

Guide to

Design structure diagrams for use in program design and other logic applications

UDC 681.3.06:003.63

NO COPYING WITHOUT BSI PERMISSION EXCEPT AS PERMITTED BY COPYRIGHT LAW



Document Sharing Hub

Committees responsible for this British Standard

The preparation of this British Standard was entrusted by the Information Systems Technology Standards Committee (IST/-) to Technical Committee IST/15 upon which the following bodies were represented:

- Association of Computer Units in Colleges of Higher Education (ACUCHE)
- British Computer Society
- British Telecommunications plc
- Business Equipment Trade Association
- Computing Services Association
- Department of Trade and Industry (National Physical Laboratory)
- Electronic Engineering Association
- H M Treasury (Central Computer and Telecommunications Agency)
- Institute of Data Processing Management
- Institution of Electrical Engineers
- LAMSAC
- Ministry of Defence
- National Computing Centre Ltd.

This British Standard, having been prepared under the direction of the Information Systems Technology Standards Committee, was published under the authority of the Board of BSI and comes into effect on 31 December 1987

© BSI 06-1999

First published May 1982
First revision December 1987

The following BSI references relate to the work on this standard:
Committee reference IST/15
Draft for comment 86/60270 DC

ISBN 0 580 16133 1

Amendments issued since publication

Amd. No.	Date of issue	Comments

Contents

	Page
Committees responsible	Inside front cover
Foreword	iv
1 Scope	1
2 Definitions	1
3 Conventions	1
3.1 Flow of control	1
3.2 Connection of blocks	1
3.3 Decision elements	2
3.4 Loops	3
3.5 Case constructs	5
3.6 Designators	6
3.7 GOTO statements	6
3.8 Connectors	7
3.9 Block symbols	7
3.10 Annotation	11
4 Additional constructs for “real time” use	11
4.1 General	11
4.2 Events	11
4.3 Concurrent processing	12
4.4 Message passing	14
5 Data structures	16
5.1 General	16
5.2 Arrays	17
5.3 Records/structures	17
5.4 Pointers	18
6 Program interfaces	19
6.1 General	19
6.2 External interfacing	19
6.3 Local interfacing	20
7 Additional constructs for design management	21
7.1 General	21
7.2 Structure	21
Appendix A Examples of design structure diagrams	25
Appendix B Machine representation	42
Appendix C Stencil design	43
Figure 1	1
Figure 2	2
Figure 3	2
Figure 4	2
Figure 5	3
Figure 6	3
Figure 7	3
Figure 8	4
Figure 9	5
Figure 10	5
Figure 11	6
Figure 12	7
Figure 13	7

	Page
Figure 14	8
Figure 15	8
Figure 16	9
Figure 17	9
Figure 18	10
Figure 19	10
Figure 20	10
Figure 21	11
Figure 22	11
Figure 23	12
Figure 24	12
Figure 25	12
Figure 26	13
Figure 27	13
Figure 28	14
Figure 29	14
Figure 30	14
Figure 31	15
Figure 32	15
Figure 33	16
Figure 34	17
Figure 35	17
Figure 36	18
Figure 37	18
Figure 38	19
Figure 39	20
Figure 40	21
Figure 41	22
Figure 42	23
Figure 43	24
Figure 44	25
Figure 45	26
Figure 46	27
Figure 47	28
Figure 48	28
Figure 49	29
Figure 50	30
Figure 51	31
Figure 52	32
Figure 53	33
Figure 54	33
Figure 55	34
Figure 56	34
Figure 57	34
Figure 58	35
Figure 59	36
Figure 60	37

	Page
Figure 61	39
Figure 62	40
Figure 63	41
Figure 64	42
Figure 65	43
Figure 66	44
Publications referred to	Inside back cover

Foreword

This British Standard has been prepared under the direction of the Information Systems Technology Standards Committee and it supersedes BS 6224:1982, which is withdrawn.

This standard has been revised to clarify and amplify the subject in the light of users' experience. It gives further guidance on use of the basic set of constructs, which remain unchanged, and introduces additional constructs that have been requested by users in order to reflect current practice. In particular, it gives proper facilities for dealing with predefined process and task definitions, it permits the depiction of formal parameters and it introduces new constructs for visibility control (for items declared within nested scopes), program interface and design management.

The charting notations for program constructs used in this standard are in accordance with those used in ISO 8631.

A British Standard does not purport to include all the necessary provisions of a contract. Users of British Standards are responsible for their correct application.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

Summary of pages

This document comprises a front cover, an inside front cover, pages i to iv, pages 1 to 44, an inside back cover and a back cover.

This standard has been updated (see copyright date) and may have had amendments incorporated. This will be indicated in the amendment table on the inside front cover.

1 Scope

This British Standard provides information and guidance on design structure diagrams and in Appendix A and Appendix B gives examples of their use. Appendix C shows a possible stencil design.

Design structure diagrams are a tree-structured graphic representation of constructs that are present in most modern high-level languages, and their use as a program design tool can assist in the production of high-level code and data and encourage the use of structured programming. The resulting program source code will correspond closely to the diagrams, thereby improving the clarity of the design and the standard of documentation.

NOTE The titles of the publications referred to in this standard are listed on the inside back cover.

2 Definitions

For the purposes of this British Standard the following definitions apply.

2.1

node

a branching point for the flow of logic control

2.2

fall-back

the return of control to a previously visited node (see 3.2.3)

2.3

event

a condition that has to occur before a particular process can continue, the timing of the occurrence of the condition not being under direct control of that process, e.g. interrupt, timeout and message transmission

2.4

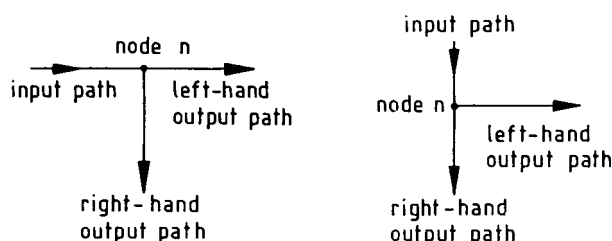
concurrent process (or task)

a particular copy or instance of a previously defined process which, after initialization, is executed independently of the initiating flow of control

3 Conventions

3.1 Flow of control

Each node consists of one input and two output paths. On reaching a node, the flow of control passes first to the left-hand output path (see Figure 1). After execution of the activities in the left-hand output path, the control falls back and continues along the right-hand output path.



