7376 HW2

Name: Xuebao Zhao NUID: 002108354

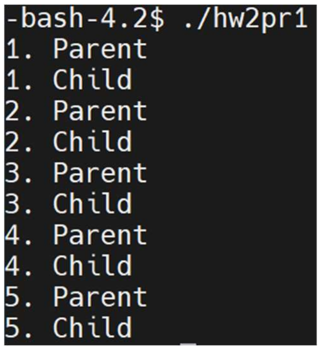
Problem1:

The program creates two pipes “fds1” and “fds2”. Pipe “fds1” serves as a parent-to-child communication channel, and the other as a child-to-parent channel.

The child process close the unused write end in parent-to-child pipe and the unused read end in child-to-parent pipe at beginning. When the amount of print is less than 5, it will repeat the following steps. First check if there is data from parent-to-child communication channel. If the read end receives something, then child will do the print, increase the number of times a print is made, and write data to child-to-parent pipe. If not, it will end.

The parent process closes the read end in parent-to-child pipe and the write end in child-to-parent pipe at beginning. When the amount of print is less than 5, it will repeat the following steps. First it will do the print, increase the number of times a print is made, and write data to parent-to-child pipe. Then check if there is data from child-to-parent communication channel. If the read end receives something, then print again. If not, then it will end.

Results:



```
-bash-4.2$ ./hw2pr1
1. Parent
1. Child
2. Parent
2. Child
3. Parent
3. Child
4. Parent
4. Child
5. Parent
5. Child
```

Problem2:

For function *void ReadArgs(char* in, char** argv, int size):* This function defines a string “temp” to temporarily store the fragment of argument. Every time using function “strtok” to get a fragment and store it in temp. When temp is not empty and the number of arguments that have been separated is less than the maximum number of arguments(“size”), copy it to “argv” and the variable “num” which counts the number of fragments plus one. Continue until the above conditions are not satisfied. Set the last element of the string array to NULL.

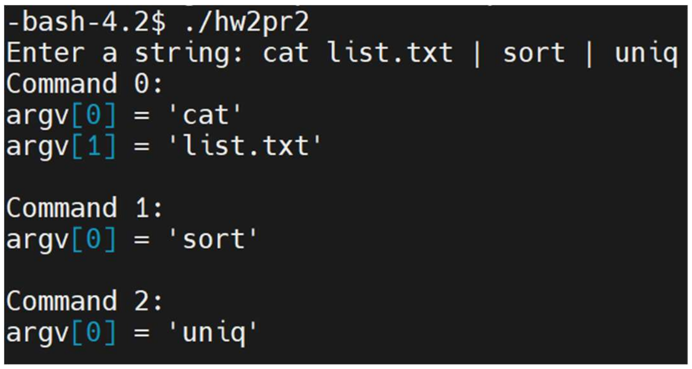
For function *void PrintArgs(char** argv):* This function stops printing arguments as soon as the NULL element is found.

For function *void ReadCommand(char* line, struct Command* command):* The function first splits the line into sub-strings with strtok using the "|" character delimiter if sub-string is not NULL and the number of sub-strings that have been separated is less than the maximum number of sub-command("MAX_SUB_COMMANDS"), just like function *ReadArgs does*. Each sub-string is duplicated and stored into the sub-command's line field. Every time a copy is executed, num_sub_commands is incremented by one. Then all sub-commands's argv fields are populated by calling ReadArgs.

For function *void PrintCommand(struct Command* command):* The function prints all arguments for each sub-command of the command passed by reference by calling PrintArgs.

Results:

Case1: the sub-command and arguments are both in the range.

