LECTURE 8:

MORE FILE I/O

COMP1002J: Introduction to Programming 2

Dr. Brett Becker (brett.becker@ucd.ie)

Beijing Dublin International College

Writing to Files

- To write to a file, the file must be opened for writing e.g.

```
fp = fopen( fname, "w" );
```

- If the file does not exist already, it will be created.
- **If the file does exist, it will be overwritten (so the old contents are lost)! This is irreversible! There is no recycle bin for this!**
- So, be careful when opening files for writing, in case you destroy a file unintentionally.
- Opening files for writing can also fail.
  - If you try to create a file in a directory where you do not have write access you will not be allowed and `fopen(…)` will fail.
  - If there is no space on the disk, it may also fail.

Character Output to Files

- The function `putc( c, fp )` writes character **c to the file** associated with the file pointer **fp.**
  - It works in the same way as `putchar()`, except it requires a file pointer to write to.
  - Remember we have also seen `getc( fp )`

- **Example:** Write a file copy program which copies the file "prog.c" to "prog.old"

- How do we solve this problem?

- *Identify what the main steps are…*

Character Output to Files

- Outline solution in pseudo code:
  - Open files (one read, one write)
  - Check open succeeded for both files
  - Read characters from prog.c
  - Write characters to prog.old until all characters are copied
  - Close files
- Lets write this program together…

Example: File Copy

- **Step 0:** Create an empty C program:

```
#include <stdio.h>
#include <stdlib.h>

main()
{
}
```

Example: File Copy

- **Steps 1&2:** Open the files and check for success…

```
fp1 = fopen( "prog.c", "r" ); /* open for reading */
fp2 = fopen( "prog.old”, "w" ) ; /* open for writing */

if ( fp1 == NULL ) /* check does file exist etc */
{
    printf( "Cannot open prog.c for reading \n" );
    exit(1); /* terminate program */
}
else if ( fp2 == NULL )
{
    printf( "Cannot open prog.old for writing\n" );
    exit(1); /* terminate program */
}
```

There is a better way to do this! What is it? See last week’s lecture!

Example: File Copy

- **Steps 3&4**: Read characters from prog.c and write to prog.old

- Before writing the code, we can take a few moments to refine our solution:

```
read character from prog.c
while not end of prog.c file
{
  write character to prog.old
  read next character from prog.c
}
```

Example: File Copy

- **Steps 3&4**: Read characters from prog.c and write to prog.old

- We can now easily implement the solution in C code:

```
c = getc( fp1 ) ; /* read char from prog.c */
while ( c != EOF )
{
    putc( c, fp2 ); /* copy to prog.old */
    c = getc( fp1 ); /* read next char */
}
```

Example: File Copy

- **Step 5**: Close the files & indicate success…

```
fclose(fp1);
fclose(fp2);

printf( "Files successfully copied.\n" );
```

- Now, save the file as prog.c, and compile, run it, and check what is in prog.old…

file: prog.c

Beyond Character I/O

- The C functions `fprintf` and `fscanf` do are the same as `printf` and `scanf` except that they work with files!

- `fprintf( fp, “I like %s\n", colour );`

  prints the string *I like blue* to the file associated with `fp`.
  (We assume the variable colour contains “blue”)

- `fscanf( fp, " %d %f", &hours , &rate );`

  reads an integer value into the variable `hours` and a floating point value into the variable `rate`.

  Both have one extra parameter (the file pointer) and otherwise work the same as `printf` and `scanf`.

Beyond Character I/O

- The C function `fgets` is similar to `gets` except that it reads from a file.

```
fgets( secret_word, 80, fp );
```

reads a string into the variable `secret_word` from the file associated with `fp`, here up to a maximum of 80 characters. It works a little differently to `gets()`…

Beyond Character I/O

- `fgets` **stores the newline character in the string and then the null character.**
  - This is **NOT** the same as `gets` which does not store the newline.
  - You may have to remove (overwrite with '\0') the newline character in some situations, depending on what you are trying to do.
  - This is a REALLY useful function: By allowing you to specify the maximum number of characters to read in, it allows you prevent trying to put more characters into a string than the string was defined to contain – string OVERFLOW.

Beyond Character I/O

- The problem with using non-character I/O is how to detect the end of file. Luckily C provides a function to check this called `feof(fp)`.
- this checks if the EOF has been reached.
- **Example**: Using `feof` to read integer values:

```
fscanf(fp, "%d", &value);
while (!feof(fp))
{
    // do something with the value...
    fscanf(fp, "%d", &value);
}
```

Beyond Character I/O

- Note that `feof()` does not check if the end of the file is *about* to be reached.

- It checks if the end of file <u>has already</u> been reached.

Problem: Making a Graph

- Create a file, “data.txt”, that contains the following values:

```
1 5 3 7 2
```

  The values are “space delimited” – there is a single space between each number.

- Write a program that reads in the values from the file and prints out a horizontal bar chart:

```
1 | *
5 | *****
3 | ***
7 | *******
2 | **
```

Problem: Making a Graph

- First, lets sketch out a solution:
  - Open the file for reading
  - Check the file opened correctly
  - Read each value from the file and print out the row in the graph
  - Close the file

Problem: Making a Graph

- **Steps 1&2**: Open file for reading, and check success

```
fp = fopen( "data.txt", "r" ); /* open for reading */

if ( fp == NULL ) /* check does file exist etc */
{
    printf( "Cannot open data.txt for reading \n" );
    exit(1); /* terminate program */
}
```

There is a better way to do this! What is it?

Problem: Making a Graph

- **Step 3**: Read each value from the file and print out the row in the graph

- Pseudo code:

```
while not at the end of file
{
  read the next integer
  print out the line of the graph as required
}
```

Problem: Making a Graph

- **Step 3**: Read each value from the file and print out the row in the graph

```
fscanf( fp, "%d", &value );
while ( !feof(fp) ) {
  printf( "%d | ", value );
  for( int i = 0; i < value; i++ ) {
      printf( "*" ); // could also use putchar( '*' );
   }
  printf( "\n" );
  fscanf( fp, "%d", &value );
}
```

Problem: Making a Graph

- **Step 3**: Read each value from the file and print out the row in the graph

```
for( int i = 0; i < value; i++ ) {
    printf( "*" ); // could also use putchar( '*' );
 }
```

Note! to declare `i` inside the for loop we must use gcc ‘c99’ mode. So to compile this we use:

```
gcc -std=c99 –o graph graph.c
```

Also, as the c99 mode is slightly different, we will get a compiler warning:

```
graph.c:4:1: warning: return type defaults to 'int'
 main()
 ^
```

So we change `main(){` to `int main(){` at the beginning of the program.

Problem: Making a Graph

- **Step 4**: Closing the file

```
fclose(fp);
```

file: graph.c

Problem: Making a Graph

- **Note!**

- If data.txt does not have a space after the final 2, only the first 4 lines will print out!
- Why?

file: graph.c

Resetting a file pointer

- What if you want to go through a file again, after you already did?
- The file pointer will be at the end of the file.
- To reset the file pointer to the beginning of the file you can use

```
rewind(fp);
```

where fp is the file pointer.