Test Plan for II. Pointers

I use ChatGPT to help me generate this test plan. Here is my prompt:

> This is II. pointers' description:
>
> Description
>
> 1. Use void* malloc(size_t size) to allocate argv[1] bytes of memory, i.e., argv[1] always is an integer. Do not write codes to check if argv[1] is an integer. Notice that you must turn argv[1], a char* type data, into the integer type to process. Please use atoi() in stdlib.h for this conversion.
>
> 2. Use char* and a for loop, assign to the block of memory, byte-by-byte, 'A' to 'Z' then '1' to '9' repeatedly, to fill up the allocated memory. Use one printf to print the string generated above.
>
> 3. Then use an int* and a while loop, in each iteration, print the memory content and its subtract 1 value in a pair of integers, e.g., 1145258561 : 1145258560
>
> 4. In the integer process loop, you need to implement a guard to prevent segmentation faults.
>
> Please write a TESTPLAN file to describe a test plan and verify the program's correctness.

➢ Objective:

- The goal is to verify that the program allocates memory correctly, fills it with a repeating pattern of characters (A-Z, 1-9), and processes this memory content using integer pointers without causing segmentation faults.

➢ Test Environment:

- **System**: Ubuntu 20.04 LTS or Windows 10
- **Compiler**: GCC version 9.3.0 (Linux), MinGW (Windows)
- **Command to Compile**: gcc -o <your_object_file> <your_c_program>
- **Command to Run**: ./pointer_program <memory_size>

➢ Test Cases:

○ Test Case 1: Small Memory Size

- **Input**: 32 bytes

- **Steps**:
    1. **Run the program with a memory size of 32 bytes**: ./<your_executable_file> 32.
    2. Observe the printed string (should be a pattern of A-Z followed by 1-6).
    3. Observe the printed pairs of integers from the allocated memory and their corresponding subtract-1 values.
- **Expected Output**:
    - The printed string is ABCDEFGHIJKLMNOPQRSTUVWXYZ123456.
    - The program prints integer pairs, ensuring the integer pointer processes memory correctly without a segmentation fault.
- **Edge Case Handling**: The while loop correctly stops before accessing memory beyond the allocated size.

- **Test Case 2: Medium Memory Size**

- **Input**: 128 bytes
- **Steps**:
    1. **Run the program with a memory size of 128 bytes**:

       ./<your_executable_file> 128.
    2. Observe the printed string (should be the pattern of A-Z, 1-9 repeated multiple times).
    3. Observe the printed pairs of integers from the allocated memory and their corresponding subtract-1 values.
- **Expected Output**:
    - The printed string is a repeating sequence of A-Z followed by 1-9.
    - The program prints integer pairs, correctly processes the memory without accessing out-of-bound addresses.

- **Test Case 3: Large Memory Size**

- **Input**: 1024 bytes
- **Steps**:
    1. **Run the program with a memory size of 1024 bytes**:

./<your_executable_file>1024.

2. Observe the printed string (a longer repetition of A-Z, 1-9).
3. Observe the printed integer pairs and subtract-1 values.

- **Expected Output**:
  - A long sequence of the repeating A-Z, 1-9 pattern.
  - Integer pairs printed without causing segmentation faults.
- **Edge Case Handling**: The program handles large memory sizes efficiently and avoids out-of-bound access.

➢ Expected Behavior Across All Test Cases:

- The program correctly allocates memory using malloc.
- The allocated memory is filled with a repeating pattern of A-Z and 1-9.
- The integer pointer processing should correctly handle memory without causing segmentation faults, even in large memory allocations.
- Proper bounds checking in the while loop ensures the program doesn't access memory beyond the allocated size.