CHAPTER 11.

ALGOL FUNDAMENTALS

In order to assist readers unacquainted with ALGOL in getting an impression of SIMULA, the basic ALGOL concepts are explained in this chapter. Readers are warned that this presentation is very incomplete. An ALGOL textbook must be consulted to get the knowledge necessary to write SIMULA programs.

11.1 Simple Variables and Declarations.

Variables to be used in ALGOL (and SIMULA) must be "declared": we write the name ("identifier") of the variable preceded by an underlined word stating the "type" to which the variable belongs.

ALGOL contains 3 types of "simple" variables: "Boolean", "integer" and "real". Boolean variables (named after the British mathematician Boole) may have one of the two "truth values", true and false. Integer variables may have a positive or negative integer or zero as values. Real variables may have any real number as values.

When variables are declared we write the type-name underlined, followed by the identifiers of the variables belonging to that type. The identifiers are separated by commas, and the list is ended by a semicolon.

Example:

```
Boolean tax paid, integer flightnr, qlength,
real baggage weight, arrivaltime, price,
```

ALGOL also contains subscripted variables, to be discussed in Section 11.6.

11.2 Statements and Programs.

An ALGOL program consists of

1. Declarations (e.g. of simple variables)
2. Statements describing actions involving the declared entities.

A program has an underlined begin as the first word (symbol), followed by the declarations and then the statements. The declarations and the statements are separated by semicolons and the program is concluded by the underlined word end.

Example:

The clerk at the check-in counter of an airport is told to report the total baggage weight as well as the weight of the checked baggage for a given flight. He reads the weight of the suitcases and then of the briefcase (the hand baggage) for each passenger as they turn up (the symbol ": =" means "becomes", and the statements are executed in sequence):

```
begin                                              (Line 1)
real tot wt, checked, suitcases, briefcase;        (Line 2)
tot wt: = 0; checked: = 0;                         (Line 3)
read(suitcases, briefcase);                        (Line 4)
    checked: = checked + suitcases;                (Line 5)
    tot wt: = tot wt + suitcases + briefcase;      (Line 6)
read(suitcases, briefcase);                        (Line 7)
    checked: = checked + suitcases;                (Line 8)
    tot wt : = tot wt + suitcases + briefcase;     (Line 9)
read(suitcases, briefcase);                        (Line 10)
```

(The same sequence is repeated for all passengers on the flight)

```
write(checked, tot wt)
end;
```

In the program we are allowed to operate upon the 4 real variables declared in Line 2. In Line 3 "tot wt" and "checked" weight are given 0 as initial value. Line 4 states that the values of "suitcases" (the weight of the suitcases) and "briefcase" now shall be read. The reading may involve a complex series of actions, which we only specify by the "procedure - call" read(-------).

We only will assume that if "x" is a declared variable, then the execution of the procedure-call read (x) will assign a value to x from some source of information. In programs executed on computers, read (x) will usually mean that the value of x is read from a punched card. We will dicuss procedures in section 11.10.

Since the suitcases are to be checked in, the weight of the checked baggage (initially zero) is increased by the weight of the suitcases (Line 5). We get the total weight by adding "suitcases" and "briefcase" to the present value of "tot wt" (Line 6).

Lines 7 - 9 repeat the actions of Lines 4 - 6, this time operating upon "suitcases" and "briefcase" for the next passenger. This sequence is now repeated till all passengers have arrived. Then a new procedure "write (-----)" is called, recording the final values of "checked" and "tot wt".

11.3 Compound Statements.

Sometimes it is necessary to transform a sequence of statements to a single "compound" statement. This is done by bracketing the statement sequence between begin and end.

Example:

The statements in Lines 4 - 6 in the previous example may be written as a compound statement

```
begin read(suitcases, briefcase);
      checked := checked + suitcases;
      tot wt := tot wt + suitcases + briefcase end;
```

We may name this statement "check in", and rewrite the program in abbreviated form (for our own purpose, not in correct ALGOL):

```
begin declarations; tot wt := checked := 0;
      check in; check in; ----------; check in;
      write(checked, tot wt) end;
```

When we return to "procedures" in section 11.10 we will demonstrate how this use of the word "check in" may be made "legal" ALGOL.

11.4 Labels and go to - statements.

In the counter example Lines 4 - 6 ("check in") are repeated for every passenger. A language not taking advantage of this will lead to very long programs.

It is possible to mark an ALGOL statement by a "label": an (undeclared) identifier followed by a colon. This label then may be used in a go to - statement.

Example:

```
begin declarations; tot wt := checked := 0;
      next: check in; go to next end;
```

When "check in" of a passenger is completed, the same sequence is repeated by stating that the "check in" statement, marked by the label "next", shall follow the go to - statement.

This new version of the program is not able to transmit results of the computation, it goes in an infinite loop. The loop shall, however, only be executed as many times as there are passengers.

11.5 if - then - statements.

The ALGOL Boolean variables have two possible values: true or false.

The statement

"7 < 1"

is false. We name statements to which we may assign a truth value "Boolean expressions". Hence the value of the Boolean expression "7 < 1" is false.

