

```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
```

```
#define MAXPAROLA 30
#define MAXRIGA 80
```

```
int main(int argc, char *argv[])
{
    int freq[MAXPAROLA]; /* vettore di contatori
delle frequenze delle lunghezze delle parole */
    char riga[MAXRIGA];
    int i, inizio, lunghezza;
    FILE *f;
```

```
for(i=0; i<MAXPAROLA; i++)
    freq[i]=0;
```

```
if(argc != 2)
```

```
{
    fprintf(stderr, "ERRORE, serve un parametro con il nome del file\n");
    exit(1);
}
```

```
f = fopen(argv[1], "r");
if(f==NULL)
```

```
{
    fprintf(stderr, "ERRORE, impossibile aprire il file %s\n", argv[1]);
    exit(1);
}
```

```
while( fgets( riga, MAXRIGA, f ) != NULL )
```



High Level Programming

The IO Library

Stefano Quer

Dipartimento di Automatica e Informatica

Politecnico di Torino

License Information

This work is licensed under the license



Attribution-NonCommercial-NoDerivatives 4.0 International

This license requires that reusers give credit to the creator. It allows reusers to copy and distribute the material in any medium or format in unadapted form and for noncommercial purposes only.

① **BY:** Credit must be given to you, the creator.

② **NC:** Only noncommercial use of your work is permitted.

Noncommercial means not primarily intended for or directed towards commercial advantage or monetary compensation.

③ **ND:** No derivatives or adaptations of your work are permitted.

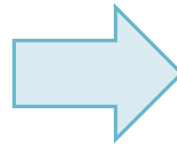
To view a copy of the license, visit:

<https://creativecommons.org/licenses/by-nc-nd/4.0/?ref=chooser-v1>

Premises

❖ Where are we?

- u01-courseIntroduction
- u02-review
- u03-cppBasics
- u04-cppLibrary
- u05-multithreading
- u06-synchronization
- u07-advancedIO
- u08-IPC

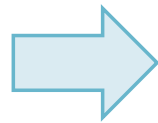


- u04s01-IOLibrary.pdf
- u04s02-sequentialContainers.pdf
- u04s03-genericAlgorithms.pdf
- u04s04-associativeContainers.pdf
- u04s05-dynamicMemory.pdf
- u04s06-copyControl.pdf
- u04s07-templates.pdf
- u04s08-exercise.pdf

Condition states

- ❖ When we perform IO, an error can occur
 - Some errors are recoverable, others are not
 - Once an error has occurred, subsequent IO operations will fail
 - The code should always check for errors in IO streams
 - The easiest way to check a state of a stream object is to use a condition

```
cin >> s;
```



```
if (cin >> s) { ... Success ... }
```

```
while (cin >> s) { ... Success ... }
```

Condition states

- ❖ The IO classes also define functions and flags that we can check to understand the status of the stream

➤ If **s** is a stream

Attention:
while (!s.eof()) { ... }
is buggy !!!

Type	Meaning
s.eof()	True if the stream hits EOF
s.fail()	True if the IO operation failed
s.bad()	True if the stream is corrupted
s.good()	True if the stream is in a valid state
s.clear()	Reset all condition values to a valid state. Return void.
etc.	

IO Library

- ❖ The header **fstream** defines the types used to read and write named files
 - These types provide the same operations as those we have used on the objects **cin** and **cout**
 - On them, we can use the standard IO operators **<<**, **>>**, and **getline**

Type	Meaning
ifstream	To read from a given file
ofstream	To write to given file
fstream	To read or write a given file

IO Library

- ❖ We can also use specific members to manage the file associated with the stream

Type	Meaning
<code>fstream fs;</code>	Creates an unbound file stream.
<code>fstream fs(f);</code>	Creates an <code>fstream</code> <code>fs</code> and open file <code>f</code> . The object <code>f</code> must be a string or a C-like pointer to a string.
<code>fstream fs(f,mode);</code>	Like the previous one but open the file in the given mode.
<code>fs.open(f)</code>	Open the file <code>f</code> and bound it to <code>fs</code> .
<code>fs.open(f,mode)</code>	Like the previous one but open the file in the given mode.
<code>fs.close()</code>	Close the file to which <code>fs</code> is bound. Return void.
<code>fs.is_open()</code>	Return a bool to indicate whether the file associated with <code>fs</code> is open.

fs: Stream
f: File name

`fstream` or
`ifstream` or
`ofstream`

IO Library

- ❖ Each stream has an associated file mode
 - The mode can be changed when the file is opened
 - In output mode, the previous content is lost (if we do not append)

Type	Meaning
in	Open for input.
out	Open for output.
app	Seek to the end before every write. All writes are at the end of the file.
ate	Seek to the end immediately after open. Then, it is possible to move around (seek).
trunc	Truncate the file. In output, the default it truncate the file (rewrite it) even if trunc is not specified.
binary	Perform IO operation in binary mode.

