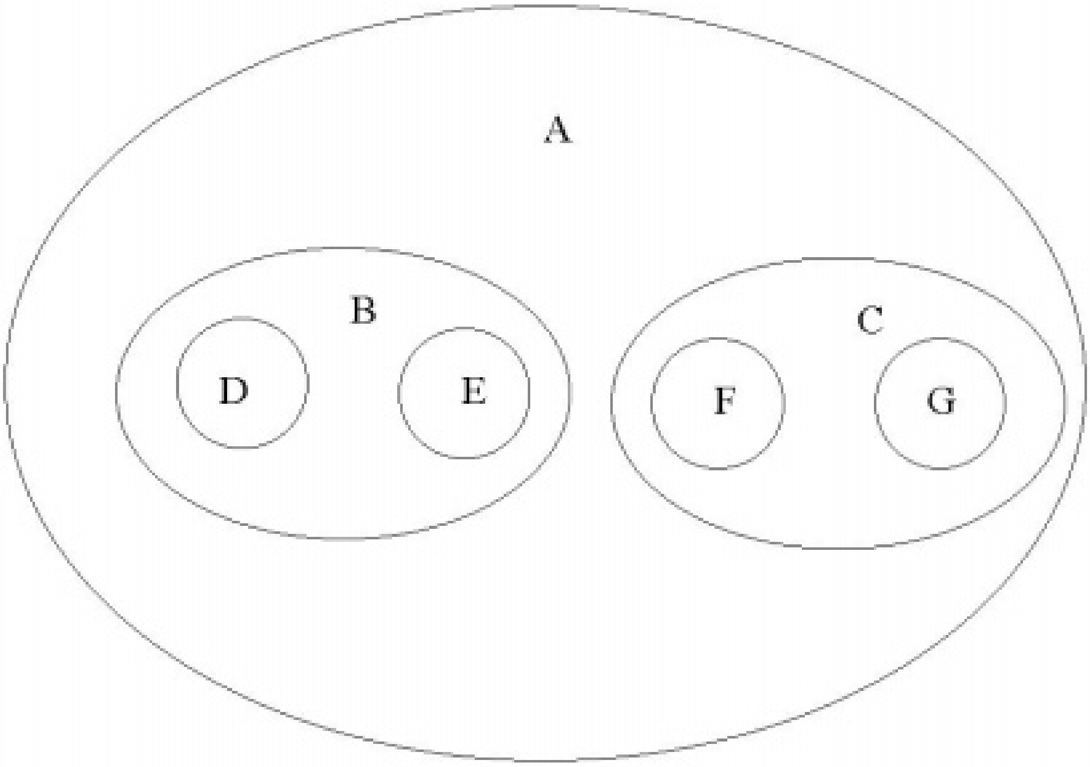
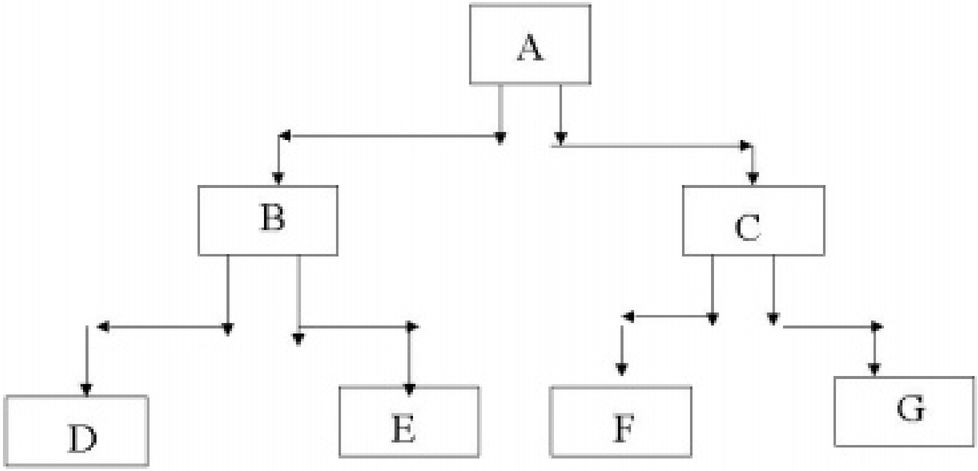


Structured Programming

Reconsidered.



