Design:
Since there are a lot of lines of code this time. Detailed code will be put in changes.pdf

//This number below corresponds to the requirement number in PA4
2)
The directoryWalker is a modified version of ls, which prevents it from printing parent directory and when it reaches a directory, it recursively prints content under that directory. Furthermore, it adds all discovered inodes to an array (which is useful for comparison later).

$directoryWalker
recursively print out each file name and inum starting from the root directory.
$directoryWalker dir1
recursively print out each file name and inum under dir1 starting from dir1 itself.

3)
We created a function inodeWalk() in fs.c, which will loop through the disk inodes and print out all the allocated inums, and add those inums to an array (for the comparison later). And we made a system call iwalk() to call this function. Protoypes of the functions with manual of system call are below:

```
int   inodeWalk(short* iarray);

int   iwalk(short* iarray);
loop through disk inodes, print out allocated ones, and mark the corresponding
element in iarray as 1. return 0 if successful.
```

$inodeTBWalker
loop through the inode table and print out all allocated inums. This user program doesn't take argument.

4)
To compare, simply make two arrays, use directoryWalker to modify one and inodeTBWalker to modify the other. Then compare two arrays and print the result.

$compare
print out the difference of two walkers or "Two walkers find the same inodes" if they are the same. It doesn't take argument.

5)
To erase the information of a directory inode, we created a function dErase() in fs.c, which takes a path of a directory, finds the inode, and checks if the inode is a directory and makes sure it's not root directory (because that will make it not recoverable). Then it'll remove every information under that inode including the data block pointers. Then we added a system call dirErase() to call this function. Protoypes of the functions with manual of system call are below:

```
int   dErase(char* path);

int   dirErase(char* path);
erase the information of the directory at path but keep the inode. return 0 if
successful.
```

For the user program, it's name erase, which takes the argument of a path and will call dirErase().

$erase dir1

damage dir1.

6)
recoverDir will recover all the damaged directories and the files under them.
It first runs compare with a modified version of directoryWalker which will check if a directory is damaged by checking the size of it (damaged directory will have size 0). And all the inodes that inodeTBwalker can find but directoryWalker cannot find must be the files under the damaged directory. Then we created a system call recDir() to find the corresponding directory inode and its parent inode, then call recoverDir(), which we created in fs.c to link all the contents back to the directory inode.
Protoypes of the functions with manual of system call are below:

```
void  recoverDir(struct inode* dp,struct inode* ip,int* inum,int num_inum);

int recDir(char* path,int* inum,int num_inum);
recover the directory inode under path, link “.” and “..” back to the directory,
and link all inodes from inum[0] to inum[num_inum-1] under this directory. Since
the content of the directory is erased, names of files under the directory will be
lost. Recovered files will be given new names.
```

The proof that file names cannot be restored is here:
http://teaching.idallen.com/cst8207/18w/notes/450_file_system.html#damaged-directories-create-orphans

Finally, the user program will recursively run compare until every directory is recovered.
$recoverDir
recover damaged directories and orphaned files

Answer to the questions of this requirement:
1. Recovery cannot be done just by simply Walkers, but by adding functions to the Walkers, recovery can be achieved.
2. There are no limit to the number of damaged directories (as long as root directory is not damaged, because otherwise we can’t even run a user program.) However, if only one directory is damaged, we can restore the file system to the exact original structure. For example, dir1 is damaged and file1 and file2 are orphaned, we can recover dir1 and put file1 and file2 under dir1. If more than one directories are damaged, we can restore the files but we can’t know if the files are the same structure as original. For example, dir1 and dir2 are both damaged, and they are both under root directory. Now file1 and file2 are orphaned, then the original structure could be dir1/file1, dir1/file2 or dir1/file1, dir2/file2 or dir1/file2, dir2/file2 or dir2/file1, dir2/file2 and we don’t know which case it is. Therefore, the recovered structure might not be the original structure but the files will all be safe.

7)(Bonus)
We designed a program that can damage the type of an inode, change a directory to file or change a file to directory. And we can recover it.
To damage an inode,int dtype(char*); we made a system call dtype(), the manual is below:

```
int dtype(char* path);
find the inode of path and change the type from T_DIR to T_FILE or from T_FILE to
T_DIR. Return 0 on successful.
```

$damagetype file1
damage the type of file1.

To recover damaged type, we designed a function recoverType() in fs.c, which will iterate through all the disk inodes. For each allocated inode, it'll check if inode→type is consistent with its content. The method we use is to assume all the files are directory, then the first directory entry must have name "." and the inum must be its own inum. If they are not, it's not a directory. Then if inode→type is inconsistent with the content, we change the type. Then we made a system call recType() to call this function. The prototypes and the manual of the system call is below:

```
int recoverType(void);
```

```
int recType(void);
loop through all the allocated inodes and change the file types to the correct
ones if they are incorrect. Return 0 on successful recovery.
```

The user program that calls recType() is recoverType.
$recoverType
iterate through allocated inodes on disk and recover the types of all the inodes. Print "recover finished" on success; "recover failed" on failure.