Part 2 Answer was:

buf =22*('10' + '\n')
print '-1' + '\n' + buf + hex2Int('\x40\xDA\xFF\xFF\xFF\x7F\x00\x00') + hex2Int(shellcode)

The goal was to find the buffer size and then overflow it so that we rewrite the return address of main to point to the next available space in memory, where I will have already placed the shellcode. By making the number of numbers to read -1, you would be able to read any number of numerical inputs that were not zero or more than 10 numerical characters in length and this would allow you to overflow the buffer (the vulnerability). Thus the first part of my string is '-1 ' followed by a '\n' to make a newline to denote a new input.

I found from gdb when setting breakpoint at line 46 in main, that the buffer for ibuff started at address 0x7fffffffd9e0 by using the command: 4xg ibuff. When using the command I got 2 addresses and values associated with them and when inputting numbers for the program when in gdb, I saw that the words I inputted started to be stored at 0x7fffffffd9e0 and not the other address that was outputted by that command. The return address of main which we know is somewhere on the stack was found to be 0x7fffffffda38 by using the command "info frames" and reading the address which RIP was located as this would be where the return address of main is stored.

We see that the difference in address between the start of the buffer and the RA start is 88 bytes, so the " 22*('10' + '\n')" part of my solution stored in variable buf was to overfill the buffer with 22 integer inputs so that the next integer input would start to overwrite the return address of main. Once again the each '\n' denotes a new input. The next part of the input which was "hex2Int('\x40\xDA\xFF\xFF\xFF\x7F\x00\x00')" is just the address 8 bytes in memory from the location which it is stored converted to an integer as that is what the program takes as input. This rewrites the return address of main so it will not return to main but instead return to that memory location (which is the next location). The last part of my input is "hex2Int(shellcode) which is integer equivalent of the hex shellcode. Since the return address was changed to point to the location that shellcode is now in, the main's return address just points to the shellcode so it executes the shellcode upon returning.

Basic Diagram: (All numbers are in hex)