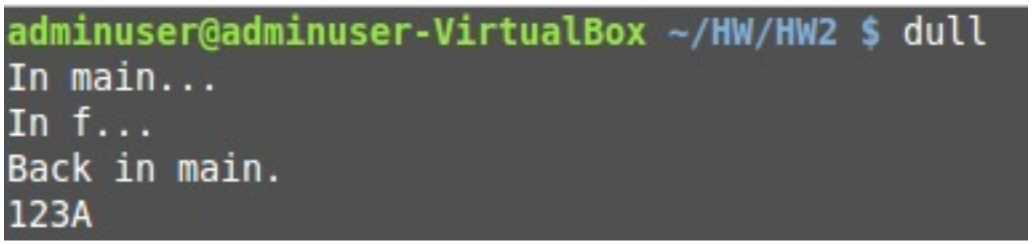
CSCI 2271 Computer Systems

Assignment 2: The Runtime Stack
Due Friday, February 5

1. Download the file *dull.c* from the course website. Its code is:

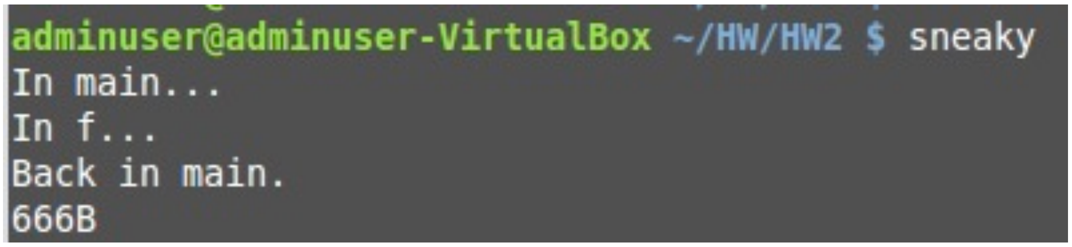
```
#include <stdio.h>
void f();

int main() {
    printf("In main...\n");
    char c = 'A';
    int n = 123;
    f();
    printf("Back in main.\n");
    printf("%d%c\n", n, c);
}

void f() {
    printf("In f...\n");
    //add your code here
}
```

Running this program produces the following output:



```
adminuser@adminuser-VirtualBox ~/HW/HW2 $ dull
In main...
In f...
Back in main.
123A
```

If we examine the code for *dull.c*, it sure looks like "123A" will be printed no matter what function *f* does. Your job is to demonstrate otherwise. Without changing any of the existing statements, and keeping *f* a void function that has no arguments, add code to function *f* so that the program prints "666B" instead. Call this new file *sneaky.c*. Here is an example of my program in action:



```
adminuser@adminuser-VirtualBox ~/HW/HW2 $ sneaky
In main...
In f...
Back in main.
666B
```

Your solution will need to define one or more variables in *f*, and then use pointer arithmetic on them to change the values of the variables in *main*. In order to do this, you need to know the addresses of these variables on the stack, which means that you will need to write code to print out addresses of variables. Consider this task to be detective work. During your detective work you are allowed to write new programs or change *dull.c* arbitrarily. Once you figure out the information you need, you can then write *sneaky.c*.

As part of your *sneaky.c code*, include comments that describe your detective work.

2. Recall that while a program executes, the system uses a special location (called the *base register*) to point to the current stack frame. By convention, stack frames are organized like this:

Local vars	Previous BR	Instruction to return to	Parameters	Return Value Address

Lower Addresses Higher Addresses

The address of each value on the stack can be expressed as an *offset* from the base register. For example, the instruction to return to is located at BR+4, and thus has offset +4.

For alignment purposes, the size of each stack frame must be a multiple of 8. Extra space gets added to its end as needed.

Download the program *HW2test.c* from the course website. Then do the following:

a) For each of the functions *main*, *f*, and *g* of *HW2test.c*,

i. Give the offsets of the local variables and parameters.
ii. Specify if extra space is needed for padding, and how much.
iii. Give the total size of the stack frame for that function.

b) Assume that the bottom of the stack is at location 256, and that the stack grows downward, towards 0. Give a table that describes each value in the stack when the stack is at its largest point – that is, when *main* calls *f* and *f* calls *g* – immediately after *g* executes the statement

```
return result;
```

but before control returns back to *f*.

For each value on the stack, your table should indicate the stack frame it belongs to, its address, size, contents, and purpose. For values you don't know (such as the location of an instruction), you should describe the value (such as "line 2 of the file hw2test.c"). For values that have not been initialized, write "garbage". For example, the first two entries in my solution look like this:

Stack Frame	Address	Size	Contents	Purpose
main	252	4	pointer to somewhere in the OS	pointer to next instruction
main	248	4	0	pointer to previous BR

Note that I am doing arithmetic in base 10, not hexadecimal. You should do the same.

HINT: In my solution, the stack uses 96 bytes at its largest point. That is, the lowest address used by the stack is 160.

When you are finished with both problems, submit your file *sneaky.c* and a file containing your answer to problem 2 to Canvas.