

CONTOUR

A method of preparing structured flowcharts

James F. Gimpel
Sperry Univac, Blue Bell, Pa.

Former address: Bell Labs, Holmdel, N.J.

Contour is a program whose purpose is to graphically illustrate a program's structure. It operates by bounding the scope of loops and conditionals by solid (or nearly solid) lines. When compound statements are embedded in other compound statements, one obtains, rather than confusion, a rather pleasant display reminiscent of the contour lines of a topographical map.

Aside from its visual appeal, the method has the advantage that it makes far fewer demands on the reader's linguistic expertise and so may be used for presenting algorithms in an almost language-independent manner (a kind of structured flowchart).

The general notion of a structured flowchart is usually attributed to Nassi and Schneiderman [1] and has been further discussed by Grouse [2]. Roy and St. Denis [3] report on their experiences at automatically converting programs into the Nassi-Schneiderman format. The effort described here differs in several respects. The format used requires less horizontal display space and this is critical in producing a practical, generally usable formatter. Also, control clauses become embedded within the flow lines to more clearly delineate the scope of while's, case's, etc.

Figure 1 shows a Pascal program taken from [3]. The result of passing this program through Contour is shown in Figure 2. Note that Begin and End keywords replaced by contour lines are dropped and any semicolon whose purpose is to separate an enclosure from its following neighbor is also dropped.

Contour was originally written to display C programs [4]. It was modified somewhat (for the purpose of this article) to accomodate Pascal programs. The Pascal program in Figure 1 was rewritten in C and is shown in Figure 3; its contour form is shown in Figure 4.

Note that although the two programs appear to be quite different when viewing just their original forms (Figures 1 and 3) they

```

1      (* input conversion of number *)
2  var  x:char;
3      sign,int,exp,j:integer;
4      number,fraction:real;
5      types:(error,fixed,float);
6
7  begin read(x);
8      repeat
9          int:=0;
10         if x='+' then begin sign:=1; read(x); end
11             else if x='-' then begin sign:=-1; read(x); end
12                 else sign:=1;
13         while (x>='0')and(x<='9') do begin (* integer part *)
14             int:=int*10+(ord(x)-ord('0'));
15             read(x);
16             end;
17         fraction:=0; j:=10; int:=int*sign;
18         if x='.' then begin (* fraction part *)
19             read(x);
20             while (x>='0')and(x<='9') do begin
21                 fraction:=fraction+(ord(x)-ord('0'))/j;
22                 j:=j*10; read(x);
23                 end;
24             types:=float;
25             end
26             else types:=fixed;
27         number:=int+sign*fraction; exp:=0;
28         if x='e' then begin (* exponent part *)
29             read(x);
30             if x='+' then begin sign:=1; read(x); end
31                 else if x='-' then begin sign:=-1; read(x); end
32                     else sign:=1;
33             while (x>='0')and(x<='9') do begin
34                 exp:=exp*10+(ord(x)-ord('0'));
35                 read(x);
36                 end;
37             j:=exp*sign; number:=number*power(10,j); types:=float;
38             end;
39         if (x ~=' ') and (x ~='eol') then types:=error;
40         while (((x=' ')or(x=eol))and(not eof(input))) do read(x);
41         case types of
42             error: write(' error in number', eol);
43             fixed: write(' ',int,eol);
44             float: write(' ',number,eol);
45             end;
46     until eof(input);