NOTE 1 The terms “left” and “right” are used solely with respect to the flow of control and not with respect to the reader.

NOTE 2 Nodes may be annotated in a convenient manner, if desired, to provide diagrammatic references.

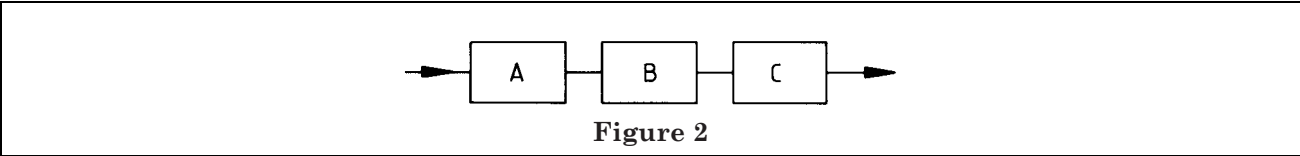
Figure 1

3.2 Connection of blocks

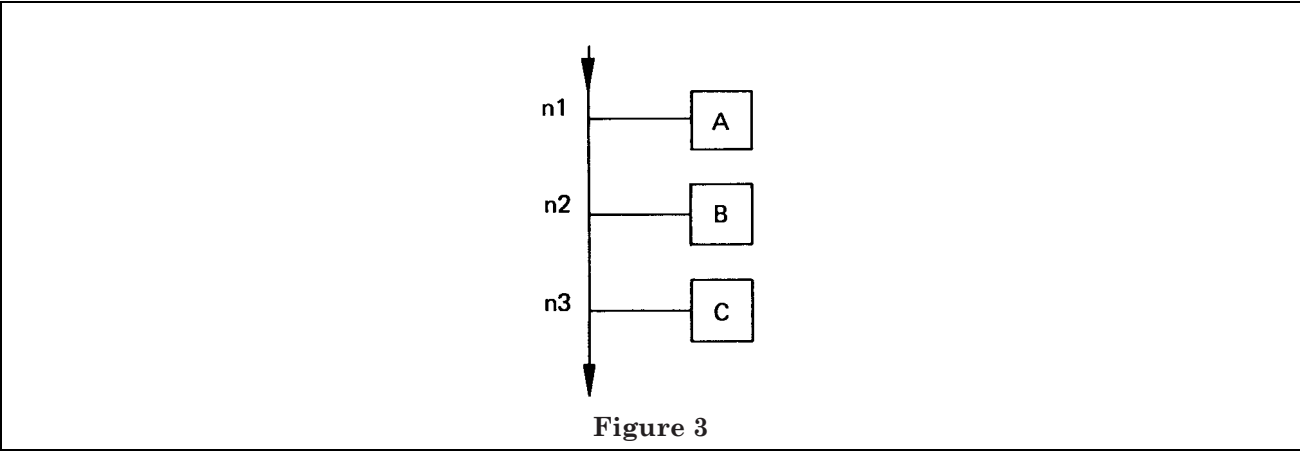
3.2.1 *General.* Blocks are connected using one of the following means:

- horizontal connections (see 3.2.2);
- fall-back representation (see 3.2.3); or
- a combination of horizontal connections and fall-back representation (see 3.2.4).

3.2.2 Horizontal connections. Blocks are connected using the concept of “single input, single output”. The example in Figure 2 represents the sequence A, B, C.



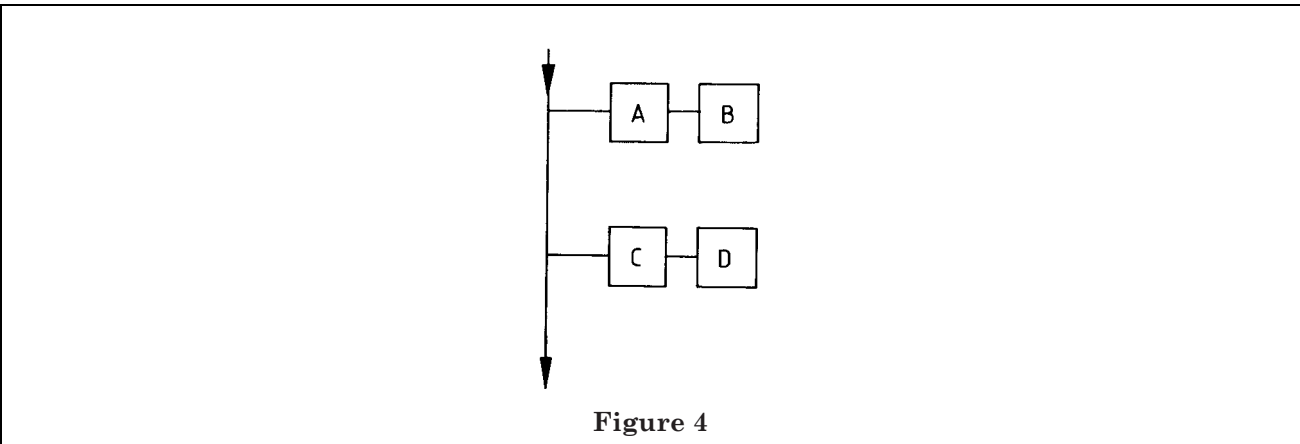
3.2.3 Fall-back representation. Fall-back representation uses the concept of “single input, zero output”. The example in Figure 3 represents the sequence A, B, C.



In Figure 3, the control path firstly takes the left-hand path at the node n1 to arrive at A. After execution, since there is no output path from A, the control falls back to n1 and continues along the right-hand path of n1 to n2 (no executive action takes place during fall-back). Blocks B and C are visited in a similar manner.

The control always falls back to the nearest node with an unvisited right-hand path.

