Computation I 5EIA1
C Exam Part 2: Unix file utilities (v1.3, October 22, 2020)
27 October 15:40-17:20

In this exam you will develop several Unix file command-line utilities.

Important

- In this exam a predefined function is available for task 1: `predefined_read_contents`. Therefore, if you get stuck on that task or want to skip it then you can use the predefined function instead of your own function. In that case you will not get the points for the test cases of task 1.
- Oncourse will show all the test cases passed, including those using the predefined functions. Points for using predefined functions will be deducted after the exam.
- To use the predefined functions you need to include `#include "predefined.h"` in your program. This is done automatically when you create a new .c file.

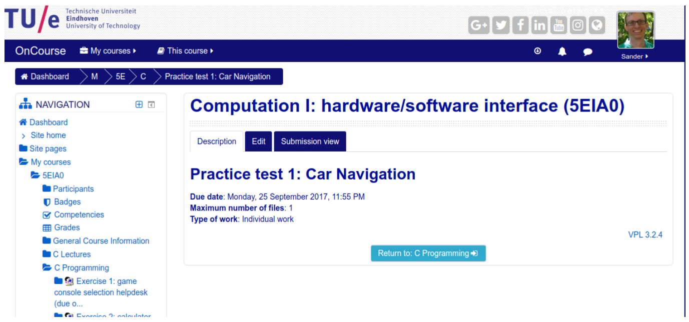
function	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	% per fn	cumulative %
quit	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	5%	5%
read file		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	20%	25%
cat						1	1														10%	35%
rev (reverse lines)								1	1												10%	45%
wc (word count)										1	1	1									15%	60%
palindrome													1	1							10%	70%
brackets															1	1	1				15%	85%
tac (reverse file)																		1	1	1	15%	100%
																					100%	

Figure 1: Test cases and points per task. You can use predefined functions to not implement a function yourself but you will not get the points for that function. All test cases are listed at the end of this document.

Important

- Your grade is based on the number of cases passed. Try to complete the cases one by one.
- 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 materials offered electronically as part of the exam (cheat sheet and K&R book) during the exam. No other electronic or printed material are allowed.
- The grade is based on your **last submission**. 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).

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



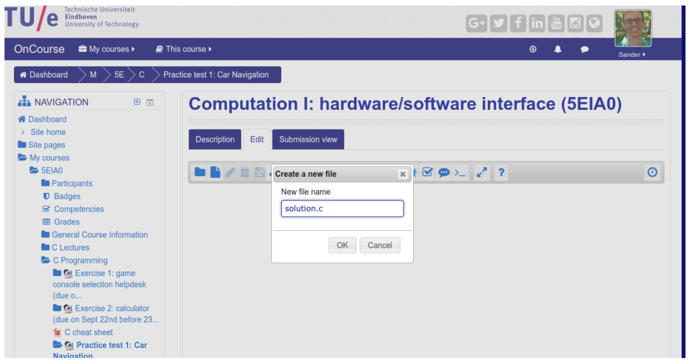
Computation I: hardware/software interface (5EIA0)

Practice test 1: Car Navigation

Due date: Monday, 25 September 2017, 11:55 PM
Maximum number of files: 1
Type of work: Individual work

VPL 3.2.4

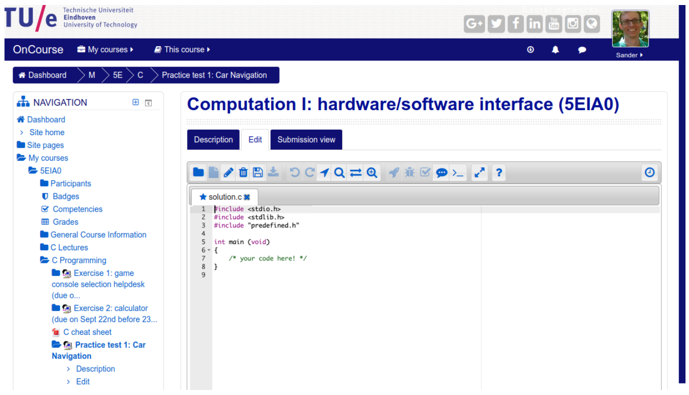
Select the "Edit" tab and the following screen will appear:



Computation I: hardware/software interface (5EIA0)

New file name: solution.c

Provide a useful name for your C file (e.g., solution.c) and press "Ok". **The file name must end in .c and cannot contain any spaces.** You will then see the following screen:



Computation I: hardware/software interface (5EIA0)

```
#include <stdio.h>
#include <stdlib.h>
#include "predefined.h"

int main (void)
{
    /* your code here! */
}

```

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. 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 program
f	read file into memory
c	display all lines of a file (cat)
r	display each line of a file in reverse (rev)
w	display the number of lines, words, and characters of a file (wc)
p	check if a string is a palindrome (all characters)
b	check if the brackets are matched in a file
t	display all lines of a file in reverse order (tac)

In this task you only need to implement the quit command. In later tasks you will implement the remainder.
Hint: make a loop in which you ask for a single character command, like this:

```
char cmd;
do {
  printf ("Command? ");
  scanf(" %c",&cmd); // notice the space before the %
  ...
} while (cmd != 'q');
```

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.

Task 2. Declare an integer array `contents` with 10000 elements inside your main function. Initialise the first element with `EOF`, which is defined in the `stdio.h` library header file. Implement the 'f' command to ask for a file name and then read the contents of the file in the `contents` array. Write and use the function `void read_contents(char *filename, int contents[])` to do this. Close the file after its contents have been read.

Give the error message `Cannot open file NAME.` with NAME replaced by the name of the file in case the file cannot be opened.
The end of the file contents is indicated by the `EOF` value in the array, i.e. the `EOF` value is the first integer in the contents array after the contents of the file. (Similar to how you would terminate a string with the `'\0'` character.) For example, reading an empty file results in `EOF` as the first element of the array. See the next page for some hints.

```
Command? f
File name? file-doesnotexist
Cannot open file file-doesnotexist.
Command? f
File name? file0
Command? q
Bye!
```

Your program should now pass test cases 1 to 5.

Hint: You can skip task 2 by using a predefined function
`void predefined_read_contents(char *filename, int contents[])`.
To do this you need to add `#include "predefined.h"` at the top of your program. You will not get the points for the test cases of task 2 if you use the predefined function. Note that Oncourse will show that you pass the tests, but the points will be deducted after the exam.

All following tasks are independent, as shown in the scoring overview on page 1. This means that you can make them in any order, or skip any tasks if you wish.

Hint: If when you run your program you receive an error message similar to:

```
/tmp/ccJmRCOb.o: In function ‘predefined_read_contents’:
  exam.c:(.text+0x0): multiple definition of ‘predefined_read_contents’
```

then you have named your own function `predefined_read_contents` instead of `read_contents`. When you write your own function then it should not contain `predefined_` in the name.

Hint: The following files are used in the automated tests, and you can also use them when running and debugging your program: file-empty, file0, file1, file2, file3, file4.

All automated test cases are also included at the end of this document.

file0:

```
one two
three four
```

file1:

```
First line.
This is a file with some text.
And another line.

Madam, I’m Adam
step on no pets
  meetsysteem
parterretrap

()()()
(()(()][[)()))
((
Some funky characters !#$%^&*().
))
>---->
->-->-
-->>--

Trailing spaces.
    Leading spaces.

Stop!
```

file2:

```
(one ((()two()()()()()((((()))))))
three) (four))
oddity
```

file3:

```
word-word word
(a bb ccc!
```

file4:

```
(one (two)
)) three (four)
```

Task 3. Implement the 'c' command (cat) to print out the contents of the file that was read into the contents array. Write a function `void cat(int contents[])` to do this.

The cat, rev, wc, brackets, and tac commands do not modify the `contents` array, and they can therefore be run multiple times after reading a file. This is illustrated below.

```
Command? f
File name? file0
Command? c
one two
three four
Command? c
one two
three four
Command? q
Bye!
```

Your program should now pass test cases 1 to 7.

Task 4. Implement the 'r' command (rev) to print out each line of the file that was read into the contents array in reverse. (The lines are printed out in the right order, but the characters on each line are printed in reverse order.)

```
File name? file0
Command? c
one two
three four
Command? r
owt eno
ruof eerht
Command? q
Bye!
```

Your program should now pass test cases 1 to 9.

