LAB 2, 1/17/2019 **MCS 253P**

Name: Xiaowei Tan
Partner: Ethel Hoshi

General Problem Description

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

Additional Problem Specifics

Q: Confused about the input, how to read arguments when they are passed by pipe
A: The input passed by pipe will be read into the program by standard input.

Sample Input

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

Proposed Algorithm

Description:

Every time it takes current character and next character into consideration.
Firstly, if the character is a space, it does nothing and for others, it output the character.
Secondly, it decides whether to output an line feed. In the following two situations it will do nothing:

1. Current character is space.
2. Current character is not space and operators. Either for next character.

In other cases, it outputs a line feed.

Correctness:

It is simply a one-pass solution. In each step, it considers the current character and next character.
It is a simple finite state machine, choosing what to next by the two characters.

Time Complexity:

O(n)

Space Complexity:

O(1)

C++ Implementation of Algorithm

```
for(int i = 0; i < line.length(); i++){
        switch(line[i]){
                case ' ':
                        break;
                default:
                        cout << line[i];break;
        }
        linefeed(line, i);
}
```

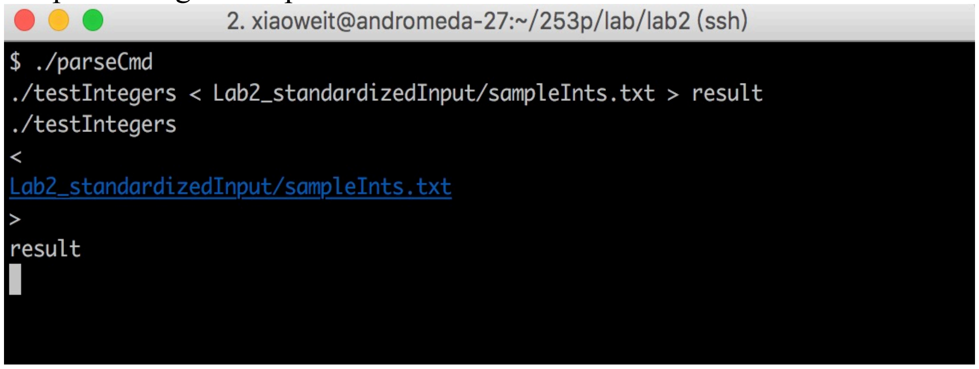
Advantages/Disadvantages of Your Algorithm and Any Other Comments

Advantages:

It’s a one-pass solution so that it is efficient enough. And also the space complexity is O(1), which means it takes constant extra space.

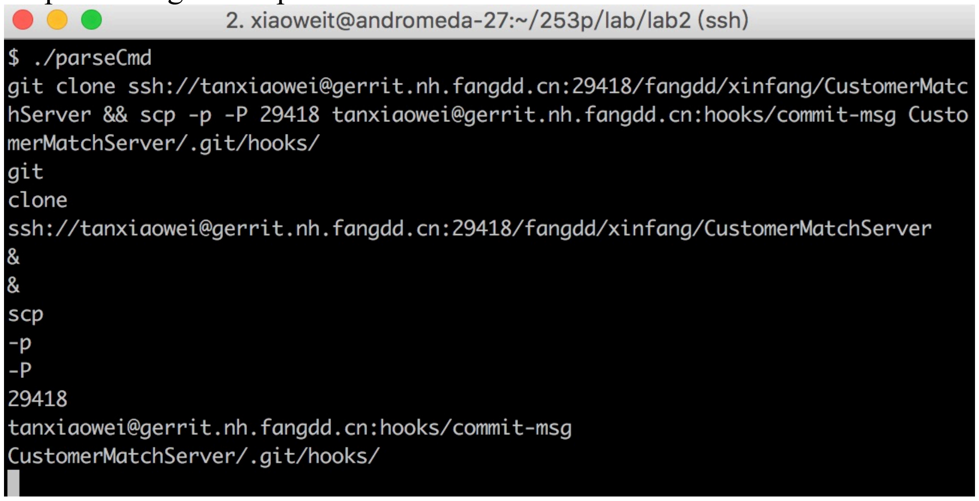
Test Cases

- Testcase1: ./testIntegers < Lab2_standardizedInput/sampleInts.txt > result
    - output we expect (want)
      ./testIntegers
      <
      Lab2_standardizedInput/sampleInts.txt
      >
      result
    - output our algorithm produces



