THE

ABL

PROGRAMMING LANGUAGE

SYNTAX EDITION

BRYAN RAMIREZ

ABL v0.8
(Array-Based Language 0.8)

QUESTIONS

Why did I create ABL?

I created ABL as a year final exam (2021).
After everything I've gone through, I've decided to take it upon myself to write a language using the C programming language. Yes, C is awesome!!! I don't intend it to be an actual programming language like Java, Python, or C++, but rather, a toy around logical language. Its syntax WAS NOT inspired by BrainFuck. I didn't know much about BrainFuck at the time. After I got done sorting my program's syntax, I decided to do some research. Little did I know, BrainFuck was on to the same thing, Array Logic. Its syntax is very much similar to mine (with the +, -, >, <, [, and]). However, my language consists of the same stuff you see in Brainfuck along with other

features I've added (Manual Array Length Memory Allocation, formatted printing, multi-operator, indexing, and soon-to-come control flow (Branching: If statements))

What is the purpose of ABL?

The purpose of ABL is to help with algorithmic thinking (step-by-step thinking). How? Because of how limited my language is, it will force people to apply some methodic and strategic thinking.

What will this guide show me?

This guide is **NOT** an in-depth guide in regards to the language's structure, background definition, and inner workings. This guide will show you the proper usage of each operation available in the ABL language.

PREFACE

ABL (Array-Based Language) is a language based on the concept of an array. What is an array? In short, an array is a set of values of n length. N depends on the size of the array (N = length of the array).

Examples of an array:
An array of integers/numbers: {45, 3, 6, 34, 78, 2, 90} (N = 7)
An array of characters: {T, A, !, #, &, L, K, B, P, H, G, ~, *} (N = 13)
An array of fractional numbers (floats): {1.5, 3.5, 2.5, 7.8., 6.5} (N = 5)
An array of characters (STRING): "Hello, world!" (N = 13)

In programming, arrays/sets are used for a bunch of things (e.g. storing data, creating strings, strings of the same data type). If arrays are used to store values, how do I access those values? In programming, arrays have index values that associate with the values of the array. These index values are used to access the values of an array. Index values start at 0 and increment and associate with the values of the array until the end.

Examples of indexing:

Array =	{56,	34,	60,	12,	32,	78,	99,	124,	4,	29,	200, 545}
Index =	0	1	2	3	4	5	6	7	8	9	10 11

Value at Array Index 5 would be 78
Value at Array Index 2 would be 60
Value at Array Index 0 would be 56
Value at Array Index 1 would be 34
Value at Array Index 3 would be 12
Value at Array Index 7 would be 124

Table of Contents

1.1: The [SIZE] Operator

In ABL, you allocate the number of bytes of memory (by bytes I mean cells to store data) to the array via the [] operator. This operator should be the first thing in your program as the array has not yet been allocated memory and therefore can not store any data. Trying to work with a NULL (0, void, empty, nothing) array can result in an index error (aka Out of Bounds). The [] operator allocates SIZE amount of bytes to the array with all values set to 0. The [] operator can be used more than once, however, if you decide to use this operator once again; all memory and data previously allocated/stored will be deallocated and overwritten with the new size of the array along with all values being reset to 0.

Example:

[20] ; Allocated 20 bytes of memory to the array

[30] ; Allocated 30 bytes of memory to the array

[5] ; Allocated 5 bytes of memory to the array

[10] ; Allocated 10 bytes of memory to the array

1.2: Basic Operators (>, <, +, -, ;, C/D/S, $)

(;) Comment operator
The “;” operator allows you to comment on your code. Anything on the line after the comment ooperator will be ignored.

```
[5] ; This English sentence will be ignored [5] <- including this

; [50]  <- this is being ignored (same with this entire sentence)
```

(>. <) Right and Left Shift operators
The > and < operators allow you to navigate through the array. In ABL, you start at index 0. The > operator increments the current index you are at by one while the < decrements the current index you are at by one.
Let’s say you were at index 3 and wanted to go to 1 and then to 6, it would look like “<<>>>>>”. You’re decrementing the current index two times which lands you on index 1 and then you’re proceeding to increment the current index 5 times which lands you on index 6.

