WORKSHEET FILE FORMAT

FROM LOTUS

APPENDIX B - THE FORMULA COMPILER

161 First Street
Cambridge, Massachusetts 02142
(617) 492-7171

Electronic Edition, December, 1984

APPENDIX B: The Formula Compiler

This appendix describes the internal workings of the formula compiler. The compiler transforms an ASCII string of characters representing a formula to its Reverse Polish code. The basic algorithm utilizes and SR parser (SR = shift and reduce). The aim of the parser is to apply a set of reduction rules which embody the syntax of the compiler to an input string. Formula code is compiled to a temporary buffer.

Lexicon Analysis

A lexical analyzer breaks up the input string into lexical units called tokens. A token is a substring of the original input string operand, operator, or special symbol (such as comma, parentheses, etc.) In addition, the lexical analyser supplies two special tokens, "beginning of formula" (boform) and "end of formula" (eoform), to facilitate the compilation process. The lexical analyzer identifies and processes literals (both number and string), cell and range references, operators, and function calls. It assigns a unique code to each distinct operator, function, or type of operand.

A function with no arguments is treated like a number.

Syntax Analysis

The syntactical analysis of a formula is accomplished by processing a list of tokens in left-to-right order. A stack called the syntax is also used during the syntactical scan. The basic algorithm is as follows:

Repeat the following steps:

1) Get the next token

2) If the token is a literal or cell reference:
   a) Push the number code on the syntax stack
   b) Push the number code on the syntax stack

3) If the token is a range reference:
   a) Compile code to push the range reference

b) Push the range code on the syntax stack

4) Otherwise push the token code for the token on the syntax stack.

For each syntax rule, if the pattern on the top of the syntax matches the rule pattern take the action associated with the rule and start scanning from the beginning for any additional rules which may apply.

When a token code is pushed on the syntax stack, an additional word of zeros is also pushed on the stack. This is used when compiling function calls to hold the function's argument count.

Rule Matching

A relatively small number of rules are used to process formulas of arbitrary complexity. If a rule matches the top of the syntax stack, then the compiler takes a specific action and rule scanning starts again with the first rule. Each rule matches certain patterns on the syntax stack. A typical rule might be: if the top of the stack is the token for right parenthesis, and the next-to-top is a number, and the second form the top is a left parenthesis, then pop the top three items from the syntax stack and push the number on the syntax stack.

This rule can be more succinctly represented as:

Stack

Before	After	Action
)		
number		
(	number	none

The Rules

The following are the syntax rules used to process formulas. Note that the order of the rules is important. The rules for compilation of operators used additional tables which assign a precedence number and opcode to each legal unary and binary operator. Thus, for example, there is a single token code for minus sign (-), but there are two opcodes one for unary minus and one for binary minus. In addition, these two operators, while lexically identical, also have different precedence. In general, operators of higher precedence will be performed before operators of lower precedence are performed left-to-right. All special operators (boform, eoform, parentheses, comma, etc.) are implicitly assigned a precedence of zero.

Rule 1 Termination test

Stack

Before	After	Action

eoform		Output a return code to compile buffer
number		Return, indicating successful compile
boform		

Rule 2 Function argument processing

Stack Before	After	Action
'		Error if range argument illegal for
number or range		function.
(	(	Increment argument count on stack
function	function	

Rule 3 Process final function argument

Stack Before	After	Action
)		Error if range argument illegal for
number or range		function.
(		Increment argument count on stack
function	number	Compile function opcode
		If list function, compile argument
		count; otherwise error is wrong
		argument count.

Rule 4 Parenthesis removal

Stack Before	After	Action

)			Compile parenthesis opcode
number			
(	number		
operator	operator		

Rule 5 Binary operators

Stack			
Before	After		Action
op2			If binary op<binary op, rule does
number			not match. Otherwise, compile opcode
op1	op2		for operator op1.

Rule 6 Unary operators

Stack			
Before	After		Action
op2			I unary op<binary op, rule does
number	op2		not match. Otherwise, compile opcode.
op1	number		for operator op 1.

Rule 7 Error detection

Stack			
Before	After		Action
eoform			Return indicating unsuccessful compile

Table 9 Operator Precedence Table

Operator	Unary Precedence	Binary Precedence
+	6	4
-	6	4
*	na	5
/	na	7
^	na	3
=	na	3
< >	na	3
< =	na	3
> =	na	3
<	na	3
>	na	3
#and#	na	1
#or#	na	1
#not#	2	na

Example:

Using the above rules, we can now see how a particular formula is compiled. Let us consider the following formula:

```
3+5*6
```

This is broken up by the lexical analyzer into seven tokens.

```
boform
```

```
3
+
5
*
6
eoform
```

The syntax scans proceed as follows until a matching rule is found:

Stack

boform	number	+	number
	boform	number	+
		boform	number
			boform

Compile buffer

push 3	push 3	push 3
		push 5

At this point, rule 5 is invoked, but since the precedence of boform is zero, no action is taken.

