Week 5: Strings and files

CS211: Programming and Operating Systems

Wednesday and Thursday, 10+11 March 2021

Usual reminders...

	Mon	Tue	Wed	Thu	Fri
09:00					
10:00					
11:00					
12:00					
13:00				***Recorded***	
14:00					
15:00	**LAB**		***Recorded***		
16:00	**LAB**				

1. The recorded classes take place Wednesdays at 15:00, and Thursdays at 13:00.
2. Lab times: **Monday 15:00-17:00**. Aim to attend for an hour. Drop in an out as needed.
3. Introduction to Lab 2 was be recorded, and is now available.
4. Using Blackboard all this week. Might switch to Zoom in the future...
5. There will be **no** class, and no recordings, next Wednesday (St. Patrick's Day).

In Week 5 of CS211:

1. Part 1: Recap on strings
2. Part 2: Multidimensional Arrays
   - Arrays of Strings
3. Part 3: Files
   - Getting started
   - Opening a file
   - Closing a file
   - An example
4. Part 4: Reading from a file
   - Using `fgets`
   - Using `fgetc`
5. Part 5: Navigating a file
   - Eg: Random Lines
6. Part 6: Writing data to a file
   - Further points
7. Exercises

Today

Thursday.

CS211
Week 5: Strings and files

Start of ...

PART 1: Recapping on strings

Wed at 3pm.

Part 1: Recap on strings

- A `char` is a one symbol, such as a letter, digit, or one of `!`, `%`, `$`, `*`, `+`, `=`, etc. Can be declared as
  `char SomeLetter='Q';`
  Note the use of single quotes.
- A **string** is a sequence of zero or more `char`'s, such as `"CS211"`, `"NUI Galway"`, or `"How the #$% is it only Wednesday?"`.
  Note the use of double quotes. Also, strings can have spaces, or indeed, any symbol (i.e., `char`).
- In C, a string is stored in a `char` array.
- A `\0` in the array designates the end of the string, irrespective of size of the declared array.
  E.g.,

```
char AWord[20]="supercilious";
printf("0: AWord=%s\n", AWord);
AWord[5]='\0';
printf("1: AWord=\%s\n", AWord);
```

Part 1: Recap on strings

- A char is a one symbol, such as a letter, digit, or one of !, %, $, *, +, =, etc. Can be declared as
  `char SomeLetter='Q';`
  Note the use of single quotes.
- A **string** is a sequence of zero or more char's, such as "CS211", "NUI Galway", or "How the #$% is it only Wednesday?". Note the use of double quotes. Also, strings can have spaces, or indeed, any symbol (i.e., char).
- In C, a string is stored in a char array.
- A \0 in the array designates the end of the string, irrespective of size of the declared array.
  E.g.,

```
char AWord[20]="supercilious";
printf("0: AWord=%s\n", AWord);
AWord[5]='\0';
printf("1: AWord=\%s\n", AWord);
```

So AWord[12] = '\0'

→ outputs "supercilious"

ignore anything after AWord[5].

→ "super".

Part 1: Recap on strings

- Other than in declarations, use the `strcpy()` ("string copy") function, from `string.h` to set a whole string:
  `strcpy(AWord, "acting superior");`
- Other functions in `string.h` are `strlen()`, `strcat()`, `strcmp()`, `strchr()`, `strstr()`, `strfry()`,...
- To output a string using `printf()`:

```
printf("%s", AWord);
```

- In input a string using `scanf()`:

```
scanf("%s", AnotherWord);
```

"make an anagram"

Character search

Part 1: Recap on strings

CS211
Week 5: Strings and files

END OF PART 1

CS211
Week 5: Strings and files

Start of ...

PART 2: Multidimensional Arrays

(for arrays of strings
= arrays of array of chars)

Part 2: Multidimensional Arrays

If an array (particularly of integers or floats) is like a mathematical vector, then how do we define a matrix?

A matrix is a two-dimensional array. For example, to declare a 3×4 matrix of floats, we would use the syntax:

```
float A[3][4];
```

(not, A[3,4] !!).

So

3 rows

$$A = \begin{pmatrix} A[0][0] & A[0][1] & A[0][2] & A[0][3] \\ A[1][0] & A[1][1] & A[1][2] & A[1][3] \\ A[2][0] & A[2][1] & A[2][2] & A[2][3] \end{pmatrix}$$

So A[i,j] corresponds to entry in row i+1, col j+1.

