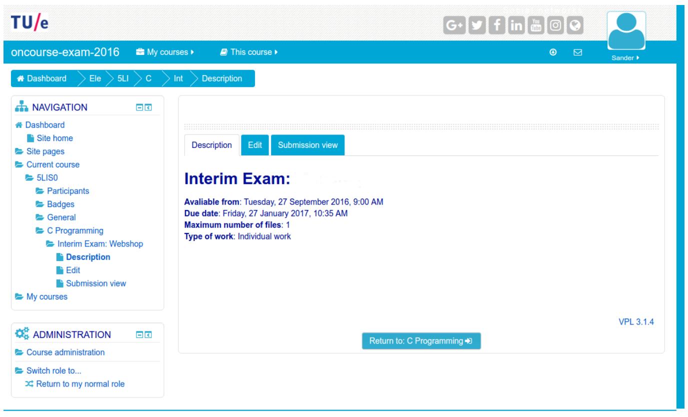
Computation I 5EIA0
Practice Exam: Dictionary
23 September 18:40–20:40

In this exercise you will develop a dictionary program. The user will be able to add and remove words from the dictionary. In addition, the user can manually swap words in the dictionary or automatically sort all words in alphabetical order.

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 use 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.
- Your program must work for all inputs, not just the test cases. We will change the test cases after the exam (but the number of test cases per task, i.e., the grade will not change).

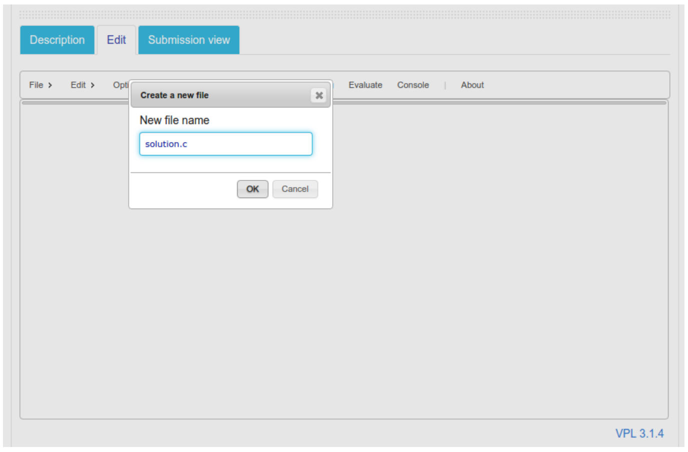
Task 1. Open the correct C programming exam on exam.oncourse.tue.nl. You will see a screen similar to:



Interim Exam:

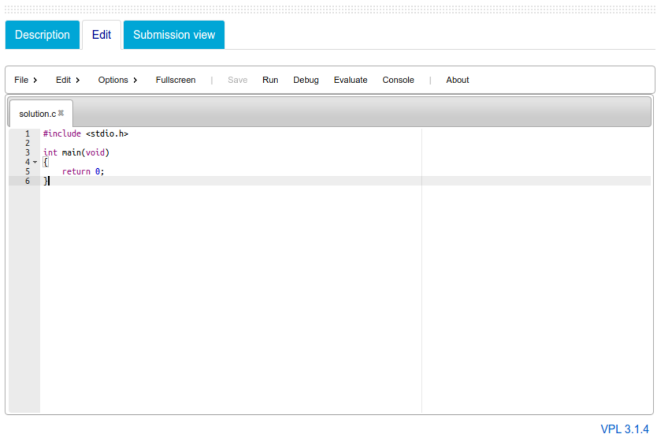
Avaliable from: Tuesday, 27 September 2016, 9:00 AM
Due date: Friday, 27 January 2017, 10:35 AM
Maximum number of files: 1
Type of work: Individual work

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



New file name: solution.c

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



```
#include <stdio.h>

int main(void)
{
    return 0;
}
```

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 2. 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
a	add word
r	remove word
s	swap words
p	print dictionary
o	sort dictionary
q	quit program

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 [arspoq]? ");
  scanf(" %c",&cmd); // notice the space before the %
  ...
} while (cmd != 'q');
```

Print the error message shown below if an invalid command is given:

```
Command [arspoq]? A
Invalid command 'A'
Command [arspoq]? q
Bye!
```

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

Task 3. Inside the `main` function declare a two-dimensional `dict` with room for 10 strings of 30 characters each using the following `#define`:

```
#define MAXWORDS 10
#define MAXLENGTH 30
```

Initialize all elements with the empty string (hint: you can use `= {};` to do this).
Write a function "`int numberOfWordsInDict(char dict[MAXWORDS][MAXLENGTH])`" that counts the number of non-empty strings stored in the array. When you call this function with the empty array that you created in the previous task, the function should obviously return the value 0. In your `main` function you should call the function like this: `i = numberOfWordsInDict(dict);` where `dict` is a variable declared in your main program.

Task 4. Write a function "`void addWord(char dict[MAXWORDS][MAXLENGTH], char word[])`" that adds the string `word` to the dictionary stored in the array `dict`. The function should add the word in the first empty position of the array. Hence, the first word that is added to the dictionary should be added at index 0 of the array `dict`. The next word needs to be added at index 1 of the array, etc. When the user tries to add more words to the dictionary than its maximal size, the function should not add the word to the dictionary. Instead it should print the *exact* error message stating "`Dictionary is already full! The word could not be added.`".

Hint: You can use the function `numberOfWordsInDict` to determine the index at which the word must be added in the array `dict`.

Hint: Use `#include <string.h>` to be able to use the string-manipulation functions. In particular, you can use the `strcpy(dict[i],word)` function to copy the string `word` into the array `dict[i]`. See Appendix B3 of the K&R book.

Task 5. Write a function “`void printDict(char dict[MAXWORDS][MAXLENGTH])`” that prints all strings stored in the array `dict`. When the dictionary contains no words, the function should print a message that the dictionary is empty.

When the dictionary contains the words “Hello”, “World”, and “Apple”, you should see the following output:

```
Dictionary:
- Hello
- World
- Apple
```

When the dictionary is empty, you should see the following output:

```
Dictionary:
The dictionary is empty.
```

Task 6. Your dictionary program already supports multiple operations (i.e., add words and print the dictionary). The user should be able to select the operation he/she wants to perform and subsequently the program should perform this operation. Adapt your main function such that the user can select the desired operation. For now, your program should support the following operations: quit, print, add.

The output of your program should now look as follows:

```
Command [arspoq]? p
Dictionary:
The dictionary is empty.
Command [arspoq]? a
Add a word: Apple
Command [arspoq]? p
Dictionary:
- Apple
Command [arspoq]? q
Bye!
```

Task 7. Write a function "`void removeWord(char dict[MAXWORDS][MAXLENGTH], char word[])`" that removes the first occurrence of the word `word` from the dictionary contained in the array `dict`. The function should ensure that words (i.e., strings) contained in the dictionary are placed at the first elements of the array. In other words, when an empty string is found at element `i` of the array, all following elements of the array should also be empty (i.e. contain the string `""`). Ensuring that this property holds will make it easier to count the number of elements in the array and to sort these elements.

As the program output below illustrates, when you remove a word you should move the last word in the dictionary to the position that the removed word occupied. For example, see how when removing "Union" the last word "Orange" takes its place.
Use the command 'r' for the operation 'remove a word'. The output of your program should then look as follows:

```
Command [arspoq]? a
Add a word: Apple
Command [arspoq]? a
Add a word: Union
Command [arspoq]? a
Add a word: Apple
Command [arspoq]? a
Add a word: Orange
Command [arspoq]? p
Dictionary:
- Apple
- Union
- Apple
- Orange
Command [arspoq]? r
Remove a word: Union
Command [arspoq]? p
Dictionary:
- Apple
- Orange
- Apple
Command [arspoq]? a
Add a word: Banana
Command [arspoq]? p
Dictionary:
- Apple
- Orange
- Apple
- Banana
Command [arspoq]? r
Remove a word: Apple
Command [arspoq]? p
Dictionary:
- Banana
- Orange
- Apple
Command [arspoq]? q
Bye!
```

Task 8. Write a function "`void swapWords(char dict[MAXWORDS][MAXLENGTH], char word1[], char word2[])`" that swap the words `word1` and `word2` inside the dictionary contained in the array `dict`. When one or both words are missing in the dictionary, the function should print an error message.

Use the command 's' for the operation 'swap two words'. The output of your program should then look as follows:

```
Command [arspoq]? a
Add a word: Apple
Command [arspoq]? a
Add a word: Union
Command [arspoq]? a
Add a word: Apple
Command [arspoq]? a
Add a word: Orange
Command [arspoq]? p
Dictionary:
- Apple
- Union
- Apple
- Orange
Command [arspoq]? s
Swap two words:
Enter first word: Union
Enter second word: Orange
Command [arspoq]? p
Dictionary:
- Apple
- Orange
- Apple
- Union
Command [arspoq]? s
Enter first word: Union
Enter second word: Banana
Cannot swap words. At least one word missing in the dictionary.
Command [arspoq]? q
Bye!
```

Task 9. Write a function "void `sortDict(char dict[MAXWORDS][MAXLENGTH])`" that orders all words in the dictionary contained in the array `dict` in alphabetical order.

Hint: The string compare function `strcmp` in the `string.h` library is very useful. See Appendix B3 of the k&R book.

Hint: The bubble sort algorithm that repeatedly compares two words and swaps them if necessary is probably the easiest way to sort the list, given that you already have the `swapWords` function. But you can also use the insertion sort, or even merge sort if you wish.

Of course you will have to extend the set of operations supported by your program. You can use the command 'o' for the operation 'sort dictionary'. The output of your program should then look as follows:

```
Command [arspoq]? a
Add a word: Apple
Command [arspoq]? a
Add a word: Union
Command [arspoq]? a
Add a word: Apple
Command [arspoq]? a
Add a word: Orange
Command [arspoq]? p
Dictionary:
- Apple
- Union
- Apple
- Orange
Command [arspoq]? o
Command [arspoq]? p
Dictionary:
- Apple
- Apple
- Orange
- Union
Command [arspoq]? q
Bye!
```

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