FOR LOOPS, FILE HANDLING AND RANDOM NUMBERS

Today

- The `for` statement
- How to read data in from a file.
- How to write data out to a file.
- Generate random numbers

  On the course website you can find the sample program `roombaLog` which shows some aspects of filehandling.

The `for` loop statement

- Imagine we put the following code in `main` in one of the `roomba` examples:
- As usual `x` and `y` are the location of the robot.

```
int myCount;

for(myCount = 1 ; myCount <= 5 ; myCount++){

   x = x + 1;
   y = y + 1;
}
```

- This would increase the value of `x` and `y` (the position of the robot) by 5.

The `for` loop statement

- General structure:

```
for(<start>; <true or false> ; <change>)
{

    <some instructions>

}
```

- This works as follows

The `for` loop statement

- At the start of the loop, the instruction in `<start>` is carried out.
- We usually use this to set the value of a counter.
- Then `<true or false>` is tested to see if it is true or false.
- This is usually a test on the counter.
- If it is false, the program will skip to the } that marks the end of the control structure.
- If it is true the `<some instructions>` are executed.
- Once they are done, the instruction in `<change>` is executed.
- This is usually something that changes the value of the counter.
- Then `<true or false>` is tested again.
- Thus `<some instructions>` will be repeatedly executed until `<true or false>` becomes false.

Examples

- Let's go back to our initial example:

```
int myCount;

for(myCount = 1 ; myCount <= 5 ; myCount++){

   x = x + 1;
   y = y + 1;

}
```

- This increases the value of `x` and `y` by 5.

- While:

```
int myCount;

for(myCount = 10 ; myCount > 5 ; myCount--)
{

   x = x + 1;
   y = y + 1;

}
```

would do the same, but with different values of `myCount`.

- What about this one?

```
int myCount;

for(myCount = 3 ; myCount < 7 ; myCount++){

   x = x + 1;
   y = y + 1;

}
```

- This increases the value of `x` and `y` by 4.

- What would

  ```
  int myCount;

  for(myCount = 2 ; myCount < 8 ; myCount+=2)
  {

     x = x - 1;
     y = y + 1;
  }
  ```

  do?

Using files

- In the same way as we use `cin` to read data from the keyboard, we can read data from files.
- In the same way as we use `cout` to write data to the screen, we can write data to files.
- This allows us to store information on the computer's hard drive, and to use it when we want it without having to type it in each time.

File preliminaries

- To use read and write data to a file, we will make use of some *library functions*.
- To use these functions we need to add:

  ```
  #include <fstream>
  ```

  at the start of the program.
- We put this in the same place as:

  ```
  #include <iostream>
  ```

Simple input

- The function `cin` allows us to read input from the keyboard.
- A typical pattern of usage is:

  ```
  cout << "Enter a number" << endl;
  cin  >> x;
  ```

  which first *prompts* the user, then reads the next thing they type into the variable `x`.
- Other examples of using `cin` can be found in the various `roomba` programs on the course website.
- `cin` is the counterpart of `cout`.

Reading from a file

- To read from a file, we have to tell the program three things:
  - That we are going to read from a file.
  - How we will refer to the file inside the program.
  - What the name of the file on the hard drive is.
- We can do those three things using one command:

```
ifstream infile("commands.txt");
```

- The `ifstream` says we are going to read from a file.
- `infile` is the name we are going to use inside the program.
- `patient.dat` is the name of the file on the hard drive.

- Alternatively we can write this as two commands:

```
ifstream infile;
infile.open("commands.txt");
```

- Now that we have defined `infile` as an *input stream*, we can read data from it.
- We use `infile` much like `cin`.
- Thus:

```
infile >> command;
```

  reads the next character from the file into the variable `command`
- Once we have finished reading from the file, we close it:

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

Writing to a file

- To write to a file, we have to tell the program three things:
  - That we are going to write to a file.
  - How we will refer to the file inside the program.
  - What the name of the file on the hard drive is.
- Again we can do those three things using one command:

```
ofstream outfile("log.txt");
```

  or using two commands:

```
ofstream outfile;
outfile.open("log.txt");
```

- Once we have defined `outfile` as an *output stream*, we can send data to it.
- We use `outfile` much like `cout`.
- Thus:

  ```
  outfile << whatIDid;
  ```

  sends the value of the variable `whatIDid` to the file.
- Once we have finished reading from the file, we close it:

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

- When writing to a file it is important to close it — if the file isn't closed, the data that we have set to the file might not be stored in it.

File open modes

- When we open a file for writing, the computer discards any information that is in the file.
- This is not always what we want to do.
- We can control what happens by specifying the *file open mode*.
- For example, instead of:

  ```
  ofstream outfile;
  outfile.open("log.txt");
  ```

  we can have:

  ```
  ofstream outfile;
  outfile.open("log.txt", ios::app);
  ```

  which will write new output to the end of the file.

- There are other options.

  ```
  outfile.open("log.txt", ios::trunc);
  ```

  will discard any information in the file.

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

  will open the file for output, and is just another way to do:

  ```
  outfile.open("log.txt");
  ```

- We also have:

  ```
  outfile.open("log.txt", ios::nocreate);
  ```

  which will fail to open the file if it doesn't already exist.

- Finally, we have have:

  ```
  outfile.open("log.txt", ios::noreplace);
  ```

  which will fail to open the file if it does already exist.
- `noreplace` is thus the dual of `nocreate`.
- There are also modes for input files.
- We have:

  ```
  ifstream myfile;
  myfile.open("commands.txt", ios::in);
  ```

  which will open the file for input.

Random numbers

- In the lab today we will use random numbers to perk up the `roomba` example a little.
- To generate random numbers we use the function `rand()`
- For example;

```
    int x;
    x = rand();
```

- This assigns a random value to `x`. The value is somewhere between 0 and (at least) 32767.
- To use `rand()`, we need to add `#include<cstdlib>` to our program.

- Each time we run our program `rand()` will produce some (apparently) random numbers.
- But it will produce the *same* numbers each time we run the program.
- To get different numbers each time we run the program, we need to *seed* the random number generator.
- The usual way to do that is to add:

  `srand(time(NULL));`
- The `time(NULL)` uses the clock to generate a seed.
- We have to add `#include<ctime>` to do this.

Summary

- This lecture started with `for` loops.
- Then we looked at simple file handling.
- In particular, we looked at:
  - reading data in from; and
  - writing data out to

  simple sequential files.
- Finally, we looked at how to generate random numbers.