How to make a shared C function and use it in a Gforth program on a Raspberry Pi

This will also work on Debian Jessie (8.3 tested Feb 2016) only issue is the libtool install now needs libtool-bin as well as libtool. This is noted in text below!

The bold italic stuff following a $ in this document is to be typed into command line on Raspberry pi! The example is a simple one but shows all steps starting with installing Gforth and libtool all the way to the final code to use the C function in the Gforth program. What is not in this document is the understanding of what is going on other then a few statements at some junctions.

$ sudo apt-get install gforth

$ sudo apt-get install libtool libtool-bin

(note Debian jessie now needs the libtool-bin and older versions may not need it)

$ sudo apt-get update

Now you have Gforth and libtool installed. You now make the C code as follows:

$ sudo nano afunction.h

int mytest();

Press control x to close nano editor and save file with the y key and enter.

$ sudo nano afunction.c

int mytest() { return 42 ; }

Press control x to close nano editor and save file with the y key and enter.

$ gcc -c -fPIC afunction.c -o afunction.o

This compiles the c code and makes object file also the code is position independent.

$ ar rcs libafunction.a afunction.o

This archives the library. Note the ".a" and the "lib" at the beginning of file name needs to be there!

$ gcc -shared -Wl,-soname,libafunction.so.1 -o libafunction.so.1.0.1 afunction.o

(note this is '-Wl,-soname,libafunction.so.1' with no spaces and after the capital W it is l as in letter)

This makes the final shared library from the object archived object file. Note the white spacing and the "," comma's are all important here check before pressing enter on this command. You should have now these files in your directory : afunction.c, afunction.o, libafunction.a, afunction.h, libafunction.so.1.0.1

$ sudo cp libafunction.so.1.0.1 /usr/lib/ # copy this shared library to where it is stored in system and used.

$ sudo ln -sf /usr/lib/libafunction.so.1.0.1 /usr/lib/libafunction.so

$ sudo ln -sf /usr/lib/libafunction.so.1.0.1 /usr/lib/libafunction.so.1 # This creates a symbolic link to this shared library. This is needed to use in other c programs with the header file. Note these last two commands are very close note the only difference is the major version number added on second command.

$ sudo ldconfig -n /usr/lib/ # I believe this tells the system to index the librarys in directory. This is needed to use in other c programs with the header file.

This copies library files to the location that they are used and the last three commands index and sets up the symbolic link to the shared library for other c code that could use this shared

library via the header file. This means that the library is shared now and sharable with a version of 1.0.1 i believe.
Now for the forth stuff!

```
$ gforth
```

This starts gforth and you are ready to code! Note there is no command prompt in forth but you are given a flashing cursor so the following stuff is simply typed in. You can do this other ways but that is beyond this document.

```
c-library mycfunction          \ note this is simply a file name for this linking process
s" afunction" add-lib          \ now use the shared library
c-function seemytest mytest -- n    \ the -- is needed and indicates to forth how to
```

handle the data type returned from the c function. In this case n means basic interger. Look at the gforth documentation for other data types.

```
end-c-library
```

Now you can use the function as follows:

```
seemytest . 42\ now you see gforth returning 42 from afunction().
```

Note the . is typed in at forth command prompt and it means return the single data value at top of stack to the console.
This now simply returns the value 42 from the data stack. It should be apparent that now you can execute any c function. The Gforth manual has more info on the use of these words and what is going on in forth. I would advise reading that and understanding how to pass values and receive values from C functions.
You can also do the following forth code:

```
s" afunction" add-lib          \ again this is the library reference that is needed
c-function seemytest mytest -- n
```

Now to use it you do the same as above. The difference is that the c-library function literal makes a new c file called mycfunction in the above example. This c file then gets compiled and linked to the gforth system running. This means that it is now able to be called without recompiling and linking steps. If you do not use c-library then you get access to the functions in afunction shared library but each time it is called it is linked i believe. Note either way this process described in this document has two parts 1) making the static library 2) using this library in gforth. So once the library is made it can be used by anybody or any time in the future without needing to remake the library. The gforth stuff is only usable for the length of the gforth image availability. Until i figure out how to make a gforth image that can be saved and restarted i think this means that both methods are really about the same.
You can also do inline c function creation as follows:

```
\c int mytest() { return 42 ; }
c-function seemytest mytest -- n
```

This allows you to keep all the c code in files you make to be read into the gforth system but does not create a shared library or make this c code available externally by any means.

Note for using this shared library in another C code.
Simply include the header file for the library that was created for this shared library. In this

example it would be like this: #include “afunction.h” . The example would be as follows:

```
$ sudo nano testingit.c          # the following is the c code you put into testingit.c file
#include “afunction.h”
#include <stdio.h>
int main( int argc, char *argv[])
{
printf(“The answer is %d\n”,mytest());
return 0;
}
```

You can see the call to the function in afunction.h as per its prototype. Now to compile this new c code to execute from command line for example do this:

```
$ gcc testingit.c -lafunction -o final_name_to_excecute       # note the -l in front of the
```

shared library name. Also note the library name is used not the libafunction just the name minus the lib part with no extention.

To run this new c program from command line simply do this:

```
$ ./final_name_to_excecute
```