MANCHESTER UNIVERSITY

COMPUTER SCIENCE DEPT.

THE ATLAS AUTOCODE MINI-MANUAL

W. F. Lunnon

G. Riding

July 1965

References & Notes

'Atlas Autocode Reference Manual' by R.A.Brooker & J.S.Rohl at Manchester University.

'Programming Manual for Atlas Basic Language' at I.C.T. Ltd, list no. CS348A. (Also CS348 with amendments.)

There follows a summary of the basic facilities of Atlas Autocode, as realised in COMPILER AB. [ ] are used for grouping and parenthesis, := and :- mean 'stands for' and 'the following', and a prime indicates 'any no. of, including none'; in this context these symbols are external to the language.

x := name of a real scalar
i := name of an integer scalar
a := name of a [real] array
yI) := name of a [real] array element
k := name of an integer array
k(I) := name of an integer array element
v := name of any variable, e.g. x, i, a(I), k(I)
A := general name
STRING := any string of characters excluding newline & semicolon
E := real expression
I := integer expression, a special case of E
C := any constant
N := any integer, a special case of C
STAT := statement (q.v.) - a permissible line of program
BP := bound pair := I:I
α-list := α[,α]' := α,α,...,α where α stands for any of the above

Characters, Names & Delimiters

Letters & digits:- A ... Z a ... z 0 ... 9
+ - · / | ( ) = < > _ , . : ' ½ ² & [ ] α β π
Compound characters:- ≠ ≠ ≥ ≤ ⊧ ; c →
All except the last of these include backspace. See also Captions.
L := letter D := digit P := prime A := name := LL'D'P'
delimiter :- + - · / ⊧ ( ) , : ; > < ≥ ≤ = ≠ → ¤ |
cycle repeat real integer array name if then unless caption comment fn spec end end of program return result and or begin stop routine switch

Labels & Jumps

LABEL := simple or switch label :- N: A(N):
JUMP :- → N → A(I)

Switches must be declared, together with their ranges, (see DECL) and the name A must be local. DECLs and routine headings should not be labelled; blank lines may be.

Assignments

OPND := operand:- v C (E) |E| function

OPTR := operator:- + - / ↑

E := real expression:- +' OPND [OPTR OPND]'

I := integer expression:- E, but with all OPNDs of type integer, and such that the result is an integer

C := constant: fixed or floating point, using α to imply a base of 10 for the exponent, e.g. fifteen may be punched: 15 0015.00 .15α2 1500α-2

Operator precedence: ( ) are evaluated first, then ↑, then * and /, then + and -. Otherwise the operations are performed in order of occurrence, e.g. a + b * (d - c ↑ e ↑ f) means (a + (b * (d - ((c ↑ e) ↑ f))))

ASST := assignment :- v = I x = E a(I) = E

Statements

Ø :- = ≠ > ≥ < ≤

SC := simple condition:- E Ø E (GC)

GC := general condition:- SC [and SC]' SC [or SC]'

UI := unconditional instruction :- ASST routine call JUMP return result = E stop caption STRING

CI := conditional instruction :- if GC then UI unless GC then UI UI if GC UI unless GC

DECL := declaration:- real x-list integer i-list array [a-list(BP-list)] - list OR real array ... integer array [k-list(BP-list)] - list switch [A-list(I:I)] - list routine spec [See Routines]

ST := unlabelled statement :- UI CI DECL routine heading begin end end of program cycle repeat | STRING comment STRING [see also Monitoring]

STAT := statement := LABEL' ST

Blocks

A block is a list of statements enclosed between begin and end. Blocks are 'open': they may be entered and left only via begin and end resp. Blocks may be nested to any depth. A label in a block or at its end may only be referred to by an instruction in the block and not in a sub-block. The 'scope' of a name is the block at the START of which it is declared (the block to which it is 'local'), together with any blocks which may be nested within (to which it is 'non-local'). A name may be used within its scope, and nowhere else; but non-local names may be re-declared.

A program is a block with end replaced by end of program and followed by optional data, (together with the necessary supervisor material).

Routines

RT := routine type :- routine real fn integer fn
Corresponding exit :- return result = E result = I
R := name of a routine of the appropriate type
Routine spec :- RT spec R(FP-list)
Routine heading :- RT R(FP-list)
Routine call :- R(AP-list)

FP := formal parameter :-	AP := actual parameter :-
integer name i	i, k(i)
real name x	v
array name a	a
integer array name k	k
integer i	I
real x	E
routine R real fn R integer fn R	R

A routine is a named block with parameters; the routine heading replaces begin, and the corresponding exit is inserted. The FPs in the heading have the force of declarations inside the routine; but when the FP is of routine type, a spec must be inserted (from which the RT may be omitted).

