PRACTICA # 9

NOMBRE: JHONNY MARTINEZ FLORES CI:8616626 RU:88682

1) ¿Qué es el 'stack' en el contexto del lenguaje ensamblador y cómo se utiliza? (10 pts)

El stack (pila) en el contexto del lenguaje ensamblador es una estructura de datos que funciona bajo el principio de Last In, First Out (LIFO), es decir, el último elemento en ser insertado es el primero en salir. Es utilizado principalmente para almacenar información temporal, como direcciones de retorno, variables locales o parámetros de funciones.

- Uso en ensamblador:
  - PUSH: Inserta un valor en el stack.
  - POP: Extrae un valor del stack.
  - CALL y RET: Usan el stack para almacenar la dirección de retorno cuando se llama a una subrutina.
  - ESP (Stack Pointer): Registro que apunta al tope del stack.

EJEMPLO

```
PUSH AX    ; Guarda el valor del registro AX en el stack
MOV AX, 5  ; Cambia el valor de AX
POP AX     ; Recupera el valor original de AX desde el stack
```

2) Escenario práctico donde el ensamblador es más ventajoso que un lenguaje de alto nivel (10 pts)

El ensamblador es preferido cuando se necesita **control absoluto sobre el hardware** o un **alto rendimiento**. Ejemplo:

Diseño de sistemas embebidos:
En microcontroladores de recursos limitados (como los usados en sensores IoT), el ensamblador permite optimizar el uso de memoria y ciclos de reloj. Esto es crítico para garantizar la eficiencia y el funcionamiento en tiempo real, que sería difícil de lograr con lenguajes de alto nivel debido al overhead de abstracción.

3) Explicación del código ensamblador (20 pts)

El código realiza operaciones aritméticas y transfiere valores entre registros:

```
MOV AX, 5      ; Línea 1
```

- **Función**: Mueve el valor inmediato 5 al registro AX.
- **Propósito**: Inicializa el registro AX con el valor 5.

```
MOV BX, 10     ; Línea 2
```

- **Función**: Mueve el valor inmediato 10 al registro BX.
- **Propósito**: Inicializa el registro BX con el valor 10.

ADD AX, BX ; Línea 3

- **Función**: Suma el contenido de BX al contenido de AX y almacena el resultado en AX.
- **Propósito**: Realiza la operación AX = AX + BX, que en este caso es 5 + 10 = 15.

MOV CX, AX ; Línea 4

- **Función**: Copia el contenido de AX al registro CX.
- **Propósito**: Almacena el resultado de la suma (15) en CX.

Resumen: Este código realiza la suma de los valores 5 y 10, almacenando el resultado (15) en el registro CX.

4) Explicación detallada de cómo funcionan los compiladores (10 pts)

Un **compilador** es un programa que traduce el código fuente escrito en un lenguaje de alto nivel (como C, Python o Java) a un lenguaje de bajo nivel (como lenguaje ensamblador o código máquina) que pueda ser ejecutado por un procesador.

Etapas principales del funcionamiento de un compilador:

1. **Análisis Léxico**:
   - Divide el código fuente en unidades más pequeñas llamadas **tokens** (palabras clave, identificadores, operadores, etc.).
   - Por ejemplo, el código int x = 5; se divide en los tokens int, x, =, 5, y ;.
2. **Análisis Sintáctico**:
   - Verifica que la secuencia de tokens cumple con las reglas gramaticales del lenguaje de programación.
   - Construye un **árbol de sintaxis abstracta (AST)** que representa la estructura jerárquica del código.
3. **Análisis Semántico**:
   - Comprueba que el código tiene sentido en términos del lenguaje (semántica).
   - Por ejemplo, verifica que las variables estén declaradas antes de ser usadas y que las operaciones sean válidas para los tipos de datos involucrados.
4. **Generación de Código Intermedio**:
   - Convierte el AST en una representación intermedia, que es más fácil de optimizar y traducir a código máquina.
   - Ejemplo: Puede generar un código tipo tres direcciones como t1 = 5; t2 = t1 + 10;.
5. **Optimización del Código**:
   - Realiza mejoras en el código intermedio para que sea más eficiente sin cambiar su comportamiento.

