

EXERCISE 2

Last update 2024-11-08 (Version 4f0c4c13-dirty)

forksort

Implement an algorithm which sorts lines alphabetically.

SYNOPSIS

`forksort`

Instructions

The program takes multiple lines as input and sorts them by using a recursive variant of merge sort¹. The input is read from `stdin` and ends when an EOF (End Of File) is encountered.

Your program must accept any number of lines.

The program sorts the lines recursively, i.e. by calling itself:

1. If the input consist of only 1 line, then write it to `stdout` and exit with status `EXIT_SUCCESS`.
2. Otherwise the input consist of $n > 1$ lines. Split them into two parts, with each part consisting of $n/2$ lines. If n is odd, one of the parts will have one line more than the other.
3. Using `fork(2)` and `execlp(3)`, recursively execute this program in two child processes, one for each of the two parts. Use two unnamed pipes per child to redirect `stdin` and `stdout` (see `pipe(2)` and `dup2(3)`). Write the first part to `stdin` of one child and the second part to `stdin` of the other child. Read the respective sorted lines from each child's `stdout`. The two child processes must run simultaneously!
4. Use `wait(2)` or `waitpid(2)` to read the exit status of the children. Terminate the program with exit status `EXIT_FAILURE` if the exit status of any of the two child processes is not `EXIT_SUCCESS`.
5. Merge the sorted parts from the two child processes and write them to `stdout`. At each step, compare the next line of both parts and write the smaller one to `stdout`, such that the lines are written in ascending order. Use the byte values of the character representation for ordering. (The program must be able to sort input strings with the following characters `[0-9]` `[A-Z]` `[a-z]` `.` `:` `-` `!=?%` and whitespace. Ordering must be the same as using `LC_ALL=C sort`².) Terminate the program with exit status `EXIT_SUCCESS`.

Hints

- In order to avoid endless recursion³, fork only if the input number is greater than 1.
- To output error messages and debug messages, always use `stderr` because `stdout` is redirected in most cases.

¹https://en.wikipedia.org/wiki/Merge_sort

²<https://www.gnu.org/software/coreutils/sort>

³http://en.wikipedia.org/wiki/Fork_bomb

Examples

```
$ cat 1.txt
```

Heinrich

Anton

Theodor

Dora

Hugo

```
$ ./forksort < 1.txt
```

Anton

Dora

Heinrich

Hugo

Theodor

Mandatory testcases

Input shown in **blue** color. Output to *stdout* (and *stderr*) shown in black. (Note that in the following output sections `EXIT_SUCCESS` equals 0, and `EXIT_FAILURE` equals 1. Refer to `stdlib.h` for further details.) `^C` indicates `CTRL+C`, `^D` indicates `CTRL+D`. The placeholder `<usage message>` must be replaced by a proper usage message (printed to *stdout*), `<error message>` must be replaced by a meaningful error message (which is printed to *stderr*).

Testcase 01: one-line

```
1 $>echo -e 'Hello world!' | ./forksort
2 Hello world!
3 $>echo $?
4 0
```

Note: Updated to use single quotes (') instead of double quotes ("). (Double quotes caused the error `-bash: !": event not found` on some systems.)

Testcase 02: easy-1

```
1 $>echo -e "Timy\nBob\nTim\nAlice" | ./forksort
2 Alice
3 Bob
4 Tim
5 Timy
6 $>echo $?
7 0
```

Testcase 03: easy-2

```
1 $>echo -e "Wade\nDave\ndave\nDave\nSeth\nbob\nIvan\nRiley\nGilbert\nJorge\nDan\nBob\nMallory
2 Stan\nJohn\nEric\nDoe\nIan\nRayn\nSeth\nSimon" | ./forksort
3 Bob
4 Dan
5 Dave
6 Dave
7 Doe
8 Eric
9 Gilbert
10 Ian
11 Ivan
12 Johnic
13 Jorge
14 Mallory
15 Rayn
16 Riley
17 Seth
18 Seth
19 Simon
20 Stan
21 Wade
22 bob
23 dave
24 $>echo $?
25 0
```

Testcase 04: easy-3

```
1 $>echo -e "0123456789\nABC\nXYZ\nabc\nxyz\n.,:~!=?%\n0h\n,\nthis\nis?\neazy..\n" | ./forksort
2
3 ,
4 .,:~!=?%
5 0123456789
6 ABC
7 0h
8 XYZ
9 abc
10 eazy..
11 is?
12 this
13 xyz
14 $>echo $?
15 0
```

