Project 8: Disassembling C on Windows (15 pts. + 10 extra credit)

What You Need

- A Windows machine, real or virtual. I used Windows 7.
- Visual Studio Express, which you installed in a previous project.
- IDA Pro Free, which you installed in a previous project.

Purpose

You will write a small C programs, compile it, and examine it in the IDA Pro disassembler to learn what it looks like in assembly language.

Starting Visual Studio Express

Click **Start**, "**VS Express 2013 for Desktop**".

Global and Local Variables

From the "Visual Studio Express 2013" menu, click **FILE**, "**New Project...**".

In the "New Project" window, on the left, expand the "**Visual C++**" container.

Click **Win32**.

In the center pane, accept the default selection of "**Win32 Console Application**".

At the bottom of the "New Project" window, type a Name of **YOURNAME-8a**, replacing "YOURNAME" with your own name. Do not use any spaces in the name.

In the "Location" line, notice the location files will be saved in--it's a subfolder of your Documents folder.

In the "New Project" window, click **OK**.

A box opens, titled "Welcome to the Win32 Application Wizard".

Click **Next**. In the next screen, accept the default settings and click **Finish**.

A window opens, showing a simple C program.

Modify this program to match the code shown in text and the image below.

Do not use the literal string "YOURNAME"--replace it with your own name.

```
// YOURNAME-8a.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

int i=1; // GLOBAL VARIABLE

int _tmain(int argc, _TCHAR* argv[])
{
int j=2; // LOCAL VARIABLE
printf("YOURNAME-8a: %d %d\n", i, j);
return 0;
}
```

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search DangVanLong

Debug x64 Local Windows Debugger GitHub Copilot

DangVanLong.cpp* What's New?

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, DangVanLong-8a!" << std::endl;
    return 0;
}
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'DangVanLong' (1 of 1 project)
  - DangVanLong
    - References
    - External Dependencies
    - Header Files
    - Resource Files
      - DangVanLong.cpp
    - Source Files

100 % No issues found Ln: 4 Ch: 57 SPC CRLF

Solution Explorer Git Changes

Ready Add to Source Control Select Repository

Type here to search ENG 1:43 AM 6/3/2024

Compiling your Program

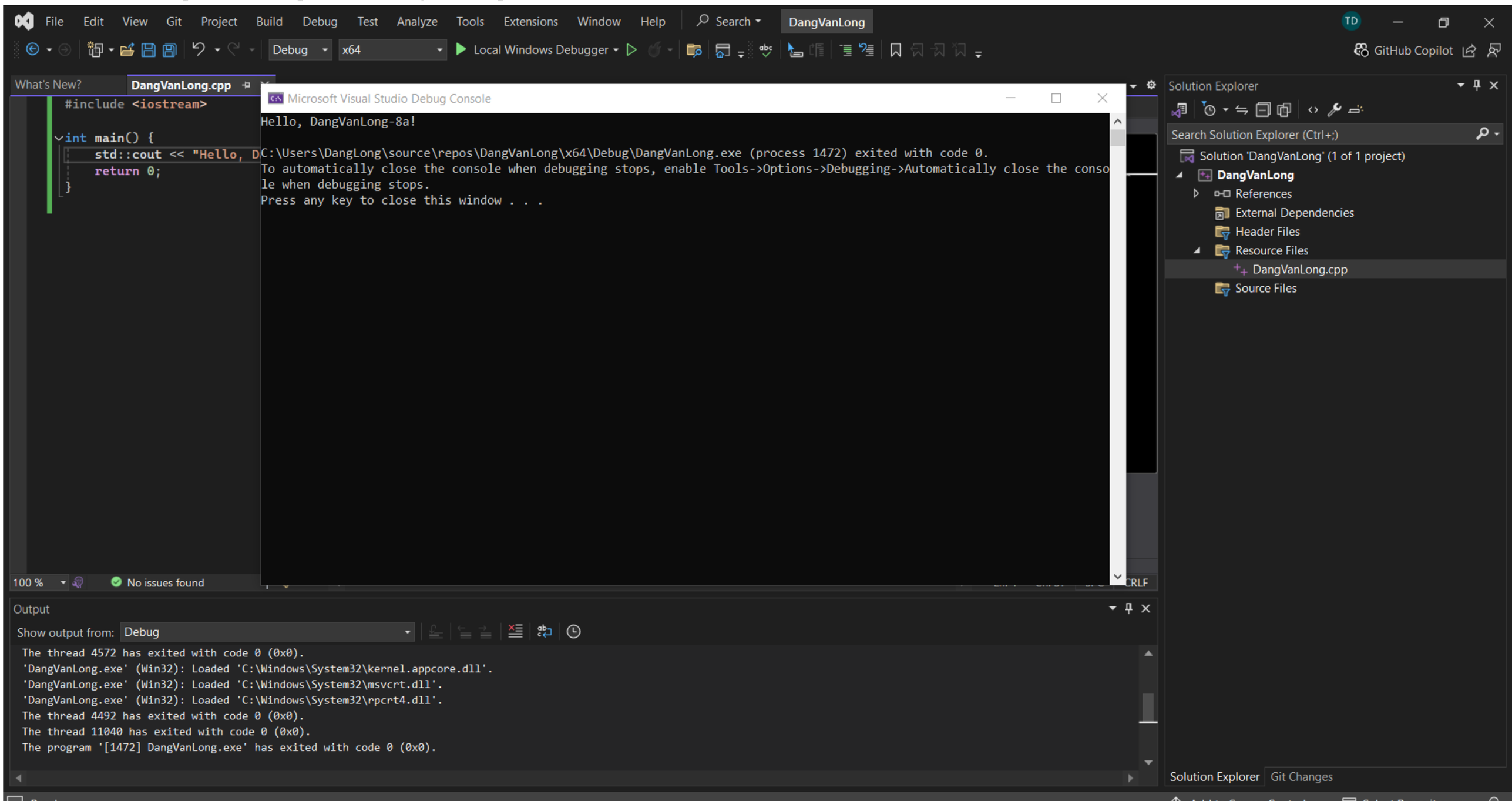
Click **BUILD**, "**Build Solution**".

