Lecture Notes - cin.getline(), plus strings, pointers, new and delete examples

James S. Plank

---

getline1.cpp - Simple use of cin.getline()

cin.getline() has the following prototype:

```
void cin.getline(char *s, const int size);
```

It assumes that **s** is an array of at least **size** characters. It reads in a single line of text from standard input and puts it into **s** as a C-style string, which means that it is terminated by the null character. If you run into the end of the file, or if you enter more than **size-1** characters on a line, it sets the failure flag so that **cin.fail()** returns true. The newline character is not included in the string.

Here's a very simple example program:

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

main()
{
  char s[11];

  while (!cin.fail()) {
    cout << "Enter some text: ";
    cin.getline(s, 11);
    cout << "You entered '"<< s << "'" << endl;
  }
}
```

And here are some examples of it running. Note that when too many characters were entered, it set the **fail()** flag, but did read **size-1** characters into the array. Note also when it encounters the end of file, it reads in an empty string and sets the failure flag:

```
UNIX> g++ getline1.cpp
UNIX> a.out
Enter some text: Hi
You entered 'Hi'
Enter some text: I am Jim
You entered 'I am Jim'
Enter some text: 123456789012345
You entered '1234567890'
UNIX> echo "Jim" | a.out
Enter some text: You entered 'Jim'
Enter some text: You entered ''
UNIX>
```

---

getline2.cpp - A programming example - finding the last line of standard input

Our job now is to read lines from standard input and to print out the last line. Here might be a first pass that you would take at such a program:

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

main()
{
  char s[1000];
  char *lastline;

  while (!cin.fail()) {
    cin.getline(s, 1000);
    if (!cin.fail()) {
      lastline = s;
    }
    cout << "Reading... " << s << endl;
  }
  cout << "The last line was '" << lastline << "'" << endl;
}
```

For this and subsequent examples, I have an input file containing opening lyrics to a classic song. I leave it as a simple exercise for you to figure out which song it is.

```
UNIX> g++ getline2.cpp
UNIX> cat input.txt
Little girl
It's a great big world
But there's only one of me
UNIX> a.out < input.txt
Reading... Little girl
Reading... It's a great big world
Reading... But there's only one of me
Reading...
The last line was ''
UNIX>
```

Note, it didn't work, so we have to figure out why. The reason is that each call to **cin.getline()** overwrites the array **s**. And since **lastline** is a pointer, it simply points to the first element of **s**. That means that each time we set **lastline** = **s**, we're setting **lastline** to point to the same element that gets overwritten at each **cin.getline()** call. This is why it prints out an empty string.

To fix this, we have to make a copy of **s** and store it in a different place. This is done below:

---

getline3.cpp - Using strcpy()

Now instead of setting **lastline** to equal **s**, we have lastline be a separate array and instead copy the contents of **s** to **lastline**:

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

main()
{
  char s[1000];
  char lastline[1000];

  while (!cin.fail()) {
    cin.getline(s, 1000);
    if (!cin.fail()) {
      strcpy(lastline, s);
    }
    cout << "Reading ... " << s << endl;
  }
  cout << "The last line was '" << lastline << "'" << endl;
}
```

This works perfectly:

```
UNIX> g++ getline3.cpp
UNIX> a.out < input.txt
Reading ... Little girl
Reading ... It's a great big world
Reading ... But there's only one of me
Reading ...
The last line was 'But there's only one of me'
UNIX>
```

---

getline4.cpp - Using C++ strings instead

C++ strings are nice in that when you set them to a C style string (or another C++ string), a copy is made automatically. Sometimes that is not what you want, but in this instance it is -- this is a nice, succinct program:

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

main()
{
  char s[1000];
  string lastline;

  while (!cin.fail()) {
    cin.getline(s, 1000);
    if (!cin.fail()) {
      lastline = s;
    }
    cout << "Reading ... " << s << endl;
  }
  cout << "The last line was '" << lastline << "'" << endl;
}
```

Here it is running:

```
UNIX> g++ getline4.cpp
UNIX> a.out < input.txt
Reading ... Little girl
Reading ... It's a great big world
Reading ... But there's only one of me
Reading ...
The last line was 'But there's only one of me'
UNIX>
```

---

getline5.cpp - Using new and delete and C-style strings

This last example shows how to use C style strings and **new/delete**. First, you set **lastline** to NULL. Then when you read a line of text successfully, you first check to see if **lastline** is NULL. If not, you must call **delete** to free up its memory, because you are about the call **new** to create more memory. After that, you do call **new** to create the string. Note, it has to be **strlen(s)+1** because of the null character. Then you do the **strcpy()** and you're done.

