1. Use Linux utilities to explore the format of a linkable object file.

   (a) Math.o file type is : REL (Relocatable file)
   Math.o magic number is:
   7f 45 4c 46 02 01 01 03 00 00 00 00 00 00 00 00
   Math.o number of section headers: 29

   (b) Section listings with memory map

Section Number	Section Name	MemoryMap
1	.text	.text
2	.rela.text	.text
3	.data	.data
4	.bss	.bss
22	.comment	.text
19	.debug_str	.debug[1]
27	.symtab	.symtab[2]

   1:
   Debug: This section holds information for symbolic debugging. The contents are unspecified. All section names with the prefix .debug are reserved for future use in the ABI.
   2:
   Symtab: This section holds a symbol table. If the file has a loadable segment that includes the symbol table, the section's attributes will include the SHF_ALLOC bit; otherwise, that bit will be off.

   (c) Symbol Table
   The variables that are in the symbol table are aa and bb.
   The functions inside the symbol table are int_mul, int_add, printf, and main.
   The variable cc is not in the symbol table because it is declared inside the function main, and thus will only be used in the main function.
   The two functions int_mul and int_add are in the symbol table because they are functions that the linker will need to find, in order to resolve references.
   This is the same reason that the function main and printf are included in the symbol table.
   If a symbols scope is outside of the file, or references something outside of the file, then it's symbol must be on the symbol table, otherwise the linker will not be able to find it. This is essentially why the symbol cc is not on the symbol table, because its scope is only within main .

2. Explore the format of an executable object file

(a) Simplemath executeable examination

Name	Type	Section
aa	OBJECT	.data
bb	OBJECT	.bss
int_add	FUNC	.text
int_mul	FUNC	.text
main	FUNC	.text

(b) **Differences between *simplemath* and *simplemath.o***
Where program headers are concered, the object file does not have any, and the executeable simplemath file does have 56 bytes of program headers, and a total of 9 program headers.
Both the object file and the executeable file contain section headers, however the executeable contains quite a few more than the object file does. The ones that are more noticable are that the executable file contains a .dynamic, .got, and initalizers for the program and for arrays. The headers are for dynamic linking information and a global offset table and finally code to initalize variables and other things before entering the main function, respectively.
It would seem that the executeable file contains section headers that are meant to set up values and initalize memory before it will start executeing code, whereas the object file contains headers that will only setup a memory offset to where those things will happen.
Symbol tables are also interesting, there is a dynamic symbol table for the executeable file, whereas there isn't one for the object file. The dynamic symbol table in the executeable file seems to link to functions or other things that are called inside the stdio.h header file used. The regular symbol table in the executeable file contains almost twice as many symbol enteries, however they do not start until about halfway into the table. The executeable file contains symbol entires to setup IO, program frames, header files, external functions, and the global offset table. The symbol table for the object file starts about 75% of the way through. It only has entries for things declared specifically inside the object files that it was linked from.

3. Build your own math library for sharing.

(a) The first thing that I did was, in my Makefile for the project, I included flags to compile all of the *.c* files into PIC *.o* files. As shown here:

```
CC= gcc
LD= ld -r
CFLAGS= -std=gnu99 -g -O -c
CSECFL= -fPIC -I -L
CFLAG3= -shared
RM= /bin/rm -f
OBJ= my_add.o my_mul.o math.o

all: math my_add my_mul

#The following will make the .o code for use with the
#simplemath and for the simpleone code, this takes care of
#everything because I use the CSECFL tag as well.
math: math.c
	$(CC) $(CFLAGS) $@.c $(CSECFL)

my_add: my_add.c
	$(CC) $(CFLAGS) $@.c $(CSECFL)

my_mul: my_mul.c
	$(CC) $(CFLAGS) $@.c $(CSECFL)

simplemath: $(OBJ)
	$(CC) $(OBJ) -o $@

simplemath.o: $(OBJ)
	$(LD) $(OBJ) -o $@

clean:
	$(RM) *.o simple* *.t $(SO)
```

After I had the correct PIC files, I was then able to incrimentally link them into a *.so* file. I then had to rename this .so file so that when I linked the next PIC object, it wouldn't overwrite the current shared object file. After doing this three times, I had the *libmymath.so* file that was desiered. I was then able to use the compiler and build the shared object into an executeable file. The command I used to do this was extremely long and redundant:

```
gcc -L./ -Wl,-rpath=\$(PWD) -o simpleone -lmymath -lmath
```

Where the shared object's I made were *libmymath.so* and *libmath.so*. The compiler qould not recognize the library otherwise.
Once I had done this, the file compiled with no errors and I was able to execute it. I did not have to make any changes to the *.c* files themselves. After playing with the compiler a bit more, I found that an easier way to do this, that wouldn't involve so many extrenuous

shared objects being made, is to link all the object files at once into a single shared object. Doing this would include all of the functions need, and then the only extra thing that would have to be done, is export the wokring diretory as the libraries path. *This can be shown here in my Makefile version 2.*

```
CC= gcc
LD= ld -r
CFLAGS= -std=gnu99 -g -O -c
CSECFL= -fPIC -I -L
CFLAG3= -shared
RM= /bin/rm -f
OBJ= my_add.o my_mul.o math.o
SO= libmymath.so

all: math my_add my_mul

#The following will make the .o code for use with the
#simplemath and for the simpleone code, this takes care of
#everything because I use the CSECFL tag as well.
math: math.c
	$(CC) $(CFLAGS) $@.c $(CSECFL)

my_add: my_add.c
	$(CC) $(CFLAGS) $@.c $(CSECFL)

my_mul: my_mul.c
	$(CC) $(CFLAGS) $@.c $(CSECFL)

simplemath: $(OBJ)
	$(CC) $(OBJ) -o $@

simplemath.o: $(OBJ)
	$(LD) $(OBJ) -o $@

#This will build the shared object file directly from all three
#PIC sources at once.
lib: $(OBJ)
	$(CC) $(CFLAG3) $(OBJ) -o $(SO)

#Links the shared object library directly into an executeable.
simpleone: $(OBJ)
	$(CC) -L./ -l:libmymath.so -o $@

clean:
	$(RM) *.o simple* *.t $(SO)
```