# Minishell edgy cases

Check every so often if there are leaks with "leaks minishell"
Also check for open fds with "lsof -c minishell"

1. General stuff, checking segfault and other weird behaviour
    a. Check only spaces, tabs, vertical tabs and newlines as input
    b. Syntax errors
        i. Pipe | at start or at the end
        ii. Two pipes after each other
        iii. Redirect before pipe or at the end of line
        iv. Redirect after redirect: < <, <<<, <>, <>>, ><, > >, > >>
        v. Unclosed quotes
    c. Check input without spaces between metacharacters
        i. cat<Makefile
        ii. Cat|wc

2. Builtins
    a. echo
        i. echo hi
        ii. echo
        iii. echo -n
        iv. echo -nnnnnnn
        v. echo -n -n hi
        vi. echo -n hi
        vii. echo -nnn -n hi
        viii. echo -nnnnnnna
        ix. echo -nnn -n hi -n
        x. echo hi -n
        xi. echo echo
        xii. echo –nnn
        xiii. echo -nnna -n hi
    b. cd, check if directories changes, do with pwd so you're testing pwd at the same time. Check also if PWD and OLDPWD change
        i. cd .
        ii. cd ..
        iii. cd -
        iv. cd
        v. cd /../../..
        vi. cd to file — exit code 1
        vii. cd to something nonexistent — exit code 1
        viii. Change permissions of dir and cd to it — exit code 1
        ix. mkdir, go inside, remove it and then try cd — shouldn't segfault and it would probably be good if you're still able to somehow exit
    c. export, echo the variable afterwards
        i. export asd=asd
        ii. export asd=asd asd1=asd1 asd3=asd4
        iii. export asd=asd
        export asd+=asd
        iv. export asd-a=asd — exit code 1
        names of variables should only have alphanumeric and _
        v. export asd=asd ads-asd=asd asd+asd=asd asd1=asd1
        Should print 2 errors and export 2 variables
        vi. export ads=
        vii. export lop
        viii. export

d. unset
    i. unset asdasdasdasdasd
    ii. unset multiple variables
    iii. unset
    iv. unset asd-a exit code 1
    Should be alphanumeric or _
    v. unset first variable
    vi. if you feel like it, unset everything and see if you get any segfaults
e. env
    i. Compare at startup of minishell if it has the same amount of variables
    ii. Env with argument, just shouldn't crash
f. Exit, check with exit codes
    i. exit should exit with last exit code, try echo then exit
    ii. exit 1
    iii. exit 20
    iv. exit 255
    v. exit 123 1
    Shouldnt exit, exit code 1
    vi. exit asd
    Should exit with 255
    vii. exit asd 1
    Should exit with 255
g. Unset PATH variable and check if builtins still work, they should
h. Check builtins in pipes to see if they adjust the environment
    i. cd | ls
    Should not change directory nor adjust PWD or OLDPWD
    ii. export asd=asd | echo $asd
    Should not have exported asd
    iii. unset PATH | echo $PATH
    PATH should still exist
    iv. exit | ls
    Should not exit
i. Check if builtins aren't run by just running the executable, I guess look at the code or try flags that are outside of the scope of the project

3. Signals
    a. Ctrl+d
        i. With empty prompt, should exit shell with last exit code
        Try running "echoa", then ctrl+d, should be exit code 127
        ii. With something in prompt does nothing
    b. Ctrl+c
        i. Should redisplay the prompt, doesn't matter if there's something already written
        ii. Sets exit code to 1 - defendable if you don't do it
        iii. Should not show ^C in interactive mode, does show it for blocking commands (cat, wc etc.)
        iv. Run heredoc, multiple ones and see if ctrl+c stops them, should have no ^C exit code 1
        v. Run cat and ctrl+c should stop cat exit code 130
        vi. Run cat | wc | echo lol write something then ctrl +c, should display lol, wc should not showup
        vii. Run minishell in minishell, then try ctrl+c
    c. Ctrl+\
        i. Should do nothing in interactive, nothing in heredoc
        ii. Should not show ^\ unless in a blocking command, shows ^\Quit: 3
        iii. In blocking command quits the command and set error code to 131
        iv. Run cat | wc | echo lol then ctrl+\
        v. Run minishell in minishell then ctrl+\
    d. Top, man command

        Top changes terminal settings and if you interrupt it with a signal it won't change them back. So run it then do ctrl+c or ctrl+\ and see what happens.

