Reverse Engineering

Midterm

Instructor:	G Leaden	**Due:**	Today End-of-Class
Email:	g.leaden1@marist.edu	**Place:**	Hancock 2023

Goals:
Test mastery of material mid-way through the semester.

Instructions:
Please answer each question with complete sentences. Answer until you feel you have competently answered the question.

1. Explain three forms of reverse engineering in software development.
2. Explain the benefits of reverse engineering in software development and why it is used.
3. Explain the benefits of reverse engineering in software security.
4. Define the following CPU Terms:
    - Instruction
    - Machine Code
    - Assembly Language
    - CPU Register
5. Why dont we write in machine code? Why do we have higher level programming languages?
6. Fill in the blanks on this table:

Hexadecimal (Radix = 16)	Binary (Radix = 2)	Decimal (Radix = 10)
		1
		2
3	0011	3
	0100	4
	0101	5
		6
		7
	1000	8
	1001	9
		10
	1011	11
	1100	
		13
	1110	14
		15

7. Explain what a debugger is and how we use them in this class.
8. Draw the compilation process.
9. Compiler optimization can be targeted towards two variables. What are they and why is that relevant to reverse engineering?

Submitting:

Hand sheet into me when complete. All midterms will be collected at the end of class time.

Grading Rubric:

1	10%
2	10%
3	15%
4	10%

5	10%
6	10%
7	15%
8	10%
9	10%

ANSWERS

Due to the nature of long-form questions. This answer sheet will just have highlights of what should be mentioned for a complete score.

1. Explain three forms of reverse engineering in software development.
    - Redocumentation
      Simplest and oldest form, considered to be unintrusive and weak form of restructuring.
      Creation or revision of a semantically equivalent representation with the same relative abstraction level.
      Creating documentation (class diagrams, functional diagrams, diagrams diagrams diagrams).
    - Restructuring
      Transformation from one representation form to another at the same relative abstraction level, while preserving external behavior (functionality, semantics).
      Altering code to improve structure in the traditional sense.
      Data Normalization is an example of restructuring.
      Restructuring can happen with knowledge of form but not meaning.
    - Reengineering
      Examination and alteration of a subject system to create a new system and implement it (to get a more abstract view).
      Requires: some form of the above forms of RE followed by forward engineering
      While reengineering involves both forward and reverse engineering, it is not a supertype of the two.

2. Explain the benefits of reverse engineering in software development and why it is used.
    - Aid maintenance, strengthen enhancement, or support replacement.
      Increase overall comprehensibility of the system for both maintenance and new development.
    - Understanding code you didnt write.
    - Cope with Complexity.
    - Recover lost information.
    - Detect side effects.
    - Synthesize higher abstractions.
    - Facilitate reuse.

3. Explain the use of reverse engineering in software security for both offense and defense.
    - Offensive
      Obtaining a desired output or state in a program with little to no knowledge on how to obtain that through intended means.
      RE techniques can be applied (dissassembling the compiled program into assembly instructions) to identify the structure and intention of the system, and you will be privy to any bug or exploit present that could help you reach your goal.
    - Defensive The implementation of tools and topics covered with RE in Software Development for security purposes.
      Understanding a system.
      Identifying flaws or bugs.
      Securing the Offense listed above.

4. Define the following CPU Terms and provide an example:

   - Instruction
     An instruction is a primitive CPU command. Some examples of instructions would be moving data between registers, working with memory and primitive arithmetic operations.
   - Machine Code
     Code that the CPU directly processes. Each instruction is usually encoded by several bytes. An example would be the code of an Instruction, in 1s and 0s.
   - Assembly Language
     Mnemonic code and some extensions, like macros, that are intended to make a programmers life easier. Typical language for writing instructions. Some examples would be ADD, MOV, POP, and PUSH.
   - CPU Register
     A CPU register is a quickly accessible location available to a computer's central processing unit (CPU). Registers usually consist of a small amount of fast storage, although some registers have specific hardware functions, and may be read-only or write-only. Some examples would be (R/E)AX, (R/E)BP, (R/E)SP.

