Unit testing

- pacman -S mingw-w64-ucrt-x86_64-gcc
- pacman -S mingw-w64-x86_64-toolchain base-devel
- pacman -S mingw-w64-x86_64-cunit
- pkg-config --modversion cunit
- cd "/c/Users/sanarama//Desktop/SE lab/unit testing"
- gcc arithmetic.c test_arithmetic.c -o test_arithmetic.exe -lcunit
- ./test_arithmetic.exe

Integration testing

- Make one folder for programs
- Save all 3 programs in it
- Then select progeam file path and type cmd (it will redirect to command prompt)
- Type " gcc bank.c test_bank.c -o bank_test"
- Type " bank_test"

Refactoring

- Make one folder for programs
- Save all 2 programs in it
- Open MSYS2 mingw
- Type cd"folder path"
- Type " gcc original.c -o original.exe"
- Type "./original.exe"
- Type " gcc refactored.c -o refactored.exe"
- Type "./refactored.exe"

CRUD

- Install node js
- Make folder for programs
- Save the code as server.js and another one as index.html (inside folder of crud u need to create sub folder for index.html as public)
- Verify the version of nodejs in command prompt " node -v"
- Type cd "folder path"
- Type "npm install express cors" (you will get found 0 vulnerabilities)
- Node server.js (you will get server listening on port 3000)s
- Open browser , type "localhost:port number (localhost:3000)
- Then you can add , delete the task

CI-CD

- Create new repository
- Then click on create new file on repository page
- Give filename as main.c and type code then click on change commit with commit message with main branch
- Then again ,Give filename as Makefile and type code then click on change commit with commit message with main branch
- Then again ,Give filename as .github/workflow/ci-cd.yml and type code then click on change commit with commit message with main branch
- Then click on action it should be in right mark

UNIT TESTING

// arithmetic.c

```
#include "arithmetic.h"
int add(int a, int b) {
   return a + b;
}
int subtract(int a, int b) {
   return a - b;
}
int multiply(int a, int b) {
   return a * b;
}
int divide(int a, int b) {
   if (b == 0) {
      return 0;  // Handle division by zero (you might want to throw an error in a real application)
   }
   return a / b;
}
```

// arithmetic.h

```
#ifndef ARITHMETIC_H
#define ARITHMETIC_H
int add(int a, int b);
int subtract(int a, int b);
int multiply(int a, int b);
int divide(int a, int b);
#endif
```

```
// test_arithmetic.c
#include "arithmetic.h"
#include <CUnit/Basic.h>

void test_add(void) {
    CU_ASSERT_EQUAL(add(2, 3), 5);
    CU_ASSERT_EQUAL(add(-1, 1), 0);
    CU_ASSERT_EQUAL(add(-1, -1), -2);
}
void test_subtract(void) {
    CU_ASSERT_EQUAL(subtract(5, 3), 2);
    CU_ASSERT_EQUAL(subtract(3, 5), -2);
    CU_ASSERT_EQUAL(subtract(0, 0), 0);
}
void test_multiply(void) {
    CU_ASSERT_EQUAL(multiply(2, 3), 6);
    CU_ASSERT_EQUAL(multiply(-2, 3), -6);
    CU_ASSERT_EQUAL(multiply(0, 5), 0);
}
void test_divide(void) {
    CU_ASSERT_EQUAL(divide(6, 3), 2);
    CU_ASSERT_EQUAL(divide(10, 2), 5);
    CU_ASSERT_EQUAL(divide(5, 2), 2);  // Integer division
    CU_ASSERT_EQUAL(divide(5, 0), 0);  // Test division by zero
}
int main(void) {
    CU_pSuite suite = NULL;
    if (CUE_SUCCESS != CU_initialize_registry()) {
        return CU_get_error();
    }
    suite = CU_add_suite("Arithmetic Suite", NULL, NULL);
```

```
if (NULL == suite) {
   CU_cleanup_registry();
   return CU_get_error();
}
if ((NULL == CU_add_test(suite, "test of add()", test_add)) ||
   (NULL == CU_add_test(suite, "test of subtract()", test_subtract)) ||
   (NULL == CU_add_test(suite, "test of multiply()", test_multiply)) ||
   (NULL == CU_add_test(suite, "test of divide()", test_divide))) {
   CU_cleanup_registry();
   return CU_get_error();
}
CU_basic_set_mode(CU_BRM_VERBOSE);  // Or CU_BRM_NORMAL
CU_basic_run_tests();
CU_cleanup_registry();
return CU_get_error();
```

