About the presentation of programs.

The purpose of this note is to explore a style of program presentation. Whenever we are aware of having adopted a convention, we shall describe and motivate it. We shall carry out our exploration by presenting a modest program. It is understood that this example will not confront us with all notational choices to be made — e.g. the example we have in mind doesn't call for the introduction of an abstract data type — . We shall give, each time, a description, followed by an explanation of the conventions used, followed by a discussion of those conventions.

Our example is a program that we call binsrch embodying the idea of the binary search. Here is the specification of binsrch .

```
|[ N,X : int
 ; M: array (0..N) of int
 ; pres: bool
   {Q0: $M(0) \le X < M(N-1)$}
   {Q1: $(\underline{A}\ i,j : 0 \le i \le j < N : M(i) \le M(j))$}
 ; binsrch
   {pres such that R: $pres = (\underline{E}\ i : 0 \le i < N : M(i) = X)$}
]|
```

The explanation of the above is the following. The opening bracket "|[" opens a scope, which the closing

bracket "]|" closes. The first three lines declare more or less in the style of PASCAL —why not?— the environment in which the more detailed specification of binsrch is to be understood. In the text, binsrch is preceded by the precondition(s) and followed by the postcondition(s), conditions being surrounded by braces. This arrangement is deemed to prescribe for the program binsrch that initial satisfaction of the precondition(s) guarantees that activation of binsrch establishes the postcondition(s) in a finite number of steps; the "pres such that" in the postcondition indicates that R should be interpreted as an equation with pres as the unknown. Since part of the experiment is trying out the convention of indicating ranges of natural numbers with lower bound included but upper bound excluded, the subscript of M satisfies $0 \leq$ subscript $< N$ (hence the "more or less" in our reference to PASCAL).
So much for the explanation; now the discussion.

ALGOL 60 uses the bracket pair "begin" "end" for different purposes: to form a compound statement and to delimit the scope of a nomenclature. We prefer to use different pairs for different purposes. Some programming languages indicate this difference by combining different opening brackets with one general closing bracket. Since reading program texts backwards should be permissible, we are not attracted by that asymmetry.

Dedicated closing brackets are well-known, such as in "<u>do</u> ... <u>od</u>" and "<u>if</u> ... <u>fi</u>". From a lexical point of view, however, these "word delimiters" are unfortunate: one has to know them as bracket pairs and one has to know which one opens and which one closes. Since we never know how many bracket pairs we might need, we have chosen a way of generating them: a prefixed "[" as opening bracket and a similarly postfixed "]" as the matching closing bracket. This is a perfectly general technique provided the square brackets are used for no other purpose. (We have considered the obvious alternative of the postfixed "[" and the prefixed "]". Since the choice seemed technically irrelevant, we felt free to follow the tradition that started with the Kleene star.)

<u>Note</u>. We conclude that the sometimes adopted convention of delimiting comments by /* and */ is deplorable. (End of Note.)

Our convention of indicating a range of natural numbers from and including a lower bound up to and excluding an upper bound has been motivated elsewhere (see EWD768) and will not be discussed here.

In naming the declared variables we have adhered to our habit of using capital letters for constants and lower-case letters for variables. Since a variable of a block can be a constant for one of its inner blocks, no attention should be paid to the distinction. Whether capital letters will be given a special rôle — such as, for instance, in the identifier "FirstElementInTheQueue" — is still an open question. Two remarks, however, about the identifier pres, which is, of course, short for "present." The traditional name for such a booleam variable is "found" —often initialized with the value false!—, but for its operational connotations this name is most unfortunate. We have abbreviated "present" to the nonexisting word pres, thus avoiding ambiguity when referring to the variable in English prose. We —and not only we— have a strong suspicion that the use of mnemonic names is responsible for many errors in programming — the empty queue has no first element!— . The reason for not radically abolishing all mnemonic names is that in a sufficiently rigorous approach they should be harmless.

We have taken the somewhat oldfashioned view that a block primarily consists of declarations and statements, separated by semicolons, and that "comments" may be inserted where desired. This view explains the occurrence of the semicolons; in placing them at the beginnings of lines we have followed a suggestion of S. Doaitse Swierstra's.

Thus the first column clearly reflects the syntactic structure of the block.

We have chosen to surround assertions by the traditional braces. Besides formulating the assertions we have named them for future reference; a further function of the closing brace is to mark the end of the formula named. Juxtaposition of assertions denotes conjunction.

Note. The reader is invited to remember that this note discusses the presentation of programs rather than their development. The names Q0 and Q1 are introduced separately because (we already know that) they are needed separately in the sequel. (End of Note.)

When an assertion need not be referred to we shall leave it anonymous. Occasionally —most likely for reasons of layout— the formulation of the assertion may be given outside the block.