4. Exit codes and errors
    a. Permissions
        i. Make file, remove read permissions
            Cat file exit code 1
            < file exit code 1
        ii. Remove write permissions
            Echo hi > file exit code 1
        iii. Make script, remove execute permissions
            Run script exit code 126
    b. Wrong command 127
    c. Run "/' should show it as a dir and 126
    d. ./asdasd no such file or dir 127
    e. /bin/ls/asdasd not a directory 126
    f. echoa | echoa | cat exit code 0
    g. echo | echoa exit code 127

5. Quotes, environment vars
    a. cat 'xd'
    b. cat "xd"
    c. cat ""
    d. cat ''
    e. cat "|" echo in general try metachacters between quotes, they shouldn't be interpreted.
    f. cat '|' echo
    g. echo "> hi < hi"
    h. echo '> hi < hi'
    i. echo '$USER'
    j. echo "$USER"
    k. echo $USER
    l. echo $USER$USER
    m. echo $USER'$USER'"$USER"
    n. echo "$U"SER
    o. echo $"USER"
    p. echo $'USER'
    q. echo " ' "
    r. echo ' " '
    s. echo """""""""""
    t. echo $U$S$E$R
    u. echo $? You've probably been using this so far so it's been tested well
    v. echo $? $?
    w. export asd='$'
       echo $asdUSER
       echo $asd$asd$asd"USER"
    x. export asd="'"
       echo $asd
       echo $asd$USER$asd
       echo $asd$asd$asd$USER
       cat $asd
    y. export asd=$USER
       echo $asd
    z. export asd='$USER'
       echo $asd
       echo $asd$USER
    aa. export asd=echo
       $asd hi
       "$asd" hi
    bb. export asd=n
       echo -$asd hi
       echo -$asd$asd$asd$asd hi

## More advanced stuff

cc. echo '"$HOME"'
dd. echo "'$HOME'"
ee. export asd="po po"
    cat $asd
    cat$asd$asd'$asd'asd
ff. Export asd="po' ' 'po' "
    cat $asd
    cat $asd$asdasd$asd"$asd"
gg. export asd="123|2<ld[]JV|"
    echo $asd
hh. echo '"'-"n' hi
ii. echo '-'""""""""""""""""""""""""""""n"'' hi
jj. echo '-'""""""""""""""""""""""""""n""""""" hi

## 6. Redirections

a. echo hi > lol
b. echo hi >> lol
c. cat < lol
d. echo hi > tmp >tmp2 >tmp3 creates all files, throws it into tmp3
e. echo hi >>tmp >> tmp2 >>tmp3 same thing, now should have 2"hi"`s
f. remove permissions on tmp and run
   echo hi > tmp > tmp4 should error, no tmp4 is created
g. cat < Makefile > tmp
h. cat < Makefile <tmp3
i. cat <asdasdasda <tmp3
j. cat < tmp3 > tmp3 should be nothing in tmp3
k. echo >tmp hi <tmp2 hello < Makefile hahahha
   should write "hi hello hahahha" into tmp
l. try to throw in redirections at beginning of command, end, middle, in between arguments and so on to see if it's parsed correctly
m. echo hi > tmp | cat < tmp just don't crash i suppose
n. echo hi <asdadasdasdasdasda | echo huh should have error and display huh
   this is to check if you handle the redirections in your children, essentially an error should not stop the whole pipeline, it just doesn't run one command in the pipeline
o. try redirections with builtins, both in pipelines and outside
p. >temp creates file, doesn't run anything, this is checking if you crash somewhere essentially
q. <Makefile
r. <asdasdasdasdasd
s. >tmp5 | > tmp6 | >tmp7 creates all three files, runs no command obviously

