Binary Files

Program Persistent Data

Lecture 4

Review

- In C there are buffers required to work with files.
- Streams are declared using **`FILE *fp;`**
- These streams are required for each file that you work on.
- To open and use the stream, error check that the file exists then close when finished:

```
fp = fopen("write.txt","w");
if (fp == NULL)
     {printf("Can't open file.\n");}
fclose(fp);
```

Review – *text* file <stdlib.h>

Instruction	Meaning
`fgetc(fp)`	Read a char from file using stream.
`fputc(fp)`	Write a char to file using stream.
`fgets(string,size,fp)`	Read a string from file using stream. It reads a string of a specified size.
`fputs(string,fp)`	Write a string to file using stream.
`fprintf(fp,"Hi %s, you are %i",s,a)`	Write the content to the file using stream.
`fscanf(fp,"%s %s %i",a,b,&c)`	Read a formatted line from file using stream.

#include <string.h>

Name	Example	Meaning
1. strlen()	`len=strlen(str);`	Get the length of a string
2. strcmp	`strcmp(str, "jane" )`	Compare 2 strings
3. strcpy	`strcpy(str, "jane");`	Copy a strings to another
4. strcat	`strcat( string, " Ferris" );`	Concatenate 2 strings
5. strstr	`Strstr(str, "jane")`	Look for a substring in a string

Binary Files

- Why are they important?
- Using Binary files.
- fread, fwrite, fseek, ftell & rewind
- Accessing data
  - Randomly
  - sequentially

Binary Files

- Stream of bytes
- It can handle any data type – not only text
- In order to use the data inside a binary file, the application needs to know how they are structured

Why use binary files?

- Faster
- Can be accessed randomly
- Can handle any data types
- More storage efficient!
- Text file cannot handle binary files!

The ASCII Table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

What's the difference between Text and Binary files?

1. Text File
   - Information (bits) are always interpreted as text

ASCII	'c'	'a'	't'
Hex	63	61	74
Binary	0110 0011	0110 0001	0111 1000

cat
dog
……

2. Binary Files
   - Information are interpreted as custom data (similar to arrays but varied)
   - Sequence of variable (called records!)

variable1	variable2	variable3	variable4	…	variableN

Storing numeric (123)

Storing numeric in text files:

- Done through ASCII characters.
  - 49, 50 & 51
- Each digit requires a store in memory.
- Can’t change the value easily re: memory

Storing numeric in binary files:

- Done through binary.
  - 01111011

What's in a C file?

Size of Variable (Ansi C)

Data Type	Size in Byte	
Char	1	One char
byte	1	Integer up to 256
int	2	Integer up to 65K
Long	4	Integer up to 4 billion
Double	4	Decimal numbers
String of n characters	n+1	One is the termination byte
struct	Sum of each variable	A struct is a record, a collection of data types describing an object

- To determine the size in the code before assigning resources = `sizeof(variableName)`

More differences of Binary & Text files

- It is possible to open a text file as binary
  - characters are bytes
- It is not possible to open a binary file as text
  - If you speak binary you may understand the content
- No EOF in binary files
  - A text file stops when reaches the EOF : a reserved special character (CTRL Z).
  - In a binary file EOF does not exist, it is just another byte.
  - Text files are best left as text as if a byte is equal to the special byte (EOF), the file won't be read completely

How to work with Binary files?

- To open the file use fopen as text files
- But add the letter “b”

```
FILE* fp = fopen(“filename.dat”, "rb");
```

rb: open for reading

wb: Truncate to zero length or create file for writing

ab: open or create file for writing at end-of-file

rb+: open for reading and writing, start at beginning

wb+: open for reading and writing (overwrite file)

ab+: open for reading and writing (append if file exists)

