Computation I 5EIA1
C Exam: Search Engine (v1.3, October 28, 2019)
29 October 13:30-16:30/17:00

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).

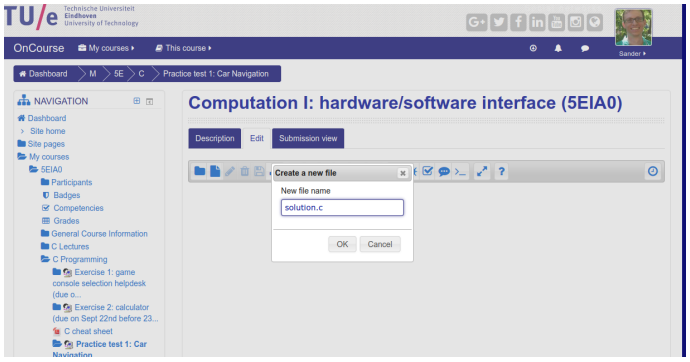
In this exam you will develop a competitor to Google! The user will be able to create web pages, link them, find which pages refer to each other, and search in them.

function	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	# cases	cases per fn	% per fn	cumulative %
quit	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	20	1	5%	5%
add page		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	19	2	10%	15%
print web				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	17	3	15%	30%
find page							1		1	1	1	1	1	1	1	1	1				10	1	5%	35%
reset visited								1							1	1	1				4	1	5%	40%
add link									1	1			1	1	1	1	1		1	1	9	2	10%	50%
remove page (not links to page)											1	1	1	1							4	2	10%	60%
remove page (and links to page)													1	1							2	2	10%	70%
reachable															1	1	1				3	3	15%	85%
search page																		1	1		2	2	10%	95%
search web																				1	1	1	5%	100%

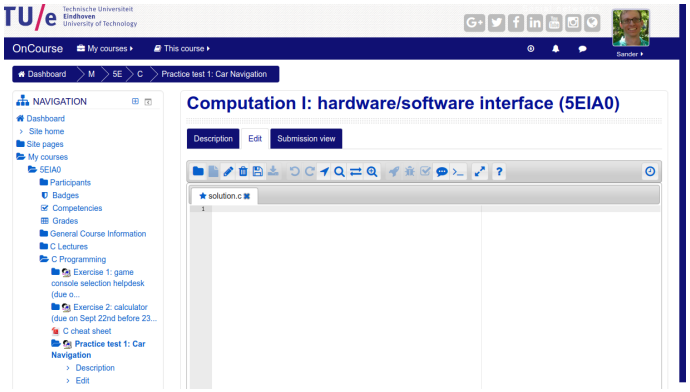
Figure 1: Test cases and points per task. The *remove page* (orange), *reachable* (green) and *search page* (blue) functions can be made independently of one another. It may be good to do the one you find easiest first.

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 competitor to the Google! The user will be able to create web pages, link them, find which pages refer to each other, and search in them. 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
p	add page
w	print all pages on the web
f	find page
v	reset the visited flag on all pages
l	add link
P	remove page
r	find all reachable pages
s	search for a word in a page
S	search for a word in all pages on the web

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 `Invalid command 'Z'` (with Z replaced by the unknown command):

```
Command? X
Invalid 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. The World Wide Web is a collection of pages each with a unique Uniform Resource Locator (URL). Your search engine will keep track of all pages in a linked list of `struct page`s that is ordered alphabetically by URL. Each `struct page` contains a fixed-size array of pointers to pages that it refers to.
Define a `struct page` structure that contains the following fields:

- A `char *url` field to store the URL of the page.
- A `int visited` field that will be used to keep track of whether we visited the page already.
- An array of `MAXLINKS` pointers to `struct page`s called `links`. Define `MAXLINKS` to be 10 using a `#define`.
- A `next` field that points to a `struct page`.

Define `struct page *web = NULL;` in your `main` function.
Define a function "`struct page *add_page (struct page *web, char *url)` that inserts a new page with URL `url` at the correct position in the ordered linked list, and returns the updated list. The `visited` field should be initialised to `-1`, and all elements of the `links` array should be initialised to `NULL`.
Add the 'p' command to the main function to call the `add_page` function after asking for the page URL (use the `"%s"` format string). If the URL already exists on the web then give an error message as shown below.

```
Command? p
Page URL? virtualmachine.compsoc.eu
Command? p
Page URL? compsoc.eu
Command? p
Page URL? compsoc.eu
URL "compsoc.eu" is already on the web
Command? q
Bye!
```

Your program should now pass test cases 1 to 3.

Task 3. Next, let's print out all pages on the web with the 'w' command. Add the function "`void print_web (struct page *web)`" for this.