A routine is 'closed': it can only be entered via a call, which causes it to be obeyed with the FPs in its heading replaced by the APs in the call. An integer or real FP will be assigned the value of the AP at time of call ('call by value'). A ...name FP will be assigned the actual store location of the AP, so that e.g. assignments to the FP alter the AP; but should the AP be an array element, its subscripts will be treated as integer parameters and remain fixed throughout the routine ('call by simple name')

In a list of FPs of the same type, the type delimiters after the first may be omitted. In the case of a parameterless routine, the (FP-list) and (AP-list) are omitted.

Cycles

```
cycle i = I1, I2, I3
        [list of statements]
        repeat
```

Here I1, I2, I3 are all of the form I, and such that (I3 - I1)/I2 is a positive integer or zero. I1, I2, I3 are evaluated at the start of the cycle, and remain unaltered throughout it.

Cycles can be nested to any depth.

Standard Functions

real fn

sin(E) cos(E) tan(E) log(E) exp(E) sqrt(E) arctan(E1, E2)
[= arctan(E2/E1), in $(-\pi/2, \pi/2)$ if E1 > 0, in $(\pi/2, 3\pi/2)$ if E1 < 0]
arcsin(E) [in $(-\frac{1}{2}\pi, \frac{1}{2}\pi)$] arccos(E) [in $(0, \pi)$]
radius(E,E) fracpt(E) mod(E) |E|

integer fn

intpt(E) [intpt(-3.73) = -4, intpt(3.73) = 3] int(E) parity(I)

Standard functions may not be substituted for FPs of routine type in routine calls (see Routines).

Input and Output

routine

select input (I)	I refers to stream in JOB descr.
select output (I)	''
read symbol (i)	See table of numerical equivalents
print symbol (I)	''
print (E, I1, I2)	Punches I1+I2+2 characs., or I1+1 if I2 = 0, unless I1 is too small
print fl (E, I)	Punches I+7 characs.
read (v-list)	Reads nos. from data 'tape' in order, punched as constants except for optional sign, terminated by space or newline, e.g. +15.0 -1.5α1 .15α+1 Initial spaces and newlines are ignored.
read binary (i)	5, 7, or 12 least significant bits
punch binary (I)	''
tab	Character positions for tab: 0 8 16 24 32 48 64 80 96 112 128 144 159 Tab advances the character position at least two spaces.
newline , newlines (I)	
space , spaces (I)	
runout (I)	no effect on lineprinter
newpage	thirty newlines on seven-hole punch

integer fn

next symbol	See table of numerical equivalents. Does not advance data 'tape'

Captions & Quotes

Instead of the numerical equivalent, the character itself may be used enclosed by quotes. An entire string of characters can be output by the statement

caption STRING

In captions or between quotes use the symbols

ß or ♮ to denote space

<u>ß</u> or <u>♮</u> '' space underlined

ſ or ⁏ '' semicolon

¶ or ↵ '' newline

e.g. print symbol (97); print symbol (65)
print symbol ('a'); print symbol ('ß')
<u>caption</u> aß

all have the same effect.

<u>Numerical equivalents</u>

A	33	?	12	stop	76	≥	11035
Z	58	&	13	:	79	⊦	11150
a	97	*	14	[	81	;	10122
z	122	/	15	]	82		
0	16	<	26	_	86	ß	14735
9	25	>	27	|	87	<u>ß</u>	1895183
		=	28	α	90	¶	14095
newline	4	+	29	β	91	ſ	1296266
(	8	-	30	½	92	♮	14807
)	9	.	31	≠	11164	<u>♮</u>	1895382
,	10	'	32	≠	3599	↵	14167
π	11	space	65	≤	11034	⁏	1435530

The numerical equivalent of a compound character is $(128^2x +) 128y + z$, where (x,) y, z are the equivalents of its constituents and (x >) y > z.

<u>Notes on punching</u>

Tab is converted to multiple space on input.

If the printout (including erases) looks right, the tape is right.

In the program:

A statement is terminated by newline or semicolon.

