The goal of this program is to simulate the myNode file indexing in C/C++. We will save the contents of a file on the disk using this simulated myNode. (myNode is a simplified version of the Unix inode for file indexing.)

Directions:

Irrespective of any design decisions you make,

1. Your program should take two input files from the command line: input_file, file_access_trace.

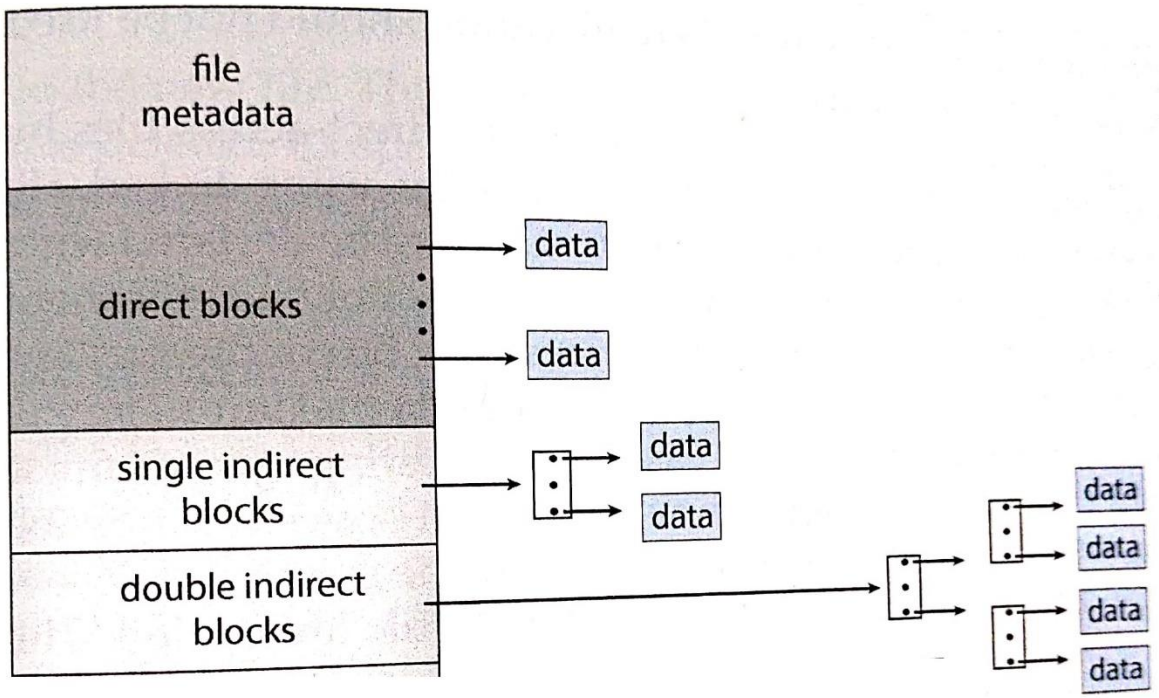
2. Refer to the provided "access_trace1.txt" and "input_file1.txt" files for expected format of the files and initial testing.
3. For final evaluation of the code, **different input files will be used**, which will be in exactly the same format as the provided files but will have different parameters, values. The code should work on different input files.
4. The program will primarily be evaluated for functionality. The greater the number of test cases they pass, the better.
5. The program shouldn't fail to do what it's designed to do.

6. Please test your program thoroughly before submitting order.

7. **Your program will be evaluated for the stated functionality.** Please ensure they compile and run as expected. Programs should not fail to compile or run.
8. Test and debug program. Before submitting make sure to create test cases and test your program.

Assume that an operating system is using the following **myNode indexing scheme** for accessing file blocks on the disk**:** In this program assume that the myNode has14 pointers of the index block. The first 12 pointers point to direct blocks, i.e., they contain addresses of blocks that contain data of the file. The last two pointers point to the **indirect blocks**. The 13th pointer

points to a **single indirect block**, which is an index block containing not data but the addresses of blocks that do contain data. The 14th pointer points to a **double indirect block**, which contains the address of a block that contains the addresses of blocks that contain pointers to the actual data blocks. The following picture illustrates myNode structure.



The myNode

Simplifying assumption: For the requirements of the program we will simulate each block in the input file as a separate file on our disk. The block pointers, in that case, will be the file names storing the corresponding block data.

Your program should take two input files from the command line: input_file, file_access_trace. "Input_file" will contain information about the file that needs to be stored in the simulated myNode. This file has two columns: "block number," and "data in block". As the names imply, "block number" indicates the number of this block in the original file to be stored (**block numbering starts from 0**). "Data in block" contains the data stored in the corresponding block number. A sample input_file file "input_file1.txt" is included. Use this file for initial testing of the program "File_access_trace" is a sequence of block requests from the input_file. It has requests for reading or writing data into the file. E.g. entries may look like:

R, 100 //read block number 100 from the file

W, 21, Hello //Write/overwrite the contents of block number 21 with the string "Hello"

Let us consider the contents of test_file.txt for example:

#block number, data in block

0, hi

1, hello
2, test
3, msg

A read request to block 2 should return the string “test,” similarly a write request “W, 2, test123” should replace the contents of block 2 with “test123.” A subsequent call “R, 2” should now return “test123.”

A sample file_access_trace (“access_trace1.txt”) accompanies this order. Use this file along with “input_file1.txt” for initial testing. (Create more files for further testing.) For final evaluation, we will use a different set of test files.