```
Command? w
Command? p
Page URL? whattheflux.com
Command? w
{URL="whattheflux.com",visited=-1} ->
Command? p
Page URL? atlas.mountains.org
Command? w
{URL="atlas.mountains.org",visited=-1} ->
{URL="whattheflux.com",visited=-1} ->
Command? p
Page URL? zelastone.de
Command? w
{URL="atlas.mountains.org",visited=-1} ->
{URL="whattheflux.com",visited=-1} ->
{URL="zelastone.de",visited=-1} ->
Command? q
Bye!
```

Note there's a space after the `->`. The URLs that the page refers should be printed after the `->` like this:

```
Command? w
{URL="atlas.mountains.org",visited=-1} ->
{URL="whattheflux.com",visited=-1} -> "zelastone.de"
{URL="zelastone.de",visited=-1} -> "whattheflux.com" "atlas.mountains.org"
Command?
```

However, we've not added any links yet. You can add the code to print links now or later, in Task 6.

Your program should now pass test cases 1 to 6.

Task 4. The 'f' command checks if there is a page on the web with a given URL. Implement the "`struct page *find_page (struct page *web, char *url)`" function that returns a pointer to the page or `NULL` otherwise. Example output is:

```
Command? w
Command? f
Page URL? emptyweb.com
URL "emptyweb.com" is not on the web
Command? p
Page URL? myfirstpage.woordpruts.com
Command? w
{URL="myfirstpage.woordpruts.com",visited=-1} ->
Command? f
Page URL? myfirstpage.woordpruts.com
URL "myfirstpage.woordpruts.com" is on the web
Command? q
Bye!
```

Your program should now pass test cases 1 to 7.

Task 5. The `visited` field in each page will be used in a later task to keep track of whether we visited the page already. For now, implement the 'v' command that sets the `visited` field of all pages to zero. Define and use the function "`void reset_visited (struct page *web)`" for this.

```
Command? w
Command? v
Command? w
Command? p
Page URL? one.no
Command? w
{URL="one.no",visited=-1} ->
Command? v
Command? w
{URL="one.no",visited=0} ->
Command? p
Page URL? two.tw
Command? p
Page URL? three.tv
Command? w
{URL="one.no",visited=0} ->
{URL="three.tv",visited=-1} ->
{URL="two.tw",visited=-1} ->
Command? v
Command? w
{URL="one.no",visited=0} ->
{URL="three.tv",visited=0} ->
{URL="two.tw",visited=0} ->
Command? q
Bye!
```

Your program should now pass test cases 1 to 8.

Task 6. The web wouldn't be a web without links. In this task implement the 'l' (links) command with the "`void add_link (struct page *web, char *source_url, char *dest_url)`" function. Ask for two URLs (`source` and `destination`) and then add the link from `source` to `destination`. This means you have to add a pointer to the `destination` page in the `links` field of the `source` page. Use the first `NULL` position in the array for the new link.
If either source or destination URL does not exist, if they are the same, or if the link already exists then give an error message (as shown below). When trying to insert a new page when all `MAXLINKS` links are used the error message "`Maximum number of links reached`" should be printed and the link should not be inserted.

Hint: The `find_page` function that you've written above is useful here.

See output on next page.

Your program should now pass test cases 1 to 10.

```
Command? p
Page URL? tracker.com
Command? p
Page URL? goggle.gr
Command? p
Page URL? fizzbook.fi
Command? p
Page URL? twotter.bu
Command? w
{URL="fizzbook.fi",visited=-1} ->
{URL="goggle.gr",visited=-1} ->
{URL="tracker.com",visited=-1} ->
{URL="twotter.bu",visited=-1} ->
Command? l
Source & destination URL? twotter.bu tracker.com
Command? w
{URL="fizzbook.fi",visited=-1} ->
{URL="goggle.gr",visited=-1} ->
{URL="tracker.com",visited=-1} ->
{URL="twotter.bu",visited=-1} -> "tracker.com"
Command? l
Source & destination URL? twotter.bu tracker.com
"tracker.com" is already a destination for "twotter.bu"
Command? l
Source & destination URL? twotter.bu fizzbook.fi
Command? w
{URL="fizzbook.fi",visited=-1} ->
{URL="goggle.gr",visited=-1} ->
{URL="tracker.com",visited=-1} ->
{URL="twotter.bu",visited=-1} -> "tracker.com" "fizzbook.fi"
Command? l
Source & destination URL? goggle.gr twotter.bu
Command? w
{URL="fizzbook.fi",visited=-1} ->
{URL="goggle.gr",visited=-1} -> "twotter.bu"
{URL="tracker.com",visited=-1} ->
{URL="twotter.bu",visited=-1} -> "tracker.com" "fizzbook.fi"
Command? l
Source & destination URL? twotter.bu twotter.bu
Source and destination URL cannot be the same
Command? l
Source & destination URL? no-page twotter.bu
Source URL "no-page" is not on the web
Command? l
Source & destination URL? twotter.bu no-page
Destination URL "no-page" is not on the web
Command? l
Source & destination URL? not1 not2
Source URL "not1" is not on the web
Command? q
Bye!
```

Task 7. Now we will implement the command 'P' with the function "`void remove_page (struct page **web, char *url)`". In this task the function only has to remove pages that have no links to them. (We complete the function in the next task.) Remove the page from the linked list and free the space used for `url` and the `struct page`. Give an error message if the URL does not exist.