```

Figure 1

A Pascal program to convert character input to number.

```

1      (* input conversion of number *)
2  var  x:char;
3      sign,int,exp,j:integer;
4      number,fraction:real;
5      types:(error,fixed,float);
6
7  begin read(x);
8      repeat
9          int:=0;
10         if x='+' then
11             sign:=1; read(x);
12         else if x='-' then
13             sign:=-1; read(x);
14         else
15             sign:=1;
16
17         while (x>='0') and (x<='9') do
18             (* integer part *)
19             int:=int*10+(ord(x)-ord('0'));
20             read(x);
21
22         fraction:=0; j:=10; int:=int*sign;
23         if x='.' then
24             (* fraction part *)
25             read(x);
26             while (x>='0') and (x<='9') do
27                 fraction:=fraction+(ord(x)-ord('0'))/j;
28                 j:=j*10; read(x);
29
30         types:=float;
31         else
32             types:=fixed;
33
34         number:=int+sign*fraction; exp:=0;
35         if x='e' then
36             (* exponent part *)
37             read(x);
38             if x='+' then
39                 sign:=1; read(x);
40             else if x='-' then
41                 sign:=-1; read(x);
42             else
43                 sign:=1;
44
45             while (x>='0') and (x<='9') do
46                 exp:=exp*10+(ord(x)-ord('0'));
47                 read(x);
48
49             j:=exp*sign; number:=number*power(10,j); types:=float;
50
51         if (x <> ' ') and (x <> eol) then
52             types:=error;
53
54         while ((x=' ') or (x=eol)) and (not eof(input)) do
55             read(x);
56
57         case types of
58             error: write(' error in number', eol);
59             fixed: write(' ',int,eol);
60             float: write(' ',number,eol);
61
62     ;
63 until eof(input);

```

Figure 2

A contourized version of Figure 1.

```

1          /* input conversion of number */
2  char x;
3  int sign, Int, exp, j;
4  float number, fraction;
5  enum {ERROR, FIXED, FLOAT} types;
6
7  x = read();
8  do {
9      Int = 0;
10     if (x == '+') { sign = 1; x = read(); }
11     else if (x == '-') { sign = -1; x = read(); }
12     else sign = 1;
13     while (x >= '0' && x <= '9') { /* integer part */
14         Int = Int * 10 + x - '0';
15         x = read();
16     }
17     fraction = 0; j = 10; Int = Int * sign;
18     if(x == '.') { /* fraction part */
19         x = read();
20         while( x >= '0' && x <= '9') {
21             fraction = fraction + (float) (x - '0') / j;
22             j = j * 10; x = read();
23         }
24         types = FLOAT;
25     }
26     else types = FIXED;
27     number = Int + sign * fraction; exp = 0;
28     if( x == 'e') { /* exponent part */
29         x = read();
30         if(x == '+') { sign = 1; x = read(); }
31         else if (x == '-') { sign = -1; x = read(); }
32         else sign = 1;
33         while ( x >= '0' && x <= '9') {
34             exp = exp * 10 + x - '0';
35             x = read();
36         }
37         j = exp * sign; number = number*power(10,j); types = FLOAT;
38     }
39     if(x != ' ' && x != '\n') types = ERROR;
40     while ( (x==' ' || x == '\n') && !feof(stdin)) x = read();
41     switch(types) {
42         case ERROR: printf(" error in number\n"); break;
43         case FIXED: printf(" %d\n", Int); break;
44         case FLOAT: printf(" %f\n", number); break;
45     }
46 } while( !feof(stdin) );

```

Figure 3

A C version of the program in Figure 1.

```

1                                     /* input conversion of number */
2 char x;
3 int sign, Int, exp, j;
4 float number, fraction;
5 enum {ERROR, FIXED, FLOAT} types;
6
7 x = read();
8 do
9 {
10     Int = 0;
11     if (x == '+')
12     {
13         sign = 1; x = read();
14     }
15     else if (x == '-')
16     {
17         sign = -1; x = read();
18     }
19     else
20     {
21         sign = 1;
22     }
23
24     while (x >= '0' && x <= '9')
25     {
26         /* integer part */
27         Int = Int * 10 + x - '0';
28         x = read();
29     }
30
31     fraction = 0; j = 10; Int = Int * sign;
32     if (x == '.')
33     {
34         /* fraction part */
35         x = read();
36         while (x >= '0' && x <= '9')
37         {
38             fraction = fraction + (float) (x - '0') / j;
39             j = j * 10; x = read();
40         }
41     }
42
43     types = FLOAT;
44     else
45     {
46         types = FIXED;
47     }
48
49     number = Int + sign * fraction; exp = 0;
50     if (x == 'e')
51     {
52         /* exponent part */
53         x = read();
54         if (x == '+')
55         {
56             sign = 1; x = read();
57         }
58         else if (x == '-')
59         {
60             sign = -1; x = read();
61         }
62         else
63         {
64             sign = 1;
65         }
66
67         while (x >= '0' && x <= '9')
68         {
69             exp = exp * 10 + x - '0';
70             x = read();
71         }
72
73         j = exp * sign; number = number * power(10, j); types = FLOAT;
74     }
75
76     if (x != ' ' && x != '\n')
77     {
78         types = ERROR;
79     }
80
81     while ( (x == ' ' || x == '\n') && !feof(stdin) )
82     {
83         x = read();
84     }
85
86     switch (types)
87     {
88     case ERROR: printf(" error in number\n"); break;
89     case FIXED: printf(" %d\n", Int); break;
90     case FLOAT: printf(" %f\n", number); break;
91     }
92 } while ( !feof(stdin) )

