CS 304
Homework 3

How to submit --

Type the following to submit after you finish all the problems:

~lyang11/bin/submit cs304 hw3 chfile.c hw3.txt

1. **File I/O**

More information about file i/o can be found https://www.cs.bu.edu/teaching/c/file-io/intro/

The following is adapted from http://www.codingunit.com/c-tutorial-file-io-using-text-files

Read the following three programs: writefile.c, readfile.c, copyfile.c

Also download the files: input.txt, output.txt, file1.txt
And run the three programs.

```
#include <stdio.h>

int main()
{
   FILE *ptr_file;
   int i;

   ptr_file =fopen("output.txt","w");
   if (!ptr_file)
           return 1;

   i = 1;
   do{
     fprintf(ptr_file, "%d %d\n", i, i*i);
   } while (i++ < 10);


        fclose(ptr_file);
      return 0;
   }
```
----

ptr_file =fopen("output", "w");

The fopen statement opens a file "output.txt" in the write (w) mode. If the file does not exist it will be created. But you must be careful! If the file exists, it will be destroyed and a new file is created instead. The fopen command returns a pointer to the file, which is stored in the variable ptr_file. If the file cannot be opened (for some reason) the variable ptr_file will contain NULL.

if (!ptr_file)

The if statement after fopen, will check if the fopen was successful. If the fopen was not successful, the program will return a one. (Indicating that something has gone wrong).

fprintf(ptr_file, "%d %d\n", i, i*i);

The fprintf statement should look very familiar to you. It can be almost used in the same way as printf. The only new thing is that it uses the file pointer as its first parameter.

fclose(ptr_file);

The fclose statement will close the file. This command must be given, especially when you are writing files. So don't forget it. You have to be careful that you don't type "close" instead of "fclose", because the close function exists. But the close function does not close the files correctly. (If there are a lot of files open but not closed properly, the program will eventually run out of file handles and/or memory space and crash.)

```
----------------------------------------
#include <stdio.h>

int main()
{
   FILE *ptr_file;
   char buf[1000], first_name[20], last_name[20];
   int mid, final;

   ptr_file =fopen("input.txt","r");
   if (!ptr_file)
           return 1;

         while (fgets(buf,1000, ptr_file)!=NULL)
     {
       printf("%s",buf);
      sscanf(buf, "%s %s %d %d",first_name, last_name, &mid, &final);
      printf("Name: %s %s Mid: %d Final: %d\n",first_name, last_name, mid, final);
     }

         fclose(ptr_file);
      return 0;
   }
----------------------------------------
```

A file "input.txt" is opened for reading using the function fopen in the mode read (r).

fgets(buf,1000, ptr_file)

The library function fgets will read each line (with a maximum of 1000 characters per line.) If the end-of-file (EOF) is reached the fgets function will return a NULL value. Each line will be printed on stdout (normally your screen) until the EOF is reached. The file is then closed and the program will end.

sscanf(buf, "%s %s %d %d",first_name, last_name, &mid, &final);

The sscanf statement sis similar to scanf. The only new thing is that it uses a string as its first parameter, so sscanf will read from a string. In this example, it will read two strings and two decimal integer numbers, seperated by whitespaces. In this case, it will skip the whitespaces between two items in the input.

```
#include <stdio.h>

int main()
{
  FILE *ptr1_file, *ptr2_file;
  char buf[1000];
   int mid, final;

   ptr1_file =fopen("file1.txt","r");
   if (!ptr1_file)
          return 1;

   ptr2_file =fopen("file2.txt","w");
   if (!ptr2_file)
          return 1;

    while (fgets(buf,1000, ptr1_file)!=NULL)
        fputs(buf, ptr2_file);

         fclose(ptr1_file);
         fclose(ptr2_file);

      return 0;
   }
```

This example copies the content of file1.txt to file2.txt.

fputs(buf, ptr2_file);

This library function fputs puts a string to a file.