ALGOL also contains statements executing the standard operations of two-valued logics:

∧ : logical "and"
∨ : logical "or"
¬ : negation
⊃ : implication
≡ : equivalence

Ecample:

Let us assume that the clerk computes the baggage weight. If the limit is 20 kilos, excess baggage freight must be paid. Then, for a given passenger, the postulate "money left" may be true or false.

We introduce the variables real excess wt and Boolean money left. The course of further action will then be dependant on the Boolean expression.

$$(\text{excess wt} > 0) \wedge (\neg \text{money left})$$

In an "if - statement", two alternate courses of action may be prescribed, depending on the value of a Boolean expression:

Let B be a Boolean expression and S1 and S2 two statements. We may write

if B then S1 else S2;

implying that S1 shall be executed if B has the value true, S2 if B has the value false. If the alternative to S1 is "no action", we may write

if B then S1;

Examples.

We now may complete the counter example. Let integer nr passengers, checked passg denote the total number of passengers on the flight and the number of passengers already checked in.

```
begin Declarations; tot wt := checked := 0; checked passg := 0;
      read(nr passengers);
next: check in; checked passg := checked passg + 1;
      if checked passg < nr passengers then go to next;
      write(checked, tot wt)
end;
```

Returning to the excess weight situation, we may want to state

```
if (excess wt > 0) ∧ ( ¬ money left) then go to home
else go to aircraft;
```

where "home" and "aircraft" are two labels in other sections of the program.

If we want to execute a sequence of statements in each "branch" of the if - statement, these sequences must be written as compound statements. S1 and S2 must each be one statement, but may be a simple or a compound statement.

11.6 Arrays.

Boolean, integer and real variables may be subscripted and form arrays. In their declarations the ranges of variation of the subscripts are specified:

```
Boolean array open counter [1:6] ;
integer array q length [1:6] ;
```

We now refer to the queue length of counter nr 5 by

```
q length [5] ,
```

and may write e.g. (k is an integer variable):

```
if open counter [k] then q length [k] := q length [k] + 1;
```

assuming that $1 \leq k \leq 6$

Arrays may also be multidimensional. Let us assume that all airports are numbered 1, 2, ..., N. Then the destination of a passenger may be specified by an integer between 1 and N.

If we need a table of ticket prices, 1st and Tourist Class (regarded as 2nd Class) this is done by introducing a real array (or just array):

```
array ticketprice [1 : 2, 1 : N] ;
```

The ticket price on Tourist Class to Airport nr. 122 may then be referred to by

```
ticketprice [2, 122] .
```

11.7 for - statements.

The ALGOL for - statements are flexible tools for specifying repetition of statements.

Example 1:

The repetition in the counter example may be written:

for checked passg := 1 step 1 until nr passengers do check in;

where "check in" is the above compound statement.

The for - statement tells us that "checked passg" shall be given the initial value 1 and "check in" be executed. Then "checked passg" shall be increased by 1 ("stepped up") and "check in" once more executed. When checked passg = nr passengers "check in" is executed for the last time and then the next statement is executed.

Example 2:

Another type of for - statement is the following:

```
begin
integer tot q, k; integer array q length [1:6] ;
-----------------------------------------------
tot q := 0;
for k := 1, 3, 6 do tot q := tot q + q length [k] ;
-----------------------------------------------
end;
```

k is successively given the values 1, 3 and 6, and for each of these values the statement after do is executed.

Example 3:

A third alternative in the counter example is to write

```
begin ----------------
      checked passg := 0;
      for checked passg := checked passg + 1 while
      checked passg ≤ nr passengers do check in;
-------------------- end;
```

11.8 Blocks.

An ALGOL "block" is a series of declarations followed by one or more statements. Each statement may be simple or compound. A block starts with begin and is concluded by end.

The declaration part of a block is called the "block head".

Example:

The passenger at the counter has to pay an excess freight for his baggage if the weight exceeds 20 kilos. For each kilo of excess weight he has to pay 1% of the 1st Class ticket price to his destination. We will assume that there are 500 possible destinations.

```
begin real suitcases, briefcase, excess freight;
      integer destination; array ticket price [1 : 500];
      ------------------------------------------------
      ------------------------------------------------
      if suitcases + briefcase > 20 then
      begin real excess wt;
      excess wt := suitcases + briefcase - 20;
      excess freight := excess wt × ticketprice [1, destination]/100
      end else excess freight := 0;
      ------------------------------------------------
end;
```

Here the whole program is a block, and the compound statement after "then" also is a block with its own block head and statements, inside the program block.

A block is itself a statement and therefore may contain other blocks, "inner blocks".

Variables declared within a block are said to be "local" to this block.

Variables local to a block are available for use in all statements inside the block, including inner blocks.

Variables local to a block are not available for use outside the block.

In the example "suitcases" and "excess freight" are declared in the outer block and hence may be used in the inner block. "excess wt" only exists within the inner block.

