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.

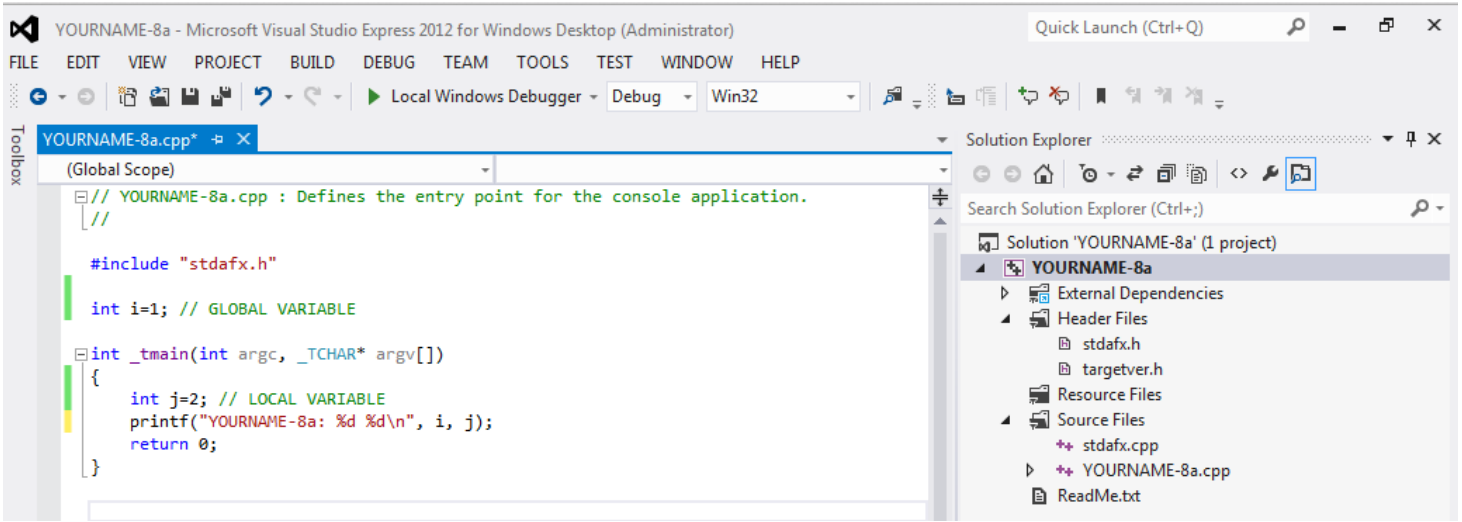
Do not use the literal string "YOURNAME"--replace it with your own name.

```
// 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;
}
```



```
// 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;
}
```