TASK:

You are asked to write a program that takes two file names as the commandline input. The first file has multiple lines each of which starts with an integer. Your job is to increment the integer in the beginning of the line and write the modified lines to the second file. Save your program in chfile.c

For example, if you run your program like the following:

>./chfile f1.txt f2.txt

It will change [f1.txt](f1.txt) and write the modified lines to f2.txt

say our f1.txt looks like the following:

2 good ,scc
3 bad,sc ,
4 hatnklxa m c .
5 make,sa./ as;a
1 sad. ,s ds
3 ted, . solsam,s
8 goodkkklllllllllllllllllllllllllllllllllllllll
109 nakd sksiow xdmd,s ama., a'al;
999 nanaksalm c xcd mcsowdokdq

In the end, f2.txt should look like:

3 good ,scc
4 bad,sc ,
5 hatnklxa m c .
6 make,sa./ as;a
2 sad. ,s ds
4 ted, . solsam,s
9 goodkkklllllllllllllllllllllllllllllllllllllll
110 nakd sksiow xdmd,s ama., a'al;
1000 nanaksalm c xcd mcsowdokdq

2. Assembly Basics (save your answers in hw3.txt)

Assume the following `gdb` output from an X86-64 machine. To understand the data in memory, we look at the first line (i.e., the line starting from address 0x0000000008048a30). The addresses for the 8 bytes are: 0x0000000008048a30, 0x0000000008048a31, 0x0000000008048a32, 0x0000000008048a33, 0x0000000008048a34, 0x0000000008048a35, 0x0000000008048a36, 0x0000000008048a37

```
0x0000000008048a30  0x55 0x89 0xe5 0x53 0x83 0xec 0x04 0x8b
0x0000000008048a38  0x45 0x08 0x8b 0x5d 0x0c 0x83 0xf8 0x01
0x0000000008048a40  0x75 0x0e 0xa1 0x68 0xa8 0x04 0x08 0xa3
0x0000000008048a48  0x84 0xa8 0x04 0x08 0xeb 0x5e 0x89 0xf6
0x0000000008048a50  0x83 0xf8 0x02 0x75 0x3b 0x83 0xec 0x08
0x0000000008048a58  0x68 0x5d 0x99 0x04 0x08 0xff 0x73 0x04
0x0000000008048a60  0xe8 0x73 0xfe 0xff 0xff 0xa3 0x84 0xa8
0x0000000008048a68  0x04 0x08 0x83 0xc4 0x10 0x85 0xc0 0x75
0x0000000008048a70  0x3b 0x83 0xec 0x04 0xff 0x73 0x04 0xff
0x0000000008048a78  0x33 0x68 0xa8 0x96 0x04 0x08 0xe8 0xc5
0x0000000008048a80  0xfd 0xff 0xff 0xc7 0x04 0x24 0x08 0x00
0x0000000008048a88  0x00 0x00 0xe8 0x19 0xfe 0xff 0xff 0x90
0x0000000008048a90  0x83 0xec 0x08 0xff 0x33 0x68 0xc5 0x96
0x0000000008048a98  0x04 0x08 0xe8 0xa9 0xfd 0xff 0xff 0xc7
0x0000000008048aa0  0x04 0x24 0x08 0x00 0x00 0x00 0xe8 0xfd
0x0000000008048aa8  0xfd 0xff 0xff 0x90 0xe8 0x83 0x06 0x00
0x0000000008048ab0  0xab 0xcd 0xef 0x01 0x23 0x45 0x67 0x89
```

Assume the following hexadecimal register contents:

%rax	0000000008048a37
%rbx	0000000006e2d975
%rcx	0000000000000030
%rsi	000000000c8a5b9c

For each of the following instructions, give the hexadecimal contents of the `%rdx` register after the instruction has been executed:

a. `movq %rax,%rdx`
b. `movq 0x08048a6b,%rdx`
c. `movq $0x08048a6b,%rdx`
d. `movq 42(%rax),%rdx`
e. `movq 0x17(%rax,%rcx),%rdx`
f. `leaq (%rax),%rdx`
g. `leaq 0xffffb29c(%rbx,%rsi),%rdx`
h. `leaq (,%rbx,2),%rdx`
i. `leaq (%rax,%rsi,4),%rdx`
j. `leaq 219(%rax,%rcx,8),%rdx`

3. Convert the following C program into assembly code.

```
int test(int x, int y)
{
  int result;
  if (x > y) goto Else;
  result = x+y;
  goto Exit;
Else:
  result = (x-y)*2;
Exit:
  return result;
}
```

Suppose we will use the following registers:
%rdi -> argument x
%rsi -> argument y
%rax -> return value

Save it to hw3.txt