The introduction of "such that" in the postcondition may amaze the reader. There were two reasons for not introducing a special character. Firstly, the most likely candidate, the colon, should not be overloaded too much; secondly —but time will show— we expect to need the "such that" so rarely that a special character would be inappropriate.

We are left with the name binsrch ; it stands for the program to be presented in more detail in the sequel. That further presentation is to be understood entirely within the context of the block we have given. (Since this block is the universe of discourse for the remainder of the presentation, this block as a whole could remain anonymous.)

Our first block gives - in our current opinion - precisely what should be given first. It states the relevant conditions, the variables in which these are expressed are enumerated in the declarations, the proof obligations regarding binsrch follow from the structure of the text, and the names introduced give us the handles we need.

Our next task is to give a more detailed description of the statement binsrch . Here it is.

```
binsrch: |[ $i$ : int
            {Q0}
          ; loct
            {Q2: $M(i) \leq X < M(i+1) \wedge 0 \leq i < N-1$}
            {Q1}
          ; pres := $(M(i) = X)$
            {R, see Note 1}
          ]|
```

The purpose of the above is to introduce a local integer variable i together with the full description of its rôle, i.e. to satisfy Q2. The first line declares it, the next three lines give most of the functional specification of the statement loct —to be detailed later—. The next statement is not named but is fully given in situ. As before, the conditions surrounding it give its functional specification; since the statement is fully given, this is the moment to supply the argument that it, indeed, meets its functional specification. For this argument, the last line refers the reader to Note 1. Finally, the reader is supposed to convince himself that the above block meets the functional specifications of binsrch. So much for the explanation; now the discussion.

The above block is to be understood in the full context previously created for binsrch. This context includes the specification that within binsrch the identifiers N, X, and M refer to constants; hence we have taken the liberty of not inserting "i such that" in front of "Q2". (Yet we feel free to insert such a phrase, though formally superfluous, when clarity seems to demand it.) This context also includes that $\{Q0 \wedge Q1\}$ is the precondition of binsrch; hence this precondition has not been repeated —say, between "binsrch:" and "|["—. Similarly, $\{R\}$ has not been repeated after "]|". The place of the assertion "$\{Q1\}$" has been chosen for two reasons. Firstly, as time will show, loct does not need

Q1 as a precondition for the establishment of Q2. Secondly, the second statement does need Q1 for the establishment of R. Since we are describing, and not developing, a program, this is not the moment to prove the satisfiability of Q2: in due time the definition of loct will provide a constructive proof.

It is now time to give the still missing Note 1.

Note 1. The Axiom of Assignment tells us to prove

$$Q1 \wedge Q2 \Rightarrow (M(i)=X) = (\underline{E}j: 0 \leq j < N: M(j)=X) .$$

In the case $M(i)=X$, this is obviously true. In the case $M(i) \neq X$, we have, thanks to Q2, $M(i) < X < M(i+1)$, and therefore, thanks to Q1, $(\underline{A}j: 0 \leq j < N: M(j) \neq X)$, i.e. the truth of the right-hand side of the implication. We draw the reader's attention to the fact that this is the only place where we use Q1, i.e. the fact that array M is ordered. (End of Note 1.)

Our next task is to give a more detailed description of statement loct. Here it is.

```
ct: |[ j: int
    ; i,j := 0, N-1
      {Q3: M(i) ≤ X < M(j) ∧ 0 ≤ i < j < N}
    ; clos
      {Q3 ∧ i+1 = j, hence}
    ]|
```

The purpose of loct was to assign to i a alue satisfying Q2, given the initial truth of Q0. he purpose of the above is to introduce a local nteger variable j together with the full descrip- ion of its rôle. Its rôle is to satisfy Q3 to tart with and to satisfy eventually $i+1=j$ as well, o that Q2 is implied and j can again be eliminated rom the argument. The first statement, being given n full, represents as before a proof obligation o be fulfilled here and now; the absence of a eference to an explanatory "Note" means that this proof is entirely left to the reader. The fact that Q3, the precondition of clos, is repeated in its postcondition states explicitly that Q3 is a so- called "invariant" for clos; if clos turns out to e a repetition, $i+1=j$ will have to follow from the alsity of its guards. So much for the explanation; now the discussion.

We have not inserted "{Q0}" in front of the irst statement. One might feel that we should have nserted it in order to capture the corresponding proof obligation, but the insertion would not have relieved the reader from consulting the context in which loct is to be understood (viz. the block called binsrch): he would have had to verify that the insertion was justified. For the same reason we have not inserted "Q2" after "hence". In the specification of clos we have stuck to our rule —to which we could adhere so far— of formulating postconditions as a single assertion. Note that this is the first case we encounter in which a postcondition could remain anonymous; this possibility is closely related to the decision not to repeat pre- and postconditions of statements in their refinements.