Code Example:

```
[5] ; Allocate 5 bytes of memory (SIZE)

>>  ; Going from index 0 to index 2
<<> ; Going from index 2 to index 0 to index 1

>>  ; Going from index 1 to index 3
```

Fault Code:

```
[5] ; Allocate 5 bytes
>>>>>  ; Going from index 0 to index 5 which is out of bounds
```

```
Array Visualized:    [0] [0] [0] [0] [0]
Indexes:               0   1   2   3   4   5??
```

Notice How 5 is out of bounds
Because of this, an out-of-bounds error will occur and your process will terminate

(+, -) Increment and Decrement operators
The + and - operators allow you to modify the value at the current index. The + operator increments the value at the current index by one while the - operator decrements the value at the current index by one.

Let's say the value at the current index you're at is 10 and you want the value to go from 4 to 13, it would look like "------+++++++++". Here, you're subtracting 10 by one 6 times which lands you on 4, and then adding 4 by one 9 times which lands you on 13.

Code Example:

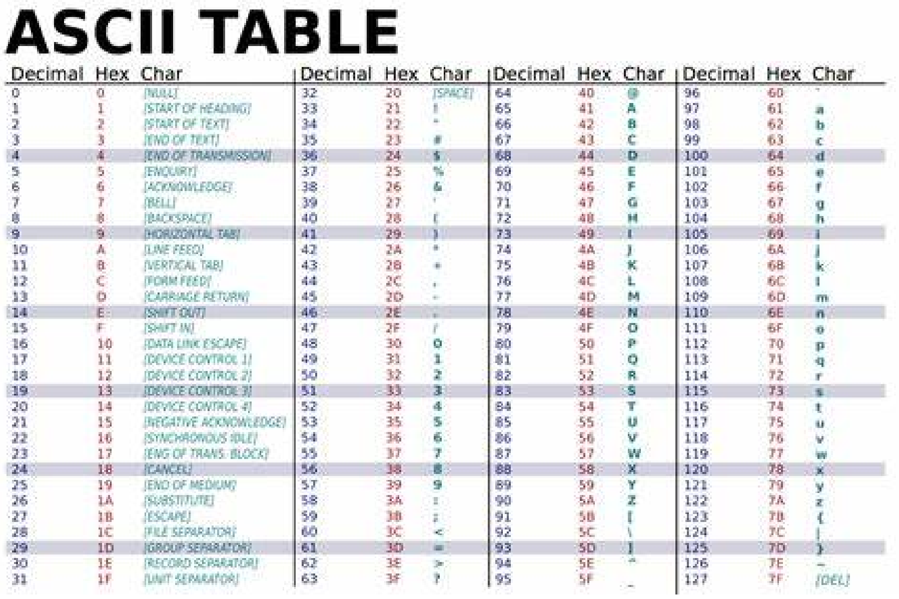
```
[5]    ; Allocate 5 bytes of memory

>>     ; Go from index 0 to index 2

+++    ; Increment the value at index 2 by 3
```

; Array Visualized: [0] [0] [3] [0] [0]
; Indexes: 0   1   2   3   4

(C/D/S) Formatted printing

ABL follows a 7-bit ASCII conversion. C/D/S are ways you can print to the console. C is short for character, C prints out the character conversion of the value at the current index. D is short for decimal, D prints out the integer/decimal conversion of the value at the current index. S is short for string, S prints out the string version of the array (Will print out all character converted decimals up to where the array index is 0 to the console).

Refer to the ASCII Chart (Google Search):



ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Example Code:

```
[4]          ; Allocate 4 bytes of memory

+>++         ; Increment Value at index 0, Go to index 1 and increment value by 2
>+++>++++
<<<
d>d>d>d
```

Console Output:

```
1234
```

Printing Out Characters Code Example:

```
[3]                         ; Allocate 3 bytes

++++++++++++++++++++
++++++++++++++++++++
++++++++++++++++++++
++++++++++++                ; increment value at index 0 by 72

C                           ; (refer to ASCII Chart (The decimal to character conversion of 72))
```

Console Output:

```
H
```

($) Input operator
The $ operator grabs input from the user and overwrites the value at the current index with the input.

Code Example:

```
[3]

$>$>++++++++++  ; Refer to ASCII Conversion Decimal: 10 <-> Char
s               ; Print String
```

Console Output:

```
Hi
Hi
```

2.1: Complex Operators (#INDEX#, {SYNTAX})

(#INDEX#) Go-to Index Operator

This operator is an advantage! This operator allows you to go to a specific index in the allocated array. Index Bounds still apply to this operator.

