C-5 Using Files

Creating a file with values

Give the file the name of RESIST.DAT

Type these numbers inside the file:

```
1000
1100
2000
 500
1000
2000
```

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

int main (void)
{
   double r1, r2, r3, r_combo;
   FILE * input_file;              // Declare a pointer variable
   FILE * output_file;             // for each file.


         // more of this program on the next slide
```

```
input_file = fopen ("resist.dat", "r");
   if(input_file == NULL)
   {
       printf("Error on opening the input file \n");
       exit (EXIT_FAILURE);   // ( ) required since exit is a function
   }

 output_file = fopen ("resist.out", "w");
   if(output_file == NULL)
   {
       printf("Error on opening the output file \n");
       exit (EXIT_FAILURE);
   }
```

// **more** of this code on the **next** slide

```
/* Now that the files are open, we can use them */
 fprintf(output_file, "\nRuthann Biel.  Resistance Program.  \n\n");

 while ((fscanf(input_file, "%lf%lf%lf ", &r1, &r2, &r3)) == 3)
 {
    r_combo = 1.0 / (1.0/r1 + 1.0/r2 + 1.0/r3);
    fprintf(output_file, "Three resistors are:  %f  %f  %f \n", r1, r2, r3);
    fprintf(output_file, "Combined Parallel Resistance:  %f  \n\n",
            r_combo);
 }
 fclose(input_file);
 fclose(output_file);

 return EXIT_SUCCESS;
}
/*-----------------------------------------*/
```

This is the contents of ***resist.out***

```
Ruthann Biel.  Resistance Program.

Three resistors are:  1000.000000   1100.000000  2000.000000
Combined Parallel Resistance =  415.094330

Three resistors are:  500.000000   1000.000000  2000.000000
Combined Parallel Resistance =  285.714294
```

Programs & Files

A program can **read** from a file containing data as input.

A program can **write** data to a file as output (rather than to the screen).

For each file used, one must have a file pointer.

FILE * my_data;

<u>The Details</u>:

FILE - the word must be capitalized.

asterisk - indicates that it is a pointer variable.

my_data - name of the file pointer.
(created by the programmer, you or me)

Next step.

Associate the **file pointer with a file name** by using an *fopen*

my_data = fopen ("body_info.dat", "r");

The Details:

my_data - file pointer name

fopen - opens the file & creates a connection between the file and your program

body_info.dat – the file name as in Windows

"r" - means the file is for "read only"

Error Checking on **fopen**

After each **fopen**, one must error check.

```
if( file_pointer_name == NULL)
{
```

- print an error message so you will know what is wrong
- do an *exit(EXIT_FAILURE);*
  to leave the program.
  Makes no sense to go on without data

```
}
```

To read the data from the file into the program, use the *fscanf* statement **(**<u>**f**</u>ile <u>**scan**</u> <u>**f**</u>unction**)**

fscanf(my_data, “%lf%lf ”, &height, &weight);

Similar to a scanf, except:

(1) use fscanf instead of scanf

(2) add the file pointer name immediately inside the parentheses

To write the data to a file from the program, use the *fprintf* statement **(f**ile **print** function)

```
FILE * out_file;
…
out_file = fopen("results.out", "w");
…
fprintf(out_file, "%f%f ", height, weight);
```

Similar to a printf, except:

(1) use fprintf instead of printf

(2) add the file pointer name immediately inside the parentheses

When one is done with the files, close them.

```
fclose(my_data);

fclose(out_file);
```

One can also use **fclose** to re-open a file for repeated use.

A Variation or Alternative Way:

<u>Style 1</u>

```
FILE *my_data;
…
my_data = fopen ("body_info.dat", "r");
        // Use the file name directly in the fopen
```

<u>Style 2</u>

```
#define IN_FILE_NAME "body_info.dat"
…
FILE *my_data;
…
my_data = fopen (IN_FILE_NAME, "r");
        // Use a variable that holds the file name
        // Notice that the quotes are on the define line
        //  and there are NO quotes in the fopen for the file name
```