Testcase 05: 10lines

```
1 $>( for i in `seq 1 10`; do echo "Test$i" | shasum | cut -d " " -f 1; done; ) | ./forksort
2 0dcb0e28829bf50d5d29a89aef0a286600dfe425
3 11d52a479a6366103a619ed762383a95cda9e27c
4 4507bff5b7e37430a046a47090375317113be5ef
5 4c9b3febf76df122c088f2ad02bc1911d6a6ed2e
6 8214d4230894cb1e30815b74e4a5d11ac17d9fff
7 825e5bed29c2ee238b794b924cfd3371bc4d2d14
8 946240e35c5f63aa4d82f7eb975c4838fa63d3f7
9 9bd69ff3472986b590ef3e4b076173ee50be475e
10 ec42839a76a9c71e1532cd22f2889e6b6949d366
11 f8bdc9f9af885bcd59778a51645ab796deb63d82
```

Testcase 06: 1500lines

```
1 $>( for i in `seq 1 1500`; do echo "ABC$i" | shasum | cut -d " " -f 1; done; ) > testfile1500
2 $>head testfile1500
3 99895838ac8e88f9cf7298f83d6a3646f71edee7
4 01eac6c58451906a0cd808b19395721939019704
5 f7947595c4380b4e1be894e31d4f6da544fb671a
6 1defaf8cda727748df41874bcd312ddd57fc6b2d
7 33cb2d9abcb10910c565bb47a295ee492f3113a
8 d128fbffe081697ac9efc37ae432ccd15e926432
9 bbba036b1938725778d1b4b6be2f8f7263d5dff
10 beff53647e9dda54ea8978b4af4900d11f679afe
11 2cd71b3cfa040a9708841a299393d63bb75a1e3f
12 9fbb4cc39390fbeda7e190b199dd2f928cf8b270
13 $>shasum testfile1500
14 47fc2100e27c5a83a2c1873461c43d7b901633e3 testfile1500
15 $>./forksort < testfile1500 > sorted1500
16 $>echo $?
17 0
18 $>head sorted1500
19 00004d1d63f2ddcd30cb663048ad0578899c9c42
20 000846b4a4801a90b5c8e637a6bfce07f5248117
21 000cf886ca39e1a62dfd2c89fc2d0d2df1de810e
22 002be9ed3fd8f20fb4230dbde1f698a851235286
23 0039404e6dd4e9797c3b02bc184a627b2d0f4642
24 0042d3549186cbe9b10cb2df5edad9f871f7a495
25 008313e33d936c97032a997d5ff25077fa185770
26 010c90d8989dfa8f5fbfd914eb167afa53e94d96
27 010f985021070a2b74100c7b4ff5165e5dbf4ecb
28 016bede1f1f97f0a4cd49168dae0ee3594c2c1de
29 $>shasum sorted1500
30 104496999e5314b5c9edcee47cdbf9864497bf36 sorted1500
```

Testcase 07: med-lines

```
1 $(< for i in `seq 1 10`; do echo "X$i" | shasum | cut -d " " -f1 | tr -d \\n; printf -- "-%.0s" \
2 {1..2000}; echo "__end$i."; done; ) > longlines
3 $>cat longlines | tr -d "-"
4 0c589a604d5b4863f35f5e481c8d93360494343d__end1.
5 fc316cca30c54977b49ab2b12b9334978585546f__end2.
6 b97f3d209f74a54e9e6c5775ee7044c90b465ea1__end3.
7 544ea82727a6d3c6fb91d2fc591b0f5398b7d91d__end4.
8 166469e872d021db54cb33365bdf7a840e79da93__end5.
9 4755176afd3e10a684fe58b5d3b52869ca8c4dfc__end6.
10 adcd997d36f559cf17629ed965753fe1ba7e91c6__end7.
11 abafcbce88dfb1530024ae0629ddf7988cf88dd5__end8.
12 4dbd85541cf9dc118b6e26a51841e035f55b71a1__end9.
13 3ef847d592aa1ea423bc9f2d7207196fa236ef6d__end10.
14 $>shasum longlines
15 ff17b71f26893e9fd245b3c6b5c85fb6e63c3f66  longlines
16 $>./forksort < longlines > longsorted
17 $>echo $?
18 0
19 $>cat longsorted | tr -d "-"
20 0c589a604d5b4863f35f5e481c8d93360494343d__end1.
21 166469e872d021db54cb33365bdf7a840e79da93__end5.
22 3ef847d592aa1ea423bc9f2d7207196fa236ef6d__end10.
23 4755176afd3e10a684fe58b5d3b52869ca8c4dfc__end6.
24 4dbd85541cf9dc118b6e26a51841e035f55b71a1__end9.
25 544ea82727a6d3c6fb91d2fc591b0f5398b7d91d__end4.
26 abafcbce88dfb1530024ae0629ddf7988cf88dd5__end8.
27 adcd997d36f559cf17629ed965753fe1ba7e91c6__end7.
28 b97f3d209f74a54e9e6c5775ee7044c90b465ea1__end3.
29 fc316cca30c54977b49ab2b12b9334978585546f__end2.
30 $>shasum longsorted
ac5f138499777244b21d5306b62884e73e03f1e7  longsorted
```