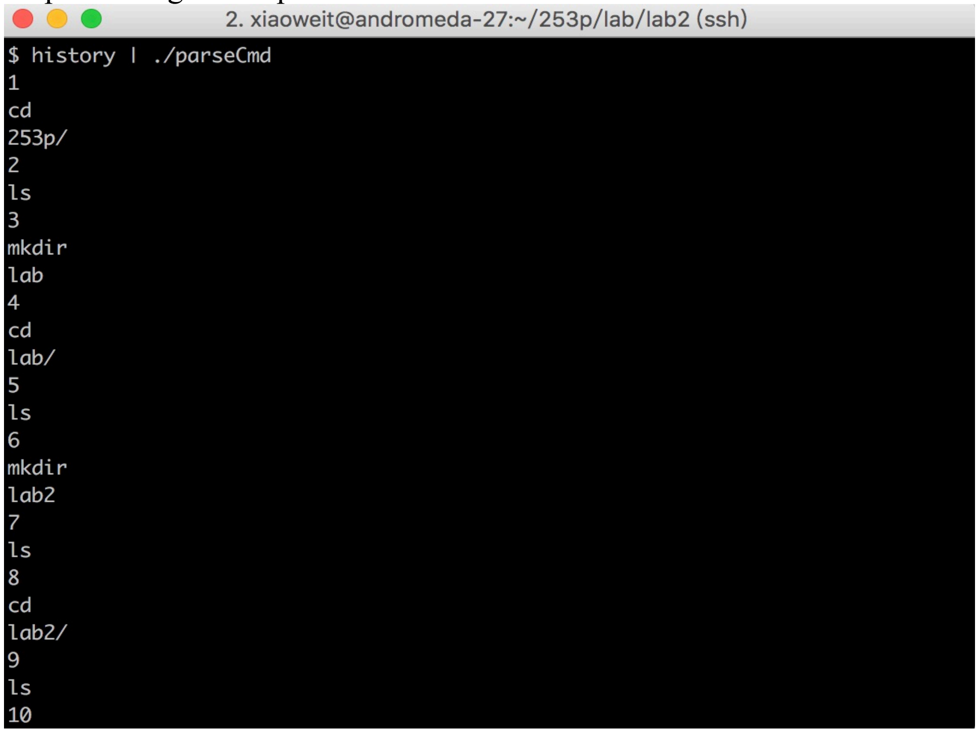
```
2. xiaoweit@andromeda-27:~/253p/lab/lab2 (ssh)
$ ./parseCmd
./testIntegers < Lab2_standardizedInput/sampleInts.txt > result
./testIntegers
<
Lab2_standardizedInput/sampleInts.txt
>
result
```

- Test case 2: git clone ssh://tanxiaowei@gerrit.nh.fangdd.cn:29418/fangdd/xinfang/CustomerMatchServer && scp -p -P 29418 tanxiaowei@gerrit.nh.fangdd.cn:hooks/commit-msg CustomerMatchServer/.git/hooks/
    - output we expect (want)
      git
      clone
      ssh://tanxiaowei@gerrit.nh.fangdd.cn:29418/fangdd/xinfang/CustomerMatchServer
      &
      &
      scp
      -p
      -P
      29418
      tanxiaowei@gerrit.nh.fangdd.cn:hooks/commit-msg
      CustomerMatchServer/.git/hooks/
    - output our algorithm produces



```
2. xiaoweit@andromeda-27:~/253p/lab/lab2 (ssh)
$ ./parseCmd
git clone ssh://tanxiaowei@gerrit.nh.fangdd.cn:29418/fangdd/xinfang/CustomerMatc
hServer && scp -p -P 29418 tanxiaowei@gerrit.nh.fangdd.cn:hooks/commit-msg Custo
merMatchServer/.git/hooks/
git
clone
ssh://tanxiaowei@gerrit.nh.fangdd.cn:29418/fangdd/xinfang/CustomerMatchServer
&
&
scp
-p
-P
29418
tanxiaowei@gerrit.nh.fangdd.cn:hooks/commit-msg
CustomerMatchServer/.git/hooks/
```