Compiling your Program

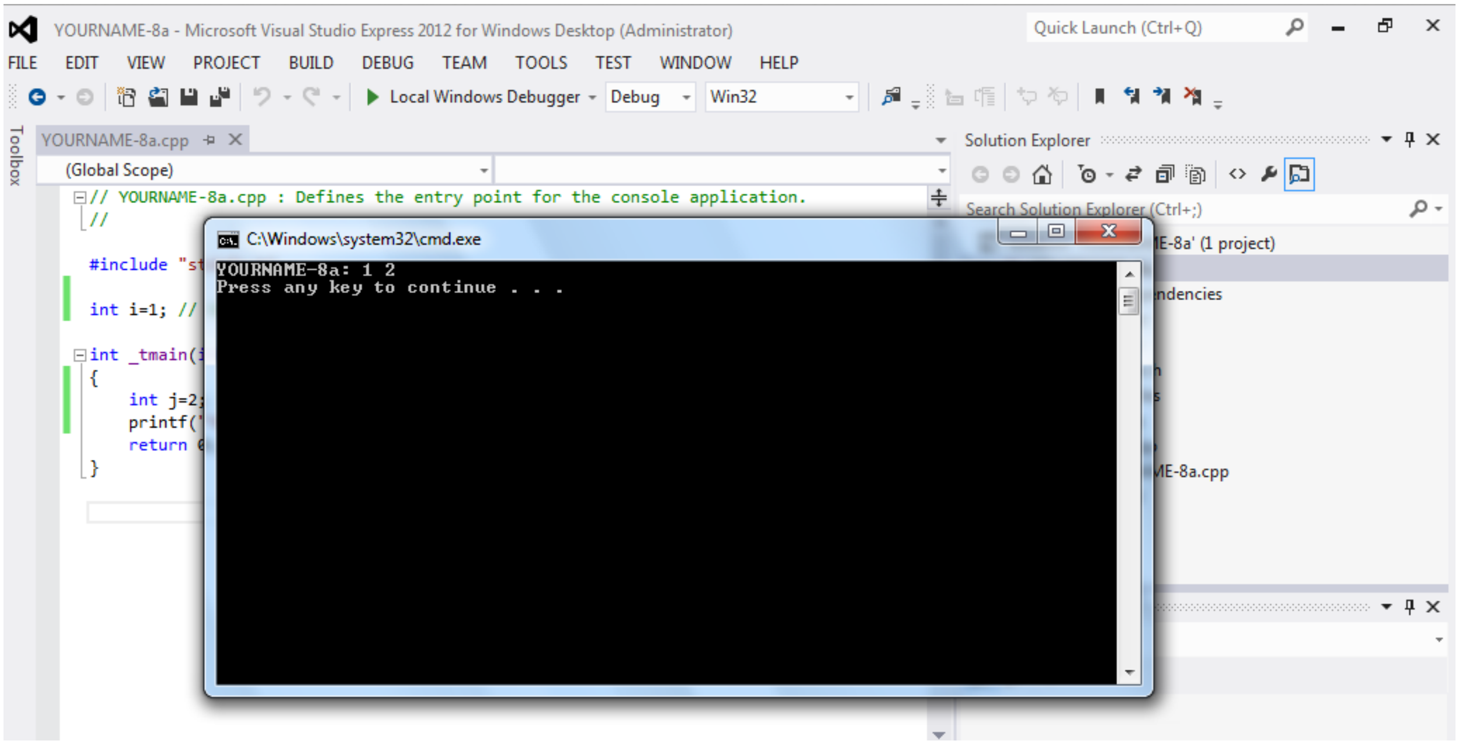
Click **BUILD**, "**Build Solution**".

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:


YOURNAME-8a - Microsoft Visual Studio Express 2012 for Windows Desktop (Administrator)
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP
Local Windows Debugger
Debug
Win32
YOURNAME-8a.cpp
(Global Scope)
Solution Explorer
C:\Windows\system32\cmd.exe

```
YOURNAME-8a: 1 2
Press any key to continue . . .
```

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.

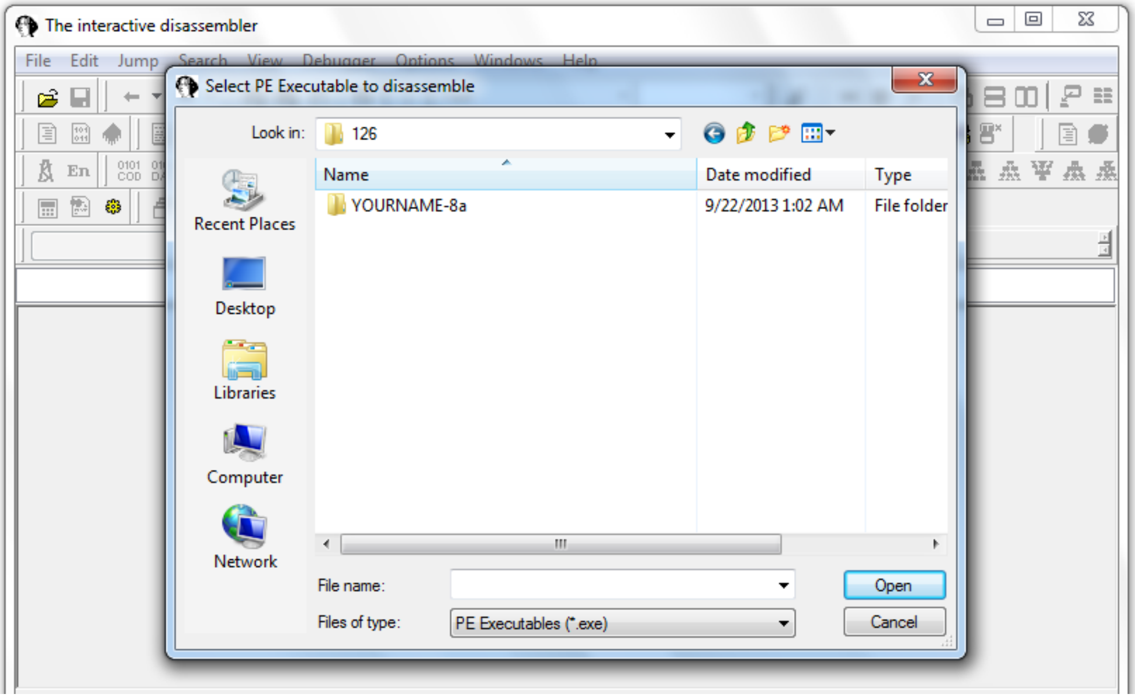
In the "New disassembly database" box, double-click "**PE Executable**".

