Name: Yu Qin

General Problem Description

1) function **atoi** (stands for "ascii" to "integer")
to convert a c-string of base-10 digits
to a signed 32 bit decimal number

2) function **itoa**
to convert a signed 32 bit decimal number
into a c-string of digits

main program, **testIntegers**, to read a series of lines as c-strings from standard input (using fgets()) then convert the string (e.g., stored in char s[100];) into a signed integer (e.g., stored in int value;) using your atoi function, then convert the integer back to a string using itoa then print the c-string to the standard output. If the generated string does not equal the input string (using **strcmp**()==0 for equality), print an appropriate error message to stderr.

Additional Problem Specifics

Valid number format in input? Like no char, in decimal…
Valid number range in input? In INT_32bits: -2147483647..2147483647
Int without point decimals?
Only one number in file? Or do in loop?
Empty line?: since make no sense, suppose valid.

Sample Input

12345
-12345
-1
1
0
2111111111
-2111111111

Proposed Algorithm

Description:

atoi:

1) **getline() to get the string, get the size;**
2) **use high = size-1, and low = start = 0 to treat the string;**
3) **judge the first char whether '-'(negative number), flag = -1, low = 1;**
4) **if valid:**
   do i from the last digit of the low to high, i--: use the loop of res2int = res2int*10 + s[i] –'0';

itoa:

1) **get the number = input_int ;**
2) **use low = start = 0 to treat the string;**
3) **judge whether the number is negative for the first '-' , low = 1;**

4) //do from the last integer to put the low position of array, and after done, swap the array between low with high;
int high = low;
while (input_int!=0) {
res2asci10[high] = input_int %10 – 0 + '0';
input_int /=10;
high++;
}
5) void swap(res, low, high){}, with resize in it just by adding '\0';

testIntegers:

Correctness:

atoi:

Right in logic, of transferring the format of array in string to int: consider the negative, and the digits one by one.

itoa:

Right in logic, of transferring a int to string format: consider the negative, and the digits one by one.

Time Complexity:

atoi: O(n)
itoa: O(n)

Space Complexity:

atoi: O(1)
itoa: O(1)

C++ Implementation of Algorithm

```
#include <iostream>
#include <fstream>
#include <stdio.h>
#include "string.h"
using namespace std;
int atoi(char* input_string){
	int high = strlen(input_string)-1, low = 0;
	int flag = 1;
	if (input_string[0] == '-') {
		flag = -1;
		low = 1;
	}
	int res2int = 0;
	for (int i = low; i <= high; i++){
		res2int = res2int*10 + input_string[i] - '0';
	}
	return res2int*flag;
}
void swap(char* res2asc, int low, int high){
	while(low<=high){
		char tmp = res2asc[low];
		res2asc[low] = res2asc[high];
		res2asc[high] = tmp;
		low++; high--;
```

```
	}
}
void itoa(int input_int, char* res2asc){
	if (input_int==0){
		strcpy(res2asc,"0");
		printf("%s\n",res2asc);
		return;
	}
	int low = 0;
	if (input_int < 0 ){
		low = 1;
		res2asc[0] = '-';
		input_int = -input_int;
	}
	int high = low;
	while (input_int!=0) {
		res2asc[high] = input_int%10 + '0';
		input_int/=10;
		high++;
	}
	high--;
	res2asc[high+1] = '\0';
	swap(res2asc,low,high);
	printf("%s\n",res2asc);
}
void MyTestIntegers(char* input_string){
	int atoi_int = atoi(input_string);
	char itoa_string[20];
	itoa(atoi_int, itoa_string);
	if (strcmp(itoa_string, input_string) != 0)
		fprintf(stderr,"Wrong
cast!input:%s,output_int:%d,output_str:%s|\n",input_string,atoi_int,itoa_string);
}
int main(int argc, char** argv){
	//open file
	if (argc < 2){
		cout<<"You choose to input number manually: \n";
		char input_string[20];
		cin >> input_string;
		MyTestIntegers(input_string);
	}
	else if (argc == 2){
		char input_string[20];
		ifstream infile(argv[1]);
		if ( infile.is_open() ){
			while ( infile.getline(input_string,20) ){
				MyTestIntegers(input_string);
			}
			infile.close();
		}
		else cout<<"Cannot open the file.\n";
	}
```

```
        else cout<<"Your input file name should not have space. \n";
}
```

Advantages/Disadvantages of Your Algorithm and Any Other Comments

Advantange: intuitively to think, implement and understand
Disadvantage: some of the whole process is trivial and seems complex