Stack

*	number
number	*
+	number
number	+
boform	number
	boform

Compile buffer

push 3	push 3
push 5	push 5
	push 6

At this point, since the binary precedence of + is lower than the binary precedence of *, rule 5 does apply, and the opcode for * is compiled. The stack is reduced by replacing number * number by number and scan is made, but no further rule applies.

Stack

number	eoform
+	number
number	+
boform	number
	boform

Compile buffer

push 3	push 3
push 5	push 5
push 6	push 6

Rule 5 applies again, and the opcode for + is compiled, reducing the stack to boform, number, eoform. Rescanning finds a match on rule 1 which compiles a return opcode and terminates. The final compiled code is thus:

```
push 3
push 5
push 6
```

```
 *
 +
 return
```

A Note on the Decompiler

The algorithm for the formula decompiler was taken verbatim from:

Writing Interactive Compilers and Interpreters, P.J. Brown, John Wiley and Sons, 1979. See chapter 6.2. The algorithm itself is described on pages 216 and 217.

This algorithm is also described in the following article.

More on the Re-creation of Source Code from Reverse Polish, P.J. Brown, Software Practice and Experience, Vol 7, 545-551 (1977).

WORKSHEET COLUMN DESIGNATORS

Most records within the 1-2-3 Condensed Worksheet format are specified with column/row designators (for example, column 0, row 0 equals A1). When determining the column designator, the table below will help make conversion easier.

Column	Hex	Dec	Column	Hex	Dec	Column	Hex	Dec
A	0	1	BA	34	52	DA	68	104
B	1	1	BB	35	53	DB	69	105
C	2	2	BC	36	54	DC	6A	106
D	3	3	BD	37	55	DD	6B	107
E	4	4	BE	38	56	DE	6C	108
F	5	5	BF	39	57	DF	6D	109
G	6	6	BG	3A	58	DG	6E	110
H	7	7	BH	3B	59	DH	6F	111
I	8	8	BI	3C	60	DI	70	112
J	9	9	BJ	3D	61	DJ	71	113
K	A	10	BK	3E	62	DK	72	114
L	B	11	BL	3F	63	DL	73	115
M	C	12	BM	40	64	DM	74	116
N	D	13	BN	41	65	DN	75	117
O	E	14	BO	42	66	DO	76	118
P	F	15	BP	43	67	DP	77	119
Q	10	16	BQ	44	68	DQ	78	120
R	11	17	BR	45	69	DR	79	121
S	12	18	BS	46	70	DS	7A	122
T	13	19	BT	47	71	DT		

7B	123						
U		14	20	BU	48	72	DU
7C	124						
V		15	21	BV	49	73	DV
7D	125						
W		16	22	BW	4A	74	DW
7E	126						
X		17	23	BX	4B	75	DX
7F	127						
Y		18	24	BY	4C	76	DY
80	128						
Z		19	25	BZ	4D	77	DZ
81	129						
AA		1A	26	CA	4E	78	EA
82	130						
AB		1B	27	CB	4F	79	EB
83	131						
AC		1C	28	CC	50	80	EC
84	132						
AD		1D	29	CD	51	81	ED
85	133						
AE		1E	30	CE	52	82	EE
86	134						
AF		1F	31	CF	53	83	EF
87	135						
AG		20	32	CG	54	84	EG
88	136						
AH		21	33	CH	55	85	EH
89	137						
AI		22	34	CI	56	86	EI
8A	138						
AJ		23	35	CJ	57	87	EJ
8B	139						
AK		24	36	CK	58	88	EK
8C	140						
AL		25	37	CL	59	89	EL
8D	141						
AM		26	38	CM	5A	90	EM
8E	142						
AN		27	39	CN	5B	91	EN
8F	143						

AO	28	40	CO	5C	92	EO	144
AP	29	41	CP	5D	93	EP	145
AQ	2A	42	CQ	5E	94	EQ	146
AR	2B	43	CR	5F	95	ER	147
AS	2C	44	CS	60	96	ES	148
AT	2D	45	CT	61	97	ET	149
AU	2E	46	CU	62	98	EU	150
AV	2F	47	CV	63	99	EV	151
AW	30	48	CW	64	100	EW	152
AX	31	49	CX	65	101	EX	153
AY	32	50	CY	66	102	EY	154
AZ	33	51	CZ	67	103	EZ	155

