difficult to write and impossible to debug if not implemented using a structured design process. There are many structured design methodologies that can be used to implement programs and solve other types of engineering problems. Using a structured design process leads to the following benefits:

- Early detection of design flaws
- Programs that can be easily modified
- Clear and complete documentation
- Modular design to improve testing
- Modular design to break up problem into smaller sections

The application of a structure design methodology greatly increases the probability of completing a successful design with a minimum amount of time and expense. By using a structured design methodology, the likelihood of finding design flaws early improves considerably. Finding design flaws early in the design process greatly reduces the cost of fixing those flaws. A programming problem that might be fixable for a few dollars early in the design process might cost many thousands of dollars to repair when the flaw is not detected until the project is near completion. Structured design methods also improve the ability to modify programs at a later date since the techniques make the production of clear and complete documentation much easier. Structured designs can also be more easily broken up into modules to improve testing and to allow development by multiple design teams with a reasonable assurance that the resulting products will be compatible with each other.

STRUCTURED FLOWCHARTS

There are many complex design methodologies for implementing large hardware and software projects such as a new corporate database or operating system. Projects implemented in embedded control, however, usually require much less code and thus need an appropriate design level. The technique discussed in this document is a top down, structured flowchart methodology.

Basic Blocks

The basic elements of a flowchart are shown in Figure 1. The START block represents the beginning of a process. It always has exactly one output. The START block is labeled with a brief description of the process carried out by the proceeding flowchart. The END block represents the end of a process. It always has exactly one input and generally contains either END or RETURN depending on its function in the overall process of the flowchart.

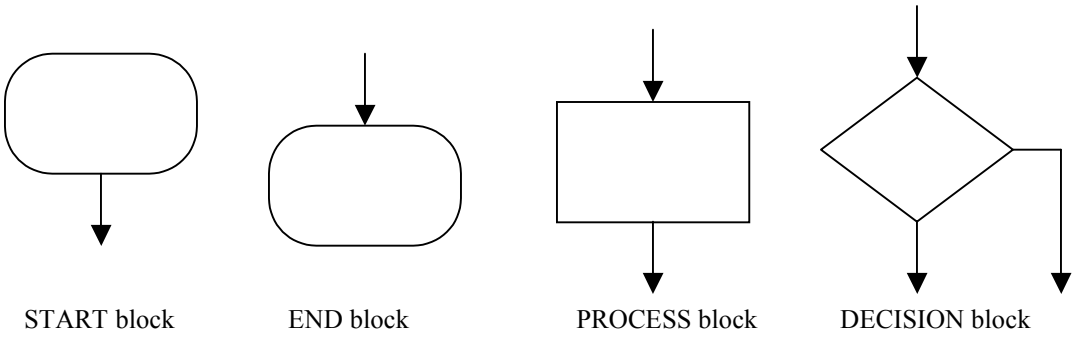


Figure 1: Basic Flowchart Blocks

A PROCESS block represents some operation carried out on an element of data. It contains a brief descriptive label describing the process being carried out on the data. It may itself be further broken down into simpler steps by another complete flowchart representing that process. If it is broken down further, the flowchart that represents the process will have the same label in the start block as the description in the process block at the higher level. A process always has exactly one input and one output.

A DECISION block always makes a binary choice. The label in a decision block should be a question that clearly has only two possible answers. The decision block will have exactly one input and two outputs. The two outputs will be labeled with the two answers to the question in order to show the direction of the logic flow depending upon the decision made.

On-page and off-page CONNECTORS may also appear in some flowcharts. For this document we will restrict ourselves to flowcharts that can be represented on a single page.

Basic Structures

A structured flowchart is one in which all of the processes and decisions must fit into one of a few basic structured elements. The basic elements of a structured flowchart are shown in Figure 2. It should be possible to take any structured flowchart and enclose all of the blocks within one of the following process structures. Note that each of the structures shown below has exactly one input and one output. Thus the structure itself can be represented by a single process block.