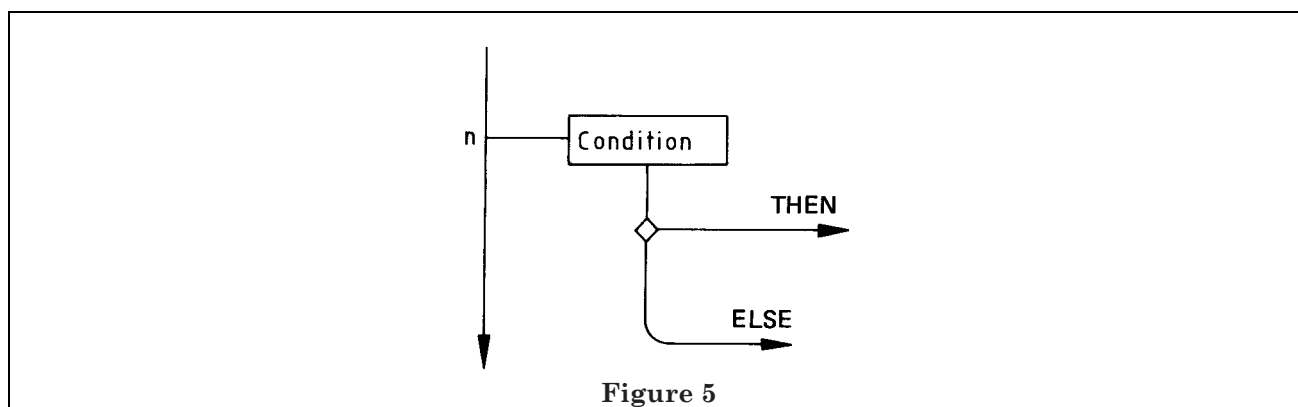
3.2.4 Combinations. Horizontal connections and fall-back representations may be combined giving rise to the concept of “single input, single or zero output”. The example in Figure 4 represents the sequence A, B, C, D.



Further examples of the use of this concept are given in example 1 of Appendix A.

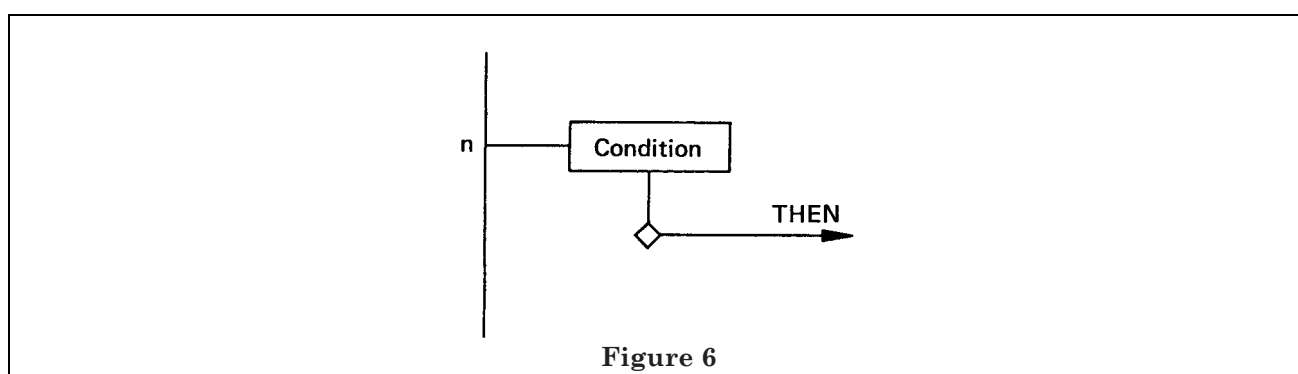
3.3 Decision elements

Decision elements represent the IF-THEN-ELSE construct and are depicted as shown in Figure 5.



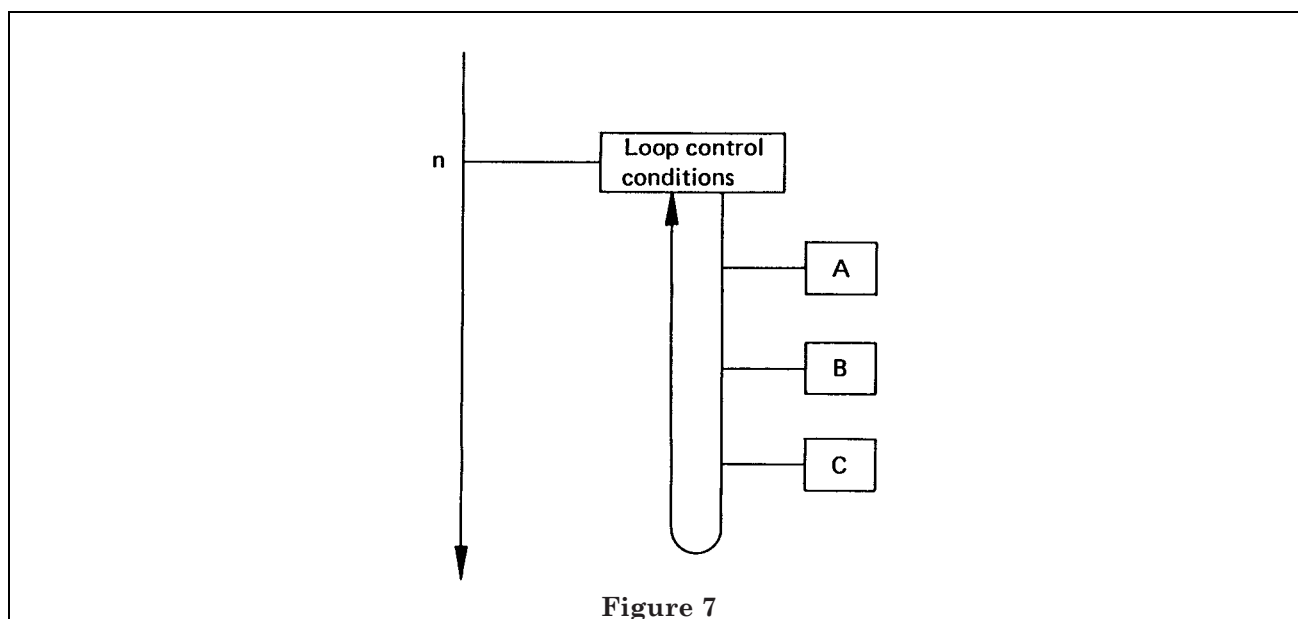
The orientation is fixed and no transposition of THEN and ELSE is allowed. After execution of either the THEN or the ELSE branch, the control falls back to node n.

A restricted form may be used that omits the ELSE branch (see Figure 6).



3.4 Loops

Loops are depicted as shown in Figure 7. As an aid to clarity, the looped line should extend at least as far as the lowest block contained within the loop construct.



Both implicit and explicit loop terminations cause the control point to fall back to node n.

If languages are used that allow specification of the point of evaluation, the graphics shown in Figure 8 should be used as appropriate.

