# Lab 1

Omar Raza

Due: 6/21/2016

**Problem 1**:

a) The nature of the output generated by the gcc is assembly language that is then used by the computer in an executable file called a.out

b/c) The three steps preformed by the gcc compiler is **first** running the C-preprocessor and implementing all the packets and libraries that the program has contacted, including stdio.h. The **second** step in the gcc performance is to contact the C-compiler that breaks the C code into readable assembly language that can interface with the computer in a very efficient and readable manner. The **last** step is the creation of an executable file known as a.out, that is then used by the computer to actually do all the calculation and generate output, or even accept user input.

d) There are two ways to change the name of a.out. The **first** is to use the command **mv** on bash:

```
mv a.out fileName
```

which basically moves the file to another file with the file name fileName. The **second** way is to actually use the **-o** command that is provided by the compiler to move the executable file to a certain other file that can also be executed.

```
gcc executableFile file.c
```

e) The C reserved keywords can be found at: http://tigcc.ticalc.org/doc/keywords.html

f) When using the **-S** command instead of generating the a.out executable file, it generates another executable file main.s.

g) The **-c** option generates another type of executable file main.o.

h) When trying to execute this is the response the bash gives:

```
bash: ./main.o: Permission denied
```

**Problem 2**:

a) The header file is located in the hard-drive of the device after C and the gcc is installed and called upon by the **#include** function in the C-preprocessor.

b) The part of the gcc that handles the included statements and libraries is the **C-preprocessor**.

c) After using the -E -P options invoked alongside the gcc, it merges the documentation of the included statements but after then, stops and just displays the merged files and merged documentation.

d) In the printf statement: "result of %d - %d is %d" the "result of", "-", and "is" are all just plain text that is outputted as is. Yet the "%d" is specially reserved for integers that is called upon by the printf command, and that goes with anything prepended by a "%" sign.

e) The printf object code is stored in the **libc.so** file that is stored in the storage that is under the stdio.h library that is then accessed by the include command that is then used by the C-preprocessor.

f) After looking at the ldd output you can see that printf along with all other commands in stdio.h are **Statically linked**.

g) Both are used in different scenarios, **Static linking** is used by the programmer when they want to actually attach the library to the compilable script so that they are merged. **Dynamic linking** is used when the programmer doesn't want to link the two files and keep both of them separate while still using the library commands that has been linked, the dynamically linked file will remain separate from the executable file as a DLL file.

**Problem 3**:

a) The role of the **&** in C is very important. The **&** is used by the scanf command to clarify a very integral part about data storage. When calling upon many different variables, there are two different parts of the memory that could be called, the address or the content at a specific junction. if there is no **&** prepended to the variable, it implies that you are calling the address in memory, not the content of the address. Prepending a called variable with the **&** call upon the variable or ASCI characters stored at that specific address, that is why scanf needs to use this. Scanf's purpose is to update content within certain addresses to the addresses newest variable. That is why you cannot have a statement such as:

```
scanf("%d %d",x,y);
```

It would call upon the actual addresses and not actually do anything except for causing confusion. The correct statement would be:

```
scanf("%d %d",&x,&y);
```

which allows from proper allocation of the variables and the variable pointers.

b) The gcc throws an error, which doesn't stop the code from compiling and executing, but it recognizes a potential error and identifies it.

c) a.out returns a segmentation error.

d) The runtime errors were produced when scanf tried to push to a address that wasn't available or existed because we didn't clarify the pointer.

e) ampersands are not used in printf statements because you do not need them to call upon the content, only to place or move content.

**Problem 4**:

a) We don't use ampersands when calling or pushing variables in different methods because we are just accessing and pushing the variables, not actually trying to change them, which we would need to actually clarify using an ampersands.

b) What the code segment:

z = mycalc(x,y);

means basically is that the variable in memory 'z' is equivalent to the result of the calculation applied on the variables 'x', and 'y', mycalc(). If you were not to use this, you could also use pointers:

mycalc(x,y,&z);

This works if there is also a change to the called upon method, after this step is done it is easy to infer why this works. It works because of the beauty of pointers, when using the &z symbol, it basically does exactly what the first line of code did, it takes the computed method result of 'x' and 'y' through the method mycalc() and points them into the content of the address of the variable address 'z'.

c) The gcc did not throw any errors and compiled.

d) None

**Problem 5**

a) After the error in which the

#include "mycalc.c"

statement wasn't included is included, it runs perfectly fine.

b) In v7, the modification is that instead of linking the two files together, you linked

mycalc.c to a header and then linked the header myheader.h to main,c.

c) The difference between both the include statements is that the first pertains to local libraries or another class in the same folder, while the second is calling a global header file or a standard header file.

d) An error is thrown in this case because stdio.h is a global header file that is referenced through ld not through our own coded header.

e) The compilation gets terminated by the C-preprocessor because there is no file or directory in such a name.

**Bonus Problem: Available through the program submission**