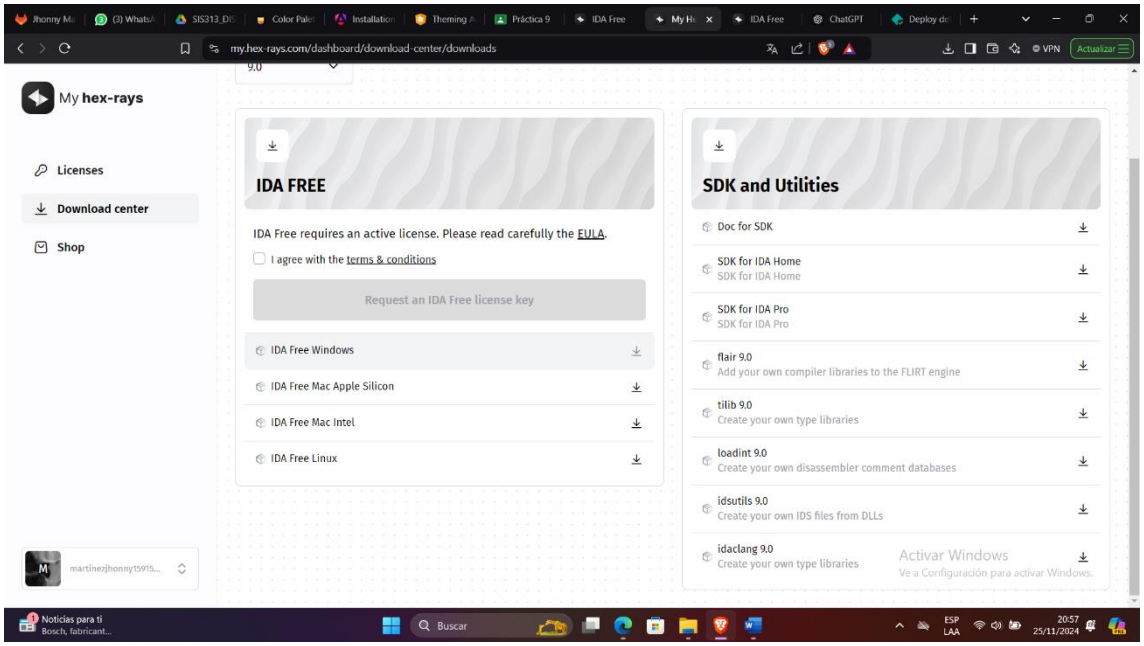
- Por ejemplo, elimina instrucciones redundantes o reordena operaciones para aprovechar mejor la arquitectura del procesador.

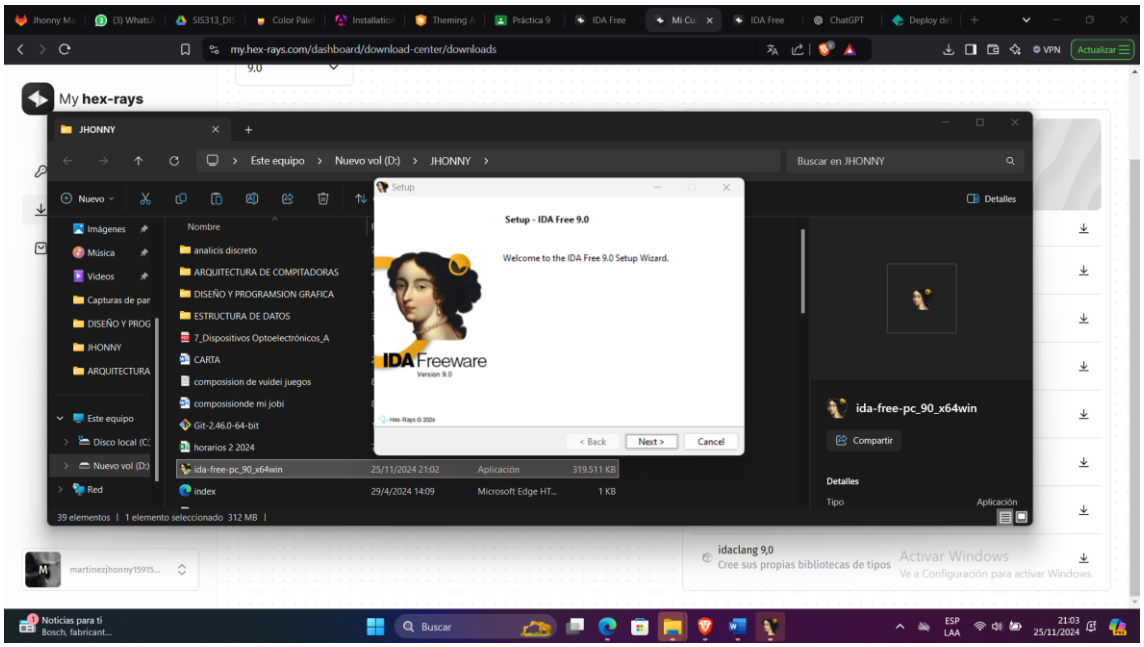
6. **Generación de Código de Máquina**:
   - Traduce el código optimizado a instrucciones específicas del procesador en un formato ejecutable. ✓
   - Por ejemplo, una suma puede traducirse a una instrucción ensambladora como ADD AX, BX.
7. **Ensamblado y Enlace**:
   - **Ensamblado**: Convierte las instrucciones del lenguaje ensamblador a código binario. ✓
   - **Enlace**: Une diferentes partes del programa (módulos o bibliotecas externas) para formar el ejecutable final.

