Chapter 7

CONCLUDING COMMENTS

In this ultimate chapter, we wish to summarize the organization of the program, introduce its presentation in the appendices, and suggest some possible extensions.

§7.1 A Guided Tour of the Program

We show, in Table 7.1, the imbrication of the procedures in the program. The main program together with initialization can be found in Appendix A. Options such as *d, blen, clen,* etc. are introduced as constants. The major options, *k, adjacency* and *surrounding* are introduced interactively. To this end, variables "**adjacency**" and "**surrounding**" are given the type "**adjacen**" and "**surroun**" and take value 1 or 2 for *full* and *restricted* respectively.

The processing of rows, *i.e., process on row, transition to the next row,* and *window,* is the subject matter of Appendix B. The central procedure *allocate* can be found in Appendix C. One may note that the code of *allocate* has been kept independent of the choices of both *adjacency* and *surrounding* options.

The ten procedures in Appendix D perform the processing of objects. The major option which is relevant there is *adjacency.* Whenever needed, (*conbelow*

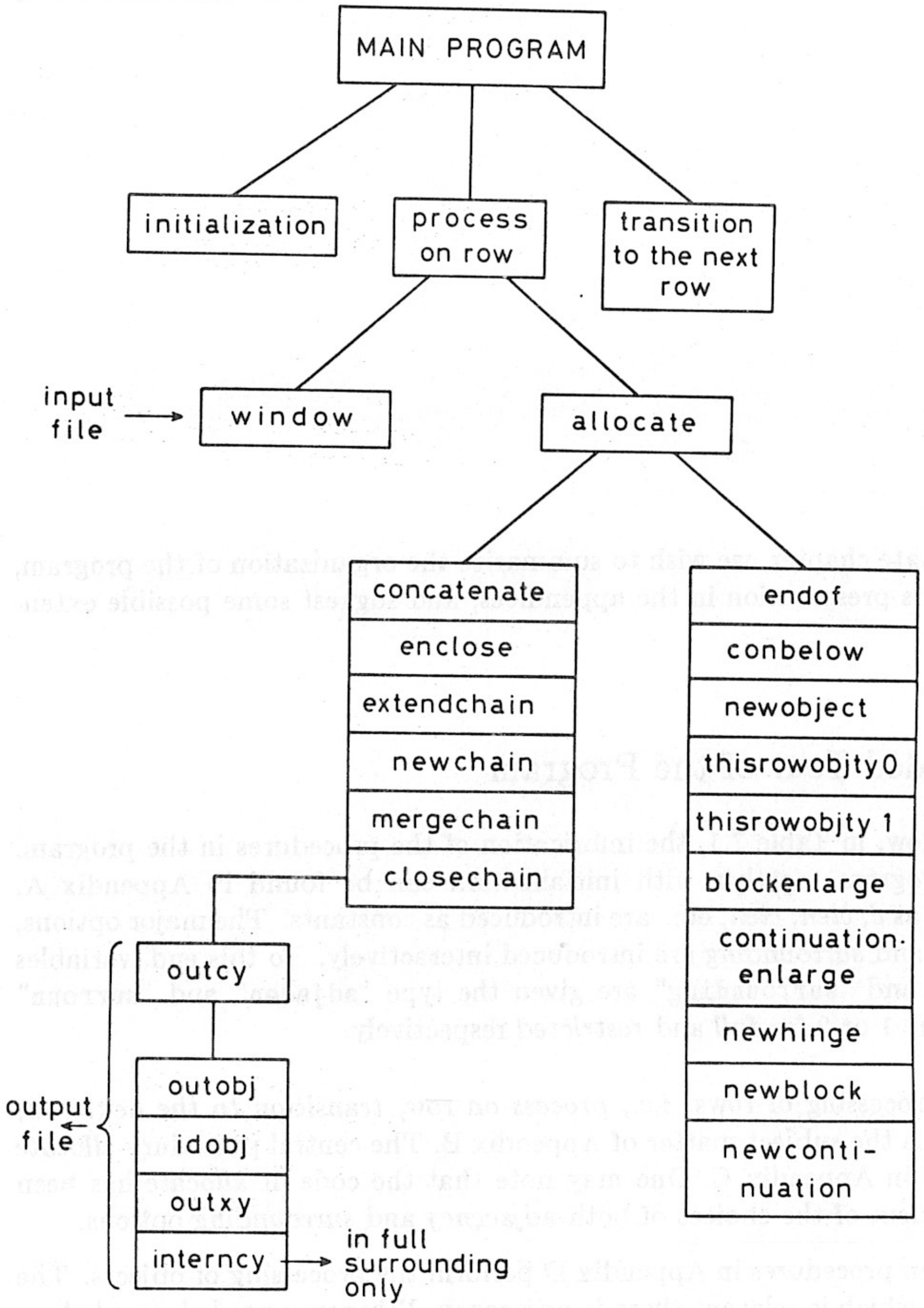

Table 7.1. **Organization of the program.**

and *newboject*), the distinction between *full* and *restricted* is realized by `CASE` statements. A glance at these procedures reveals that it should be straightforward to re-write specialized versions.

The six procedures performing the processing of cycles are collected in Appendix E. The major option there is *surrounding*. Two of these procedures (*concatenate*, and *enclose*) differ significantly according to whether *full* or *restricted surrounding* is chosen. Therefore, these procedures are presented in two versions numbered 1 and 2. In the other four procedures, the distinction is again realized by `CASE` statements.

Appendix F contains output procedures two of which (*outcy, outobj*) are also going by numbered versions according to the surrounding option chosen.

The code is sprinkled with brief `{comments}`. When necessary, longer comments are collected in subsections or deferred to the end of the appendix. Comment #C.1 is referred to, in the code, by the notation `[C.1]`.

The program was tested successfully, under each option, on a variety of data—ranging from good quality characters acquired by a facsimile reader to artificial images containing highly intertwinned, sophisticated components—on the VAX 11/780 Computer at the Philips Research Laboratory, Brussels.

§7.2 Some Possible Extensions

As we already mentioned (see Chapter 1), there are essentially two approches to real-time detection of connected components. The first is based on border finding techniques and is well exemplified by Cederberg's algorithm, [1]. Cederberg's algorithm produces a description of connected components coded in the so-called raster-scan chain-code, (a variation of Freeman's chain code) which is appropriate for data-compression purposes, but less effective for feature extraction purposes. Incidentaly, let us note that, aside from differences in component descriptions, Cederberg's program corresponds, roughly speaking, to what we have called *restricted adjacency* and *full surrounding*.

