Lab 3

Working with Masks

Using bitwise AND, OR, XOR and NOT operators.
Designing masks for specific purposes.
Understanding Linux file permissions.

In this lab, you will explore **number representation** and **bitwise operators** (masks) using a simple Java program. You will also gain hands-on experience with the Linux `chmod` command, observing its role in managing file permissions.

Setting up the Java Program

1. Create a sub-directory in your home directory called **lab3** and set it as your current working directory.
2. Display the contents of the directory:

   `/library/students/comp2531/w2025/lab3`

A Teaching Moment!

The unadorned `ls` command only displays the names of the files in a directory. But what if you need more details, such as the file's owner, access rights, or last modification date? For this, you can use the 'long listing' option of `ls` by running:

```
> ls -l
```

To discover all the options for a command, you can refer to its manual page using: `man <command>`. For example, to view all options for `ls`, use:

```
> man ls
```

Interpreting the Long Listing Output

When using `ls -l`, the directory listing will display information in a specific format. From right to left, the details include:

- The name of the file.
- The last modification date and time of the file.
- The size of the file in bytes.
- The owner and group of the file. For example, `adadlani` as the owner and `faculty` as the group. Your files will typically show `students` as the group.
- A number indicating the number of files in the directory (if it is a directory) or `1` (one) for regular files.
- A 10-character file permission string. The first character is `d` for directories or – for regular files. Other possibilities exist but are beyond the scope of this discussion. The next 9 characters denote permissions in three groups of three for the *owner*, *group*, and *others*, respectively.

Revised: January 27, 2025

3. Copy the file `Masks.java` from the above directory to your **lab3** directory. Take note of the ownership and permissions of the copied file.
4. Examine the file `Masks.java`. It is a simple Java program that uses only a `main` method, which is typically not the best practice for anything beyond basic demonstrations like this one.
5. Compile the program by running the following command:

   ```
   > javac Masks.java
   ```

6. Run the program using:

   ```
   > java Masks
   ```

   This program takes three numbers as input and produces a variety of outputs:

   - First, enter the base for the numbers you wish to input. Common bases include 10 (decimal), 2 (binary), 16 (hexadecimal), and 8 (octal). However, you can also experiment with other bases (e.g., 3 or 21), which the program should handle.
   - Next, enter two numbers in the base you specified.
   - The program will display the numbers and the results of the `NOT`, `AND`, `OR`, and `XOR` operations in binary, octal, hexadecimal, and decimal formats.

7. Feel free to modify the program to explore additional features or try out other functionalities.

EXERCISES

I. Masking:

Answer the following questions:

1. Create 8-bit binary masks for the following situations. (*Hint:* Refer to this page: `http://en.wikipedia.org/wiki/Mask_(computing)`.) For each situation, provide an example to demonstrate that your solution works.
   (a) Create a mask and suggest a bitwise operator that can be used to turn the 2nd, 4th, 6th, and 7th bits **on** (i.e., set to 1) in a resulting number when applying the mask and operator to any binary number of fixed length 8 bits.
   (b) Create a mask and suggest a bitwise operator that can be used to turn the 3rd, 5th, and 7th bits **off** (i.e., set to 0) in a resulting number when applying the mask and operator to any binary number of fixed length 8 bits.

(c) Create a mask that can be used to determine if the 7th bit in a binary number is set to **1**.

2. Using a Bit Mask:

   Recall the `ls -l` command described earlier, which displays file permissions as a 10-character string like `-rw-rw----`. How can you change these permissions? The command is `chmod`, which takes two arguments: a three-digit octal number and a file name. For example:

   ```
   > chmod 432 Masks.java
   ```

   The last nine characters of the string represent a bit string, where each – is a 0 and each permission (e.g., `r`, `w`, `x`) is a 1. For example:

   ```
   r---wx-w-
   100011010
   ```

   To convert this binary string into an octal number:

   - Group the bits into sets of three, from left to right.
   - Write the corresponding octal number for each group.

   In this example, the binary string `100011010` converts to the octal value `432`. The three octal digits set permissions as follows:

   - The first digit sets the permissions for the **owner**.
   - The second digit sets the permissions for the **group**.
   - The third digit sets the permissions for **others**.

   Typical modes would be:

   - `660`: Grants read and write permissions to the owner and group, but no permissions to others.
   - `664`: Grants read and write permissions to the owner and group, and read permissions to others.

   Try changing the permissions on your `Masks.java` file using `chmod` with various octal strings. Verify the effect of each change using the `ls -l` command.

II. Number Representation

For the following questions, assume a fixed length of 10-bit binary numbers.

1. How many distinct numbers can be represented?
2. What is the largest representable unsigned integer? Write your result in **binary**.

3. What is the largest representable unsigned integer? Write your result in **hexadecimal**.
4. What is the largest representable unsigned integer? Write your result in **octal**.
5. What is the largest representable unsigned integer? Write your result in **decimal**.
6. What is the smallest representable unsigned integer? Write your result in **binary**.
7. What is the smallest representable unsigned integer? Write your result in **hexadecimal**.
8. What is the smallest representable unsigned integer? Write your result in **octal**.
9. What is the smallest representable unsigned integer? Write your result in **decimal**.

III. Addition

1. Add the following two hexadecimal numbers together: $3EE_{16} + 21_{16}$. Assume a fixed length of 10 bits. Show your final answer in **hexadecimal**.
2. Add $3E4_{16} + E24_{16}$. Assume a fixed length of 12 bits. Does this result in an unsigned overflow? Explain your answer.

IV. Logical Operators and Subtraction

For each pair of numbers below, perform the following operations:

- Subtraction.
- Logical bitwise **AND**.
- Logical bitwise **OR**.
- Logical bitwise **XOR**.

1. $3F_{16}$ and 11_{16}
2. $1B_{16}$ and 16_{16}
3. 10001_{16} and 0101_{16}
4. 777_{16} and 14_{16}
5. $11F_{16}$ and $E2_{16}$