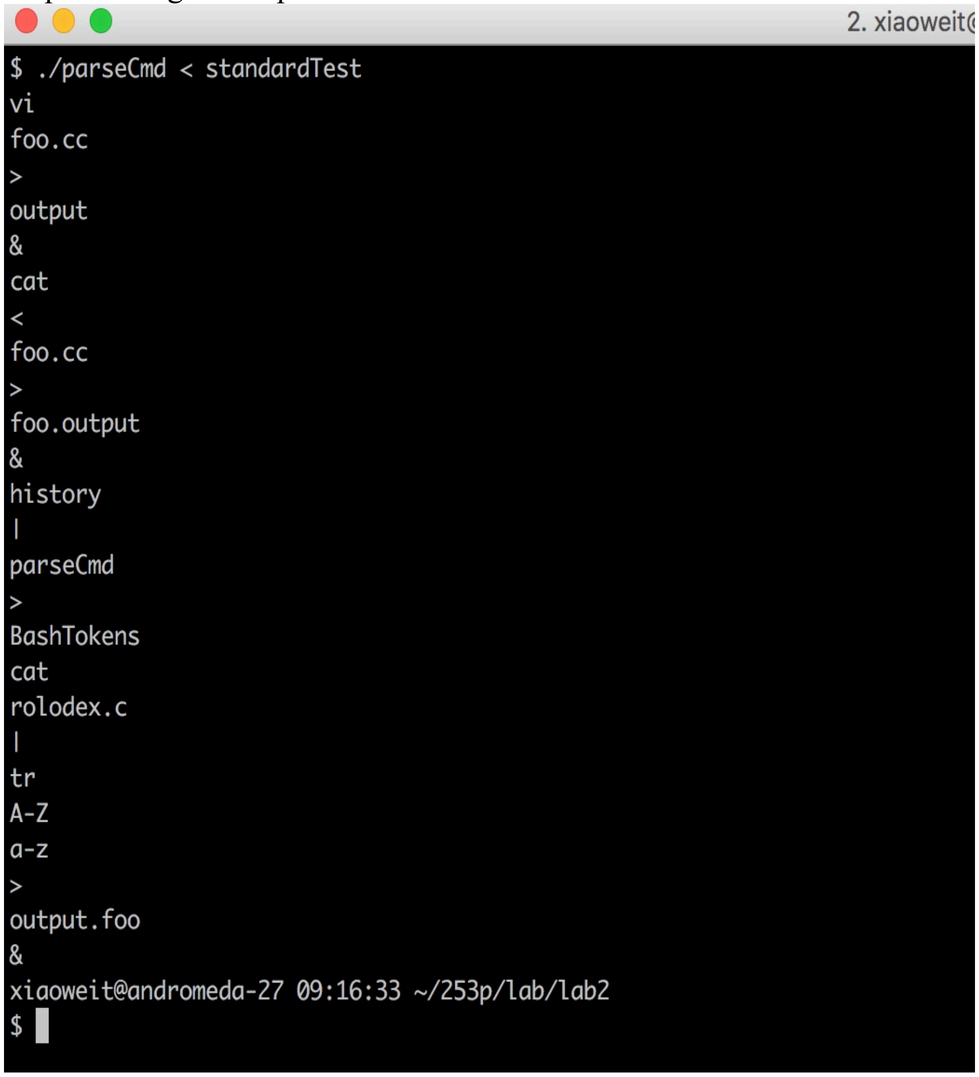
- Test case 3: input from history command
  - output we expect (want)
    1
    cd
    253p/
    2
    ls
    3
    mkdir
    lab
  - output our algorithm produces



2. xiaoweit@andromeda-27:~/253p/lab/lab2 (ssh)