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.



The interactive disassembler
File Edit Jump Search View Debugger Options Windows Help
Select PE Executable to disassemble
Look in: 126
Recent Places
Desktop
Libraries
Computer
Network
Open
Cancel

Name	Date modified	Type
YOURNAME-8a	9/22/2013 1:02 AM	File folder

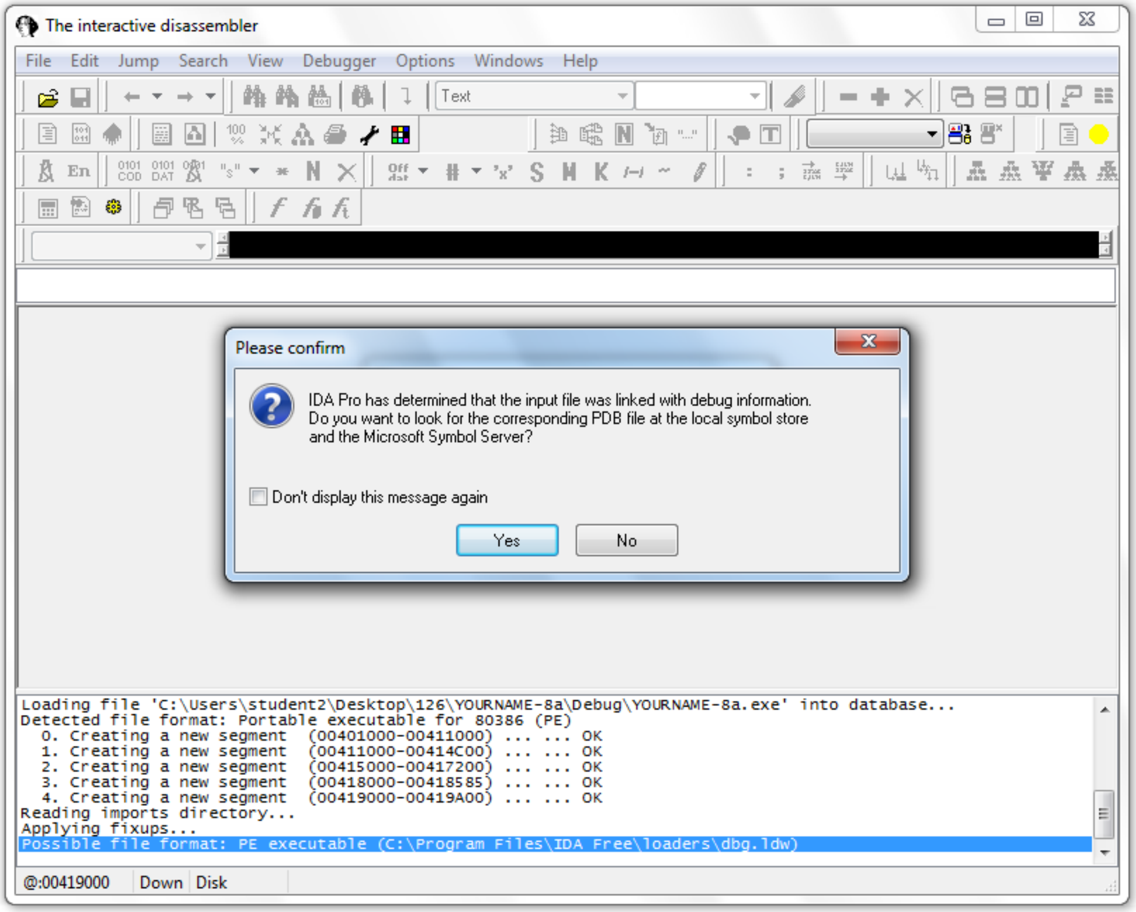
File name:

Files of type: PE Executables (*.exe)

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**

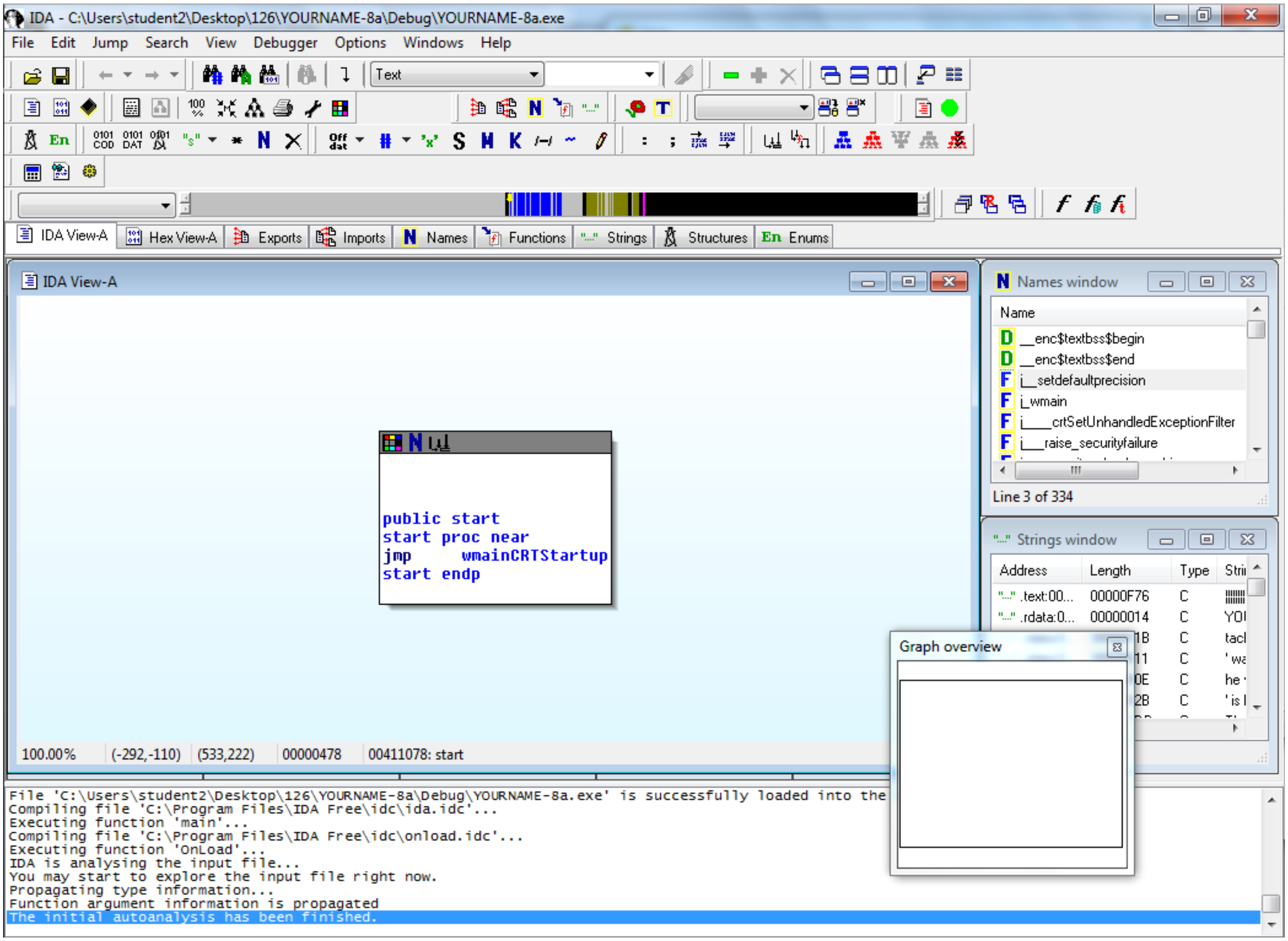

The interactive disassembler
File Edit Jump Search View Debugger Options Windows Help
Text
Please confirm
Yes
No
@:00419000
Down
Disk

