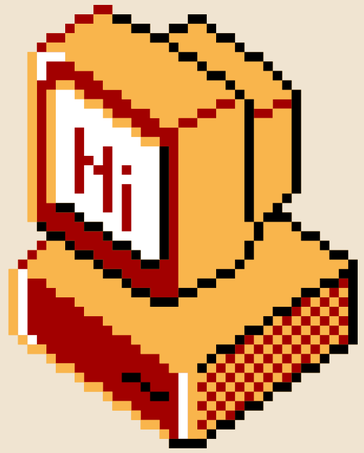
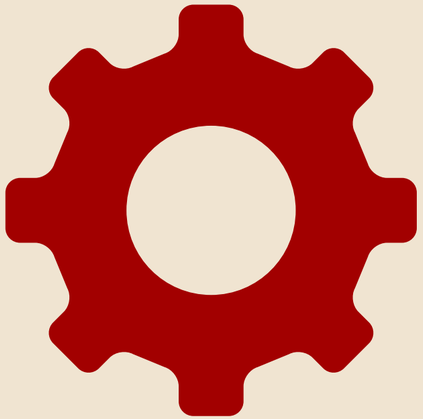




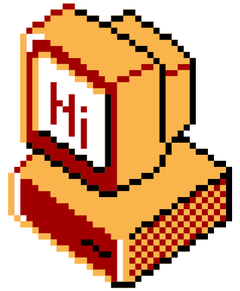
PERTH SOCIALWARE

0x03:

Reverse Engineering Workshop

Part 2

$ ~/: groups "socialware"

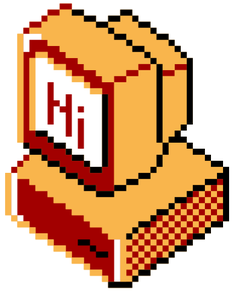


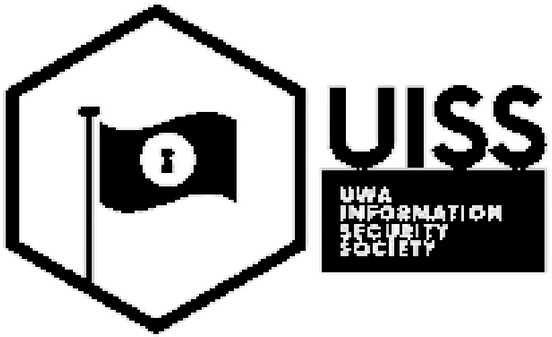
Welcome!

About & Aims

Enjoy!

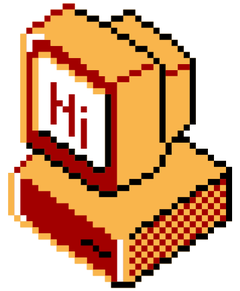
$ ~/: groups "socialware"





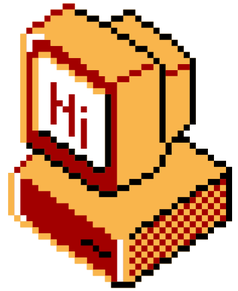
Thanks to UWA UISS for the venue!

$ ~/: cat ./housekeeping



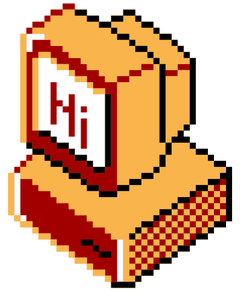
- Don’t break stuff
- If you break stuff tell us
- Be respectful
- Have fun.

$ ~/: cat ./housekeeping



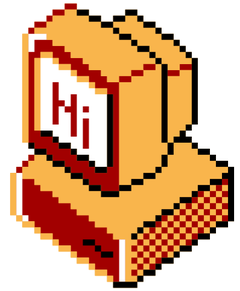
- Also, we don't have access to a guest Wi-Fi network here - trying to connect to them is **out of scope**
- Please use your own hotspots, or ask one of us to lend you our hotspot

$ ~/: groups "socialware"



Acknowledgement of Country

$ ~/: whoami



Emu Exploit

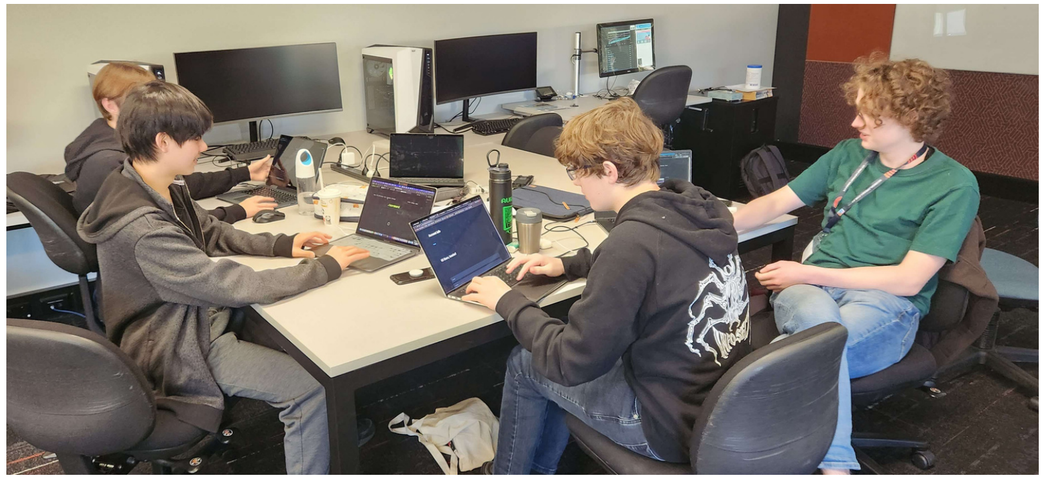
- We are a competitive hacking team current rank #1 in Australia on CTFtime.org
- Founded in 2021, the team consists of many highschoolers as well as industry professionals