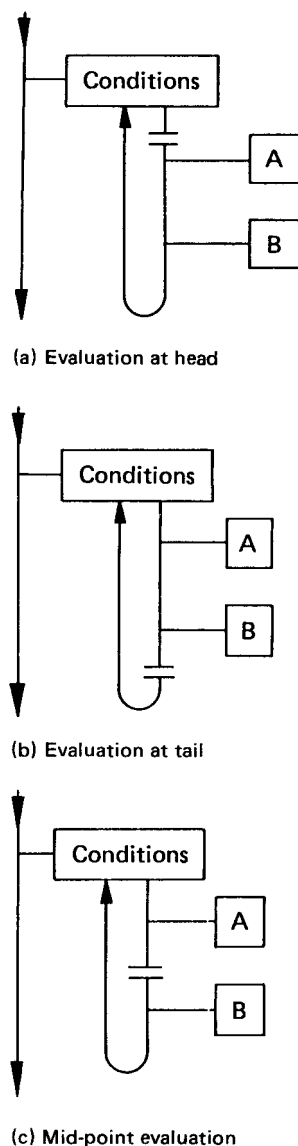


Figure 8

Where no evaluation point is shown, evaluation will either be at the loop head [normal default; see Figure 9(a)] or be null if the conditions specify a continuous loop [i.e. loop forever; see Figure 9(b)].

Execution of the loop can be terminated by a TERMINATION operation placed at one or more points within the loop [see Figure 9(c)]. Execution of the operation will cause the control to fall back to the node just prior to the loop head. A termination operation may be placed within the following inner nested constructs: decision, case, multiple alternative event and loop. Such action will cause these constructs to terminate in an orderly manner (from the inner outwards) until the loop has terminated.

If the operation is within an inner nested loop, it should identify which loop is to be terminated and the notation for this is shown in Figure 9(d).

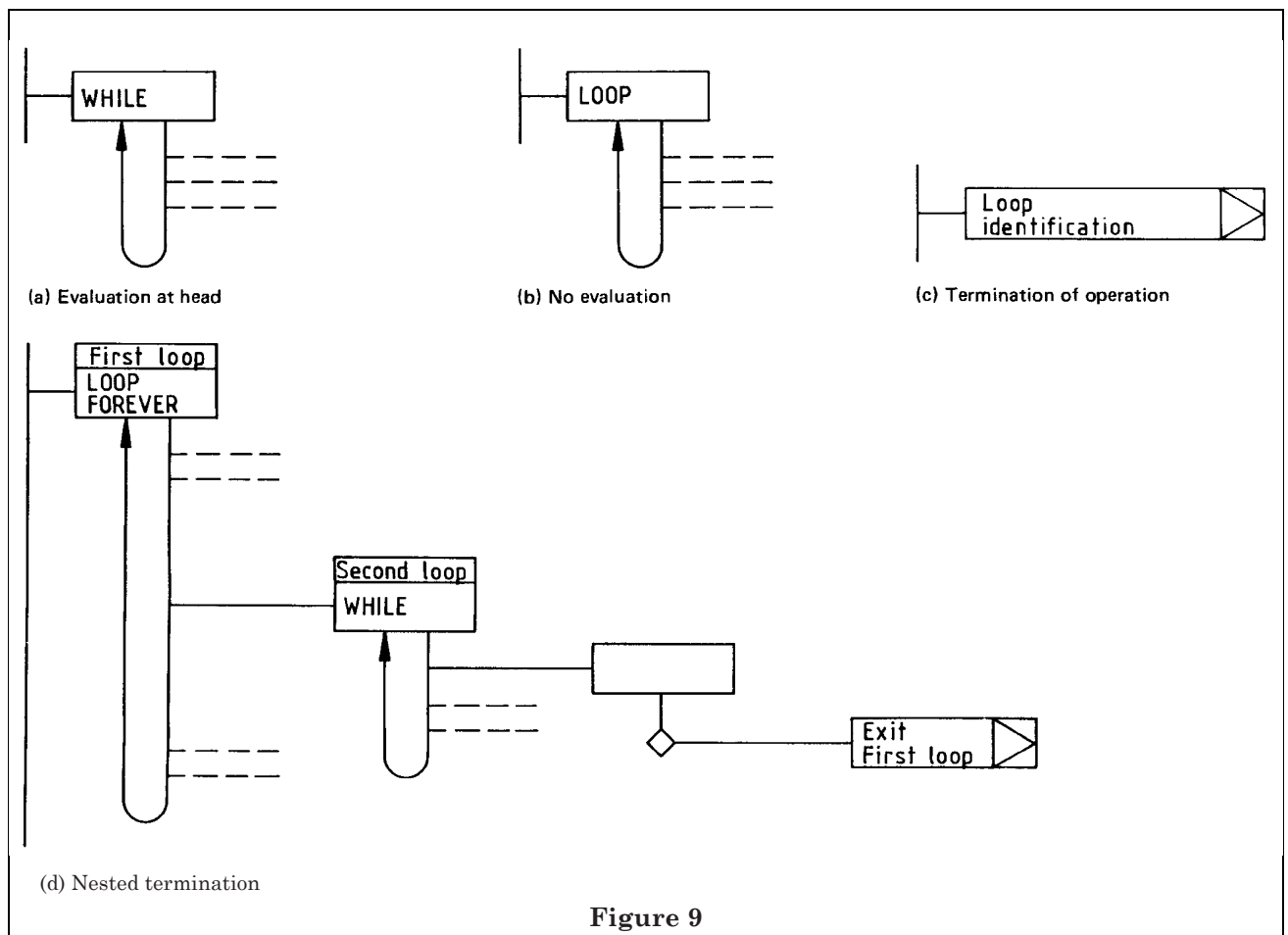
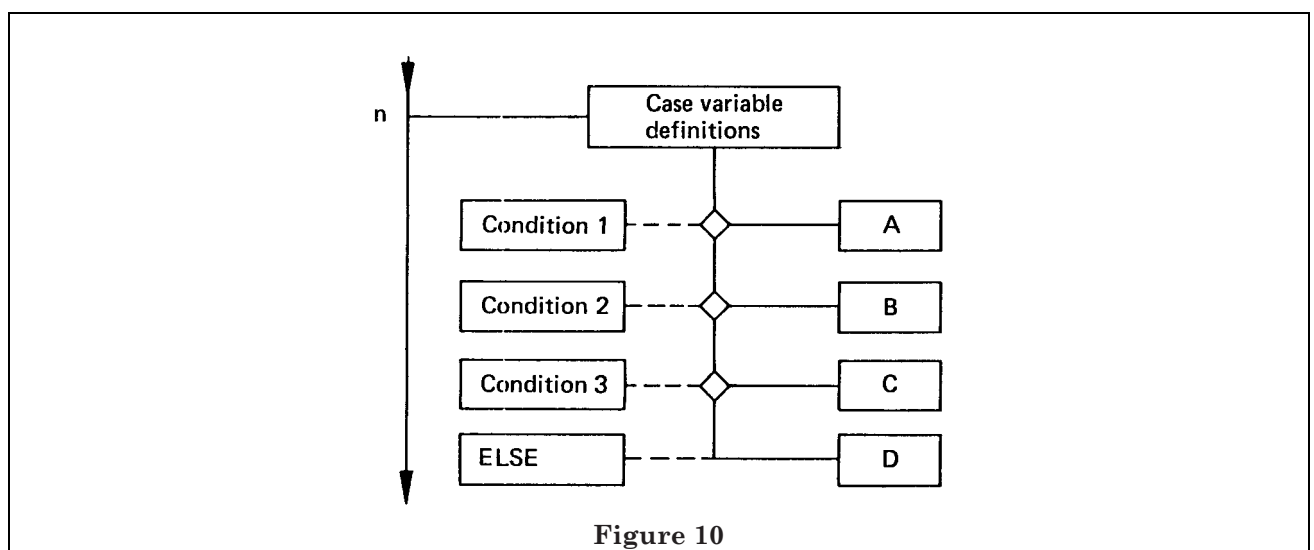