In general an $n \times m$ array is declared as

```
float A[n][m];
```

For a 1D array: float v[5].

Part 2: Multidimensional Arrays

If a program has the line:

```
int A[3][4];
```

What really happens is that the system creates **three** arrays, each of length **four**. More precisely, it

- declares 3 pointers to type `int`: `A[0]`, `A[1]`, and `A[2]`,
- space for storing an integer is allocated to each of the addresses `A[0]`, `A[0]+1`, `A[0]+2`, `A[0]+3`, `A[1]`, `A[1]+1`, ..., and `A[2]+3`.

ie A[i][j] ⟺ *(A[i] + j)

This means that if `A[][]` is declared as a two-dimensional 3×4 array, then the following are equivalent:

- `A[1][2]`
- `*(A[1] + 2)`
- `*( *(A + 1) + 2)`
- `*( &A[0][0] + 4 + 2)`

Part 2: Multidimensional Arrays

01Matrix.c

```
#include <stdio.h>
int main(void )
{
  int A[3][4]={{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};

  // Here are 4 different ways to access the "7" in this array.
  printf("A[1][2] = %d\n", A[1][2]);
  printf("*(A[1]+2) = %d\n", *(A[1] + 2));
  printf("*(*(A+1)+2) = %d\n", *( *(A + 1) + 2));
  printf("*(&A[0][0] + 4 + 2) = %d\n",
    *( &A[0][0] + 4 + 2));

  return(0);
}
```

Row 0 Row 1 Row 2

Part 2: Multidimensional Arrays

In another example , we'll sum all the entries of a 3×4 array.

02Sum_a_matrix.c

```
 6 #include <stdio.h>

 8 int sum(int A[][4]);

10 int main(void )
   {
12   int n;
     int A[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};

     n = sum(A);

     printf("Sum of the entries in A is %d \n",n);
18   return(0);
   }
```

ROWS

Part 2: Multidimensional Arrays

02Sum_a_matrix.c

```
    int sum(int A[][4])
22  {                          ← Dont explicity have # of rows,
      int i,j, ans=0;
24    for (i=0; i < 3; i++)
        for (j=0; j< 4; j++)
26        ans += A[i][j];

28    return(ans);
    }
```

Important: Notice that this function is defined only for arrays of size 3×4. Even if we passed n and m as arguments to the function, we would still have to declare that `A` has 4 columns.

Multidimensional arrays often occur when dealing with arrays of strings.

Recall that in C, a string (collection of characters) is stored as a `char` array.

name [3]

```
char Name[20]="Ada Lovelace";
```

This means that we have declared `Name` to be an array of 20 characters:

- `'A'` is stored in `Name[0]`
- `'d'` is stored in `Name[1]`
- `'a'` is stored in `Name[2]`
- ...
- `'c'` is stored in `Name[10]`
- `'e'` is stored in `Name[11]`
- and `'\0'` is stored in `Name[12]`.

The remaining entries, `Name[13]`, . . . , `Name[19]` are unused.

Part 2: Multidimensional Arrays — Arrays of Strings

If a single string is stored as a character array, then an array of strings is an ***Array of Arrays of `chars`***, more often called a ***two dimensional array***.

Example

```
char Names[10][20];
strcpy(Names[0],"A. Lovelace");
strcpy(Names[1],"C. Babbage");
...
strcpy(Names[8],"D. Richie");
strcpy(Names[9],"K. McNulty"); [a]
```

[a] For more about Donegal's greatest computer scientist, see https://en.wikipedia.org/wiki/Kathleen_Antonelli

We can think of this as a matrix, and visualise it as

	0	1	2	3	4	5	6	7	8	9	10	11
Name[0]	A	.		L	o	v	e	l	a	c	e	\0
Name[1]	C	.		B	a	b	b	a	g	e	\0	-
	⋮				⋮				⋮			⋮
Name[8]	D	.		R	i	c	h	i	e	\0	-	-
Name[9]	K	.		M	c	N	u	l	t	y	\0	-

Clearly there is some waste of memory space. On another day, we might study the use of "ragged arrays" can avoid this.

CS211
Week 5: Strings and files

END OF PART 2

CS211
Week 5: Strings and files

Start of ...

PART 3: Files

Part 3: Files

Most useful programs obtain their input from a **file**, and store their output to a file.

For example, in Lab 3 we'll write a crossword helper that uses data stored in a file.

Further details can be found in Chap. 22 of King's "***C Programming***" or Chap 11 of Kelley and Pohl's "***A Book on C***".

