today: loops and file operations

- loops
- file operations

loops

- *looping*, or *iteration*, means doing something more than once, perhaps doing something over and over and over and ... and over again
- there are times when you want your program to do something once, and there are other times when you want your program to do something more than once—without having to repeat the code again
- when you write a loop, you need to decide several things:
  - how will the program know when to stop looping?
  - will anything change about the behavior of the program each time the loop runs?
- in C++, there are two "control structures" that facilitate looping:
  - `while` : generally facilitates *condition-controlled* looping
  - `for` : generally facilitates *counter-controlled* looping

types of loops

- controlled, non-infinite loops have an end
- loops end in two ways:
  - because they have run for a certain number of times; these are called *counter-controlled loops*
  - because a condition has changed that causes them to stop running; these are called *condition-controlled* loops

condition-controlled loops: "while"

- we have already used condition-controlled loops:

```
boolean q;
q = false;
while ( q==false ) {
  ...
} // end of while loop
```

- the syntax for a *while* loop is:

```
while ( <condition> ) {
  <body-of-while-loop>
} // end of while loop
```

- the `<condition>` is something like `q==false` or anything that evaluates to a boolean value

- it is important that something happens in the body of the loop to change the value of the condition, eventually; otherwise you will have an infinite loop
- note that the condition can be false before the loop begins, in which case the loop will never execute!

counter-controlled loops: "for"

- a *for* loop is used when you know how many times you want something to run
- the syntax for a *for* loop is:

```
for ( <initialize-statement> ; <condition> ; <continuation-statement> )
  <body-of-for-loop>
} // end of for loop
```

- for example:

```
int i;
for ( i=0; i<10; i++ ) {
  cout << "hello\n";
}
```

this example will print the word `hello` on the screen ten times, each word on its own line

- the `<initialize-statement>` is something like `i=0;`
  typically, it initializes a variable referred to as the *loop counter*;
  this variable keeps track of how many times the loop executes
- the `<condition>` is something like `i<10;`
  typically, it evaluates the loop counter to make sure it has not exceeded its maximum (i.e., the number of times the loop should run)
- the `<continuation-statement>` is something like `i++`
  typically, it increments (or decrements) the loop counter
- it is important that something happens in the continuation statement (or the body of the loop) to change the value of the condition, eventually; otherwise you will have an infinite loop
- note that the condition can be false before the loop begins, in which case the loop will never execute!

infinite loops

- a loop that never ends is called an *infinite* loop
- an infinite loop will run as long as the program is running
- it is common when writing programs for robots to write infinite loops—programs that run as long as the robot is turned on
- however, it is *not* common and typically *unadvisable* to write infinite loops for programs that run on a computer
- in case, by mistake (!), you create an infinite loop on your computer, you can usually get the program to stop by pressing `Ctrl-C` (the control "Ctrl" key and the "C" key at the same time)
- if that doesn't work, try closing the window where the program is running
- if that doesn't work, you may have to kill the program using the TaskManager, which is invoked as follows:
  — on Windows, by pressing `Ctrl-Alt-Del`
  — on Mac, by pressing `option-apple-esc`

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 either the `<ifstream>` header file (for files you want to read from) or the `<ofstream>` header file (for files you want to write to)

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()` takes two 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
  - `mode`: which is set to `ios::in` when opening a file for input

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
```


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", ios::out ); // 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()` takes two 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
  - `mode`: which is set to `ios::out` when opening a file for output

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 reading from or writing to a file, you need to close the file
- you do this using the `close()` function, which is part of both `ifstream` and `ofstream`
- so, to close a file that you opened for reading, you have do this:

```
ifstream.close(); // close input file
```

- and, to close a file that you opened for writing, you have do this:

```
ofstream.close(); // close output file
```

- that's all!