Design Doc 1.1 A1

Process to go through 1 command = Exe

1 Set the command flag using
check ~~pipe~~ Args (args)

```
    if there are any pipes in args
        cmdFlag = 2
    else if there's a command & it's not exit
        cmdFlag = 1
    else
        cmdFlag = 0
```

2 open any files, set the file descriptor,
& get the position of the operation symbol
ex. > >> < | using
SetFile Descriptors(args, fileName, fileDesc, operatorPos)

```
    fileDesc = open( fileName, O_CREAT, O_WRONLY, )
    operatorPos = i
```

3 Use the command flag to pick which
command to execute

```
    if cmdFlag == 1
        exec simpleCmd (args)
    else if cmdFlag == 2
        ~~exec~~ parse pipe using
~~pipeCmd1~~    parsePipe1 (args, cmd1) &
        parsePipe2 (args, cmd2)

        execPipedCmd ( ~~pipeCmd1~~ cmd1, ~~pipeCmd2~~ cmd2 )

                ex: ls -al | more
```

when doing multiple commands separated
by semicolon,
we parse args if block
to get each individual command to splitArgs

```
if args != NULL && args != exit {
    numCmds = 1
    splitArgs = NULL all [illegible] strs
ctr = counter = 0
   for (i=0 ; i < argLength ; i++)
   {
       if args[i] != ";"
           splitArgs [counter] = args[i]
           counter ++
       else
           there must be more than 1 command
           numCmds ++
           Exe
           reset numCmds = 1
           reset counter = 0 to get next cmd
           reset splitArgs to NULL
   }
```

The last cmd of multiple cmds won't
end in ";" , add a NULL to that str

```
  ~~[illegible]~~ if numCmds > 1
           splitArgs [ctr] = NULL
           ~~reset counter = 0~~ Exe
           reset counter = 0
```

ex. sort -nr < grades ; ls -l > bar ↑ NULL ctr

if numCmds == 1 Exe ~~[illegible]~~ like normal }

Example Commands = cmds

EX.1
```
ls  -l  >  bar   = args
i = 0   1  2  3

cmdFlag = checkArgs(args)

no pipe, cmdFlag = 1

setFileDescriptors(args, fileName, fileDesc, operatorPos)
    fileName = args[3]
    fileDesc = open(fileName, O_CREAT|O_WRONLY, )
    operatorPos = 2 = i

cmdFlag == 1
    execSimpleCmd(args, argLength, fileName, )
        fork() == 0
            dup2(fileDesc, std_out)
            args[2] = NULL
            execvp(args[0], args)
            close(fileDesc)
        fork() != 0
            wait()
```

$\underbrace{\text{ls -l}}_{\text{execvp}}\quad \underset{\text{stop}}{\text{NULL}}\quad \underbrace{\text{bar}}_{\text{output to this file}}$

EX.2
```
cat report.txt | head -2

cmdFlag = checkArgs(args)
has a pipe, cmdFlag = 2

setFileDescriptor(args, fileName, fileDesc, operatorPos)
```

```
operatorPos != 2

cmdFlag == 2
    have 2 str arrays, cmd1 & cmd2,
    to hold parsed pipe commands

    parsePipe1(args, cmd1,     )

    cat  report.txt | head -2
    \_____________/
         cmd 1
    add NULL to end
    cmd1 = cat  report.txt, NULL

    parsePipe2(args, cmd2,     )

    ( cat report.txt | head  -2
                      \_______/
                        cmd 2
     add NULL to end
    cmd2 = head -2 NULL

    execPipedCmd(args, pipeDescs, operatorPos,
                    cmd1, cmd2 )

    pipeDescs[1] = writing part of pipe
    pipeDescs[0] = reading part of pipe
     ~~dup2~~
     2 forks
fk 1 : if    fork() == 0
              dup2(pipeDescs[1], stdout)
              args[operatorPos] = args[2] = NULL
               execvp(args[0], cmd1
      else
          wait()
```

```
EX 2:  if fork() == 0
            dup2(pipes[0], std_in)
            execvp(args[3], cmd2)
       else
            wait()
```

cat report.txt NULL head -2 NULL

execvp stop execvp stop

Ex. 3 sort -nr < grades; ls -l > bar

have str array splitArgs

go through for loop until hit semicolon

splitArgs = sort -nr < grades

add NULL

splitArgs = sort -nr < grades NULL

do same as Ex. 1

keep going through loop

→ reset splitArgs

splitArgs = ls -l > bar

add NULL

splitArgs = ls -l > bar NULL

do same as Ex. 1