Tutor1:

The target is a program I coded in c (pure win32 code). When you run it, it tells you that it is unregistered, and that you should register it.

STEP 1:
Run the target program. And study what it looks like. Look what it does to tell you are not registered. Look when certain messages pop up. Look what messages you get when you enter fake serials.

Let's run the target program. It just tells you that you should register, nothing more. There is no option to enter a serial, no option to register.

STEP 2:
Disassemble the executable. First look if you can find the messages / nags you saw when you ran the program. Then search the program for interesting things like unregistered, thank you for registering....

Now let's look at our nice program... how was it written? I made two different dialog boxes. One which shows extensively that you are registered and thanks you very much. The other one shows you that you are unregistered and that you should register your program. Here is the source code for it.

This is the main function of the program:

```
////////////////////// Code snip ///////////////////////////
int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int);
BOOL IsProgramRegistered(void);
char RegistrationString[30];

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                   LPSTR lpCmdLine, int nCmdShow)
{
                                                // Application starts here
       if (IsProgramRegistered())            // check if program is registered
       {
         strcpy(RegistrationString, "REGISTERED" ); // copy a string registered for later
         DialogBox(hInstance, MAKEINTRESOURCE(IDD_REGDIALOG),
                                   NULL, RegDlgProc);  // show the registered dialog
       }
       else                                     // our program is unregistered
       {
         strcpy(RegistrationString, "*UN*REGISTERED !!!");  // copy a string for later use
         DialogBox(hInstance, MAKEINTRESOURCE(IDD_BADDIALOG),
                                   NULL, RegDlgProc);  // show the unregistered dialog
       }
    return FALSE;                               // give control back to windows
}
////////////////////// Code snip ///////////////////////////
```

Take a look at it and try to understand how it works ...   On the first line, the program calls a function with the name: IsProgramRegistered. And depending on the result of this program it shows you the Bad dialog (IDD_BADDIALOG)
or the registered dialog   (IDD_REGDIALOG).

From the function definition above we see that the IsProgramRegistered function returns a Boolean value. That is a true or a false. So the above code is pretty much like normal speaking language.... If program is registered then show registered dialog, else show unregistered dialog...

Now, take the case that we coded this source code, and that we wanted to make sure that the registered dialog showed
up for us ... Then we would make sure that the first piece of the code would be executed. We could do this by for example removing the if ... else   construction with the checking call to IsProgramRegistered....

then our code would look like this

```
///////////////////// Code snip /////////////////////////
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                   LPSTR lpCmdLine, int nCmdShow)
{
                                                     // Application starts here
         strcpy(RegistrationString, "REGISTERED"); // copy a string registered for later
         DialogBox(hInstance, MAKEINTRESOURCE(IDD_REGDIALOG),
                                   NULL, RegDlgProc);  // show the registered dialog

    return FALSE;                                    // give control back to windows
}
///////////////////// code snip /////////////////////////
```

If we would want to keep the if .. else intact then we could do the following..

```
///////////////////// code snip /////////////////////////
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                   LPSTR lpCmdLine, int nCmdShow)
{
                                                     // Application starts here
       if (IsProgramRegistered())           // check if program is registered
       {
         strcpy(RegistrationString, "REGISTERED"); // copy a string registered for later
         DialogBox(hInstance, MAKEINTRESOURCE(IDD_REGDIALOG),
                                   NULL, RegDlgProc);  // show the registered dialog
       }
       else                                          // our program is unregistered
         strcpy(RegistrationString, "REGISTERED"); // copy a string registered for later
         DialogBox(hInstance, MAKEINTRESOURCE(IDD_REGDIALOG),
                                  NULL, RegDlgProc);  // show the registered dialog
       }
    return FALSE;                                    // give control back to windows
}
///////////////////// Code snip /////////////////////////
```

Now let's look at the disassembly. Open the executable program up in w32dasm.   After you disassembled it, open up
the string references by choosing: Refs->String Data Reference from the Menu. When the String References Dialog opens, you will see all the strings that were used in this program. And of course you will find our two little strings: "REGISTERED" and "*UN*REGISTERED !!!" back in the   disassembly. Now double click on "REGISTERED" and
close the string references dialog box. You will land here in the disassembly:

```
///////////////////// Code snip /////////////////////////

 ADDRESS MACHINE CODE        ASSEMBLER INSTRUCTIONS

:0040106E E88DFFFFFF              call 00401000
:00401073 85C0                    test eax, eax
:00401075 741F                    je 00401096

* Possible StringData Ref from Data Obj ->"REGISTERED"
                                  |
:00401077 6844504000              push 00405044
:0040107C 68E05E4000              push 00405EE0
:00401081 E84A000000              call 004010D0
:00401086 83C408                  add esp, 00000008
:00401089 6A00                    push 00000000
:0040108B 6803104000              push 00401003
:00401090 6A00                    push 00000000

* Possible Reference to Dialog: DialogID_0065
```

```
                                  |
:00401092 6A65                    push 00000065
:00401094 EB1D                    jmp 004010B3

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401075(C)
|

* Possible StringData Ref from Data Obj ->"*UN*REGISTERED !!!"
                                  |
:00401096 6830504000              push 00405030
:0040109B 68E05E4000              push 00405EE0
:004010A0 E82B000000              call 004010D0
:004010A5 83C408                  add esp, 00000008
:004010A8 6A00                    push 00000000
:004010AA 6803104000              push 00401003
:004010AF 6A00                    push 00000000

* Possible Reference to Dialog: DialogID_007B
                                  |
:004010B1 6A7B                    push 0000007B

////////////////////// Code snip ///////////////////////////
```

