National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Faculty of Computing

SE-314: Software Construction

Class: BESE 13AB

Lab 07: Recursion

CLO-03: Design and develop solutions based on Software Construction principles.
CLO-04: Use modern tools such as Eclipse, NetBeans etc. for software construction.

Date: 28th Oct 2024

Time: 10:00 AM - 12:50 PM
02:30 PM – 04:50 PM

Instructor: Dr. Mehvish Rashid
Lab Engineer: Mr. Aftab Farooq

Lab 07: Recursion

Introduction:

Students will have hands-on experience on designing, testing, and implementing recursive problems. Given a scenario, you will write the specifications and implement it by dividing into base case and recursive step. You may design helper methods to simplify your implementations. Write unit tests that check for compliance with the specifications.

Lab Tasks

Task 1: Recursive File Search

Objective: The objective of this lab task is to create a Java program that recursively searches for a file within a directory and its subdirectories. This exercise will help you practice the principles of software construction and recursion.

Instructions:+

1. Create a Java program that takes two command-line arguments: a directory path and a file name to search for.
2. Implement a recursive function to search for the specified file within the given directory and its subdirectories.
3. The program should display a message when it finds the file, including the full path to the file, or a message indicating that the file was not found.
4. Follow good coding practices, including meaningful variable names, comments, and modular code.
5. Implement error handling to handle cases where the specified directory does not exist or other exceptions may occur.
6. Use appropriate data structures and algorithms to efficiently search through the directory tree.
7. Test your program with different directory paths and file names to ensure its correctness and reliability.

Important: Do not forget to write the specifications and unit tests for the code.

Optional Enhancements:

1. Allow the program to search for multiple files in a single run.
2. Implement a feature to count the number of times a specific file appears within the directory and its subdirectories.
3. Provide an option to specify whether the search should be case-sensitive or case-insensitive.

Code:

Recursive_file_search:

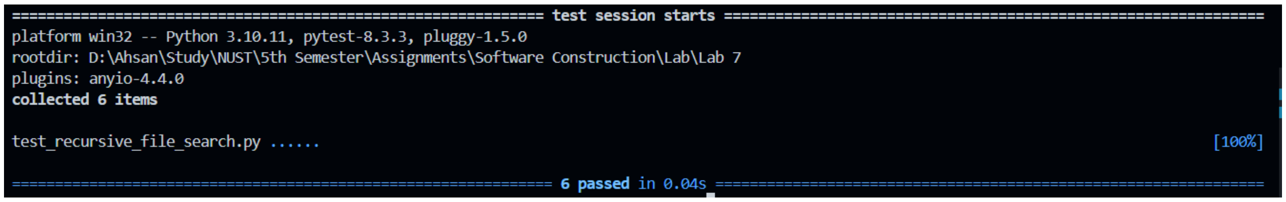
```python
import os
import sys

def find_file(directory, target_file):
    try:
        entries = os.listdir(directory)
    except FileNotFoundError:
        print(f"Error: The directory '{directory}' does not exist.")
        return None
    except PermissionError:
        print(f"Warning: Permission denied for directory '{directory}'.")
        return None
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        return None

    for entry in entries:
        entry_path = os.path.join(directory, entry)

        if os.path.isfile(entry_path) and entry == target_file:
            return entry_path

        elif os.path.isdir(entry_path):
            found_path = find_file(entry_path, target_file)
            if found_path:
                return found_path  # File found in a subdirectory

    return None


def main():
    if len(sys.argv) != 3:
        print("Usage: python search_file.py <directory_path> <file_name>")
        return

    directory_path = sys.argv[1]
    file_name = sys.argv[2]

    result = find_file(directory_path, file_name)
    if result:
        print(f"File found: {result}")
    else:
        print(f"The file '{file_name}' was not found in the directory
'{directory_path}' or its subdirectories.")

if __name__ == "__main__":
    main()
```

Test Cases:

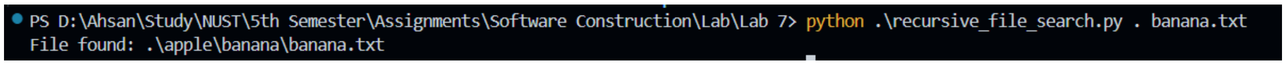
```
import os
import tempfile
import pytest
from recursive_file_search import find_file

@pytest.fixture
def setup_test_directory():
    """
    Creates a temporary directory structure for testing purposes.
    """
    with tempfile.TemporaryDirectory() as tmp_dir:
        os.mkdir(os.path.join(tmp_dir, "subdir1"))
        os.mkdir(os.path.join(tmp_dir, "subdir2"))
        os.mkdir(os.path.join(tmp_dir, "subdir1", "subsubdir1"))

        with open(os.path.join(tmp_dir, "testfile.txt"), "w") as f:
            f.write("This is a test file.")

        with open(os.path.join(tmp_dir, "subdir1", "testfile1.txt"), "w") as f:
            f.write("This is another test file.")

        with open(os.path.join(tmp_dir, "subdir2", "testfile2.txt"), "w") as f:
            f.write("This is yet another test file.")

        with open(os.path.join(tmp_dir, "subdir1", "subsubdir1",
"targetfile.txt"), "w") as f:
            f.write("This is the target file.")

        yield tmp_dir


def test_find_file_exists_in_root(setup_test_directory):
```

```python
    tmp_dir = setup_test_directory
    result = find_file(tmp_dir, "testfile.txt")
    assert result == os.path.join(tmp_dir, "testfile.txt")


def test_find_file_exists_in_subdirectory(setup_test_directory):
    tmp_dir = setup_test_directory
    result = find_file(tmp_dir, "testfile1.txt")
    assert result == os.path.join(tmp_dir, "subdir1", "testfile1.txt")


def test_find_file_exists_in_nested_subdirectory(setup_test_directory):
    tmp_dir = setup_test_directory
    result = find_file(tmp_dir, "targetfile.txt")
    assert result == os.path.join(tmp_dir, "subdir1", "subsubdir1",
"targetfile.txt")


def test_file_not_found(setup_test_directory):
    tmp_dir = setup_test_directory
    result = find_file(tmp_dir, "nonexistent.txt")
    assert result is None


def test_directory_does_not_exist():
    result = find_file("/non/existent/directory", "testfile.txt")
    assert result is None


def test_permission_denied(monkeypatch):
    def mock_os_listdir(path):
        raise PermissionError("Permission Denied")

    monkeypatch.setattr(os, "listdir", mock_os_listdir)
    result = find_file("/some/protected/directory", "testfile.txt")
    assert result is None
```

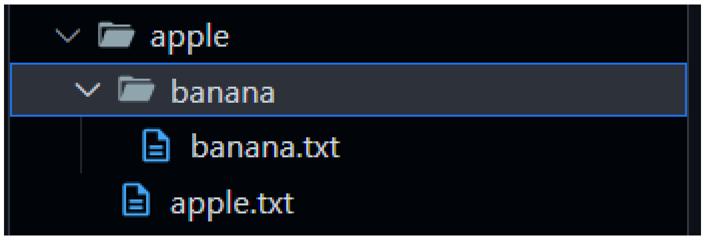
Output:



```
=============================================================== test session starts ===============================================================
platform win32 -- Python 3.10.11, pytest-8.3.3, pluggy-1.5.0
rootdir: D:\Ahsan\Study\NUST\5th Semester\Assignments\Software Construction\Lab\Lab 7
plugins: anyio-4.4.0
collected 6 items

test_recursive_file_search.py ......                                                                                                         [100%]

================================================================ 6 passed in 0.04s ================================================================
```



```
PS D:\Ahsan\Study\NUST\5th Semester\Assignments\Software Construction\Lab\Lab 7> python .\recursive_file_search.py . banana.txt
File found: .\apple\banana\banana.txt
```



```
apple
  banana
    banana.txt
  apple.txt
```

Task 2: Recursive String Permutations

Objective: The objective of this lab task is to create a Java program that generates all permutations of a given string using a recursive algorithm. This exercise will help you practice recursion and algorithm design.

Instructions:

1. Create a Java program that generates all permutations of a given string using a recursive function.
2. Implement a recursive function **generatePermutations** that takes a string as input and returns a list of all its permutations.
3. Use a recursive approach to generate permutations. You can consider swapping characters in the string to create different permutations.
4. Follow good coding practices, including meaningful variable names, comments, and modular code.
5. Implement error handling to handle cases where the input string is empty or other exceptions may occur.
6. Analyze the time complexity of the recursive algorithm. How does the time complexity compare to an iterative solution for large strings?

Optional Enhancements:

1. Provide an option for the user to choose whether to include or exclude duplicate permutations, as some characters in the input string may be identical.
2. Implement a non-recursive algorithm for generating permutations and compare its performance with the recursive solution for large strings.

Code:

String_permutations.py:

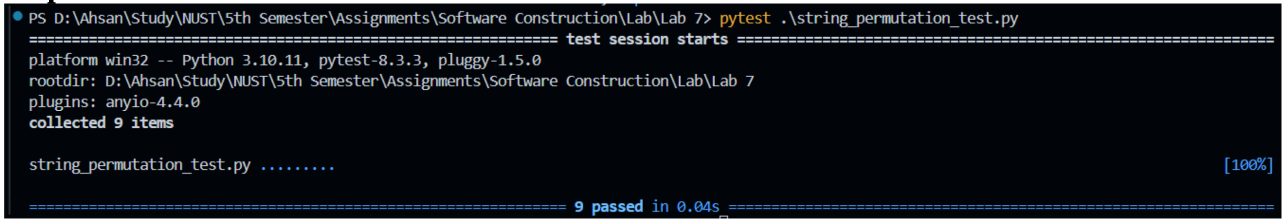
```
def generate_permutations(string):
    if not string:
```

```python
        print("Error: The input string is empty.")
        return []

    if len(string) == 1:
        return [string]

    permutations = []

    for i in range(len(string)):
        current_char = string[i]

        remaining_string = string[:i] + string[i+1:]

        for perm in generate_permutations(remaining_string):
            permutations.append(current_char + perm)

    return permutations


if __name__ == "__main__":
    user_input = input("Enter a string to generate its permutations: ")
    try:
        result = generate_permutations(user_input)
        if result:
            print(f"Permutations of '{user_input}':")
            for perm in result:
                print(perm)
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
```