Additional testcases

Input shown in **blue** color. Output to *stdout* (and *stderr*) shown in black. (Note that in the following output sections `EXIT_SUCCESS` equals 0, and `EXIT_FAILURE` equals 1. Refer to `stdlib.h` for further details.) `^C` indicates `CTRL+C`, `^D` indicates `CTRL+D`. The placeholder `<usage message>` must be replaced by a proper usage message (printed to *stdout*), `<error message>` must be replaced by a meaningful error message (which is printed to *stderr*). These testcases are not executed automatically in the submission system, but should be no problem for your implementation.

Testcase: additional-1

```
1 $>( a=""; for i in `seq 1 50`; do m=`echo $a | shasum | cut -b 1-5` a=$a$m; echo $a | rev; done; ) > \
    additional1
2 $>head -n 5 additional1
3 38cda
4 718e538cda
5 d920c718e538cda
6 58875d920c718e538cda
7 2640a58875d920c718e538cda
8 $>./forksort < additional1 > additional1sorted
9 $>echo $?
10 0
11 $>cat additional1 | LC_ALL=C sort > additional1reference
12 $>diff additional1sorted additional1reference
13 $>echo $?
14 0
```

Testcase: additional-2

```
1 $>echo -n "01234567890abcdefghijklmnopqrstuvwxyz.-!=% ABCDEFGHIJKLMNOPQRSTUVWXYZ" > allchars
2 $>cat allchars; echo "]"
3 01234567890abcdefghijklmnopqrstuvwxyz.-!=% ABCDEFGHIJKLMNOPQRSTUVWXYZ]
4 $>( for i in `cat allchars | sed "s/./\0\n/g"`; do echo $i | shasum | tr -d "\n"; echo -n "x"; cat \
    allchars | sed "s/./\0$i/g"; echo ""; done; ) > additional2
5 $>head -n 1 additional2
6 09d2af8dd22201dd8d48e5dcfaed281ff9422c7 \
    -x0010203040506070809000a0b0c0d0e0f0g0h0i0j0k0l0m0n0o0p0q0r0s0t0u0v0w0x0y0z0.0,0:0-0!0=0?0%0 \
    OAOBOCODOEOFOGOHOIOJOKOLOMONOONOPOQOROSOTOUOVOWOXOYOZO
7 $>./forksort < additional2 > additional2sorted
8 $>echo $?
9 0
10 $>cat additional2 | LC_ALL=C sort > additional2reference
11 $>diff additional2sorted additional2reference
12 $>echo $?
13 0
```

Testcase: additional-2500lines

```
1 $>( for i in `seq 1 2500`; do echo "Line$i" | shasum | cut -d " " -f 1; done; ) > testfile2500
2 $>head testfile2500
3 e43f9db6b4ac5664808daeaedcb490f4ed935273
4 aaaa2056cdd98db04d5b107cb53b9587e2f37d90
5 b59fea5df089db2b4b41e083efbb97e2d04294ba
6 195b8009c8b1dc6a6e47bb09f419587ca2acc0c8
7 cc6b0d47ffcd73375209f988213079ff9f929510
8 f82efcdee1c6567b237fde65c6c78f40aafcea83
9 39f3f18ac79e0378d867ab534979243c5a72d3cf
10 d388ed02fd03f5ec0e369848ff5125ce3a80c7a3
11 fea5369c6d5ecc7c711cabac3b918d66826b1d19
12 a67262d6ad1e77937f9c49e2cd8b7eff59d2fdad
13 $>shasum testfile2500
14 f24a88e0c6d5a435dc7ba491d1065e5f28ab63d0 testfile2500
15 $>./forksort < testfile2500 > sorted2500
16 $>echo $?
17 0
18 $>head sorted2500
19 00141605ec94515da1cadbf66348f7ea8eefcbf2
20 005085d8247aeab8e5728c158c7c29144f46adc1
21 005e68b41447d7241e17ed3bb30c5408628c937b
22 006abdc5d1b832163f44904f4edd0c3d9b105506
23 006cb5704e0df7ee03b8651a6be44f77c3f344a3
24 007fa262949b83ebc5e5fe479c30b57a735fc9e6
25 00ab1ac194c2b8c7778700f64b1bbc244ecac0c
26 00cf6b761b8ee7e1ee29362304643fbbb708efbe
27 00e97f2b955662c736c6825685a75420f8e3f1de
28 00ffc8553c4821cddd21cd038c187f92aa123438
29 $>shasum sorted2500
30 0655f27a8499018671d06a247391ce7eebce4cd2 sorted2500
```