REFACTORED

//original code

```
#include <stdio.h>

// Original code with code smells
int calculate_and_print(int a, int b, int operation) {
    int result;

    if (operation == 1) { // 1 means addition
        result = a + b;
        printf("Addition result: %d\n", result);
    } else if (operation == 2) { // 2 means subtraction
        result = a - b;
        printf("Subtraction result: %d\n", result);
    } else if (operation == 3) { // 3 means multiplication
        result = a * b;
        printf("Multiplication result: %d\n", result);
    } else {
        printf("Invalid operation\n");
        return -1; // Indicate error
    }

    return result;
}

int main() {
    calculate_and_print(10, 5, 1);
    calculate_and_print(10, 5, 2);
    calculate_and_print(10, 5, 3);
```

```
    calculate_and_print(10, 5, 4);

    return 0;
}
```

//Refactored code

```
#include <stdio.h>

// Using enums for better readability
typedef enum {
    ADDITION = 1,
    SUBTRACTION,
    MULTIPLICATION,
    INVALID
} Operation;

// Function to perform the calculation
int calculate(int a, int b, Operation op) {
    switch (op) {
        case ADDITION:
            return a + b;
        case SUBTRACTION:
            return a - b;
        case MULTIPLICATION:
            return a * b;
        default:
            return -1; // Indicate error
    }
}
// Function to print the result
void print_result(int result, Operation op) {
```

```
    switch (op) {
        case ADDITION:
            printf("Addition result: %d\n", result);
            break;
        case SUBTRACTION:
            printf("Subtraction result: %d\n", result);
            break;
        case MULTIPLICATION:
            printf("Multiplication result: %d\n", result);
            break;
        default:
            printf("Invalid operation\n");
    }
}

int main() {
    int result;

    result = calculate(10, 5, ADDITION);
    print_result(result, ADDITION);

    result = calculate(10, 5, SUBTRACTION);
    print_result(result, SUBTRACTION);

    result = calculate(10, 5, MULTIPLICATION);
    print_result(result, MULTIPLICATION);

    result = calculate(10, 5, INVALID);
    print_result(result, INVALID);

    return 0;
}
```

INTEGRATION

//bank.c

```
#include "bank.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

Bank* create_bank() {
    Bank* bank = (Bank*)malloc(sizeof(Bank));
    if (bank) {
        bank->account_count = 0;
    }
    return bank;
}

bool create_account(Bank* bank, const char* account_number, double initial_balance) {
    if (bank->account_count >= 100) return false; // Max accounts reached
    if (get_account(bank, account_number)) return false; // Account exists

    Account* new_account = (Account*)malloc(sizeof(Account));
    if (!new_account) return false;

    strncpy(new_account->account_number, account_number, sizeof(new_account->account_number) - 1);
    new_account->account_number[sizeof(new_account->account_number) - 1] = '\0'; // Ensure null termination
    new_account->balance = initial_balance;

    bank->accounts[bank->account_count++] = new_account;
    return true;
}
```

```
Account* get_account(Bank* bank, const char* account_number) {
    for (int i = 0; i < bank->account_count; i++) {
        if (strcmp(bank->accounts[i]->account_number, account_number) == 0) {
            return bank->accounts[i];
        }
    }
    return NULL;
}

bool transaction_deposit(Account* account, double amount) {
    if (amount > 0) {
        account->balance += amount;
        return true;
    }
    return false;
}

bool transaction_withdraw(Account* account, double amount) {
    if (amount > 0 && account->balance >= amount) {
        account->balance -= amount;
        return true;
    }
    return false;
}

Transaction* create_transaction() {
    Transaction* transaction = (Transaction*)malloc(sizeof(Transaction));
    if (transaction) {
        transaction->deposit = transaction_deposit;
        transaction->withdraw = transaction_withdraw;
    }
```

```
    return transaction;
}

bool deposit(Bank* bank, const char* account_number, double amount) {
    Account* account = get_account(bank, account_number);
    if (account) {
        Transaction* transaction = create_transaction();
        bool result = transaction->deposit(account, amount);
        free(transaction);
        return result;
    }
    return false;
}

bool withdraw(Bank* bank, const char* account_number, double amount) {
    Account* account = get_account(bank, account_number);
    if (account) {
        Transaction* transaction = create_transaction();
        bool result = transaction->withdraw(account, amount);
        free(transaction);
        return result;
    }
    return false;
}
```