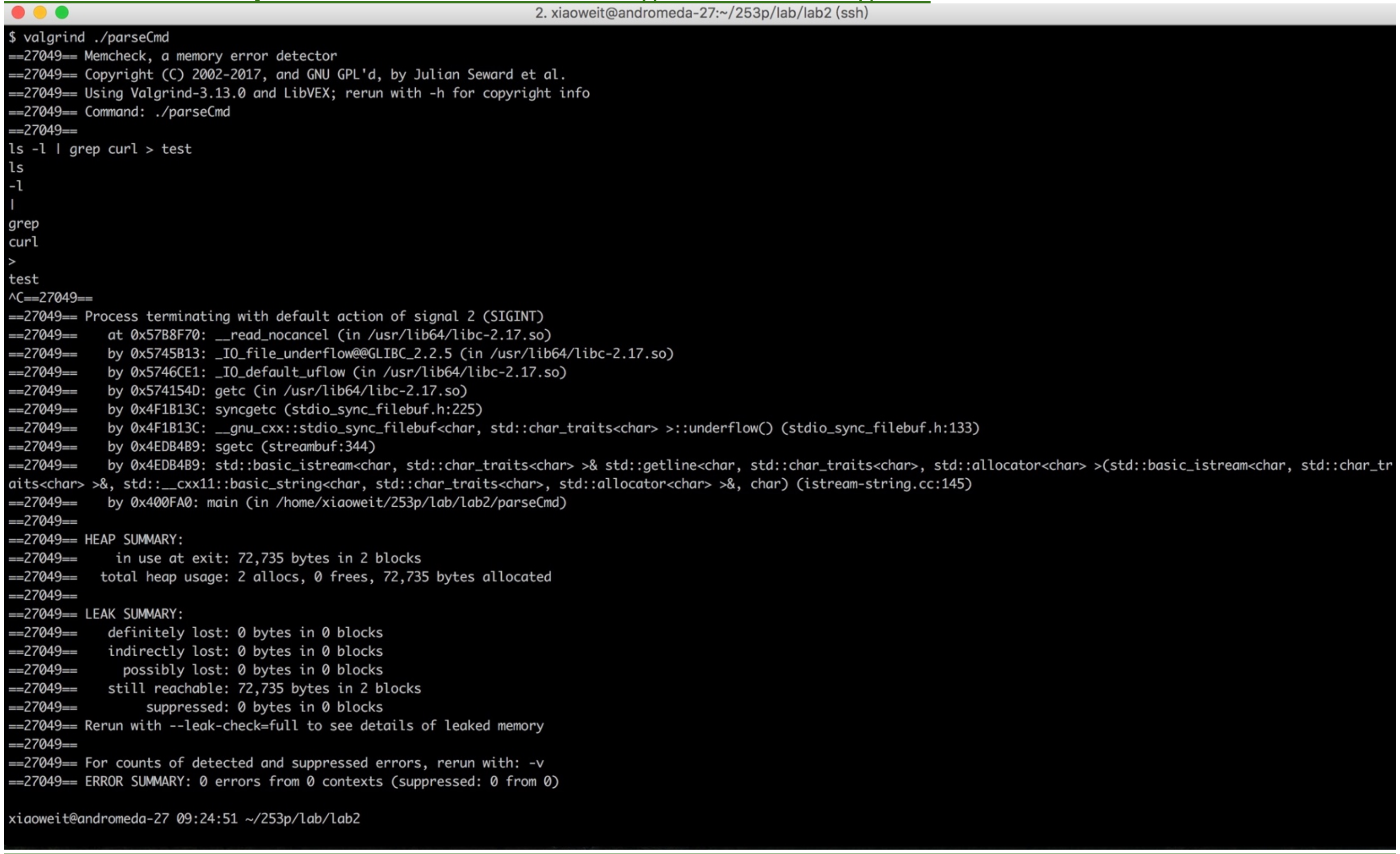
```
$ history | ./parseCmd
1
cd
253p/
2
ls
3
mkdir
lab
4
cd
lab/
5
ls
6
mkdir
lab2
7
ls
8
cd
lab2/
9
ls
10
```

- Test case 4: standardized test given by instructor
    - output we expect (want)
      vi
      foo.cc
      >
      output
      &
      cat
      <
      foo.cc
      >
      foo.output
      &
    - output our algorithm produces



```
$ ./parseCmd < standardTest
vi
foo.cc
>
output
&
cat
<
foo.cc
>
foo.output
&
history
|
parseCmd
>
BashTokens
cat
rolodex.c
|
tr
A-Z
a-z
>
output.foo
&
xiaoweit@andromeda-27 09:16:33 ~/253p/lab/lab2
$ 
```