IDA Pro has determined that the input file was linked with debug information. Do you want to look for the corresponding PDB file at the local symbol store and the Microsoft Symbol Server?

□ Don't display this message again

```
Loading file 'C:\Users\student2\Desktop\126\YOURNAME-8a\Debug\YOURNAME-8a.exe' into database...
Detected file format: Portable executable for 80386 (PE)
  0. Creating a new segment  (00401000-00411000) ... ... OK
  1. Creating a new segment  (00411000-00414C00) ... ... OK
  2. Creating a new segment  (00415000-00417200) ... ... OK
  3. Creating a new segment  (00418000-00418585) ... ... OK
  4. Creating a new segment  (00419000-00419A00) ... ... OK
Reading imports directory...
Applying fixups...
Possible file format: PE executable (C:\Program Files\IDA Free\loaders\dbg.ldw)
```

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


IDA - C:\Users\student2\Desktop\126\YOURNAME-8a\Debug\YOURNAME-8a.exe
File Edit Jump Search View Debugger Options Windows Help
IDA View-A
Hex View-A
Exports
Imports
Names
Functions
Strings
Structures
Enums
Names window
Strings window
Graph overview

```
public start
start proc near
jmp      wmainCRTStartup
start endp
```

100.00% (-292,-110) (533,222) 00000478 00411078: start

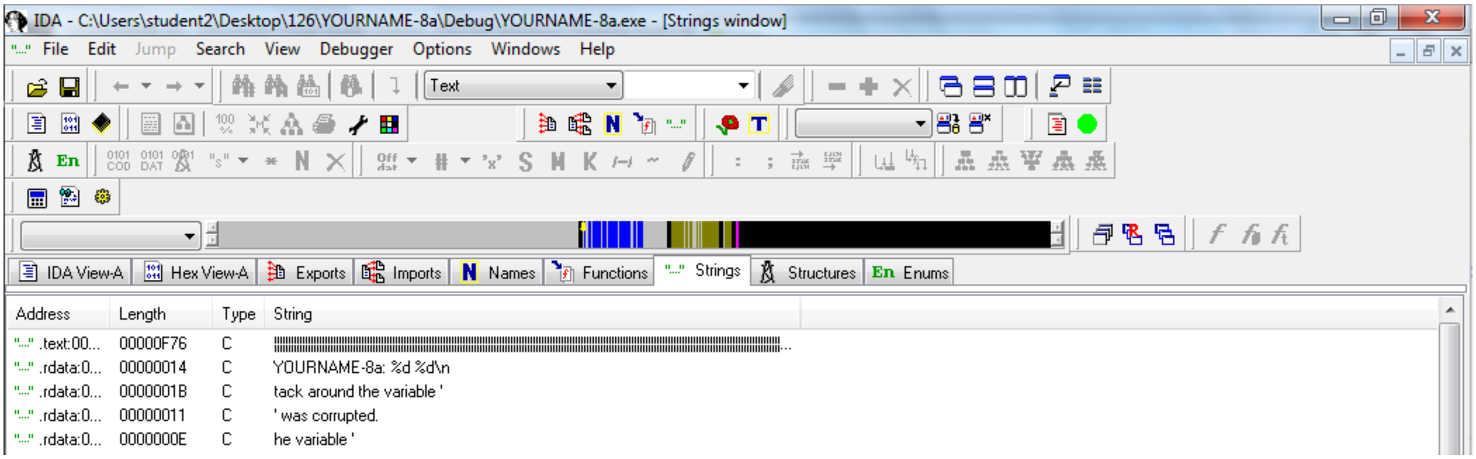
Name
D __enc$textbss$begin
D __enc$textbss$end
F j__setdefaultprecision
F j_wmain
F j___crtSetUnhandledExceptionFilter
F j___raise_securityfailure

