**Code: Fixes, Errors, and Lessons Learned**

This document explains how I approached building the project, the errors I encountered along the way, how I fixed them, and what caused those issues. It’s a straightforward summary of the process, highlighting the challenges and the solutions I found to make things work.

**1. Documentation and Build Process**

**1.1 How to Build**

1. **Ensure Project Structure:** Your project directory should have the following structure:

class\_db/

├── src/

│ ├── main.c

│ ├── repl.c

│ ├── repl.h

│ ├── btree.c

│ ├── btree.h

├── Makefile

1. **Makefile Setup:** Use the following Makefile to build the project:

CC=gcc

CFLAGS=-Wall -Wextra -g

SOURCES=src/main.c src/repl.c src/btree.c

OBJECTS=$(SOURCES:.c=.o)

EXECUTABLE=db

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)

$(CC) $(CFLAGS) -o $@ $^

%.o: %.c

$(CC) $(CFLAGS) -c -o $@ $<

clean:

rm -f $(OBJECTS) $(EXECUTABLE)

1. **Build Commands:**
   * To clean previous builds:
   * make clean
   * To compile the project:
   * make
2. **Run the Program:**
3. ./db

**1.2 Attention Points During Build**

* Ensure all .h files include necessary dependencies without circular dependencies.
* Verify function prototypes match their definitions.
* Check for unused variables or warnings during compilation.

**2. Encountered Errors and Solutions**

**Error 1: Unknown Type Name 'Row'**

**Cause:** The Row structure was used in btree.h before being fully defined.

**Solution:**

* Added forward declarations in btree.h:
* typedef struct Row Row;
* typedef struct Table Table;

**Error 2: Implicit Declaration of Functions**

**Cause:** Functions like insert\_row were used without being declared correctly.

**Solution:**

* Verified that function prototypes in btree.h match their implementations in btree.c.
* Ensured btree.h was included in repl.c.

**Error 3: Conflicting Types for Functions**

**Cause:** Function prototypes in headers did not match the implementations.

**Solution:**

* Updated prototypes in btree.h and repl.h to reflect actual function definitions.

**Error 4: Duplicate IDs Allowed**

**Cause:** The insert\_row function did not check for existing IDs.

**Solution:**

* Added an id\_exists function to verify if an ID already exists before insertion:
* bool id\_exists(TreeNode\* root, int id);
* Modified insert\_row to return false if the ID exists.

**Error 5: Redundant Messages on Errors**

**Cause:** Errors like "ID already exists" were reported multiple times due to unfiltered execution flow.

**Solution:**

* Updated execute\_insert to handle the return value of insert\_row and print messages conditionally.

**Error 6: Deletion of Nodes by ID**

**Cause:** Initial implementation lacked functionality to delete a node by ID.

**Solution:**

* Implemented delete\_by\_id with proper tree node rebalancing for binary search trees.

**Error 7: Undefined Commands in the Terminal**

**Cause:** No user guidance was displayed when running the .exe.

**Solution:**

* Added a print\_help function to display available commands at the start of the application.

**3. Compiler Flags**

**Selected Flags and Justifications**

1. **-Wall:** Enables all commonly used warning messages to identify potential issues in the code.
2. **-Wextra:** Adds extra warnings beyond -Wall, such as unused parameters and comparisons.
3. **-g:** Generates debugging information, allowing tools like gdb to inspect runtime behavior.
4. **-c:** Compiles source files into object files without linking, ensuring modular compilation.

**Compiler Version**

* Recommended GCC version: **9.3.0 or later**
* Verified compatibility with **MinGW GCC 14.2.0**.

**4. Best Practices for Maintenance**

1. **Documentation:**
   * Clearly comment functions and structures in .h files.
   * Maintain a changelog to track modifications.
2. **Error Handling:**
   * Ensure all error messages are clear and non-redundant.
   * Use meaningful return values (bool, enum) to indicate success or failure.
3. **Testing:**
   * Create test cases for each command (insert, select, delete).
   * Use boundary cases (e.g., empty tree, duplicate IDs).
4. **Modular Code:**
   * Keep functionalities in separate files (btree, repl).
   * Avoid tight coupling between modules.
5. **Continuous Integration:**
   * Automate builds and tests using CI tools like GitHub Actions or Jenkins.

By following this guide, you can ensure efficient building, maintenance, and debugging of your database application.