```
Command? P
Page URL? no.page.pl
URL "no.page.pl" is not on the web
Command? p
Page URL? oncourse.tue.nl
Command? p
Page URL? exam.oncourse.tue.nl
Command? p
Page URL? canvas.tue.nl
Command? w
{URL="canvas.tue.nl",visited=-1} ->
{URL="exam.oncourse.tue.nl",visited=-1} ->
{URL="oncourse.tue.nl",visited=-1} ->
Command? P
Page URL? exam.oncourse.tue.nl
Command? w
{URL="canvas.tue.nl",visited=-1} ->
{URL="oncourse.tue.nl",visited=-1} ->
Command? P
Page URL? oncourse.tue.nl
Command? w
{URL="canvas.tue.nl",visited=-1} ->
Command? q
Bye!
```

Your program should now pass test cases 1 to 12.

Task 8. Extend the `remove_page` function such that it also correctly removes pages that have links to them from other pages (i.e. other pages refer to them in their `links` array). Set entries in the `links` array that refer to the page to be removed to `NULL`.

Hint: Note that `NULL` entries and non-`NULL` (pointing to other pages) may be arbitrarily interleaved in the `links` array. Recall that when inserting a new page it should use the first `NULL` entry. This is illustrated below.

```
Command? w
{URL="affligem.be",visited=-1} -> "heineken.nl" "duvel.be"
{URL="bochkarev.ru",visited=-1} ->
{URL="duvel.be",visited=-1} -> "affligem.be"
{URL="heineken.nl",visited=-1} ->
{URL="moretti.it",visited=-1} -> "heineken.nl"
Command? P
Page URL? heineken.nl
Command? w
{URL="affligem.be",visited=-1} -> "duvel.be"
{URL="bochkarev.ru",visited=-1} ->
{URL="duvel.be",visited=-1} -> "affligem.be"
{URL="moretti.it",visited=-1} ->
Command? l
Source & destination URL? affligem.be bochkarev.ru
Command? w
{URL="affligem.be",visited=-1} -> "bochkarev.ru" "duvel.be"
{URL="bochkarev.ru",visited=-1} ->
{URL="duvel.be",visited=-1} -> "affligem.be"
{URL="moretti.it",visited=-1} ->
Command? l
Source & destination URL? affligem.be moretti.it
Command? w
{URL="affligem.be",visited=-1} -> "bochkarev.ru" "duvel.be" "moretti.it"
{URL="bochkarev.ru",visited=-1} ->
{URL="duvel.be",visited=-1} -> "affligem.be"
{URL="moretti.it",visited=-1} ->
Command? q
Bye!
```

Your program should now pass test cases 1 to 14. Notice that the cases 11-14 for this task are independent from cases 15-17 (reachable) and case 18-20 (search).

Task 9. The 'r' command that uses the "`void reachable (struct page *page)`" function displays all the pages that are reachable from page. The pages reachable from page A are page A itself, all pages in its `links` list, and all pages reachable from those. Implement a recursive function that displays all pages reachable from page in a depth-first order. Given an error message (shown below) if the start page does not exist.

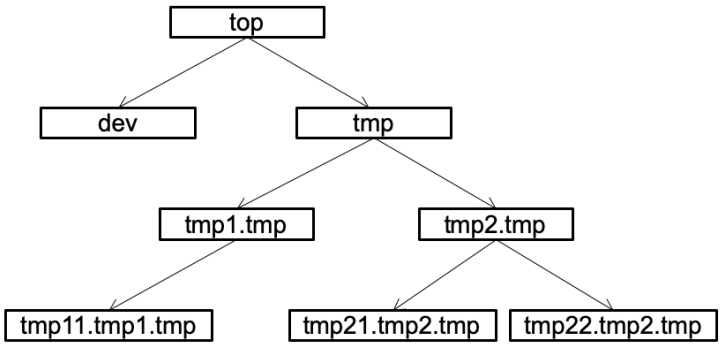


Figure 2: The web in the example output below (also shown by the first 'w' command).