Figure 9

3.5 Case constructs

Case constructs are depicted as shown in Figure 10.



The conditions are evaluated in turn and if one of the conditions is satisfied then the control passes to the designated sequence (e.g. to B for condition 2). After the sequence has been executed, the control falls back to node n.

If none of the conditions are satisfied, the ELSE sequence is executed, after which the control falls back to node n.

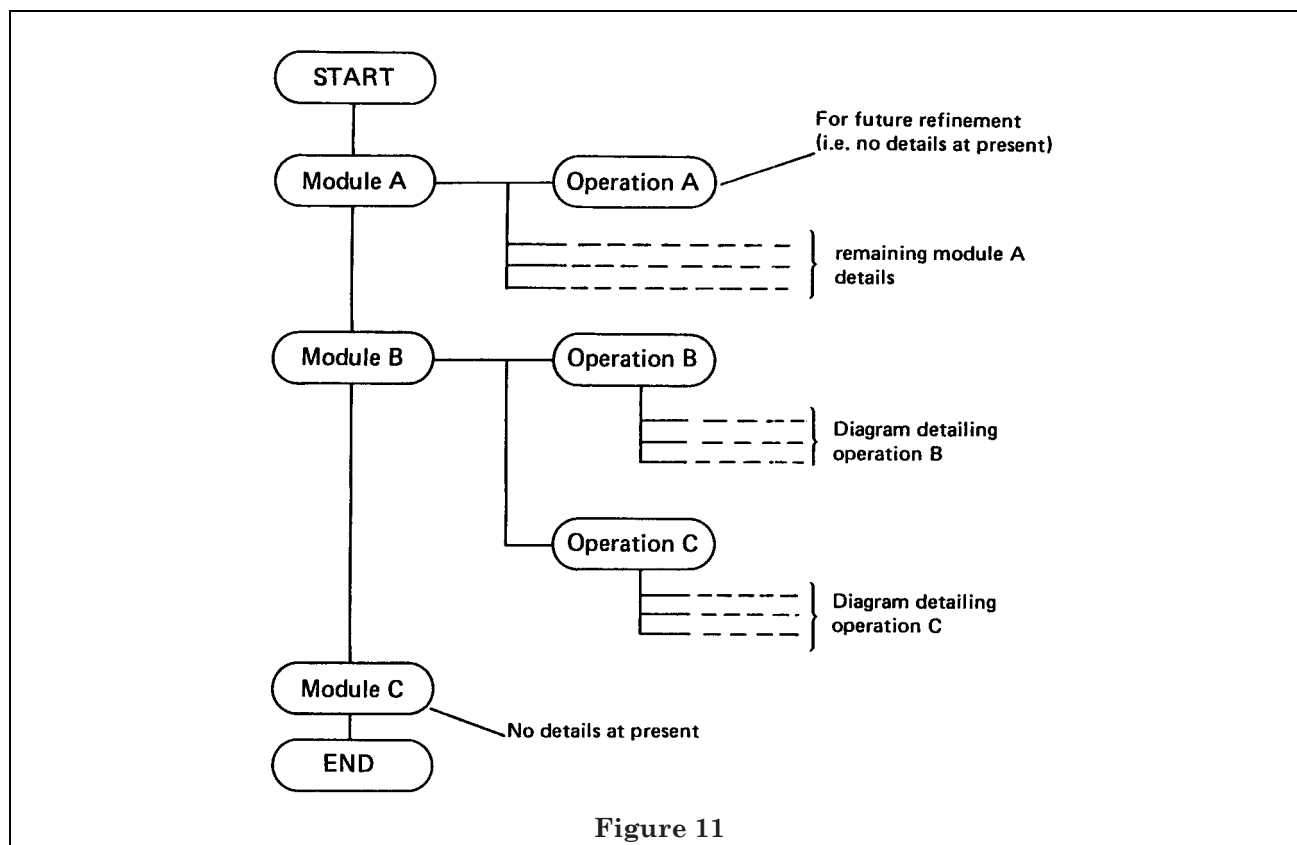
3.6 Designators

Designators may be placed anywhere on the diagram to designate subsections of the program due for further refinement, e.g. blocks, modules, processes and sections.

NOTE 1 Designators are an extension of the BS 4058 terminator symbol.

NOTE 2 The use of designators assists in the “top down” design process.

A general example of the use of designators is shown in Figure 11.



