Computation I 5EIA1
C Exam: File System
21 January 2020 18:00-21:00/21:30

Important

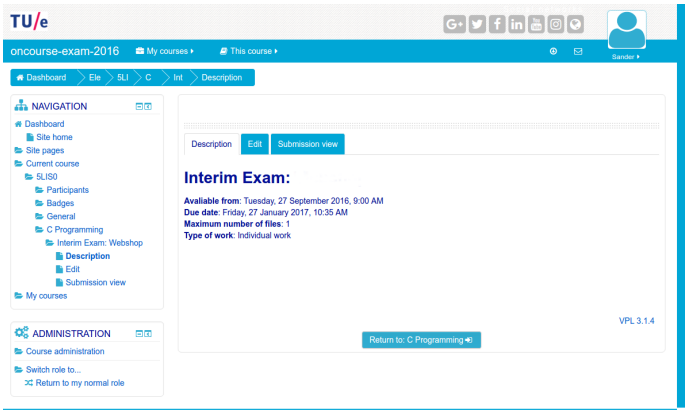
- Your grade is based on the number of cases passed. So try to complete the cases one by one. Note that there may be additional test cases that are only run after the exam.
- You can press "evaluate" as often as you like during the exam to evaluate your solution. We advise you to do this regularly during the exam.
- You must follow the instructions of the exam. For example, you may not use an array if a linked-list implementation is asked for.
- You are only allowed to use the Kernighan & Ritchie paper book in the Dutch or English language as a reference during the exam. No other electronic or printed material are allowed.
- The grade is based on your **last submission**. So make sure you submit a working version that completes as many cases as possible. It's also wise not to make last-minute changes.
- Your program must work for all inputs, not just the test cases. For example, when a `#define MAX 10` must be defined as part of the exam, then your program should work for all values of `MAX`. We will change the test cases after the exam (but the number of test cases per task, i.e., the grade will not change).

In this exam you will develop a simple file system. The user will be able add files and directories, print their contents (also recursively), move files and directories and sort them alphabetically.

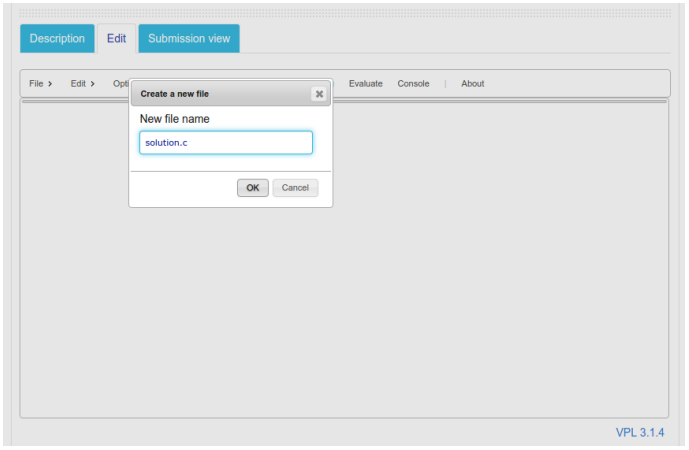
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	#cases	cases per task	% per task	Cumulative %
no function	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	20	2	10%	10%
addInode			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	18	2	10%	20%
printDirectory					1	1	1				1	1	1	1							7	1	5%	25%
changeDirectory						1	1	1	1	1		1				1					7	3	15%	40%
printFullPath									1								1	1			3	1	5%	45%
printDirectoryRecursive										1									1	1	3	1	5%	50%
removeInode											1	1									2	2	10%	60%
sortDirectory													1	1							2	2	10%	70%
findInodeAnywhere															1	1					2	2	10%	80%
changeDirectoryAbsolute																	1	1	1	1	4	2	10%	90%
move																			1	1	2	2	10%	100%

Figure 1: Test cases and points per task. Various functions are independent of each other. It may be good to complete these functions in the order you find easiest first.

