Take Assessment: Exercise 1: Decoding Lab

Decoding Lab: Understanding a Secret Message

You have just intercepted an encoded message. The message is a sequence of bits which reads as follows in hexadecimal:

```
6363636363636363724646636F6D6F72
466D203A65693A7243646E206F54540A
5920453A54756F0A6F6F470A21643A6F
594E2020206F776F7972757445632000A
6F786F686E6963736C206765796C656B
2C3365737420346E20216F74726F5966
7565636F202061206C61676374206C6F
20206F74747865656561727632727463
6E617920680A64746F697661206646E69
2168746763002065 6C6C786178742078
6578206F72747878786361780078317 4
```

You have no idea how to decode it, but you know that your grade depends on it, so you are willing to do *anything* to extract the message. Fortunately, one of your many agents on the field has stolen the source code for the decoder. This agent (007) has put the code and the message in the file *secret.cpp*, which you can *download* from the laboratory of your technical staff (Q).

Q has noticed that the decoder takes four integers as arguments. Executing the decoder with various arguments seems to either crash the program or produce unintelligible output. It seems that the correct four integers have to be chosen in order for the program to produce the decoded message. These four integers are the "secret keys."

007 has been unable to find the keys, but from the desk of the encrypting personnel he was able to cunningly retrieve the first five characters of the unencoded message. These characters are:

From:

Assignment

Your assignment is to decode the message, and find the keys.

Reminders

This exercise is not extremely difficult. However, the strategy of trying things until something works will be ineffective. Try to understand the material in the course, particularly the following:

- Memory contains nothing but bits. Bits are interpreted as integers, characters, or instructions by the compiler, but they have no intrinsic type in memory.

- The compiler can be strong-armed into interpreting integers as characters, or even as instructions, and vice versa.
- Every group of 8 bits (a byte) has an address.
- A pointer in C is merely a stored memory address.
- The activation records for each function call are all together in memory, and they are organized in a stack that grows downwards and shrinks upwards on function calls and returns respectively.
- The return address of one function as well as the addresses of all of its local variables are allocated within one activation record.

Strategy

The designers of this decoder weren't very good. They made it possible for us to attack the keys in two independent parts. Try to break the first two keys first, and do not try to break the third and fourth keys until you have succeeded with the first two.

You can do the first part by specifying only two integer arguments when you execute the decoder. If you get the first and second keys right, a message that starts with From: will appear. This message is not the true message, but a decoy. It is useful, however, to let you know that you have indeed broken the first two keys.

In breaking the first two keys, realize that the function **process_keys12** must be somehow changing the value of the **dummy** variable. This must be so, because the variables **start** and **stride** control the extraction of the message, and they are calculated from the value of **dummy**.

In breaking the third and fourth keys, try to get the code to invoke **extract_message2** *instead* of **extract_message1**. This modification must somehow be controlled from within the function **process_keys34**.

Files

When you are done, write a brief report that includes at least the following:

1. The secret message.
2. The secret keys.
3. One paragraph describing, in your own prose, what process_keys12 does. For example, you might say that it modifies a specific program variable.
4. The meaning of the first two keys in terms of variables and addresses in the decoder program. For example, you might describe key2 by saying that its X-Y bits contain the value to which variable start is set. Or you might describe key1 by saying, for example, that it must be set equal to the number of memory addresses separating the address of two specific variables. These are only examples.
5. One paragraph describing, in your own prose, what **process_keys34** does.
6. One paragraph describing the line of source code that is executed when the first call to **process_keys34** returns.
7. The meaning of the third and fourth keys in terms of variables and addresses in the decoder program.

Be precise, clear, and brief in each of the points above. Your report should not, in any case, be longer than one page. Do not get frustrated if this takes a little longer than you expected: brief and clear text often requires more time to write than rambling prose.

Your teacher can tell you what word processors you may use to write your report. Chances are that you can write your report in a number of formats, and for simplicity's sake, you might even want to write it using Notepad.

Enjoy!