Today's Presenters

- Riley (toasterpwn) - Captain
- Rainier (teddy / TheSavageTeddy) - Vice Captain
- Torry (torry2)
- Orlando (q3st1on)
- Avery (nullableVoidPtr)

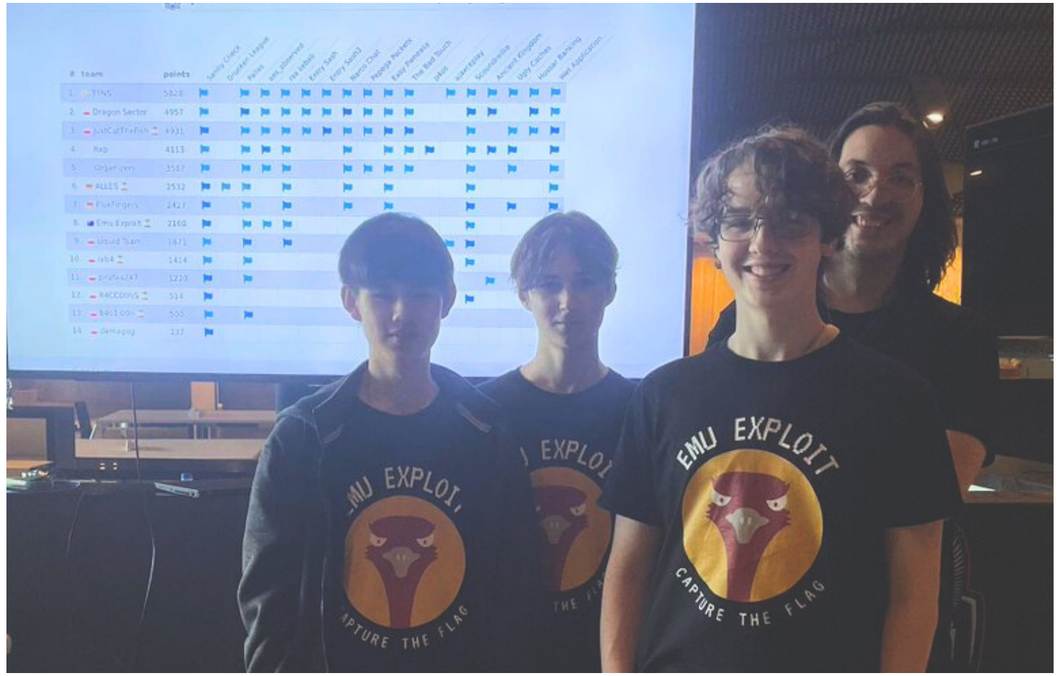
Emu Exploit at Pecan CTF 2023

$ ~/: whoami



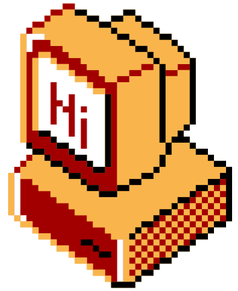
Pecan CTF 2023

WACTF 0x05



p4CTF in Katowice, Poland

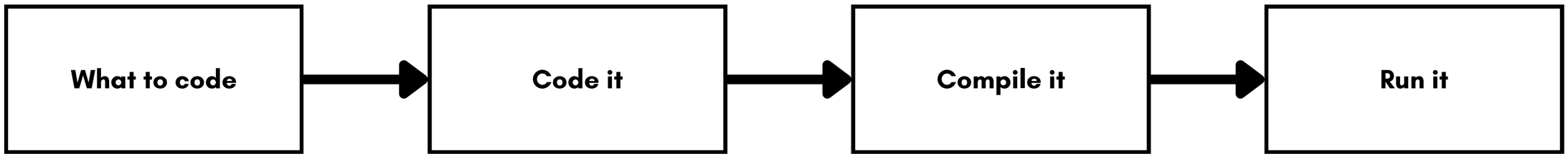
$ ~/: reverse engineering



First of all, what is **reverse engineering**?

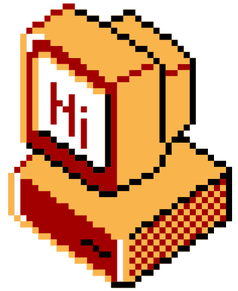
Consider the process of building a program:

- You figure out **what** you want to code
- You **implement** it in code
- You **compile** the code
- You **run** the code



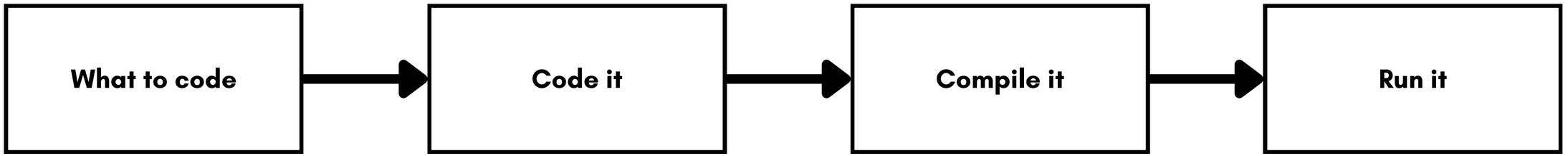
Information is lost at every stage!

