**Jake Karasik**
**Nicholas Petriello**

## General

Our program uses the LOLS algorithm implemented in the void LOLS() function in our functions.c file for both the threads and the processes. Our compressR_LOLS() function spawns as many processes as needed to create the request number of parts unless parts is negative or larger than the number of characters in the file. In these cases it will give an error and exit the program. On valid input, it will loop through the correct number of times fork()ing and execvp()ing our compressR_worker_LOLS executable, which calls the LOLS() function from the functions.c file. At the end, it loops through the spawned children's ID's and waits on them. Our compressT_LOLS() works similarly. However, instead of fork() and execvp() it uses threads so it uses pthread_create() to make threads with the thread_worker() function that calls LOLS() and then it uses pthread_join() to wait for the threads to finish.

## Important Things to Note

1. When making compressed files, if the compressed file name already exists, it will be deleted before the new compression.
2. Our program uses additional files including: compress.h, and functions.c so they must be properly linked/used or our makefile can be used.
    - compressT_LOLs
        - gcc -Wall -g -pthread -c functions.c compressT_LOLS.c
    - compressR_LOLS
        - gcc -Wall -g -c functions.c compressR_LOLS.c
        - gcc -Wall -g functions.c compressR_worker_LOLS.c -o compressR_worker_LOLS
    - Your tester with the main function
        - gcc -Wall -pthread <MAIN_FILE_NAME> functions.o compressT_LOLS.o compressR_LOLS.o -o <DESIRED_EXECUTABLE_NAME>

3. Our program ignores non-alphabetic characters when compressing, but it calculates the size of the partitions including the ignored characters which may result in some files being empty.

## Functions

**void compressT_LOLS(char * file, int parts):**
This is the multi-thread version of compression. The main setup is that a main thread creates new threads to do the compression and then joins them at the end.

**void compressR_LOLS(char * file, int parts):**
This is the multi-process version of compression. The main setup is that a parent process spawns off additional worker child processes and reaps them at the end.

**void LOLS(int start, int end, int part_number, char * file):**
LOLS compression algorithm. It compresses a file given a start position, end position, part number and file name. The part_number is the index of the compressed file being made. The algorithm goes through the indicated range and copies to the compressed file only alphabetical characters. All other symbols are ignored. WARNING! If a compressed file of the original file already exists with the same part number, it will be removed.

**int length_of_int(int n):**
Returns number of digits in int 'n'.

**void * thread_worker(file_data * data):**
Worker function called by pthread create. This calls the LOLS() function.