Test Cases

- sample description of a test case
    - output we expect (want)
    - output our algorithm produces

  input expect produce result
- 12345 ---------- 12345----------12345
- -12345---------- -12345---------- -12345
- -1, -1, -1
- 1, 1, 1
- 0,0, 0
- 2111111111,same
- -2111111111,same
- sampleInts.txt:

  12345
  -12345
  -1
  1
  0
  2111111111
  -2111111111

  output:

  itoa retult: 12345
  itoa retult: -12345
  itoa retult: -1
  itoa retult: 1
  itoa retult: 0
  itoa retult: 2111111111
  itoa retult: -2111111111
- sampleInts.txt:

  12345

  -12345

  -1

  1

  0

  2111111111

  -2111111111

output:

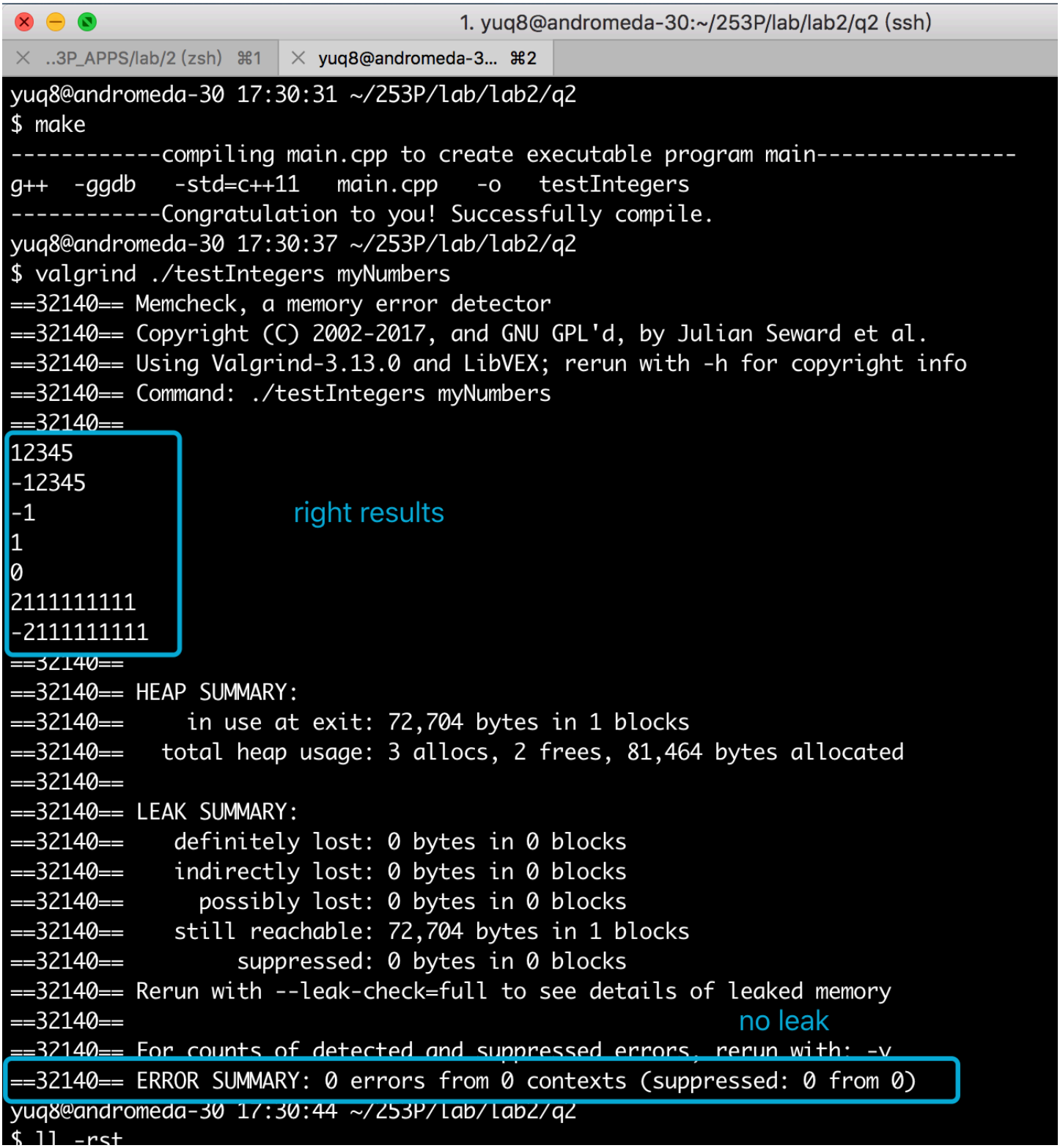
itoa retult: 12345
itoa retult: -12345
itoa retult: -1
itoa retult: 1
itoa retult: 0
itoa retult: 2111111111
itoa retult: -2111111111

