600.120 Intermediate Programming, Spring 2017*

Misha Kazhdan

**Much of the code in these examples is not commented because it would otherwise not fit on the slides. This is bad coding practice in general and you should not follow my lead on this.*

Outline

- Binary File I/O

Recall

When working with file handles we:

1. Create a file handle
2. Access the file's contents
3. Close the handle

main.c

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
        unsigned int values[100];
        for( int i=0  ;i<100 ; i++ ) values[i] = rand();
        FILE* fp = fopen( "foo.txt" , "w" );
        if( !fp ) return 1;
        for( int i=0 ; i<100 ; i++ ) fprintf( fp , "%u\n" , values[i] );
        fclose( fp );
}
```

Recall

When working with file handles we:

1. Create a file handle
   - `fopen` with the file-name and mode
     - "w" for (ASCII) write
     - "r" for (ASCII) read

main.c

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
        unsigned int values[100];
        for( int i=0  ;i<100 ; i++ ) values[i] = rand();
        FILE* fp = fopen( "foo.txt" , "w" );
        if( !fp ) return 1;
        for( int i=0 ; i<100 ; i++ ) fprintf( fp , "%u\n" , values[i] );
        fclose( fp );
}
```

Recall

When working with file handles we:

2. Access the file's contents
   - `fprintf` with file handle, format string, and values for (ASCII) write
   - `fscanf` with file handle, format string, and addresses for (ASCII read)

main.c

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
        unsigned int values[100];
        for( int i=0  ;i<100 ; i++ ) values[i] = rand();
        FILE* fp = fopen( "foo.txt" , "w" );
        if( !fp ) return 1;
        for( int i=0 ; i<100 ; i++ ) fprintf( fp , "%u\n" , values[i] );
        fclose( fp );
}
```

Recall

When working with file handles we:

3. Close the handle
   - `fclose` with file handle

main.c

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
        unsigned int values[100];
        for( int i=0  ;i<100 ; i++ ) values[i] = rand();
        FILE* fp = fopen( "foo.txt" , "w" );
        if( !fp ) return 1;
        for( int i=0 ; i<100 ; i++ ) fprintf( fp , "%u\n" , values[i] );
        fclose( fp );
}
```

Limitations

If we write out a list of 100 ints to a file

Q: How big would the file be?

Q: How would we get the 7th value?

main.c

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
        unsigned int values[100];
        for( int i=0  ;i<100 ; i++ ) values[i] = rand();
        FILE* fp = fopen( "foo.txt" , "w" );
        if( !fp ) return 1;
        for( int i=0 ; i<100 ; i++ ) fprintf( fp , "%u\n" , values[i] );
        fclose( fp );
}
```

Limitations

If we write out a list of 100 **int**s to a file

Q: How big would the file be?

A: 1056 bytes

We have 32 bits per **int**:

⇒ range of [0,4,000,000,000]

⇒ 9-10 decimal places (average)
+1 for the "**\n**"

⇒ $10 \times 100 - 11 \times 100$ bytes

⇒ Size is not fixed

But the values always require 400 bytes in memory!!!

```
>>./a.out
>> ls -l foo.txt
-rw-------. 1 misha users 1056 Mar 30 23:17 foo.txt
>>
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
        unsigned int values[100];
        for( int i=0  ;i<100 ; i++ ) values[i] = rand();
        FILE* fp = fopen( "foo.txt" , "w" );
        if( !fp ) return 1;
        for( int i=0 ; i<100 ; i++ ) fprintf( fp , "%u\n" , values[i] );
        fclose( fp );
}
```

Limitations

If we write out a list of 100 **int**s to a file

Q: How would we get the 7th value?

A: 64 characters in

We have 32 bits per **int**

⇒ $10 \times 6 - 11 \times 6$ bytes

⇒ Offset is not fixed

✗ We cannot jump to the 7th value since we don't know the sizes of the values that come before

⇒ **fscanf** one **int** at a time until we get the 7th value

```
>>./a.out
>> more foo.txt
1804289383
846930886
1681692777
1714636915
1957747793
424238335
719885386
...
>>
```

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
        unsigned int values[100];
        for( int i=0  ;i<100 ; i++ ) values[i] = rand();
        FILE* fp = fopen( "foo.txt" , "w" );
        if( !fp ) return 1;
        for( int i=0 ; i<100 ; i++ ) fprintf( fp , "%u\n" , values[i] );
        fclose( fp );
}
```

Binary File I/O

- Until now, all files we've accessed in C have been plain *text files*
  - Write: Convert everything to a string of characters that is written to the file.
  - Read: Convert everything from a string of characters that is read from the file

- Non-text files are known as *binary files* in C
  - Write: perform a bit-by-bit copy from memory to the file
  - Read: perform a bit-by-bit copy from the file to memory

Binary File I/O

As with text files, we:

1. Open the file
2. Access the file's contents
3. Close the file

main.c

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
        unsigned int values[100];
        for( int i=0 ; i<100 ; i++ ) values[i] = rand();
        FILE* fp = fopen( "foo.dat" , "wb" );
        if( !fp ) return 1;
        fwrite( values , sizeof(int) , 100 , fp );
        fclose( fp );
}
```

Binary File I/O

```
FILE* fopen( const char* fileName , const char* mode );
```

1. Open the file
   - Use `fopen` to create a file handle:
     - `fileName`: name of the file
     - `mode`: mode of I/O
       - To open a file in binary mode, add the "b" flag in the string of `mode` characters*
     - Returns a file pointer (or NULL if the `fopen` failed)

main.c

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
        unsigned int values[100];
        for( int i=0 ; i<100 ; i++ ) values[i] = rand();
        FILE* fp = fopen( "foo.dat" , "wb" );
        if( !fp ) return 1;
        fwrite( values , sizeof(int) , 100 , fp );
        fclose( fp );
}
```

*The file extension does not affect how the file is opened.