//test.c

```
#include <CUnit/CUnit.h>
#include <CUnit/Basic.h>
#include <stdio.h>

void test_example() {
    CU_ASSERT(2 + 2 == 4);
}

int main() {
    CU_initialize_registry();
    CU_pSuite suite = CU_add_suite("Example Suite", 0, 0);
    CU_add_test(suite, "Test Example", test_example);
    CU_basic_run_tests();
    CU_cleanup_registry();
    return 0;
}
```

//test_bank.c

```
#include <stdlib.h>
#include "bank.h"
#include <stdio.h>

int main() {
    Bank* bank = create_bank();
    create_account(bank, "12345", 1000.0);
    Account* account = get_account(bank, "12345");

    if (deposit(bank, "12345", 500.0)) {
        printf("Deposit successful. New balance: %.2f\n", account->balance);
    } else {
        printf("Deposit failed.\n");
    }

    if (withdraw(bank, "12345", 300.0)) {
        printf("Withdrawal successful. New balance: %.2f\n", account->balance);
    } else {
        printf("Withdrawal failed.\n");
    }

    if (!withdraw(bank, "12345", 2000.0)) {
        printf("Insufficient funds check passed. Balance: %.2f\n", account->balance);
    } else {
        printf("Insufficient funds check failed.\n");
    }

    if (!deposit(bank, "invalid", 100.0)) {
        printf("Invalid account deposit check passed.\n");
```

```
    } else {
        printf("Invalid account deposit check failed.\n");
    }

    if (!withdraw(bank, "invalid", 100.0)) {
        printf("Invalid account withdraw check passed.\n");
    } else {
        printf("Invalid account withdraw check failed.\n");
    }

    // Memory cleanup
    for (int i = 0; i < bank->account_count; i++) {
        free(bank->accounts[i]);
    }
    free(bank);

    return 0;
}
```

//bank.h

```
#ifndef BANK_H
#define BANK_H

#include <stdbool.h>

typedef struct {
    char account_number[20];
    double balance;
} Account;

typedef struct {
    Account* accounts[100]; // Array of account pointers
```

```
    int account_count;
} Bank;

typedef struct {
    bool (*deposit)(Account* account, double amount);
    bool (*withdraw)(Account* account, double amount);
} Transaction;

// Bank functions
Bank* create_bank();
bool create_account(Bank* bank, const char* account_number, double initial_balance);
Account* get_account(Bank* bank, const char* account_number);
bool deposit(Bank* bank, const char* account_number, double amount);
bool withdraw(Bank* bank, const char* account_number, double amount);

// Transaction functions
bool transaction_deposit(Account* account, double amount);
bool transaction_withdraw(Account* account, double amount);
Transaction* create_transaction();

#endif // BANK_H
```

CICD

//main.c

```
#include <stdio.h>

int main() {
    int num1 = 10;
    int num2 = 5;
    int sum = num1 + num2;
    printf("Sum: %d\n", sum);
    return 0;
}
```

//Makefile

```
CC = gcc
CFLAGS = -Wall -Wextra

all: main

main: main.c
	$(CC) $(CFLAGS) main.c -o main

clean:
	rm -f main
```

