Q2 Write-Up

Combining C and Assembly Language Programs

Working of the Program(s)

In the programs, first, the main() calls function A(). A() asks the user for inputting a 64 bit integer. This 64 bit integer is passed as a parameter when A() calls B() by doing B(num). After calling B with the required argument, the control now passes on to function B(), which is written in assembly.
On reaching B(), we set up the stack frame for B(). The argument is present in the register rdi. We move it to rax, and call the custom made _printASCII() function to perform the task.

To print the ASCIIs, I have used the **logic of dividing** the number by 256 (2^8) every time, until the number becomes 0. On division by 256, the remainder that we would get each time would represent an 8 bit integer than can be represented as an ASCII character. Using this method, we can output a string of ASCII Characters. Since this method would parse the ASCII characters in reverse, I used another logic to conquer this. I have declared memory 'base' of 100 bytes in the .bss section. In this, we keep on storing the numbers in reverse and keep incrementing the address so the next 8 bit integer (the remainder) could be stored in the next address. Once the number becomes 0, we decrement the address in order to access, and thereby, print the ASCII numbers in the correct order.

So, in _printASCII, we move the 'base' address to r9, and first we store the newline character '10' at this address and then increment r9 so r9 points to next address. Now, in the _storeASCII loop, we keep on dividing the number by 256, and store the remainder (in dl) into the address pointed to by r9 and increment r9 so that it further points to next address. We keep on looping until the number becomes 0. After this, in the loop _reverseASCII, we perform the opposite, by decrementing r9, so that it points to the previous address. We now start printing the ascii characters in the addresses one by one using the write syscall, and keep decrementing r9 and eventually, we get a string of ASCII characters in correct order. This loop continues until the address at r9 is same as the 'base' address (which was the beginning). We then ret back to B:

Now back in B, after cleaning the stack frame using 'leave', we need to modify the return address, so that function returns back to C() instead of A(). For this, we move the address of function C() into rax register (C() has been declared as extern in the beginning of program). Now, since the base_pointer+8 contains the return address, we modify it with C()'s address by moving rax into [rbp+8]. Now the return address has been modified, and on performing **ret**, the control would instead move into function C().

In function C(), we show that the control is present in it, and perform exit() system call, to exit from the program.

Compiling and running of Program(s)

Simply running 'make' in the same directory as the question, would compile and execute the program. The actual step by step process is as follows:

```
nasm -f elf64 B.asm
gcc A.c B.o C.c -o executable -no-pie
./executable
```