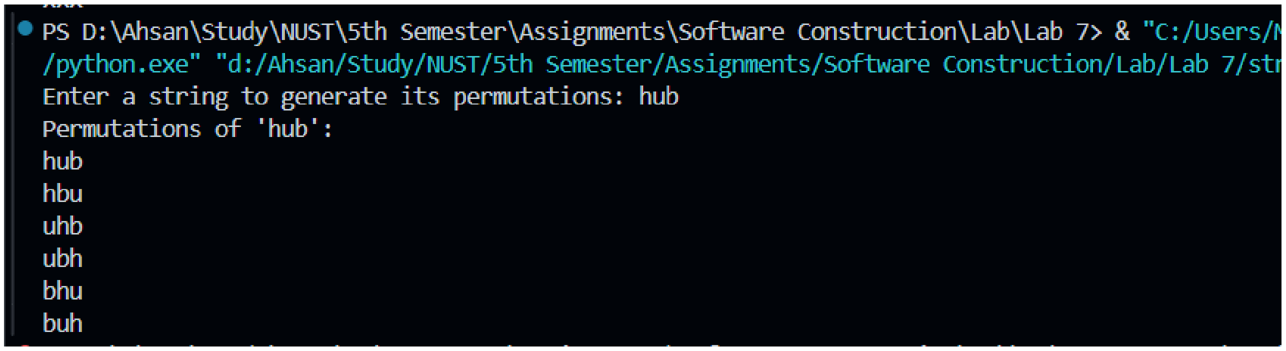
String permutation test:

```python
import pytest
from string_permutations import generate_permutations  # replace with the actual module name

def test_empty_string():
    """
    Test that an empty string returns an empty list.
    """
    assert generate_permutations("") == []

def test_single_character():
    """
    Test that a single character string returns a list with the string itself.
    """
    assert generate_permutations("a") == ["a"]

def test_two_characters():
    """
    Test that a two-character string returns two permutations.
    """
    result = generate_permutations("ab")
    assert sorted(result) == sorted(["ab", "ba"])

def test_three_characters():
    """
    Test that a three-character string returns six permutations.
    """
    result = generate_permutations("abc")
    assert sorted(result) == sorted(["abc", "acb", "bac", "bca", "cab", "cba"])

def test_large_input():
    """
    Test a slightly larger string to ensure the function does not crash.
    """
    result = generate_permutations("abcd")
    assert len(result) == 24  # 4! = 24

def test_non_alphabetic_characters():
    """
    Test that the function works with non-alphabetic characters.
    """
    result = generate_permutations("1a!")
    assert sorted(result) == sorted(["1a!", "1!a", "a1!", "a!1", "!1a", "!a1"])

@pytest.mark.parametrize("input_str, expected_length", [
    ("abc", 6),  # 3! permutations
    ("abcd", 24),  # 4! permutations
    ("abcde", 120)  # 5! permutations
])
def test_permutations_length(input_str, expected_length):
    """
    Parametrized test to check if the number of permutations matches n!.
    """
    result = generate_permutations(input_str)
    assert len(result) == expected_length
```

Ouput:



```
PS D:\Ahsan\Study\NUST\5th Semester\Assignments\Software Construction\Lab\Lab 7> pytest .\string_permutation_test.py
============================================================= test session starts ==============================================================
platform win32 -- Python 3.10.11, pytest-8.3.3, pluggy-1.5.0
rootdir: D:\Ahsan\Study\NUST\5th Semester\Assignments\Software Construction\Lab\Lab 7
plugins: anyio-4.4.0
collected 9 items

string_permutation_test.py .........                                                                                                      [100%]

============================================================== 9 passed in 0.04s ===============================================================
```



```
PS D:\Ahsan\Study\NUST\5th Semester\Assignments\Software Construction\Lab\Lab 7> & "C:/Users/M
/python.exe" "d:/Ahsan/Study/NUST/5th Semester/Assignments/Software Construction/Lab/Lab 7/str
Enter a string to generate its permutations: hub
Permutations of 'hub':
hub
hbu
uhb
ubh
bhu
buh
```

Deliverables:

Compile a single word document by filling in the solution part and submit this Word file on LMS. In case of any problems with submissions on LMS, submit your Lab assignmentsby emailing it to aftab.farooq@seecs.edu.pk.