```
Command? r
Page URL? hello.org
No such page "hello.org"
Command? ... insert all pages and links ...
Command? w
{URL="dev",visited=-1} ->
{URL="tmp",visited=-1} -> "tmp1.tmp" "tmp2.tmp"
{URL="tmp1.tmp",visited=-1} -> "tmp11.tmp1.tmp"
{URL="tmp11.tmp1.tmp",visited=-1} ->
{URL="tmp2.tmp",visited=-1} -> "tmp21.tmp2.tmp" "tmp22.tmp2.tmp"
{URL="tmp21.tmp2.tmp",visited=-1} ->
{URL="tmp22.tmp2.tmp",visited=-1} ->
{URL="top",visited=-1} -> "dev" "tmp"
Command? r
Page URL? tmp11.tmp1.tmp
tmp11.tmp1.tmp
Command? r
Page URL? tmp1.tmp
tmp1.tmp
tmp11.tmp1.tmp
Command? r
Page URL? top
top
dev
tmp
tmp1.tmp
tmp11.tmp1.tmp
tmp2.tmp
tmp21.tmp2.tmp
tmp22.tmp2.tmp
Command? q
Bye!
```

Hint: If the starting page exists, then you will need to first reset and then use the `visited` field.

Your program should now pass test cases 1 to 17. Notice that the cases 15-17 for this task are independent from cases 11-14 (remove page) and case 18-20 (search).

Task 10. We will now automate searching the entire web for a key word. Since the web is too large, the directory in which your program runs contains copies of only the following web pages:

- `alcohol.co.uk`

```
stones
xxxx
guinness stout lager
two-packets-of-crisps
glenfiddich
```

- `beers.be`

```
leffe
chimay chouffe
stella
```

- `drink4all.com`

```
stones guinness stout lager leffe
glenfiddich
bordeaux champagne
```

- `water.nl`

```
bar-le-duc
```

- `wines.fr`

```
chateau-neuf-du-pape
bordeaux
beaujolais
champagne burgundy shiraz zinfadel
condrieu
```

In your program you can open these files for reading using the filename `"beers.be"`, etc.
First implement the 's' command with the "`int search_page (struct page *page, char *keyword)`" function. The function opens the file with name equal to the URL (i.e. `page->url`, e.g. `"beers.be"`) in the directory in which the program runs. It then compares the `keyword` with all words in the file. The function returns 1 if the key word was found in the file and 0 otherwise. An error message is given if the file cannot be opened. This is an example output:

```
Command? w
Command? s
Search word in page? leffe beers.be
No page "beers.be"
Command? p
Page URL? beers.be
Command? w
{URL="beers.be",visited=-1} ->
Command? s
Search word in page? heineken beers.be
Page "beers.be" does not contain the word "heineken"
Command? s
Search word in page? leffe beers.be
Page "beers.be" contains the word "leffe"
Command? s
Search word in page? leffe no-beers.be
No page "no-beers.be"
Command? q
Bye!
```

Hint: As illustrated by the first commands a page must be added before you can search it.

Hint: Recall that successive words can be read from the standard input with `scanf("%s",&s);` automatically skipping spaces and newlines. Use something similar in your function.

Your program should now pass test cases 1 to 19. Notice that the cases 18-19 for this task are independent from cases 11-14 (remove page) and 15-17 (reachable).

Task 11. The final task is to search all pages on the web (command 'S') with the function "`void search_web (struct page *web, char *keyword)`".
In the example output `stella` is only in `beers.be`. `leffe` is in both `drink4all.com` and `beers.be`.

```
Command? p
Page URL? alcohol.co.uk
Command? p
Page URL? beers.be
Command? p
Page URL? drink4all.com
Command? w
{URL="alcohol.co.uk",visited=-1} ->
{URL="beers.be",visited=-1} ->
{URL="drink4all.com",visited=-1} ->
Command? S
Search web for word? stella
beers.be
Command? S
Search web for word? leffe
beers.be
drink4all.com
Command? S
Search web for word? guinness
alcohol.co.uk
drink4all.com
Command? S
Search web for word? grolsch
Command? q
Bye!
```

Your program should now pass test cases 1 to 20. Notice that the case 20 for this task is independent from cases 11-14 (remove page) and 15-17 (reachable).

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