191.002 VU Betriebssysteme
EXERCISE 1B

Last update 2023-10-17 (Version 2c77a055)

Aufgabenstellung – 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 sort[1]. 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 recursion[3], 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.

[1]`https://en.wikipedia.org/wiki/Merge_sort`
[2]`https://www.gnu.org/software/coreutils/sort`
[3]`http://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

```
$>echo -e 'Hello world!' | ./forksort
Hello world!
$>echo $?
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

```
$>echo -e "Timy\nBob\nTim\nAlice" | ./forksort
Alice
Bob
Tim
Timy
$>echo $?
0
```

Testcase 03: easy-2

```
$>echo -e "Wade\nDave\ndave\nDave\nSeth\nbob\nIvan\nRiley\nGilbert\nJorge\nDan\nBob\nMallory\nStan \
     \nJohn\nEric\nDoe\nIan\nRayn\nSeth\nSimon" | ./forksort
Bob
Dan
Dave
Dave
Doe
Gilbert
Ian
Ivan
Johnic
Jorge
Mallory
Rayn
Riley
Seth
Seth
Simon
Stan
Wade
bob
dave
$>echo $?
0
```

Testcase 04: easy-3

```
$>echo -e "0123456789\nABC\nXYZ\nabc\nxyz\n.,:-!=?%\nOh\n,\nthis\nis?\neazy..\n" | ./forksort

,
.,:-!=?%
0123456789
ABC
Oh
XYZ
abc
eazy..
is?
this
xyz
$>echo $?
0
```

Testcase 05: 10lines

```
$>( for i in `seq 1 10`; do echo "Test$i" | sha1sum | cut -d " " -f 1; done; ) | ./forksort
0dcb0e28829bf50d5d29a89aef0a286600dfe425
11d52a479a6366103a619ed762383a95cda9e27c
4507bff5b7e37430a046a47090375317113be5ef
4c9b3febf76df122c088f2ad02bc1911d6a6ed2e
8214d4230894cb1e30815b74e4a5d11ac17d9fff
825e5bed29c2ee238b794b924cfd3371bc4d2d14
946240e35c5f63aa4d82f7eb975c4838fa63d3f7
9bd69ff3472986b590ef3e4b076173ee50be475e
ec42839a76a9c71e1532cd22f2889e6b6949d366
f8bdc9f9af885bcd59778a51645ab796deb63d82
```

Testcase 06: 1500lines

```
$>( for i in `seq 1 1500`; do echo "ABC$i" | sha1sum | cut -d " " -f 1; done; ) > testfile1500
$>head testfile1500
99895838ac8e88f9cf7298f83d6a3646f71edee7
01eac6c58451906a0cd808b19395721939019704
f7947595c4380b4e1be894e31d4f6da544fb671a
1defaf8cda727748df41874bcd312ddd57fc6b2d
33cb2d9abcba10910c565bb47a295ee492f3113a
d128fbffe081697ac9efc37ae432ccd15e926432
bbba036b1938725778d1b4b6be2f8f7263d5dffc
beff53647e9dda54ea8978b4af4900d11f679afe
2cd71b3cfa040a9708841a299393d63bb75a1e3f
9fbb4cc39390fbeda7e190b199dd2f928cf8b270
$>sha1sum testfile1500
47fc2100e27c5a83a2c1873461c43d7b901633e3  testfile1500
$>./forksort < testfile1500 > sorted1500
$>echo $?
0
$>head sorted1500
00004d1d63f2ddcd30cb663048ad0578899c9c42
000846b4a4801a90b5c8e637a6bfce07f5248117
000cf886ca39e1a62dfd2c89fc2d0d2df1de810e
002be9ed3fd8f20fb4230dbde1f698a851235286
0039404e6dd4e9797c3b02bc184a627b2d0f4642
0042d3549186cbe9b10cb2df5edad9f871f7a495
008313e33d936c97032a997d5ff25077fa185770
010c90d8989dfa8f5fbfd914eb167afa53e94d96
010f985021070a2b74100c7b4ff5165e5dbf4ecb
016bede1f1f97f0a4cd49168dae0ee3594c2c1de
$>sha1sum sorted1500
104496999e5314b5c9edcee47cdbf9864497bf36  sorted1500
```

Testcase 07: med-lines