Task 5. Implement the 'w' command (wc) to print out the number of lines, words, and ASCII characters in the file that was read into the contents array, as shown below. A line is a sequence of zero or more characters followed by a newline `'\n'` character. A word is defined as a sequence of one or more letters. Note that white space is included in the character count. This is why file0 has 19 characters: the 17 visible characters, plus two newlines. (All test files finish with a newline character immediately before the `EOF`. `EOF` is not a character or a newline.)

```
Command? f
File name? file-empty
Command? w
0 lines, 0 words, 0 characters.
Command? f
File name? file0
Command? c
one two
three four
Command? w
2 lines, 4 words, 19 characters.
Command? q
Bye!
```

Your program should now pass test cases 1 to 12.

Task 6. Implement the 'p' command (palindrome) to check if a string is a palindrome. A palindrome is string that is the same when read forwards and backwards. An example is "parterretrap". Strings of length one are palindromes, e.g. the string "x" is a palindrome. Read the string using the "%s" format string (i.e. leading spaces are skipped as shown below in the second string).

```
Command? p
String? parterretrap
Is a palindrome.
Command? p
String?     x
Is a palindrome.
Command? p
String? FooBar
Is not a palindrome.
Command? p
String? <--<
Is a palindrome.
Command? q
Bye!
```

Your program should now pass test cases 1 to 14.

Task 7. Implement the 'b' command (brackets) that checks if every left round bracket `'('` is matched with a right round bracket `')'` in the file that was read into the contents array. All other characters must be ignored. A file containing no round brackets is matched. The function prints one of the following:

```
Brackets are matched at end of file.
Brackets are not matched at end of file.
Brackets not matched at line X character Y.
```

Start counting at line 0, character 0. The message "line X character Y" indicates the first unmatched bracket. The character count is reset at the start of every line (e.g. line 1, character 0). Nested brackets must be taken into account correctly, as illustrated below.

```
()        Brackets are matched at end of file.
(())      Brackets are matched at end of file.
()()      Brackets are matched at end of file.
(         Brackets are not matched at end of file.
)         Brackets not matched at line 0 character 0.
())(()    Brackets not matched at line 0 character 2.
```

```
Command? f
File name? file-empty
Command? b
Brackets are matched at end of file.
Command? f
File name? file0
Command? c
one two
three four
Command? b
Brackets are matched at end of file.
Command? f
File name? file2
Command? c
(one ((()two()()()()()((((()))))))
three) (four))
oddity
Command? b
Brackets are matched at end of file.
Command? f
File name? file3
Command? c
word-word word
(a bb ccc!
Command? b
Brackets are not matched at end of file.
Command? f
File name? file4
Command? c
(one (two)
)) three (four)
Command? b
Brackets not matched at line 1 character 1.
Command? q
Bye!
```

Your program should now pass test cases 1 to 17.

Task 8. Implement the 't' command (tac) to print out the lines of the file that was read into the contents array in reverse order.

```
Command? f
File name? file0
Command? c
one two
three four
Command? t
three four
one two
Command? f
File name? file2
Command? c
(one ((()two()()()()((((()))))))
three) (four))
oddity
Command? t
oddity
three) (four))
(one ((()two()()()()((((()))))))
Command? q
Bye!
```

Your program should now pass test cases 1 to 20.

Submission: Your *last* submission will be graded. Note that points will be deducted after the exam for using predefined functions.

Input / output test cases

Long lines have been wrapped at 70 characters for legibility. When your program output is compared to the expected output lines will not be wrapped.

Case 01

Input:

```
X
q
```

Output:

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

Case 02

Input:

```
f
file0
q
```

Output:

```
Command? File name? Command? Bye!
```

Case 03

Input:

```
f
file-doesnotexist
q
```

Output:

```
Command? File name? Cannot open file file-doesnotexist.
Command? Bye!
```

Case 04

Input:

```
f
file-empty
q
```

Output:

```
Command? File name? Command? Bye!
```

Case 05

Input:

```
f
file-empty
f
file0
f
file1
f
file2
f
file3
f
file4
q
```

Output:

```
Command? File name? Command? File name? Command? File name? Command?
File name? Command? File name? Command? File name? Command? Bye!
```