Block I/O

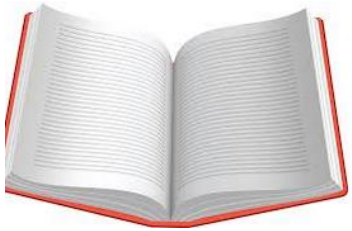
- We can move inside a binary file, not only sequential access but random access with `read()` and `write()`.
- The `fread()` and `fwrite()` function takes four parameters:
  1. A memory address via pointer
  2. Number of bytes to read per block
  3. Number of blocks to read //accommodate arrays
  4. A file variable pointer

fwrite()

fwrite(var, size, number, FILEpointer);

```
FILE* fp = fopen("sales.dat", "wb+");
char buffer[8]; //pointer (char array)
char string[]="string literal";
int i=100;
fwrite(string, sizeof(char), 8, fp);
fwrite(string, strlen(c)+1, 1, fp);//for /0
fwrite(c, 1, 8, fp); // write first 8 char (8 bytes)
fwrite (c, 1, 8, fp); // write next 8 char (8 bytes)
fwrite (&i, 1, 8, fp);
```

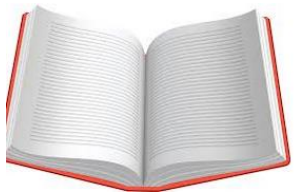
fread()



fread(memPointer, size, number, FILEpointer);

```
FILE* fp = fopen("sales.dat", "wb+");
char buffer[8];
char c[]="string literal";
double num[8];
fread(buffer, sizeof(char), 8, fp);
fread(buffer, strlen(c)+1, 1, fp);//for /0
fread(buffer, 1, 8, fp); // Read first 8 char (8 bytes)
fread(buffer, 1, 8, fp); // Read next 8 char (8 bytes)
```

//if the type was not Array or char then require pointer to the variable



```
#include <stdio.h>
#include <stdlib.h>
main ()
{
int i; //used to index - to read the array
int numArray[10];
FILE *fp;
```

```
fp=fopen("binary.dat", "rb");
fread(n, sizeof(int), 10, fp);
```

```
for (i=0;i<10; i++)
printf("numArray[%d] == %d\n", i, numArray[i]);
fclose (fp);
}
```

```
#include <stdio.h>
#include <stdlib.h>
main ()

{
int i; //to write the array
int j; //to read the array
int array []= {0,1,2,3,4,5,6,7,8,9};
FILE *fp;
if ((fp=fopen("file.dat", "wb")==NULL)
{     puts("Can't open that file!");
}//.dat are a windows file use .csv in linux based
i=fwrite(array, sizeof (int),10, fp);
i=fread(array, sizeof (int),10, fp);
```

```
for (j=0; j<10; j++)
printf("%d\n", array[j]);

fclose (fp);
}
```

fseek()

- For positioning in a file `fseek()` will move the file pointer.
- The purpose is to access the files as we do with pointers in arrays.
- Returns 0 if operation was correct and clears EOF

```
int fseek(FILE, offset in byte, whence);
```

whence is the position:

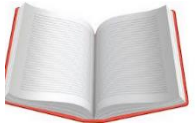
SEEK_SET – Default: Beginning of the file.
SEEK_CUR - Current location plus offset
SEEK_END - Set position to the end of the file

```
fseek(file,10,SEEK_CUR)
fseek(file,10*sizeof(double),SEEK_CUR)
```

```
#include <stdio.h>
#include <stdlib.h>
#include <stringh>
main ()
{
FILE *fp= fopen("file.txt","w+");
fputs("This is something", fp);

fseek(fp,8 , SEEK_SET);

fputs(" altered text!", fp);

fclose (fp);
}
```

```
/*copy text from one file to another using a buffer
variable */
main()
{
FILE * fpR =fopen("myfile.txt","rb");
FILE * fpW =fopen("myfileout.txt","wb");
int num;
double buffer[10];
```

```
fseek(file,10,SEEK_SET)  //go to the 10th byte
      while(feof(fpR)==0) // while not end of file
      {
```



```
        fread(buffer,sizeof(double),10,fpR);
      //read 100 variable double
        fwrite(buffer, sizeof(double),10,fpW);
      //write 100 variables double
      }
      fclose(fpR); fclose(fpW);   }
```