Screenshot of Compilation and Execution of Program Under Valgrind



2. xiaoweit@andromeda-27:~/253p/lab/lab2 (ssh)

```
$ valgrind ./parseCmd
==27049== Memcheck, a memory error detector
==27049== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==27049== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==27049== Command: ./parseCmd
==27049==
ls -l | grep curl > test
ls
-l
|
grep
curl
>
test
^C==27049==
==27049== Process terminating with default action of signal 2 (SIGINT)
==27049==    at 0x57B8F70: __read_nocancel (in /usr/lib64/libc-2.17.so)
==27049==    by 0x5745B13: _IO_file_underflow@@GLIBC_2.2.5 (in /usr/lib64/libc-2.17.so)
==27049==    by 0x5746CE1: _IO_default_uflow (in /usr/lib64/libc-2.17.so)
==27049==    by 0x574154D: getc (in /usr/lib64/libc-2.17.so)
==27049==    by 0x4F1B13C: syncgetc (stdio_sync_filebuf.h:225)
==27049==    by 0x4F1B13C: __gnu_cxx::stdio_sync_filebuf<char, std::char_traits<char> >::underflow() (stdio_sync_filebuf.h:133)
==27049==    by 0x4EDB4B9: sgetc (streambuf:344)
==27049==    by 0x4EDB4B9: std::basic_istream<char, std::char_traits<char> >& std::getline<char, std::char_traits<char>, std::allocator<char> >(std::basic_istream<char, std::char_tr
aits<char> >&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >&, char) (istream-string.cc:145)
==27049==    by 0x400FA0: main (in /home/xiaoweit/253p/lab/lab2/parseCmd)
==27049==
==27049== HEAP SUMMARY:
==27049==     in use at exit: 72,735 bytes in 2 blocks
==27049==   total heap usage: 2 allocs, 0 frees, 72,735 bytes allocated
==27049==
==27049== LEAK SUMMARY:
==27049==    definitely lost: 0 bytes in 0 blocks
==27049==    indirectly lost: 0 bytes in 0 blocks
==27049==      possibly lost: 0 bytes in 0 blocks
==27049==    still reachable: 72,735 bytes in 2 blocks
==27049==         suppressed: 0 bytes in 0 blocks
==27049== Rerun with --leak-check=full to see details of leaked memory
==27049==
==27049== For counts of detected and suppressed errors, rerun with: -v
==27049== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

xiaoweit@andromeda-27 09:24:51 ~/253p/lab/lab2
```

Name: Xiaowei Tan
Partner: Ethel Hoshi

General Problem Description

Write a function atoi (stands for "ascii" to "integer") to convert a c-string of base-10 digits to a signed 32 bit decimal number and write a function itoa to convert a signed 32 bit decimal number into a c-string of digits.

Additional Problem Specifics

Sample Input

Atoi: -123_frq
Itoa: -123

Proposed Algorithm

Description:

Atoi： Every time it takes a character into consideration.

- If the current character is space and it's at the start of the string, then skip it.
- If the character is '+' or '-' after all the spaces, then it knows the sign of the number.
  If the character is digit, then add it to the result. If the result is beyond the scope of integer(-2147483647..2147483647), then output the min(max) value of integer.
- If the character is not digit, terminate the loop and return number according to the sign.

Itoa: At first, confirm the sign of the number. And if it is negative, convert it to its opposite. Then Every time devide the number by base(2, 8, 10, 16), and store the quotient into the result. And the assign the remainder to the number. Continue the calculation until number equals zero. Then if the sign is minus, put a '-' at the end of the result. Finally, reverse the result string and return it.

Correctness:

Atoi starts with the first non-space character, and ends at the first non-digit character. The algorithm simulates the process.
Itoa starts with the last digit of the number and get the result in an inverted order.

Time Complexity:

Atoi: O(n)
Itoa: O(n)

Space Complexity:

Atoi: O(1)
Itoa: O(n)

C++ Implementation of Algorithm

Atoi:
```
        skipSpace(str, index);
        sign = getSign(str, index);
        num = getNum(str, index, sign);
```
Itoa:
```
       int sign = getSign(i);
```

```
    getStr(str, index, base, i);
    if(sign < 0)
        str[index++] = '-';
    str[index] = '\0';
    reverse(str, index);
```

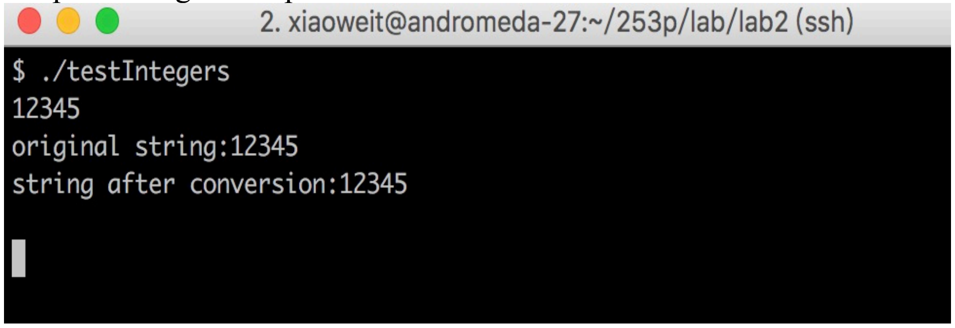
Advantages/Disadvantages of Your Algorithm and Any Other Comments

Advantages:

Atoi is a one-pass solution and itoa is two-pass. They are both efficient enough and can deal with the corner cases like 2147483647 or -2147483647.