$ ~/: reverse engineering

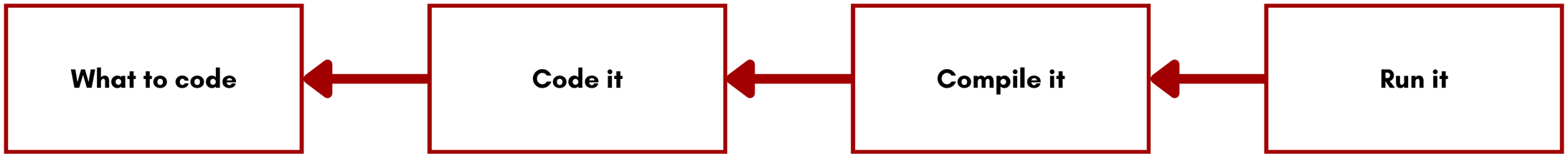


How can we get back the information that was lost?

This is what reverse engineering is!

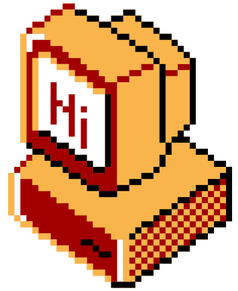


Information is lost at every stage!



How can we get it back?

$ ~/: cat content



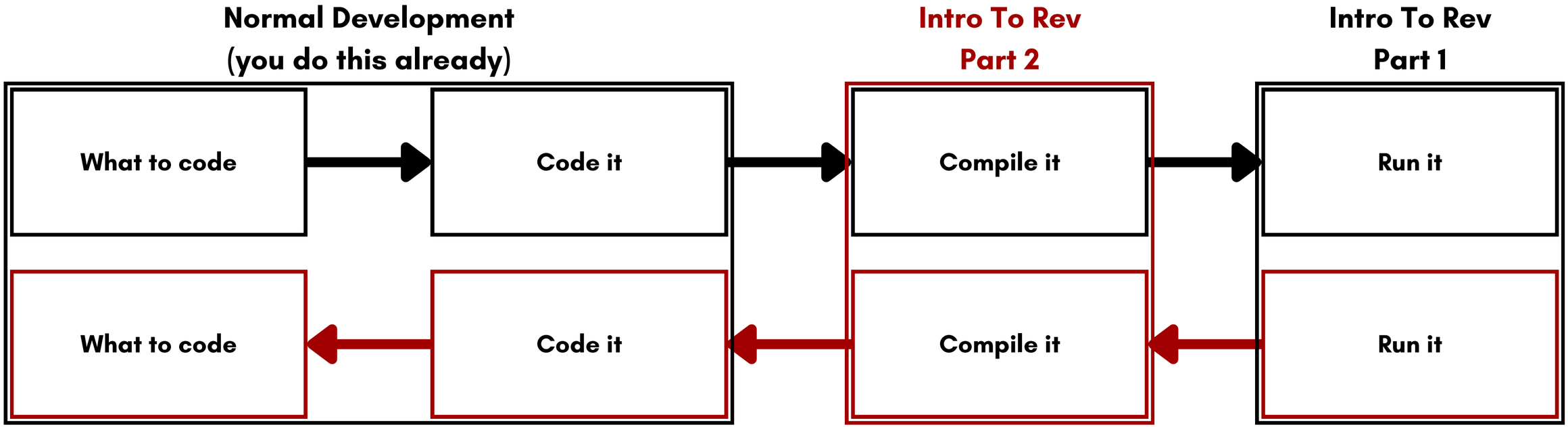
Presentation [6:00] -> Workshop [6:30] -> End [8:00]

Art of Reverse Engineering

The C Programming Language

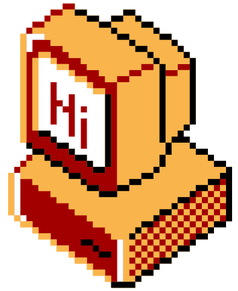
Decompilation

Static Analysis



Workshop Filedrop: https://emu.team/filedrop

$ ~/: The C Programming Language



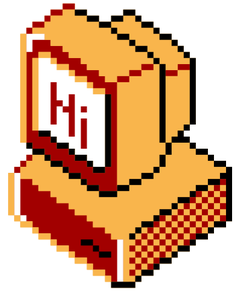
- General Purpose Programming Language
  - can be initially intimidating

- Statically Typed & Compiled
- Used for low level systems & applications

- Influential in computing and development
- E.g Used In: Operating systems, drivers and applications

- Understanding of computer memory is helpful to learn C
- Understanding of C is helpful to reverse a variety of applications

$ ~/: C - Syntax



Syntax in C:

- Lines deliminated by “;” semicolins;
- Comments are // for single line or /* multi line*/

```
error: expected ‘;’ before ‘return’
    5 |          puts("Hello, World!")
```

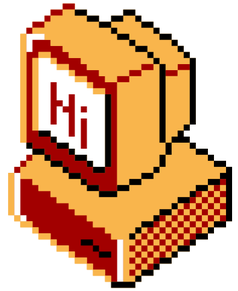
Include Statements: “#include <library>”
Declarations: “<datatype> <name> <operator> <value>”
Keywords: “for”, “if”, “const”, “return”
Operators: “+”, “-”, “*”, “/”, “==”, “&”, “|” and more...