Examples

Opening a file in
different mode

```
// Out and trunc are implicit
ofstream out("myfile");

// Trunc is implicit
ofstream out("myfile", ofstream::out);

ofstream out("myfile", ofstream::out | ofstream::trunc);

ofstream out("myfile", ofstream::out | ofstream::app);

// Out is implicit
ofstream out("myfile", ofstream::app);
```

Examples


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

typedef struct my_s {
    string title;
    int x, y;
} my_t;

my_t test;
test.title = ""; test.x = test.y = 0;
ofstream file;

file.open ("example.txt", ofstream::out);
file << test.title;
file << "\t" << test.x;
file << "\t" << test.y;

file.close();
```



Write some file
content

Examples

```
#include <iostream>
#include <fstream>

using std::cout; using std::cerr;
using std::endl; using std::string;
using std::ifstream;

int main() {
    string filename("input.txt");
    int number;

    ifstream infile(filename);
    if (!infile.is_open()) {
        cerr << "Error: " << filename << endl;
        return EXIT_FAILURE;
    }
    ...
    infile.close();

    return EXIT_SUCCESS;
}
```

Opening a file in
reading mode

Main reading cycle
... see ahead

Examples

```
int number;

while (infile >> number) {
    cout << number << "; ";
}
cout << endl;
```

Read integer
values

```
while (infile.get(c)) {
    cout << c << "; ";
}
cout << endl;
```

Read single
characters

Examples

Does not read
(and print)
newlines

```
string s;  
while (getline(infile,s)) {  
    cout << s << "; ";  
}  
cout << endl;
```

Read entire file
lines

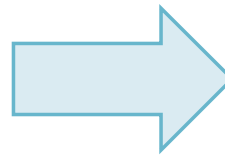
```
string s;  
while (infile >> s) {  
    cout << s << "; ";  
}  
cout << endl;
```

Read single
strings

Exercise: IO in C, UNIX, and C++

❖ An IO library comparison

- Write a segment of C code that writes and then reads a sequence of N integer values
- Use the ASCII and binary forms
 - C library
 - UNIX library
 - C++ library
- Compare file size and runtimes



```
#include <iostream>
#include <fstream>
#include <cstdlib>

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <unistd.h>
#include <fcntl.h>

using std::cout;
using std::endl;
using std::fstream;
using std::ofstream;
using std::ifstream
```


Solution

Time library

Computing times

```
#include <chrono>
```

```
long long microseconds;
```

```
auto start = std::chrono::high_resolution_clock::now();
```

```
read();
```

```
OR
```

```
write();
```

Either write or read the file

```
auto elapsed = std::chrono::high_resolution_clock::  
    now() - start;
```

```
microseconds = std::chrono::duration_cast<std::chrono::  
    microseconds>(elapsed).count();
```

```
cout << " C    Write Time: " << microseconds << endl;
```

Solution

The C library is buffered

```
setbuf(fp, 0);  
fp = fopen ("...", "w");  
for (i=0; i<n; i++) {  
    val = rand();  
    fprintf (fp, "%d\n", val);  
}  
fclose (fp);
```

C Buffered

Force a buffer size
equal to zero

```
fp = fopen ("...", "r");  
while (fscanf (fp, "%d", &val) != EOF) {  
    i++;  
}  
fclose (fp);
```

Solution

UNIX

```
fd = open ("...", O_WRONLY | O_CREAT | ...),  
for (i=0; i<n; i++) {  
    val = rand();  
    write (fd, &val, sizeof (int));  
}  
close (fd);
```

```
fd = open ("...", O_RDONLY);  
while (read (fd, &val, sizeof (int)) != 0) {  
    i++;  
}  
close (fd);
```

Solution

Trade-off Time-Memory

UNIX +

```
v = (int *) malloc (n * sizeof (int));  
fd = open ("...", O_WRONLY | O_CREAT | ...);  
for (i=0; i<n; i++) {  
    v[i] = rand();  
}  
write (fd, v, n*sizeof (int));  
close (fd);
```

```
fd = open ("...", O_RDONLY);  
read (fd, v, n*sizeof (int));  
close (fd);
```

Solution

C++

```
s.open ("...", ofstream::out);  
for (i=0; i<n; i++) {  
    val = rand();  
    s << val << endl;  
}  
s.close();
```

Force an fflush
Unbuffered

```
s.open ("...", ifstream::in);  
while (s >> val) {  
    i++;  
}  
s.close();
```

Solution

C++ Binary

```
s.open ("...",  
        ofstream::out | ofstream::binary);  
for (i=0; i<n; i++) {  
    val = rand();  
    s.write ((char *) &val, sizeof (int));  
}  
s.close();
```

```
s.open ("...", ifstream::in | ifstream::binary);  
while (s.read((char *) &val, sizeof (int))) {  
    i++;  
}  
s.close();
```


Solution

❖ File Size [MBytes]

Variable
(due to the size of
random integers)

$$n = 2 \cdot 10^6$$

Library	ASCII	Binary
C, UNIX, C++	20.965	8.00

❖ CPU Time [seconds]

Fixed

Library	Write	Read
C (ASCII, buffered)	0.172	0.125
UNIX (binary)	1.513	0.411
C++ (ASCII, unbuffered)	1.762	0.102

Solution

❖ File Size [MBytes]

Variable
(due to the size of
random integers)

$$n = 2 \cdot 10^6$$

Library	ASCII	Binary
C, UNIX, C++	20.965	8.00

❖ CPU Time [seconds]

Fixed

Library	Write	Read
C (ASCII, buffered)	0.172	0.125
C (ASCII, unbuffered)	1.806	0.139
UNIX (binary)	1.513	0.411
UNIX (binary, array)	0.028	0.001
C++ (ASCII, unbuffered)	1.762	0.102
C++ (binary)	0.046	0.022