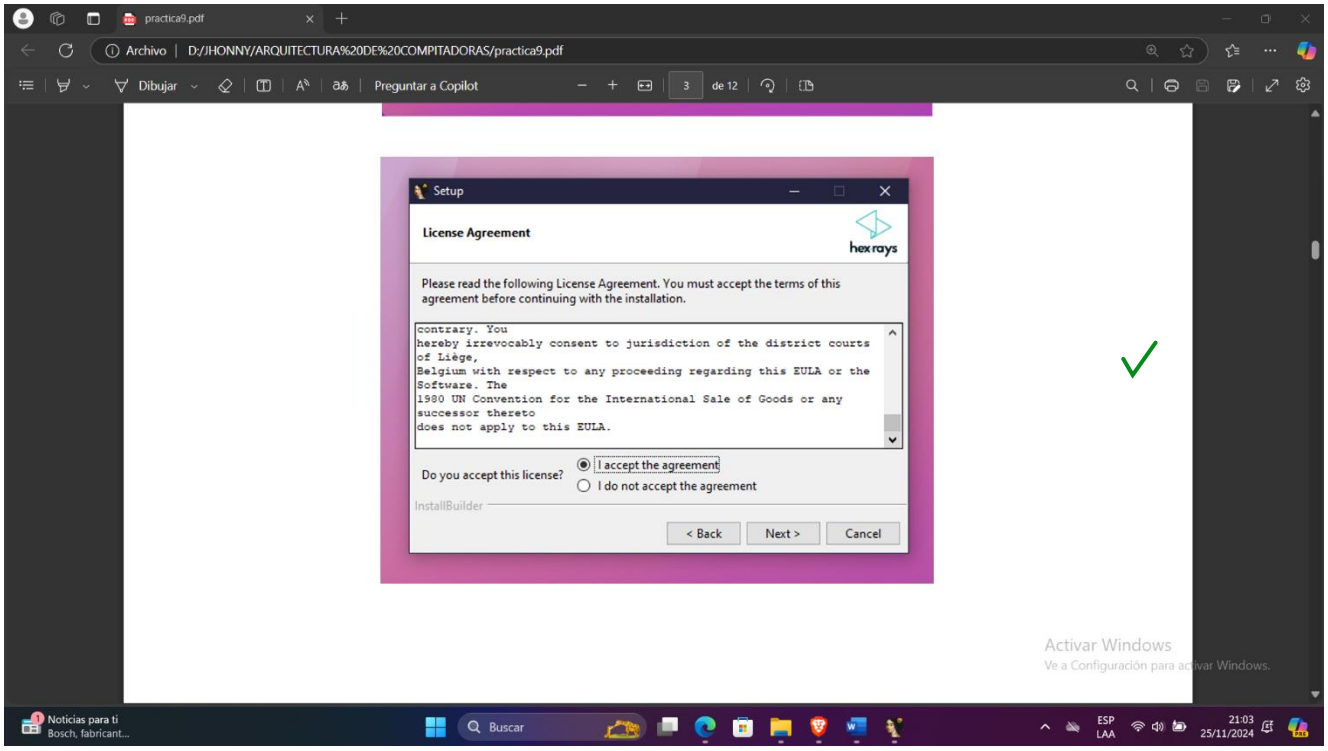
Funcionamiento General:

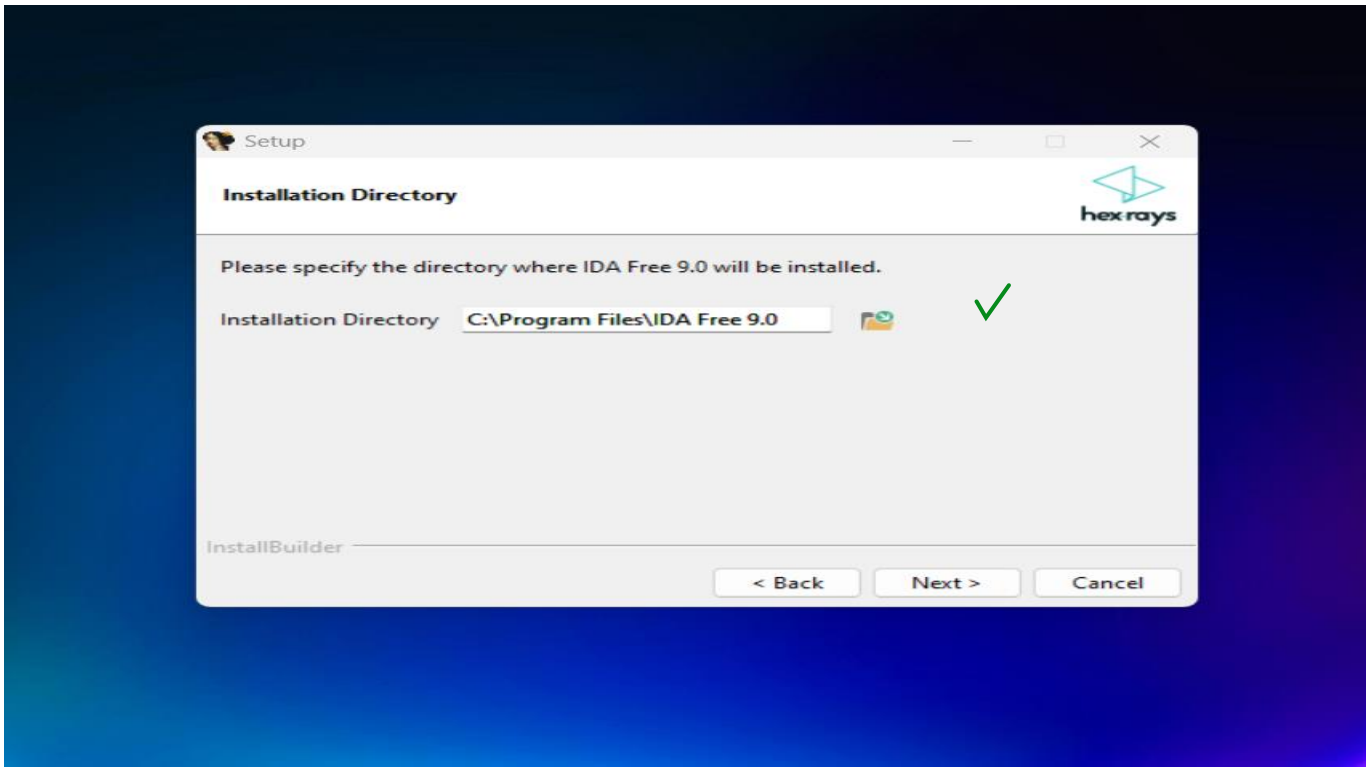
El compilador realiza todas estas etapas de manera automática. El desarrollador simplemente proporciona el código fuente, y el compilador devuelve un archivo ejecutable que puede ser ejecutado por el sistema operativo.