For most other purposes, especially for feature extraction purposes, the second approach based on runs which was adopted here seems to be the best choice [2, p. 348]. Clearly, it would be a simple matter to modify the program in order to extract, in real-time, such features as component perimeter and area which are of importance in visual inspection tasks. Likewise, component moments could be readily computed from the entries in *objrec* records. More importantly, the decomposition of connected components into objects, *hinges, blocks, d-blocks,* and *block-continuations* can be used to lay bare many of the structural features of the components in the figure. In this respect, the best example that may come to mind is probably Optical Character Recognition, and it is worthwhile to briefly turn our attention to it.

In a review of character parameterization techniques which have proved useful for character recognition, Ullmann cites [4]:

- the lengths or numbers of runs per row in a character;
- a strip-classification code number per row that depends only on the intersection of the character with the row;
- character profiles, and measurements derived from profiles;
- number of holes, limb ends, relative position of limb ends, etc.

There should be no need to insist that such features could be readily extracted from the output of our program, or that our approach could specialized to extract several of these in real-time. Moreover, it goes without saying that there is only a small number of possible ways to describe the structure of a given character in terms of the objects it is made of. Such structural descriptions could, in turn, be represented as strings, or trees, or graphs of pattern building-blocks and their relationships, thereby opening the way to syntactic methods of pattern recognition. These structural descriptions could be further combined with mathematical or statistical object features. This would give rise to a *hybrid* approach to pattern recognition, very much in the style of methods based on attributed grammars [3].

REFERENCES

[1] R. Cederberg, *On the Coding, Processing, and Display of Binary Images*, Ph.D. Dissertation No. 57, Linköping Univ., Dept. Electr. Engrg., Linköping, Sweden, 1980.

[2] A. Rosenfeld, and A.C. Kak, *Digital Picture Processing* New York: Academic Press, 1976.

[3] W-H. Tsai, and K.S. Fu, "Attributed grammars—A tool for combining syntactic and statistical approaches to pattern recognition", *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-10, pp. 873–885, Dec. 1980.

[4] J.R. Ullmann, "Advances in character recognition", in *Applications of Pattern Recognition,* K.S. Fu Ed., Boca Raton Fl.: CRC Press, Inc., 1982, pp. 197–236.