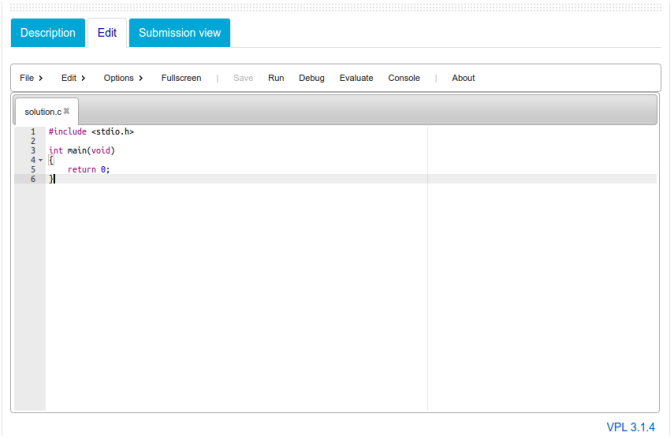
Open the correct C programming exam on exam.oncourse.tue.nl. You will see a screen similar to:



Select the "Edit" tab and the following screen will appear:



Provide a useful name for your C file (e.g., solution.c) and press "Ok".
You will then see the following screen:



You can write your own C program in the text editor that is now shown in your browser. Once you press "Save" you can "Run" and "Evaluate" your program. Using the "Run" command you will see a terminal where you can provide input to your program. You can use this to debug your program. Using the "Evaluate" command all test cases are evaluated and at the end the results are displayed (grade and errors if present).

Task 1. In this exam you will develop a simple file system. The user will be able add files and directories, print their contents (also recursively), move files and directories and sort them alphabetically. Write a C program that asks the user to select the command that needs to be performed. The commands that need to be supported are listed in the following table:

command	operation
q	quit the program
a	add inode (directory or file) to working directory
p	print content of working directory
c	change working directory
f	print full path of working directory
P	list content of working directory and all of its sub directories
r	remove inode from working directory
s	sort current working directory alphabetically
F	find inode anywhere (find directory or file in current directory and its sub-directories)
C	change working directory absolute path
m	move inode from source to destination

In this task you only need to implement the quit command. In later tasks you will implement the remainder. If an invalid character is given then print the error message `Unknown command 'Z'` (with Z replaced by the unknown command):

```
Command? X
Unknown command 'X'
Command? q
Bye!
```

Your program must produce the *exact* output, including all spaces, capitalisation, quotes, etc.

Your program should now pass test case 1 and 2.

Task 2. A file system is used to organize a collection of files on a physical disk. A central element in Unix-style file system is the so called inode (index node). The inode is a data structure that describes a file-system object such as a file or a directory. Each inode stores the name of the file or directory, the parent (directory) it belongs to. In case the inode is a directory itself, it will also contain the list of children (files or directories) it contains.

Your file system will keep track of all files and directories in a tree of `struct inodess`. Each `struct inode` contains a name, pointer to a parent, and a fixed-size array of pointers to children.

Define a `struct inode` structure that contains the following fields:

- A `char *name` field to store the name of the file/directory.
- A `struct inode *parent` field that points to the directory containing this inode.
- An array of `MAX_CHILDREN` pointers to `struct inodes` called `children`. Define `MAX_CHILDREN` to be 4 using a `#define`.

Define a function "`struct inode *addInode(struct inode *fs, char *name)` that inserts a new inode with name `name` as a child of the inode `fs`, and returns a pointer to the newly created inode. The newly created inode should be inserted in the first position with value NULL in `fs->children`. All elements of the `children` array of the newly created inode should be initialised to `NULL`.

Define `struct inode *fc = addInode(NULL, "/");` in your `main` function. This function call creates the top-level directory ("/") that contains all inodes (files and directories) in the file system. The variable `fc` will be used in the program to as a pointer to the working directory. I.e., its value will change when the user requests a change in directory (to be implemented in a later task).