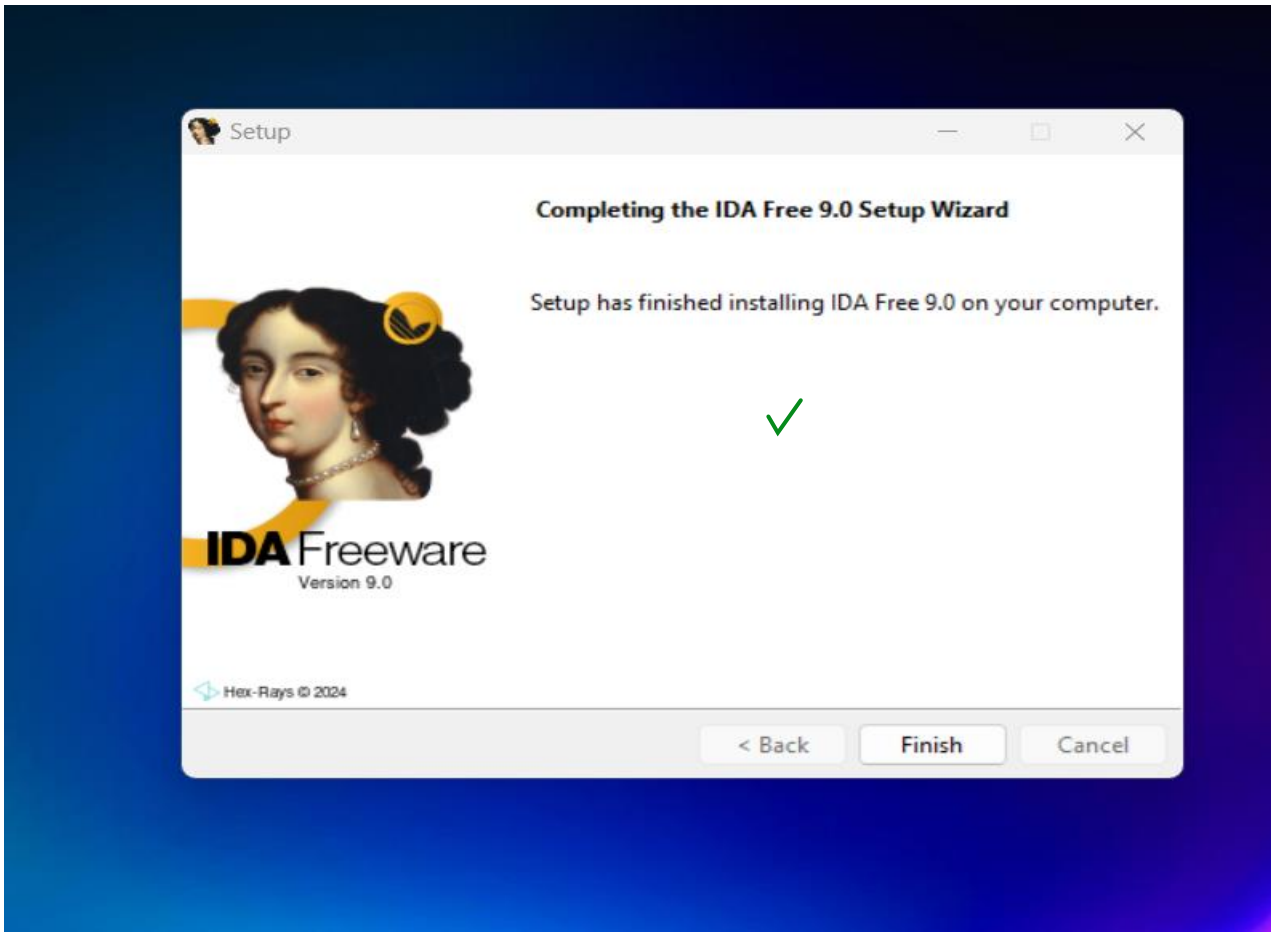
5) Realizar sus propias capturas de pantalla del siguiente procedimiento: (50 pts) ✓



