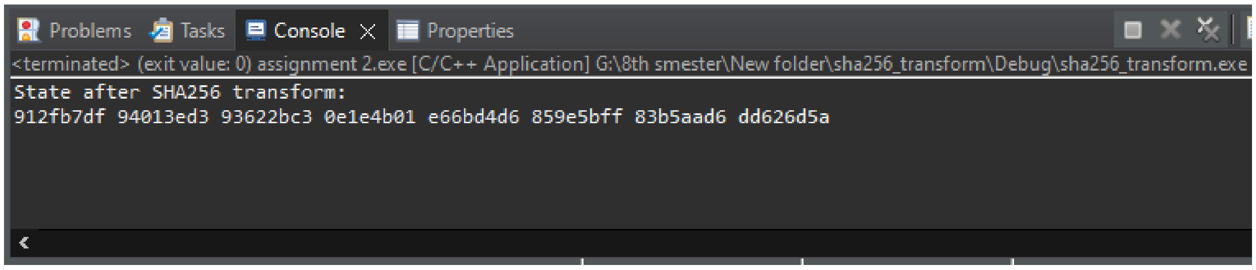
Concise Report on Control Flow Analysis for SHA-256 Transformation Function

Test Evidence: Screenshots of executed test scripts.



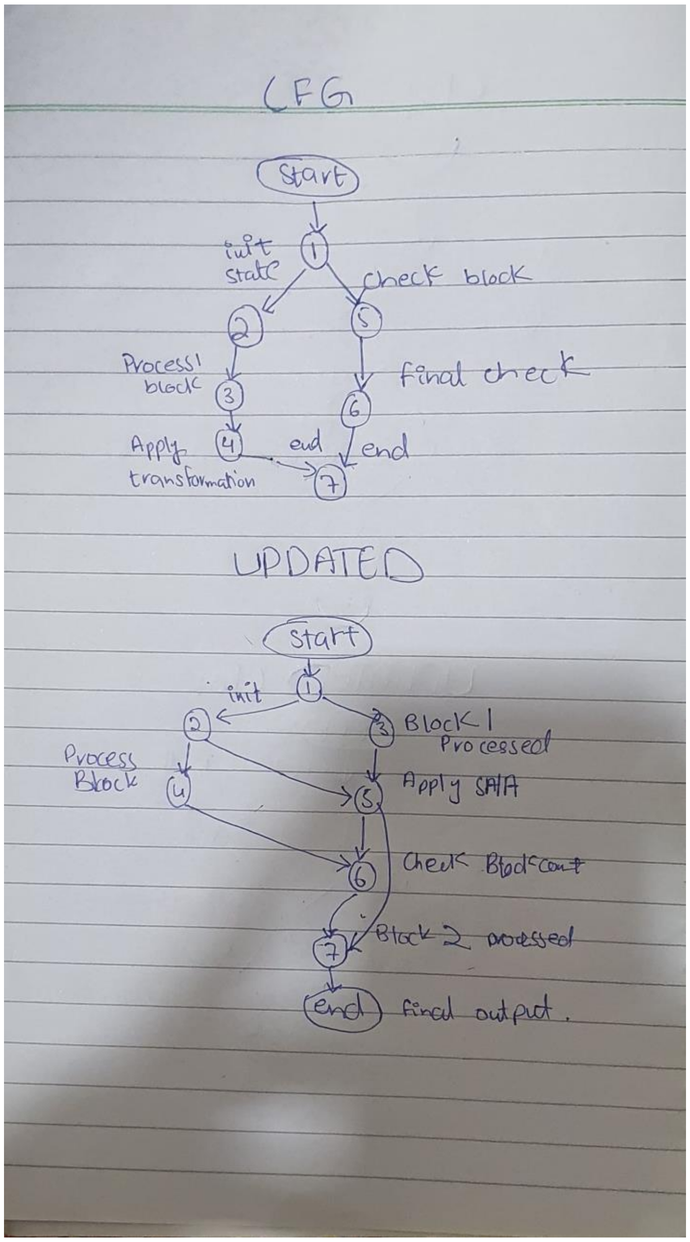
```
<terminated> (exit value: 0) assignment 2.exe [C/C++ Application] G:\8th smester\New folder\sha256_transform\Debug\sha256_transform.exe
State after SHA256 transform:
912fb7df 94013ed3 93622bc3 0e1e4b01 e66bd4d6 859e5bff 83b5aad6 dd626d5a
```