Line 3 of 334

```
File 'C:\Users\student2\Desktop\126\YOURNAME-8a\Debug\YOURNAME-8a.exe' is successfully loaded into the
Compiling file 'C:\Program Files\IDA Free\idc\ida.idc'...
Executing function 'main'...
Compiling file 'C:\Program Files\IDA Free\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Propagating type information...
Function argument information is propagated
The initial autoanalysis has been finished.
```

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**".


IDA - C:\Users\student2\Desktop\126\YOURNAME-8a\Debug\YOURNAME-8a.exe - [Strings window]
File Edit Jump Search View Debugger Options Windows Help
IDA View-A
Hex View-A
Exports
Imports
Names
Functions
Strings
Structures
Enums

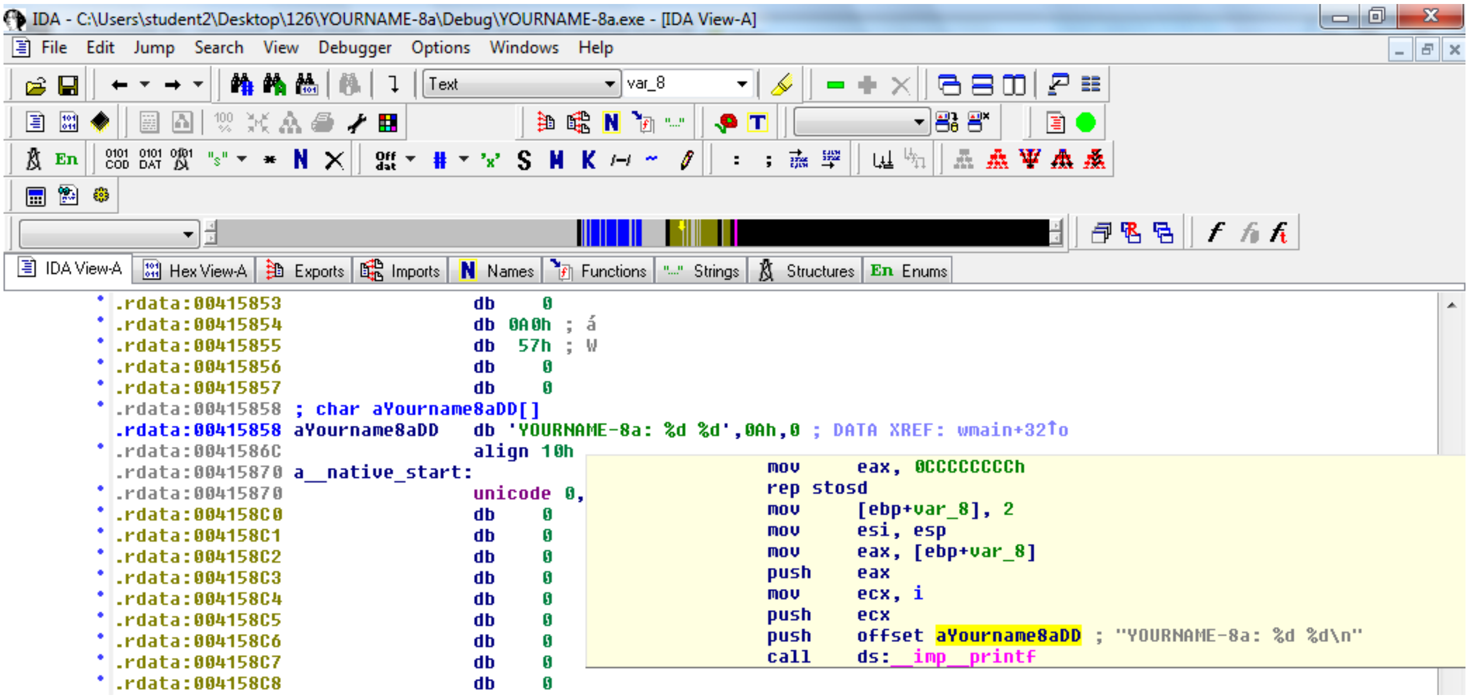
Address	Length	Type	String
"..." .text:00...	00000F76	C	||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||...
"..." .rdata:0...	00000014	C	YOURNAME-8a: %d %d\n
"..." .rdata:0...	0000001B	C	tack around the variable '
"..." .rdata:0...	00000011	C	' was corrupted.
"..." .rdata:0...	0000000E	C	he variable '

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.


