PHASE 1: MiniBASIC DESIGN
Subhajit Sahu (2018801013)

MiniBASIC is a subset of QuickBASIC. It is thus a case-insensitive procedural programming language that is easy to program with. Unlike C, it has no “main” function, and statements can be executed right away. Also functions cannot access global variables by default, unless specified with the **SHARED** keyword.

Here are a few quick examples:

```
‘this is a comment
PRINT “Monsoon 2019”
```

```
Monsoon 2019
```

```
CLS
INPUT “Name: “, name$
PRINT “Hello ”; name$
‘CLS => clear screen
‘name$ => name is a STRING
```

```
Name: Raja
Hello Raja
```

```
CLS
INPUT “N: “, n%

sum% = 0
FOR i% = 1 TO n%
sum% = sum% + i% ^ 3
NEXT
PRINT “Sigma n^3: “; sum
‘n% => n is an INTEGER
```

```
N: 3
Sigma n^3: 36
```

```
DECLARE FUNCTION isprime%(n AS INTEGER)
DIM n AS INTEGER

CLS
INPUT “N: “, n

IF isprime%(n) = 1 THEN
PRINT n; “ is prime”
ELSE
PRINT n; “ is not prime”
END IF


FUNCTION isprime%(n as INTEGER)
DIM i as INTEGER

isprime% = 0
FOR i = 2 TO n - 1
IF n MOD i = 0 THEN EXIT FUNCTION
NEXT

isprime% = 1
END FUNCTION
```

```
N: 7
7 is prime
```

DATA TYPES

Datatypes in MiniBASIC can be specified either through variable name suffixes or by using the **DIM** keyword. Single and multidimensional arrays can be specified using the **DIM** keyword, and may later be resized with **REDIM** or freed with **ERASE**. Arrays is MiniBASIC can start with 1, unlike C. The size of a particular dimension of the array can be found using **UBOUND** function.

SUFFIX	**TYPE NAME**
%	INTEGER
&	UNSIGNED
!	SINGLE
#	DOUBLE
$	STRING
@	CHARACTER
?	BOOLEAN

```
‘define a string using suffix
name$ = “ajit doval”

‘define a double using DIM
DIM article AS DOUBLE
article = 370.0

‘define a 1D integer array
DIM votes(29) AS INTEGER
votes(1) = 34

‘define a 3D single array
DIM heat(10, 10, 10) AS SINGLE
heat(10, 10, 10) = 0.8

‘resize votes array
REDIM votes(28)
```

ARITHMETIC OPERATORS

The following arithmetic and boolean operators can be used:

OPERATOR	**EXAMPLE**
Exclusive-Or	a **XOR** b

Or	a **OR** b
Modulus	a **MOD** b
Implication	a **IMP** b
Equivalence	a **EQV** b
And	a **AND** b
Not	**NOT** a
Integer divide	a \ b
Power	a **^** b
Others (+ - * / = <> < > <= >=)	

CONTROL STATEMENTS

IF-THEN-ELSE, which is used for conditional execution / branching, can be used with both single line and block formats. Looping is possible through the use of the convenient **FOR-NEXT** loop. Other alternatives include **WHILE-WEND**, and **DO-LOOP** which can be used for either entry or exit control. Ternary operator is achievable through a single line **IF**.

```
‘single line if
IF 1 = 1 THEN PRINT “Math wins” ELSE PRINT “Random wins”

‘block if
IF 0 = 1 THEN
PRINT “0 = 1”
ELSEIF 1 = 1 THEN
PRINT “1 = 1”
ELSE
PRINT “Neither”
END IF

‘for loop
FOR i = 1 TO 10 STEP 2
PRINT i
NEXT
```

```
‘exit for loop
FOR i = 1 to 10 STEP 2
PRINT i
IF i > 5 THEN EXIT FOR
NEXT

‘while loop
i = 1
WHILE i <= 10
PRINT i
i = i + 2
WEND

‘do loop (entry control)
i = 12
DO WHILE i <= 10
PRINT i
IF i > 5 THEN EXIT DO
LOOP

‘do loop (exit control)
i = 12
DO
PRINT i
LOOP UNTIL i > 10

‘ternary condition
i = 12
IF i <= 10 THEN ok = 1 ELSE ok = 0
```