Add the 'a' command to the main function to call the `addInode` function after asking for the filename (use the `"%s"` format string). If the user supplied filename ends with a slash '/' then the inode is assumed to be a directory. Otherwise it is assumed to be a regular file. If a file or directory with the supplied name already exists in the current directory then an error message as shown below should be printed. New inodes are inserted at the first NULL position in the array.

```
Command? a
Give filename: hello.txt
Command? a
Give filename: hello.txt
File hello.txt already exists in /
Command? q
Bye!
```

In case the directory is full (i.e., the directory already contains `MAX_CHILDREN` children), then the following error message should be printed.

```
Command? a
Give filename: 1.txt
Command? a
Give filename: 2.txt
Command? a
Give filename: 3.txt
Command? a
Give filename: 4.txt
Command? a
Give filename: 5.txt
Directory / is full.
Command? q
Bye!
```

Your program should now pass test cases 1 to 4.

Task 3. Next, let's print the content of the current directory with the 'p' command in a depth-first fashion (i.e. print the content of the subdirectories as soon as possible). Add the function "`void printDirectory(struct inode *fs)`" for this. Before every file inside the directory you should print "File: " and before every directory you should print "Directory: ".

```
Command? a
Give filename: tmp/
Command? a
Give filename: hello.txt
Command? p
Directory: tmp/
File: hello.txt
Command? q
Bye!
```

Your program should now pass test cases 1 to 5.

Task 4. The 'c' command changes the current working directory. Implement the "`void changeDirectory(struct inode **fs, char *dir)`" function. The variable `fs` contains a pointer to the inode of the current working directory. Its value is updated within this function call. The sub directory that should be entered is specified with the variable `dir`. The variable `dir` may also have a value ".." which implies that the current working directory should be changed to the parent. Of course, when the current working directory is the root node "/" then ".." should have no effect. Example output is:

```
Command? a
Give filename: tmp/
Command? c
Change path to: tmp/
Command? a
Give filename: hello.txt
Command? p
File: hello.txt
Command? c
Change path to: ..
Command? p
Directory: tmp/
Command? c
Change path to: ..
Command? p
Directory: tmp/
Command? q
Bye!
```

When the user specified sub directory does not exist in the current working directory, the following error should be printed.

```
Command? c
Change path to: tmp/
Directory does not exist!
Command? q
Bye!
```

Your program should now pass test cases 1 to 8.

Task 5. Extend the program with the 'f' command which prints the full path from the root to the working directory. Implement the function "`void printFullPath(struct inode *fs)`" for this.

```
Command? a
Give filename: tmp/
Command? c
Change path to: tmp/
Command? a
Give filename: tmp/
Command? c
Change path to: tmp/
Command? f
Full path: /tmp/tmp/
Command? q
Bye!
```

Your program should now pass test cases 1 to 9.

Task 6. In a previous task you have implemented the `printDirectory` function which prints the content of the current working directory. In this function you are going to create an alternative print function that prints not only the content of the current working directory, but recursively also the content of all its sub directories. Implement the function "`void printDirectoryRecursive(struct inode *fs)`" for this purpose. The function should be called with the command 'P'.

```
Command? a
Give filename: hello.txt
Command? a
Give filename: tmp/
Command? c
Change path to: tmp/
Command? a
Give filename: world.txt
Command? a
Give filename: var/
Command? c
Change path to: var/
Command? a
Give filename: boo.txt
Command? c
Change path to: ..
Command? P
File: world.txt
Directory: var/
File: boo.txt
Command? c
Change path to: ..
Command? P
File: hello.txt
Directory: tmp/
File: world.txt
Directory: var/
File: boo.txt
Command? q
Bye!
```

Your program should now pass test cases 1 to 10.