IDA - C:\Users\student2\Desktop\126\YOURNAME-8a\Debug\YOURNAME-8a.exe - [IDA View-A]
File Edit Jump Search View Debugger Options Windows Help
Text
var_8
IDA View-A
Hex View-A
Exports
Imports
Names
Functions
Strings
Structures
Enums

```
.rdata:00415853                 db    0
.rdata:00415854                 db  0A0h ; á
.rdata:00415855                 db  57h ; W
.rdata:00415856                 db    0
.rdata:00415857                 db    0
.rdata:00415858 ; char aYourname8aDD[]
.rdata:00415858 aYourname8aDD   db 'YOURNAME-8a: %d %d',0Ah,0 ; DATA XREF: wmain+32↑o
.rdata:0041586C                 align 10h
.rdata:00415870 a__native_start:
.rdata:00415870                 unicode 0,
.rdata:004158C0                 db    0
.rdata:004158C1                 db    0
.rdata:004158C2                 db    0
.rdata:004158C3                 db    0
.rdata:004158C4                 db    0
.rdata:004158C5                 db    0
.rdata:004158C6                 db    0
.rdata:004158C7                 db    0
.rdata:004158C8                 db    0
```

```
mov     eax, 0CCCCCCCCh
rep stosd
mov     [ebp+var_8], 2
mov     esi, esp
mov     eax, [ebp+var_8]
push    eax
mov     ecx, i
push    ecx
push    offset aYourname8aDD ; "YOURNAME-8a: %d %d\n"
call    ds:__imp__printf
```

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

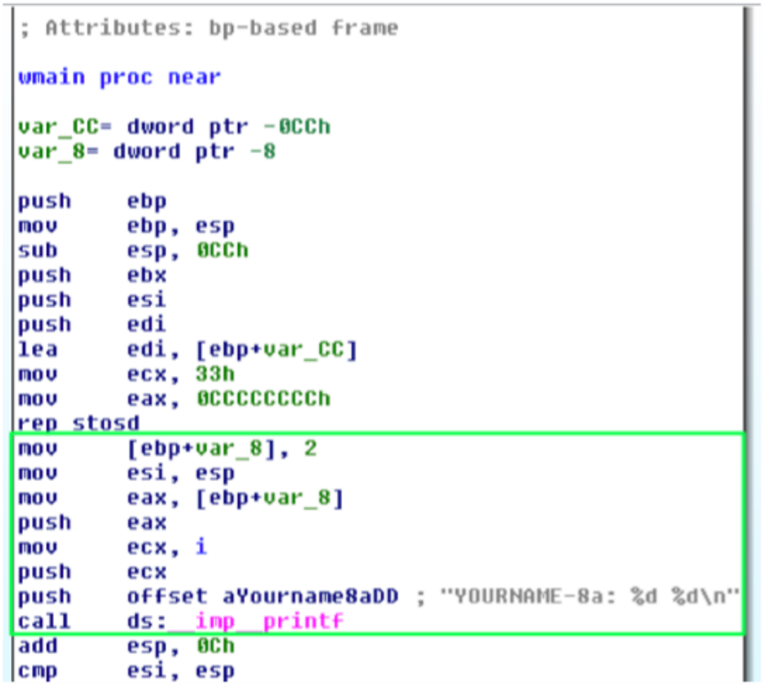
Now the assembly code that performs the task you wrote in C appears, as shown below.