Code Example:

```
[5]
+++++          ; Add 5 to value at index 0
#4#            ; Go to index 4
+++++          ; Add 5 to value at index 4
+++++          ; Add another 5 to value at index 4
<              ; Go from index 4 to index 3
+++++-         ; Add 5 to value at index 3 and decrement it by one
#0#            ; Go back to index 0
d>d>d>d>d      ; Print decimal conversions of the values from index 0 to index 4
```

```
; Array Visualized:  [5] [0] [0] [4] [10]
; Indexes:            0   1   2   3    4
```

Console Output:

```
500410
```

({SYNTAX}) Non-Zero Loop Operator

This is the only looping mechanic in this language. SYNTAX is the code you want to be looped. The code within the curly braces ({ }) will only loop/run if the current index value is NOT zero. For the code within the loop to run, the value at the current index has to be non-zero. For the code to keep on looping the value at the current index has to be non-zero. The only operator that **CAN'T** be in the non-zero loop operator is the comment operator (;).

Code Example:

```
[3]

+++++>   ; Increment value at index 0 by 5 and go from index 0 to index 1
++++++++++<  ; Increment value at index 1 by 10 and go from index 1 to index 0
{
        d->c<
}

; Within the loop. Since the value at index 0 is NOT zero this code will loop
; This code prints out the value at index 0, decrements it, goes from index 0
; to index 1, print the character conversion of the value at index 1, and finally
; go from index 1 to index 0.
; it will do this until the value at index 0 is 0 as seen in the code
```

Console Output:

```
5
4
3
2
1
```

2.2: Multi-Operator

The multi-operator is an operator that takes TWO parameters. It accepts an operator you want to perform and the value you want to use. The multi-operator performs the operator with the value at the current index **AND** the value you provide it with (Operator Number/Value/0). If the value given is 0 then it will only perform the operator with the value at the current index.
The operators that you can perform using the multi-operator are:

+ - * / %
> < #

2.3: Multi-Operator Operations

(+): (Example: (+5)) Passing in this operator will add the number given to the value on the current index

(-): (Example: (+5)) Passing in this operator will subtract the number given from the value on the current index

(*): (Example: (*5)) Passing in this operator will multiply the number given to the value on the current index

(/): (Example: (/5)) Passing in this operator will divide the value given to the value on the current index

(%): (Example: (%5)) Passing in this operator will overwrite the value at the current index with the result of a modulus operation on the number given and the value at the current index

(>): (Example: (>5)) Passing in this operator will increment the index by the number given

(<): (Example: (<5)) Passing in this operator will decrement the index by the number given

(#): (Example: (#5)) Passing in this operator will overwrite the value at the current index with the value at the given index. 0s are accepted.

Code Example:

```
[5]        ; Allocate 5 bytes of memory

(+72)     ; Increment the value at index 0 by 72
>
(+105)  ; Increment the value at index 1 by 105
>
(+5)      ; Increment the value at index 2 by 5
(*4)       ; Multiply the value at index 2 by 4
>           ; Go from index 2 to index 3
(#2)       ; Copy from index 2
(-13)      ; Subtract the value at index 3 by 13
#0#         ; Set the index to 0
c>c>d>dd   ; print the character conversion of index 0 and index 1, go to index 2,
; print the decimal conversion, go to index 3, print the decimal conversion 2 times
```

Console Output:

```
Hi2077
```

3.1: Advice

My advice to you guys is to only use this language to help you with any algorithmic problems. This language is very limited which forces you to use your brain and apply some thinking into your code. I don't have much to say other than be responsible for how you tamper with this language.

3.2: ABL Programs

A program that raises the base to the power of the exponent

```
[10]
;                Result
;                    \/      \/ <- Exponent
(+62)>(+32)>+>(+4)>(+10)<{
  <(*3)>-
}  ;  /\ <- Base
#0#c>c>d#4#c ; <- Print
```

Console Output:

```
81
```

Code producing Triangle

```
[10]
(+10)>(+42)>(+32)>>(+7){
  #5#(#4)-
  {<<<c>>>-}
  #3#(#4)(-8)
  {<<c>c>+}
  <<<c#4#-
}#0#c
```

Console Output:

```
      *
     * *
    * * *
   * * * *
  * * * * *
 * * * * * *
* * * * * * *
```

THANK YOU!