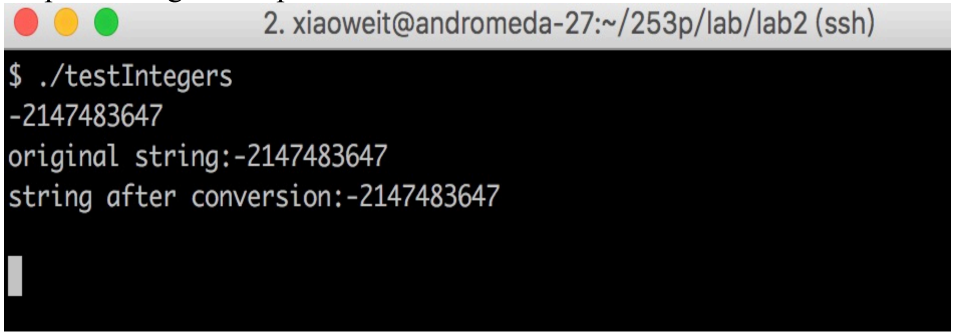
Test Cases

- Testcase1: 12345
  - output we expect (want)
    original string:12345
    string after conversion:12345
  - output our algorithm produces



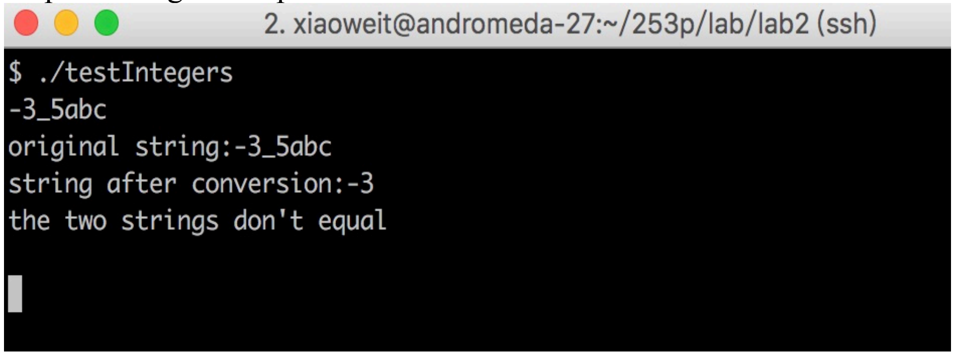
```
2. xiaoweit@andromeda-27:~/253p/lab/lab2 (ssh)
$ ./testIntegers
12345
original string:12345
string after conversion:12345
```

- Test case 2: -2147483647
  - output we expect (want)
    original string:-2147483647
    string after conversion:-2147483647
  - output our algorithm produces



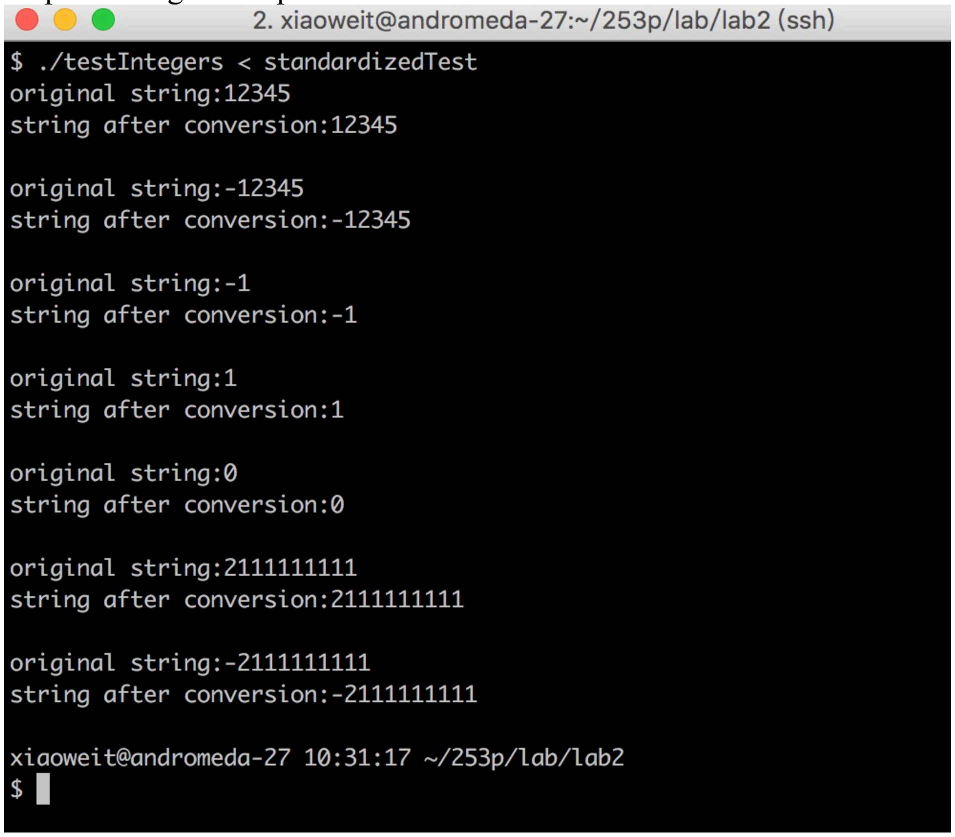
```
2. xiaoweit@andromeda-27:~/253p/lab/lab2 (ssh)
$ ./testIntegers
-2147483647
original string:-2147483647
string after conversion:-2147483647
```