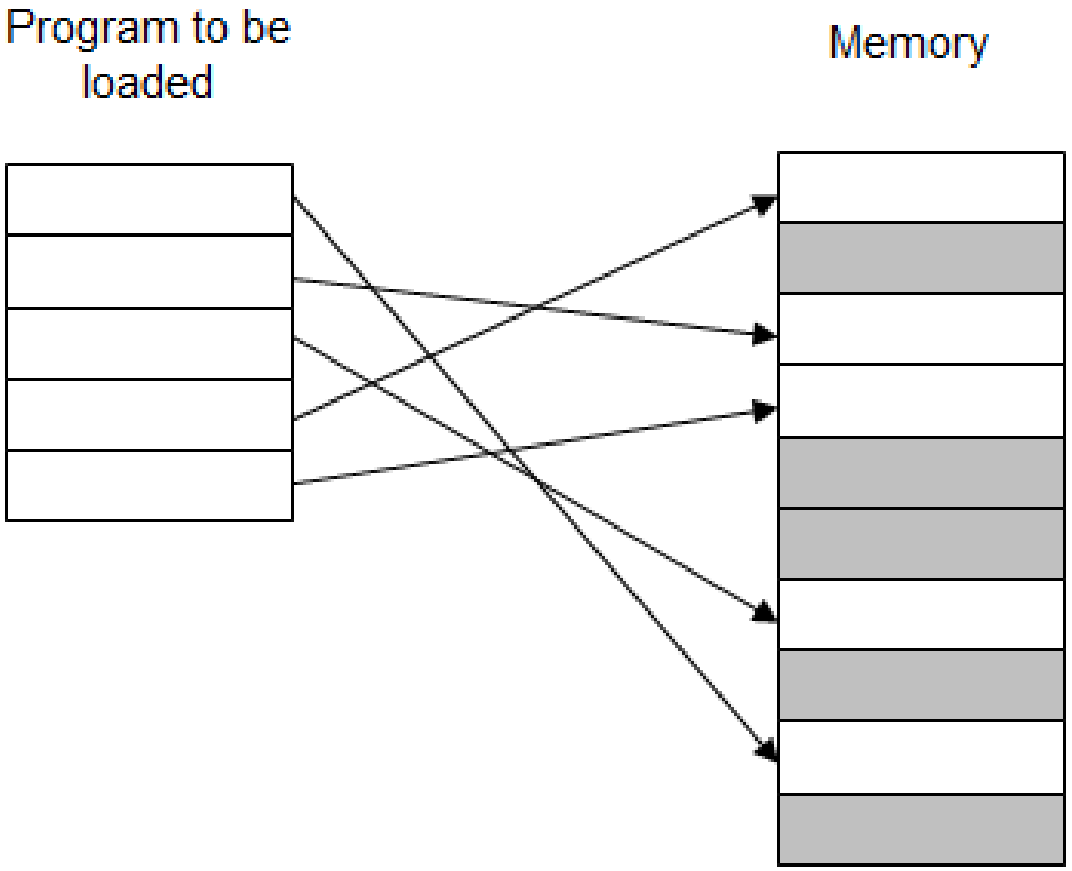
So we copied a file … again

- Many techniques achieve the same goal.
- What helps make the decision as to which is best suited?
- Maybe they are coded for specific file types?
- Maybe they are computationally quicker?
- Use the system to record time such as time in linux or

```
#include <time.h>
 begin = clock(); //start the clock

      //add your code here

 end = clock(); //stop the clock
```