Functions are defined with “<returntype> <name>()” and contents are wrapped in “{}”

- a “main” function is **always** the starting point for a C program

- Reading C becomes intuative

$ ~/: C - Data Types



- Example Data Types

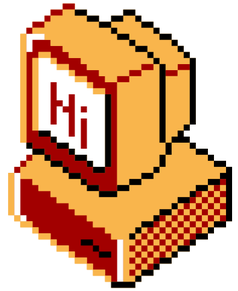
int	2 or 4 bytes	Whole Numbers
float	4 bytes	Numbers with decimals
char	1 byte	Single value (e.g ASCII character)
struct		Collection of elements of different data types

- Specifies the size and type of information to be stored

https://www.w3schools.com/c/c_data_types.php

- Data types can be “casted” for conversion
  - This is done via (<type name>) <expression>

$ ~/: C - Control Flow

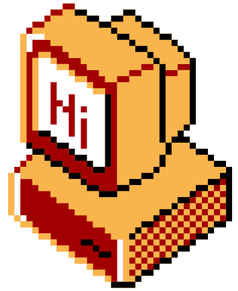


- Keywords used to define flows
- Wrapped in {} similar to functions

- if / else
- for loop
- while loop
- break / continue
- switch / case

```c
if (condition){
  something
}

while (condition){
  something
}

for (int i = 0; i < 1337; i++){
  something iteration
}
```

$ ~/: C - Common Pitfalls



Return Values:

- Functions in C expect to be returned to a value
- e.g int main() {} is the main function expecting a return value of type int

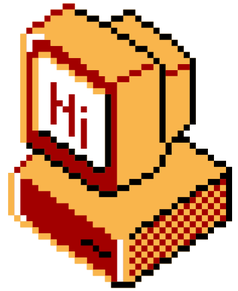
Char Arrays:

- Strings in C are “char arrays”
- This is an Array of characters that make up the string, these arrays end in a “null byte” to terminate the string
- e.g char string[] = “example”;

Indexing:

- Indexing arrays and similar are counted from 0

$ ~/: C - Common Pitfalls

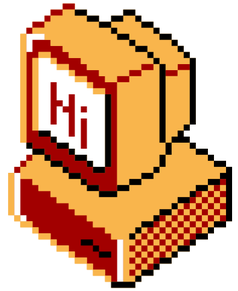


Pointers:

- Denoted by “*” character (to create and dereference)
- Pointers are a variable storing the memory address of another variable, denoted by
- Commonly seen as a difficult concept however quite simple
- E.g point to variable “foo” is the value of “foo”s memory address, plenty of googlable resources explain it well

```c
int number = 1337;      // Variable
int* ptr = &number;  // Pointer

printf("%p\n", ptr); // (0x7ffe5367e044)

printf("%d\n", *ptr); // 1337
```

$ ~/: “Hello World” - C <-> ASM

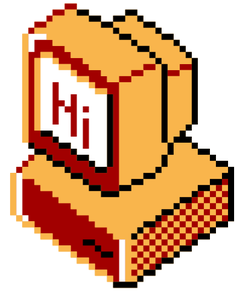


“Hello World” in C vs. Assembly

```c
#include <stdio.h>

int main()
{
  puts("Hello, World!")
  return 0;
}
```

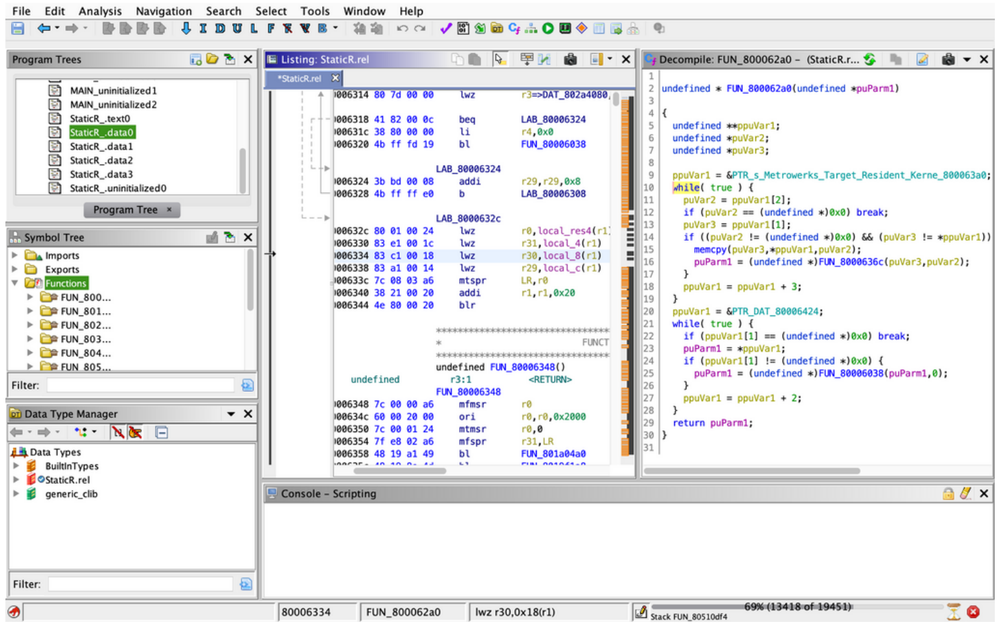
```asm
// hello.s - gcc hello.c -O0 -masm=intel -S

        .file   "hello.c"
        .intel_syntax noprefix
        .text
        .section        .rodata
.LC0:
        .string "Hello, World!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        push    rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        mov     rbp, rsp
        .cfi_def_cfa_register 6
        lea     rax, .LC0[rip]
        mov     rdi, rax
        call    puts@PLT
        mov     eax, 0
        pop     rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Debian 13.2.0-2) 13.2.0"
        .section        .note.GNU-stack,"",@progbits
```

$ ~/: Static Analysis

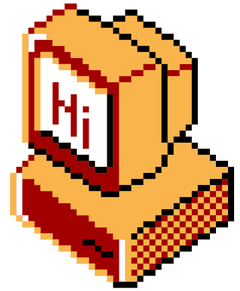


Reverse-engineering through methods of examining available code or binaries *without* executing it.