- Test case 3: -3_5abc
  - output we expect (want)
    original string:-3_5abc
    string after conversion:-3
    the two strings don't equal
  - output our algorithm produces



```
2. xiaoweit@andromeda-27:~/253p/lab/lab2 (ssh)
$ ./testIntegers
-3_5abc
original string:-3_5abc
string after conversion:-3
the two strings don't equal
```

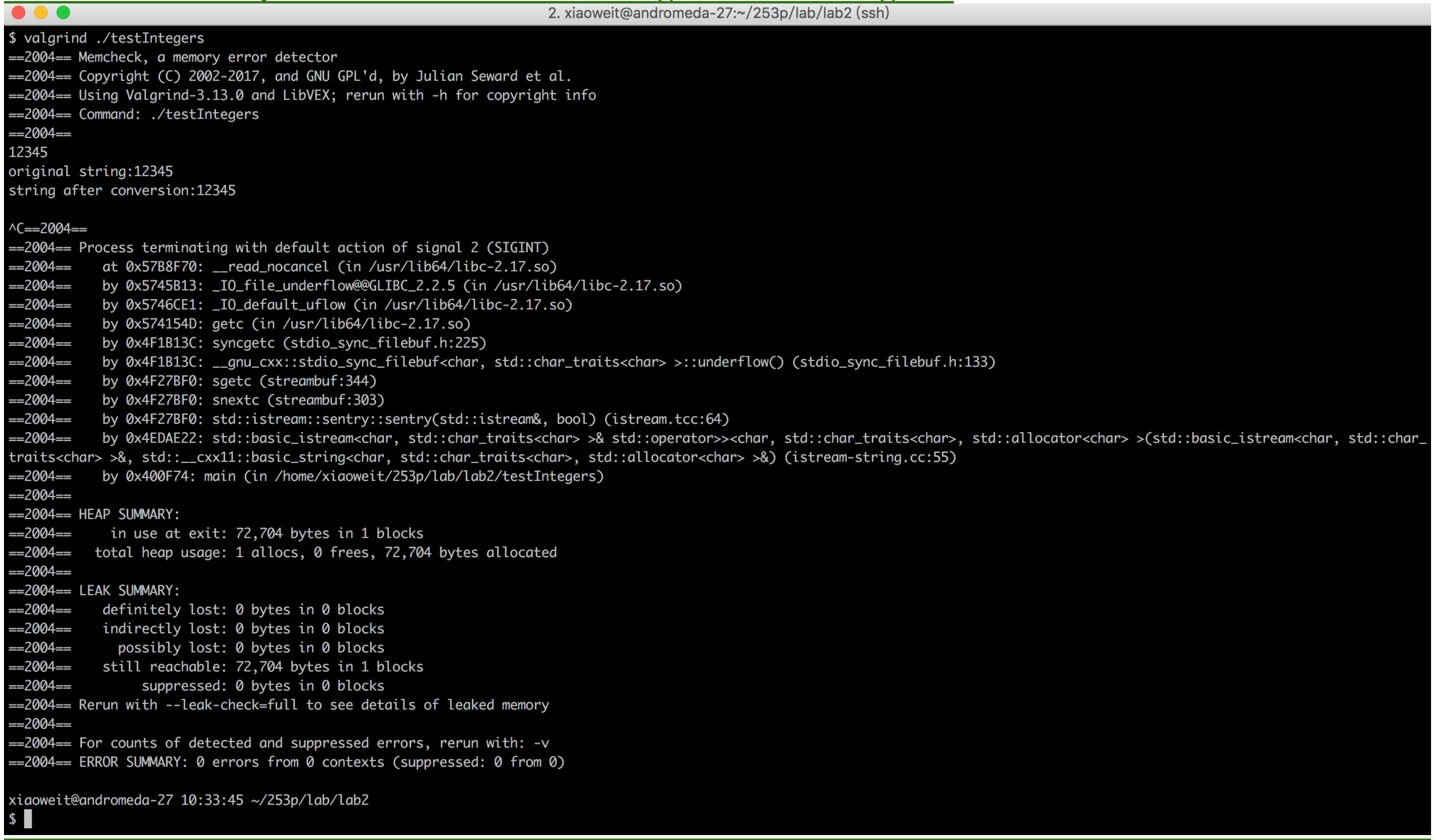
- Test case 4: standardized test given by instructor
    - output we expect (want)
      original string:12345
      string after conversion:12345
      original string:-12345
      string after conversion:-12345
      original string:-1
      string after conversion:-1
      original string:1
      string after conversion:1
      original string:0
      string after conversion:0
      original string:2111111111
      string after conversion:2111111111
      original string:-2111111111
      string after conversion:-2111111111
    - output our algorithm produces