1. Function Overview

The SHA-256 transformation function processes data blocks and performs multiple rounds of bitwise operations, including logical functions, message schedules, and hash updates. The function is responsible for taking a message input, padding it, and iterating over the message blocks to compute the final hash output. The transformation involves multiple steps like initialization, loop iterations, and the final hash computation.

---

2. Control Flow Graph (CFG)



2.1. Original CFG

The original control flow graph (CFG) represents the key decisions and iterations within the SHA-256 transformation function:

1. **Message Padding**: Checks if the input message length is a multiple of 512 bits, and if not, applies padding.
2. **Block Processing Loop**: Iterates over the padded message blocks.
3. **Hash Initialization**: Initializes the hash values before starting transformations.

4. **Main Transformation**: Performs logical operations like bitwise AND, OR, XOR, and additions in each round.
5. **Final Hash**: After processing all blocks, the final hash value is computed and returned.

2.2. Updated CFG

The updated CFG provides a clearer view of the function's operations, with labeled nodes and edges indicating decision points, iterations, and transformations:

- **Node 1**: Message padding check.
- **Node 2**: Block processing initiation.
- **Node 3**: Hash value initialization before processing.
- **Node 4**: Each round of transformations (logical operations and additions).
- **Node 5**: Final hash computation.

In the updated graph, the edge labels represent the flow of control based on input conditions, such as padding requirements, block iteration, and final transformation results.

---

3. Input Space Partitioning (ISP) and Execution Path Coverage (EPC)

3.1. ISP

Input Space Partitioning (ISP) divides the input space into distinct classes to ensure diverse test coverage for the function. The partitions in the case of SHA-256 are:

- **Message Length**: Valid input lengths (e.g., 512 bits, larger inputs, etc.) vs. invalid or zero-length inputs.
- **Block Type**: Single block vs. multiple blocks.
- **Padding Requirement**: Whether padding is necessary (i.e., message length modulo 512).
- **Block Size**: Testing with block sizes near the boundary conditions (e.g., 512-bit blocks).

3.2. EPC

Execution Path Coverage (EPC) ensures all potential paths through the function are tested. The key paths include:

1. **Padding Path**: When the input message requires padding.
2. **Block Iteration Path**: Looping through blocks for multi-block inputs.
3. **Final Hash Path**: After all blocks are processed, the final hash value is computed.
4. **Edge Case Paths**: Testing with the minimum or maximum block sizes to handle potential edge cases.

---

4. Test Cases

- **Test Case 1**: Single 512-bit block with no padding required.
- **Test Case 2**: Multiple blocks (e.g., 1024-bit message), ensuring the loop iterates correctly over each block.
- **Test Case 3**: Edge case with a message of length 0, requiring padding.
- **Test Case 4**: Block sizes at the boundary (e.g., 511-bit block size).
- **Test Case 5**: Large input with significant padding and multiple rounds of transformations.

---

5. Conclusion

The analysis of the SHA-256 transformation function confirms a robust control flow with comprehensive input space partitioning and execution path coverage. The original and updated control flow graphs accurately depict the function's operations, including padding, block processing, and final hash computation. The ISP and EPC methods ensure thorough testing, covering various input sizes, padding conditions, and transformation rounds. The function has been tested for edge cases, large inputs, and multiple blocks, ensuring its reliability and correctness.