- Methods: Disassembly & Decompilation
- Techniques: Annotation of Types and Functions



$ ~/: Disassemblers



- Reading programs as a human is tedious
  - Manually decode the instructions from the binary
  - Keep track of which pointers target where
  - Identifying and documenting where structs are used
  - Naming and documenting functions and different blocks of code
- Use compilers for forward-engineering; disassemblers for reverse-engineering
  - What about **de**compilers?

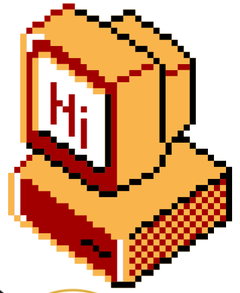
$ ~/: Disassemblers

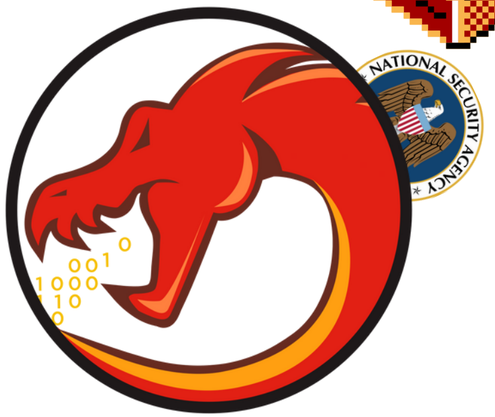
IDA Pro
by Hex Rays

US$365 USD for base version

Binary Ninja
by Vector35

Free for Cloud
US$300 for Full Ver.





Ghidra
by the National Security Agency

Free (and Open Source!)

$ ~/: Disassemblers

IDA Pro
by Hex Rays

US$365 USD for base version
Upwards of $10000 for Pro version with all features

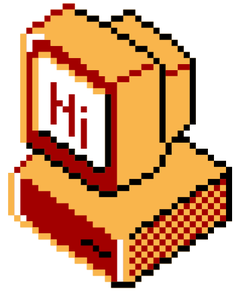
Binary Ninja
by Vector35

Free for Cloud
US$300 for Full Ver.

Ghidra
by the National Security Agency

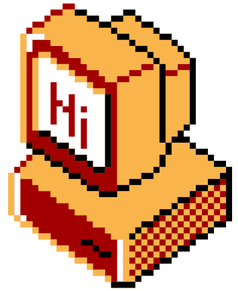
Free (and Open Source!)

$ ~/: Decompilation



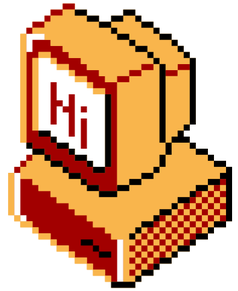
- Compilers have to use an assembler internally
- Decompilers need a disassembly, from a disassembler
- IDA Pro has a *really* good decompiler
  - (+) Fast! Reliable! Concise!
  - (-) Expensive!
- Both Binary Ninja and Ghidra have (okay) decompilers as well
- Ghidra and Binary Ninja Demo version is free...

$ ~/: Decomp Disclaimers



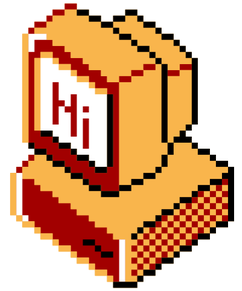
- Decompilation is simply not accurate
  - You still need to annotate
  - Do not entirely rely on decompilation for all of your reverse-engineering
  - Information is not recovered perfectly and can have some parts missing
    - Entire lines of code can be lost to optimisations done in compiling the original code
  - There can be cases where a decompiler will crash when tried on a file, either done deliberately or not
- While a lot more to read and can appear more confusing, every disassembly is more accurate than its decompilation as its a “direct translation” rather than a “best guess”.

