

# PROJECT :: Inode Inspector

## Start Assignment

- Due May 10 by 9pm
- Points 100
- Submitting a file upload
- File Types c
- Available Apr 25 at 12am - May 10 at 11pm

## Project Overview: Inode Inspector

**Objective:** Develop the "inspect" command-line tool that provides detailed inode information for specified files and directories, offering both detailed and human-readable outputs.

## Background:

In Unix-like operating systems, each file and directory is associated with an inode, which stores metadata about the filesystem objects. This metadata includes details such as inode number, file type, permissions, link count, owner (UID), group (GID), size, timestamps (last access, modification, and status change), and pointers to the disk blocks that store the file's contents.

The Inode Inspector aims to give students practical experience with filesystem internals, command-line interface development, and effective error handling.

## Goals:

1. Learn to access file metadata using system calls in C.
2. Understand inode properties and their roles in the filesystem.
3. Develop a functional CLI tool for real-world application.

## Requirements:

### 1. Core Functionality:

- Retrieve and display inode information for specified files.
- Output information in both detailed and human-readable formats.
- Support multiple output formats: JSON and plain text.
- Implement a command-line interface with comprehensive options for user interaction.

### 2. Command-Line Options:

- `-?, --help`: Display help information about the tool and its options.
  - Example: `inspect -?`

- Note that this is also the default behavior if the user types the command `inspect` without any arguments.
- `-i, --inode <file_path>`: Display detailed inode information for the specified file.
  - Example: `inspect -i /path/to/file`
  - Note that this flag is optional and the default behavior is identical if the flag is omitted.
- `-a, --all [directory_path]`: Display inode information for all files within the specified directory. If no path is provided, default to the current directory.
  - Optional flag: `-r, --recursive` for recursive listing.
  - Example: `inspect -a /path/to/directory -r`
- `-h, --human`: Output all sizes in kilobytes (K), megabytes (M), or gigabytes (G) and all dates in a human-readable form.
  - Example: `inspect -i /path/to/file -h`
- `-f, --format [text|json]`: Specify the output format. If not specified, default to plain text.
  - Example: `inspect -i /path/to/file -f json`
- `-l, --log <log_file>`: Log operations to a specified file.
  - Example: `inspect -i /path/to/file -l /path/to/logfile`

## 1. Error Handling:

- The tool must handle and report errors effectively, such as invalid file paths, insufficient permissions, and unsupported file types.

## Starting Point Example in C:

Here's a simple example using C to retrieve and display inode information for a single file. This example uses the `stat` system call.

```
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>

int main(int argc, char *argv[]) {
    struct stat fileInfo;
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <file_path>\n", argv[0]);
        return 1;
    }

    if (stat(argv[1], &fileInfo) != 0) {
        fprintf(stderr, "Error getting file info for %s: %s\n", argv[1], strerror(errno));
        return 1;
    }

    printf("Information for %s:\n", argv[1]);
    printf("File Inode: %lu\n", fileInfo.st_ino);
    printf("File Type: ");
    if (S_ISREG(fileInfo.st_mode))
        printf("regular file\n");
    else if (S_ISDIR(fileInfo.st_mode))
        printf("directory\n");
    else if (S_ISCHR(fileInfo.st_mode))
        printf("character device\n");
    else if (S_ISBLK(fileInfo.st_mode))
        printf("block device\n");
    else if (S_ISFIFO(fileInfo.st_mode))
        printf("fifo\n");
    else if (S_ISLNK(fileInfo.st_mode))
        printf("symlink\n");
    else
        printf("unknown\n");
}
```

```
    printf("character device\n");
else if (S_ISBLK(fileInfo.st_mode))
    printf("block device\n");
else if (S_ISFIFO(fileInfo.st_mode))
    printf("FIFO (named pipe)\n");
else if (S_ISLNK(fileInfo.st_mode))
    printf("symbolic link\n");
else if (S_ISSOCK(fileInfo.st_mode))
    printf("socket\n");
else
    printf("unknown?\n");

printf("Number of Hard Links: %lu\n", fileInfo.st_nlink);
printf("File Size: %lu bytes\n", fileInfo.st_size);
printf("Last Access Time: %ld\n", fileInfo.st_atime);
printf("Last Modification Time: %ld\n", fileInfo.st_mtime);
printf("Last Status Change Time: %ld\n", fileInfo.st_ctime);

return 0;
}
```

## Instructions:

- Use the example as a starting point to build the full functionality.
- Pay close attention to error handling and user feedback.
- Test the tool extensively to ensure it handles various edge cases and file types.

## JSON Output Schema

The JSON output should be structured as a dictionary where each key corresponds to a specific piece of inode information. This will ensure that the output is both machine-readable and easily accessible for grading scripts.

```
{
  "filePath": "/path/to/file",
  "inode": {
    "number": 123456,
    "type": "regular file",
    "permissions": "rw-r--r--",
    "linkCount": 3,
    "uid": 1000,
    "gid": 1000,
    "size": "4K",
    "accessTime": "YYYY-MM-DD HH:MM:SS",
    "modificationTime": "YYYY-MM-DD HH:MM:SS",
    "statusChangeTime": "YYYY-MM-DD HH:MM:SS"
  }
}
```

## Key Descriptions:

- **filePath**: The path to the file for which the inode information is provided.
- **inode**:
  - **number**: The inode number of the file.

