GWBASIC vs. Liberty BASIC

Liberty BASIC is designed to be a lot like GWBASIC, but there are differences. Some of these differences are incompatibilities, and some are enhancements.

Here you will find information about:

PRINT
INPUT
Variable Names
Variable Types
Line Numbers
DIM
ELSEIF...ENDIF
File Handles
INPUT$( )
Boolean Operators
TIME$( ) & DATE$( )
Random Numbers
READ, DATA, RESTORE

Print

PRINT is -almost- the same. The comma cannot be used as a formatter.

For example:

```
print a, b, c
```

is valid in GWBASIC, but not in Liberty BASIC

Also, PRINT USING isn't supported, but there is a using(function instead.

```
GWBASIC:          print using "The total is #####.##", ttl
Liberty BASIC:    print "The total is "; using("####.##", ttl)
```

GWBASIC inserts spaces before and after numbers (called padding) when it prints them. Liberty BASIC doesn't pad, but if you want to, you can add padding spaces in your code.

Input

GWBASIC permits this:

```
input "Give me the next number"; n(x)
```

In Liberty BASIC, you must do this:

```
input "Give me the next number"; n
n(x) = n
```

GWBASIC automatically adds a question mark and an extra space onto the end of the prompt (using above as example: Give me the next number? _). Liberty BASIC doesn't do this. For example:

```
input "Give me the next number >"; n
```

appears as:

```
Give me the next number >_
```

using Liberty BASIC. If desired, simply changing the > to a ? will produce the GWBASIC-like effect.

Variable Names

Liberty BASIC accepts GWBASIC variable names. But it also lets you use upper and lower case letters.

someTime and sometime

are considered different variables in Liberty BASIC.

Variable Types

Liberty BASIC only supports two variable types, numeric and string.

The numeric variable holds either an integer or real value, and in this case Liberty BASIC optimizes automatically. If a real loses its non-integer part through some calculation, then Liberty BASIC converts it into an integer to speed up operations. Integers can be enormously large (see factorial.bas). Reals are single precision.

The string variable holds a character string that can be as large as available memory.

NOTE: Variables are actually untyped in Liberty BASIC. A string variable name can contain a numeric value, and a numeric variable name can contain a string. But arrays ARE typed, just as in other BASICs.

Line Numbers

Liberty BASIC lets you use the conventional line numbers if you like. You can also choose not to use them, and use descriptive branch labels instead. For example:

In GWBASIC:

```
10 print "Let's loop!"
20 x = x + 1
30 print x
40 if x < 10 then 20
50 print "Done."
60 end
```

In Liberty BASIC:

```
    print "Let's loop!"
[mainLoop]
    x = x + 1
    print x
    if x < 10 then [mainLoop]
    print "Done."
    end
```

You can see here that instead of jumping to line 20, as in the first example, we can instead jump to [mainLoop]. You can name your branch points and subroutines whatever you like. They need to start and end with [and], respectively, and they can't have any spaces.

Legal branch labels:

```
[mainLoop]
[loop1]
[exceptionHandler]
```

Illegal branch labels:

```
[main loop]
mainLoop
```

DIM

The DIM statement can only dimension ONE array per statement, so instead of:

```
dim a(20), b(20), c(20)
```

do this:

```
dim a(20) : dim b(20) : dim c(20)
```

ELSEIF...ENDIF

Liberty BASIC supports IF...THEN...ELSE, but not the ELSEIF...ENDIF combinations.

File Handles

GWBASIC lets you use only numbers as file handles. Liberty BASIC lets you use numbers, and it lets you use letters also.

```
open "autoexec.bat" for input as #1 ' GWBASIC

open "autoexec.bat" for input as #autoexec   ' Liberty BASIC
```

Now here's the catch. Whenever you reference the file handle, you MUST use have a # in front of it.

```
if eof(1) > 0 then 10020 ' GWBASIC

if eof(#autoexec) > 0 then [reachedEnd]   ' Liberty BASIC
```

Additionally:

```
print # 1, "buffers = 30"   ' this works fine in GWBASIC
         ^--------this extra space not allowed in Liberty BASIC
```

INPUT$()

In GWBASIC, there are two uses for INPUT$().

1) To fetch x number of characters from a sequential file
2) To fetch x number of characters from the keyboard

```
a$ = input$(1, 10) 'Fetch 10 characters from file handle 1
a$ = input$(1) 'Fetch 1 character from the keyboard
```

In Liberty BASIC, there are also two uses for INPUT$().

1) To fetch x number of characters from a sequential file
2) To fetch only 1 character from the keyboard

```
a$ = input$(#1, 10) 'Fetch 10 characters from file handle #1
a$ = input$(1) 'Fetch a single character from the keyboard
```

Using input$(1) to read from the keyboard only works in the main window, and not with any windows created using OPEN.

Boolean Operators

Liberty BASIC supports AND & OR, much like GWBASIC. These are typically used in IF...THEN statements like so:

```
if x < limit and userAbort = 0 then [mainLoop]

if (c$ >= "A" and c$ <= "Z") or (c$ >= "a" and c$ <= "z") then [inRange]
```

However, bitwise ANDing and ORing aren't supported. If numbers are ANDed or ORed, then a Zero is always a Boolean False, and any Non-Zero is always a Boolean True, and 0 and 1 are the only possible return values.

Consider:

```
print 1 and 1          ' produces 1
print 1 and 0          ' produces 0
print 1 or 0           ' produces 1
print 0 or 23          ' produces 1
print -5 and 0         ' produces 1
```

GWBASIC supports bitwise operations such as:

```
print 8 or 16          ' produces 24
print 7 and 15         ' produces 7
```

Liberty BASIC doesn't support this format.

TIME$() & DATE$()

In GWBASIC you get the time or date by using the special variables time$ and date$. For example:

```
print time$              'this produces the current time
print date$              'this produces the current date
```

In Liberty BASIC it looks like this:

```
print time$()            'this produces the current time
print date$()            'this produces the current date
```

Random Numbers

In Liberty BASIC, random numbers between 0 and 1 are generated with the expression RND(1). GWBASIC will produce the same result with that expression. Apparently GWBASIC also lets you substitute just RND, and it assumes the (1) part in this case. Both of these lines produce the same result in GWBASIC.

```
x = int(rnd(1)*10)+1   ' this format works with both BASICs
x = int(rnd*10)+1      ' this format doesn't
```

The second line is not permitted in Liberty BASIC. It will always equal 1, because rnd will be parsed as a numeric variable, and will equal zero, unless it is assigned some other value.

READ, DATA, RESTORE

Even though Liberty BASIC 0.9x supports READ, DATA, and RESTORE, versions 1.x don't yet. This will be added on later.