Soon post to library reading list.

Taking input from a file is not much different that taking input from the keyboard. All we do is:

1. Declare an identifier for the file, (`FILE *`)
2. open the file, (`fopen`)
3. read from it, (or write)
4. close the file. (`fclose`)

Data type (kinda) for files.

Declaring a ***File Identifier*** is easy:

```
FILE *datafile;
```

So `datafile` is now a pointer that we can associate with a file or, more generally, a `stream`.

Opening a file

```
fileptr = fopen(char *FileName, char *Mode);
```

The `fopen()` function is used for file opening. It takes two arguments: the `name` of the file to open and the ***mode*** it will operate in. A file pointer is returned.

mode

The most important modes for file operation are reading and writing, but there is also appending.

So FileName is a string.

```
fileptr = fopen(char *FileName, char *Mode);
```

Read mode: "r"

Use `fopen(FileName, "r")` to open a file that we want to read from. It is assumed that the file already exists. If it doesn't, `NULL` is returned.

Example

```
FILE *infile;
infile = fopen("OldFile.txt", "r");
```

file name

open for reading

Write mode: "w"

Use `fopen(FileName, "w")` to open a file we want to `w`rite to. If the file does **not** already exist, it is created. If it is already in the file system, the contents are deleted.

Example

```
FILE *outfile;
outfile = fopen("NewFile.txt", "w");
```

There is also ***append*** mode: `"a"`, used to to `a`ppend data to end of the file. The file is opened in **write** mode, but new data is added to the end, i.e., its existing contents are not overwritten.

In our examples, we assume that

- The we only want to read from the file.
- That we know its name in advance.

So our code includes

```
FILE *fileptr;
fileptr=fopen("list.txt", "r");
```

If the file can't be opened, `NULL` is returned.

- Because it does not exist, or...
- We don't have permission.

When we are done, we should close the file

```
fclose(fileptr);
```

Example

Give a segment of code that prompts the user for name of an input file, and opens it in `r`ead mode. If a file *cannot* be opened, an error should be returned.

```
int main( void)
{
  char infilename[20];
  FILE *infile;

  printf("Enter file to read from: ");
  scanf("%s", infilename);
  infile=fopen(infilename, "r");

  if (infile == NULL)
  {
    printf("Error: couldn't open for reading");
    return (EXIT_FAILURE);
  }
  else
     printf("Opened %s for reading\n", infilename);
```

Apart from `fopen` and `fclose`, the important functions for manipulating files are

- **Reading:** `fgetc` and `fgets` (also: `fscanf`) } Part 4
- **Writing:** `fputc`, `fputs` and `fprintf` } Part 6
- **Check and change file counter:** `rewind`, but also `ftell` and `fseek`. } Part 5

used to navigate in the file.

CS211
Week 5: Strings and files

END OF PART 3

Finished here
Wed @ 3.50.

CS211
Week 5: Strings and files

Start of ...

PART 4: Reading from a file

Recorded Thursday @ 1 (ish)

Part 4: Reading from a file

There are quite a number of functions for reading data from a file. We'll look at two functions: `fgetc()` and `fgets()`

`fgets` : reads a string from a file

```
fgets(string, n, fileptr)
```

reads in a line of text from the *fileptr* stream and stores at most `n` characters in array `sting`. The new line character is stored.
If the string can't be read, because we have reached the end of the file, then `NULL` is returned.

`fgets` : reads a character from a file

```
c = fgetc(fileptr)
```

reads the next character in the file and stores it in the `char` variable `c`. If the end of the file has been reached, `EOF` is returned.

Part 4: Reading from a file

Also: `fscanf(fileptr, "%s", CharArray);`
works rather like `scanf()` except that the input stream is `fileptr` rather than `stdin`.

(But I prefer not to use it, since `fgets` and, especially, `fgetc` are more predictable and easier to debug).

of course, could use %d, %f, etc

BTW: scanf() is actually just fscanf()!

scanf("%d", &x) <=> fscanf(stdin, "%d", &x)

Example 1: Write a function that counts the number of lines in a file using `fgets()`

03CountLinesWithfgets.c

```
   #include <string.h>

   int file_length(FILE *);

   int main( void)
12 {
     char FileName[30];
14   FILE *file;

16   strcpy(FileName, "03CountLinesWithfgets.c");
     file=fopen(FileName, "r");

     printf("%s has %d lines\n", FileName,
20     file_length(file));
     return(EXIT_SUCCESS);
22 }
```

This program reads itself!

