

CSc 352

Bit Manipulation

Benjamin Dicken

Bit Operations

C supports a number of operations to manipulate the ones and zeros in memory

Shifting: >> <<

Masking: & | ^

Flipping: ~

Shifting

```
uint8_t x = 21;           // x = 00010101
x = x << 2;               // x = 01010100
x = x >> 4;               // x = 00000101
printf("%u\n", x);       // what does it print?
```

Shifting

```
uint8_t y;  
y = 1;      // 00000001  
for (int i = 0; i < 8; i++) {  
    y = y<<1;  
    printf("%u\n", y);  
}
```

Viewing bits on stdout

Implement the function

```
void print_bits(uint8_t data);
```

Should print out the 1s and 0s stored in **data** to standard output

For example:

```
uint8_t x = 4;  
print_bits(x); // Should print 00000100
```

```
void print_bits(uint8_t data){
    for(int j = 0; j < 8; j++){
        uint8_t temp = data;
        temp = temp<<(7-j);
        temp = temp>>7;
        printf("%u", temp != 0 ? 1 : 0);
    }
    printf("\n");
}
```

Viewing bits on stdout

Implement the function

```
void print_bits(uint8_t * data, int size);
```

Should print out the 1s and 0s stored in the array of length **size** that **data** points to

For example:

```
uint16_t x = 4;  
print_bits(x, 2); // Should print 00100000 00000000
```

```
void print_bits(uint8_t * data, int size){
    uint8_t* copy = malloc(size);
    memcpy(copy, data, size);
    for(int i = 0; i < size; i++){
        for(int j = 0; j < 8; j++){
            uint8_t temp = copy[i];
            temp = temp<<(7-j);
            temp = temp>>7;
            printf("%u", temp != 0 ? 1 : 0);
        }
        printf(" ");
    }
    printf("\n");
    free(copy);
}
```

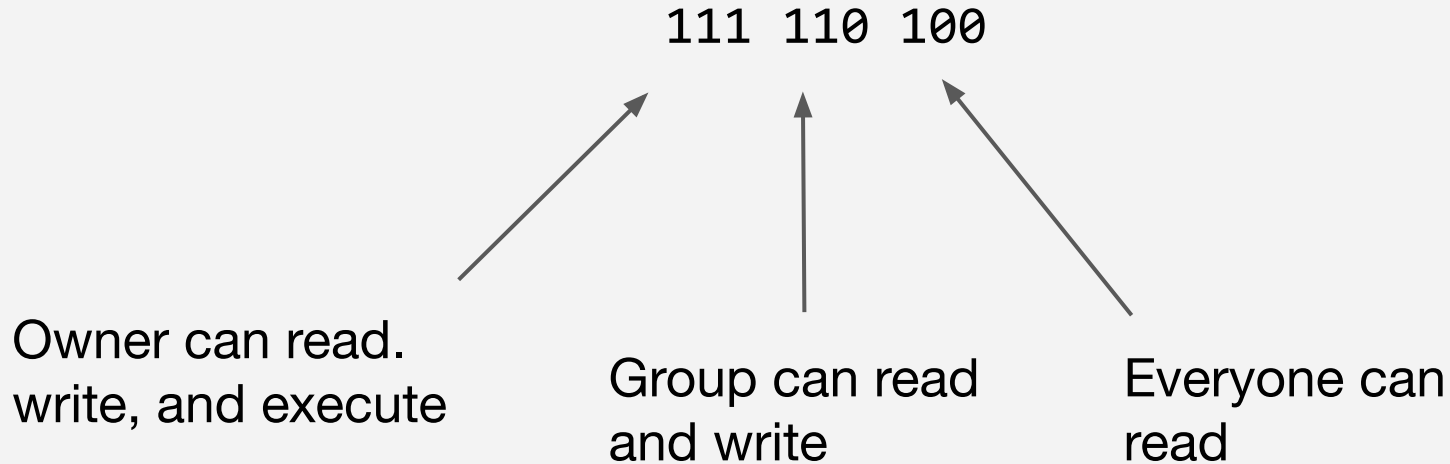

xxdb

Implement a program: **xxdb.c**

- Similar to `$ xxd -b`
- Gets file name from command line
- Prints out 8 bytes per line, keeps track of byte offset
- Does not need to print out ascii values
- Use `print_bits` function

Permissions

Recall that permissions for files can be represented as a binary sequence:



Permissions

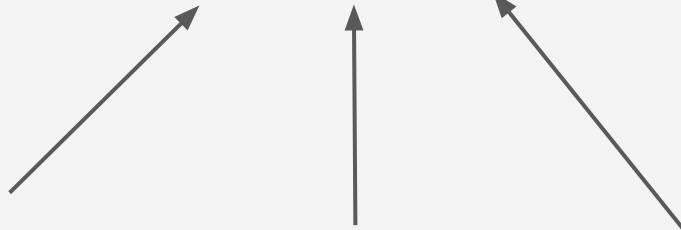
Could represent this with a `uint16_t`

0000000 111 110 100

Owner can read.
write, and execute

Group can read
and write

Everyone can
read



Permissions

Implement the function

```
uint16_t owner_permissions(uint16_t * permissions);
```

Should take the Owner permissions and set those same permissions as the group and every permissions too, return the number

For example:

```
uint16_t x = 272; // 000000000 100 010 000  
owner_permissions(x); // Should return 000000000 100 100 100
```

Setting File Permissions

Set the file permission using **chmod** function and **mode_t**

Use **<sys/stat.h>** and **<sys/types.h>**

```
$ man 2 chmod
```

```
$ man 7 inode
```