A C++ program for doing the same task:

```
// 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();
}
```

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:

```
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 ouput;
- `"r+"`: open an existing file for input and output;
- `"w+"`: create a new file, or truncate an existing one, for input and output;
- `"a+"`: create a new file, or append an existing one, for input and output;
- `"rb", "wb", "ab","r+b","w+b","a+b"`: same as above but the file is open in binary mode.

How to do it in C++:

```
fstream outfile;
outfile.open("myfile.txt",ios::out);
```

The second argument is an integer indicating the mode.
Its value is set as a "bitwise or" (operator `|`) of constants defined in the class `ios`:

- `ios::in` open for input;
- `ios::out` open for output;
- `ios::app` seek to the end of file before each write;
- `ios::ate` initially position at the end of file;
- `ios::trunc` always create a new file (truncate if exists);
- `ios::binary` open in binary mode (rather than text mode).

C:	`r`	`w`	`a`
C++:	`in`	`out|trunc` or `out`	`out|app`
C:	`r+`	`w+`	`a+`
C++:	`out|in`	`out|in|trunc`	`out|in|app`

All obtions above, if followed by `b` in C, would have `|binary`.

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):
   ```
   afile.open("myfile.txt",
   ios::in|ios::out|ios::app);
   ```
2. create a new file, or truncate existing one, for output:

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 :

```
#include <fstream>
using namespace std;
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();
}
```

Consider the file example: **A.txt**

```
87358CARROLLALICE IN WONDERLAND      <nl>
03818FOLK   FILE STRUCTURES          <nl>
79733KNUTH  THE ART OF COMPUTER PROGR<nl>
86683KNUTH  SURREAL NUMBERS          <nl>
18395TOLKIENTHE HOBITT               <nl>
```

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

Consider the following sample program:

```
#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'<<'X';
  myfile.seekp(3,ios::cur);
  myfile<<'Y';
  myfile.seekp(-2,ios::end);
  myfile<<'Z';
  myfile.close();
  return 0;
}
```

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

| | | | | | | | | | | | | | | | | | | | | | | | | | |

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

| | | | | | | | | | | | | | | | | | | | | | | | | | |