```
$>( for i in `seq 1 10`; do echo "X$i" | sha1sum | cut -d " " -f1 | tr -d \\n; printf -- "-%.0s" \
    {1..2000}; echo "__end$i."; done; ) > longlines
$>cat longlines | tr -d "-"
0c589a604d5b4863f35f5e481c8d93360494343d__end1.
fc316cca30c54977b49ab2b12b9334978585546f__end2.
b97f3d209f74a54e9e6c5775ee7044c90b465ea1__end3.
544ea82727a6d3c6fb91d2fc591b0f5398b7d91d__end4.
166469e872d021db54cb33365bdf7a840e79da93__end5.
4755176afd3e10a684fe58b5d3b52869ca8c4dfc__end6.
adcd997d36f559cf17629ed965753fe1ba7e91c6__end7.
abafcbce88dfb1530024ae0629ddf7988cf88dd5__end8.
4dbd85541cf9dc118b6e26a51841e035f55b71a1__end9.
3ef847d592aa1ea423bc9f2d7207196fa236ef6d__end10.
$>sha1sum longlines
ff17b71f26893e9fd245b3c6b5c85fb6e63c3f66  longlines
$>./forksort < longlines > longsorted
$>echo $?
0
$>cat longsorted | tr -d "-"
0c589a604d5b4863f35f5e481c8d93360494343d__end1.
166469e872d021db54cb33365bdf7a840e79da93__end5.
3ef847d592aa1ea423bc9f2d7207196fa236ef6d__end10.
4755176afd3e10a684fe58b5d3b52869ca8c4dfc__end6.
4dbd85541cf9dc118b6e26a51841e035f55b71a1__end9.
544ea82727a6d3c6fb91d2fc591b0f5398b7d91d__end4.
abafcbce88dfb1530024ae0629ddf7988cf88dd5__end8.
adcd997d36f559cf17629ed965753fe1ba7e91c6__end7.
b97f3d209f74a54e9e6c5775ee7044c90b465ea1__end3.
fc316cca30c54977b49ab2b12b9334978585546f__end2.
$>sha1sum 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

```
$>( a=""; for i in `seq 1 50`; do m=`echo $a | sha1sum | cut -b 1-5` a=$a$m; echo $a | rev; done; ) > \
     additional1
$>head -n 5 additional1
38cda
718e538cda
d920c718e538cda
58875d920c718e538cda
2640a58875d920c718e538cda
$>./forksort < additional1 > additional1sorted
$>echo $?
0
$>cat additional1 | LC_ALL=C sort > additional1reference
$>diff additional1sorted additional1reference
$>echo $?
0
```

Testcase: additional-2

```
$>echo -n "01234567890abcdefghijklmnopqrstuvwxyz.,:-!=?% ABCDEFGHIJKLMNOPQRSTUVWXYZ" > allchars
$>cat allchars; echo "]"
01234567890abcdefghijklmnopqrstuvwxyz.,:-!=?% ABCDEFGHIJKLMNOPQRSTUVWXYZ]
$>( for i in `cat allchars | sed "s/./\0\n/g"`; do echo $i | sha1sum | tr -d "\n"; echo -n "x"; cat \
     allchars | sed "s/./\0$i/g"; echo ""; done; ) > additional2
$>head -n 1 additional2
09d2af8dd22201dd8d48e5dcfcaed281ff9422c7  \
     -x00102030405060708090000a0b0c0d0e0f0g0h0i0j0k0l0m0n0o0p0q0r0s0t0u0v0w0x0y0z0.0,0:0-0!0=0?0%0 \
     0A0B0C0D0E0F0G0H0I0J0K0L0M0N0O0P0Q0R0S0T0U0V0W0X0Y0Z0
$>./forksort < additional2 > additional2sorted
$>echo $?
0
$>cat additional2 | LC_ALL=C sort > additional2reference
$>diff additional2sorted additional2reference
$>echo $?
0
```

Testcase: additional-2500lines

```
$>( for i in `seq 1 2500`; do echo "Line$i" | sha1sum | cut -d " " -f 1; done; ) > testfile2500
$>head testfile2500
e43f9db6b4ac5664808daeaedcb490f4ed935273
aeaa2056cdd98db04d5b107cb53b9587e2f37d90
b59fea5df089db2b4b41e083efbb97e2d04294ba
195b8009c8b1dc6a6e47bb09f419587ca2acc0c8
cc6b0d47ffcd73375209f988213079ff9f929510
f82efcdee1c6567b237fde65c6c78f40aafcea83
39f3f18ac79e0378d867ab534979243c5a72d3cf
d388ed02fd03f5ec0e369848ff5125ce3a80c7a3
fea5369c6d5ecc7c711cabac3b918d66826b1d19
a67262d6ad1e77937f9c49e2cd8b7eff59d2fdad
$>sha1sum testfile2500
f24a88e0c6d5a435dc7ba491d1065e5f28ab63d0  testfile2500
$>./forksort < testfile2500 > sorted2500
$>echo $?
0
$>head sorted2500
00141605ec94515da1cadbf66348f7ea8eefcbf2
005085d8247aeab8e5728c158c7c29144f46adc1
005e68b41447d7241e17ed3bb30c5408628c937b
006abdc5d1b832163f44904f4edd0c3d9b105506
006cb5704e0df7ee03b8651a6be44f77c3f344a3
007fa262949b83ebc5e5fe479c30b57a735fc9e6
00ab1ac194c2b8c7778700f64b1bbc244ecacb0c
00cf6b761b8ee7e1ee29362304643fbbb708efbe
00e97f2b955662c736c6825685a75420f8e3f1de
00ffc8553c4821cddd21cd038c187f92aa123438
$>sha1sum sorted2500
0655f27a8499018671d06a247391ce7eebce4cd2  sorted2500
```