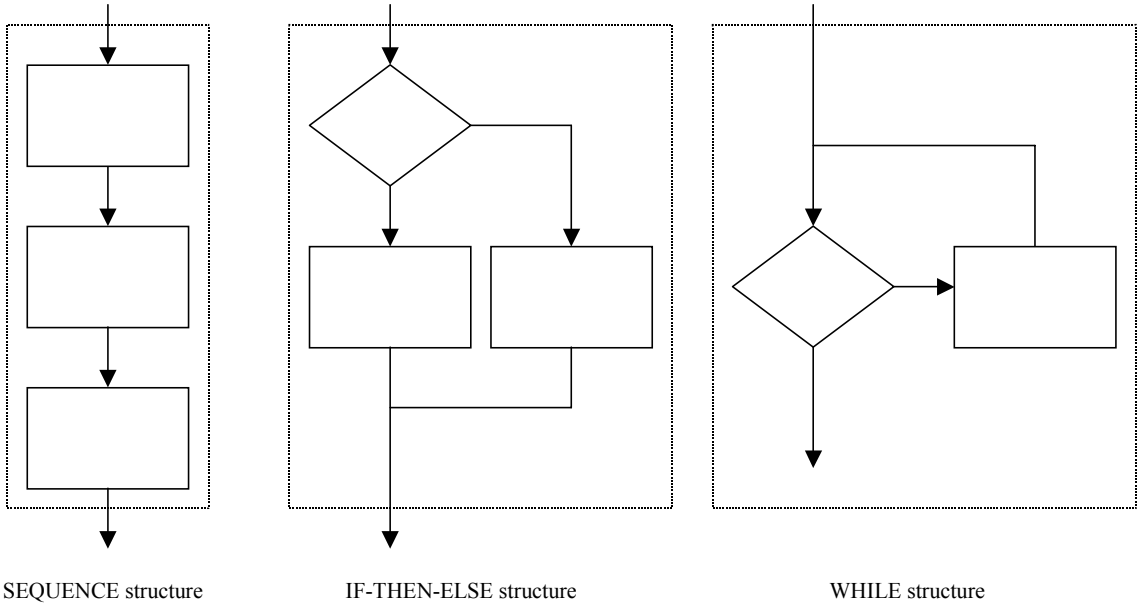


Figure 2: Basic Flowchart Structures

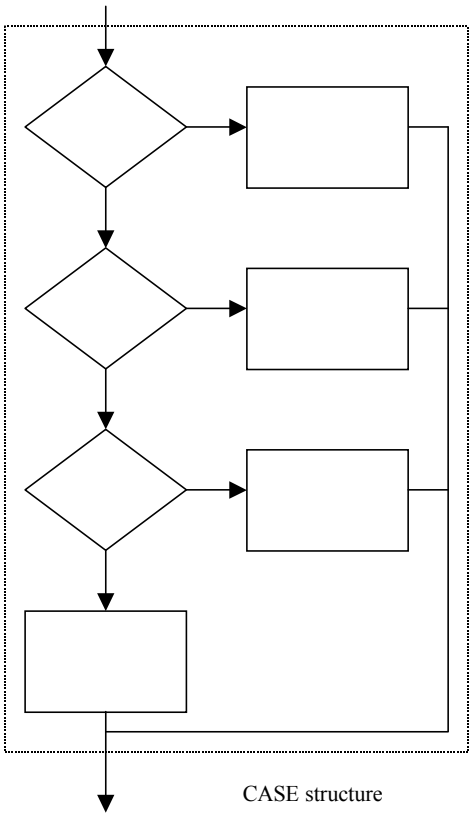
The SEQUENCE process is just a series of processes carried out one after the other. Most programs are represented at the highest level by a SEQUENCE, possible with a loop from the end back to the beginning.

The IF-THEN-ELSE process logically completes the binary decision block by providing two separate processes. One of the processes will be carried out in the each path from the binary decision.

