cisc1110 fall 2010 lecture VII

- *files*
- file I/O: input from and output to files
- file operations

data files

- the "file system" on your computer keeps track of what is stored on the computer's hard disk
- most file systems organize files into folders and subfolders (also called directories and subdirectories, which is the older, pre-graphical user interface terminology)
- files have different *types*
- we have talked about file name *extensions* and how most operating systems use the file name extension as a way of identifying the type of the file; some examples:
  - `.cpp` indicates a C++ source code file
  - `.exe` indicates an "binary executable" file
  - `.doc` indicates a Microsoft Word document file
  - `.ppt` indicates a Microsoft Powerpoint file
  - `.html` indicates an HTML file
- today we will talk about *data* files — these are different from the source code file and different from the executable file which comprise your program;
  these are files that your program can *open* and *read* and/or *write*

file operations

- file handling involves three steps:
  1. opening the file (for reading or writing)
  2. reading from or writing to the file
  3. closing the file
- files in C++ are *sequential access*
- think of a cursor that sits at a position in the file;
  with each read and write operation, you move that cursor's position in the file
- the last position in the file is called the "end-of-file", which is typically abbreviated as `eof`
- all the functions described on the next few slides are defined in the `<fstream>` header file

opening a file for writing

- first you have to define a variable of type `ofstream`;
  this "output file" variable will act like the cursor in the file and will point to the end of the file, advancing as you write characters to the file
- then you have to open the file:

```
ofstream outfile; // declare output file variable
outfile.open( "myfile.dat" ); // open the file
```

- you should check to make sure the file was opened successfully; if it was, then `outfile` will be assigned a number greater than 0; if there was an error, then `outfile` will be set to 0, which can also be evaluated as the boolean value `false`; so you can test like this:

```
if ( ! outfile ) {
  cout << "error opening output file!\n";  // output error message
  exit( 1 );  // exit the program
}
```

- note that the method `ofstream.open()` requires one arguments:
  - `filename`: a string containing the name of the file you want to open; this file is in the current working directory or else you have to include a full path specification
- note that you can also open a file in the same line where you declare the file variable:

```
// declare output file variable and open file
ofstream outfile( "myfile.dat" );
```

but you still should check if `outfile > 0`, to make sure that the file was opened okay

writing to a file

- once the file is open, you can write to it
- you write to it in almost the same way that you write to the screen
- when you write to the screen, you use `cout << ...`
- when you write to your output file, you use `outfile << ...`
- here is an example:

```
outfile << "hello world!\n";
```

- here is another example:

```
int x;
outfile << "x = " << x << endl;
```


closing a file

- when you are done writing to a file, you need to **close** the file
- you do this using the `close()` function, which is part of `ofstream`
- so, to close a file that you opened for writing, you have to invoke:

```
void ofstream.close(); // function header for closing an output file
```

- for example, if you opened `outfile` as in the previous slides, then you would close it like this:

```
outfile.close();
```


writing to a file: complete example

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  ofstream outfile;
  outfile.open( "test.dat" );
  if ( ! outfile ) {
    cerr << "error opening output file :-(" << endl;
    exit( 1 );
  }
  outfile << "hello world\n";
  outfile.close();
  return 0;
} // end of main()
```

opening a file for reading

- first you have to define a variable of type `ifstream`;
  this "input file" variable will act like the cursor in the file and will point sequentially from one character in the file to the next, as you read characters from the file
- then you have to open the file:

```
ifstream infile; // declare input file variable
infile.open( "myfile.dat", ios::in ); // open the file
```

- you should check to make sure the file was opened successfully;
  if it was, then `infile` will be assigned a number greater than 0;
  if there was an error, then `infile` will be set to 0, which can also be evaluated as the boolean value `false`; so you can test like this:

```
if ( ! infile ) {
  cout << "error opening input file!\n";  // output error message
  exit( 1 );  // exit the program
}
```


- note that the method `ifstream.open()` requires one argument:
  - `filename`: a string containing the name of the file you want to open; this file is in the current working directory or else you have to include a full path specification
- note that you can also open a file in the same line where you declare the file variable:

```
// declare input file variable and open file
ifstream infile( "myfile.dat" );
```

but you still should check if `infile > 0`, to make sure that the file was opened okay

reading from a file

- once the file is open, you can read from it
- you read from it in almost the same way that you read from the keyboard
- when you read from the keyboard, you use `cin >> ...`
- when you read from your input file, you use `infile >> ...`
- here is an example:

```
int x, y;
infile >> x;
infile >> y;
```

- here is another example:

```
int x, y;
infile >> x >> y;
```


- when reading from a file, you will need to check to make sure you have not read past the end of the file;
  you do this by calling:

```
infile.eof()
```

which will:
— return `true` when you have gotten to the end of the file (i.e., read everything in the file)
— return `false` when there is still something to read inside the file

- for example:

```
while ( ! infile.eof() ) {
  infile >> x;
  cout << "x = " << x << endl;
} // end of while loop
```

closing a file

- when you are done reading from a file, you need to **close** the file
- you do this using the `close()` function, which is part of `ifstream`
- so, to close a file that you opened for reading, you have to invoke:

```
void ifstream.close(); // function header for closing an input file
```

- for example, if you opened `infile` as in the previous slides, then you would close it like this:

```
infile.close();
```


reading from a file: complete example

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main() {
  ifstream infile;
  string msg;
  infile.open( "test.dat" );
  if ( ! infile ) {
    cerr << "error opening input file :-(" << endl;
    exit( 1 );
  }
  while ( ! infile.eof() ) {
    msg = "";
    infile >> msg;
    if ( msg.length() > 0 ) {
      cout << "msg=[" << msg << "]\n";
    }
  }
  infile.close();
} // end of main()
```


reading from a file using **getline**

- you can also use the `getline()` function to read from a file
- remember how using `cin >>` will only let you read one entity at a time, where each entity is separated by *whitespace*; whereas `getline()` will read until the end of a line
- the same rules apply to reading from a file as reading from the keyboard
- for example, if the file looks like this:
  `hello world`
  and you have:

```
ifstream infile( "myfile.dat" );
string s;
getline( infile, s );
```

then `s` will be assigned to `"hello world"` whereas, if you use:

```
infile >> s;
```

then `s` will be assigned to `"hello"`

reading from a file: another complete example

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main() {
  ifstream infile;
  string msg;
  infile.open( "test.dat" );
  if ( ! infile ) {
    cerr << "error opening input file :-(" << endl;
    exit( 1 );
  }
  while ( ! infile.eof() ) {
    msg = "";
    getline( infile, msg );
    if ( msg.length() > 0 ) {
      cout << "msg=[" << msg << "]\n";
    }
  }
  infile.close();
} // end of main()
```