You should see the message "Build: 1 succeeded" at the bottom of the window. If you see errors, you need to correct them and re-compile the program.

Running your Program

Click **DEBUG**, "**Start Without Debugging**".

A Command Prompt window opens, showing the output of "1 2", as shown below:



Disassembling the EXE

Click in the Command Prompt window, and press Enter to close it.

Minimize the Visual Studio Express window.

Start IDA Pro Free.

In the "About" box, click **OK**.

Agree to the license.

Close the Help window.

In the "Welcome to IDA!" box, click the **New** button.

In the "New disassembly database" box, double-click "**PE Executable**".

In the "Select PE Executable to disassemble" box, navigate to the folder you used to save your program. The default location is in your Documents folder, in a subfolder named "visual studio 2013\Projects".

Double-click the "**YOURNAME-8a**"folder.

Double-click the **Debug** folder.

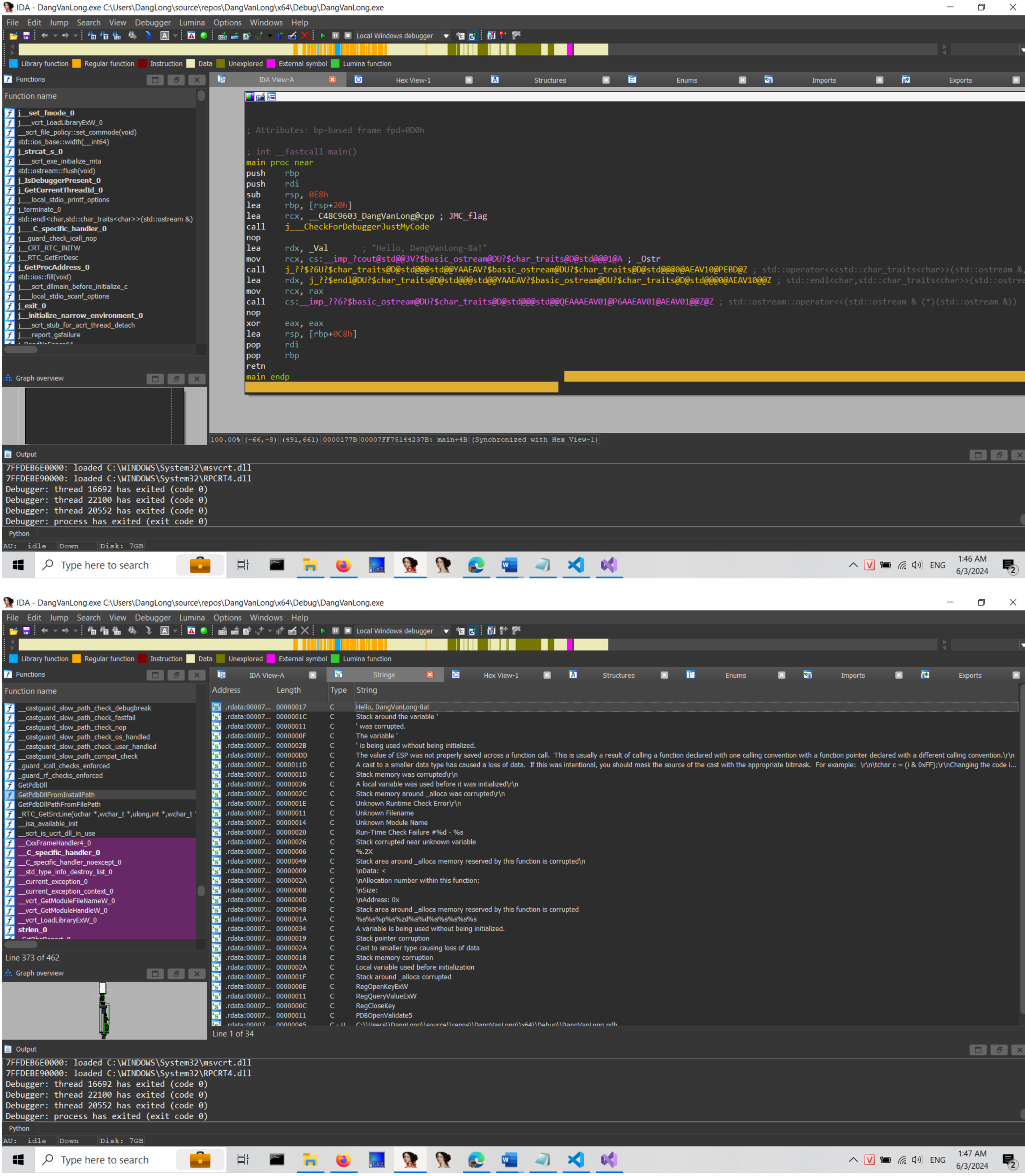
Double-click the **YOURNAME-8a.exe** file.

In the "PE Executable file loading Wizard", click **Next**, **Next**, **Finish**.

A box appears, saying this file was linked with debug information, as shown below. This is a luxury you won't often have with malware, but it's nice for this project.

Click **Yes**

IDA Pro loads the file. Unfortunately, the graph mode isn't much use, as shown below.



However, we can still find the code. Expand the **Strings** window and find "**YOURNAME-8a %d %d\n**", as shown below.