Binary File I/O

```
size_t fwrite( const void* ptr , size_t sz , size_t count , FILE* fp );
```

2. Access the file's contents
   - Use `fwrite` to write to a binary file:
     - `ptr`: starting address of data to write out
     - `sz`: size of a data element
     - `count`: number of a data element
     - `fp`: file handle to write to
     - Returns the number of elements written

main.c

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
        unsigned int values[100];
        for( int i=0 ; i<100 ; i++ ) values[i] = rand();
        FILE* fp = fopen( "foo.dat" , "wb" );
        if( !fp ) return 1;
        fwrite( values , sizeof(int) , 100 , fp );
        fclose( fp );
}
```

Binary File I/O

```
size_t fread( void* ptr , size_t sz , size_t count , FILE* fp );
```

2. Access the file's contents
   - Use `fread` to read from a binary file:
     - `ptr`: starting address of data to read into
     - `sz`: size of a data element
     - `count`: number of a data element
     - `fp`: file handle to read from
     - Returns the number of elements read

main.c

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
        unsigned int values[100];
        for( int i=0 ; i<100 ; i++ ) values[i] = rand();
        FILE* fp = fopen( "foo.dat" , "wb" );
        if( !fp ) return 1;
        fwrite( values , sizeof(int) , 100 , fp );
        fclose( fp );
}
```

Binary File I/O

```
int fclose( FILE* fp );
```

3. Close the file
   - Use `fclose` to close the file handle
     - `fp`: file handle
     - Returns 0 if the stream was closed

main.c

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
        unsigned int values[100];
        for( int i=0 ; i<100 ; i++ ) values[i] = rand();
        FILE* fp = fopen( "foo.dat" , "wb" );
        if( !fp ) return 1;
        fwrite( values , sizeof(int) , 100 , fp );
        fclose( fp );
}
```

Binary File I/O

If we write out a list of 100 ints to a file

Q: How big would the file be?

A: 400 bytes. Always!

```
>>./a.out
>> ls -l foo.dat
-rw-------. 1 misha users 400 Mar 30 23:17 foo.dat
>>
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
        unsigned int values[100];
        for( int i=0 ; i<100 ; i++ ) values[i] = rand();
        FILE* fp = fopen( "foo.dat" , "wb" );
        if( !fp ) return 1;
        fwrite( values , sizeof(int) , 100 , fp );
        fclose( fp );
}
```

Binary File I/O

As we read/write data, the `FILE` pointer tracks our position in the file:

- In some cases, we would like to change our position in the file:
  - Writing: To over-write something that was previously written
  - Reading: To jump to where the data we are interested in resides

Binary File I/O

```
int fseek( FILE* fp , long int offset , int whence );
```

- Use `fseek` to change the position of the file pointer
  - `fp`: file pointer
  - `offset`: number of bytes to move
    - Could be positive or negative, depending on whether we move forward or back
  - `whence`: where we move from:
    - `SEEK_SET`: beginning of the file
    - `SEEK_CUR`: current position
    - `SEEK_END`: end of the file
  - Returns zero if the change succeeded

Binary File I/O

main.c (part 1)

```
#include <stdio.h>
#include <stdlib.h>
unsigned int getValue( FILE* fp , size_t idx )
{
	if( fseek( fp , sizeof(unsigned int)*idx, SEEK_SET ) )
	{
		fprintf( stderr , "Failed to seek\n" );
		return -1;
	}
	unsigned int v;
	if( fread( &v , sizeof( int ) , 1 , fp )!=1 )
	{
		fprintf( stderr , "Failed to read\n" );
		return -1;
	}
	return v;
}
```

main.c (part 2)

```
int main( void )
{
	FILE* fp = fopen( "foo.dat" , "rb" );
	if( !fp )
	{
		fprintf( stderr , "Failed to open\n" );
		 return -1;
	}
	printf( "%u\n" , getValue( fp , 7 ) );
	fclose( fp );
}
```

Binary File I/O

```
size_t fwrite( const void* ptr , size_t sz , size_t count , FILE* fp );
size_t fread( void* ptr , size_t sz , size_t count , FILE* fp );
```

Note:

- The fread/fwrite functions only need to be able to read the bits/bytes from memory, they don't need to know the data-type stored.

⇒ We are not limited to reading/writing integers and numbers

main.c

```
#include <stdio.h>
typedef struct{ ... } MyStruct;
int main( void )
{
	unsigned MyStruct values[100];
	FILE* fp = fopen( "foo.dat" , "rb" );
	if( !fp ) return 1;
	fread( values , sizeof(MyStruct) , 100 , fp );
	fclose( fp );
}
```

Binary File I/O

```
size_t fwrite( const void* ptr , size_t sz , size_t count , FILE* fp );
size_t fread( void* ptr , size_t sz , size_t count , FILE* fp );
```

Note:
If the **struct** contains pointers, the address is written out, not the contents at the address!!!

- The **fread/fwrite** functions only need to be able to read the bits/bytes from memory, they don't need to know the data-type stored.

⇒ We are not limited to reading/writing integers and numbers

main.c

```
#include <stdio.h>
typedef struct{ ... } MyStruct;
int main( void )
{
        unsigned MyStruct values[100];
        FILE* fp = fopen( "foo.dat" , "rb" );
        if( !fp ) return 1;
        fread( values , sizeof(MyStruct) , 100 , fp );
        fclose( fp );
}
```

Piazza → Resources section → Resources tab → Exercise 8-1