Testcase: additional-long-lines

```
1 $>( for i in `seq 1 10`; do echo "X$i" | shasum | cut -d " " -f1 | tr -d \n; printf -- "-%.0s" \
2 {1..8000}; echo "__end$i."; done; ) > longlines
3 $>cat longlines | tr -d "-"
4 0c589a604d5b4863f35f5e481c8d93360494343d__end1.
5 fc316cca30c54977b49ab2b12b9334978585546f__end2.
6 b97f3d209f74a54e9e6c5775ee7044c90b465ea1__end3.
7 544ea82727a6d3c6fb91d2fc591b0f5398b7d91d__end4.
8 166469e872d021db54cb33365bdf7a840e79da93__end5.
9 4755176afd3e10a684fe58b5d3b52869ca8c4dfc__end6.
10 adcd997d36f559cf17629ed965753fe1ba7e91c6__end7.
11 abafcbce88dfb1530024ae0629ddf7988cf88dd5__end8.
12 4dbd85541cf9dc118b6e26a51841e035f55b71a1__end9.
13 3ef847d592aa1ea423bc9f2d7207196fa236ef6d__end10.
14 $>shasum longlines
15 36c34349bc0e86b6ed15bb159b3b59502391003a longlines
16 $>./forksort < longlines > longsorted
17 $>echo $?
18 0
19 $>cat longsorted | tr -d "-"
20 0c589a604d5b4863f35f5e481c8d93360494343d__end1.
21 166469e872d021db54cb33365bdf7a840e79da93__end5.
22 3ef847d592aa1ea423bc9f2d7207196fa236ef6d__end10.
23 4755176afd3e10a684fe58b5d3b52869ca8c4dfc__end6.
24 4dbd85541cf9dc118b6e26a51841e035f55b71a1__end9.
25 544ea82727a6d3c6fb91d2fc591b0f5398b7d91d__end4.
26 abafcbce88dfb1530024ae0629ddf7988cf88dd5__end8.
27 adcd997d36f559cf17629ed965753fe1ba7e91c6__end7.
28 b97f3d209f74a54e9e6c5775ee7044c90b465ea1__end3.
29 fc316cca30c54977b49ab2b12b9334978585546f__end2.
30 $>shasum longsorted
894345344b579a83c9c7d4a1db56b424a2526b8a longsorted
```

Recommendations and Guidelines

In order to make all solutions more uniform and to avoid problems with different C dialects, please try to follow the following guidelines.

General Guidelines

Try to follow the following guidelines closely.

1. All source files of your program(s) should compile with

```
$ gcc -std=c99 -pedantic -Wall -D_DEFAULT_SOURCE -D_BSD_SOURCE -D_SVID_SOURCE  
-D_POSIX_C_SOURCE=200809L -g -c filename.c
```

without *warnings and info messages* and your program(s) should link without warnings too.
2. There should be a Makefile implementing the targets: **all** to build the program(s) (i.e. generate executables) from the sources (this must be the first target in the Makefile); **clean** to delete all files that can be built from your sources with the Makefile.
3. Make all targets of your Makefile idempotent. I.e. execution of **make clean**; **make clean** should yield the same result as **make clean**, and should not fail with an error.
4. The program shall operate according to the specification/assignment without major issues (e.g., segmentation fault, memory corruption).
5. Arguments should be parsed according to UNIX conventions (we strongly encourage the use of **getopt(3)**). The program should conform to the given synopsis/usage in the assignment. If the synopsis is violated (e.g., unspecified options or too many arguments), the program should terminate with the usage message containing the program name and the correct calling syntax. Argument handling should also be implemented for programs without arguments.
6. Correct (=normal) termination, including a cleanup of resources.
7. Upon success the program shall terminate with exit code 0, in case of errors with an exit code greater than 0. We recommend to use the macros **EXIT_SUCCESS** and **EXIT_FAILURE** (defined in **stdlib.h**) to enable portability of the program.
8. If a function indicates an error with its return value, it *should* be checked in general. If the subsequent code depends on the successful execution of a function (e.g. resource allocation), then it is very *important* to check the return value.
9. Functions that do not take any parameters should be declared with **void** in the signature, e.g., **int get_random_int(void);**.
10. Procedures (i.e., functions that do not return a value) should be declared as **void**.
11. Error messages shall be written to **stderr** and should contain the program name **argv[0]**.
12. Avoid the use of the functions: **gets**, **scanf**, **fscanf**, **atoi** and **atol** to avoid crashes due to invalid inputs.
13. Documentation is a nice bonus. Format the documentation in Doxygen style (see Wiki and Doxygen's intro).
14. Write meaningful comments. For example, meaningful comments describe the algorithm, or why a particular solution has been chosen, if there seems to be an easier solution at a first glance. Avoid comments that just repeat the code itself (e.g., **i = i + 1; /* i is incremented by one */**).