2. xiaoweit@andromeda-27:~/253p/lab/lab2 (ssh)

```
$ ./testIntegers < standardizedTest
original string:12345
string after conversion:12345

original string:-12345
string after conversion:-12345

original string:-1
string after conversion:-1

original string:1
string after conversion:1

original string:0
string after conversion:0

original string:2111111111
string after conversion:2111111111

original string:-2111111111
string after conversion:-2111111111

xiaoweit@andromeda-27 10:31:17 ~/253p/lab/lab2
$ 
```

Screenshot of Compilation and Execution of Program Under Valgrind



2. xiaoweit@andromeda-27:~/253p/lab/lab2 (ssh)

```
$ valgrind ./testIntegers
==2004== Memcheck, a memory error detector
==2004== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2004== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==2004== Command: ./testIntegers
==2004==
12345
original string:12345
string after conversion:12345

^C==2004==
==2004== Process terminating with default action of signal 2 (SIGINT)
==2004==    at 0x57B8F70: __read_nocancel (in /usr/lib64/libc-2.17.so)
==2004==    by 0x5745B13: _IO_file_underflow@@GLIBC_2.2.5 (in /usr/lib64/libc-2.17.so)
==2004==    by 0x5746CE1: _IO_default_uflow (in /usr/lib64/libc-2.17.so)
==2004==    by 0x574154D: getc (in /usr/lib64/libc-2.17.so)
==2004==    by 0x4F1B13C: syncgetc (stdio_sync_filebuf.h:225)
==2004==    by 0x4F1B13C: __gnu_cxx::stdio_sync_filebuf<char, std::char_traits<char> >::underflow() (stdio_sync_filebuf.h:133)
==2004==    by 0x4F27BF0: sgetc (streambuf:344)
==2004==    by 0x4F27BF0: snextc (streambuf:303)
==2004==    by 0x4F27BF0: std::istream::sentry::sentry(std::istream&, bool) (istream.tcc:64)
==2004==    by 0x4EDAE22: std::basic_istream<char, std::char_traits<char> >& std::operator>><char, std::char_traits<char>, std::allocator<char> >(std::basic_istream<char, std::char_
traits<char> >&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >&) (istream-string.cc:55)
==2004==    by 0x400F74: main (in /home/xiaoweit/253p/lab/lab2/testIntegers)
==2004==
==2004== HEAP SUMMARY:
==2004==     in use at exit: 72,704 bytes in 1 blocks
==2004==   total heap usage: 1 allocs, 0 frees, 72,704 bytes allocated
==2004==
==2004== LEAK SUMMARY:
==2004==    definitely lost: 0 bytes in 0 blocks
==2004==    indirectly lost: 0 bytes in 0 blocks
==2004==      possibly lost: 0 bytes in 0 blocks
==2004==    still reachable: 72,704 bytes in 1 blocks
==2004==         suppressed: 0 bytes in 0 blocks
==2004== Rerun with --leak-check=full to see details of leaked memory
==2004==
==2004== For counts of detected and suppressed errors, rerun with: -v
==2004== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

xiaoweit@andromeda-27 10:33:45 ~/253p/lab/lab2
$ 
```