Flocoder

D. Morris, T. G. Kennedy, and L. Last

Department of Computer Science, University of Manchester

Flocoder is a system for designing, documenting and generating programs, using flow-charts. A file of flow-chart descriptions is set up, and the system may be called to draw the charts on a graph-plotter. The chart descriptions can be edited and the diagrams redrawn as necessary. When the descriptions contain sufficient detail the system may be called to generate the program.

(Received January 1971)

The Flocoder system has been developed as a tool for 'engineering' software. In particular it is being used for the design and production of the operating system and some compilers of MU5. Its principal merit is that the software produced this way is readable throughout all stages of its development. By readable is meant that for a given piece of software:

1. Its designers are always able to quickly aquaint themselves with the detail of the implementation.
2. The coders of one section can easily check details in another section.
3. It can be understood by an outsider in the order of a day or two.

In contrast, high level languages, being primarily designed for good 'writeability', can be difficult to read if used 'raw'.

Following engineering precedent the design of a piece of software in the Flocoder system is expressed diagrammatically. These diagrams (or flow-charts) would have a hierarchical structure. Thus, the more complicated boxes on a flow-chart would themselves be described by flow-charts. For the other simpler boxes a translation in program text is given. Flocoder uses this heirarchical description in order to generate the program.

The system is intended for use with ALGOL-like languages, based on the program model shown in **Fig. 1.** This starts with a heading, followed by declarations, possibly including procedures, followed by an arbitrary arrangement of statements, blocks and/or compound statements, finishing with an 'END'. Inside each procedure or block, the pattern may repeat recursively, and, in general, each procedure, block and compound statement will be described by a separate flow-chart. The distinguishing feature of a compound statement is that it may have multiple entry and exit points, since it contains no declaratives.

Flow-chart description

A flow-chart consists of a number of standard symbols, here referred to as 'boxes', arranged in columns and rows, with flow-lines between them. Each 'box' describes one stage in the process.

There are three sections in the description. The first defines each box individually by allocating to it a number, stating what symbol it represents, and listing the text to appear inside it. If it is a test box, captions to appear on the exits may be specified. There are five types of box available—caption, null, circle, rectangle and test. The caption box has no lines around it, and is mainly used to add comments to the chart; the null box provides a method of improving the layout and controlling the paths taken by flow-lines. The remaining three are used for connectors, computational sequences and branch instructions respectively.

In order to describe the positions of the boxes relative to each other, columns and rows are defined, together with a list of the numbers of the boxes contained therein. The third section of the chart description defines the flow-lines between the boxes.

Each flow-chart description commences with a title statement, of form TITLE followed by the user's title for the chart, and ends with the word END. As an example of the use of notation, the encoding for the flow-chart in **Fig. 2** is given in **Appendix 1.**

Code description

After the flow-chart has been defined, a translation rule for each box is given, except for caption and null boxes which have graphic significance only. The translation rule may define a box in one of three ways:

(*a*) as a piece of program text, e.g.
BOX 3 := **for** *i* := 1 **step** 1 **until** *n* **do** *a*[*i*] := 0;

(*b*) equivalent to, i.e. having the same translation as another box on the same diagram, e.g.
BOX 6 = BOX 2