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.



```
; Attributes: bp-based frame

wmain proc near

var_CC= dword ptr -0CCh
var_8= dword ptr -8

push    ebp
mov     ebp, esp
sub     esp, 0CCh
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_CC]
mov     ecx, 33h
mov     eax, 0CCCCCCCCh
rep stosd
mov     [ebp+var_8], 2
mov     esi, esp
mov     eax, [ebp+var_8]
push    eax
mov     ecx, i
push    ecx
push    offset aYourname8aDD ; "YOURNAME-8a: %d %d\n"
call    ds:__imp__printf
add     esp, 0Ch
cmp     esi, esp
```

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()
```

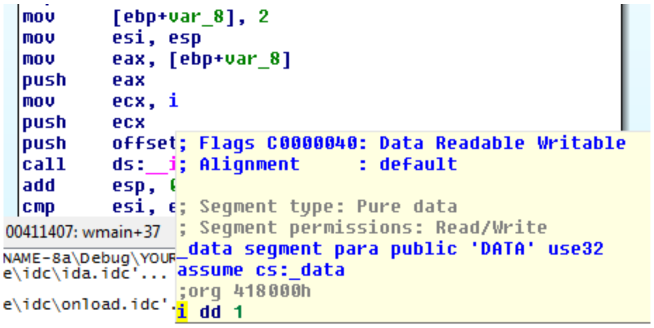
j is a local variable, so it is simply stored on the stack at the location **ebp+var_8**. It's temporary, only available to the function it's defined in.

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.



```
mov     [ebp+var_8], 2
mov     esi, esp
mov     eax, [ebp+var_8]
push    eax
mov     ecx, i
push    ecx
push    offset; Flags C0000040: Data Readable Writable
call    ds:__i; Alignment     : default
add     esp, (
cmp     esi, e; Segment type: Pure data
00411407: wmain+37 ; Segment permissions: Read/Write
NAME-8a\Debug\YOUR_data segment para public 'DATA' use32
e\idc\ida.idc'... assume cs:_data
                  ;org 418000h
e\idc\onload.idc'.i dd 1
```

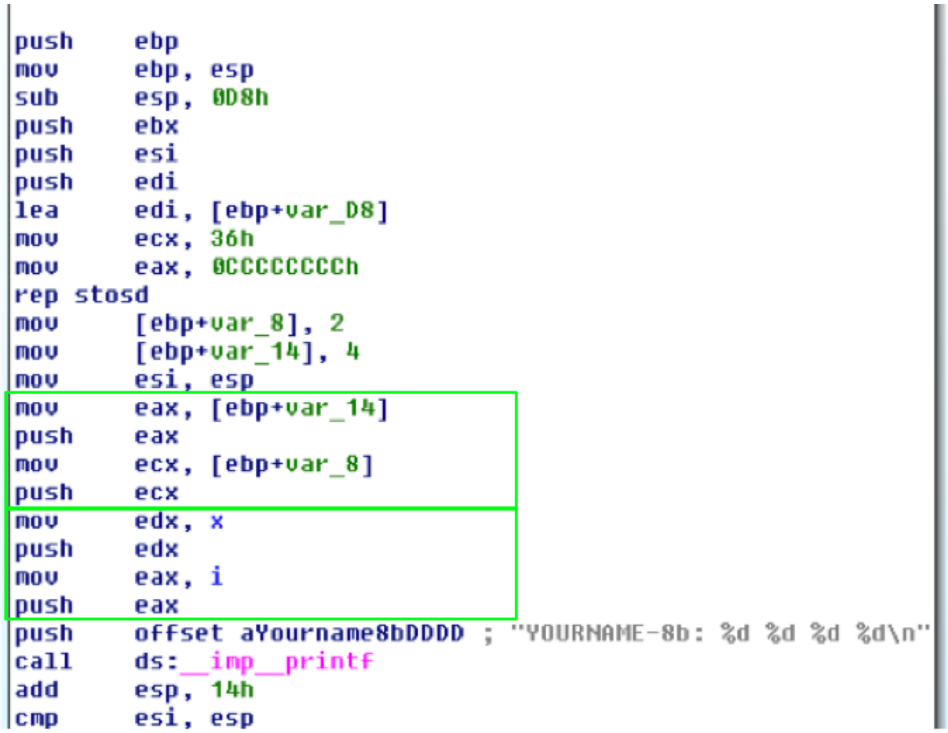
CHALLENGE: 10 Pts. Extra Credit

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