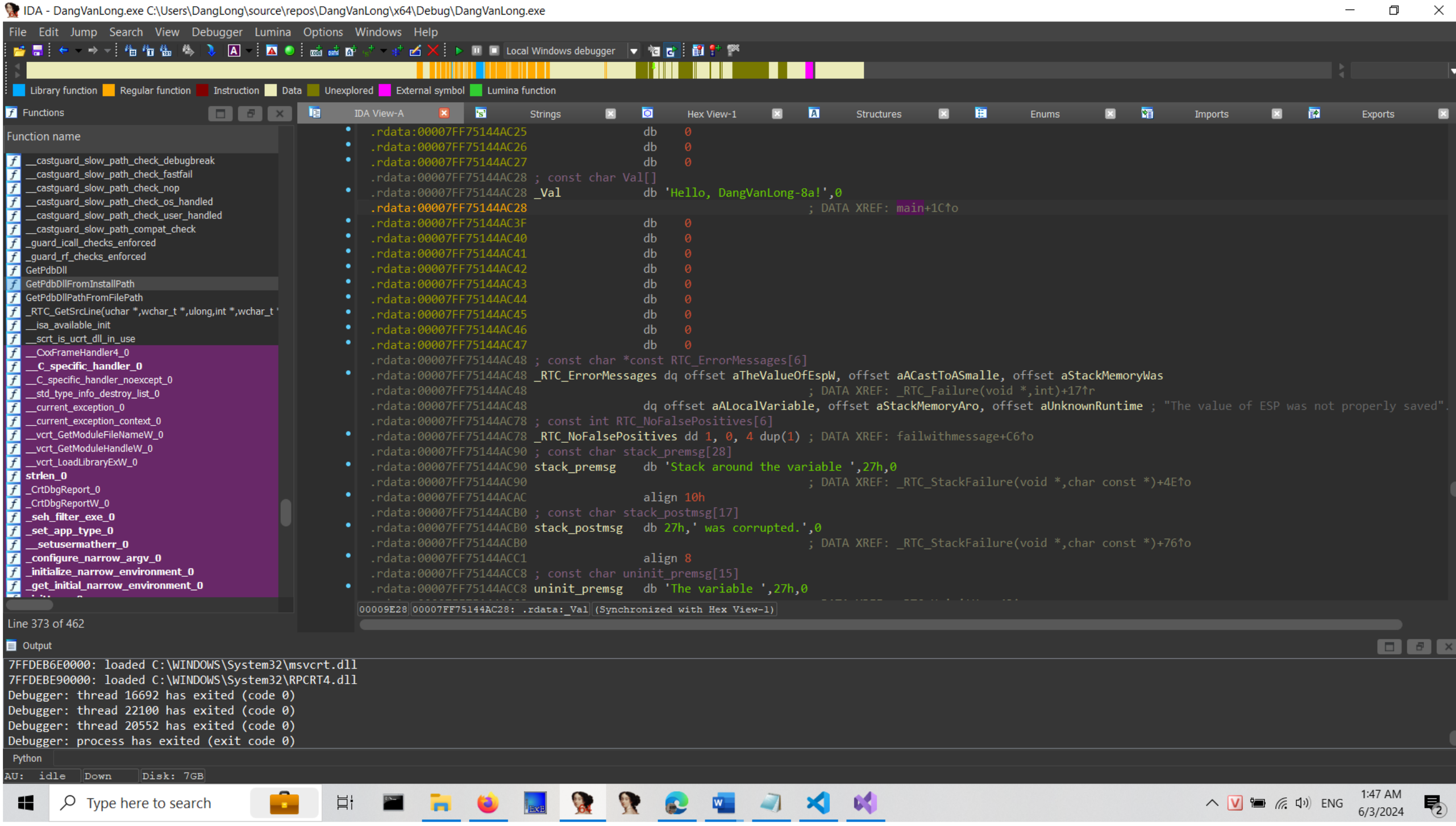
Double-click "**YOURNAME-8a %d %D\n**".

The location containing the string appears, as shown below.

This is in the .rdata section of the file, which contains data but not executable instructions.

To the right of "YOURNAME-8a" there is a "DATA XREF" comment. Hover over the address to the right of "DATA XREF", which was "**wmain+32**" when I did it.

The instructions that use this string appear in a yellow pop-up box, as shown below.



Double-click "**wmain+32**".

Now the assembly code that performs the task you wrote in C appears, as shown below.

Notice the region in the green box in the figure below.

These commands perform these C statements:

```
int j=2;
printf("YOURNAME-8a: %d %d\n", i, j);
```

The **call** at the end jumps into the printf() function.

Saving the Screen Image

Make sure you can see the command showing

```
push    offset aYourname8aDD ; "YOURNAME-8a: %d %d\n
```

as shown above. The offset value may be different, but it should contain **push** and **YOURNAME**.

On your keyboard, press the PrntScrn key.

Click **Start**, type in **PAINT**, and open Paint.

Press **Ctrl+V** to paste in the image of your desktop.

YOU MUST SUBMIT WHOLE-DESKTOP IMAGES TO GET FULL CREDIT.

Save the image with a filename of "**Proj 8a from YOUR NAME**".