Results Screenshots

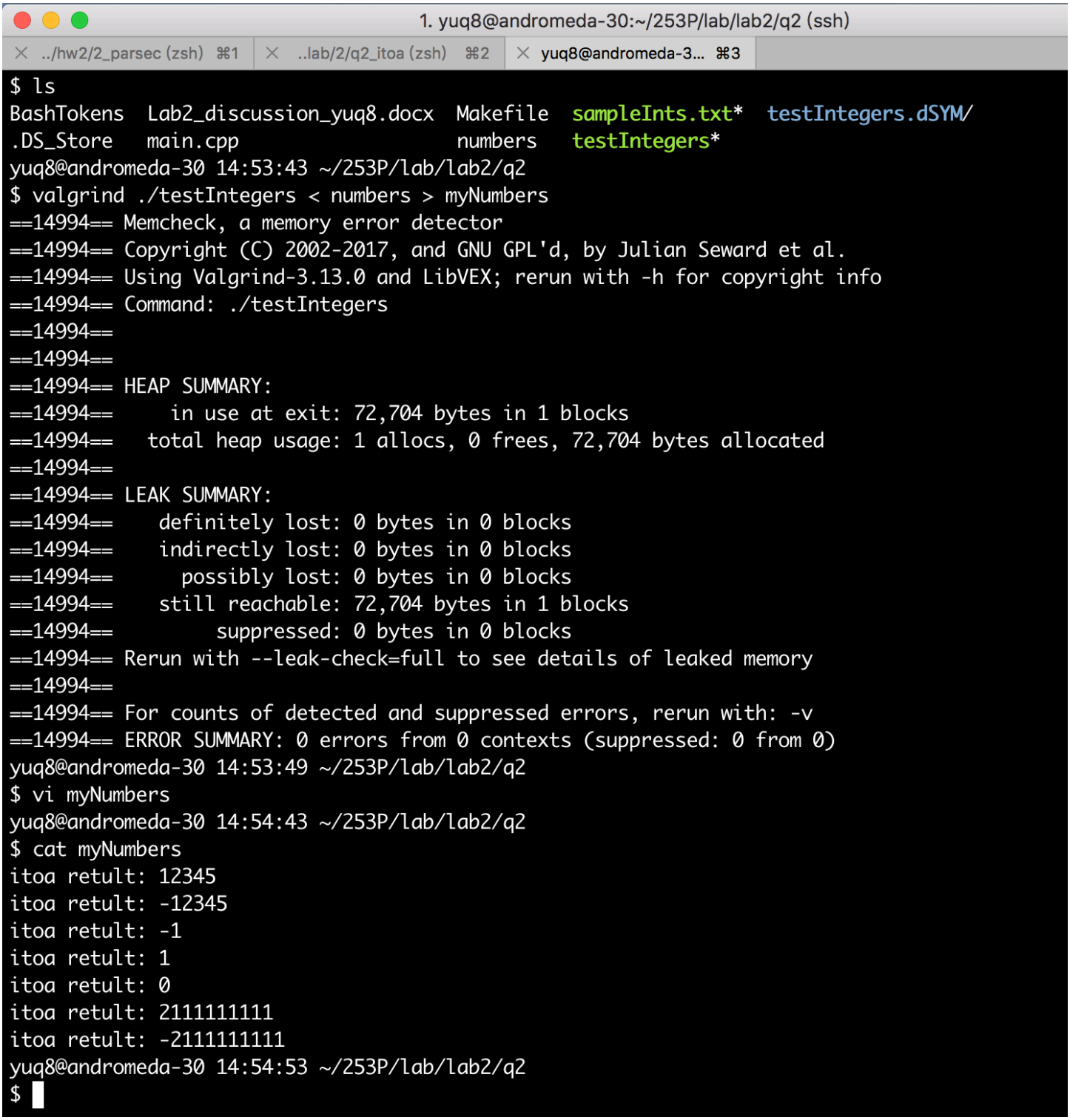
==========start of the write-up==========

Problem 2:

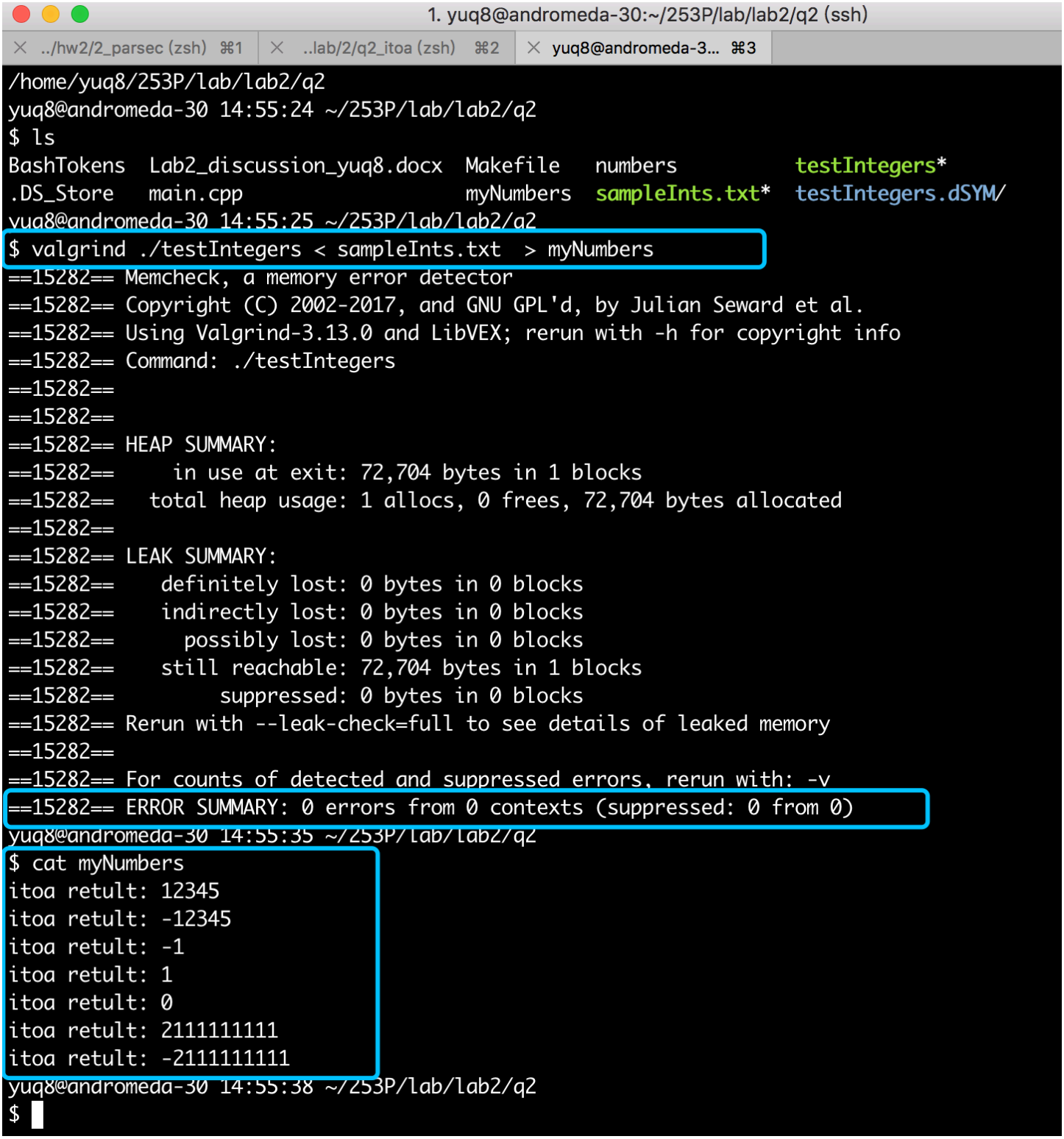
valgrind run, get right results and no memory leak.



change the command API using <, >:


1. yuq8@andromeda-30:~/253P/lab/lab2/q2 (ssh)
../hw2/2_parsec (zsh) ⌘1
..lab/2/q2_itoa (zsh) ⌘2
yuq8@andromeda-3... ⌘3
$ ls
BashTokens Lab2_discussion_yuq8.docx Makefile sampleInts.txt* testIntegers.dSYM/
.DS_Store main.cpp numbers testIntegers*
yuq8@andromeda-30 14:53:43 ~/253P/lab/lab2/q2
$ valgrind ./testIntegers < numbers > myNumbers
==14994== Memcheck, a memory error detector
==14994== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==14994== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==14994== Command: ./testIntegers
==14994==
==14994==
==14994== HEAP SUMMARY:
==14994== in use at exit: 72,704 bytes in 1 blocks
==14994== total heap usage: 1 allocs, 0 frees, 72,704 bytes allocated
==14994==
==14994== LEAK SUMMARY:
==14994== definitely lost: 0 bytes in 0 blocks
==14994== indirectly lost: 0 bytes in 0 blocks
==14994== possibly lost: 0 bytes in 0 blocks
==14994== still reachable: 72,704 bytes in 1 blocks
==14994== suppressed: 0 bytes in 0 blocks
==14994== Rerun with --leak-check=full to see details of leaked memory
==14994==
==14994== For counts of detected and suppressed errors, rerun with: -v
==14994== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
yuq8@andromeda-30 14:53:49 ~/253P/lab/lab2/q2
$ vi myNumbers
yuq8@andromeda-30 14:54:43 ~/253P/lab/lab2/q2
$ cat myNumbers
itoa retult: 12345
itoa retult: -12345
itoa retult: -1
itoa retult: 1
itoa retult: 0
itoa retult: 2111111111
itoa retult: -2111111111
yuq8@andromeda-30 14:54:53 ~/253P/lab/lab2/q2
$