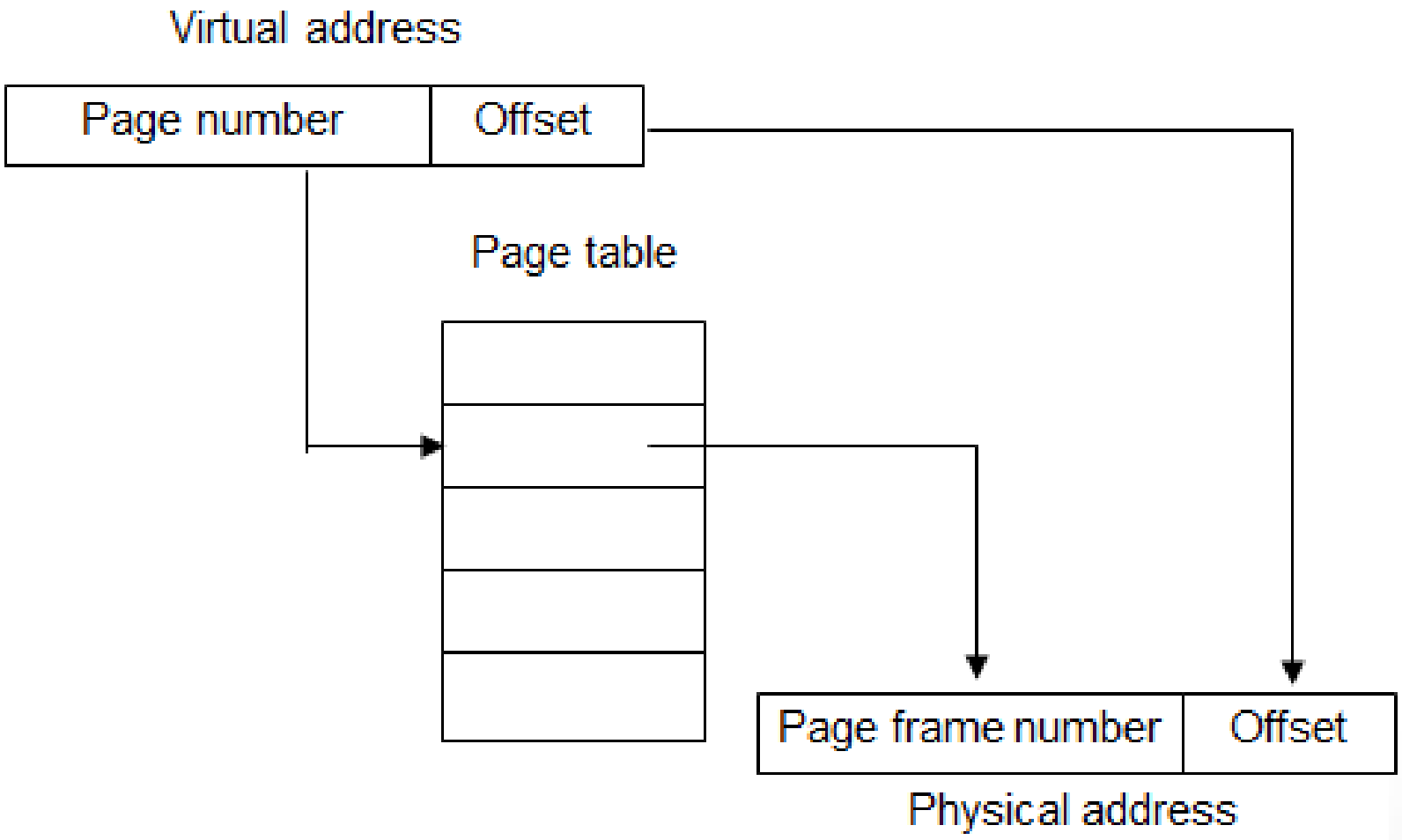
```
/*Computing Execution Time*/
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
main()
{
  clock_t begin, end;
  double time_spent;

  …
  begin = clock(); //start the clock
/* code that is to be analysed for computational time*/

  end = clock(); //stop the clock
time_spent = (double)(end - begin)
printf("%f",time_spent);
//if you want to print the time (in seconds), remember it is a
float

…
}
```

Virtual memory review



- If the unit of the *pages* are 4K (4K in binary is 0x1000) binary addresses are:
  - page 0 is 0x00000000 to 0x00000FFF
  - page 1 is 0x00001000 to 0x00001FFF
  - page 2 is 0x00002000 to 0x00002FFF
- Page number is first 20 bits (5 hex digits)
- last 12 bits (3 hex digits) give address within the page : *OFFSET*

Memory review



`ftell()`

- `ftell(FilePointer);`
- This reports the current position of the pointer.
- Will return the offset in memory of the next byte.