Our next task is to refine clos .

```
clos:  *[ $i+1 \ne j$
          → |[ h: int {$j-i \geq 2$}
              ; aver {h such that $i < h < j$} {Q3}
              ; -[ $M(h) \leq X$ → $i := h$  {Q3, inc i}
                 [] $X < M(h)$ → $j := h$  {Q3, dec j}
                 ]-
            ]| {Q3, dec (j-i)}
       ]*
```

The purpose of clos was to establish $i+1 = j$ under invariance of Q3 . We use the bracket pair "*[" and "]*" —instead of the old "do ... od"— to form a repetitive construct. The repeatable statement is a block since it is the scope of the local integer constant h. The assertion $j-i \geq 2$ being mentioned without reference to an explanatory "Note", the reader is supposed to verify himself that it follows from the invariant and the guard. The rôle of constant h, which has mainly been introduced for brevity's sake, is to satisfy $i < h < j$. We use the bracket pair "-[" and "]-" —instead of the old "<u>if</u>...<u>fi</u>"— to form an alternative construct. The absence of a reference to an explanatory "Note" in the final assertion leaves the reader with the obligation to verify that the block maintains the truth of $Q3$, that the block decreases the value of $j-i$ and that the latter value is bounded from below. (The assertions following the alternatives can be viewed as hints towards this purpose.) So much for the explanation; now the discussion.

We have taken the liberty of placing assertions where we liked them. Similarly we have taken the liberty of writing simple guarded commands on single lines. We would like to stress that we have taken the position that the reader should be thoroughly familiar with the proof rules; the invariant relation and the convergence criterion —here denoted by "<u>dec</u> $(j-i)$"— have been given explicitly, since they are specific for this repetitive construct. The indication "<u>dec</u> $(j-i)$" is to be understood to refer to the complete preceding guarded command of the repetitive construct. Determining the "ranges" of the internal hints "<u>inc</u> i" and "<u>dec</u> j", however, is left to the intelligent reader.

In a book presenting algorithms, the proof rules should be stated explicitly —in the introduction— for the sake of self-consistency of the text. They are so few and so ubiquitously used that explicit reference to them in the program presentations would be repetitious. In retrospect the explicit reference to the Axiom of Assignment in Note 1 was probably a mistake.

Our final task is the refinement of aver . We give three options:

aver: $h := i+1$

aver: $h := j-1$

aver: $h := (i+j)$ <u>div</u> 2 {see Note 2}

<u>Note 2</u>. With <u>div</u> rounding to a nearest integer, aver establishes

$$2\cdot h = i+j+d \quad \text{with} \quad abs(d) \leqslant 1 \quad .$$

Adding $j \geqslant i+2$ yields $2\cdot h \geqslant 2\cdot i + 2 + d > 2\cdot i$, hence $i < h$; $h < j$ follows similarly from adding $i \leqslant j-2$. (End of Note 2.)

We have given the different options for aver in order to show the extent to which the linear and the binary search can be mapped on each other. Note 2 has been given to stress this algorithm's independence of whether <u>div</u> rounds up or down. Our last refinement reveals another reason for introducing a name: "aver" had to be introduced because we wished to supply different options.

* * *

Our presentation fails to express that only the first N-1 elements of M matter: the last element, which, by definition, exceeds X, is never accessed and need not be stored. The coding trick, which we regret in retrospect, wouldn't have been necessary if we had replaced in Q3 the term $X < M(j)$ by

$$j = N\text{-}1 \ \underline{\mathrm{cor}} \ X < M(j) \quad .$$

The constant N should have been chosen in such a way that the first N elements of M matter. In retrospect we feel that the introduction of the "plus infinity" at the far end of the array was a mistake.

* * *

When Wim Feijen saw our first pages, he was horrified by our choice of the identifier "binsrch".

It is indeed terrible: "bins" would have been less obnoxious. (We once erroneously wrote "binsearch" in full.) We then chose loct for locate, clos for close, and aver for average. Each time the choice of identifier was a pain in the neck; in contrast, the choice of the identifiers i, j, h, R, Q0, Q1, Q2, and Q3 was absolutely painless. We now have the feeling that naming our statements, say, S0, S1, etc. would have been preferable.

We thank W.H.J. Feijen for a number of pertinent remarks while the above was being written.

Our final task is to solicit the reader's comments and advice.

16 March 1981

drs. A.J.M. van Gasteren
BP Venture Research Fellow
Department of Mathematics
University of Technology
5600 MB EINDHOVEN
The Netherlands

prof. dr. Edsger W. Dijkstra
Burroughs Research Fellow
Platoanstraat 5
5671 AL NUENEN
The Netherlands