CS 241
Data Organization using C

File Input / Output: IO

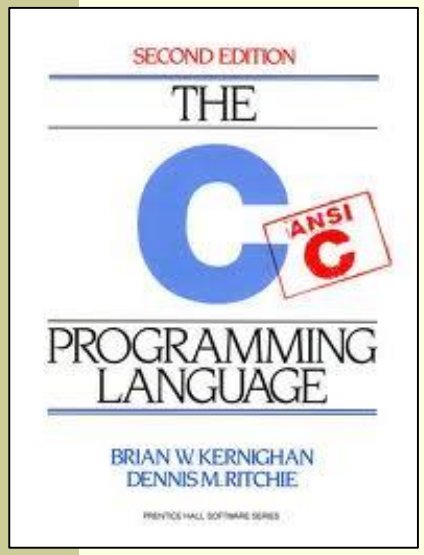
Instructor: **Joel Castellanos**

e-mail: joel@unm.edu

Web: http://cs.unm.edu/~joel/

11/7/2019

Chapter 7: Input and Output



fprintf()

w: write
r: read
a: append

```
void main(void)
{ FILE *outFilePt = fopen("myFile.txt", "w");
  if (outFilePt == NULL)
  { printf("Error opening file for write\n");
    return;
  }
  int n1 = 1;
  int n2 = 1;
  while (n2 < 55)
  { fprintf(outFilePt, "%d ",n2);
    int n3 = n1+n2;
    n1 = n2;
    n2 = n3;
  }
  fclose(outFilePt);  //do not forget to close the file.
}
```

myFile.txt:

```
1 2 3 5 8 13 21 34
```

fscanf()

```
void main(void)
{ int n1;
  FILE *inFilePt = fopen("myFile.txt", "r");
  if (inFilePt == NULL)
  { printf("Error opening file for read");
    return;
  }
  while (fscanf(inFilePt,"%d",&n1) != EOF)
  { printf("%d, ", n1);
  }
  fclose(inFilePt);
}
```

Output:

```
1, 2, 3, 5, 8, 13, 21, 34,
```

Reading String Data: Fast & Safe

```
#include <stdio.h>
void main(void)
{
  FILE *inFile = fopen("myDataFile.txt","r");
  int bufferSize = 1024;
  char data[bufferSize];

  //fgets guarantees NULL terminated
  while (fgets(data, bufferSize, inFile))
  { printf("[%s]\n", data);
  }
}
```

Writing Blocks of Binary Data: `fwrite`

```
size_t fwrite(const void *array,
  size_t size, size_t count, FILE *stream)
```

- The **`fwrite`** function writes a block of data to the stream.
- It will write an array of elements to the current position in the stream.
- For each element, it will write size bytes.
- The function will return the number of elements written successfully. The return value will be equal to count if the write completes successfully.

Binary File Writer: part 1 of 2

In C, the 2-D array: **a[***row***] [***column***]**

```
int a[2][3] = { {1, 2, 3}, {4, 5, 6} };
```

Is laid-out in linear memory as: **1 2 3 4 5 6**

```
#include <stdio.h>
void main(void)
{ int a[2][3] =
  { {1, 2, 3},
    {4, 5, 6}
  };
  writeBinaryArray(a, 2, 3);
}
```

Binary File Writer: part 2 of 2

```
void writeBinaryArray(int* array,
                      int width, int height)
{
  FILE *fp; //file pointer

  fp = fopen("array","wb");

  fwrite(&width, sizeof(int), 1, fp);
  fwrite(&height, sizeof(int), 1, fp);
  fwrite(array, sizeof(int),
     width*height, fp);

  fclose(fp);
}
```

"wb": write binary

What are the advantages / disadvantages of writing *binary files* verses *text files*?

BMP file format

- The BMP file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter).
- The BMP file format is capable of storing two-dimensional digital images both monochrome and color, in various color depths, and optionally with data compression, alpha channels, and color profiles.

BMP 24 bits/pixel, uncompressed

Bitmap file header	**14 bytes**	`'B','M'` **// magic number** `0,0,0,0` **// size in bytes** `0,0,0,0` **// reserved** `54,0,0,0` **// offset to start of pixel data**
DIB header	**40 bytes**	**see next slide**
Pixel array	**Variable -size**	**Each row in the Pixel array is padded to a multiple of 4 bytes in size.**

DIB header

```
40,0,0,0,   // DIB size
0,0,0,0,    // image width
0,0,0,0,    // image height
1,0,        // number color planes
24,0,       // bits per pixel
0,0,0,0,    // compression is none
0,0,0,0,    // image byte size
0x13,0x0B,0,0, // horz resolutions in pixel / m
0x13,0x0B,0,0, // vert resolutions
                        (0x03C3 = 96 dpi, 0x0B13 = 72 dpi)
0,0,0,0,    // #colors in palette
0,0,0,0,    // #important colors
```

Example: Sword_8x8.bmp

vim:

You can use xxd command to transform a file in vim to Hex representation:

```
:%!xxd
```

8x8 or 64 pixels

row= 8 pixels = 8*3 = 24 bytes. Since 24 % 4 = 0, no row padding is needed.

Total data: = 64*3 = 192 bytes + 54 header = 246 bytes +