In this instructions for this program we will simulate each block in the file as a separate file on our disk, e.g., a new file in a subdirectory inside the current working directory. For example, you can create a subdirectory called “input_file1_dir” to store all the “blocks” corresponding to the “input_file1.txt.” Let us again consider the contents of test_file.txt (mentioned above). This file has four blocks. The myNode of this file will contain: file name, i.e., “test_file.txt,” and four pointer values. In our simulation these pointers are the names of the files which store these blocks. For instance, if we store block 0 (containing “hi”) in “zero.txt,” block 1 (containing “hello”) in “one.txt,” block 2 (containing “test”) in “two.txt,” block 3 (containing “msg”) in “three.txt,” then the contents of myNode would be: “test_file.txt,” “zero.txt,” “one.txt,” “two.txt,” “three.txt.” You should save the myNode in a special file named “super_block.txt” in the same directory as other files. You can choose any format for saving the myNode contents. For simplicity, in the program we will assume that, in case of indirect block access, each block can store 100 file pointers (or names in this case). Therefore, the 13th pointer value in our myNode will point to a file which can contain, at max, 100 file block pointers (remember in our case these pointers are the file names). Similarly, the 14th pointer value will point to a file which contains (at max) 100 file names, each of these files in turn will contain (at max) 100 file names each of which will contain actual data blocks.

While serving a read or write request for the file your program should print all the intermediate files it reads. For example, on servicing the “R, 12” request, it should print something like: “Accessed pointer 12 of myNode; next, read the 0th entry of the file ‘level1_indirection.txt’ .” (Here we are assuming the counting starts from 0). Similarly, when servicing double indirection blocks, you should print all the intermediate files read and the positions/pointers that were accessed.

The order will be reviewed for the following requirements:

1. Correct functioning of Read operations:
   A. Read operation in direct blocks.
   B. Read operation in **single indirect block.**
   C. Read operation in **double indirect block.**
2. Correct functioning of Write operations:
   A. Write operation in direct blocks.
   B. Write operation in **single indirect block.**
   C. Write operation in **double indirect block.**

Please note that part of the correctness of your Read and Write operations will be determined by the intermediate print messages, as mentioned above. So please make sure these messages are accurate and readable.

3. If the "input_file" specifies a file greater than the maximum file size, your program should throw an error message: "input file greater than max supported file size!"

4. If the trace file tries to access an invalid block number, your program should throw an error message: "Invalid block number!"

Answer the following in your readme file
that must be created with the program:
5. What is the maximum number of blocks (in a file) that the above myNode can support? Why? And show your work.

6. What naming convention do you use for naming the files containing the input file blocks? (You should explain how do you name the files pointed by the direct block pointers in the myNode? Similarly, how do you name the files pointed by the **single indirect block** and **double indirect block**?) A sample file naming convention is outlined in the "Sample expected output" section below. (Feel free to use it if you like.)

7. Prepare a "README.txt" file for your order The file should contain the following:

a) Instructions for compiling and executing your program(s). Include an example command line.

b) Answers to questions 5 and 6.

8. You should also comment your code well. The best way to go about it is to write comments while coding.

What should you submit?

Upload your order as a zip file. The zip file should contain:
1. All your code files and any other files that might be needed for executing your code.
2. README.txt

Sample expected output

The sample output below for the program is assuming the following file naming convention for saving the file blocks: Indices in the myNode (inside the "super_block.txt" file) will be called zero.txt, one.txt, ... , thirteen.txt. (Note that the file names are numbered from zero to thirteen for the fourteen pointers in myNode.) From there, levels of indirection are denoted with an underscore, followed by an index number. For example, the indirect block pointed to by the myNode ptr at index twelve will be called twelve_0.txt. Block number 112, the first

data block from Node Thirteen (which uses level-2 allocation) is called thirteen_0_0.txt.

Assuming you write your program in file named myNode.cpp, the following commands can be used for compiling and running your program on on a Linux VDI (virtual desktop infrastructure):

```
g++ -std=c++11 myNode.cpp -o myNode.out
./myNode.out input_file1.txt access_trace1.txt
```

Running the program using the input_file1.txt and access_trace1.txt files supplied with the assignment, would produce the following sample output.

```
Read request for block number 0.
Accessed pointer 0 of myNode: zero.txt
Reading from zero.txt.
Contents:
   hi_0

Read request for block number 12.
Accessed pointer 12 of myNode: twelve.txt
Accessed pointer 0 of twelve.txt: twelve_0.txt
Reading from twelve_0.txt.
Contents:
   hi_12

Read request for block number 100.
Accessed pointer 12 of myNode: twelve.txt
```

Accessed pointer 88 of twelve.txt: twelve_88.txt
Reading from twelve_88.txt.
Contents:
  hi_100

Write request for block number 100.
Accessed pointer 12 of myNode: twelve.txt
Accessed pointer 88 of twelve.txt: twelve_88.txt

Read request for block number 100.
Accessed pointer 12 of myNode: twelve.txt
Accessed pointer 88 of twelve.txt: twelve_88.txt
Reading from twelve_88.txt.
Contents:
   Hello_100

Write request for block number 12.
Accessed pointer 12 of myNode: twelve.txt
Accessed pointer 0 of twelve.txt: twelve_0.txt

Read request for block number 12.
Accessed pointer 12 of myNode: twelve.txt
Accessed pointer 0 of twelve.txt: twelve_0.txt
Reading from twelve_0.txt.
Contents:
   Hello_12

Read request for block number 1000.
Accessed pointer 13 of myNode: thirteen.txt
Invalid block number!

Read request for block number 212.
Accessed pointer 13 of myNode: thirteen.txt
Accessed pointer 1 of thirteen.txt: thirteen_1.txt.
Accessed pointer 0 of thirteen_1.txt: : thirteen_1_0.txt.
Reading from thirteen_1_0.txt.
Contents:
  hi_212

NOTE: The picture below would help you visualize how the data blocks are organized, save and the text file naming convention used in the output above.