(CONTINUED)

Column	Hex	Dec	Column	Hex	Dec
FA	9C	156	HA	DO	

208					
	FB	9D	157	HB	D1
209					
	FC	9E	158	HC	D2
210					
	FD	9F	159	HD	D3
211					
	FE	AO	160	HE	D4
212					
	FF	A1	161	HF	D5
213					
	FG	A2	162	HG	D6
214					
	FH	A3	163	HH	D7
215					
	FI	A4	164	HI	D8
216					
	FJ	A5	165	HJ	D9
217					
	FK	A6	166	HK	DA
218					
	FL	A7	167	HL	DB
219					
	FM	A8	168	HM	DC
220					
	FN	A9	169	HN	DD
221					
	FO	AA	170	HO	DE
222					
	FP	AB	171	HP	DF
223					
	FQ	AC	172	HQ	EO
224					
	FR	AD	173	HR	E1
225					
	FS	AE	174	HS	E2
226					
	FT	AF	175	HT	E3
227					
	FU	BO	176	HU	E4
228					

	FV	B1	177	HV	E5
229					
	FW	B2	178	HW	E6
230					
	FX	B3	179	HX	E7
231					
	FY	B4	180	HY	E8
232					
	FZ	B5	181	HZ	E9
233					
	GA	B6	182	IA	EA
234					
	GB	B7	183	IB	EB
235					
	GC	B8	184	IC	EC
236					
	GD	B9	185	ID	ED
237					
	GE	BA	186	IE	EE
238					
	GF	BB	187	IF	EF
239					
	GG	BC	188	IG	FO
240					
	GH	BD	189	IH	F1
241					
	GI	BE	190	II	F2
242					
	GJ	BF	191	IJ	F3
243					
	GK	CO	192	IK	F4
244					
	GL	C1	193	IL	F5
245					
	GM	C2	195	IM	F6
246					
	GN	C3	195	IN	F7
247					
	GO	C4	196	IO	F8
248					
	GP	C5	197	IP	F9

249					
	GQ	C6	198	IQ	FA
250					
	GR	C7	199	IR	FB
251					
	GS	C8	200	IS	FC
252					
	GT	C9	201	IT	FD
253					
	GU	CA	202	IU	FE
254					
	GV	CB	203	IV	FF
255					
	GW	CC	204		
	GX	CD	205		
	GY	CE	206		
	GZ	CF	207		

ANALYSIS OF 1-2-3 WORKSHEET FILE

The worksheet shown below was created in 1-2-3 and saved to disk.

Key:

A2..A5 Named Range (code 11)

EXAMPLE A2: Label (code 15)

100 A3: Integer (code 13)

12.5 A4: Number (code 14)

87.5 A5: Formula (+A3-A4)

(code

16)

The example shown below is a partial hex dump of this worksheet file. By reading each record header, you can determine the type of record you are encountering. The record header will also tell you the length of that follows the header. By analyzing the record header, you can read the records you want and skip unrelated records.

```
    362B:0100                                 06 00 08 00 00 00
00 00 00 00
    362B:0110        04 00 2F 00 01 00  01 02 00 01 00 FF
03 00 01 00
    362B:0120        00 04 00 01 00 00  05 00 01 00 FF 07
00 1F 00 00
    362B:0130        00 01 00 71 00 09  00 08 00 14 00 00
00 00 00 00
    362B:0140        00 00 00 00 00 00  00 04 00 04 00 48
00 00 0B 00
    362B:0150        18 00 54 45 53 54  00 00 00 00 00 00
00 00 00 00
    362B:0160        00 00 00 00 01 00  00 00 04 00 18 00
19 00 00 FF
    362B:0170        FF 00 00 FF FF 00  00 FF FF 00 00 FF
FF 00 00 FF
    362B:0180


    362B:05C0
    362B:05D0        00 00 00 00 00 00  00 00 00 00 00 00
00 00 00 00
    362B:05E0        00 00 00 00 00 00  00 00 00 00 00 00
00 00 00 00
    362B:05F0        00 00 00 00 71 71  01 00 0F 00 0E 00
FF 00 00 01
    362B:0600        00 27 45 58 41 4D  50 4C 45 00 0D 00
```

```
07 00 FF 00
    362B:0610        00 02 00 64 00
    362B:0620                          10 00 1B 00 FF 00
00 04 00 00
    362B:0630        00 00 00 00 E0 55  40 0C 00 01 00 80
FE BF 01 00
    362B:0640        80 FF BF 0A 03
```