//.github/workflows

```
name: C Build CI

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Install GCC
      run: sudo apt update && sudo apt install -y build-essential

    - name: Build project
      run: make

    - name: Run executable
      run: ./main
```

CRUD

```
//server.js

const express = require('express');
const cors = require('cors');

const app = express();
const PORT = 3000;

// Middleware
app.use(cors());
app.use(express.json());
app.use(express.static('public'));

// Request logging
app.use((req, res, next) => {
  console.log(`${req.method} ${req.path} - ${new Date().toISOString()}`);
  next();
});

let tasks = [
  { id: 1, title: 'Learn Express', completed: false },
  { id: 2, title: 'Build a simple app', completed: true },
];
let nextTaskId = 3;

// GET all tasks
app.get('/tasks', (req, res) => {
  console.log('Sending all tasks:', tasks);
  res.json(tasks);
});
```

```
// POST a new task
app.post('/tasks', (req, res) => {
  console.log('Received task creation request. Body:', req.body);
  if (!req.body || !req.body.title) {
    console.log('Invalid request: Missing title');
    return res.status(400).json({ error: 'Title is required' });
  }
  const newTask = { id: nextTaskId++, title: req.body.title, completed: false };
  tasks.push(newTask);
  console.log('Task created successfully:', newTask);
  res.status(201).json(newTask);
});

// PUT (update) a task
app.put('/tasks/:id', (req, res) => {
  const taskId = parseInt(req.params.id);
  console.log(`Updating task ${taskId}. Body:`, req.body);
  const task = tasks.find(t => t.id === taskId);
  if (!task) {
    console.log(`Task ${taskId} not found`);
    return res.status(404).json({ error: 'Task not found' });
  }
  task.title = req.body.title || task.title;
  task.completed = req.body.completed !== undefined ? req.body.completed : task.completed;
  console.log(`Task ${taskId} updated:`, task);
  res.json(task);
});

// DELETE a task
app.delete('/tasks/:id', (req, res) => {
  const taskId = parseInt(req.params.id);
  console.log(`Deleting task ${taskId}`);
```

```
  const initialLength = tasks.length;
  tasks = tasks.filter(t => t.id !== taskId);
  if (tasks.length === initialLength) {
    console.log(`Task ${taskId} not found for deletion`);
    return res.status(404).json({ error: 'Task not found' });
  }
  console.log(`Task ${taskId} deleted successfully`);
  res.json({ message: 'Task deleted successfully' });
});

// Error handler
app.use((err, req, res, next) => {
  console.error('Unhandled error:', err);
  res.status(500).json({ error: 'Internal server error' });
});

app.listen(PORT, () => {
  console.log(`Server listening on port ${PORT}`);
});
```

//index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Task Manager</title>
  <style>
    body { font-family: Arial, sans-serif; max-width: 600px; margin: auto; padding: 20px; }
    h1 { text-align: center; color: #333; }
    .error-message { color: red; display: none; margin-bottom: 10px; }
```

```
    .add-task { display: flex; margin-bottom: 20px; }
    #taskInput { flex: 1; padding: 10px; border: 1px solid #ddd; border-radius: 4px 0 0 4px; }
    #addButton { padding: 10px 15px; border: none; background: #4CAF50; color: #fff; border-radius: 0 4px 4px 0; cursor: pointer; }
    #addButton:hover { background: #45a049; }
    .task-list { list-style: none; padding: 0; }
    .task-item { display: flex; align-items: center; padding: 10px; border-bottom: 1px solid #eee; }
    .task-item.completed .task-title { text-decoration: line-through; color: #888; }
    .task-checkbox { margin-right: 10px; }
    .task-title { flex: 1; }
    .delete-btn { background: #f44336; color: #fff; border: none; padding: 5px 10px; border-radius: 4px; cursor: pointer; }
    .delete-btn:hover { background: #d32f2f; }
    .status { text-align: center; margin-top: 20px; padding: 10px; border-radius: 4px; }
    .status.success { background: #dff0d8; color: #3c763d; }
    .loading { text-align: center; display: none; margin: 20px 0; }
  </style>
</head>
<body>
  <h1>Task Manager</h1>
  <div id="errorMessage" class="error-message"></div>
  <div class="add-task">
    <input type="text" id="taskInput" placeholder="Add a new task..."/>
    <button id="addButton">Add Task</button>
  </div>
  <div id="loading" class="loading">Loading tasks...</div>
  <ul id="taskList" class="task-list"></ul>
  <div id="status" class="status"></div>