FUNCTIONS

In MIniBASIC, procedures which return a value are called **FUNCTION**s, and which do not return any values are called **SUB**routines. Arguments to these are passed by reference by default, and can be passed by value using **BYVAL**. The return value of function is set by using the function name as a variable, and setting its value (before exit). Function names require a type suffix in order to specify the returned data type. Both subroutines and functions must be declared before being used in the program. Usually function / subroutine definition is placed at the end of the program.

```
DECLARE SUB printlines(n AS INTEGER)
DECLARE FUNCTION countspaces%(s AS STRING)
DECLARE FUNCTION factorial%(n AS INTEGER)

CLS
PRINT “Printing 3 empty lines”
printlines 3

name$ = “harry kumar potter”
PRINT “Spaces in “; name$; “: “; countspaces%(name$)

‘a recursive function
num% = 6
PRINT “Factorial of”; n; “: “; factorial%(num%)


SUB printlines(n AS INTEGER)
FOR i% = 1 TO n
PRINT
NEXT
END SUB

FUNCTION countspaces%(s AS STRING)
count% = 0
FOR i% = 1 TO LEN(s)
IF MID$(s, i%, 1) = “ “ THEN count% = count% + 1
NEXT
countspaces% = count%
END FUNCTION

FUNCTION factorial%(n AS INTEGER)
factorial% = 1
IF n <= 1 THEN EXIT FUNCTION
factorial% = n * factorial%(n - 1)
END FUNCTION
```

I/O ROUTINES

Reading from, and writing to files can be done using a very similar syntax of **INPUT** and **PRINT**. All file operations are performed through file numbers. A file needs to be opened before reading or writing to it, and it must be closed after all such operations are complete in order to ensure properly saved on disk.

```
PRINT “Vote count:”
OPEN “votes.csv” FOR INPUT AS 1
WHILE NOT EOF(1)
INPUT #1, state$, count%
PRINT state$; “ provided”; count%; “ votes”
WEND
CLOSE #1
PRINT

OPEN “expenses.txt” FROM APPEND AS 2
PRINT #2, “butter”, 450
PRINT #2, “cashew”, 950
CLOSE #2

PRINT “Alice in Wonderland:”
OPEN “alice.txt” FOR INPUT AS 2
DO WHILE NOT EOF(2)
LINE INPUT #2, line$
PRINT line$
LOOP
CLOSE #2
PRINT
```

PTO

MACRO SYNTAX

Here is the macro syntax of MiniBASIC expressed in context-free grammar:

S	**main_stmt S |** ϶
main_stmt	**declare | sub | function | stmt**
declare	**declare_sub | declare_fn**
declare_sub	*DECLARE SUB* **name** *(***lpar***)*
declare_fn	*DECLARE FUNCTION* **name_t** *(***lpar***)*
sub	*SUB* **name** *(***lpar***)* **lstmt** *END SUB*
function	*FUNCTION* **name_t** *(***lpar***)* **lstmt** *END FUNCTION*
lstmt	**stmt** **lstmt |** ϶
stmt	**comment | sub_call | define | assign | io | branch | loop**
sub_call	**name lexpr**
fn_call	**name_t | name_t** *(***lexpr***)*
define	**dim | redim | shared | static | type**
dim	*DIM* **dim_shared ldef1**
dim_shared	*SHARED* **|** ϶
redim	*REDIM* **larr1**
shared	*SHARED* **lpar1**
static	*STATIC* **lpar1**
type	TYPE **name** **ldef1_blk** END TYPE
assign	**let | const | assign_dir**

let	*LET* **assign_dir**
const	*CONST* **assign_dir**
assign_dir	**var_t** = **expr**
io	**input** | **print** | **open** | **close**
input	**input_cmd** | **input_file**
input_cmd	*INPUT* **prompt** **lvar**
input_file	*INPUT* **fnum_h**, **lvar**
prompt	**string**, | **string**; | ϶
print	**print_cmd** | **print_file**
print_cmd	*PRINT* **print_fmt** **print_lexpr**
print_file	*PRINT* **fnum_h**, **print_fmt** **print_lexpr**
print_fmt	*USING* **string**; | ϶
print_lexpr	**expr**, **print_lexpr** | **expr**; **print_lexpr** | ϶
open	**open_long** | **open_short**
open_long	*OPEN* **fname** **fmode1** **facc** *AS* **fnum**
open_short	*OPEN* **fmode2**, **fnum_h**, **fname**
fname	**expr**
fmode1	*FOR* **fmode1_type** | ϶
fmode1_type	*OUTPUT* | *INPUT* | *RANDOM* | *BINARY* | *APPEND*
facc	*ACCESS* **facc_type** | ϶
facc_type	*READ* | *WRITE* | *READ WRITE*
fmode2	*“O”* | *“I”* | *“R”* | *“B”* | *“A”*
close	*CLOSE* **lfnum1**
branch	**branch_dir** | **branch_cond**
branch_dir	**goto** | **gosub** | **return** | **exit**
goto	GOTO **label**