Testcase: additional-long-lines

```
$>( for i in `seq 1 10`; do echo "X$i" | sha1sum | cut -d " " -f1 | tr -d \\n; printf -- "-%.0s" \
    {1..8000}; echo "__end$i."; done; ) > longlines
$>cat longlines | tr -d "-"
0c589a604d5b4863f35f5e481c8d93360494343d__end1.
fc316cca30c54977b49ab2b12b9334978585546f__end2.
b97f3d209f74a54e9e6c5775ee7044c90b465ea1__end3.
544ea82727a6d3c6fb91d2fc591b0f5398b7d91d__end4.
166469e872d021db54cb33365bdf7a840e79da93__end5.
4755176afd3e10a684fe58b5d3b52869ca8c4dfc__end6.
adcd997d36f559cf17629ed965753fe1ba7e91c6__end7.
abafcbce88dfb1530024ae0629ddf7988cf88dd5__end8.
4dbd85541cf9dc118b6e26a51841e035f55b71a1__end9.
3ef847d592aa1ea423bc9f2d7207196fa236ef6d__end10.
$>sha1sum longlines
36c34349bc0e86b6ed15bb159b3b59502391003a  longlines
$>./forksort < longlines > longsorted
$>echo $?
0
$>cat longsorted | tr -d "-"
0c589a604d5b4863f35f5e481c8d93360494343d__end1.
166469e872d021db54cb33365bdf7a840e79da93__end5.
3ef847d592aa1ea423bc9f2d7207196fa236ef6d__end10.
4755176afd3e10a684fe58b5d3b52869ca8c4dfc__end6.
4dbd85541cf9dc118b6e26a51841e035f55b71a1__end9.
544ea82727a6d3c6fb91d2fc591b0f5398b7d91d__end4.
abafcbce88dfb1530024ae0629ddf7988cf88dd5__end8.
adcd997d36f559cf17629ed965753fe1ba7e91c6__end7.
b97f3d209f74a54e9e6c5775ee7044c90b465ea1__end3.
fc316cca30c54977b49ab2b12b9334978585546f__end2.
$>sha1sum longsorted
894345344b579a83c9c7d4a1db56b424a2526b8a  longsorted
```

Coding Rules and Guidelines

Your score depends upon the compliance of your submission to the presented guidelines and rules. Violations result in deductions of points. Hence, before submitting your solution, go through the following list and check if your program complies.

Rules

Compliance with these rules is essential to get any points for your submission. A violation of any of the following rules results in 0 points for your submission.

1. All source files of your program(s) must compile via
   ```
   $ gcc -std=c99 -pedantic -Wall -D_DEFAULT_SOURCE -D_BSD_SOURCE -D_SVID_SOURCE
                                  -D_POSIX_C_SOURCE=200809L -g -c filename.c
   ```
   without *errors* and your program(s) must link without *errors*. The compilation flags must be used in the Makefile. The feature test macros must not be bypassed (i.e., by undefining these macros or adding some in the C source code).
2. The functionality of the program(s) must conform exactly to the assignment. The program(s) shall operate according to the specification/assignment given the test cases in the respective assignment. Additional white spaces or any other deviation from the specified input and output format may lead to a failure of the respective test case.

General Guidelines

Violation of following guidelines leads to a deduction of points.

1. All source files of your program(s) must 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) must link without warnings.
2. There must 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. All targets of your Makefile must be idempotent. I.e. execution of `make clean; make clean` must yield the same result as `make clean`, and must 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 have to be parsed according to UNIX conventions (we strongly encourage the use of `getopt(3)`). The program has to conform to the given synopsis/usage in the assignment. If the synopsis is violated (e.g., unspecified options or too many arguments), the program has to 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 has to 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 the return value *must* be checked.

9. Functions that do not take any parameters have to 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) have to be declared as `void`.

11. Error messages shall be written to `stderr` and should contain the program name `argv[0]`.

12. It is forbidden to use the functions: `gets`, `scanf`, `fscanf`, `atoi` and `atol` to avoid crashes due to invalid inputs.

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

13. Documenation is mandatory. 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 */`).

15. The documentation of a module must 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 has to 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 `struct`s) too.

17. Documentation, names of variables and constants shall be 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 fist 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 must 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 more times. Your program is not 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 forbidden.
30. If asked in the assignment, you must 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 are only allowed to to print any debug information if the compile flag `-DDEBUG` is set.

Exercise 1B Guidelines

Violation of following guidelines leads to a deduction of points in exercise 1B.

1. Correct use of fork/exec/pipes as tought 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)`).