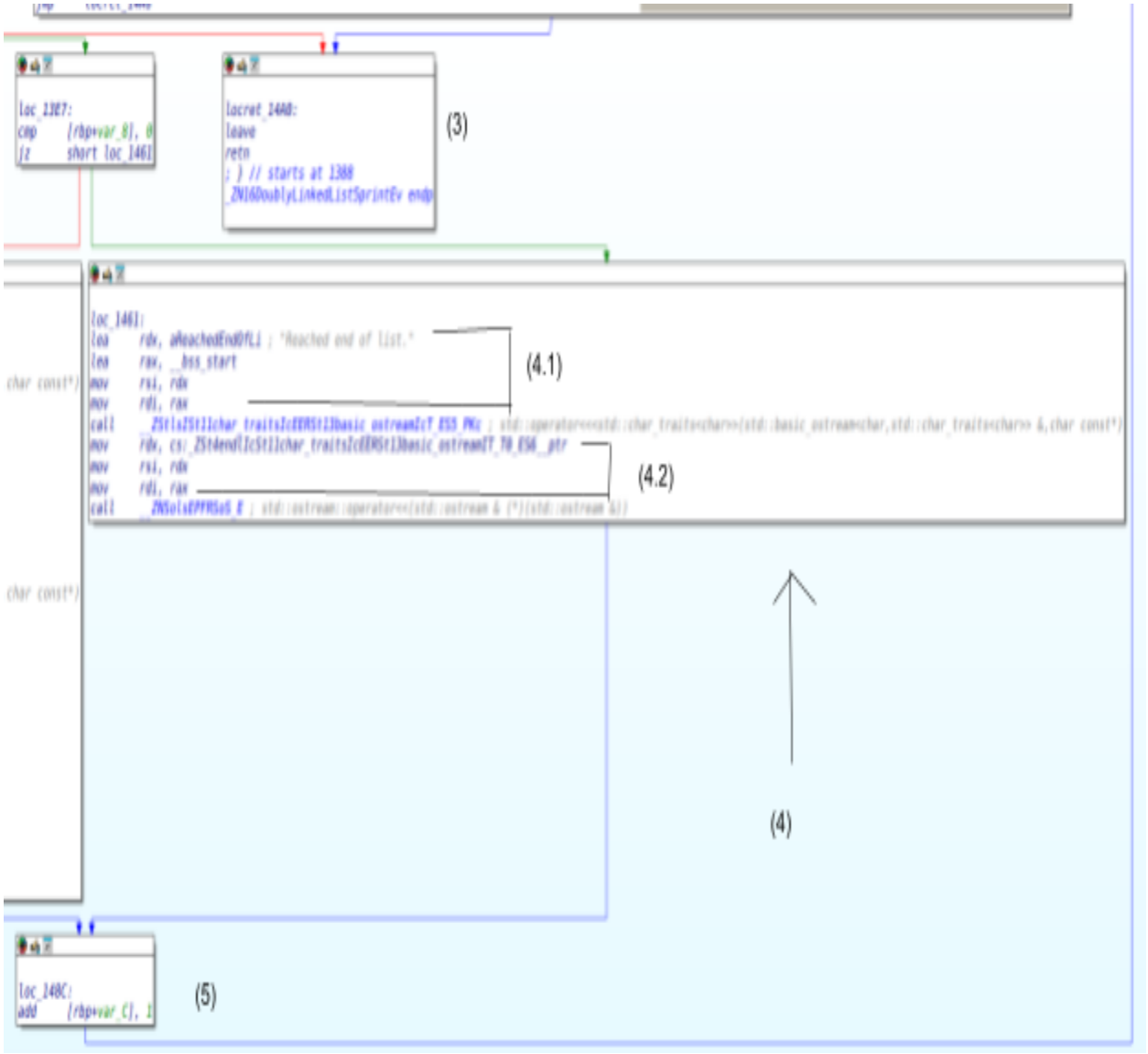
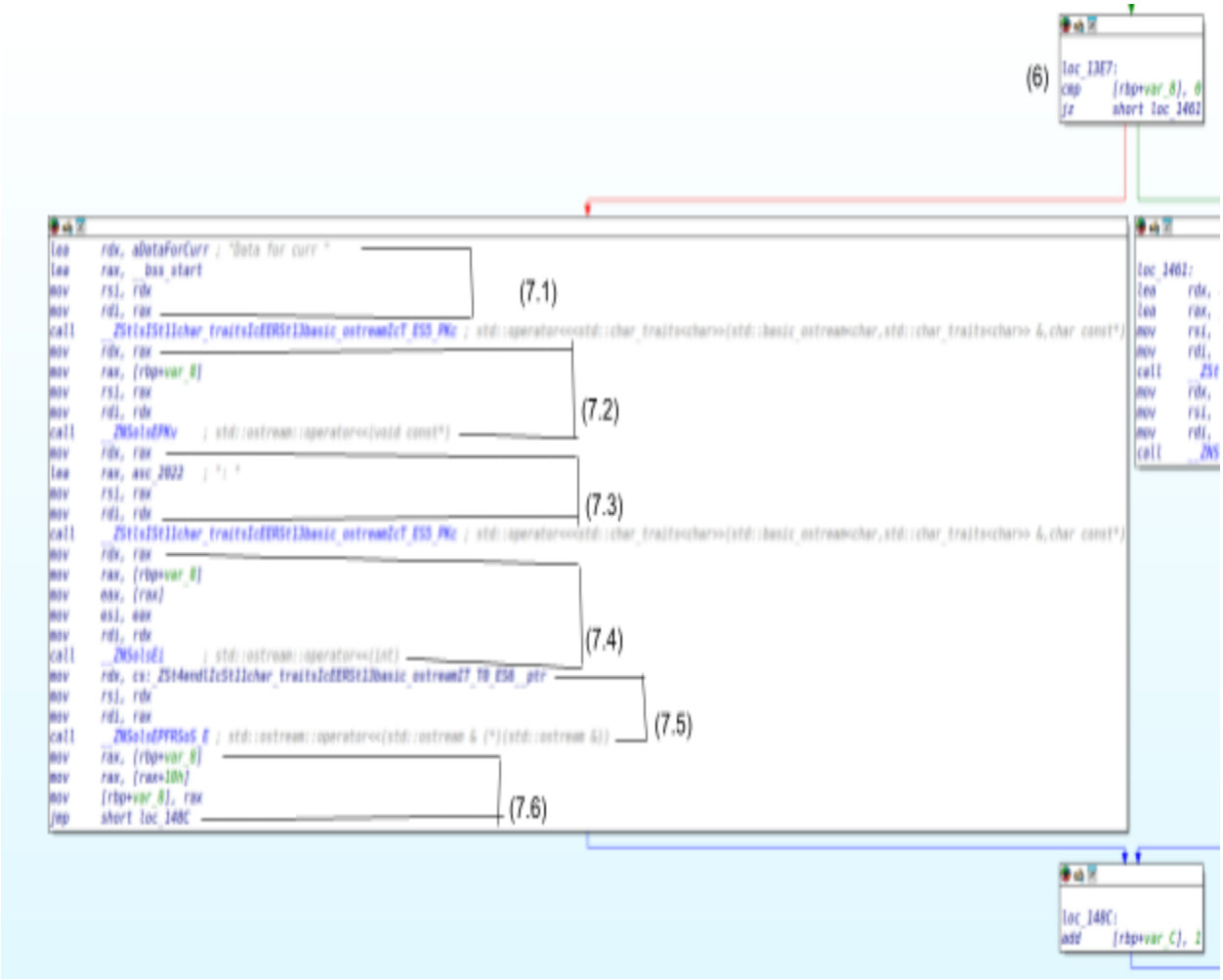
NOTE: use namespace std is defined locally within source code and the functions from std are just the names and not std:name