Hermann Schmitt
Dipl.Volkw., Dipl. Math
e-mail: 100042.1471@compuserve.com
Germany, July 6. 1998

Structured Programming Reconsidered.

The Implementation of Structured Programming.

The fundamental idea of structured programming is, to build the programms out of three basic structures: condition, iteration and sequence.
In order to implement structured programming, often new control structures were introduced into the programming language, goto's were no more allowed. The main aim of structured programming is, to make the programs more understandable. I think, that this is accomplished to a certain degree.

But structured programming made it possible to decompose the programs. This possibility was not taken into consideration. Instead the control structures of the programming languages led - in my opinion - to an excessive nesting of structures into each other. I think this nesting of structures does not make it easier to understand the programs. This is especially true for conditions with several branches, into which other structures are nested.

In this article it is distinguished between the fundamental idea of structured programming and its implementation. It is shown, that structured programming could be implemented in another way and it is explained, what advantages could be gained.

The OO method is essentially another criterion for the structuring of software. But I think, these two methods of structuring do not exclude but complement one another. With the OO method the software is structured into classes and methods. But the OO method does not give any advice for the structuring of a single method (program). Here, I think, the method of structured programming is still valid. And although the OO method and structured programming are very dissimilar starting points, the outcome ist very similar, if structured programming is implemented in the way proposed in this article.

In programming languages it is frequently recommended to indent the statements in order to show the structure. This is - in the first place - a proof of my opinion, that nested structures - as expressed by the programming languages - are difficult to understand. Further the indentation has several drawbacks itself. To begin with, the indentation shows the structure the programmer believes the program has. The real structure may differ because of errors. Then it is difficult to keep the indentation consistent, if the structures extend over several pages, and if the indentation would be held consisistent., it would be difficult to discern, which statements on different pages have the same nesting level i.e. belong to the same control structure. Further, the space left for writing statements gets narrower with the depth of the structure, therefore indentation is also not generally feasable.

To understand a program it is most important first to see one structure in its entirety, structures nested into this structure can to be considered later.

Instead of indentation I propose, that with a service program comments are inserted at those places into a programm, where the nesting level changes in either direction and to specify the nesting level as a number in this comments. This does not overcome the drawback that the components of a structure may be far apart, but the components of a structure can easily be identified, because they have the same nesting level.

It makes sense to indent, when writing small parts of a program. But later on comments specifying the nesting level should be inserted into the program and the identation should be removed.

I think, that the specification of the nesting levels as comments in the program makes the program more understandable, but is remains the drawback, that the components of one structure may be far apart from each other. This problem is adressed in the following.

Hermann Schmitt
Dipl.Volkw., Dipl. Math
e-mail: 100042.1471@compuserve.com
Germany, July 6. 1998

The Representation of Tree Structures.

Structured programming leads to a program structure, which is similar to a tree structure. Tree structures are ubiquitous in computer science They are represented in many different ways, depending on the context. In the following only two representations of tree structures are shown which are important for this article.

The most common representation of a tree structure is probably the following:

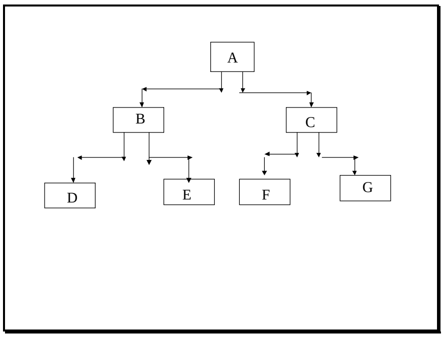


Figure A.

Another represenataion is the following:

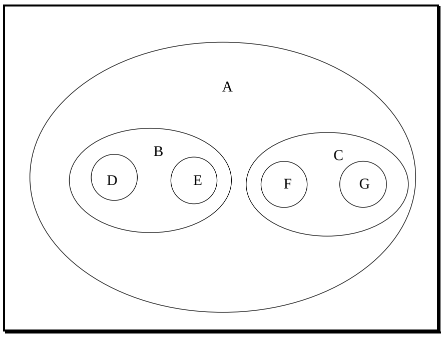


Figure B.