- **type**: The type of file (e.g., "regular file", "directory", "symbolic link").
- **permissions**: The file permissions in symbolic format (e.g., "rw-r--r--").
- **linkCount**: The number of hard links pointing to the file.
- **uid**: The user ID of the file's owner.
- **gid**: The group ID of the file's owner.
- **size**: The size of the file, optionally formatted as human-readable (e.g., "4K", "2M").
- **accessTime**, **modificationTime**, **statusChangeTime**: Timestamps of the last access, modification, and status change, either in **Unix epoch time format** [↗\(https://www.epochconverter.com/\)](https://www.epochconverter.com/) or, optionally, formatted in a standard human readable format as shown above.

*Note that both the dates and sizes shown in the example example above are in human readable format. The -h flag works on both console and JSON output.*

## Usage in Command-Line Tool:

To produce JSON output in this format, the tool would include an option like **-f json** to specify the output format.

```
inspect -i /path/to/file -h -f json
```

As part of this project, you will need to develop routines to output JSON formatted data based on the inode information of files. This output will be in a linear format, as outlined in the provided JSON dictionary format. The goal here is to ensure that you can generate clean, well-structured JSON without relying on external libraries.


## JSON Output Requirements:


Your task is to manually create JSON output that captures inode information. This JSON data should be structured as a series of dictionaries (one for each file or directory inspected), each containing relevant inode data. Each dictionary should be printed as a separate JSON object within an array.

## Steps to Manually Write JSON:

1. **Begin the JSON Array**: Start your output with an opening square bracket **[** to signify the start of an array of JSON objects.
2. **Generate JSON for Each Inode**: For each file or directory:
  - Open a JSON object with **{**.
  - Include key-value pairs within the object. Each key should be a string (enclosed in quotes), and the corresponding value should be formatted according to its type (strings in quotes, numbers without).
  - Make sure to correctly place commas between key-value pairs and check that special characters in strings (like quotes and backslashes) are properly escaped.

- Close each JSON object with .

3. **Handle Commas Between Objects:** Ensure that there is a comma  between each JSON object, except after the last one in the array.

4. **Close the JSON Array:** End your output with a closing square bracket  to signify the end of the array.

## Tips for Success:

- **Test Thoroughly:** Make sure you thoroughly test your JSON output for various file types and scenarios to ensure it's always valid JSON.
- **Simplify Your Code:** Aim for simplicity in your functions. Overcomplicating your code can lead to errors and harder-to-maintain scripts.
- **Keep Performance in Mind:** Although the focus is on correctness, also consider the efficiency of your implementation, especially if you're processing large directories.

This task is designed to improve your understanding of JSON formatting and give you practical experience in generating structured output data, which are valuable skills in many software development scenarios. Make sure to follow the structure carefully to ensure your output meets the project specifications and can be easily read and graded.

## Testing and Evaluation of Your Tool

### JSON Output Testing

The primary functionality of your **Inode Inspector** tool will be tested through the JSON output it generates. This structured output allows for automated grading to assess whether your tool correctly gathers and displays inode information as specified in the project requirements. Make sure that your JSON output adheres precisely to the format outlined, as this will be crucial for passing the automated tests.

### Console Output Evaluation

In addition to the automated testing of JSON output, your console output will also be manually reviewed. This part of the evaluation will focus on the neatness, uniformity, and visual appeal of the information presented in the console. Here are key aspects to consider:

- **Neatness:** Ensure that the output on the console is well-organized and easy to read. Avoid cluttered or chaotic information presentation, which can make it difficult to quickly understand the output.
- **Uniformity:** Your output should be consistent across different runs of the program. This includes consistent use of formatting, spacing, and styling in the information presented. Uniform output not only looks more professional but also makes it easier to compare results from different files or directories.

- **Visual Appeal:** While functionality is critical, the visual appeal of your output can also impact the usability of your tool. Consider the alignment of text, the use of whitespace, and even the inclusion of color (if appropriate) to enhance readability and user experience.

## Recommendations for Achieving High Quality Output

- Use formatting functions or techniques in C that allow you to control output layout precisely, such as `printf` with formatting options.
- Test your output on different terminals and with various file paths to ensure it remains consistent and readable under different conditions.
- Consider peer reviews or feedback sessions to gather insights on how others perceive the usability and appearance of your output.

By focusing on both the accuracy of the JSON output and the quality of the console presentation, you can ensure that your **Inode Inspector** tool is robust, user-friendly, and meets the grading criteria effectively.

## Naming Your Source File

Ensure your source file is named `inspect.c`.

## Compiling Your Source File

Use the following command in a terminal to compile your program:

```
gcc -o inspect inspect.c
```

You will submit the single source file `inspect.c`

## Release History

You are always responsible for the latest release. The release may be updated at any time to fix bugs or add clarification. Get used to it, programmers are expected to adapt to changing requirements.

**V002 : Updated the default behavior when no flag is specified.**

V001 : Initial release, expect a few bugs.