# Lab #09: Program Layout and Command Line

## Getting started

Download lab09 materials from D2L (including this instruction and starter codes)
Enter Mimir IDE.
Change into the cse220 directory.
Create a new directory called lab09.
Change into the new directory.
Uncompress and upload three starter codes to Mimir IDE (do not upload the whole zip file), save them in /home/(your_username)/cse220/lab09/
Implement the programs below in your lab09 directory.

## Program 1: ROT Transformation:

You will write a program called **rot_stdin.c**

The first thing that you should implement is a function that given a string performs the ROT13 transformation on the input and returns the modified string. You will want to prompt the user to input a string in the main function, and output the result to aid in testing. The string will be composed entirely of lowercase letters with a length less than 100.

Sample i/o for this step:

```
Please enter a string: abcdef
The string transformed ROT13: nopqrs
```

Next, you will want to modify your function so that it takes an integer as the shift index, so that it can perform any ROT transformation (0-25) (i.e. instead of just doing char+13, you can do char+1, char+2, …)
Sample i/o for this step:

```
Please enter a string: abcdef
Please enter an integer for rotation amount: 3
The string transformed ROT3: defghi
```

After you finish program 1, your code should be able to run the following two examples:

```
user@mimir: ~/cse220/lab09 > ./program1
Please enter a rotation amount:
13
Please enter a string:
abcdef
Your original string is: abcdef
The string ROT13 transformed is: nopqrs
user@mimir: ~/cse220/lab09 > ./program1
Please enter a rotation amount:
3
Please enter a string:
abcdef
Your original string is: abcdef
The string ROT3 transformed is: defghi
```

## Program 2: Command Line Arguments

We have used command line arguments throughout the semester, most notably in the commands that we use to compile our code. Today, you will write a program that makes use of command line arguments. We take your code of program 1 as an example, you modify the code of program 1 so that it makes use of command line arguments.

The command line arguments can be accessed by having parameters on your main function as so:

```
int main ( int argc, char *argv[] )
```

The integer is the number of arguments, and the argv is an array of character arrays that holds each of the individual arguments as a separate string (remember, strings are character arrays). The first element of the array is the name of the program.

The next step for this assignment is to take in the string and the integer as command line arguments rather than as a user prompted input. Note that the parameters are always passed as character arrays and thus the integer parameter will need to be converted to an integer type. You write a program **rot_command_line.c** based on your code from program 1, and then compile it to rotCipher.

After you finish program 2, **you *no longer* use ./rotCipher to run your code**. Instead, you should be able to use **./rotCipher 3 vanek** to run your code, and your code should output the following results:

```
user@mimir: ~/cse220/lab09 > gcc -o rotCipher rot_command_line.c
user@mimir: ~/cse220/lab09 > ./rotCipher 3 vanek
Your original string is: vanek
Your rotation amount is: 3
The string ROT3 transformed is: ydqhn
```

Note that **./rotCipher 3 vanek** is just an example for which the rotation amount is 3 and the input string is vanek.