A C++ program for doing the same task:

```cpp
// listcpp.cpp
#include <fstream>  // to use fstream class
using namespace std; // to use standard C++ library

main() {
    char ch;
    fstream infile;
    infile.open("A.txt", ios::in);
    infile.unsetf(ios::skipws);
    // set flag so it doesn't skip white space
    infile >> ch;
    while (! infile.fail()) {
        cout << ch;
        infile >> ch;
    }
    infile.close();
}
```

In C programming language, this variable is declared as follows:

```c
FILE * outfile;
```

In C++ the logical name is the name of an object of the class fstream:

```cpp
fstream outfile;
```

In both languages, the logical name `outfile` will be associated to the physical file `myfile.txt` at the time of opening the file as we will see next.

---

### Opening Files

Opening a file makes it ready for use by the program. Two options for opening a file:

- open an **existing** file
- create a **new** file

When we open a file we are positioned at the beginning of the file.

**How to do it in C:**

```c
FILE * outfile;
outfile = fopen("myfile.txt", "w");
```

The first argument indicates the physical name of the file. The second one determines the “mode”, i.e., the way, the file is opened. The mode can be:

- "r": open an existing file for input (reading);
- "w": create a new file, or truncate existing one, for output;
- "a": open a new file, or append an existing one, for output;
- "x": open an existing file for input and output;
- "ux": create a new file, or truncate an existing one, for input and output;
- "ax": create a new file, or append an existing one, for input and output;
- "rb", "wb", "ab", "rb+", "wb+", "ab+": same as above but the file is open in binary mode.
How to do it in C++:

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

int main()
{
    std::ofstream outfile;
    outfile.open("myfile.txt", ios::out);
    outfile << "This is the output file.
    
    After closing a file, the logical name is free to be associated to another physical file.
    
    Closing a file used for output guarantees that everything has been written to the physical file. 
    
    Files are usually closed automatically by the operating system at the end of program’s execution. 
    
    It’s better to close the file to prevent data loss in case the program does not terminate normally.
    
    In C :
    fclose(outfile);
    
    In C++ :
    outfile.close();
    
    Exercise: Open a physical file "myfile.txt" associating it to the logical file "afile" and with the following capabilities:
    1. input and output (appending mode):
       afilie.open("myfile.txt",
                   ios::in|ios::out|ios::app);
    2. create a new file, or truncate existing one, for output:
       ofstream outfile;
       outfile.open("myfile.txt", ios::out);

    in C (or in C++ using C streams):
    char c; // a character
    char a[100]; // an array with 100 characters
    FILE * infile;
    infile = fopen("myfile.txt","r");
    fread(&c,1,1,infie); // reads one character
    fread(a,1,10,infie); // reads 10 characters
    for:
    1st argument: destination address (address of variable c) 
    2nd argument: element size in bytes (a char occupies 1 byte) 
    3rd argument: number of elements
    4th argument: logical file name
```

Closing Files

This is like "hanging up" the line connected to a file.

After closing a file, the logical name is free to be associated to another physical file.

Closing a file used for output guarantees that everything has been written to the physical file.

We will see later that bytes are not sent directly to the physical file one by one; they are first stored in a buffer to be written later as a block of data. When the file is closed the leftover from the buffer is flushed to the file.

Files are usually closed automatically by the operating system at the end of program’s execution.

It’s better to close the file to prevent data loss in case the program does not terminate normally.

In C :
    fclose(outfile);

In C++ :
    outfile.close();
Contents of today's lecture:

- Field and record organization (textbook Section 4.1)
- Sequential search and direct access (textbook Section 5.1)
- Seeking (textbook Section 2.5)


Files as Streams of Bytes

So far we have looked at a file as a stream of bytes. Consider the program seen in the last lecture:

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

int main()
{
    char ch;
    ifstream inf;
    inf.open("A.txt", ios::in);
    inf.unsetf(ios::skipws);
    // set flag so it doesn't skip white space
    while (! inf.fail()) {
        cout << ch;
        inf >> ch;
    }
    inf.close();
}
```

Consider the file example: A.txt

87353CApUELLALICE IN WONDERLAND <nl>
88435HOL FILE STRUCTURES <nl>
78733KNUTH THE ART OF COMPUTER PROGRAMMING <nl>
86683KURT SURREAL NUMBERS <nl>
18895TOLKIEN THE HOBBIT <nl>

(above we are representing the invisible newline character by <nl>)

Field and Record Organization

Definitions:

- **Record** = a collection of related fields.
- **Field** = the smallest logically meaningful unit of information in a file.
- **Key** = a subset of the fields in a record used to identify (uniquely, usually) the record.

In our sample file "A.txt" containing information about books:

Each line of the file (corresponding to a book) is a record.

Fields in each record: ISBN Number, Author Name and Book Title.

**Primary Key:** a key that uniquely identifies a record. Example of primary key in the book file:

**Secondary Keys:** other keys that may be used for search. Example of secondary keys in the book file:

Note that in general not every field is a key (keys correspond to fields, or combination of fields, that may be used in a search).
Consider the following sample program:

```c
#include <fstream>
using namespace std;
int main() {
    fstream myfile;
    myfile.open("test.txt",ios::in|ios::out|ios::trunc|ios::binary);
    myfile<<"Hello, world.\nHello, again."
    myfile.seekp(12,ios::beg);
    myfile<<'X'<<'Y';
    myfile.seekp(3,ios::cur);
    myfile<<'Z';
    myfile.seekp(-2,ios::end);
    myfile<<'Z';
    myfile.fclose();
    return 0;
}
```

Show "test.txt" after the program is executed:

```
Hello, world.
Hello, again.
```

Remove `ios::binary` from the specification of the opening mode. Show test.txt after the program is executed under DOS:

```
Hello, world.
Hello, again.
```

---

**Contents of today’s lecture:**

- Secondary storage devices
- Organization of disks
- Organizing tracks by sector
- Organizing tracks by blocks
- Nondata overhead
- The cost of a disk access
- Disk as a bottleneck


---

**Secondary Storage Devices**

Since secondary storage is different from main memory we have to understand how it works in order to do good file designs.

Two major types of storage devices:

- **Direct Access Storage Devices (DASDs)**
  - Magnetic Disks
    - Hard Disks (high capacity, low cost per bit)
    - Floppy Disks (low capacity, slow, cheap)
  - Optical Disks
    - CD-ROM = Compact Disc, read-only memory
      (Read-only/write once, holds a lot of data, cheap reproduction)

- **Serial Devices**
  - Magnetic tapes (very fast sequential access)