"read mode"

`03CountLinesWithfgets.c`

```c
   int file_length(FILE *file)
36 {
     int lines;
38   char dummy[100];

40   rewind(file);

42   lines=0;
     while( fgets(dummy, 100, file) != NULL)
44     lines++;

46   rewind(file);

48   return(lines);
   }
```

already opened in main

store data from file

"not NULL" means we are not at the end of the file.

see Part 5 for "rewind".

We'll redo this example but using `fgetc`. It reads one character at a time so we'll just count the number of times a newline is read. ie '\n'.

Note that `EOF` — `End of File` — is returned when we try to read beyond the end of the file.

04CountLinesWithfgetc.c

```
26  int file_length(FILE *file)
    {
28    int lines;
      char c;

      rewind(file);

      lines=0;
34    do {
        c = fgetc(file);
36      if (c == '\n')
          lines++;
38    } while(c != EOF);

40    rewind(file);
      return(lines);
42  }
```

will do these at least once.

"c is not the End OF File value.

CS211
Week 5: Strings and files

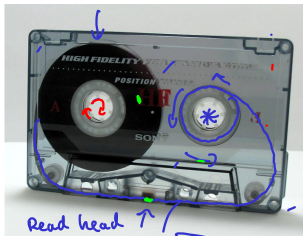
END OF PART 4

CS211
Week 5: Strings and files

Start of ...

PART 5: Navigating a file

Recall "rewind" from Part 4.



Part 5: Navigating a file

Each time a character is read from the input stream, a counter associated with the stream is incremented.

In `03CountLinesWithfgets.c` we saw this when we used the `rewind` function:

rewind

`rewind(fileptr)` sets the indicator to the start of the file. This was used in our earlier examples.

There are some other useful function which can be used

- To determine here in the file we are: `ftell`
- To move to a particular location in the file: `fseek`.

Part 5: Navigating a file

ftell

To check the current value of the file position indicator, use:

```
long ftell(FILE *stream);
```

It will return the current value of the file position indicator, in the form of a long int.

For example, if we are at the beginning of the file, then `ftell(file)` should evaluate as `0`.

long int

Part 5: Navigating a file

fseek

To modify the value of the indicator:

```
fseek(fileptr, offset, place)
```

The value of *offset* is the amount the indicator will be changed by, while `place` is one of

- *SEEK_SET* (0), refers to the start of the stream,
- *SEEK_CUR* (1), refers to the current position of the indicator,
- *SEEK_END* (2), refers to the end of the stream,

For example,
`fseek(file, 0, SEEK_SET)`
is equivalent to
`rewind(file)`.

Part 5: Navigating a file

Example

Here is an easy way of counting the number of characters in a file:

```
fseek(file, 0, SEEK_END);
printf("There are %ld chars in the file\n",
    ftell(file));
```

Part 5: Navigating a file

Example

Write a programme that will open a file and output its contents in reverse.

05Reverse.c

```
   int main( void)
10 {
     FILE *InFile;
12     char c;

14     InFile=fopen("0[illegible]Reverse.c", "r");
       if ( InFile == NULL )
16     {
         printf("Error - could not open the file\n");
18       exit(1);
       }
```

05Reverse.c (cont.)

```c
    // First go to the end of the file
22  fseek(InFile, 0, SEEK_END);

24  // Now read lines in reverse order
    while (ftell(InFile) != 0)
26  {
      c=fgetc(InFile);
28    putchar(c);
      fseek(InFile, -2, SEEK_CUR);
30  }

32  fclose(InFile);
    return(0);
34 }
```

See also the exercise on Slide 56.

In our next example, we'll write a program that reads a number of lines from a file and then outputs them at random.

It contains the following

- Some comments
- Some `#include directives`
- The beginning of the `main` function, followed by some variable declarations.
- Copies the string `06RandomLines.c` to the array `FileName`; tries to open the file for reading; if that fails, generate an error and exit.
- Reads each line of the file into the two dimensional `char` array `lines[][]`; for each line, increments the variable `NumberOfLines` ; closes the file.

- Set the integer variable `Deleted` to `0`.
- Until all lines have been "***deleted***",
  - generate a random number between `0` and `NumberOfLines`
  - If the corresponding line has not yet been deleted,
    - > display the line,
    - > "delete" the line by setting the first char to `\0`
    - > increment the `Deleted` variable.