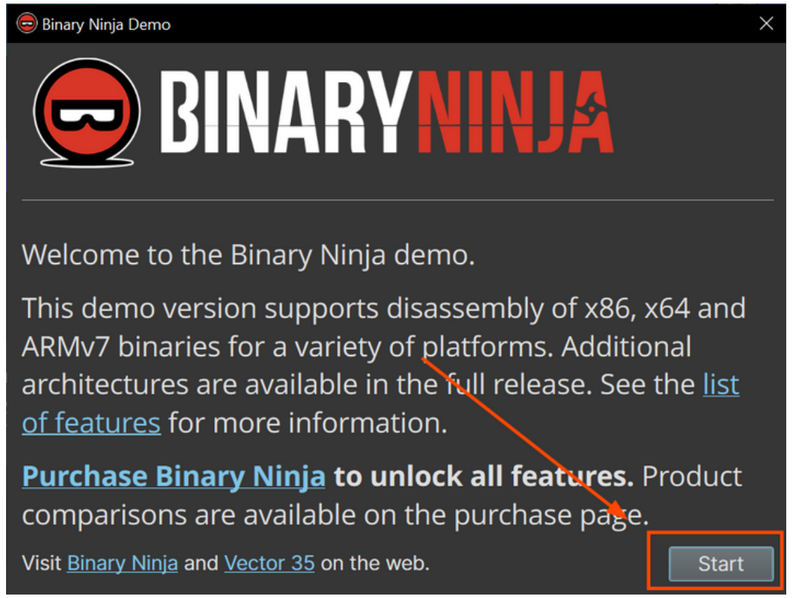
$ ~/: Tooling

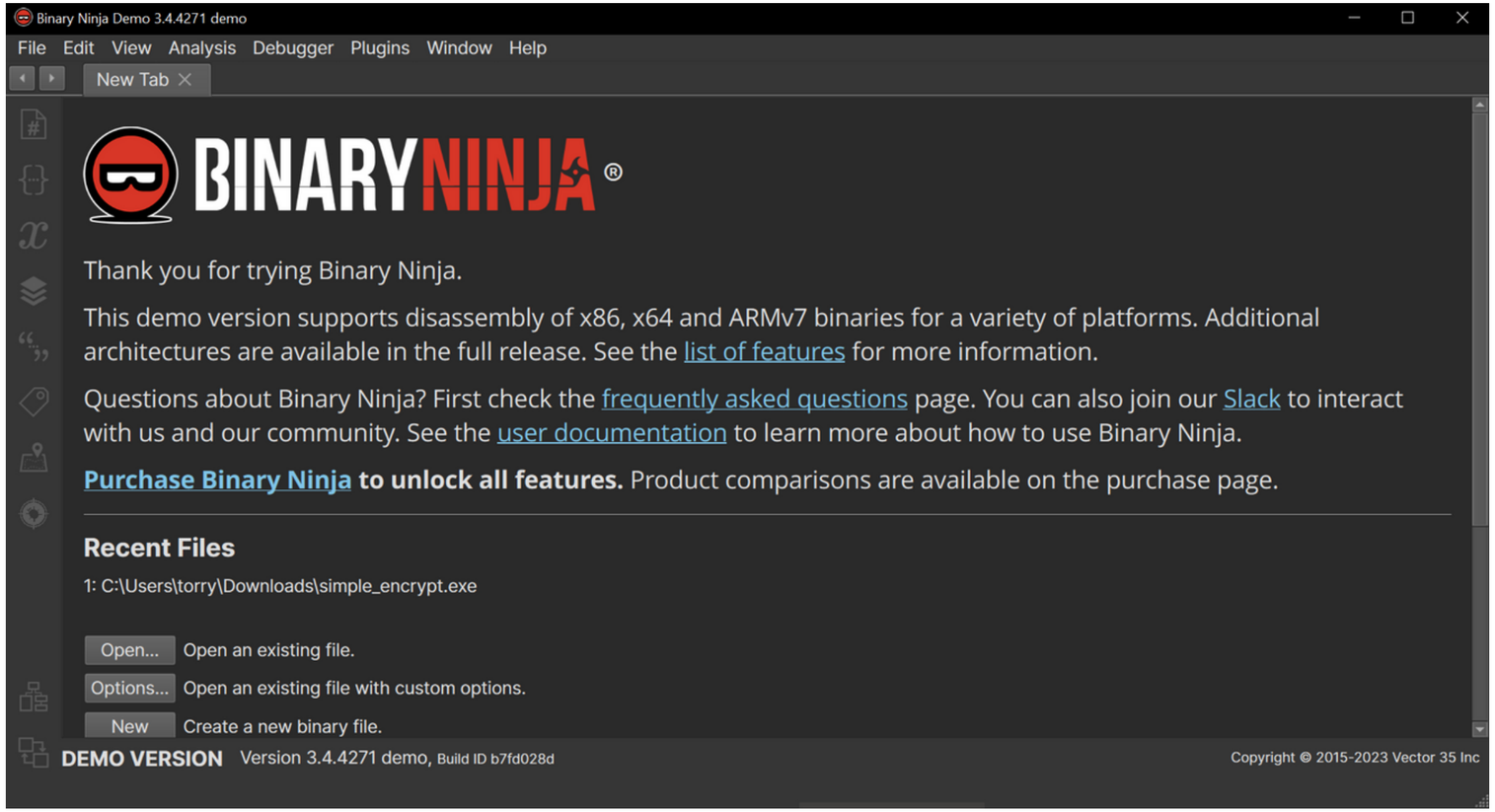


- All mentioned tools have different features and quirks
- **There is no best tool for this**
  - Different programs, architectures, compilers
- Down to personal preference
- IDA, Binary Ninja and Ghidra all have scripting capabilities
  - Develop your own scripts!