sample:



My Comments

Good question, basic and realistic to be used.
For question 1, the parser, it is more make sense to handle with input from file. But what if we need to parse command from keyboard? It may be further done.
For question 2, it is an easy question in human kind, but the basic process should be detailed processed, so also not less codes to write.

LAB Q1:

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

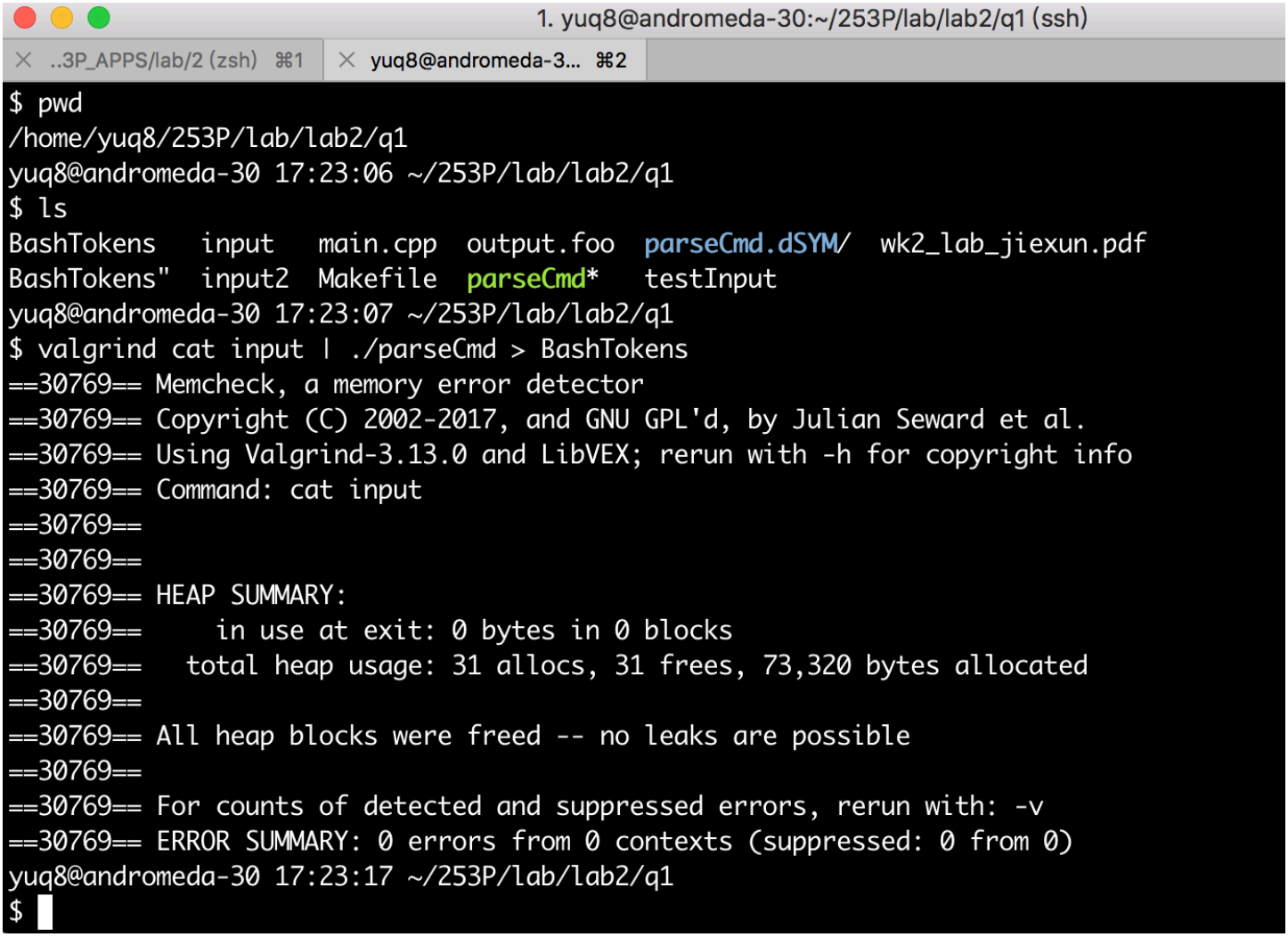
Results Screenshots

==========start of the write-up==========

Problem 1:

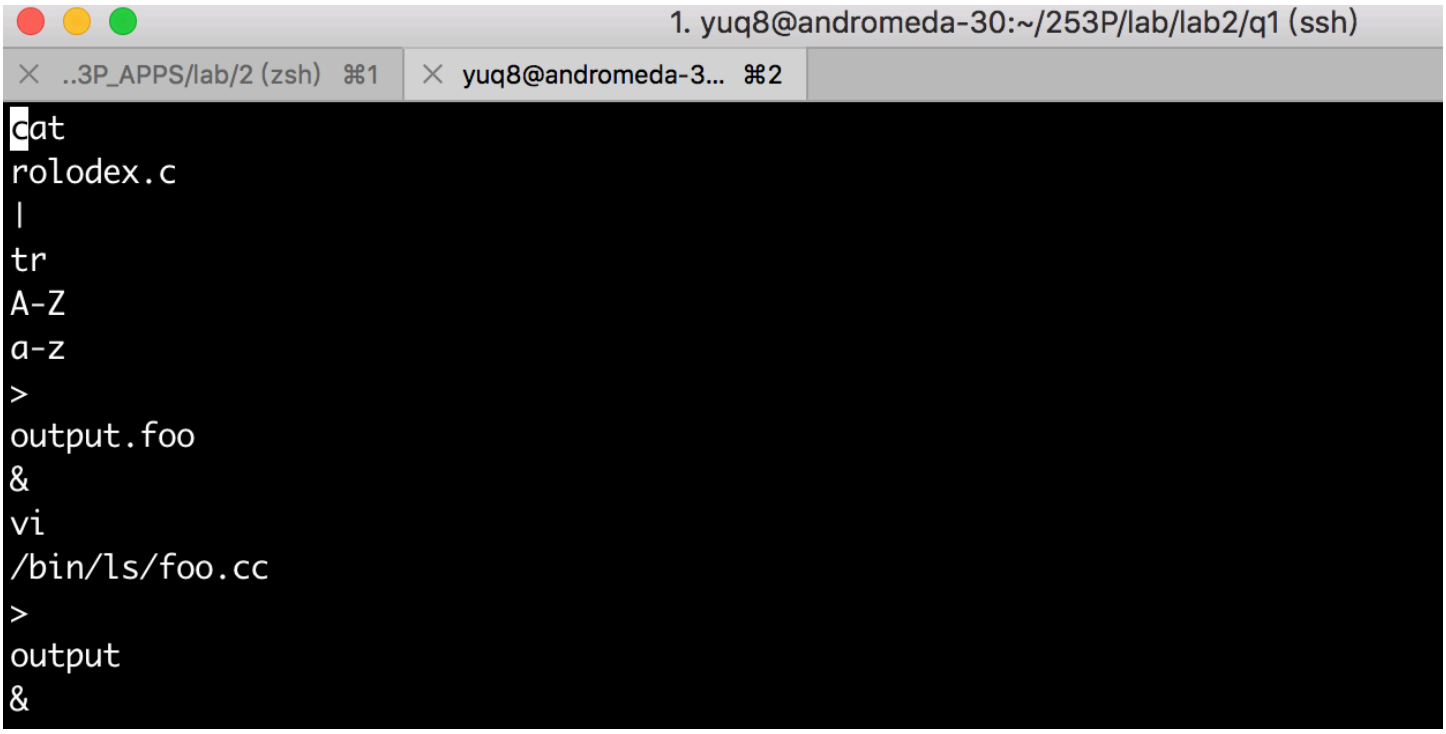
Test case 1:

Step1.1: valgrind run, without memory leak.