gosub	GOSUB **label**
return	RETURN | RETURN **label**
exit	EXIT **exit_from**
exit_from	DO | FOR | FUNCTION | SUB
branch_cond	**if** | **select**
if	**if_then** | **if_blk**
if_then	*IF* **cond then_stmt else_stmt**

if_blk	IF **cond then_blk**
	lelseif_blk
	else_blk
	endif

then_stmt	*THEN* **stmt**
then_blk	*THEN* **lstmt**
lelseif_blk	**elseif_blk** **lelseif_blk** | ϶
elseif_blk	*ELSEIF* **cond** **lstmt** | ϶
else_stmt	*ELSE* **stmt** | ϶
else_blk	*ELSE* **lstmt** | ϶
endif	*ENDIF* | *END IF*

select	*SELECT CASE* **expr**
	lcase
	END SELECT

lcase	**case_expr** **lcase**? | **case_else**
case_expr	*CASE* **expr** (*TO* **expr**)? **lstmt**
case_else	*CASE ELSE* **lstmt**

loop	**for** | **while** | **do**
for	*FOR* **var** = **expr** *TO* **expr** (*STEP* **expr**)? **lstmt** *NEXT* **var**?
while	*WHILE* **cond** **lstmt** *WEND*
do	**do_entry** | **do_exit**
do_entry	*DO* (*WHILE* | *UNTIL*) **cond** **lstmt** *LOOP*
do_exit	*DO* **lstmt** *LOOP* (*WHILE* | *UNTIL*) **cond**
name_t	**name dtype_s**
sym	**name** | **name** *()*
sym_t	**name_t** | **name_t** *()*
var	**name** | **name** *(***lexpr1***)*
var_t	**name_t** | **name_t** *(***lexpr1***)*
arr_t	**name_t** *(***lexpr1***)*
par	**sym** *AS* **dtype_n** | **sym_t**
def	**var** *AS* **dtype_n** | **var_t**
larr	**arr_t**, **larr1** | **arr_t** | ϶
larr1	**arr_t**, **larr1** | **arr_t**
lvar	**var**, **lvar1** | **var** | ϶
lvar1	**var lvar1** | **var**
lpar	**par**, **lpar1** | **par** | ϶
lpar1	**par**, **lpar1** | **par**
ldef	**def**, **ldef1** | **def** | ϶

ldef1	def, ldef1 | def
ldef_blk	def ldef1_blk | def | ϶
ldef1_blk	def ldef1_blk | def
lexpr	expr, lexpr1 | expr | ϶
lexpr1	expr, lexpr1 | expr
fnum	fnum_h | num
fnum_h	#num
lfnum	fnum, lfnum1 | fnum | ϶
lfnum1	fnum, lfnum1 | fnum
dtype_n	*INTEGER* | *UNSIGNED* | *SINGLE* | *DOUBLE* | *STRING* | *CHAR* | *BOOLEAN*
dtype_s	% | & | ! | # | $ | @ | ? | ϶
cond	expr
bin_log	*AND* | *OR* | *XOR* | *IMP* | *EQV*
bin_ari	*MOD*
bin_add	+ | -
bin_mul	* | / | \
bin_pow	^
una_log	*NOT*
una_add	+ | -
expr	expr bin_log expr | expr_1
expr_1	expr bin_ari expr | expr_2
expr_2	expr bin_add expr | expr_3
expr_3	expr bin_mul expr | expr_4
expr_4	expr bin_pow expr | expr_5

expr_5	**una_log expr | expr_6**
expr_6	**una_ari expr | expr_7**
expr_7	**litr | var_t | fn_call | *(*expr*)***
litr	**integer | float | string | boolean**

MICRO SYNTAX

Here is the micro syntax of MiniBASIC expressed in regular expressions:

name	[A-Za-z_]\w*
integer	[-+]?\d+
float	[-+]?([0-9]*[.])?[0-9]+([eE][-+]?\d+)?
string	\”.*?\”
boolean	TRUE|FALSE (i)
comment	\‘.*|REM\s.* (i)

Note: **(i)** stands for ignore case.