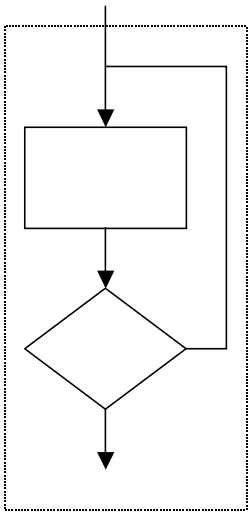
The WHILE process allows for the representation of a conditional loop structure within a program. The decision to execute the process in the loop is made prior to the first execution of the process.

Derived Structures

Although all flowcharts can be represented by the above basic structures, it is sometimes useful to employ some additional structures, each of which can themselves be constructed from the above structures. These derived structures are shown in Figure 3.



CASE structure



DO-WHILE structure

Figure 3: Derived Flowchart Structures

The DO-WHILE structure differs from the WHILE structure in that the process contained within the loop is always executed at least one time. This is equivalent to performing the process once before going into a WHILE loop. In the WHILE structure the process may never be executed. Although the WHILE structure is preferred, the DO-WHILE structure is sometimes more intuitive.

Similarly, the CASE structure is useful in representing a series of IF-THEN-ELSE statements where there are more than two choices to be made. Hence the DECISION blocks are identical except for the choice being compared. For example, the DECISION could be ‘is the color of the sock …’ Each DECISION block would then have a different color as the choice. The true result always flows to the right, with the false result flowing into the next DECISION block. There will always be one less DECISION block than the number of choices.

Examples of Good and Bad Structured Flowcharts

Figure 4 shows an example of a properly and improperly structured flowchart. The unstructured flowchart is an example of what can happen if a program is written first and then a flowchart is created to represent the program. This type of unstructured flow is often called 'spaghetti' programming and normally has elements of its structure impossibly intertwined around other elements. A program of this sort is very difficult to understand, implement, debug, and maintain.