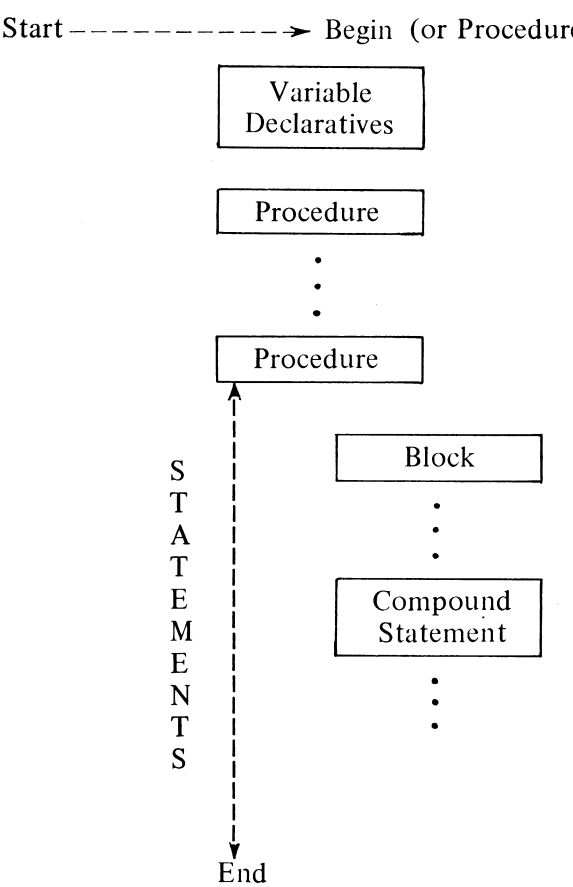

Fig. 1. Model of program

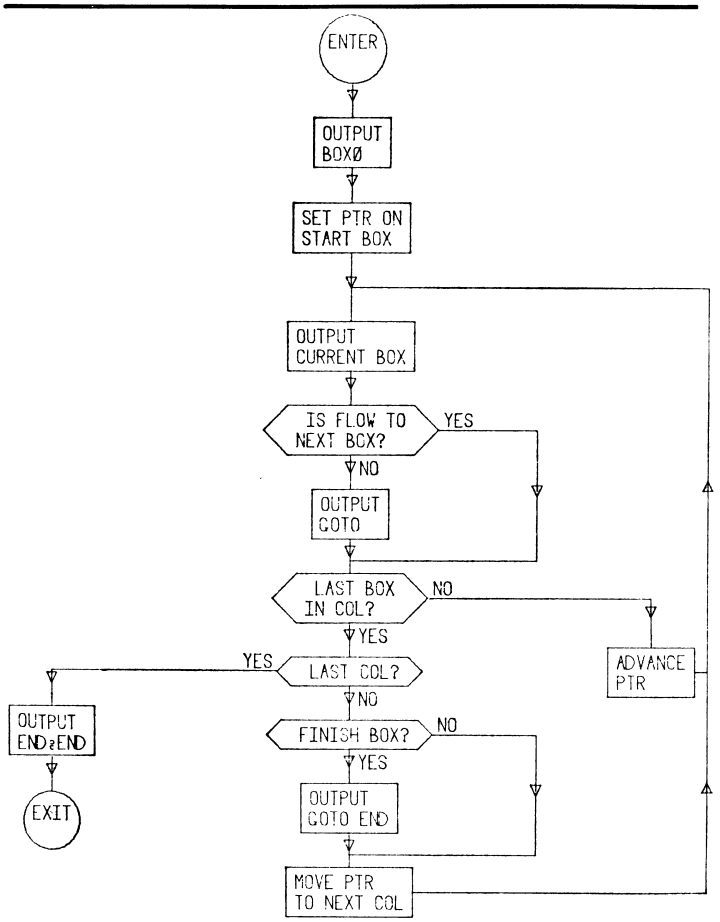

Fig. 2. Code generation algorithm

(*c*) equivalent to a label, i.e. a connector, e.g.
BOX 9 = LB/FAULTEXIT
where 'FAULTEXIT' is the label.

In addition, the starting and finishing points of the code must be defined thus:

BOX 1 = START
BOX 27 = FINISH

The starting box must be at the head of a column and the finishing box at the end of a column.

When the translation of a box is defined by a piece of text, any line in the text beginning with a 'π' is treated specially. The rest of the line is assumed to be the title of another flow-chart whose translation is to be inserted at that point.

Box 0 is reserved for a special function. It is used to hold the heading of the procedure, block or compound statement. Its contents are always output first.

Generation of code

At the head of the program the translation rule for Box 0 is applied. Normally Box 0 will be defined as a piece of program text starting with a procedure heading or BEGIN. Except in the case of a compound statement, some declarations would follow. Procedures defined by other flow-charts may be inserted at this point by means of the 'π' facility mentioned above.

The rest of the code is generated by working through the definitions a column at a time beginning with the column headed by the 'start' box. GOTO statements are inserted as appropriate. These occur mainly at the ends of columns which have flow out of them. In particular, if the finish box is encountered before the last column is output a GOTO END is generated. All boxes which are referenced by GOTO statements will be labelled by the system.

The system assumes that application of the translation rule for a test box will generate code terminating with a conditional GOTO to which a label must be appended. If one of the flow-lines from the test box is to the next box in the column then the label will be assigned on the other path. Otherwise it will be assigned to the flow-path defined last and an extra GOTO will be inserted to the other (first defined) path. All labels generated by the system are formed from the letters 'LB' followed by decimal digits representing a unique number. Thus the user should avoid introducing names which have this structure.

