

Laboratory Exercise 1 (C++ Review)

Topics: C++ Review

Compiling and executing C++ programs on a Linux platform

C++ strings

file input

console display output

loops

decision statements

simple arithmetic

cin.fail() condition test

using a debugger

Goals: Upon successful completion of this lab you should be able to:

1. Start a linux session
2. Enter and edit a C++ program
3. Compile and run a C++ program
4. Rename and submit a C++ program to the drop box
5. Complete a linux session
6. Write a C++ input statement
7. Write a C++ output statement
8. Predict when the C++ cin.fail () test will return 1
9. Use the gdb debugger

Related text sections:

Chapter 2 (C++ program format)

emacs reference

linux reference

Laboratory Exercise 1 Instructions

Part 1

1. Boot the computer and select the Linux operating system.
2. Logon as “student”
3. Start a console session.
4. Use ssh to remotely connect to the mainframe:
ssh username@linux.cs.sonoma.edu

Note: steps 1 through 4 can also be done using the Windows operating system and TeraTerm SSH.

5. You may need to change your password:
passwd
6. Create and switch to a CS 215 subdirectory and a Lab1 directory within your CS 215 directory:
mkdir cs215
cd cs215
mkdir Lab1
cd Lab1
7. Using emacs, vi, or another text editor, enter the following C++ program (call the file Lab1a.cpp):

```
#include <iostream>
using namespace std;

int main ()
{
    int v;
    cout << "Enter the data set input: ";
    cin >> v;
    cout << v << (cin.fail() ? " true" : " false") << endl;
    cin >> v;
    cout << v << (cin.fail() ? " true" : " false") << endl;
    cin >> v;
    cout << v << (cin.fail() ? " true" : " false") << endl;
    return 0;
}
```

8. Compile the program using the g++ compiler (call the executable lab1a):
g++ Lab1a.cpp -o lab1a

9. Execute the program several times:
./lab1a

Use each of the data sets in the following table as input. Each time the program asks you to enter the data set, enter all of the characters in the data set the first time the program pauses for input. Record the output.

9. (continued)

Case	Input data	Observed Output
1.	5 12 15	
2.	5 -12 15	
3.	5 a 12	
4.	5.13 -12 15	
5.	abc 5 12	

10. Write 1 to 3 sentences describing your observations of the rules used by the input operator (>>) when it is used to read an integer value.

11. Modify the program by changing the type of v from int to float.

12. Recompile and run the program with each of the data sets in the following table (Enter all of the input in the data set the first time the program pauses for input). Record the output.

Case	Input data	Observed Output
1.	5.1 12.05 15.23	
2.	5.1 -12.05 15.23	
3.	5 12 15.23	
4.	5.13 a 15	
5.	abc 5 12	

13. Write 1 to 3 sentences describing your observations of the rules used by the input operator (>>) when it is used to read a float value.

14. Modify the program by changing the type of v from float to char.

15. Recompile and run the program with each of the data sets in the following table (Enter all of the input in the data set the first time the program pauses for input). Record the output.

Case	Input data	Observed Output
1.	a b c	
2.	a b c	
3.	abc	
4.	5 a 12.5	
5.	abc 5 12	

16. Write 1 to 3 sentences describing your observations of the rules used by the input operator(>>) when it is used to read a character value.

17. Modify the program by changing the type of v from char to string. Note: you will also need to add another include directive: `#include <string>`

18. Recompile and run the program with each of the data sets in the following table (Enter all of the input in the data set the first time the program pauses for input). Record the output.

Case	Input data	Observed Output
1.	abc def ghi	
2.	abc def ghi	
3.	a b c d e f g h	
4.	5.13 -12 15	
5.	abc 5 12	

19. Write 1 to 3 sentences describing your observations of the rules used by the input operator(>>) when it is used to read a C++ string value.

20. Using a text editor, create a document called Lab1a.txt that describes the rules for reading integers, floats, characters, and C++ strings from an input stream. Each of your rules should describe exactly what will be read from the input stream and when the stream will fail. After you have completed your rules, use the following exercises to verify that they are correct.

