**LAB [insert lab number here], [insert date here] MCS 253P**

**Name:**

**Partner:**

**General Problem Description**

Write a program to parseCmd shell commands into parts. Handle commands and arguments (words) and the operators <>&|.

**Additional Problem Specifics**

* Remember words may contain paths like /bin/ls.
* Spaces are only required bertween words (which may be the command names and the arguments to the commands while they are optional between operators and words).
* Input will be read from stdin and will have one command per line and the output, sent to stdout, will be each token in the command, one per line.
* Use Bash I/O redirection to test this program on your saved command history like this:

$ history | parseCmd > BashTokens

**Sample Input**

Here is a sample input file

 vi foo.cc>output&

 cat<foo.cc>foo.output&

 history|parseCmd>BashTokens

 cat rolodex.c|tr A-Z a-z>output.foo&

Here is a what the output should be for just the last command above

cat

rolodex.c

|

tr

A-Z

a-z

>

output.foo

&

**Proposed Algorithm**

***Description:***

Scan the command and treat the command case in switch respectively.

***Correctness:***

It is literally right with every char and logically right with all the cases.

***Time Complexity:***

O(n)

***Space Complexity:***

O(n)

**C++ Implementation of Algorithm**

#include <iostream>

#define STRMAX 100

using namespace std;

void lineParse(char\* line){

char single\_ele, pre\_elem;

int i = 0;

while ( (single\_ele = line[i]) != '\0'){

i++;

switch(single\_ele){

case ' ':

case '\t':

case '\n':

if (pre\_elem != '\n' )

cout << '\n';

pre\_elem = '\n';

break;

case '<':

case '>':

case '&':

case '|':

cout << '\n' << single\_ele << '\n';

pre\_elem = '\n';

break;

default:

cout << single\_ele;

pre\_elem = single\_ele;

}

}

}

int main(){

char line[STRMAX];

while (cin.getline(line,STRMAX)){

lineParse(line);

}

}

**Advantages/Disadvantages of Your Algorithm and Any Other Comments**

Advantange: intuitively to think, implement and understand

Disadvantage: sometimes need to be considered with more cases (and the more compact packaging are implemented in HW2 Q1)

**Test Cases**

* sample description of a test case
  + output we expect (want)
  + output our algorithm produces
* cat rolodex.c|tr A-Z a-z>output.foo&
  + output we expect (want)

cat

rolodex.c

|

tr

A-Z

a-z

>

output.foo

&

* + output our algorithm produces

as wish as above

* vi /bin/ls/foo.cc>output&
  + output we expect (want)

vi

/bin/ls/foo.cc

>

output

&

* + output our algorithm produces

as wish as above

* vi foo.cc>output&
  + output we expect (want)

vi

foo.cc

>

output

&

* + output our algorithm produces

as wish as above

* sampleBashCmds.txt:

vi foo.cc>output&

cat<foo.cc>foo.output&

history|parseCmd>BashTokens

cat rolodex.c|tr A-Z a-z>output.foo&

* + output we expect (want)

vi

foo.cc

>

output

&

cat

<

foo.cc

>

foo.output

&

history

|

parseCmd

>

BashTokens

cat

rolodex.c

|

tr

A-Z

a-z

>

output.foo

&

* + output our algorithm produces

as wish as above

**Screenshot of Compilation and Execution of Program Under Valgrind**

**Important Notes (you may erase these notes when pasting your screenshots):**

* **Screenshots should demonstrate your program handling each of your proposed test cases. You may need to create your own custom inputs to demonstrate your test cases.**
* **Your program outputs should be CLEARLY LABELED on the terminal so that you (or any colleague that runs your program) can easily understand each of the following:**
  + **the details of the test case being tested**
  + **what your program’s output is for that test case**
  + **whether your program correctly gave the desired output or not**
    - **one way is to provide the expected output to use as comparison**
    - **or you can use your own method, as long as the person running your program can easily understand from the output whether the program worked as expected or not**

**Since your screenshots capture your program output, this information should also be clearly visible within your screenshots!**

* **To keep things simple, please address all your test cases within a single run of your program.**
  + **one way to do this by using functions to facilitate the testing of the different test cases**
  + **as a final test case, don’t forget to include running your program with the test-case that is provided to you by the instructional staff**
* **Screenshots for each run of your program should start with a screen shot that includes your command to run the program under Valgrind and end with a screenshot that shows the final memory and error report.**
* **For additional notes, please see the Dr. Klefstad’s HW submission guide.**