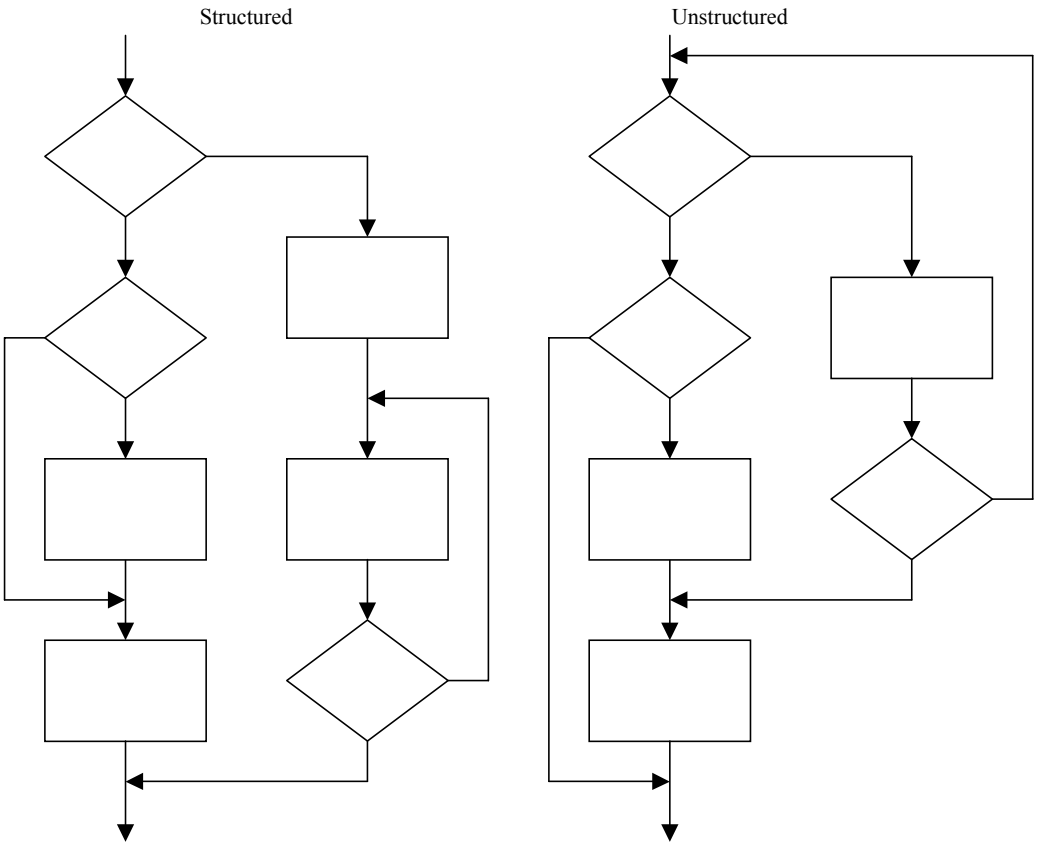


Figure 4: Structured and Unstructured Flowchart Examples

FLOWCHART IMPLEMENTATION USING C

Writing C code from a structured flowchart is very straightforward. Each type of process structure described previously has a corresponding C code program flow statement. Figures 5 and 6 illustrate the C code implementation of the three basic and two derived process structures. Note that the single process blocks can be implemented with a single line of code, multiple lines of code, a function call, or another group of structured process blocks.

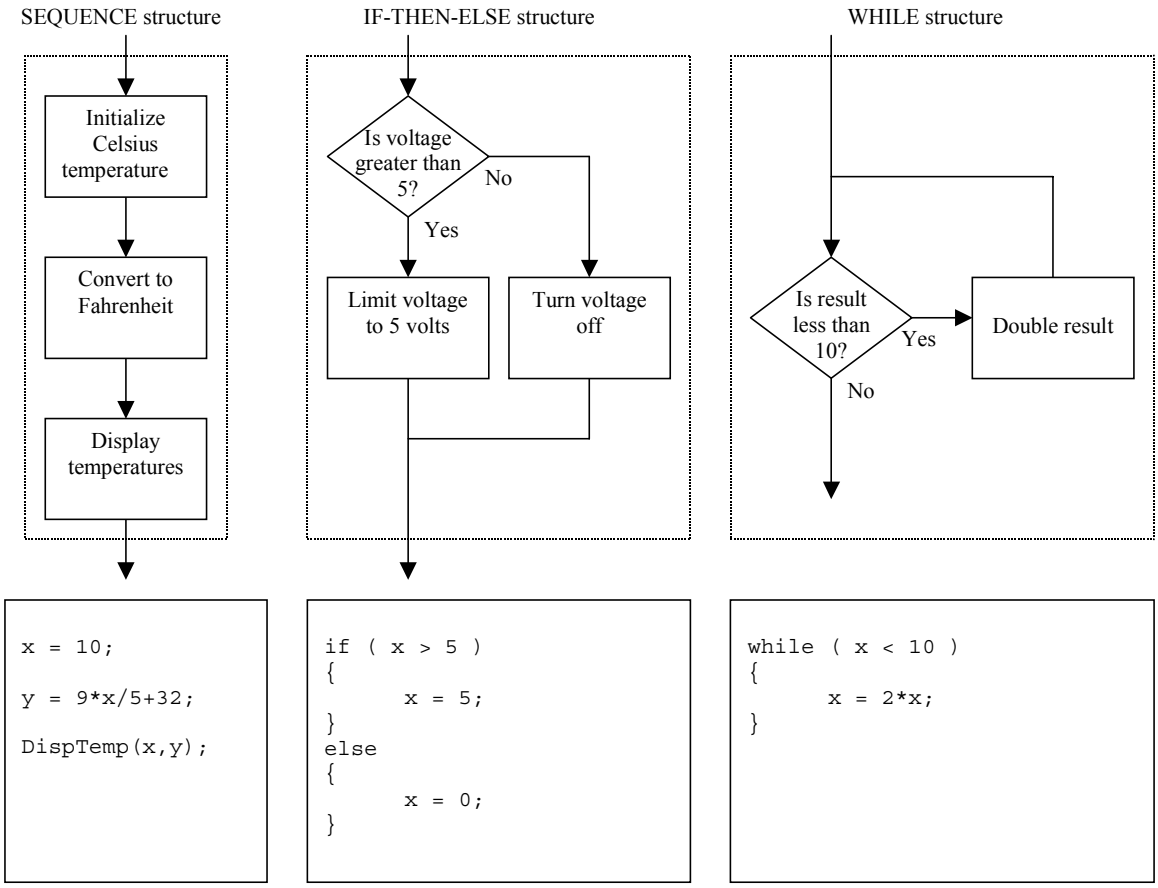
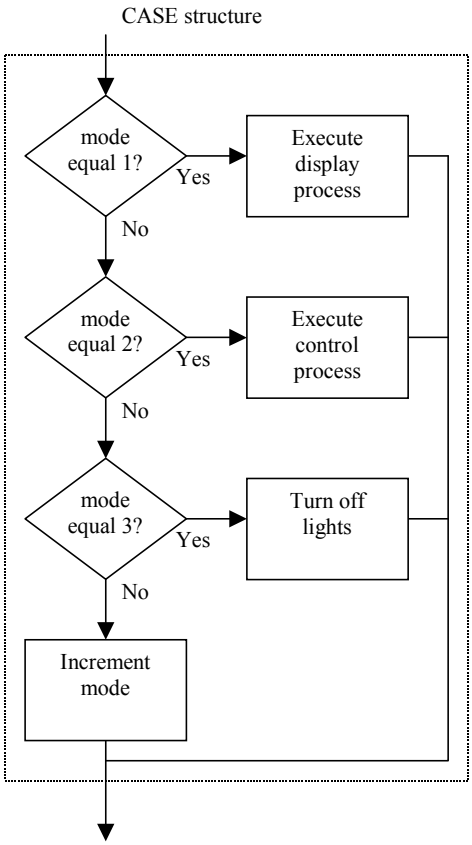
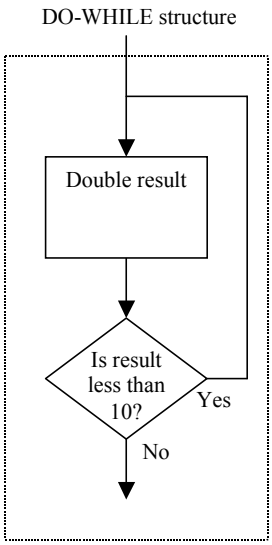