$ ~/: Getting started

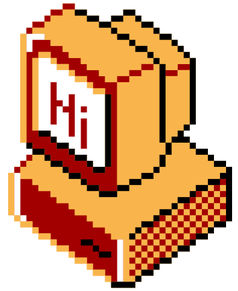


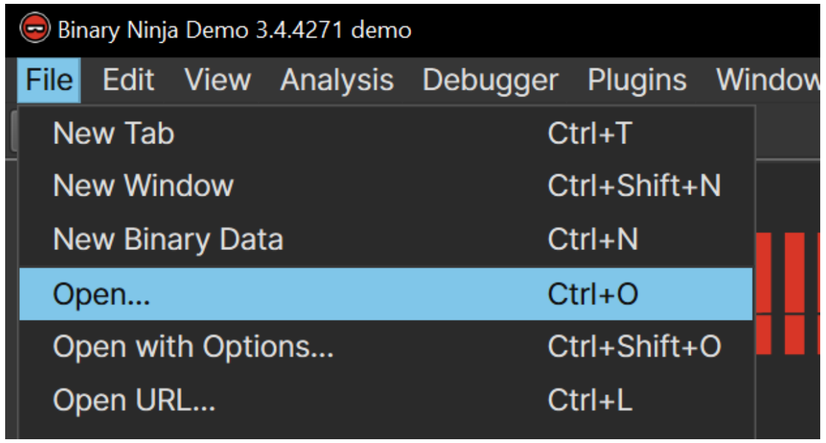




- Start & Main Window

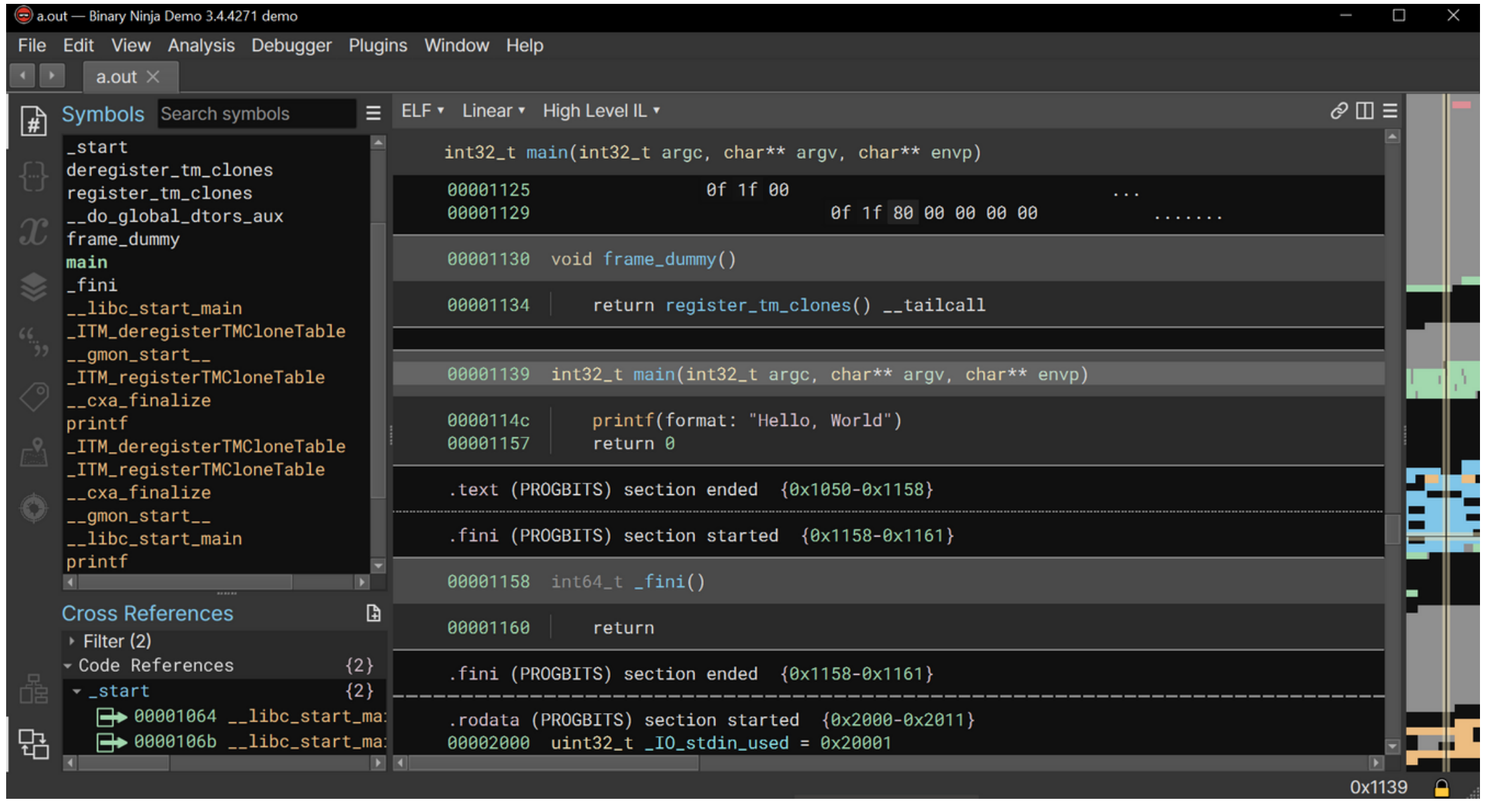
$ ~/: Getting started



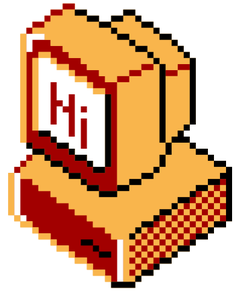


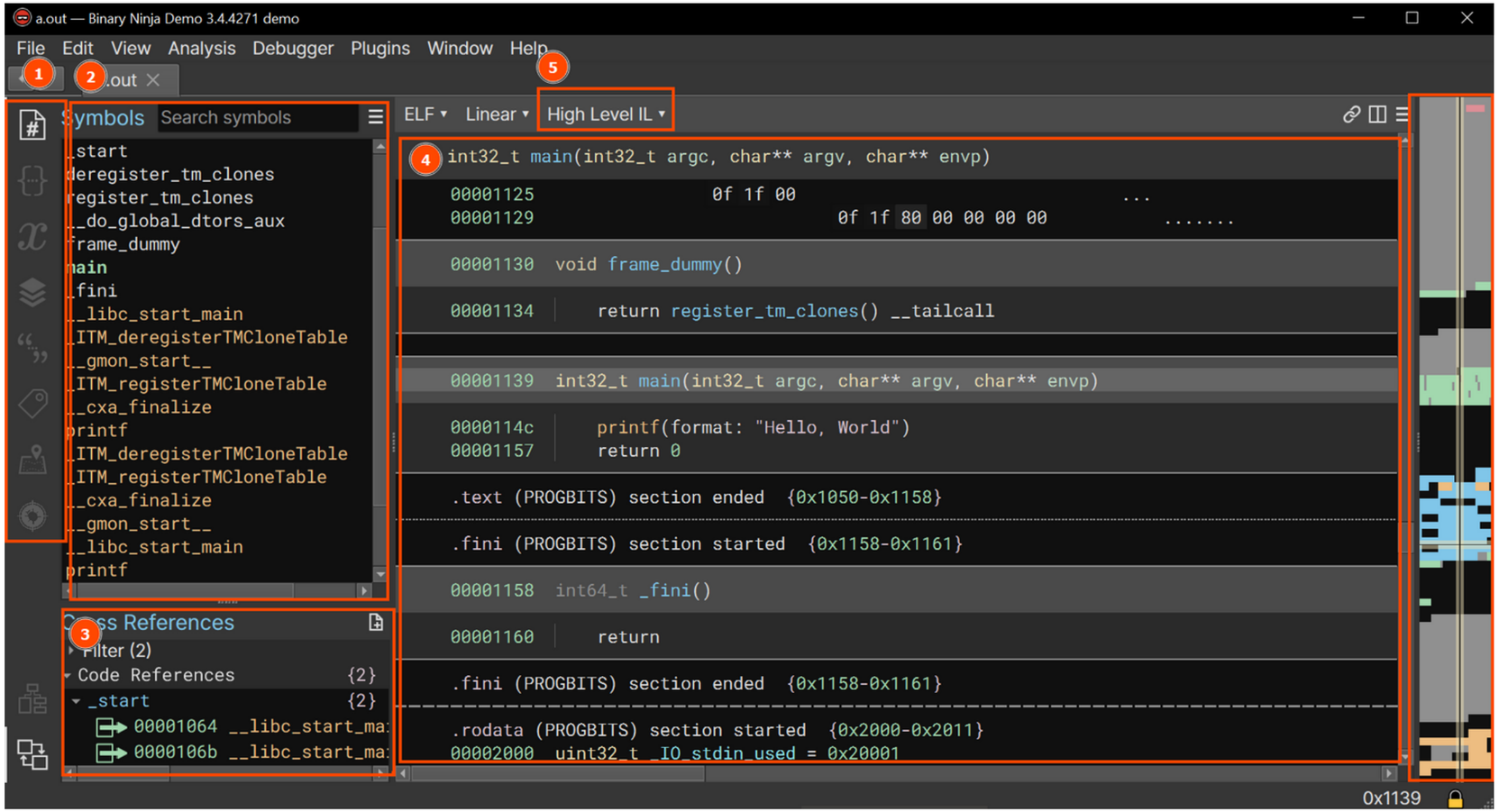
- File -> Open (Ctr+O)

- LoadBinary & View



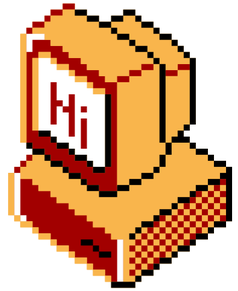
$ ~/: Getting started





1. Context Menus
2. Menu
3. Cross References
4. Window
5. Selection

~/: pause



Workshop/Networking will now commence!

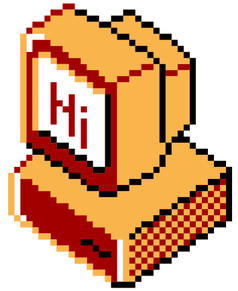
Filedrop! Find the exercise and challenge files here:

