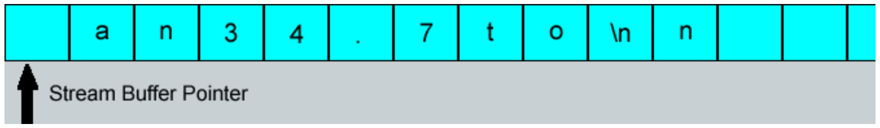
Below is a representation of a *stream* which is a sequence of characters used for input or output

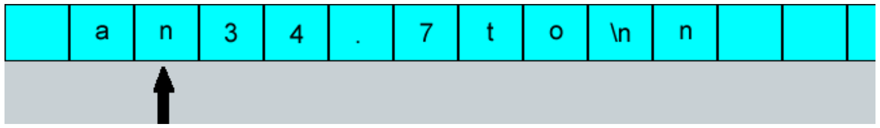


Each stream has a *stream buffer pointer* which advances through the steam character by character.

If the above were the input stream and we wrote the following code

```
char next;
cin >> next;
```

The initial space would be skipped. The pointer would advance to the next *non-whitespace* character, `a`, and read that into `next`, then move to the character after, leaving it like so



If we were to write a loop

```
char next;
cin >> next;
while (!cin.fail()) {
   // do something with next
   cin >> next;
}
```

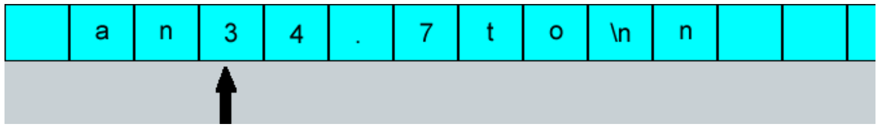
and the pointer were to start at the `n` as above, `next` would be set to `'n', '3', '4', '.', '7', 't', 'o', 'n'` and finally `'1'` in turn. The whitespace (`' '` and `'\n'`) would be skipped over.

The `fail()` method of class `istream` returns `true` *after we try to read in the character* if the operation failed for any reason whatsoever (for example, there are no more characters in the `istream`).

There is no way to look at the `istream` before we read the character to tell if its going to fail.

Extracting Formatted Data from `cin`

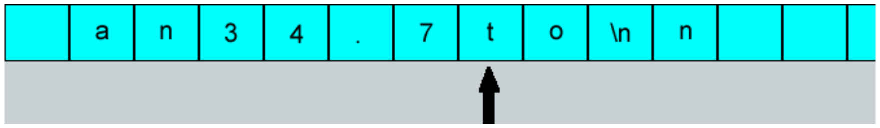
Suppose we had the pointer in this position



and were to write

```
float next;
cin >> next;
```

Since `next` has been declared a `float` here, the input stream extraction operator will try to extract a number. In this case `next` would be set to `34.7` and the pointer would end up like so



That is, the pointer stops as soon as the character under it cannot be part of the number.

Where would it have stopped if `next` had been declared an `int`?

Obviously, to use the extraction operator successfully, *we must know ahead of time what kind of data to expect.*

If we try to extract to an `int` variable with the pointer in the position shown above, the variable will end up undefined.

Reading All Characters

There are times when we would like to read every character (including whitespace). Instead of using the extraction operator >> we can utilize another member function of class `istream`, the `get()` function.

Here's a program to count words in the input stream. It assumes the stream only contains text.

RUN word_count.cpp

```
int main(){
    long wordCount = 0;
    char next;
    bool inWord = false;

    cin.get(next);
    while (!cin.fail()) {
        if (isWhiteSpace(next))
            inWord = false;
        else if (!inWord) { // First letter in word
            inWord = true;
            wordCount++;
        }
        cin.get(next);
    }
    cout << "\n\nThere were " << wordCount << " words." << endl;

    return 0;
}

bool isWhiteSpace(char c){
    return c == ' ' || c == '\n' || c == '\t';
}
```

Files

A file is a named area on a secondary storage device (e.g., disk).

In C++ files are streams, similar to `cout` and `cin` .

Files have to be opened before they can be used, (and should be closed when we're done with them).

file.cpp

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

int main(){
  ifstream inData;
  int x;

  inData.open("mydatafile.dat");
  inData >> x; // read from inData into the variable x
  // use file ...
  inData.close(); // finished with file.
}
```

File Types

`ifstream` —input file stream. Program can read from it using >> , `get` or `getline` .

`ofstream`—output file stream. Program can write to it using << .

Input/Output Redirection

On Unix, Cygwin, or in a DOS window:

`myprog < mydata.txt`—run `myprog` using `mydata.txt` for the standard input (`cin`).

`myprog > myoutput.txt`—run `myprog` using `myoutput.txt` for the standard output (`cout`).

`myprog < mydata.txt > myoutput.txt`—run `myprog` using `mydata.txt` for the standard input, and `myoutput.txt` for the standard output.

Here's a little program that reads the input stream and copies it to the output stream, capitalizing every letter as it goes.

RUN

to_caps.cpp

```
int main(){
    char next;

    cin.get(next);
    while (!cin.fail()) {
        if (next >= 'a' && next <= 'z')
            next += 'A' - 'a';
        cout << next; // Copy next to output
        cin.get(next);
    }

    return 0;
}
```

Programs that simply relay chars from the input to the output stream, possibly changing them as they do so, are known as *filters*.

The program above doesn't appear to run so well in the TM because both the input stream and the output stream appear on the console together. But they actually are separate streams!

Filter programs are really designed to work with files, using i/o redirection. For example, if I take the executable version of the above program, I could use it to capitalize any file by running the following command:

```
     to_caps < someFile.txt > newFile.txt
```

This page last updated on Monday, March 22, 2004