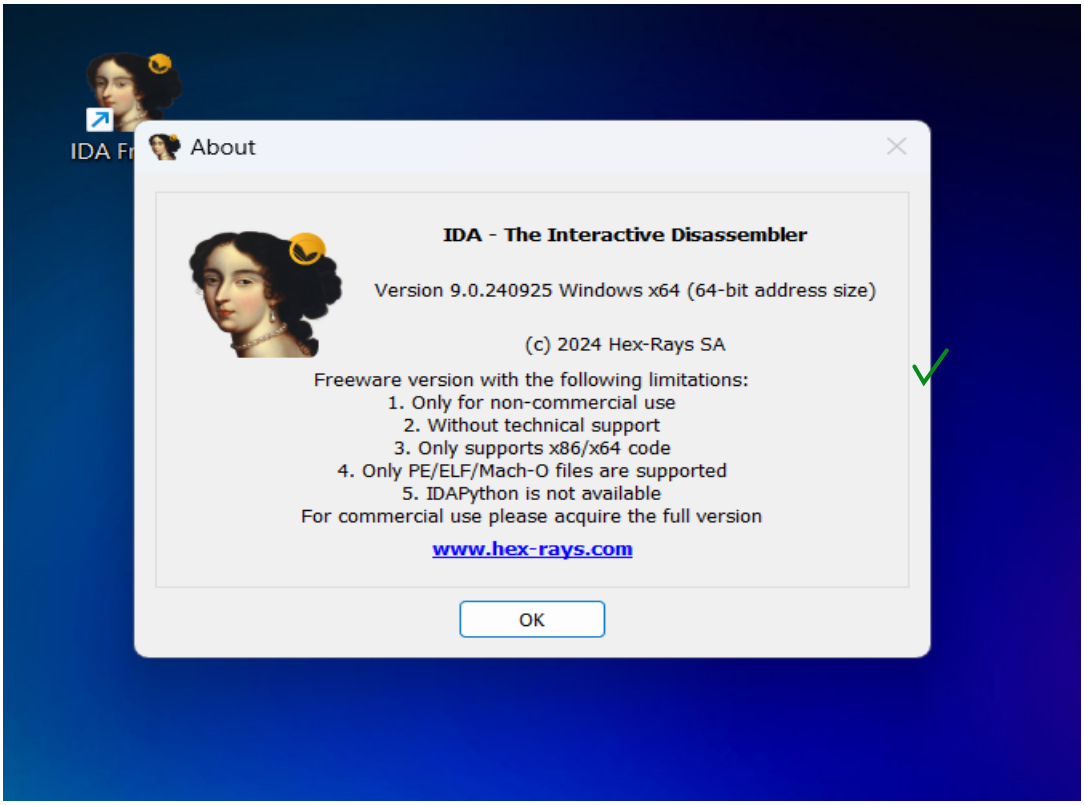

my.hex-rays.com/dashboard/download-center/downloads
My hex-rays
JHONNY
Este equipo › Nuevo vol (D:) › JHONNY ›
Buscar en JHONNY
Setup - IDA Free 9.0
Welcome to the IDA Free 9.0 Setup Wizard.
IDA Freeware
< Back
Next >
Cancel
ida-free-pc_90_x64win
Compartir
Detalles
idaclang 9.0
Cree sus propias bibliotecas de tipos
Activar Windows
Ve a Configuración para activar Windows.

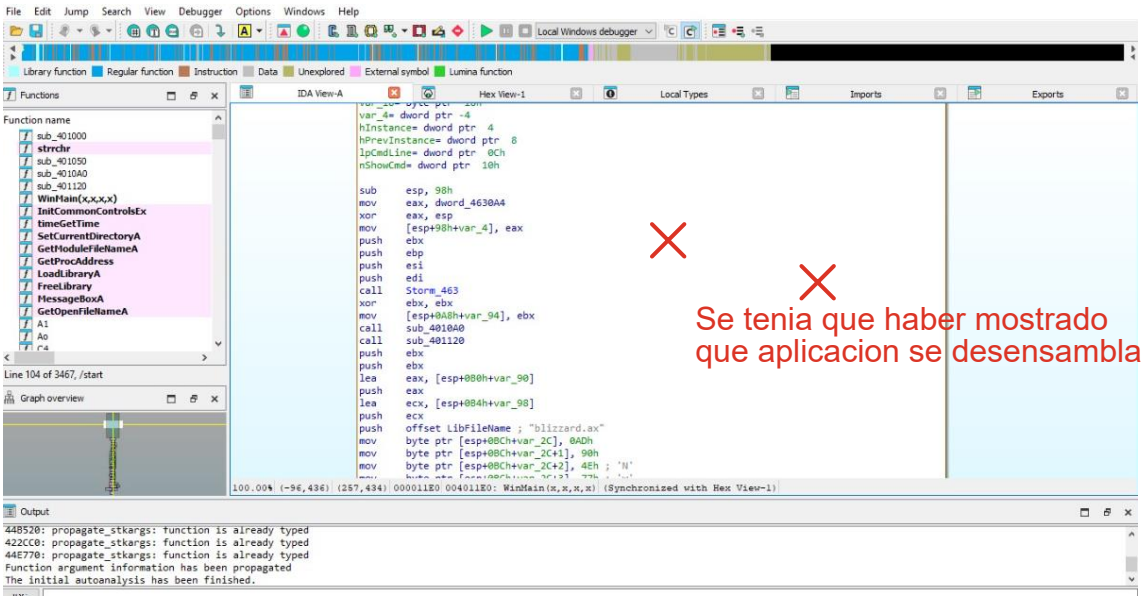

practica9.pdf
Archivo | D:/JHONNY/ARQUITECTURA%20DE%20COMPITADORAS/practica9.pdf
Dibujar
Preguntar a Copilot
3
de 12
Setup
License Agreement
hex rays
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.
contrary. You
hereby irrevocably consent to jurisdiction of the district courts of Liège,
Belgium with respect to any proceeding regarding this EULA or the Software. The
1980 UN Convention for the International Sale of Goods or any successor thereto
does not apply to this EULA.
Do you accept this license?
I accept the agreement
I do not accept the agreement
InstallBuilder
< Back
Next >
Cancel
Activar Windows
Ve a Configuración para activar Windows.