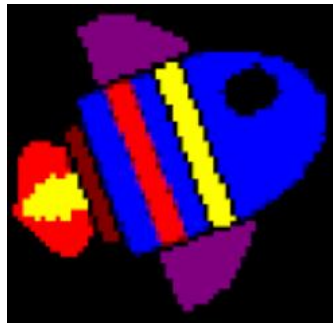
```
/*Using ftell to identify offset location */



main()
{
FILE * fp =fopen(“myfile.txt",“w");

fprintf(fp, “This is text”);

puts(“The file pointer is now at:”, ftell(fp));

fclose(fp);      }
```

rewind()

```
rewind(FilePointer);
```

- Rewind is useful, it places the pointer back to the start of the stream and clears the errno.
- This is a way of resetting a file without closing and re-opening

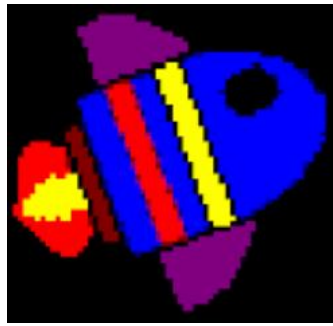
BitMaPs .bmps



- https://www.youtube.com/watch?v=1LZWCSKj45g

- BMP files are historic (but still commonly used) file format for Windows.
- BMP images can range from black and white (1 bit per pixel) up to 24 bit colour (16.7 million colours).
- While the images can be compressed, this is rarely used in practice.
- BMP file consists of a header, followed by an information section, if the image is indexed colour then the palette follows, and last of all is the pixel data.

Extracting the size of a BMP



- Bitmap is a standard uncompressed picture file format
- It is composed by:
  - A header (54 bytes) contains information on the file, such as type of format, dimensions, number of colors used…
  - A data section, containing information about each pixel. The color of the pixel is stored. In a 24-bit bmp, for each pixel 24 bits (3 bytes) are saved + 1 control byte.
  - The dimension of the picture is stored in two variable of type long (4 bytes) at position 18^{th} and 22^{nd}
  - Can you extract this information from a bmp file?

18 bytes	4 bytes (width)	4 bytes (heigth)	28 bytes	many bytes (pixels color data)

```
#include <stdlib.h>
#include <stdio.h>
main()
{
       FILE * fp;
       unsigned char header[54];
       long  width;
       long  heigth;

       if((fp=fopen("in.bmp","rb"))==NULL)
{ printf("open read file error.\n");    }
/* getting into the header */
       fseek(fp,18,SEEK_CUR);
       fread(&heigth,  sizeof(long),1, fp);
       fread(&width, sizeof(long),1, fp);
       printf("Length: %i , Heigth: %i \n",heigth,width);
       fclose(f);
}
```

Non text files

Name	Example	Meaning
fread	`fread(var, size, number, FILEpointer);`	Read from position in file
fwrite	`fwrite(var, size, number, FILEpointer);`	Write to position in file
fseek	**`fseek(FILE, offset in byte, whence);`** SEEK_SET/ _CUR or _END	Go to 1 of 3 places; start, current or end of file
ftell	`ftell(FilePointer);`	Where is the pointer now?
rewind	`rewind(FilePointer);`	Point to start if the file

Review

- Binary files are non text files that are more efficient than text files.
- However some systems don’t allow binaries and they are not interchangeable.