21. In the boxes below, predict the output that will be generated by this program for each of the sets of keyboard input data. Do not enter or run the program!

```
#include <iostream>
#include <string>

using namespace std;
int main ()
{
    int i;
    float f;
    char c;
    string s;
    bool failflag;
    cout << "Enter line of data: ";
    cin >> i;
    failflag = cin.fail();
    // Set breakpoint 1 here
    while (i != 0)
    {
        cin >> f >> c >> s;
        failflag = cin.fail();
        // Set breakpoint 2 here
        cout << "The output is: ";
        cout << i << ' ' << f << ' ';
        cout << c << ' ' << s << endl;
        cout << "Enter line of data: ";
        cin >> i;
        failflag = cin.fail();
        // Set breakpoint 3 here
        cout << "At end of loop\n";
    }
    return 0;
}
```

Data Set 1 keyboard input

```
5 3.14 a bcd
7.13 4 abcd
0
```



predicted output

Data Set 2 keyboard input

```
5 3.14 a bcd
7.14 4 a bcd
3.14 4 a bcd
0
```



predicted output

22. Enter and run the program with each of the data sets. (Call the program Lab1b.cpp and the executable lab1b.)

Data Set 1 keyboard input

```
5 3.14 a bcd
7.14 4 abcd
0
```



actual program output

Data Set 2 keyboard input

```
5 3.14 a bcd
7.13 4 a bcd
3.14 4 a bcd
0
```



actual program output

23. Recompile your program so that it can run within the debugger:

```
g++ -g Lab1b.cpp -o lab1b
```

24. Using the debugger, look at the values in each variable after each failflag assignment statement. Record the values in the following table.

```
gdb lab1b
```

You will need to use the following gdb commands:

```
list
break <line number>
run
display <variable>
continue
quit
```

Data Set 1

Breakpoint	I	f	c	s	failflag
1					
2					
3					
1					
2					
3					
1					
2					
3					

Data Set 2

Breakpoint	i	f	c	s	failflag
1					
2					
3					
1					
2					
3					
1					
2					
3					
1					
2					
3					

The debugging session for data set 1 is listed at the end of this lab exercise.

24.. After verifying that your rules are correct, submit your Lab1a.txt file as yourlastnameL1.txt. Print a copy of the file to bring to the next lecture.

```
cp Lab1a.txt yourlastnameL1.txt
cp yourlastnameL1.txt ~tiawatts/cs215drop/.
lpr -PSal2006 Lab1a.txt
```

Part 2

1. Write a program called Lab1c.cpp based on the following specifications:
 - A. Your program should open, for input, a text file called “words.txt”. This file will contain several lines of text.
 - B. Your program should count the number of words in the file. A word is defined as a string of characters delimited by white space.
 - C. Your program should count the number of vowels in the file. Only A(a), E(e), I(I), O(o), and U(u) should be counted as vowels by your program.
 - D. Your program should count the number of consonants in the file.
 - E. Your program should count the number of digits in the file.
 - F. Your program should count the number of special characters in the file. Any non-white-space character that is not included in the counts described in statements C, D, or E should be counted as special characters by your program.
 - G. Your program should print out a summary of the counts.

H. Sample Input and Output:

words.txt

```
This file contains 15 words.  
It has lots of letters and  
very few special  
characters!
```

console output

```
Words: 15  
Vowels: 22  
Consonants: 44  
Digits: 2  
Special characters: 2
```

2. Compile your program using the g++ compiler; create an executable call lab1c:
3. Using a text editor, create a file called words.txt for testing your program; you can use the sample data shown above.
4. Run the executable:

```
./lab1c
```
5. If your program does not execute correctly, modify it, recompile it and execute it again.
6. When you are sure that your program is executing correctly, make a copy of the program to put in the drop box and move your copy to the drop box:

```
cp Lab1c.cpp yourlastnameL1.cpp  
mv yourlastnameL1.cpp ~tiawatts/cs215drop/.
```
7. Wait about 2 minutes and then check the web page to verify that you file has reached the drop box:
<http://www.cs.sonoma.edu/~tiawatts/cs215/lab1sub.txt>