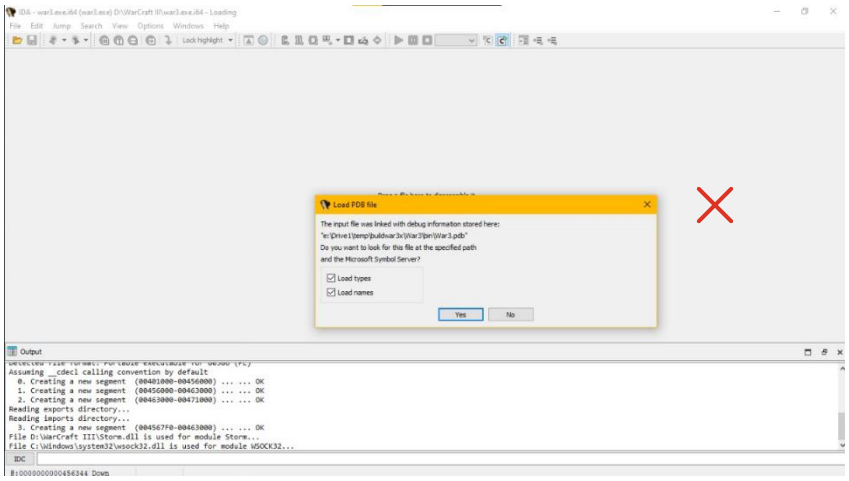

Setup
Installation Directory
hex rays
Please specify the directory where IDA Free 9.0 will be installed.
Installation Directory
C:\Program Files\IDA Free 9.0
InstallBuilder
< Back
Next >
Cancel

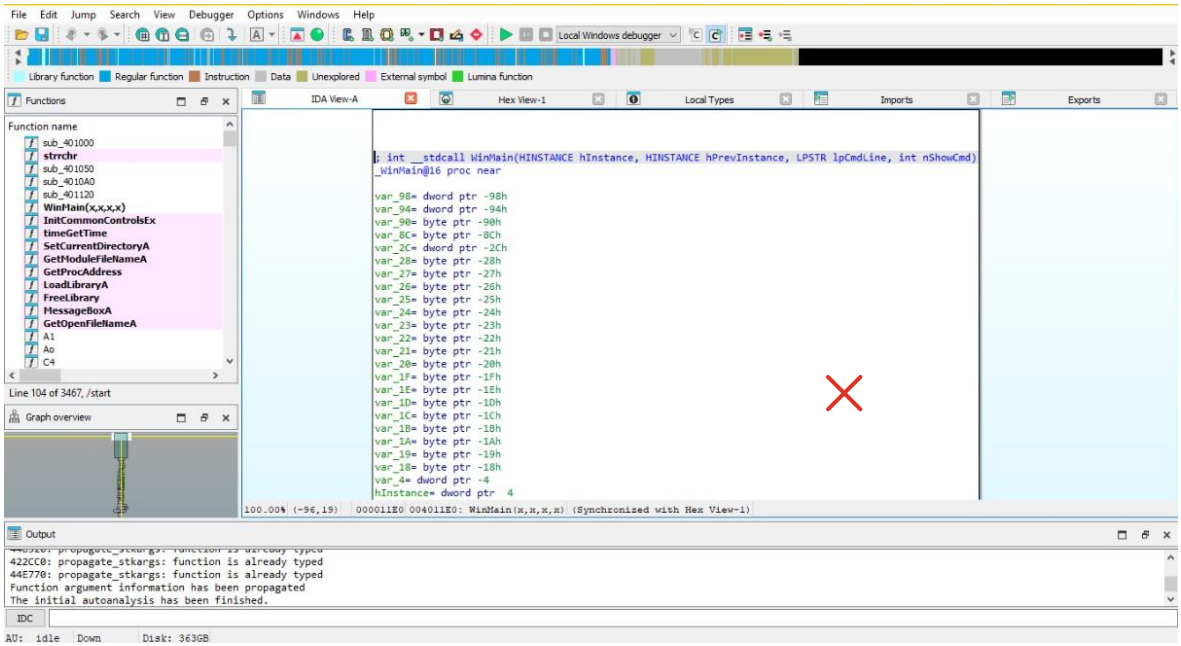

Setup
Completing the IDA Free 9.0 Setup Wizard
Setup has finished installing IDA Free 9.0 on your computer.
IDA Freeware
Version 9.0
Hex-Rays © 2024
< Back
Finish
Cancel