  <script>
    const API_URL = 'http://localhost:3000';
    const taskInput = document.getElementById('taskInput');
```

```
const addButton = document.getElementById('addButton');
const taskList = document.getElementById('taskList');
const errorMessage = document.getElementById('errorMessage');
const statusDiv = document.getElementById('status');
const loadingDiv = document.getElementById('loading');

function showError(msg) {
  errorMessage.textContent = msg;
  errorMessage.style.display = 'block';
  setTimeout(() => errorMessage.style.display = 'none', 5000);
}
function showStatus(msg, ok = true) {
  statusDiv.textContent = msg;
  statusDiv.className = ok ? 'status success' : 'status error';
  setTimeout(() => statusDiv.textContent = '', 3000);
}
function showLoading(on = true) { loadingDiv.style.display = on ? 'block' : 'none'; }

async function loadTasks() {
  showLoading(true);
  try {
    const res = await fetch(`${API_URL}/tasks`);
    if (!res.ok) throw new Error(`${res.status} ${res.statusText}`);
    const tasks = await res.json();
    renderTasks(tasks);
  } catch (err) {
    showError(`Failed to load: ${err.message}`);
  } finally {
    showLoading(false);
  }
}
```

```
function renderTasks(tasks) {
  taskList.innerHTML = '';
  if (tasks.length === 0) {
    const li = document.createElement('li');
    li.textContent = 'No tasks yet.';
    taskList.appendChild(li);
    return;
  }
  tasks.forEach(t => {
    const li = document.createElement('li');
    li.className = `task-item ${t.completed ? 'completed' : ''}`;
    li.dataset.id = t.id;

    const cb = document.createElement('input');
    cb.type = 'checkbox';
    cb.checked = t.completed;
    cb.addEventListener('change', () => updateTask(t.id, cb.checked));

    const span = document.createElement('span');
    span.className = 'task-title';
    span.textContent = t.title;

    const btn = document.createElement('button');
    btn.className = 'delete-btn';
    btn.textContent = 'Delete';
    btn.addEventListener('click', () => deleteTask(t.id));

    li.append(cb, span, btn);
    taskList.appendChild(li);
  });
}
```

```
async function addTask() {
  const title = taskInput.value.trim();
  if (!title) return showError('Please enter a task title');
  try {
    const res = await fetch(`${API_URL}/tasks`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ title })
    });
    if (!res.ok) throw new Error(`${res.status} ${res.statusText}`);
    taskInput.value = '';
    showStatus('Task added!');
    loadTasks();
  } catch (err) {
    showError(`Add failed: ${err.message}`);
  }
}

async function updateTask(id, completed) {
  try {
    const res = await fetch(`${API_URL}/tasks/${id}`, {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ completed })
    });
    if (!res.ok) throw new Error(`${res.status} ${res.statusText}`);
    showStatus(`Task ${completed ? 'done' : 'reopened'}`);
  } catch (err) {
    showError(`Update failed: ${err.message}`);
  }
}
async function deleteTask(id) {
```

```
      try {
        const res = await fetch(`${API_URL}/tasks/${id}`, { method: 'DELETE' });
        if (!res.ok) throw new Error(`${res.status} ${res.statusText}`);
        showStatus('Task deleted');
        loadTasks();
      } catch (err) {
        showError(`Delete failed: ${err.message}`);
      }
    }
    addButton.addEventListener('click', addTask);
    taskInput.addEventListener('keypress', e => { if (e.key === 'Enter') addTask(); });
    document.addEventListener('DOMContentLoaded', loadTasks);
  </script>
</body>
</html>
```