The Representation of Program Structures

Comparing this two representations of tree structures with a program that is structured according to the common rules of structured programming, is is immedately apparent:

The representation of a program structured according to the common rules of structured programming corresponds to the representation of a tree structure according to figure B.

But the represention of at tree structure according to figure A is - in my opinion - in most cases better understandable.

This leads to the question, if programs structured according to the rules of structured programming could also be represented in a way which corresponds to figure A, and if this is the case, what advantages and disadvantages this could have compared with a representation according to figure B.

A representation according to figure A can be accomplished by storing structures in different files and inserting references at those points in the source, where decomposed parts have to be inserted. By a preprocessor the parts could be put together to a program which can be treated by the compiler.

In order to make the described method of programming usable for programmers, the composed program, which is the input for the compiler, should be transparent for the programmers. For this it is e.g. necessary, that error messages of the compiler are modified in such a way, that the location of the error is specified by naming the file, where the affected component is stored, and the row number in this component instead of the row number in the composed program. Of course one could envisage a compiler, which takes the components as input directly.

The described representation of programs would have the following advantages:
- the structure of the program would be better recognizable.
- the components of one control structure would not be so far apart.
- The components could be reusable be it in the same program or in different programs.
- it could be easier to test parts of a program by removing references or reference more primitive routines, instead of the original ones.
- In the test phase test code could be automaticlly inserted before and after the separately stored components, for instance test code, which prints the name of the component.
- Changes of the strucuture can sometimes be accomplished by changing, adding or removing references.
- it would be easier to put parts of a program into a subroutine or on the other side integrate subroutines into a source program. I.e. programming would get more flexible in this repect.
- if the structure of the components is correct, the structure of the composed program is correct. Errors of the structure can be found more easily in a single components than in the composed program. The compiler cannot assist to find errors of the structure.
- it would be easy to give an outline of the program according to figure A, by putting the names of the files (components) into the boxes.

It is not feasable to put each structure into a separate file. It seems to be suitable to develop the program top-down. A new component should be started, if an important condition with more than one branch is encountered. As already mentioned, conditions may be difficult to understand. As the control structures are only partially expressed by components and references, the control structures of the programming languages should not be removed. Data fields should be defined in that component, in which they are used, in order to make the components more reusable.

In this way the components has a condition as its basic structure. This condition should be expressed in a special way to make it more visible. In order to achieve this, the notion of program array (P-Array) as extension of the notion of arrays of datafields is introduced. The conditions for the different branches are conceived as indices and the routines belonging to the conditions are concieved as the contents of the indices. In certain cases it may be practical to express the indices as integers beginning with zero, for instance, to identify the routine of a special branch more easily. The indices of P-Arrays are an ordered set.
For practical purposes, components without a condition as basic structure are allowed, too. I think e.g. in Java a facility is missing, which allows to insert source statements at compilation time.

As a consequence a program should no longer be conceived as a linear sequence of statements but as a tree structure, where each elements of the tree is an array with the conditions as indices and the routines belonging to the indices as "values" of the indices. In a further step the components could be stored in a database structured into indices and "values" of indices.

In my opinion programming is made easier, because the programmes can always concentrate on one condition, which is the basic structure of a component. Further it is easier to add or remove branches of conditions. The problem is splitted into parts, which may be solved one after the other.

Taking into account the satructure of the P-Arrays an outline of the programm could additionally contain the indices (conditions) of each component. If the conditions themselves are not meaningful, comments should be inserted into the conditions.

Conclusion.

In software development systems often the emphasis is laid on the graphical user interface and the visual composition of the graphical user interface out of components. This is very impressive, but I think for larger programs it is of equal importance that the program is structured correctly and that especially the conditions are coded in a form, that can verified and checked easily. I think the programming method explained in this article can help to accomplish this.

I have developeled software, which makes it possible to build programs in the way described in this article. The software is coded in Java 1.1. It is usable for programs to be written in the proprammming languages Java and C/C++. Because the software is written in Java it can be used on all platforms.