

Exam: Sort Word and Numbers (wordnum.c) – Time: [60 min]

Distance Study Electronics and Business – Computer Science 2

2021-06-17

Implement a C program that sorts numbers and skips equal words. For this, your program should allow the user to enter an arbitrary number of words and numbers with up to 63 characters. After entering "." the program should output the sorted numbers (starting with the smallest) and all words, if the words are equal only one should be printed.

```
10 word sky 45 10 word sky .  
  
500 wall floor 25 wall 10 Wall 1052 1052 1 .  
  
Green 100 green .
```

Input

```
word sky  
10 10 45  
  
wall floor Wall  
1 10 25 500 1052 1052  
  
Green green  
100
```

Output

.) Required procedure:

Use the procedure `scanf` for the input and first check whether it's a word or a number. Allocate memory on the heap for each word and each number. Trace them with two separated tables (See figure below). One only consists of words, and the other consists only of numbers (`wordTable` & `numTable`). Since we do not know the size of those tables, they should also be stored on the heap and resized using `realloc` if necessary. If you follow this strategy, the table is a pointer to multiple strings/numbers. The final datatype for those tables should be `char **` (a pointer of pointers to characters) and `int **` (a pointer of pointers to ints).

While reading the input, you need sort out duplicated words, which are present at least twice (case-sensitive → e.g. `sky` != `Sky`). Attention: At the end only one entry of the word identified as multiple should be present in the table.

For the `numTable`, you need to sort the numbers in ascending order (starting with the smallest). Conversions from `str` to `int`, use `atoi()` from `<stdlib.h>`.

Attention: In this case, you should not sort out any number, even if they are identical.

Finally for the output, simply print both tables (`wordTable` & `numTable`).

.) Base your program on functions with following names:

(1) allocAndReadInput()

- logic: read input, distinguish between word and number, check for duplicated words, allocate memory + reallocate if necessary and store in ****wordTable** / ****numTable**

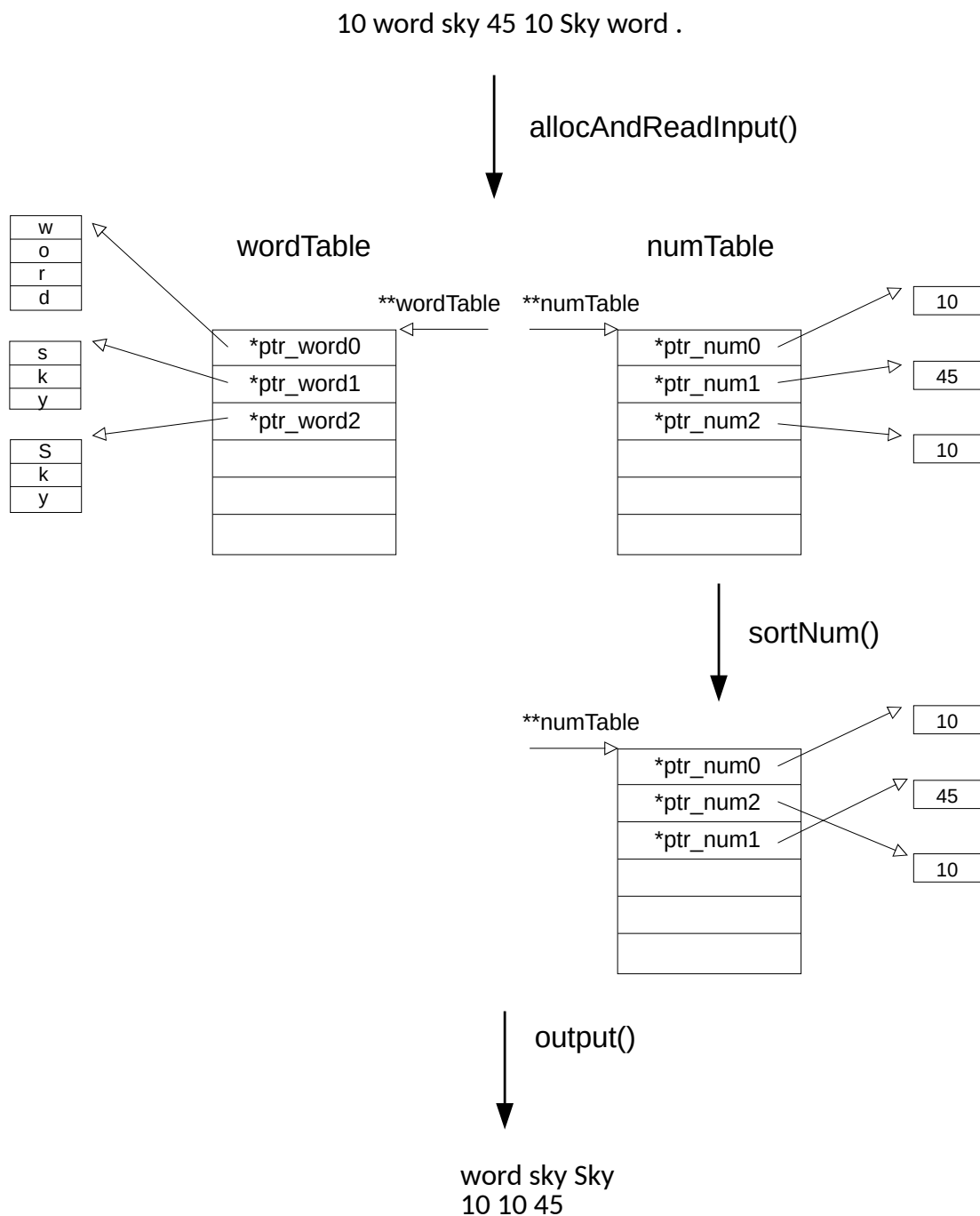
(2) sortNum()

- logic: e.g. bubble-sort, quick-sort, etc.

(3) output()

- logic: print both ****wordTable** & **numTable**

The figure below shows the program-flow with the required memory handling.



.) Grading (60 Points):

- 10: create and use wordTable (char**) & numTable (int**)
- 5: allocation of initial memory
- 5: encapsulate the logic of (1) in the function properly → allocAndReadInput()
- 5: distinguish between word and number
- 5: reallocate memory as needed
- 5: sort out duplicated words
- 5: sort the numTable with algorithm (in a function (2) → sortNum())
- 5: print both tables to console (in a function (3) → output())
- 5: free memory (all blocks need to be freed)
- 10: Code quality

Hint: You can use checkproject to check the behavior of your code. Test-cases are available in Moodle.