What do we see here?

```
:0040106E E88DFFFFFF              call 00401000  ; call IsProgramRegistered
:00401073 85C0                    test eax, eax  ; eax equal to zero  ?
:00401075 741F                    je 00401096   ;if eax is equal to zero jump, else dont
```

There is a call made to address 401000. After this call the program checks if the value of eax is equal to zero. Depending on this test, it decides to jump or decides not to jump to address 401096. Well if we look good, we can recognize the if.. else construction in these three lines :-)

If we follow the jump to 401096 (you can do this in wdasm by double clicking on the line :00401075 and by choosing
Execute Text -> Execute Jump from the menu.) We will see that we end up at the place where the "*UN*REGISTERED!!!" string shows. The thing is, we never want to get there! So what would be the most logical thing to do? To make sure this jump was never executed. So if we removed this jump, our program would always display the Registered dialog. That would be cool.   Our program would look like the second version of the code we wrote. If IsProgramRegistered Then ShowRegisteredDialog Else ShowRegisteredDialog :-)

So, we are going to remove the jump. It is not possible to just remove it. Instead in cracking we use some other tricks to reach the same effect. The nicest to do in this case is replacing the JE instruction with NOPs. NOP stands for no operation, so if this code is executed there will nothing happen (for +/-2 milliseconds) and the program will go on with
showing the Registered dialogbox :-)

STEP 3:
Open up a copy of the program in HIEW and patch the needed bytes. Drag a copy of the program you want to crack on top of hiew and drop it. This will open the program in hiew. This can be done fastest if you have a shortcut on your desktop. Every time you want to open a program in hiew, you drag it and drop it on top of the shortcut to hiew, and the program to crack will open in hiew. Now switch back to wdasm. Go and stand on the line you want to patch. Now look in the bottom (Status Bar) of w32dasm.

It should say: Line 169 Pg. 3 of 66 Code Data @:00401075 @Offset 00000475h in File Tutor1.exe.

The important number here to remember is 475, the Offset number.
Now switch back to Hiew. Press Enter twice. This will get you into Decode mode of Hiew. (This mode can also be accessed through choosing F4). Now you are in decode mode, you can type in the offset you want to go to. Push F5

and
type in 475. When you press enter you will land straight at the place you also saw in wdasm.

You should now stand right on this line in hiew:

```
ADDRESS    MACHINE CODE                    ASSEMBLER INSTRUCTIONS

.00001075: 741F                        je       .000001096   -------- (1)
.00001077: 6844504000                  push      000405044
.0000107C: 68E05E4000                  push      000405EE0
.00001081: E84A000000                  call     .0000010D0   -------- (2)
.00001086: 83C408                      add       esp,008
.00001089: 6A00                        push      000
.0000108B: 6803104000                  push      000401003
```

If you scroll up you will also see the call and the test. Now push F3 (Edit) and then F2 (Assemble). Now we are going
to assemble a new instruction instead of the JE instruction. Now type in NOP and press enter.

```
ADDRESS    MACHINE CODE                    ASSEMBLER INSTRUCTIONS

 00000475: 90                          nop
 00000476: 1F                          pop       ds
 00000477: 6844504000                  push      000405044
 0000047C: 68E05E4000                  push      000405EE0
 00000481: E84A000000                  call      0000004D0
 00000486: 83C408                      add       esp,008
 00000489: 6A00                        push      000
 0000048B: 6803104000                  push      000401003
 00000490: 6A00                        push      000
 000
 000  +- Pentium(R) Pro Assembler -------------------------------------+
 000  ¦ pop       ds__________________________________________________ ¦
 000  +----------------------------------------------------------------+
 000
```

Hiew now assembled a nop instruction. But you see that not everything went the same as we wanted it to... The je .000001096 went away ... but the side effect of this is that we got another instruction which wasn't meant to be there and which unlike our nop instruction does something with the program memory.... If we look good, we can see that the machine code of je is 741F .... However the machine code of nop is 90 .... What happened when we assembled the nop instruction instead of the je instruction ? The nop instruction replaced the 74 of the je instruction, but the rest of the je (1F) remained intact and was reassembled into a pop ds instruction by hiew. To fix this unwanted side effect, we assemble another nop instruction which overwrites the pop ds instruction. and we have the effect we want :-)

Now your code should look like this:

```
.00001075: 90                          nop
.00001076: 90                          nop
.00001077: 6844504000                  push      000405044
.0000107C: 68E05E4000                  push      000405EE0
.00001081: E84A000000                  call     .0000010D0   -------- (1)
```

Now, to get out of the assemble mode we push escape once. And to save the modification we press F9. Press F10 to close hiew.

Now run the program and enjoy the power of letting your programs behave the way you want them to behave!