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

main()
{
  char s[1000];
  char *lastline = NULL;

  while (!cin.fail()) {
    cin.getline(s, 1000);
    if (!cin.fail()) {
      if (lastline != NULL) delete lastline;
      lastline = new char [strlen(s)+1];
      strcpy(lastline, s);
    }
    cout << "Reading ... " << s << endl;
  }
  cout << "The last line was '" << lastline << "'" << endl;
}
```

Again, this one works fine. Go over this example and make sure that you understand it.

```
UNIX> g++ getline5.cpp
UNIX> a.out < input.txt
Reading ... Little girl
Reading ... It's a great big world
Reading ... But there's only one of me
Reading ...
The last line was 'But there's only one of me'
UNIX>
```

---

getline6.cpp - The last two lines of text - C++ style

Ok, now we're going to print out the last two lines of text. To do this, the easiest thing is to use two C++ strings, **lastline**, and **penultimate**, which will hold the last line and the second to last line read respectively. The program is straightforward. Note, we have to set **penultimate** before we set **lastline**. Think about that. Think about what would happen were those two lines to be reversed.

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

main()
{
  char s[1000];
  string lastline, penultimate;

  while (!cin.fail()) {
    cin.getline(s, 1000);
    if (!cin.fail()) {
      penultimate = lastline;
      lastline = s;
    }
    cout << s << endl;
  }
  cout << "The last two lines were '" << penultimate <<
       "' and '" << lastline << "'" << endl;
}
```

Here it is running. It works well on **input.txt**. However, if you give it one line (the **echo** command below), there is an issue as to what **penultimate** will be, since it is copied from **lastline**, which in the first iteration of the loop is an uninitialized variable. Below, it contains an empty string. However, that may differ depending on your compiler and runtime system. We'll revisit this problem.

```
UNIX> g++ getline6.cpp
UNIX> a.out < input.txt
Little girl
It's a great big world
But there's only one of me

The last two lines were 'It's a great big world' and 'But there's only one of me'
UNIX> echo "Hi" | a.out
Hi

The last two lines were '' and 'Hi'
UNIX>
```

---

getline7.cpp - Using strcpy() instead

Now, we're going to modify the code above so that **lastline** and **penultimate** are arrays, and we use **strcpy()** instead of string assignment as in **getline6.cpp**.

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

main()
```

```
{
  char s[1000];
  char lastline[1000];
  char penultimate[1000];

  while (!cin.fail()) {
    cin.getline(s, 1000);
    if (!cin.fail()) {
      strcpy(penultimate, lastline);
      strcpy(lastline, s);
    }
    cout << s << endl;
  }
  cout << "The last two lines were '" << penultimate <<
        "' and '" << lastline << "'" << endl;
}
```

This works decently well, but we have the same problem as before -- what is in **lastline** when the loop is first executed? The answer is that we don't know -- and that could be the source of some nasty bugs. When we run it with one line, you'll see that there is a single **'@'** in **lastline**. It isn't a horrendous bug, but it's a bug nonetheless. We need to fix it.

```
UNIX> a.out < input.txt
Little girl
It's a great big world
But there's only one of me

The last two lines were 'It's a great big world' and 'But there's only one of me'
UNIX> echo "Hi" | a.out
Hi

The last two lines were '@' and 'Hi'
UNIX>
```

---

getline8.cpp - Fixing that bug. Kind of.

Here's an easy thought -- let's fix the bug by setting **lastline** to be an empty string when the program first runs. That will fix the bug, since now **lastline**'s contents are known, and the first **strcpy()** won't be problematic.

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

main()
{
  char s[1000];
  char lastline[1000];
  char penultimate[1000];

  lastline = "";

  while (!cin.fail()) {
    cin.getline(s, 1000);
    if (!cin.fail()) {
```

```
      strcpy(penultimate, lastline);
      strcpy(lastline, s);
    }
    cout << s << endl;
  }
  cout << "The last two lines were '" << penultimate <<
       "' and '" << lastline << "'" << endl;
}
```

Unfortunately, it won't compile (see below). This is because you can't set an array to be a copy of another array. Instead, you need to call **strcpy()**. We'll see that in our next program.

```
UNIX> g++ getline8.cpp
getline8.cpp: In function 'int main()':
getline8.cpp:10: error: incompatible types in assignment of 'const char [1]' to 'char [1000]'
UNIX>
```

---

getline9.cpp - Fixing the bug correctly

Ok, now we have that initialization of **lastline** done correctly:

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

main()
{
  char s[1000];
  char lastline[1000];
  char penultimate[1000];

  strcpy(lastline, "");

  while (!cin.fail()) {
    cin.getline(s, 1000);
    if (!cin.fail()) {
      strcpy(penultimate, lastline);
      strcpy(lastline, s);
    }
    cout << s << endl;
  }
  cout << "The last two lines were '" << penultimate <<
       "' and '" << lastline << "'" << endl;
}
```

It runs beautifully.