`06RandomLines.c`

```
4  #include <stdio.h>
   #include <stdlib.h>
6  #include <string.h>

8  int main(void )
   {
10   int i, NumberOfLines=0, Deleted, WhichLine;
     char lines[100][100], FileName[30];
12   FILE *infile;
```

06RandomLines.c

```
14   strcpy(FileName, "06RandomLines.c");
     infile = fopen(FileName, "r");
16   if (infile == NULL)
     {
18     printf("Error: can't open %s for reading",
              FileName);
20     exit(EXIT_FAILURE);
     }

     for (i=0; (fgets(lines[i], 99, infile)) != NULL; i++)
24     NumberOfLines++;

26   fclose(infile);
```

Reading in file

06RandomLines.c

```
28   // Now display non-empty lines in a random order
     Deleted=0;
30   while(Deleted < NumberOfLines)
     {
32     WhichLine = rand()%NumberOfLines;
       if (lines[WhichLine][0] != '\0')
34     {
         printf("%s", lines[WhichLine]);
36       lines[WhichLine][0]='\0';
         Deleted++;
38     }
     }
40   return(EXIT_SUCCESS);
```

[have not yet deleted the line].

CS211
Week 5: Strings and files

END OF PART 5

CS211
Week 5: Strings and files

Start of ...

PART 6: Writing data to a file

Last

Part 6: Writing data to a file

Finally, we will study how to create a new file and write data to it.

First, as usual, declare a file pointer:

```
FILE *outfile;
```

Can be any identifier you choose.

Then open a new file in write mode:

```
outfile=fopen("NewList.txt", "w");
```

To write to the file, use one of

- `fprintf(FILE *stream, ...)`: works just like `printf()` except that its first argument is the output stream.
- `fputc(char c, FILE *stream)`: writes the character `c` to the stream,
- `fputs(char *str, FILE *stream)`: writes the string *str* to the stream, without its trailing `'\0'`

Part 6: Writing data to a file

Example

Write a program that copies every fifth line from an input file into an output file.

07DeleteLines.c

```
   int main(void)
12 {
     FILE *infile, *outfile;
14   char InFileName[99], OutFileName[99], Line[99];
     int i;

     printf("Enter the name of the input file: ");
18   scanf("%s", InFileName);
     printf("Enter the name of the output file: ");
20   scanf("%s", OutFileName);
```

Part 6: Writing data to a file

07DeleteLines.c

```
22    infile = fopen(InFileName, "r");
      if (infile == NULL)
24    {
        printf("Can't open %s in read mode\n",
26              InFileName);
        exit(EXIT_FAILURE);
28    }
      outfile = fopen(OutFileName, "w");
30    if (outfile == NULL)
      {
32      printf("Can't open %s in write mode\n",
                OutFileName);
34      exit(EXIT_FAILURE);
      }
```

Checking that open worked.

Part 6: Writing data to a file

07DeleteLines.c

```
    i=0;
38  while ( fgets(Line, 99, infile) != NULL )
    {
40    i++;
      if (i%5 == 0)
42      fputs(Line, outfile);
    }

    fclose(infile);
46  fclose(outfile);

48  return(EXIT_SUCCESS);
}
```

(count lines).

← read one line at a time

i%5 ==0 is true if i=0, i=5, i=10, i=15, ...

(Eg if i=14, then i%5 would be 4.

Part 6: Writing data to a file

Issues concerning the use of files in C, but which we haven't covered, include

append

- There are in fact 6 modes a file can have: `r`, `w`, `a`, `r+`, `w+`, `a+`.
- To open a binary file, also include the letter `b` as part of the mode.
- `freopen()` attaches a new file to an existing stream
- `tmpfile()` opens a temporary file in binary read/write (w+b) mode. The file is automatically deleted when it is closed or the program terminates.
- `fflush()` flushes a buffer
- `remove()` and `rename()` can be used to manipulate files in a directory.
- `int feof(FILE *stream)` returns a nonzero character if the file position indicator is at the end of the file.

Exercises

Exercise (Exer 6.1)

In the `04CountLinesWithfgetc.c` we used `rewind()` to move the file position indicator to the start of the file, before counting the number of lines, and then rewind it when we are done. This means that, after any call to `file_length()` the file position indicator is set to the start of the file; that is, we lose the current position.
Improve the code so that in the `file_length()` function

- *first stores the current file position;*
- *then `rewind`s the file;*
- *counts the the number of lines;*
- *resets the file position indicator.*