- https://emu.team/filedrop
- 2 Exercises +crackme challenge ! (solutions soon)

Download “Binary Ninja”: (cross platform)

- https://binary.ninja/demo/

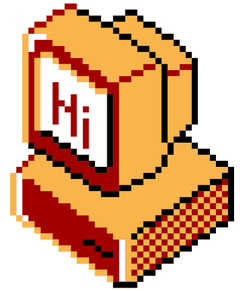
~/: Compiling C online

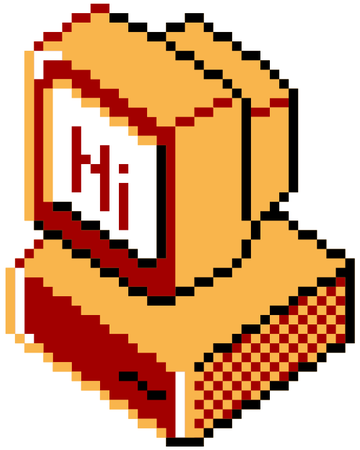


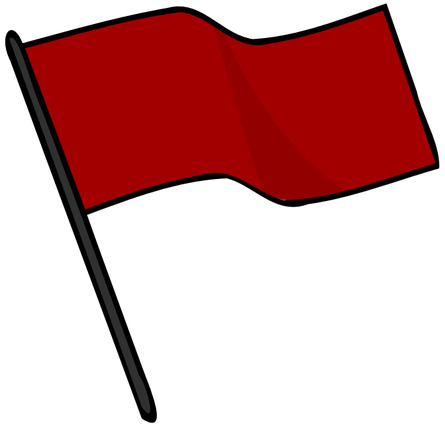
If you cannot compile C code on your machine, use this website:
https://cplayground.com/

Online decompiler (may give better results than Binary Ninja):
https://dogbolt.org/

$ ~/: questions

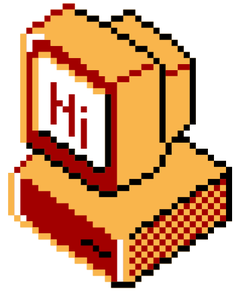






Questions!

~/: shutdown



Thank you!