Purpose:
This document specifies the design of an implementation of a MINIX shell program. A shell is a program that allows users to run other programs.

Assumptions:

1. The shell will be able to run the internal shell command “exit”.
2. The shell will be able to run internal shell commands with no arguments.
3. The shell will be able to run internal shell commands with arguments.
4. The shell will be able to redirect the output of commands with or without arguments to a file.
5. The shell will be able to redirect the input to commands with or without arguments from a file.
6. The shell will be able to pipe the output of a command with or without arguments as the input to another command.

Rules for running and compiling the shell:
Using the Makefile that was provided, the shell can be compiled by typing in “make” while in the MINIX directory that the source files are in. After compiling, the shell can be run by typing “./myshell” on the same place that “make” was typed in. After doing so, the user may proceed to run MINIX based commands such as ls, exit, echo, clear, pwd, and so on.

Data:

1. getline() – handles the arguments passed to main()
   - It can contain up to 10 elements

2. status – status of a process
   - The status of a process if passed into it

3. pid – the process id
   - It is used to determine what the current process is

4. args – the entire collection of arguments that the user inputs
   - Contains the commands that the user wants to execute

5. command – a minix command that needs to be executed
   - The command at position 0 of args, args[0]

6. lhs – left hand side of args before an operand
   - Part before “<”, “>”, or “|”

7. rhs – right hand side of args after an operand

- Part after “<”, “>”, or “|”

8. red – the index of redirection operand is at in args
    - if a redirect operand exists in args, it takes that value
    - otherwise, it takes the value of 0.

9. lpi – the index of pipe operand is at in args
    - if a pipe operand exists in args, it takes that value
    - otherwise, it takes the value of 0.

10. operand – flag for redirect operands
    - If the value is 1, the “>” operand is being executed
    - If the value is 0, “the “<” operand is being executed

Operations:

1. procResolutions(int r)
    - Description: a function that prints out the error value that is passed into it.
    - Input: Int type
    - Output: a print to the console
    - Result: displays error value to the user for debugging purposes

2. exeCommand(char *command, char **args)
    - Description: a function that takes the command in args[0] and the rest of args then executes execvp.
    - Input: a command, args
    - Output: nothing
    - Result: checks if a command is “cd” or not, then executes procResolution()

3. newProc(char *command, char **args)
    - Description: a function that takes in the commands in args[0] and the rest of args then forks a new process and calls exeCommand().
    - Input: a command, args
    - Output: nothing
    - Result: creates a child process

4. myPipe(char **lhs, char **rhs)
    - Description: a function that takes in the left and right sides of a pipe operand
    - Input: lhs, rhs
    - Output: nothing
    - Result: pipes the arguments passed into args

5. myRedir(char **lhs, char **rhs, int operand)
    - Description: a function that takes in the left and right sides of a redirect operands

- Input: lhs, rhs
- Output: nothing
- Result: pipes the arguments passed into args

6. myExec(char **args)
   - Description: a function that checks if the arguments in args need to be redirected, piped, or simple executed.
   - Input: args
   - Output: nothing
   - Result: executes the commands given by args based on a series of conditions

7. main(void)
   - Description: the main function that prints introduction messages and calls myExec().
   - Input: nothing
   - Output: Int exit status
   - Result: calls myExec() to evaluate the arguments.

Algorithm for main():

```
1. While the user has not exited the shell
2.     Get the user's commands
3.     If the command is "exit"
4.         Exit the shell
5.     If the command is "cd"
6.         Call exeCommand()
7.     Else
8.         Call myExec()
```

Algorithm for exeCommand():

```
1. If the command is "cd"
2.     If the argument of cd is null
3.         Call procResolution() to chdir to the root directory
4.     Else
5.         Call procResolution() to chdir to the argument of cd
6. Else
7.     If the command is not "cd" then just call execvp
```

Algorithm for myExec():

```
1. For all the arguments of args that are not NULL
2.     If the argument at that position i is ">"
3.         red = i
4.         operand = 1
```

```
5.         Else if the argument at that position i is "<"
6.             red = i
7.         Else if argument at that position i is "|"
8.             lpi = i
9.  If there is not a redirect or pipe
10.        Call newProc() to execute the command
11. Else if there is a redirect or a pipe
12.        Call either myRedir() or myPipe() for the respective operand
```

Algorithm for myPipe():

```
1.  Create a storage for the file descriptors called pfd
2.  Fork a new process
3.  If the process is the child
4.         Close the standard output and dup pfd[0]
5.  Else the process is the parent
6.         Close the standard input and dup pfd[1]
```

Algorithm for myRedir():

```
1.  Fork a new process
2.  If the process is the child
3.         If the operand is 1
4.             Create a file descriptor called f
5.             Close the standard output and dup f
6.             Call exeCommand()
7.         Else the operand is 0
8.             Create a file descriptor called f
9.             Close the standard input and dup f
10.            Call exeCommand()
11. Else the process is the parent
12.        Wait for the child process to die
```

Algorithm for newProc():

```
1.  Fork a new process
2.  If the process is the child
3.         Call exeCommand()
4.  Else the process is the parent
5.         Wait for the child process to die
```