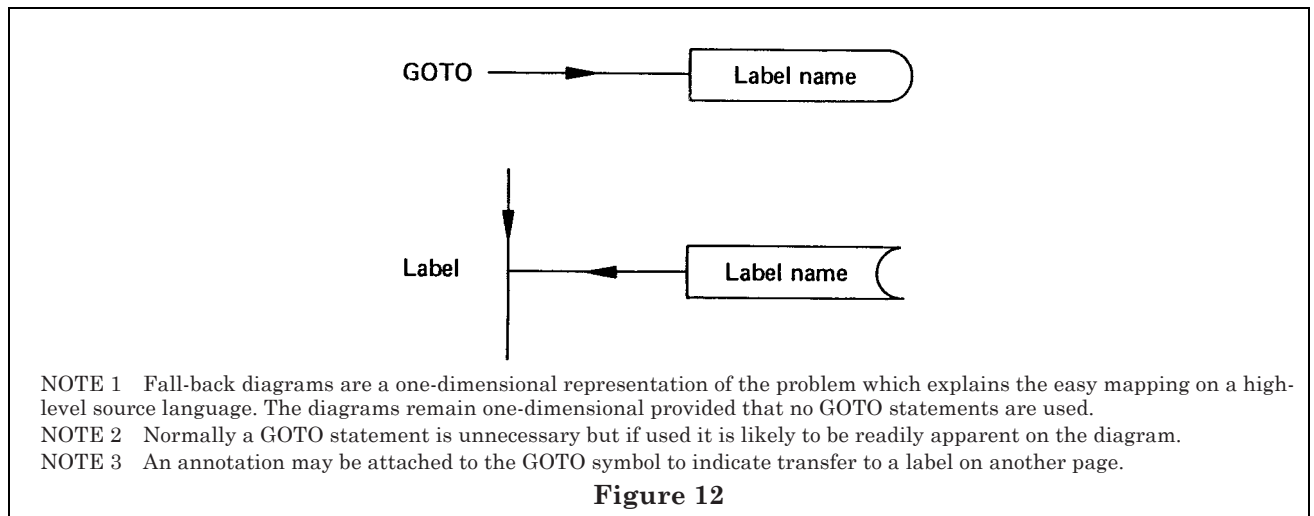
NOTE 3 In practice, meaningful names would be given to the module and operation designators.

NOTE 4 It is assumed in Figure 11 that module C and operation A have not yet been defined.

NOTE 5 The top level of the program structure may be viewed by looking along the vertical line between the start and the finish designators. Lower levels may be viewed by looking from the left.

3.7 GOTO statements

GOTO statements and their corresponding labels introduce more than one route to a particular point and should only be used with discretion. They are depicted as shown in Figure 12.

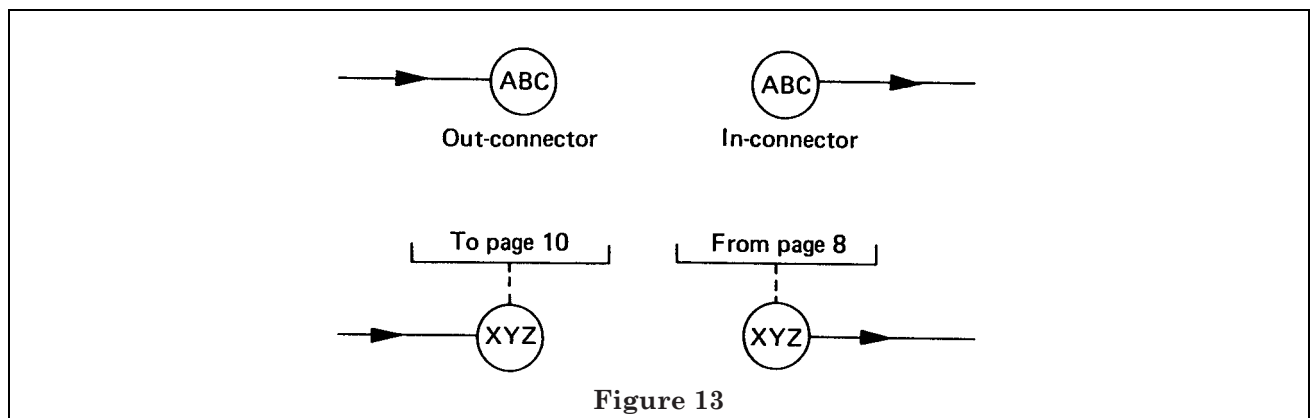


3.8 Connectors

If it becomes necessary to interrupt a line, to preserve documentation clarity or to continue on another page, a connector symbol should be placed at either end of the interruption in the line and uniquely identified by means of an identifier placed in each connector symbol.

Any inter-page references shall be indicated by annotations (see 3.10) attached to each connector.

Connectors are depicted as shown in Figure 13.



3.9 Block symbols

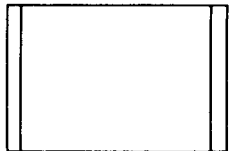
Block symbols should be depicted as shown below and may be named by including an identifier within the symbol.

a) *A process*



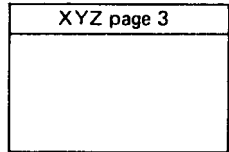
Any kind of processing operation or logical sequence of operations.

b) A predefined process



A call to a procedure or subroutine.

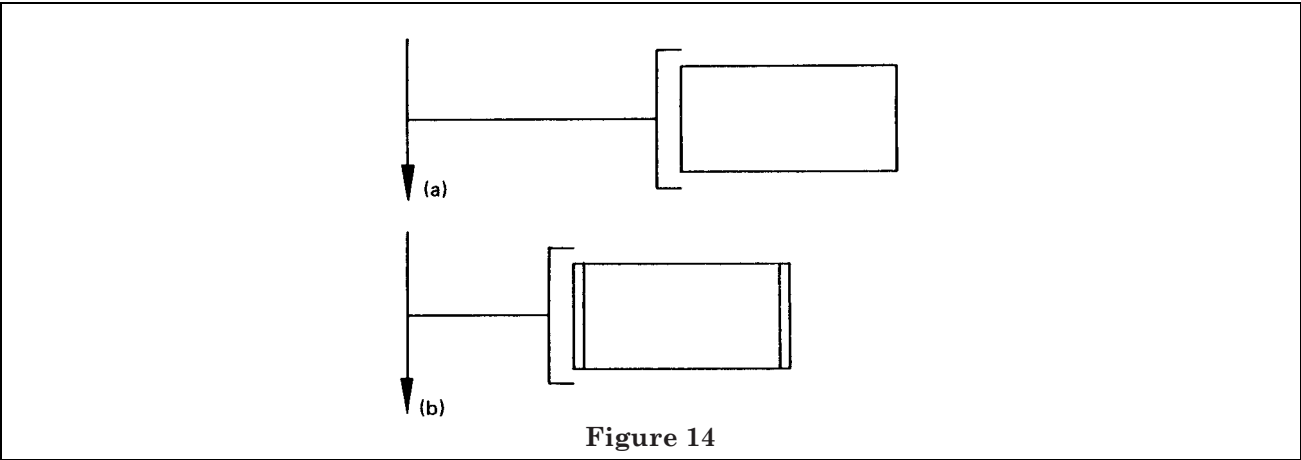
c) A documentation expansion (striped symbol)