These properties of ALGOL blocks are very useful, since large data structures only needed in a small part of the program do not occupy space in the computer memory when other parts are executed, if the part where these data are needed is written as a block. Also the same name may be used in different blocks B1 and B2 for different purposes without confusion. If B1 is an outer block to B2, the "outer variable" is not accessible within B2. The common name has its "local" significance within B2.

The block structure of ALGOL is fundamental for its usefulness as component of a simulation language.

11.9 ALGOL programs.

An ALGOL program is a block or a compound statement having no outer block and not being a part of another compound statement.

11.10 Procedures.

In the counter example we have used the abbreviation "check in" for the compound statement

```
begin read(suitcases, briefcase);
      checked := checked + suitcases;
      tot wt := tot wt + suitcases + briefcase end;
```

The use of this abbreviation may be made "legal" by declaring "check in" as a procedure.

Together with the declarations of variables in the block head we write a "procedure declaration":

```
procedure check in;
      begin read(suitcases, briefcase);
            checked := checked + suitcases;
            tot wt := tot wt + suitcases + briefcase end;
```

This being done, it is permissible in ALGOL to use the statement "check in" in the above manner. The statement is usually said to be a "procedure call" (correctly: "a procedure statement").

In the above procedure declaration, the first line

```
procedure check in;
```

is called the "procedure heading". The compound statement following the heading is called the "procedure body". In general the procedure body is a block, possibly having its own declarations.

A procedure statement ("call") prescribes an execution of the procedure body at this place in the program.

The procedure "check in" results in operations on the same variables each time it is called. The read-procedure mentioned earlier may be used to assign values from an input device to any variable. The variable which shall get a value is given as a parameter to the call:

read (nr passg), read(suitcases).

The procedures for input and output of data are not standardized in ALGOL and have to be specified for the machine which is used for the computation.

Certain commonly used mathematical functions are, however, considered to be part of the language, e.g. the sine and cosine functions, the square root, the exponential function e^x.

The calls for evaluation of e^x and $\sqrt{x}$ (where x is a real variable) are written

exp(x), sqrt(x)

If we in a computation need repeated evaluation of a function not being part of ALGOL, we may declare a function procedure in our program.

Example.

We want to declare the hyperbolic cosine cosh(x) as a procedure. cosh(x) has a real value, and

$$\cosh(x) = \tfrac{1}{2}\,(e^x + e^{-x})$$

```
real procedure cosh(x); real x;
                        cosh := (exp(x) + exp(-x)) / 2;
```

In the procedure heading we have specified that x is a real parameter, which means that the "actual parameter" given in a call for the procedure should be some real expression. Such specifications are an optional part of the language, but many compilers require specification of all parameters, since this information makes it possible to generate more efficient object programs.

A call for a function procedure is called a "function designator". It is an expression whose type is specified by the first word of the procedure heading. The function designator "cosh(y)" evaluates the hyperbolic cosine of the real variable y, as stated in the procedure declaration above. y is the actual parameter of the call. The actual parameter can in this case be any arithmetic expression: cosh(log(y) + 0.5/z[i]) gives the hyperbolic cosine of the value of the expression.

There is an important difference between parameters to the "read" and "cosh" procedures. The call read(y) assigns a value to the variable y, whereas the cosh procedure only uses the value of the given actual parameter for its own purpose.

Formal parameters of the second kind can be included in a "value specification" in the procedure heading:

```
real procedure cosh(x); value x; real x; -----
```

A value specification improves the efficiency of the object program and should be given if possible.

If a formal parameter is included in a value specification, it is said to be "called by value", otherwise it is said to be "called by name".

A parameter called by value is similar to a local variable, except that it has an initial value defined by the actual parameter of the call. An assignment to a value parameter has no effect outside the procedure body.

A parameter called by name is considered to be replaced by whatever expression is given as the actual parameter of the call. If an assignment is made to a name parameter, the actual parameter must be a variable. This variable recieves a value as a result of the procedure call. Such a parameter is often called an "output parameter", since it transmits output from the procedure.

A procedure can refer directly to variables and other items non-local to the procedure body, i.e. to items local to the block containing the procedure declaration, or to outer blocks. If the value of a non-local variable is changed within the procedure body, this is called a "side-effect" of the procedure.

As an example on the general procedure concept we will formulate a procedure for evaluating the polar coordinates (r, v) of a point whose cartesian coordinates are (x, y). (x to the n'th power is written x↑n in ALGOL.)

```
procedure polar (x, y, r, v); value x, y; real x, y, r, v;
if x = 0 ∧ y = 0 then r := v := 0
else begin r := sqrt (x↑2 + y↑2);
                    v := arctg (x, y) end;
```

where arctg evaluates the arc in a suitable range.

Clearly r and v must be called by name, since they are output parameters, whereas x and y can be called by value.

The procedure statement

```
polar (3, 7, f, g)
```

will assign the polar coordinates of the point (3, 7) to the real variables f and g.

The call

```
polar (z + 1, w - u, f, g)
```

will assign to f and g the polar coordinates of the point whose cartesian coordinates are the current values of the expressions z + 1 and w - u.