Summary of Reading Data in C++	
Data Delimiters	**Data Sources**

If input is:
`0711 John Q Adams`
>> *reads until next space*
`int bd;`
`char nm;`
`cin >> bd >> nm;`
reads 0711 into `bd`
and reads "John" into `nm`.
 	
`getline()` *reads until endl:*
`int bd;`
`char nm;`
`cin >> bd ;`
`getline(cin, nm);`
reads 0711 into `bd`
and reads "John Q Adams" into `nm`.
	To read data from a file on the disk, do:
	[a] `#include <fstream>`
	[b] Make a stream variable:
	`ifstream s;`
	[c] Connect stream to file:
	`s.open(filename);`
	[d] Check if it worked:
	`if (not s.is_open())`
	`complain_and_exit();`
	[e] Do input using >> and/or getline()
	[f] Close the file:
	`s.close()`

Input: >> vs getline

Consider this list of names:

```
Canada
Cape Verde
Cayman Islands
Central African Republic
Chad
Chile
```

Each line has the name of one country. Some names have one word, some have two words, one has three words. Can you write C++ code to read in and store these names in an array? Would this work:

```
string countries[N];
for (int i = 0; i < N; i++)
    cin >> countries[i];
```

The code compiles and runs, but the array will contain the strings "Canada", "Cape", "Verde", "Cayman", "Islands", ...

The reason for this result is that **>>** reads in one item at a time. With strings **>>** reads until end of the word.

Here, though, we want to read until we get to the *end of the line*. C++ has a function that does exactly that: `getline`. This revised code solves the problem:

```
string countries[N];
for (int i = 0; i < N; i++)
    getline(cin, countries[i]);
```

The `getline` function takes two arguments. The first argument is the source of the data, in this case `cin`. The second argument is a string variable. The function reads and stores string data until end of line.

Mixing >> and getline

You can use **>>** and `getline` together if the input contains both separate items and chunks of text. For example, consider this list of populations and country names:

```
35.5M  Canada
518.5K Cape Verde
55.5K  Cayman Islands
4.7M   Central African Republic
13.2M  Chad
17.8M  Chile
```

M stands for million, and K stands for thousand. How can we read in this data? We need to read in a float, a char, then the rest of the line. Code to do so is:

```
double pop;
char   scale;
string name;
for (int i = 0; i < N; i++) {
    cin >> pop >> scale;
    getline(cin, name);
    if (scale == 'M')
        pop *= 1000000;
    else if (scale == 'K')
        pop *= 1000;
    data[i].pop = pop;
    data[i].nm  = nm;
}
```

We know that there is one double for the population, then one char for the scale factor, then some bunch of text for the name. The **>>** operator is ideal for single, specific items, and the `getline` function is ideal for reading and storing

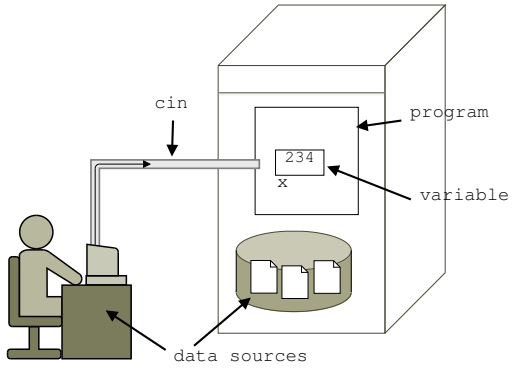
the rest of the line.

Input: `cin` vs Files

How do you read in and store data? You probably have used `cin`, as in:

```
int x;       // create storage
cin >> x;    // read and store
```

Here is a model of this action:

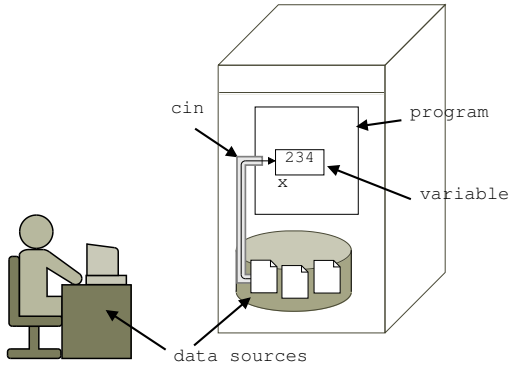


A user at a keyboard presses the keys '2', '3', and '4', and the value flows through a wire into a variable.