```
UNIX> g++ getline9.cpp
UNIX> a.out < input.txt
Little girl
It's a great big world
But there's only one of me

The last two lines were 'It's a great big world' and 'But there's only one of me'
UNIX> echo "Hi" | a.out
```

```
Hi

The last two lines were '' and 'Hi'
UNIX>
```

---

getlineA.cpp - Recognizing 0 and 1 line inputs

If standard input has zero lines or one line then we really shouldn't print out that there are last and penultimate lines. The program below puts a counter in to count the number of lines, and print the appropriate output. Note that this helps us fix the bug of initializing **lastline**. In this program we don't initialize it, and instead only copy **lastline** to **penultimate** when we know that **lastline** has already been initialized.

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

main()
{
  char s[1000];
  char lastline[1000];
  char penultimate[1000];
  int line;

  line = 0;
  while (!cin.fail()) {
    cin.getline(s, 1000);
    if (!cin.fail()) {
      line++;
      if (line > 1) strcpy(penultimate, lastline);
      strcpy(lastline, s);
    }
    cout << s << endl;
  }
  if (line == 0) exit(0);
  if (line == 1) {
    cout << "The last line was '" << lastline << "'" << endl;
  } else {
    cout << "The last two lines were '" << penultimate <<
        "' and '" << lastline << "'" << endl;
  }
}
```

Now it runs well with all inputs (**/dev/null** is an empty file).

```
UNIX> g++ getlineA.cpp
UNIX> a.out < input.txt
Little girl
It's a great big world
But there's only one of me

The last two lines were 'It's a great big world' and 'But there's only one of me'
UNIX> echo "Hi" | a.out
Hi

The last line was 'Hi'
```

```
UNIX> a.out < /dev/null

UNIX>
```

---

getlineB.cpp - New and Delete

I'll give you two versions that use **new** and **delete**. The first sets both **lastline** and **penultimate** to NULL, and calls **delete** on **penultimate** only when it is non-NULL. It also makes sure that **lastline** is non-NULL when it copies it to penultimate.

Note, we call **new** to create **lastline**, but when we copy it, we simply copy the pointer. Then we only delete **penultimate**. Trace your way through this code to make sure you understand it. If you need to, copy it and put some **cout** statements in it so that you can trace through what's going on when it runs.

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

main()
{
  char s[1000];
  char *lastline;
  char *penultimate;

  lastline = NULL;
  penultimate = NULL;

  while (!cin.fail()) {
    cin.getline(s, 1000);
    if (!cin.fail()) {
      if (lastline != NULL) {
        if (penultimate != NULL) delete penultimate;
        penultimate = lastline;
      }
      lastline = new char [strlen(s)+1];
      strcpy(lastline, s);
    }
    cout << s << endl;
  }
  if (lastline == NULL) exit(0);
  if (penultimate == NULL) {
    cout << "The last line was '" << lastline << "'" << endl;
  } else {
    cout << "The last two lines were '" << penultimate <<
        "' and '" << lastline << "'" << endl;
  }
}
```

It works as we would think:

```
UNIX> g++ getlineB.cpp
UNIX> a.out < input.txt
Little girl
It's a great big world
But there's only one of me
```

```
The last two lines were 'It's a great big world' and 'But there's only one of me'
UNIX> echo "Hi" | a.out
Hi

The last line was 'Hi'
UNIX> a.out < /dev/null

UNIX>
```

---

getlineC.cpp - Example #2 of new and delete

In this example we use the line number to tell us when it is safe to copy or call **delete**, and to tell us how to print things out. Again, trace through this so that you understand it:

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

main()
{
  char s[1000];      // s is a character array and value of s cannot be changed
  char *lastline;    // lastline is a variable pointer to a char
  char *penultimate;
  int line;

  line = 0;

  while (!cin.fail()) {
    cin.getline(s, 1000);
    if (!cin.fail()) {
      line++;
      if (line > 2) delete penultimate;
      if (line > 1) penultimate = lastline;
      lastline = new char [strlen(s)+1];
      strcpy(lastline, s);
    }
    cout << s << endl;
  }
  if (line == 0) exit(0);
  if (line == 1) {
    cout << "The last line was '" << lastline << "'" << endl;
  } else {
    cout << "The last two lines were '" << penultimate <<
        "' and '" << lastline << "'" << endl;
  }
}
```

Here's the output:

```
UNIX> g++ getlineC.cpp
UNIX> a.out < input.txt
Little girl
It's a great big world
But there's only one of me
```

```
The last two lines were 'It's a great big world' and 'But there's only one of me'
UNIX> echo "Hi" | a.out
Hi

The last line was 'Hi'
UNIX> a.out < /dev/null

UNIX>
```