Task 7. Now we will implement the command 'r' with the function "`void removeInode(struct inode *fc, char *fn)`". In this task the function removes an inode with name `fn` from the current working directory `fc`. The remaining inodes in the directory are left unchanged in the `children` array. If the inode is a sub directory, then it is only removed if the sub directory is empty otherwise an error message should be printed. An error message should also be printed when the user specified file is not present in the directory.

```
Command? a
Give filename: hello.txt
Command? a
Give filename: tmp/
Command? c
Change path to: tmp/
Command? a
Give filename: foo.txt
Command? c
Change path to: ..
Command? P
File: hello.txt
Directory: tmp/
File: foo.txt
Command? r
Give filename: tmp/
Cannot delete non-empty directory.
Command? r
Give filename: hello.txt
Command? P
Directory: tmp/
File: foo.txt
Command? r
Give filename: hello.txt
File or directory hello.txt does not exist in /
Command? q
Bye!
```

Your program should now pass test cases 1 to 12.

Task 8. The function "`void sortDirectory(struct inode *fs)`" sorts the children of the current working directory `fs` in alphabetical order. This function is executed using the 's' command.

```
Command? a
Give filename: tmp/
Command? a
Give filename: bla.txt
Command? p
Directory: tmp/
File: bla.txt
Command? s
Command? p
File: bla.txt
Directory: tmp/
Command? q
Bye!
```

Your program should now pass test cases 1 to 14.

Task 9. The 'F' command that uses the "`void findInodeAnywhere(struct inode *fs, char *name)`" function displays all the locations (full path) within the current working directory `fs` where an inode with the supplied `name` is found. No output is given when the file is not found.

```
Command? a
Give filename: hello.txt
Command? a
Give filename: tmp/
Command? c
Change path to: tmp/
Command? a
Give filename: hello.txt
Command? a
Give filename: foo.txt
Command? a
Give filename: var/
Command? c
Change path to: var/
Command? a
Give filename: hello.txt
Command? c
Change path to: ..
Command? F
Give filename: hello.txt
/tmp/hello.txt
/tmp/var/hello.txt
Command? c
Change path to: ..
Command? F
Give filename: hello.txt
/hello.txt
/tmp/hello.txt
/tmp/var/hello.txt
Command? F
Give filename: bla.txt
Command? q
Bye!
```

Your program should now pass test cases 1 to 16.

Task 10. The function `changeDirectory` allows the user to make a relative change in the current working directory by going to its parent or into one of its sub directories. In this task you will implement the function "`struct inode *changeDirectoryAbsolute(struct inode *fs, char *dir)`" that allows the user to change immediately to a new absolute position in the file system. The function gets as arguments the current working directory `fs` and a string `dir` that holds the concatenated names of all directories from the root directory to which the current working directory should be changed. The function returns this new working directory (if it exists) or otherwise an error message is printed and a NULL pointer is returned. Note that in the latter case the current working directory in the `main` function should not be updated.

```
Command? a
Give filename: tmp/
Command? c
Change path to: tmp/
Command? a
Give filename: foo/
Command? c
Change path to: foo/
Command? f
Full path: /tmp/foo/
Command? C
Change path to: /
Command? f
Full path: /
Command? C
Change path to: /tmp/foo/
Command? f
Full path: /tmp/foo/
Command? C
Change path to: /tmp
Directory does not exist.
Command? f
Full path: /tmp/foo/
Command? q
Bye!
```

Your program should now pass test cases 1 to 18.

Task 11. The final task is to move a directory inode with all of its children to a new location in the file system. This move command 'm' is implemented with the function "`void move(struct inode *fSrc, struct inode *fDst)`". The function takes as arguments the `fSrc` inode that needs to be made a child of the `fDst` inode. Remember that when moving the `fSrc` inode it should be removed from the children of its original parent. There are also two error conditions that need to be handled correctly (see output below).

```
Command? a
Give filename: tmp/
Command? a
Give filename: foo/
Command? c
Change path to: tmp/
Command? a
Give filename: var/
Command? c
Change path to: var/
Command? a
Give filename: hello.txt
Command? C
Change path to: /
Command? P
Directory: tmp/
Directory: var/
File: hello.txt
Directory: foo/
Command? m
Source path: /tmp/var/
Destination path: /foo/
Command? P
Directory: tmp/
Directory: foo/
Directory: var/
File: hello.txt
Command? q
Bye!
```

Your program should now pass test cases 1 to 20.

Submission: Your *last* submission will be automatically graded.