Understanding Global and Local Variables

Before the **call**, the three arguments are pushed onto the stack in reverse order: first j, then i, then the string "YOURNAME-8a: %d %d\n", as detailed below.

```
mov     [ebp+var_8], 2                                  ; PUT 2 into j
mov     esi, esp
mov     eax, [ebp+var_8]                                ; PUT j into eax
push    eax                                             ; PUSH j onto the stack
mov     ecx, i                                          ; PUT i into ecx
push    ecx                                             ; PUSH i onto the stack
push    offset aYourname8aDD ; "YOURNAME-8a: %d %d\n"   ; PUSH the address of the string onto the stack
call    ds:__imp__printf                                ; CALL printf()
```

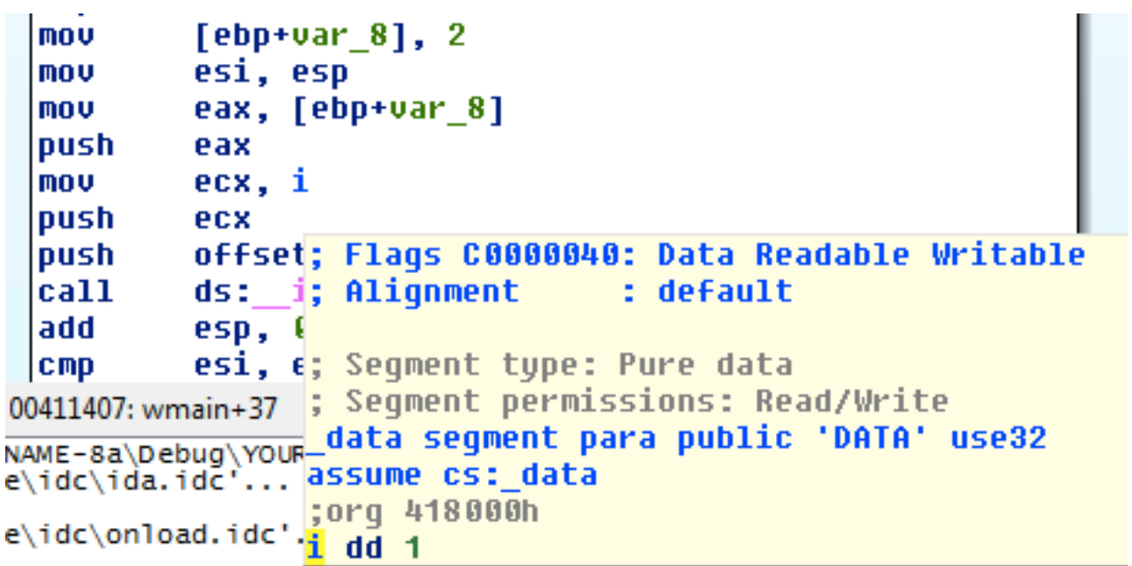
j is a local variable, so it is simply stored on the stack at the location **ebp+var_8**. It's temporary, only available to the function it's defined in.

i is a global variable, and in this case IDA was able to refer to it by name in the "mov ecx, i" instruction.

To see where i is stored, hover the mouse over it.

A yellow box pops up showing where it is stored. When I did it, it was stored at location 418000, as shown below.

This variable will be available everywhere in the program, to any function.



CHALLENGE: 10 Pts. Extra Credit

Modify the C program to contain a second global variable named x and a second local variable named y.

Compile it and disassemble it.

It must show these features, as shown below:

- **Two local variables** as shown in the top green box in the figure below: two **mov** instructions referencing stack locations such as **[ebp+var_14]**, each followed by a **push** instruction.
- **Two global variables** as shown in the lower green box in the figure below: two **mov** instructions referencing named variables such as **x**, each followed by a **push** instruction.
- **YOUR NAME** in the string.
- A **call** operation to **printf**.

```
push    ebp
mov     ebp, esp
sub     esp, 0D8h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_D8]
mov     ecx, 36h
mov     eax, 0CCCCCCCCh
rep stosd
mov     [ebp+var_8], 2
mov     [ebp+var_14], 4
mov     esi, esp
mov     eax, [ebp+var_14]
push    eax
mov     ecx, [ebp+var_8]
push    ecx
mov     edx, x
push    edx
mov     eax, i
push    eax
push    offset aYourname8bDDDD ; "YOURNAME-8b: %d %d %d %d\n"
call    ds:__imp__printf
add     esp, 14h
cmp     esi, esp
```

Turning in Your Project

Email the images to: **cnit.126sam@gmail.com** with a subject line of **Proj 8 From Your Name**, replacing Your Name with your own first and last name. Send a Cc to yourself.

Last Modified: 9-22-14 3:42 pm