The code generation algorithm is further described by the flow-charts comprising **Fig. 2** and **Fig. 3.**

Conclusions

In its present form the system can be used to draw flow-charts and generate programs. The main part is written in ALGOL. It has been used to generate itself. The flow-charts are actually drawn by a Calcomp plotter attached to a PDP8. Paper tape output produced by the ALGOL program is used to steer the plotter.

Although in its present form the system is adequate and its inherent simplicity is a boon, development is inevitable. Direct control of the plotter from ALGOL code procedures would be advantageous, but the most exciting prospects lie with the use of on-line graphics terminals for creating and interrogating the flow-chart structure.

Acknowledgements

The Flocoder system has been developed entirely by final year undergraduate students in the Department of Computer Science of the University of Manchester. Major contributions have been made by the following earlier participants: R. J. Collins, H. D. Ellison, T. Mott, and R. Phillips. We are also indebted to our colleagues whose assistance has been freely given.

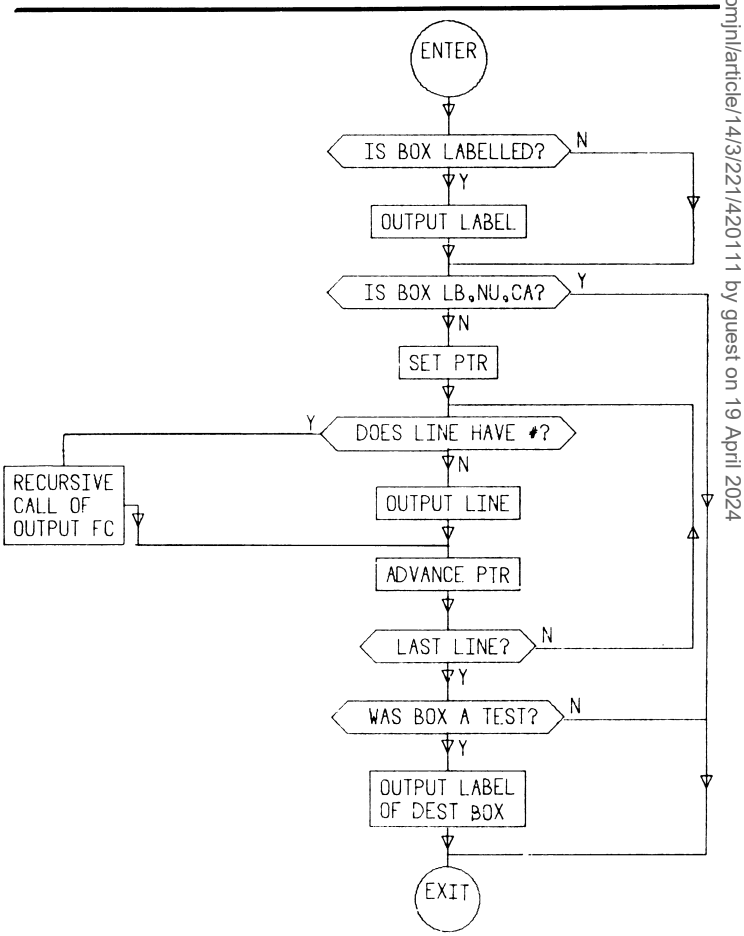

Fig. 3. Apply translation rule for box

Appendix 1

```
TITLE FIG II
0 CA FIG II  CODE GENERATION ALGORITHM
1 CI  ENTER
2 RE OUTPUT | BOX0
3 RE SET PTR ON | START BOX
4 RE OUTPUT | CURRENT BOX
5 TE | 6-NO | | 7-YES | IS FLOW TO | NEXT BOX?
6 RE OUTPUT | GOTO
7 TE | 8-YES | | 14-NO | LAST BOX | IN COL?
8 TE | 12-YES | | 9-NO | LAST COL?
9 TE | 10-YES | | 11-NO | FINISH BOX?
10 RE OUTPUT | GOTO END
11 RE MOVE PTR | TO NEXT COL
12 RE OUTPUT | END : END
13 CI  EXIT
14 RE ADVANCE | PTR
15 NU
16 NU
17 NU

0 COL 12.13
1 COL 1.2.3.15.4.5.6.7.8.9.10.11.0
2 COL 16.14.17

0 ROW 15.16
1 ROW 8.14
2 ROW 12.9
3 ROW 11.17

FLOW 1–2–3–15–4–5–6–7–8–9–10–11–17–16–15
FLOW 5–7–14–16
FLOW 8–12–13
FLOW 9–11

END
```