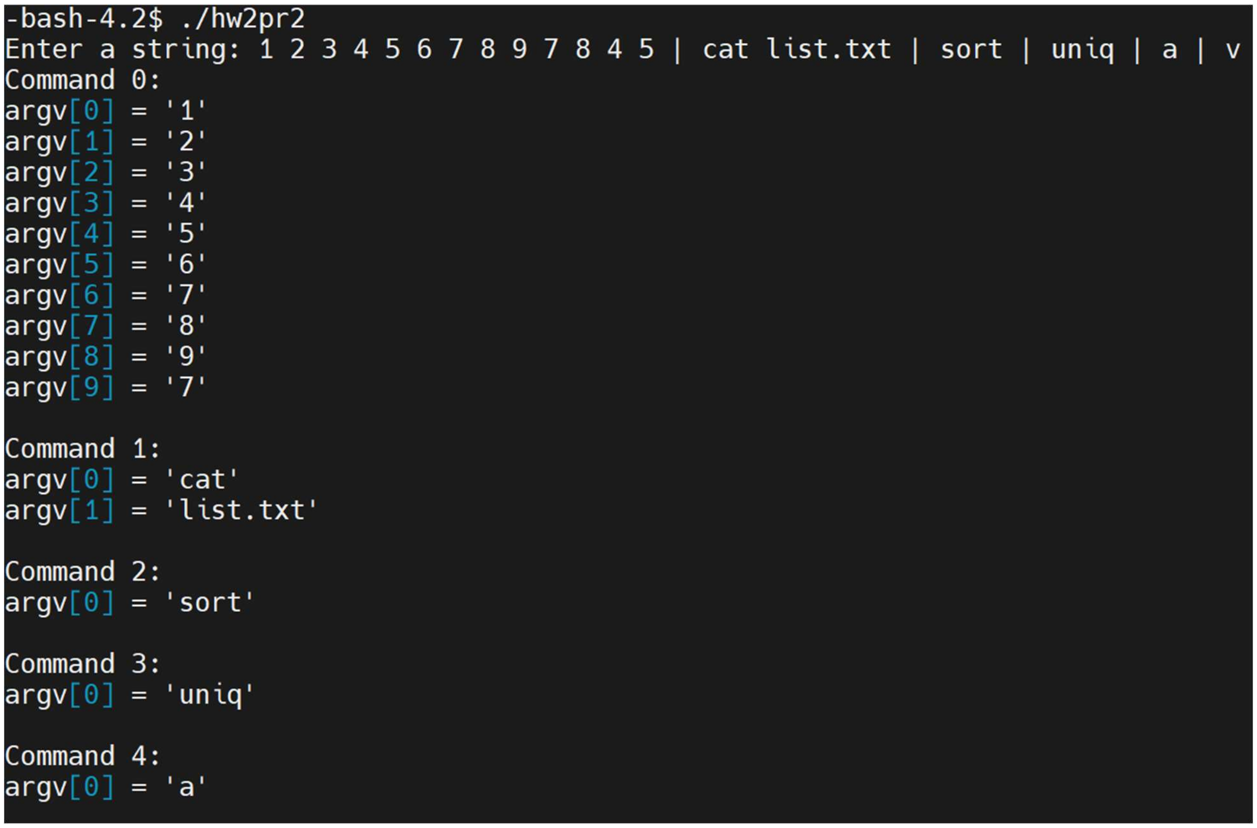


```
-bash-4.2$ ./hw2pr2
Enter a string: cat list.txt | sort | uniq
Command 0:
argv[0] = 'cat'
argv[1] = 'list.txt'

Command 1:
argv[0] = 'sort'

Command 2:
argv[0] = 'uniq'
```

Case1: the sub-command and arguments are both out of range.



```
-bash-4.2$ ./hw2pr2
Enter a string: 1 2 3 4 5 6 7 8 9 7 8 4 5 | cat list.txt | sort | uniq | a | v
Command 0:
argv[0] = '1'
argv[1] = '2'
argv[2] = '3'
argv[3] = '4'
argv[4] = '5'
argv[5] = '6'
argv[6] = '7'
argv[7] = '8'
argv[8] = '9'
argv[9] = '7'

Command 1:
argv[0] = 'cat'
argv[1] = 'list.txt'

Command 2:
argv[0] = 'sort'

Command 3:
argv[0] = 'uniq'

Command 4:
argv[0] = 'a'
```

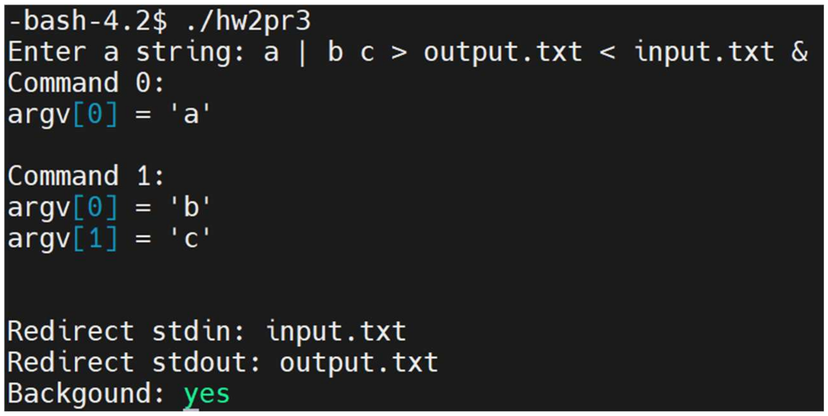
Problem3:

For function *void ReadRedirectsAndBackground(struct Command *command):* The function internally scans the arguments from the last sub-command in reverse order, extracting trailing &, > file, or < file patterns in a loop. If the argument is equal to "&", set the "background" to 1. If the argument is equal to ">" and the next argument is not equal to NULL or "<" or "&", set the "stdout_redirect" to the next argument. If the argument is equal to "<" and the next argument is not equal to NULL or ">" or "&", set the "stdin_redirect" to the next argument.

For function *void PrintCommand(struct Command *command):* The function prints num_sub_commands sub commands, It also prints stdin_redirect, stdout_redirect, background.