Alternate: Input Redirection

What if the user does not want to type the value? The user can tell the program to read from the disk by using the `funnel´ to change the source of data to a disk file.

```
./a.out < filename
```

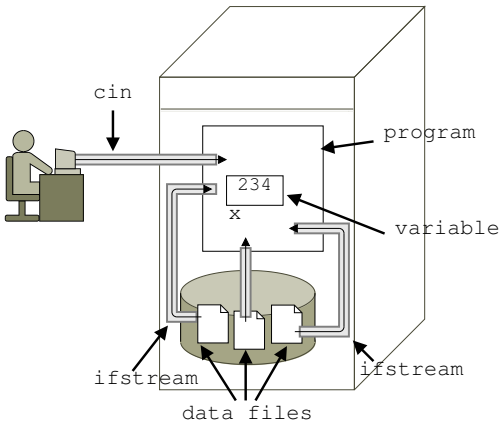


But what if you wanted to read from a disk file directly, not using cin and using a funnel to replace a keyboard with a disk file?

Direct Input from Files

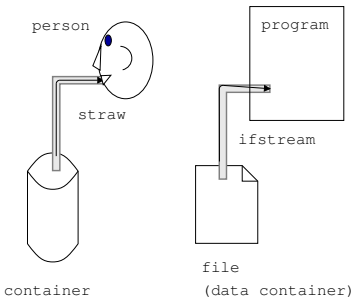
When you are using a word processor and click on `File→Open`, the word processing program connects to the file directly and reads the data from that file, no make-believe keyboard is involved. How does that work?

The following picture shows the ideas behind reading directly from files:



Look at the diagram carefully. We still have a user with a keyboard connected to the program using `cin`. The program can still get data from the keyboard.

But we also see three files on the disk. From each of those files is the computer equivalent of a drinking straw. To pull liquid from a bottle or glass, you put a straw into the bottle and the other end in your mouth. Then you transfer liquid from the bottle into your mouth. Transferring data from a file into a variable works pretty much the same way.



Syntax for File Input

Here is idea and syntax, side by side:

		`#include <fstream>`
[a] Create a data stream (get a straw)		`ifstream s;`
[b] Connect stream to a file (put straw in can)		`s.open(fn)`
[c] Check connection (is the can open?)		`if (not s.is_open())` `  report_and_quit()`
[d] Read data (draw in liquid)	or	`s >> x` `getline(s, nm)`
[e] Close connection (remove straw)		`s.close()`

The code on the right is the sequence of steps to connect, read from, and close a stream to a file. Step [d] is likely to include more processing than the simple example shown.

Special Details

Here are a few special details you need to know about reading from files.

Argument to open() The argument to the open function (in step [b]) is the name of the file to open. This value *must* be a value of type *C-string*. A *C-string* is not the typical C++ string. It is an array of chars. The following examples all work:

```
s.open("fishdata.txt");

string name = "fishdata.txt";
s.open(name);  // use -std=c++11 to compile

char name[] = "fishdata.txt";
s.open(name);
```

We have never written code like the third example, so that probably does not look familiar.

The second example is the most useful for C++ programmers. You often work with regular C++ strings, so you will need to pass a C-string version to open.

Passing ifstreams to functions An ifstream is an object that has some internal data members. When you read data from the stream, those internal data members change to keep track of how far through the file you have read.

Therefore, it is best to pass ifstream object by reference to functions that use the stream. That way, the state of the stream is available to the caller. We could pass pointers to these streams, but people don't do that. Instead, they use a special notation to pass the stream by reference.

You will see functions declared as:

```
int read_from_stream(ifstream& f);
```

The ampersand says that the stream should be passed by reference. You do not have to use the ampersand when you call the function or when you refer to the string.

Sample Program

Here is a sample program that compares two files line by line and reports the first line where they differ.

```
#include <iostream>
#include <fstream>
using namespace std;
//
// line-compare
// compare two files.  Report first diff
//
bool open_file(ifstream&);
int  compare_files(ifstream&, ifstream&);

int main()
{
    ifstream in1, in2;
    int result;

    if (open_file(in1) and open_file(in2))
    {
        result = compare_files(in1, in2);
        in1.close();
        in2.close();
    }
    return result;
}
//
// open_file -- opens file
//   rets: true if worked, false if not
//
bool open_file(ifstream& f)
{
    string name;

    cout << "name: ";
    cin  >> name;
    f.open(name);             // try to open
    return f.is_open();       // report result
}

//
// compare files: read line by line
//    rets: 0 if no diffs, 1 if diffs
//
int compare_files(ifstream& f1, ifstream& f2)
{
    string line1, line2;

    getline(f1, line1);   // read 1st lines
    getline(f2, line2);   // into strings

    while(not f1.eof() and not f2.eof())
    {
        // -- compare strings
        if (line1 != line2) {
            cout << "differ: " << endl;
            cout << line1 << endl
                 << line2 << endl;
            return 1;
        }

        // -- ok so far, read next line
        getline(f1, line1);
        getline(f2, line2);

        // -- if one runs out first, report
        if (f1.eof()and not f2.eof()) {
            cout << "file 1 is shorter"
                 << endl;
            return 1;
        }
        if (not f1.eof() and f2.eof()) {
            cout << "file 2 is shorter"
                 << endl;
            return 1;
        }
    }
    return 0;
}
```