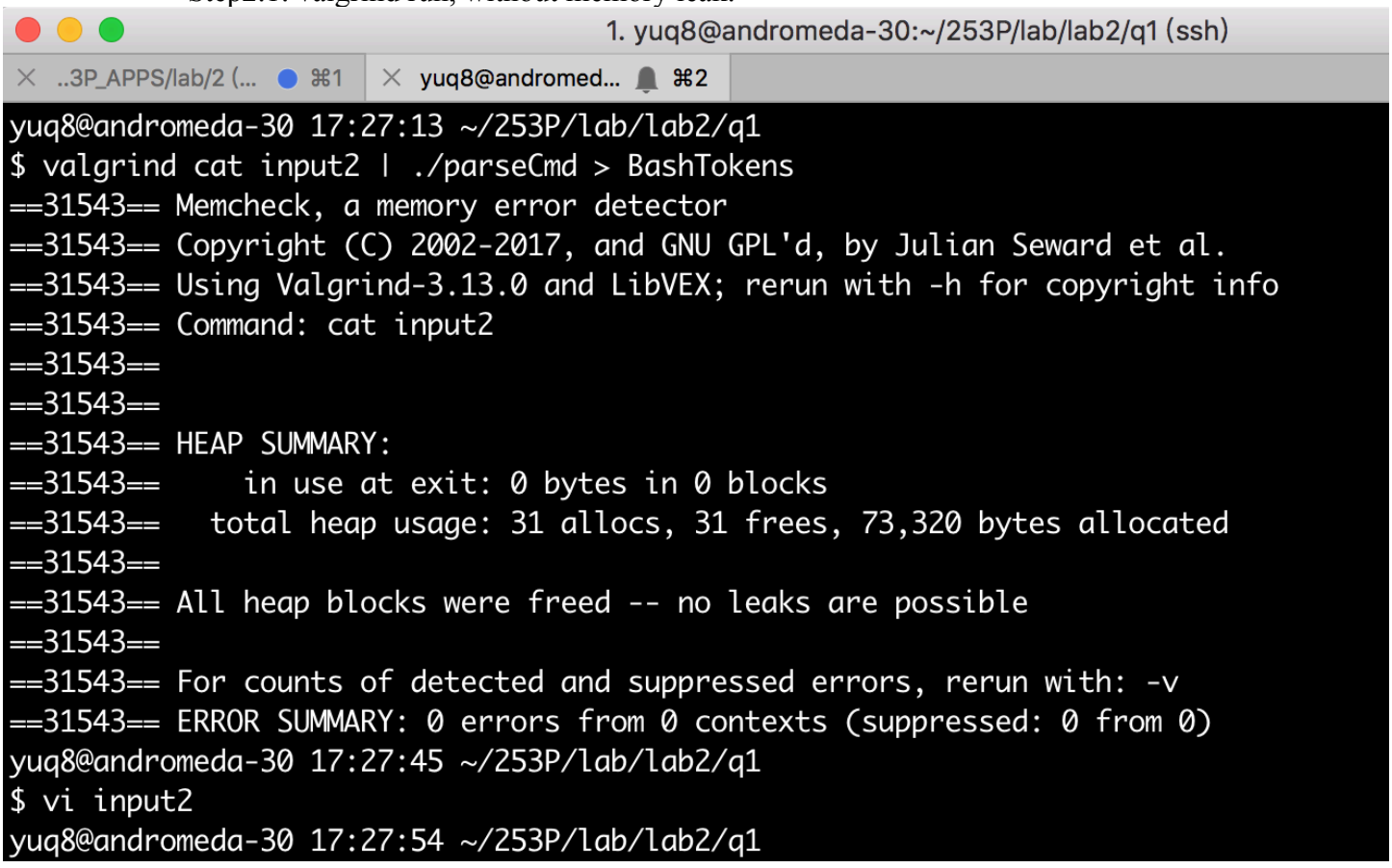
Step1.2: check input:

Step1.3: check output:



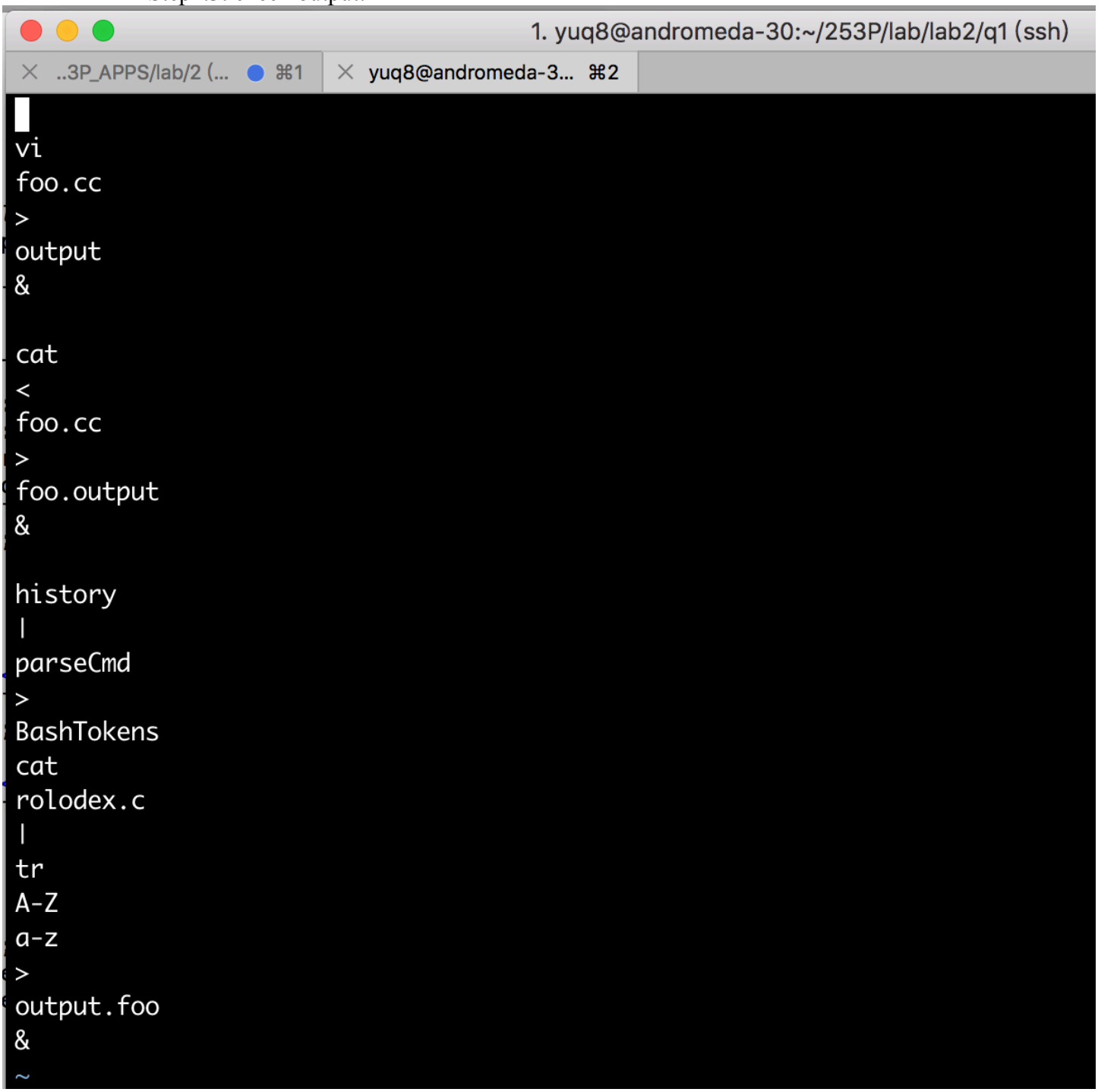
Test case 2:

Step2.1: valgrind run, without memory leak.



Step2.2: check input:

Step2.3: check output:



sample:

```
yuq8@andromeda-30 14:59:18 ~/253P/lab/lab2/q1
$ make
------------compiling main.cpp to create executable program main----------------
g++  -ggdb   -std=c++11   main.cpp   -o   parseCmd
------------Congratulation to you! Successfully compile.
------------input (history | ./parseCmd > BashTokens) to parse all history command on screen
------------input (cat [filename] | ./parseCmd > BashTokens) to parse all history command on screen, eg.:
------------cat input | ./parseCmd > BashTokens
yuq8@andromeda-30 14:59:21 ~/253P/lab/lab2/q1
$ valgrind cat sampleBashCmds.txt | ./parseCmd > BashTokens
==15978== Memcheck, a memory error detector
==15978== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==15978== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==15978== Command: cat sampleBashCmds.txt
==15978==
==15978==
==15978== HEAP SUMMARY:
==15978==     in use at exit: 0 bytes in 0 blocks
==15978==   total heap usage: 31 allocs, 31 frees, 73,320 bytes allocated
==15978==
==15978== All heap blocks were freed -- no leaks are possible
==15978==
==15978== For counts of detected and suppressed errors, rerun with: -v
==15978== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
yuq8@andromeda-30 14:59:34 ~/253P/lab/lab2/q1
$ cat BashTokens
vi
foo.cc
>
output
&

cat
<
foo.cc
>
foo.output
&

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

yuq8@andromeda-30 14:59:39 ~/253P/lab/lab2/q1
$
```

Conclusion: question 1 right.