```
/* Using the alternative method */

#include <stdio.h>
#include <stdlib.h>
#define IN_FILE_NAME  “body_info.dat”
#define OUT_FILE_NAME “results.out”
int main (void)
{
   FILE * my_data;
   FILE * out_file;
   …
   my_data = fopen (IN_FILE_NAME, “r”);
   if (my_data == NULL)
   {
      printf (“Error opening the input file \n”);
      exit (EXIT_FAILURE);
   }                                //more on next slide
```

```
    out_file = fopen (OUT_FILE_NAME, “w”);
    if (out_file == NULL)
    {
        printf (“Error opening the output file \n”);
        exit (EXIT_FAILURE);
    }
    …
    fscanf(mydata, …);
    …
    fprintf(out_file, …);
    …
    fclose (my_data);
    fclose (out_file);
    ….
}
```

Reading a data file:

Contents of the file (in blue):

Date	Rain
1	0.0
2	0.1
3	0.9
4	1.5
5	2.0
6	1.1

```
while((fscanf(infile, "%d%f ", &date, &rain)) == 2)
{   ....
}
```

fscanf returns the number of values read; so here we continually read **two** values, until no more data.

Controlling Files and their End.

First example – FOR loop.

(1) We have a known number of records or lines in a file, so we can use a <u>for</u> loop.

```
int main (void)
{
   int i, n, max = 0;
   FILE * infile;
   infile = fopen("d.dat", "r");
   if (infile == NULL)
   {
      printf ("Error on input file open\n");
      exit (EXIT_FAILURE);
   }
   for (i = 1; i <= 5; i ++)
   {
      fscanf (infile, "%d", &n);
      if (n > max)
            max = n;
   }
   printf ("\nMax is %d \n\n", max);
   fclose (infile);
   return EXIT_SUCCESS;
}
```

FILE CONTENTS:

```
14
65
24
72
40
```

Classroom Program
Files_For.c

Second Example – DO WHILE loop

(2) We have a known trailer signal or sentinel signal
or a "dummy value" in the file,
so we can use a <u>do while</u> loop,
continuing to loop until we find the marker at the end.

```
int main (void)
{
   int i, n, max = 0;
   FILE * infile;
   infile = fopen("d2.dat", "r");
   fscanf (infile, "%d", &n);
   max = n;
   do
   {
      if (n > max)
         max = n;
      fscanf (infile, "%d", &n);
    } while (n > -1);

   printf ("Max is %d \n", max);
   fclose (infile);
   return EXIT_SUCCESS;
}
```

FILE CONTENTS:

```
14
65
24
72
40
-1
```

Classroom Program
Files_DoWhile.c

Third example –
FOR loop with length in file.

(3) We have a file where the first value in the file tells us how many values follow in the file, so we can use a <u>for</u> loop.

```
int main (void)
{
    int i, n, max = 0, end;
    FILE * infile;
    infile = fopen("d3.dat", "r");
    fscanf (infile, "%d", &end);

    for (i = 1; i <= end; i ++)
    {
        fscanf (infile, "%d", &n);
        if (n > max)
            max = n;
    }
    printf ("Max is %d \n", max);
    fclose (infile);
    return EXIT_SUCCESS;
}
```

FILE CONTENTS:

```
5
14
65
24
72
40
```

Classroom Program
Files_For_length.c

Fourth Example – Looking for the end.

(4) We look for a good read from the file. In this case, we are reading one value at a time.

```
while ( (fscanf(my_data, "%f ", &x) ) == 1)
```

If the fscanf returns a "1",
then we know we got a "good" read.

If the fscanf returns a "0",
then we know we are out of numbers.

```
int main (void)
{
    int n, max, n_pts = 0;
    FILE * infile;
    infile = fopen("d.dat", "r");
    while( (fscanf (infile, "%d", &n) ) ==1)
    {
        n_pts++;
        if (n_pts == 1)
            max = n;
        if (n > max)
            max = n;
    }
    printf ("Max is %d \n", max);
    fclose (infile);
    system("pause");
    return EXIT_SUCCESS;
}
```

FILE CONTENTS:

```
14
65
24
72
40
```

Classroom Program
Files_While_EOF.c

C-5 Files

Input/Output And Using Files

The End