<u>c</u> at the end of a line causes the newline to be ignored, i.e. the statement continues on to the next line and the line number is not advanced.

Spaces, underlined spaces, erased characters and superfluous terminators are ignored.

½ and ² are converted to .5 and ⊦2 on input.

Comments may be inserted by means of

<u>comment</u> STRING or | STRING

♯ is an alternative to ≠ .

In the data:

Erased characters are ignored.

For the supervisor material (JOB descr., etc.) see elsewhere.

Monitoring

monitor statement:- compile queries ignore queries
compile array bound check stop array bound check
fault [N-list -> LABEL]-list ASST ?

Array subscripts are tested dynamically for overflow if they appear in the program between compile ... and stop array bound check, and ? - which causes the value in the preceding ASST to be printed each time the ASST is obeyed - is similarly controlled by compile ... and ignore queries.

Faults detected at compile time are noted in the program 'map' printed during compiling, and the program is not then entered. Dynamic faults, unless trapped, cause the private monitor to print the current routine and line number and a summary of the stack (working space), then terminate the run. Faults occurring just before a JUMP, return, etc. may escape detection till after the jump has been obeyed.

Some faults may be trapped by the statement fault ..., causing a jump to the LABEL (which must be simple) if fault N subsequently occurs. The stack is then restored to its extent at the time when the fault statement was obeyed; but some variables may have been altered in the meantime. fault ... is dynamic but not nested - the LABEL used is that in the last fault ... obeyed containing N - so it should normally be confined to the outermost block.

Common trappable faults:-

TRAP NO. N	NATURE OF FAULT
1	Division by 0
2	Exponent overflow
4	More store required
5	Square root of no. < 0
6	Logarithm of no. ≤ 0
8	Trig. fn. out of range
9	No more data
14	Data fault: spurious character at start of no. (this character is the 'next symbol')
16	'' : real no. instead of integer (the no. is read in off the 'tape')

Sample program

The program overleaf is badly written (superfluous instructions and cumbersome method) to involve more facilities of the language, and its blocks are delineated for emphasis. The output is shown after.

```
JOB
URN, EILEEN DOVER, PRINT BINOM COEFFS
COMPILER AB
```

```
begin

  comment tabulate the binomial coefficients      c
  iCj, with i down the left margin and j across   c
  the bottom, for i = 0 (1) n
  real fn spec fact (integer p)
  routine spec binom (real name ans, integer p, q)
  integer n ; read (n)
  caption ¢BINOMIAL¢COEFFICIENTS¢
  -> 1 unless n < 0
  caption N¢<¢0¢ ; -> 2
1:begin

    array B (0:n, 0:n)
    integer i, j ; real x
    cycle i = 0, 1, n
    |   newline
    |   print (i, 2, 0)
    |   cycle j = 0, 1, i
    |   |   binom (x, i, j)
    |   |   B(i, j) = x
    |   |   print (B(i, j), 6, 0)
    |   L_ repeat
    L_  repeat
    newline ; spaces (3)
    cycle j = 0, 1, n
    |   print (j, 6, 0)
    L_  repeat
    stop
    end

  routine binom (real name ans, integer p, q)

    comment puts pCq in ans
    ans = fact(p)/(fact(q)*fact(p-q))
    end

  real fn fact (integer p)

    comment result = p !
    result = 1 if p = 0
    if p ≠ 0 then result = p*fact(p-1)
    end

2:end of program

5
```

```
***Z
```

```
                [title etc (supervisor)]

        1     BEGIN BLOCK  NO = 91  ADDRESS =00115050
        9       BEGIN BLOCK  NO = 94  ADDRESS =00116000
       27       END BLOCK  OCCUPIES 192 LOCATIONS
       28       BEGIN ROUTINE <BINOM>  NO = 93  ADDRESS =00121010
       31       END ROUTINE <BINOM>  OCCUPIES 52 LOCATIONS
       32       BEGIN REAL FN <FACT>  NO = 92  ADDRESS =00121660
       36       END REAL FN <FACT>  OCCUPIES 43 LOCATIONS
       37     END BLOCK  OCCUPIES 350 LOCATIONS
PROGRAM ENTERED

                        [newpage]

BINOMIAL COEFFICIENTS

  0     1
  1     1     1
  2     1     2     1
  3     1     3     3     1
  4     1     4     6     4     1
  5     1     5    10    10     5     1
        0     1     2     3     4     5

                        [newpage]
                [logging information (supervisor)]
```