Results:

Case1:

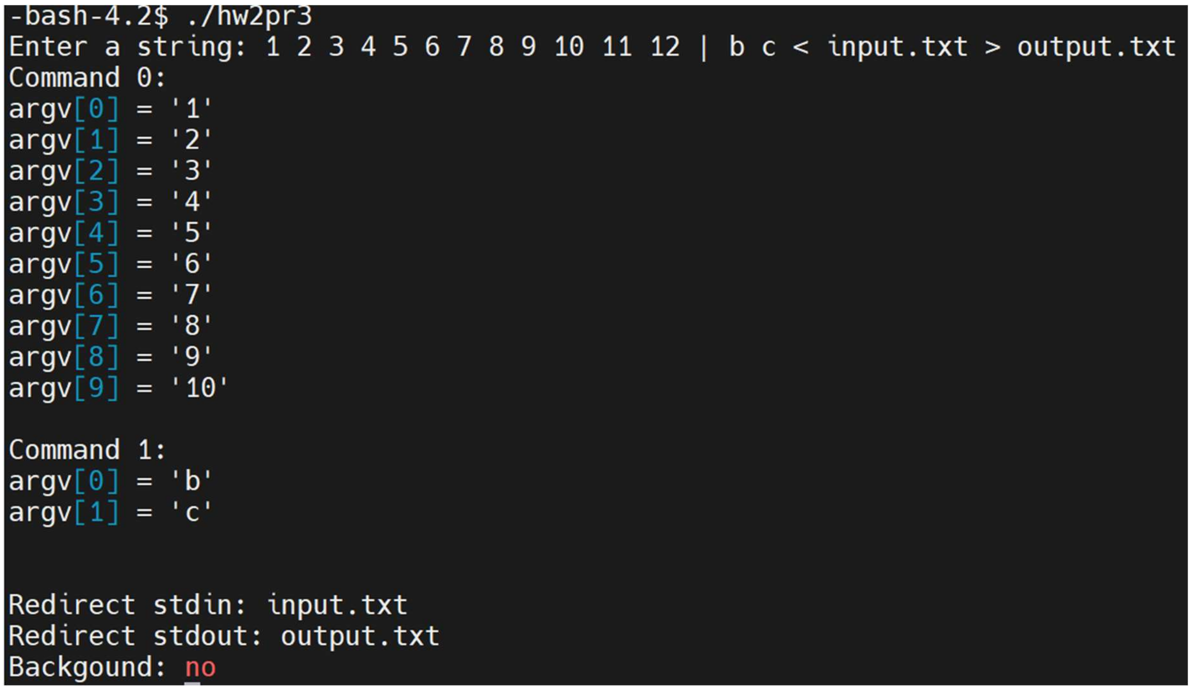


```
-bash-4.2$ ./hw2pr3
Enter a string: a | b c > output.txt < input.txt &
Command 0:
argv[0] = 'a'

Command 1:
argv[0] = 'b'
argv[1] = 'c'


Redirect stdin: input.txt
Redirect stdout: output.txt
Backgound: yes
```

Case2:

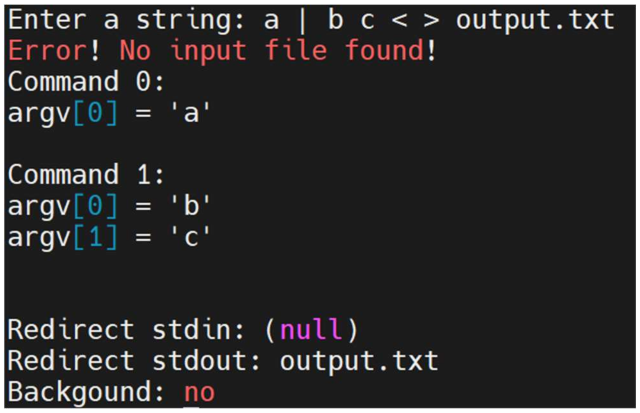


```
-bash-4.2$ ./hw2pr3
Enter a string: 1 2 3 4 5 6 7 8 9 10 11 12 | b c < input.txt > output.txt
Command 0:
argv[0] = '1'
argv[1] = '2'
argv[2] = '3'
argv[3] = '4'
argv[4] = '5'
argv[5] = '6'
argv[6] = '7'
argv[7] = '8'
argv[8] = '9'
argv[9] = '10'

Command 1:
argv[0] = 'b'
argv[1] = 'c'


Redirect stdin: input.txt
Redirect stdout: output.txt
Backgound: no
```

Case3:



```
Enter a string: a | b c < > output.txt
Error! No input file found!
Command 0:
argv[0] = 'a'

Command 1:
argv[0] = 'b'
argv[1] = 'c'


Redirect stdin: (null)
Redirect stdout: output.txt
Backgound: no
```