Plain BASIC Talk
Stack Woes: When things are STACKed against you!
By Ken Lill
July 24, 2021
Version 1.1

The "Plain BASIC Talk" series, written by Ken Lill, originally appeared in 1982, 1983 and 1985 in the "Arcadian" newsletter for the Bally Arcade/Artocade. This new article was written in 2021.

If you are newish to BASIC programming with the either the Bally or Astrocade BASIC cartridges, there are a couple of things you will need to know about the STACK.

The STACK is a dedicated part of memory that is used by both the BASIC program and the Z-80 microprocessor.

They are in different places, but they are used quite similarly. I won't get into the Z-80 STACK because it involves machine language.

But, in BASIC, it can be a GAME CHANGER!!

If you happen to make it overflow, it WILL crash your entire program and may even erase all of it from memory in a FLASH!

There are 2 commands that you will need to watch out for mainly. They are GOSUB and the FOR-TO-NEXT-STEP loop.

Let me explain why these can be bad and how to prevent them from crashing your program.

GOSUB means to go to the line that is numbered after this command and execute the program from there until there is a RETURN command. When the RETURN is found, clear the stack of the information it uses to find out where to go back to.

Example:

```
10 gosub 20;goto  50
20 for a =0to 20
30 next a
40 return
```

The program is executing line 10. It sees that a GOSUB is being asked for. It saves the next 2 bytes on the STACK, which have the address in the program that it must return to when the RETURN is found. Then it jumps to the line 20 and starts doing what that line says. It goes through the loop and then finds the RETURN. It then jumps back and resets the STACK pointer, which is where the STACK goes to find its next position. The stack pointer is now cleared of the information it stored and can continue working.

However, if you do this:

```
10 gosub 20;goto  50
20 for A=0to 20
30 next A;goto 10
40 return
```

it will NEVER get to the return but will continually add the return point to the stack. It will do this until it overflows and doesn't know where to go...CRASH!!

This will take some time to fill up the STACK.

But when you mess up on a FOR-TO-NEXT-STEP loop, that's a different story!

In line 20 above there is a proper loop. Let's see just what happens.

First, the loop is recognized by the program, so it needs to set up the STACK.
It saves the A variable's address. Let's just say it is located @ 20220. That takes up 2 bytes of the STACK space. It sees that the first number in the loop is 0. It puts that number into address 20220. Then it needs to save the value after the TO command. That takes up 2 more bytes. Then it looks to see what the STEP value is. If it is not assigned by the programmer, it sets it to 1 automatically. Another 2 bytes. It then must set up where the address is after that, so it knows where to come to when it encounters the NEXT command. 2 MORE bytes. So in total, it needs to save 8 bytes on the STACK.

Now let's see how this can be dangerous to your program really quick.

Let's change the program above:

```
10 gosub 20;goto  50
20 for A=0to 20;goto 10
30 next A
40 return
```

First off, it saves the 2 bytes for the GOSUB, it goes to line 20; it saves the 8 bytes for the loop, and then goes back to line 10 and does it all over again. Now the STACK will crash 5 times faster!

However, if you go back to the first program, it saves the 2 for the gosub, then the 8 for the loop, but it resets the STACK pointer for the loop when it encounters A has gotten to 21 and it goes back to the start of 8 bytes it was using, goes on to the next command. That is the RETURN. It gets the address after the GOSUB 20 and resets the STACK POINTER back 2 bytes. The Stack is now where it needs to be for it to continue on with the program... GOTO 50.

Also, be cautious about nesting a loop inside another loop.

I know this is a lot to chew on, but as you get further on down the line, it will all make sense to you.

```
10 gosub 20;goto  50
20 for A=0to 20;for B=1to 10; print “HI
30 next b; next A
40 return
```

What this does is start loop A, then starts loop B; does the printing, goes to the next B, finishes the 9 other printings. Then it goes onto loop A and increments A, prints the 10 lines, and continues until A is 21. It RETURNS and then GOTO 50.

IF you get this wrong, thinking that A was first, so it's NEXT should be first....

30 next A; next B

This will be an endless loop. It will do loop A, save the 8 bytes for the B loop, Print “HI” once, then go back until A is satisfied, and tries to do loop B once, and goes through all of this again, never clearing the STACK of the B information.

If you have any questions, please feel free to ask. My email is
kenzre@yahoo.com
Make SURE that you put into the subject line that it's a question about BASIC.