1. Function Preamble
2. Loads in the node's head to check if it exists
    - 2a.1. Loads in the head to set as the current node and sets i to 0 before starting the loop.
    - 2a.2. Main loop logic, loads in whatever the current node is and checks if i is less than length of the list (it loads in len from memory)
    - 2b.1. Grabs cerr from memory (stored within .bss to be loaded into at runtime) and moves it into rdi to be used with the "List is empty" string stored within rsi and then calls the iostream to start printing out the string.
    - 2b.2. Changes rdx to point to endl's .got entry and uses the returned object from cerr (an ostream obj) to continue printing to terminal.



3. Exit statement for 2b.1/2b.2. Note that there is no return in the source code.
4. It's important to note that this case is never reached due to being under the curr check. (and thus the statement is not printed out). After that check is made, i will be incremented and checked against the list's length, exiting if it's greater denoting the end of the list. To fix this, we moved the cout statement to after the for loop.
    - 4.1. Loads in cout (which is located at the start of the .bss file) to start the iostream
    - 4.2. Same as 2b.2.
5. Increments i to be checked against len in the loop. Also goes back to 2a.2.



6. Checks if the current node exists (Note, only goes to case 7 due to 4 not being reachable)
7.
   - 7.1. Same cout style as 4.1 but with “Data for curr”
   - 7.2. Loads in where the current node is located within memory to print in the IOstream. Important to node that it only prints a memory address (as curr is an object stored within memory)
   - 7.3. Same cout style as 4.1 but with “:”
   - 7.4. Loads in curr’s data (a number) to print out.
   - 7.5. Endl setup, like in 4.2
   - 7.6. Sets the current node to be current’s next node to move forward in the list