Debugging session for data set 1

```
$g++ -g Lab1b.cpp -o lab1b
gdb lab1b
GNU gdb Red Hat Linux (6.3.0.0-1.143.el4rh)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...Using host libthread_db
library "/lib/tls/libthread_db.so.1".
```

```
(gdb) list
1      #include <iostream>
2      #include <string>
3
4      using namespace std;
5      int main ()
6      {
7          int i;
8          float f;
9          char c;
10         string s;
(gdb) list
11         bool failflag;
12         cout << "Enter line of data: ";
13         cin >> i;
14         failflag = cin.fail();
15         // Set breakpoint 1 here
16         while (i != 0)
17         {
18             cin >> f >> c >> s;
19             failflag = cin.fail();
20             // Set breakpoint 2 here
(gdb) list
21             cout << "The output is: ";
22             cout << i << ' ' << f << ' ';
23             cout << c << ' ' << s << endl;
24             cout << "Enter line of data: ";
25             cin >> i;
26             failflag = cin.fail();
27             // Set breakpoint 3 here
28             cout << "At end of loop\n";
29         }
30         return 0;
(gdb) break 15
Breakpoint 1 at 0x8048c28: file Lab1b.cpp, line 15.
(gdb) break 20
Breakpoint 2 at 0x8048c79: file Lab1b.cpp, line 20.
(gdb) break 27
Breakpoint 3 at 0x8048d4c: file Lab1b.cpp, line 27.
(gdb) run
Starting program: /home/tiawatts/cs215f07/lab1b
Enter line of data: 5 3.14 a bcd

Breakpoint 1, main () at Lab1b.cpp:16
16         while (i != 0)
(gdb) display i
1: i = 5
```

```

(gdb) display f
2: f = 3.9911027e-34
(gdb) display c
3: c = 0 '\0'
(gdb) display s
4: s = {static npos = 4294967295,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No
data fields>}, <No data fields>}, _M_p = 0x27040c ""}}
(gdb) display failflag
5: failflag = false
(gdb) c
Continuing.

```

```

Breakpoint 2, main () at Lab1b.cpp:21
21          cout << "The output is: ";
5: failflag = false
4: s = {static npos = 4294967295,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No
data fields>}, <No data fields>}, _M_p = 0x905e04c "bcd"}}
3: c = 97 'a'
2: f = 3.1400001
1: i = 5
(gdb) c
Continuing.
The output is: 5 3.14 a bcd
Enter line of data: 7.13 4 abcd

```

```

Breakpoint 3, main () at Lab1b.cpp:28
28          cout << "At end of loop\n";
5: failflag = false
4: s = {static npos = 4294967295,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No
data fields>}, <No data fields>}, _M_p = 0x905e04c "bcd"}}
3: c = 97 'a'
2: f = 3.1400001
1: i = 7
(gdb) c
Continuing.
At end of loop

```

```

Breakpoint 1, main () at Lab1b.cpp:16
16          while (i != 0)
5: failflag = false
4: s = {static npos = 4294967295,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No
data fields>}, <No data fields>}, _M_p = 0x905e04c "bcd"}}
3: c = 97 'a'
2: f = 3.1400001
1: i = 7
(gdb) c
Continuing.

```

```

Breakpoint 2, main () at Lab1b.cpp:21
21          cout << "The output is: ";
5: failflag = false
4: s = {static npos = 4294967295,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No
data fields>}, <No data fields>}, _M_p = 0x905e064 "abcd"}}
3: c = 52 '4'
2: f = 0.129999995

```

```

1: i = 7
(gdb) c
Continuing.
The output is: 7 0.13 4 abcd
Enter line of data: 0

Breakpoint 3, main () at Lab1b.cpp:28
28             cout << "At end of loop\n";
5: failflag = false
4: s = {static npos = 4294967295,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No
data fields>}, <No data fields>}, _M_p = 0x905e064 "abcd"}}
3: c = 52 '4'
2: f = 0.129999995
1: i = 0
(gdb) c
Continuing.
At end of loop

Breakpoint 1, main () at Lab1b.cpp:16
16             while (i != 0)
5: failflag = false
4: s = {static npos = 4294967295,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No
data fields>}, <No data fields>}, _M_p = 0x905e064 "abcd"}}
3: c = 52 '4'
2: f = 0.129999995
1: i = 0
(gdb) c
Continuing.

Program exited normally.
(gdb) quit
$

```