Figure 5: C Code Implementation for Basic Flowchart Structures






```
switch (mode) {
case 1:
      /* Display process */
      break;
case 2:
      /* Control process */
      break;
case 3:
      red_light = 0;
      blue_light = 0;
      break;
default:
      mode = mode + 1;
      break;
}
```

```
do
{
      x = 2*x;
}
while ( x < 10 );
```

Figure 6: C Code Implementation for Derived Flowchart Structures

TOP-DOWN DESIGN

Structured design using flowcharts generally involves a top-down analysis of the problem to be solved. This begins with a carefully structured textual description of the problem to be solved. This phase often involves talking with the customer to determine just exactly what will be needed in the design implementation. At this point external inputs and outputs are defined. From the text description, a top-level flowchart is created. This describes the processes to be carried at a high level with a minimum of detail. Often this highest-level flowchart is a sequence of processes. In an embedded application where most programs never end, the END block may be replaced by a loop back to the beginning or some other point in the program.

Each process block in this high level flowchart is then broken down into greater detail through another flowchart that describes how the higher-level process is to be implemented. In a complex design, many levels may be required before reaching a sufficiently detailed level of description to write the code. If carried out properly, the each block in the lowest level flowchart will represent no more than one, or a few, assembly language instructions.

Properly executed, the analysis and flowcharting stages of the design process will probably consume 80% or more of the total program design. The actual code generation should be very simple after the flowcharts have been completed. The labels in the blocks on the flowcharts will become an outline of the comments to be provided within the program itself.