Agenda
- Bitwise Operators (Recap) + Example
- Type casting
- Recursion + Factorial Example
- GDB (GNU Debugger)
- Arrays + Maze Example
# Bitwise Operators (Recap)

<table>
<thead>
<tr>
<th>AND</th>
<th>OR</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a &amp; b</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
## Bitwise Operators (Recap)

### Left Shift

<table>
<thead>
<tr>
<th>X</th>
<th>0 0 1 0 1 1 0 1 1 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>X &lt;&lt; 1</td>
<td>0 1 0 1 1 0 1 0 1 0</td>
</tr>
<tr>
<td>X &lt;&lt; 2</td>
<td>1 0 1 1 0 1 0 0 0</td>
</tr>
</tbody>
</table>

### Right Shift

<table>
<thead>
<tr>
<th>X</th>
<th>1 0 1 1 0 1 0 0 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>X &gt;&gt; 1</td>
<td>0 1 0 1 1 0 1 0 1 0</td>
</tr>
<tr>
<td>X &gt;&gt; 2</td>
<td>0 0 1 0 1 1 0 1 0 1</td>
</tr>
</tbody>
</table>
LSB Replacement Example

Write a C expression that will yield a word consisting of the least significant byte of x, and the remaining bytes of y.

For example:

\[
\text{x = 0x89ABCDEF and y = 0x76543210, this would give 0x765432EF.}
\]

Note: You can’t use constants larger than 255
Typecasting

- **Implicit**
  - Automatically performed when a value is copied to a compatible type
  - Promotion from a lower size type to a higher size one:
    - `short/char → int → unsigned int → long → unsigned long → long long → unsigned long long → float → double → long double`

```
short x = 255;
double p = x * 2.0;
```
Typecasting

- **Explicit**
  - Specified by the programmer, especially when a different interpretation is needed
  - `(Type)Expr2Cast

```java
int x = 0xABCD;
short y = (short)x;
float f = (float)x;
```
Recursion

- **A function calling itself**
  - Each instance has a copy of local variables for itself
  - FILO: First call returns last
  - Useful for divide and conquer
    - Break a big problem into smaller ones that are solvable
    - Merge the solutions
  - Example: We know that $n! = n \times (n-1)!$, $1! = 1$

```
First Call

fact(5) \rightarrow calls \rightarrow fact(4) \rightarrow calls \rightarrow fact(3) \rightarrow calls \rightarrow fact(2) \rightarrow calls \rightarrow fact(1) \rightarrow calls

Returns 5*fact(4)
Returns 4*fact(3)
Returns 3*fact(2)
Returns 2*fact(1)
Returns 1 (Trivial Sol.)
```
GNU Debugger

Compile it with -g switch
- $ gcc -o gdbtest gdbtest.c -g
  -g will instruct the compiler to embed debugging information into the executable

Attach gdb to the program with bugs
- $ gdb ./myexec

Display source file
- (gdb) list

Find the line numbers and set breakpoints
- (gdb) break [SourceFileName]:[line-number]

Run the program
- (gdb) run

Display values of variables
- (gdb) display/print [variable name]

Step into/over
- (gdb) step/next

Quit from GDB
- (gdb) quit
Arrays

• **Definition**
  
  • A number of variables with same type and name, but indexed!

  • Consecutive storage in memory

```c
Type name[SIZE]; // 1D w/o initialization
Type name[] = {e_0, e_1, ..., e_{n-1}}; // 1D with initialization
Type name[M][N]; // 2D w/o initialization
Type name[][] = { {e_{0,0}, ..., e_{0,N-1}}, // 2D with initialization
  .
  .
  {e_{M-1,0},...,e_{M-1,N-1}}
};
char myStr[] = "Hello"; // equal to {'H','e','l','l','o','\0'}
```
Arrays

• **Access (R/W)**
  • By means of index (from 0 to SIZE-1)

• **Passing to functions**
  • By reference (No private copy for callee)
  • Size of dimensions should be written (except for 1st dim)

```c
void printList(int a[], int n){
    int i;
    for (i = 0 ; i < n ; i++)
        printf("%d ", a[i]);
}
void printMatrix(int m[][10], int n1, int n2){
    ...
}
```
More challenging example

• **Problem**
  
  Given a matrix of zeros and ones where ones correspond to a path in a maze, find out how many possible paths exist from point (0,0) to (M-1, N-1)

• **Solution**
  
  Start from (0,0)
  
  Look for possible adjacent cells which are equal to 1 and move to the new cell
  
  Call the function with the new position (recursion)
  
  If reached (M-1, N-1) return 1 to indicate a path has been found