Case 06

Input:

```
f
file4
c
c
f
file3
c
q
```

Output:

```
Command? File name? Command? (one (two)
)) three (four)
Command? (one (two)
)) three (four)
Command? File name? Command? word-word word
(a bb ccc!
Command? Bye!
```

Case 07

Input:

```
c
f
file-empty
c
c
f
file0
c
q
```

Output:

```
Command? Command? File name? Command? Command? Command? File name?
Command? one two
three four
Command? Bye!
```

Case 08

Input:

```
f
file0
r
f
file-empty
r
q
```

Output:

```
Command? File name? Command? owt eno
ruof eerht
Command? File name? Command? Command? Bye!
```

Case 09

Input:

```
r
f
file1
r
f
file3
r
q
```

Output:

```
Command? Command? File name? Command? .enil tsriF
.txet emos htiw elif a si sihT
.enil rehtona dnA

madA m'I ,madaM
step on no pets
  meetsysteem
parterretrap

)()()(
)))()[[]](()((
((
.)(*&^%$#! sretcarahc yknuf emoS
))
>---->
->-->-
-->>--

     .secaps gniliarT
.secaps gnidaeL

!potS
Command? File name? Command? drow drow-drow
!ccc bb a(
Command? Bye!
```

Case 10

Input:

```
f
file0
w
q
```

Output:

```
Command? File name? Command? 2 lines, 4 words, 19 characters.
Command? Bye!
```

Case 11

Input:

```
f
file-empty
w
f
file0
w
f
file1
w
f
file2
w
f
file3
w
f
file4
w
q
```

Output:

```
Command? File name? Command? 0 lines, 0 words, 0 characters.
Command? File name? Command? 2 lines, 4 words, 19 characters.
Command? File name? Command? 22 lines, 30 words, 256 characters.
Command? File name? Command? 3 lines, 5 words, 56 characters.
Command? File name? Command? 2 lines, 6 words, 26 characters.
Command? File name? Command? 2 lines, 4 words, 27 characters.
Command? Bye!
```

Case 12

Input:

```
w
f
file-doesnotexist
w
q
```

Output:

```
Command? 0 lines, 0 words, 0 characters.
Command? File name? Cannot open file file-doesnotexist.
Command? 0 lines, 0 words, 0 characters.
Command? Bye!
```

Case 13

Input:

```
p
parterretrap
p
    x
p
FooBar
p
<--<
q
```

Output:

```
Command? String? Is a palindrome.
Command? String? Is a palindrome.
Command? String? Is not a palindrome.
Command? String? Is a palindrome.
Command? Bye!
```

Case 14

Input:

```
p
x
p
xx
p
xxx
p
hello
p
parterretrap
p
MadamImadaM
p
>---->
p
<--->
q
```

Output:

```
Command? String? Is a palindrome.
Command? String? Is a palindrome.
Command? String? Is a palindrome.
Command? String? Is not a palindrome.
Command? String? Is a palindrome.
Command? String? Is a palindrome.
Command? String? Is a palindrome.
Command? String? Is not a palindrome.
Command? Bye!
```

Case 15

Input:

```
f
file-empty
b
f
file0
b
q
```

Output:

```
Command? File name? Command? Brackets are matched at end of file.
Command? File name? Command? Brackets are matched at end of file.
Command? Bye!
```

Case 16

Input:

```
f
file2
b
f
file3
b
q
```

Output:

```
Command? File name? Command? Brackets are matched at end of file.
Command? File name? Command? Brackets are not matched at end of file.
Command? Bye!
```

Case 17

Input:

```
f
file4
b
f
file1
b
q
```

Output:

```
Command? File name? Command? Brackets not matched at line 1 character
1.
Command? File name? Command? Brackets are matched at end of file.
Command? Bye!
```

Case 18

Input:

```
f
file0
t
q
```

Output:

```
Command? File name? Command? three four
one two
Command? Bye!
```

Case 19

Input:

```
f
file2
t
q
```

Output:

```
Command? File name? Command? oddity
three) (four))
(one (((()two()()()()()((((())))))
Command? Bye!
```

Case 20

Input:

```
t
f
file-empty
t
q
```

Output:

```
Command? Command? File name? Command? Command? Bye!
```