```

Figure 4

A contourized version of Figure 3.

actually are quite similar as a comparison of Figures 2 and 4 reveals.

Further Notes

- It is, of course, possible to produce 'contour lines' with characters chosen from the standard Ascii set (such as vertical bar and the underscore). The result is generally unpleasant. Contour happens to use an enhanced character set consisting of the four "corner" characters (┌, ┐, └, ┘) and the horizontal line (─) that links them (as opposed to the minus (-) that does not). These seem important to the point of being critical for the application. Note that the standard vertical bar links with the corner characters.
- It was felt that a special case should be made of constructs of the form: `if() ... else if() ... else if() else ...` since this is often used as a multi-way decision and is not appropriately rendered as a nested sequence. Rather, a ladder-like structure is employed as is shown in Figures 2 and 4.
- The author has used Contour for quite some time in the complete absence of other forms of listings; this was only practicable, however, after Contour was modified to prepend line numbers (where appropriate) to output lines. Note that the line numbers correspond to the physical line number of the file rather than the line number in the listing.
- Information that is not comprehended by contour is simply displayed as it is found. This practice greatly reduces user frustration in the case of pre-processor usage compromising the basic syntax of C.

Implementation

A line is read from the input stream and scanned to see if it starts a special statement form; if so, an attempt is made to read lines until either an entire statement is matched or until some preset limit is reached (experience indicates that about 200 lines is adequate). If a match cannot be made, we simply continue processing as if the special form were not recognized and with the input stream extracting lines from those that had been buffered before proceeding with the rest of the file. If a full statement was recognized then one or more recursive calls is made with the "input stream" set equal to a substatement of the statement recognized. For example if the statement recognized is of the form `if b then s1 else s2 then` recursive calls are made with the

stream set equal to s1 and s2. The result of the recursive call is to convert a stream into a formatted block of characters (where block has the meaning described in [5]). The caller must then format the result in accordance with the formatting conventions desired. For example, if the statement were

if b then s1 else s2

the caller is in a position where s1 and s2 have been formatted into a block of characters of known width and height. He then needs to surround these with appropriate contour lines and embed the if clause to produce a returnable object.

The program was written in SNOBOL4 [6] for convenience.

REFERENCES

- [1] Nassi I. and Shneiderman, B. Flowchart techniques for structured programming. Sigplan Notices, 8:8 (Aug. 1973), 12-26.
- [2] Grouse, P. FLCWBLOCKS -- A technique for structured programming. Sigplan Notices, 13:2 (Feb. 1978), 46-56.
- [3] Roy, P. and St. Denis, R. Linear flowchart generator for a structured language. Sigplan Notices, 11:11 (Nov. 1976), 58-64.
- [4] Kernighan, B. and Ritchie, D. The C Programming Language. Prentice-Hall, Englewood Cliffs, N. J., 1978.
- [5] Gimpel, J. F. Blocks -- a new datatype for SNOBOL4. CACM, 15:6 (June 1972), 438-447.
- [6] Griswold, R. E., Poage, J. F. and Polonsky, I. P. The SNOBOL4 Programming Language. Prentice-Hall, Englewood Cliffs, N. J., 1971.