7. Heredocs, I consider these a whole different monster
   Obviously try to write something and then the delimiter if everything works correctly
   a. cat << hi
   b. cat << hi
      try to finish the heredoc with:
      "hi "
      " hi"
      "h"
      "hii"
      "hihi"
   c. cat << ""  should end heredoc on an empty line
   d. << hi
   e. heredocs also end when giving ctrl+d
   f. cat << h << k
      handles both only the one with the k is gonna be redisplayed
   g. cat << h < Makefile
      handles heredoc and cats from Makefile
   h. cat <<h << l << k <<p
   i. ls | ls | ls | ls << hi
      handles heredoc first, before running any commands
      if you interrupt the heredoc with ctrl+c it won't run anything either
   j. cat << $USER
      $USER should not be expanded, try to write the expanded $USER and just $USER and see when the heredoc stops
      The delimiter never gets expanded
   k. cat << hi
      try to write environment variables in the heredoc
      $USER
      "$USER"
      '$USER'
      these always get expanded and the quotes do not get removed
   l. cat << "hi"
      now try environment variables
      these should never get expanded when the delimiter contains any sort of quotes, both single and double
   m. identify how heredocs are implemented
      i. if it's a file, see how they name the file, if it gets deleted afterward, if it gets deleted in case of signal interruption
         if there is multiple heredocs, do all files get deleted and so one
         Look at the naming of the files, where are they put and try to create a file with that name there, does this break the minishell (this is probably a bit nitpicky)
      ii. if it's a pipe, write a lot in the heredoc and see if the pipe gets full, what happens

n. check for more than 16 heredocs, bash says this is too much and exits the session for some reason.
o. another weird one is
<<h << k cat | echo > > | << k

## 8. Pipes

a. cat | cat | ls  checks if pipes are closed correctly
b. echo hi | cat | ls
c. echo hello hi | echo hello hi | wc
d. wc | sleep 5
e. sleep 5 | sleep 5
f. time sleep 5 | time sleep 6
g. cate | cate  |cate | cate
h. cate | cate | cate | cate | echo hi
i. cate | echo $? | cat
j. cate | cat asdasdasd  echo $?  1
k. cate | cat < asdasdsa  echo $?  1
l. a lot of pipes, so fork goes wrong
m. check pipes sometime with lsof -c minishell

## 9. Unset PATh, executables

a. make a script
    i. run with a PATH
        1. ./script should run
        2. script should not run
    ii. run without PATH
        1. script should work
        2. ./script as well
    iii. change name to ls
    iv. with Path
        1. ./ls should run script
        2. ls runs actual ls
    v. without PATH
        1. ls runs script
        2. ./ls run script
    vi. add PWD to PATH variable and now run ls
        This should still run the ls from /bin/ls
        This is to check if PATH is done from left to right
b. unset PATH /bin/ls
c. unset PATH ls
d. ./wc shouldnt run anything
e. export PATH to nothing and check if it segfaults or runs anything
f. save PATH, unset it then set it back to see if it still works
g. unset PATH and run builtins
h. unset PATH, run ./minishell and see what happens
i. go up a directory and run previous script through a relative path
    eval/script
    this is to check relative paths aren't just implemented through ./script, but also work from different dirs

10. Go wild. There is a bunch of things here that might be failable or not depending on the defense of the evaluees.
    Ask about stuff they might have implemented that I missed, it's the worst feeling when you tried hard at something and the evaluator misses it.
    a. is tilde expansion implemented (outside scope)
    b. SHLVL (outside scope)
    c. export TEST="123||2<>ld[[]JC|"
       echo $TEST
       cat $TEST
       export TEST="$TEST"
       cat $TEST
       export TEST="$TEST$TEST"
       cat $TEST
    d. "e"x'p'"o"'r"t' "T"=hi
       echo $T
    e. export T="|"
       echo hi $T cat
       should not run in pipeline
    f. ""
    g. "  "
    h. sleep 0 | cat | cat
    i. cat /dev/random | cat
    j. ifconfig | cat | awk -v count=20 -v FS="f" 'NR==1,NR==5{print $1} END {count="active"} END{print count}'
       should print 5 lines until the letter "f" and at the end "active"