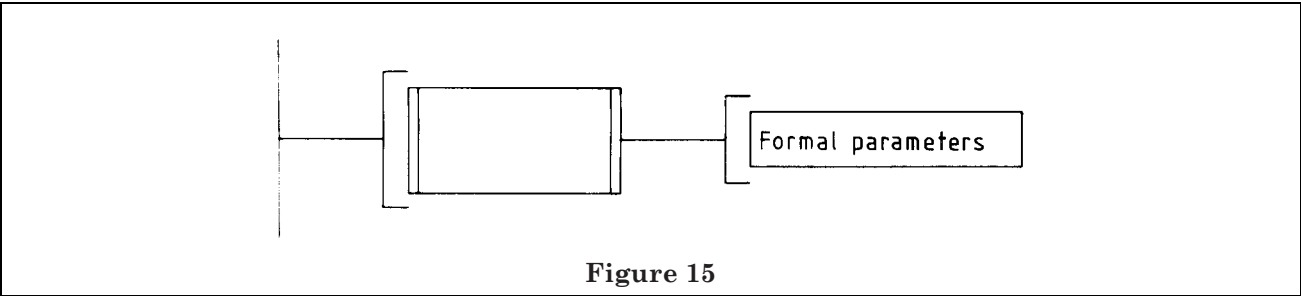
The particular text/diagram that is to be included at the place occupied by the symbol is written/drawn elsewhere in the documentation. The cross-reference is provided in the horizontal stripe.

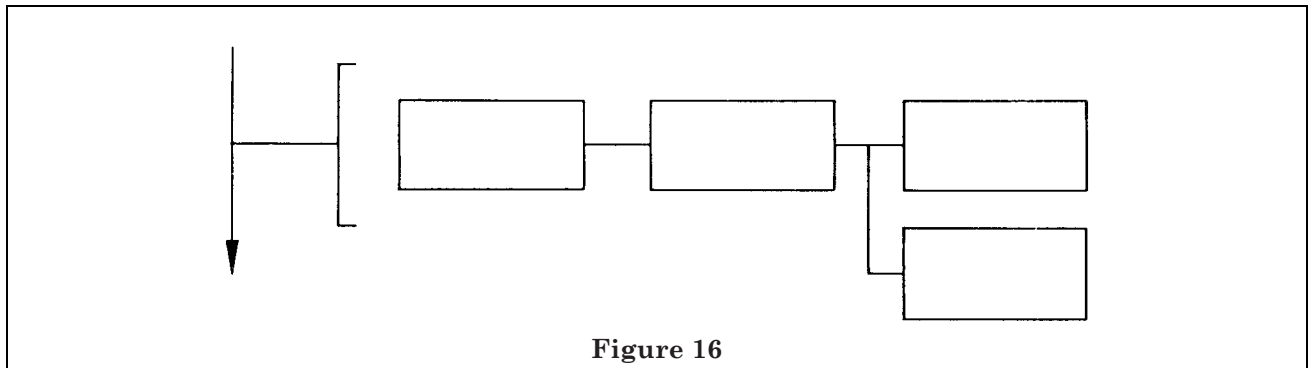
The horizontal stripe may be added to any block symbol (including the task symbol shown in 4.3.1).

d) Definitions or declarations. Definitions or declarations are introduced by a modified form of the annotation symbol as shown in Figure 14 and are used to represent declarations of variables or data, etc. For example, Figure 14(b) indicates a procedure definition.



Formal parameters should be depicted by an additional attachment as shown in the example in Figure 15. Declarations may be grouped together to aid clarity, as shown in the example in Figure 16.

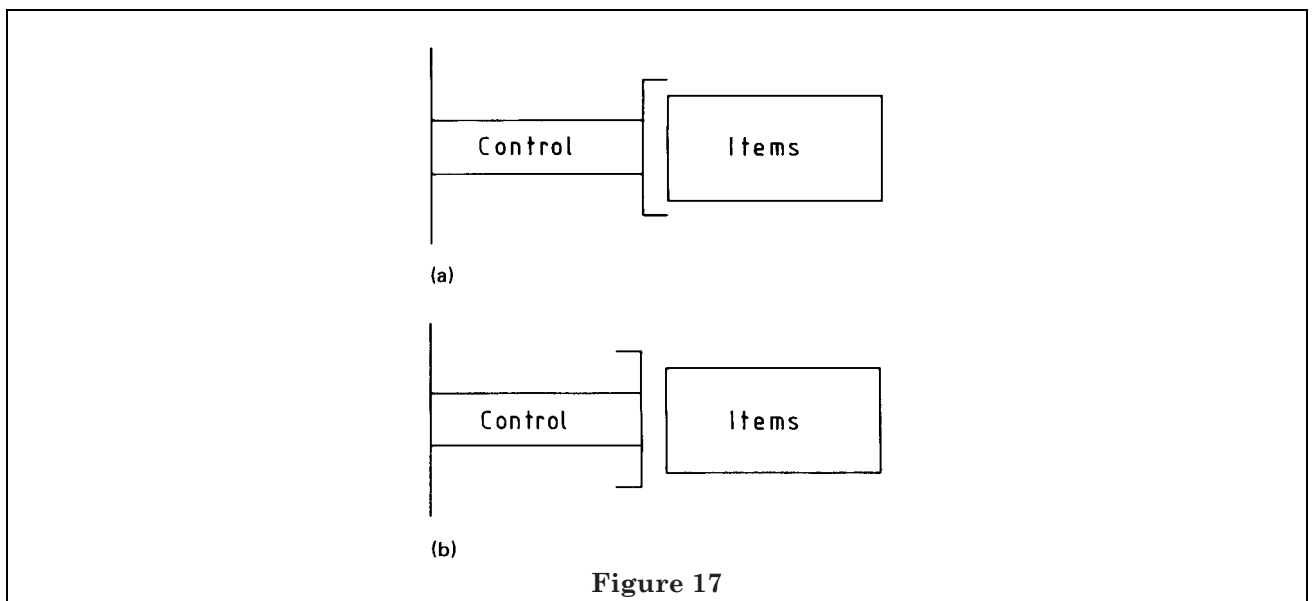




e) *Visibility control*. In some systems it is possible to extend or restrict the visibility of items declared within nested scopes (i.e. control of the area of program over which a declared item can be used).