AVOID	USE INSTEAD
gets	fgets
scanf	fgets, sscanf
fscanf	fgets, sscanf
atoi	strtol
atol	strtol

15. The documentation of a module should include: name of the module, name and student id of the author (`@author` tag), purpose of the module (`@brief`, `@details` tags) and creation date of the module (`@date` tag).

Also the Makefile should include a header, with author and program name at least.

16. Each function shall be documented either before the declaration or the implementation. It should include purpose (`@brief`, `@details` tags), description of parameters and return value (`@param`, `@return` tags) and description of global variables the function uses (`@details` tag).

You should also document `static` functions (see `EXTRACT_STATIC` in the file `Doxyfile`). Document visible/exported functions in the header file and local (`static`) functions in the C file. Document variables, constants and types (especially `structs`) too.

17. Documentation, names of variables and constants is best done in English.
18. Internal functions shall be marked with the `static` qualifier and are not allowed to be exported (e.g., in a header file). Only functions that are used by other modules shall be declared in the header file.
19. All exercises shall be solved with functions of the C standard library. If a required function is not available in the standard library, you can use other (external) functions too. Avoid reinventing the wheel (e.g., re-implementation of `strcmp`).
20. Name of constants shall be written in upper case, names of variables in lower case (maybe with first letter capital).
21. Use meaningful variable and constant names (e.g., also semaphores and shared memories).
22. Avoid using global variables as far as possible.
23. All boundaries shall be defined as constants (macros). Avoid arbitrary boundaries. If boundaries are necessary, treat its crossing.
24. Avoid side effects with `&&` and `||`, e.g., write `if(b != 0) c = a/b;` instead of `if(b != 0 && c = a/b).`
25. Each `switch` block should contain a `default` case. If the case is not reachable, write `assert(0)` to this case (defensive programming).
26. Logical values shall be treated with logical operators, numerical values with arithmetic operators (e.g., test 2 strings for equality by `strcmp(...) == 0` instead of `!strcmp(...)`).
27. Indent your source code consistently (there are tools for that purpose, e.g., `indent`).
28. Avoid tricky arithmetic statements. Programs are written once, but read many times. Your program is not always better if it is shorter!
29. For all I/O operations (read/write from/to `stdin`, `stdout`, files, sockets, pipes, etc.) use *either* standard I/O functions (`fdopen(3)`, `fopen(3)`, `fgets(3)`, etc.) *or* POSIX functions (`open(2)`, `read(2)`, `write(2)`, etc.). Remember, standard I/O functions are buffered. Mixing standard I/O functions and POSIX functions to access a common file descriptor can lead to undefined behaviour and is therefore a bad idea.

30. If asked in the assignment, you should implement signal handling (`SIGINT`, `SIGTERM`). You must only use *async-signal-safe* functions in your signal handlers.
31. Close files, free dynamically allocated memory, and remove resources after usage.
32. Don't waste resources due to inconvenient programming. Header files shall not include implementation parts (exception: macros).
33. To comply with the given testcases, the program output must exactly match the given specification. Therefore you should only print debug information if the compile flag `-DDEBUG` is set, in order to avoid problems with the automatic checker.

Exercise 2 Guidelines

The following points are import for exercise 2.

1. Correct use of `fork/exec/pipes` as taught in the lectures. For example, do not exploit inherited memory areas.
2. Ensure termination of child processes without `kill(2)` or `killpg(2)`. Collect the exit codes of child processes (`wait(2)`, `waitpid(2)`, `wait3(2)`).