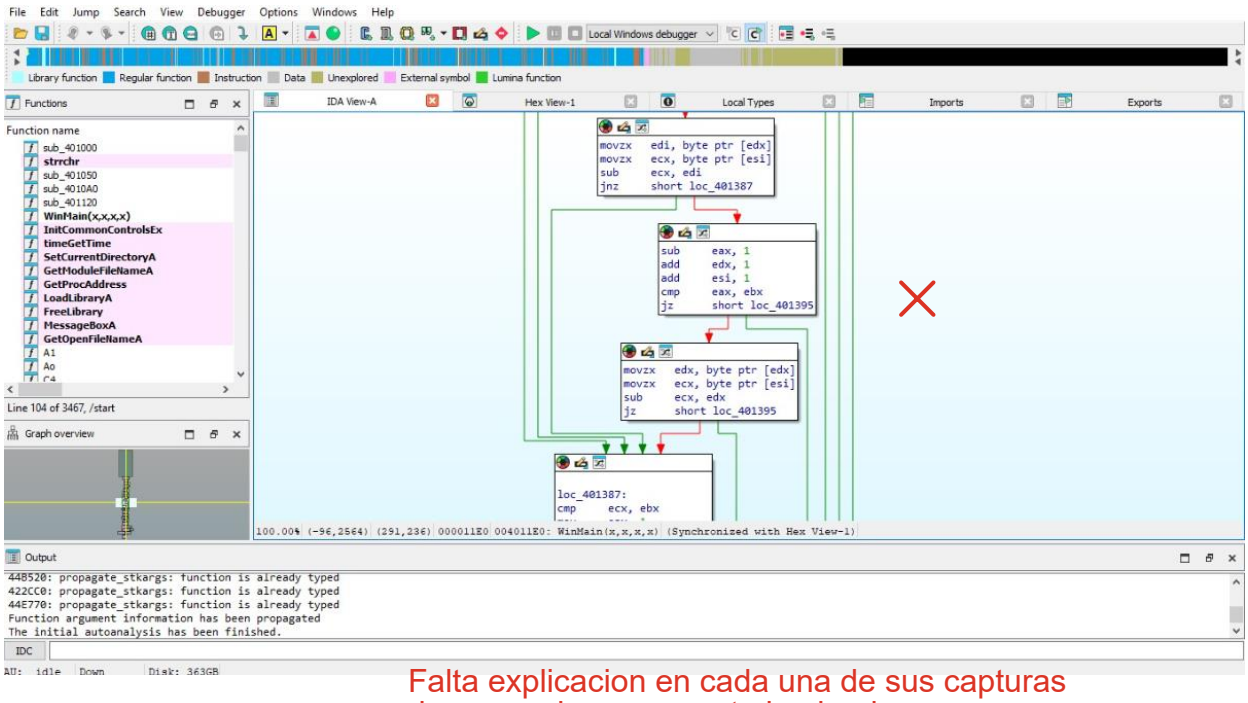

IDA Free 9.0


About
IDA - The Interactive Disassembler
Version 9.0.240925 Windows x64 (64-bit address size)
(c) 2024 Hex-Rays SA
Freeware version with the following limitations:
1. Only for non-commercial use
2. Without technical support
3. Only supports x86/x64 code
4. Only PE/ELF/Mach-O files are supported
5. IDAPython is not available
For commercial use please acquire the full version
www.hex-rays.com
OK


Se tenia que haber mostrado
que aplicacion se desensamblará


Load PDB file
The input file was linked with debug information stored here:
Do you want to look for this file at the specified path
and the Microsoft Symbol Server?
Load types
Load names
Yes
No
Output


File Edit Jump Search View Debugger Options Windows Help
Local Windows debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions
Function name
sub_401000
strrchr
sub_401050
sub_4010A0
sub_401120
WinMain(x,x,x,x)
InitCommonControlsEx
timeGetTime
SetCurrentDirectoryA
GetModuleFileNameA
GetProcAddress
LoadLibraryA
FreeLibrary
MessageBoxA
GetOpenFileNameA
A1
A6
C4
Line 104 of 3467, /start
Graph overview
IDA View-A
Hex View-1
Local Types
Imports
Exports
; int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
_WinMain@16 proc near
var_98= dword ptr -98h
var_94= dword ptr -94h
var_90= byte ptr -90h
var_8C= byte ptr -8Ch
var_2C= dword ptr -2Ch
var_28= byte ptr -28h
var_27= byte ptr -27h
var_26= byte ptr -26h
var_25= byte ptr -25h
var_24= byte ptr -24h
var_23= byte ptr -23h
var_22= byte ptr -22h
var_21= byte ptr -21h
var_20= byte ptr -20h
var_1F= byte ptr -1Fh
var_1E= byte ptr -1Eh
var_1D= byte ptr -1Dh
var_1C= byte ptr -1Ch
var_1B= byte ptr -18h
var_1A= byte ptr -1Ah
var_19= byte ptr -19h
var_18= byte ptr -18h
var_4= dword ptr -4
hInstance= dword ptr 4
100.00% (-96,19) 00001LE0 004011E0: WinMain(x,x,x,x) (Synchronized with Hex View-1)
Output
422CC0: propagate_stkargs: function is already typed
44E770: propagate_stkargs: function is already typed
Function argument information has been propagated
The initial autoanalysis has been finished.
IDC
AU: idle Down Disk: 363GB



Falta explicacion en cada una de sus capturas de que es lo que se esta haciendo