The eof Member Function

1. When a file is created and saved, a special invisible character called the *eof* character (which stands for the end of file character), is automatically placed at the end of the file.

2. Every input file stream has a member function called *eof*, that can be used to detect this *eof* character to determine if there is any more data to be read from the file that is connected to the input file stream.

3. This member function is used as follows:

   Input_Stream.eof( );

   The *eof* function takes no argument and returns a Boolean value that can be used to control loops and *if*-statements.

4. The Boolean value returned by the *eof* function is *true* only if the program has attempted to read past the end of the input file, otherwise the value returned is *false*.

5. If fin is an input file stream that has been connected to an external file, then the entire content of the file is read and then written to the screen with the

following while-loop:

```
char next;
fin.get(next);
while ( !fin.eof( ) )
{
cout << next;
fin.get(next);
}
```

6. The ordering of the above programming codes is absolutely crucial because fin.eof( ) does not become *true* until an attempt to read past the end of the file is made. Changing the order of the codes will likely result in an undesire output for the final looping sequence. For example, someone may write the above codes as follow:

```
char next;
while ( !fin.eof( ) )
{
fin.get(next);
cout << next;
}
```

If you read these lines superficially, it appears that the correct things are being done: while the end-of-file has not been reached, read the next character from the file and print it out onto the screen. It turns out the above loop almost works except that the last character is always printed twice. To see that, consider what happens inside the loop while there is still one last character to be read. The character is read without the member function *eof( )* of fin to become *false*. The character is then displayed on the screen. The loop is entered one more time, but this time the attempt to read a character from fin fails and *eof( )* becomes *true*. Notice as a result, the value stored in the variable "next" remains unchanged. However we are still inside the loop. The following output statement to the screen forces the last character to be displayed a second time. The loop is then finally exited.

7. Although using the *eof* function to detect the end of a file can work with the file containing numerical as well as text data, some people uses the *eof* function for text data and uses the extraction operator >> for numerical data.

8. //Program to create a file called cplusad.dat which is identical to the file
   //cad.dat, except that all occurrences of 'C' are replaced by "C++".
   //Assumes that the uppercase letter 'C' does not occur in cad.dat, except
   //as the name of the C programming language.

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

void add_plus_plus(ifstream& in_stream, ofstream&
out_stream);
//Precondition: in_stream has been connected to an
input file with open.
//out_stream has been connected to an output file
with open.
//Postcondition: The contents of the file connected
to in_stream have been
//copied into the file connected to out_stream, but
with each 'C' replaced
//by "C++". (The files are not closed by this function.)

int main( )
{
   ifstream fin;
   ofstream fout;

   cout << "Begin editing files.\n";

   fin.open("cad.dat");
   if (fin.fail( ))
   {
      cout << "Input file opening failed.\n";
      exit(1);
```

```
    }

    fout.open("cplusad.dat");
    if (fout.fail())
    {
      cout << "Output file opening failed.\n";
      fin.close( );
      exit(1);
    }

    add_plus_plus(fin, fout);

    fin.close( );
    fout.close( );

    cout << "End of editing files.\n";
    return 0;
}

void add_plus_plus(ifstream& in_stream, ofstream&
out_stream)
{
    char next;

    in_stream.get(next);
    while (! in_stream.eof())
    {
      if (next == 'C')
      out_stream << "C++";
```

```
        else
            out_stream << next;

            in_stream.get(next);
        }
    }
```