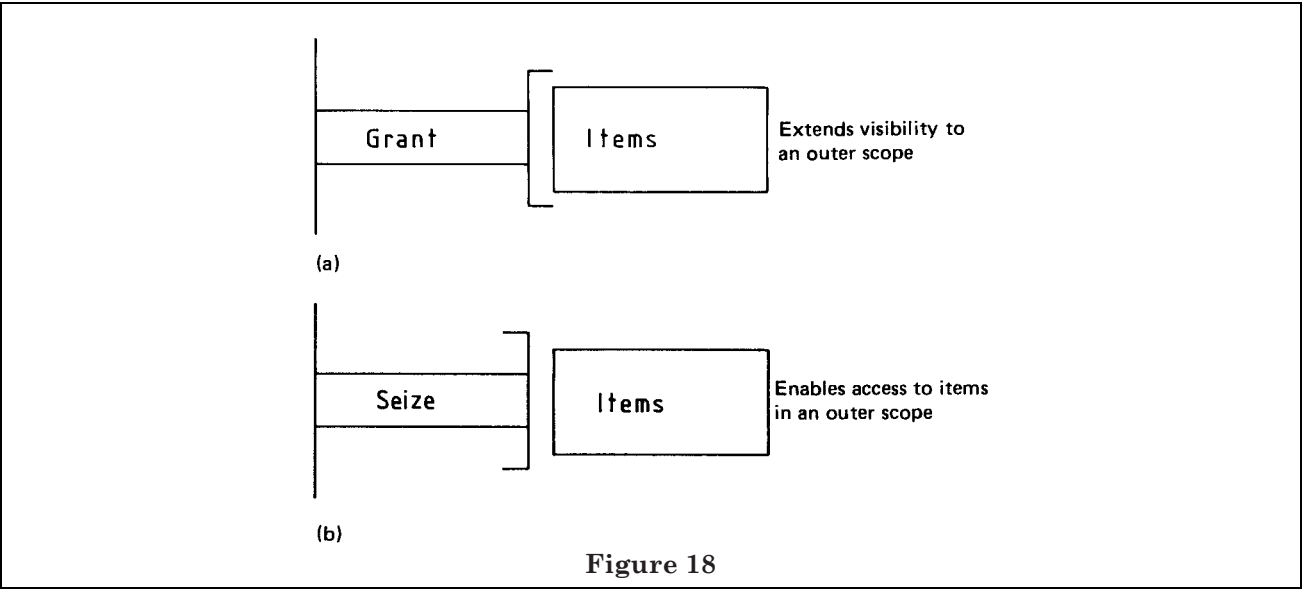
Figure 17(a) shows an example of a declaration symbol that extends the visibility of the declared item beyond the normal scope boundary.

Figure 17(b) shows an example that allows access to be made to items across a scope boundary.

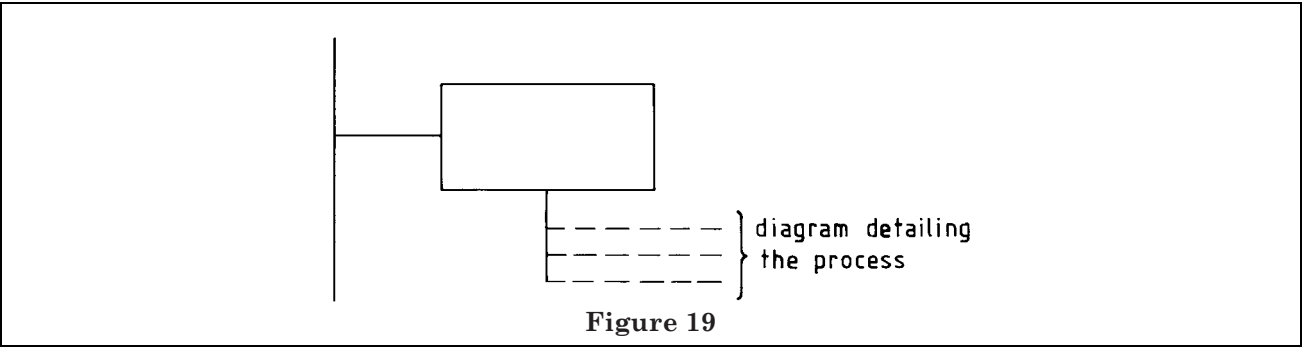


Items may be grouped in a similar way to that shown in Figure 16.

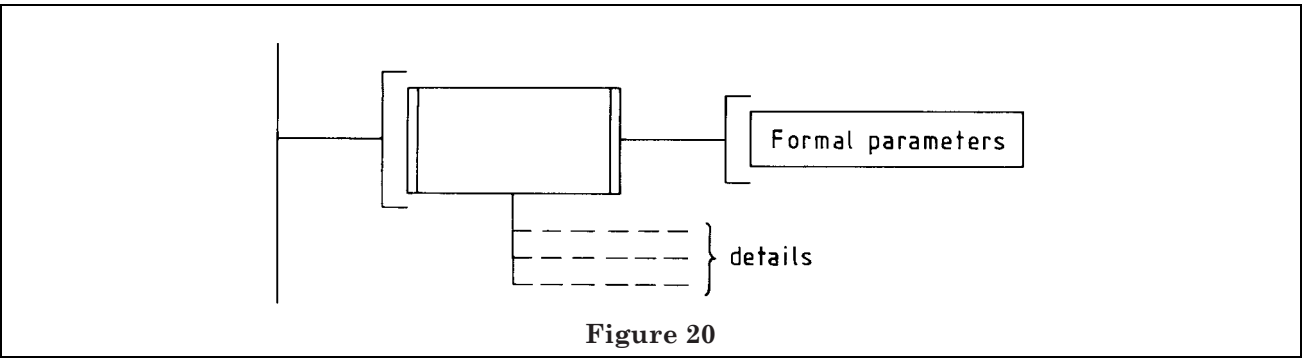
Figure 18 shows examples taken from the CCITT language CHILL.



f) *Block exposure.* The inner details of any block except an expansion symbol may be revealed by attaching the details as shown in Figure 19.



Thus a fully detailed procedure declaration is as shown in Figure 20.



NOTE For additional information on procedure or task definitions see example 10 in Appendix A.