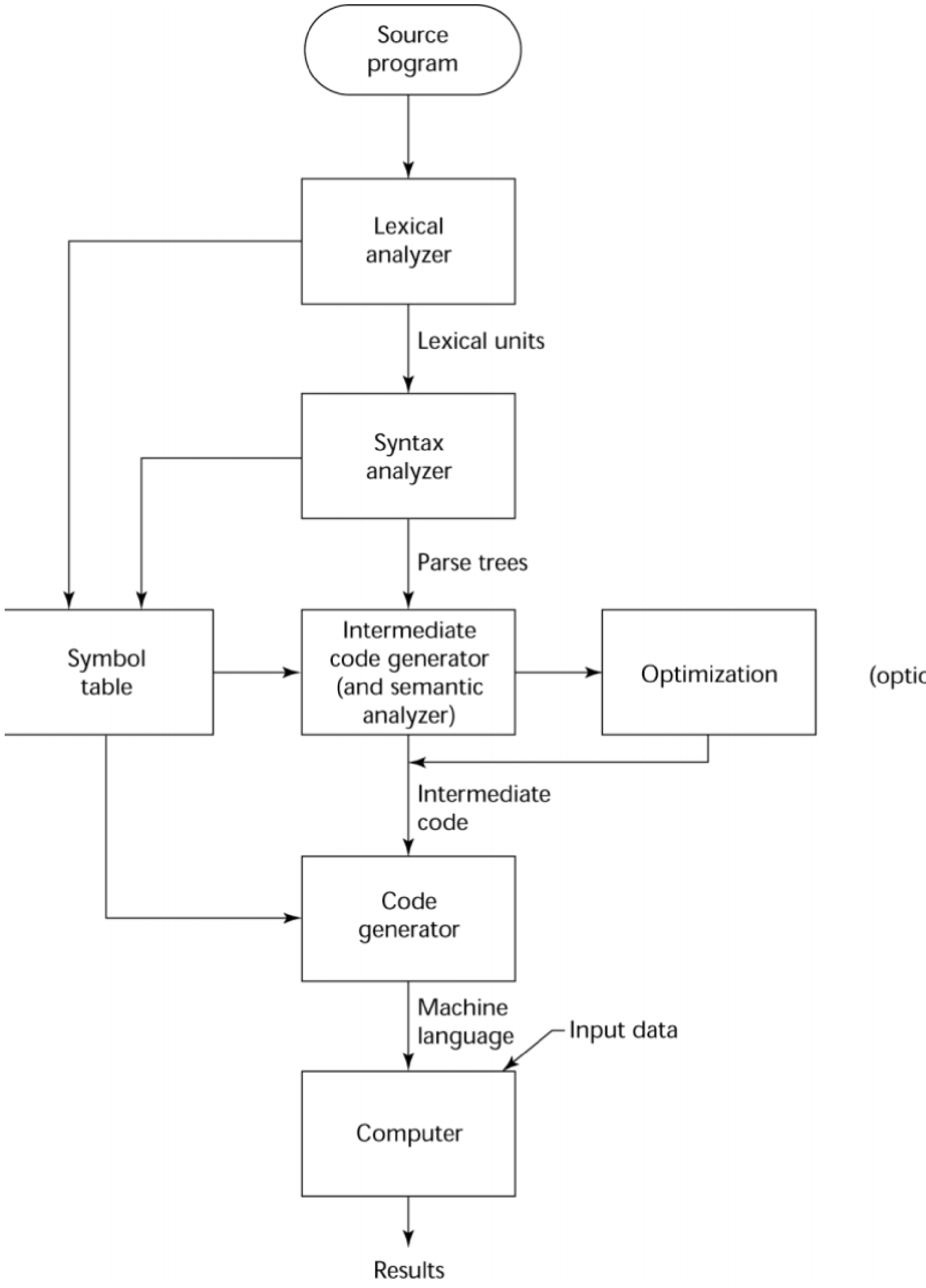
5. Why dont we write in machine code? Why do we have higher level programming languages?
   It is much easier for humans to use a high-level PL like C/C++, Java, or Python, but it is easier for a CPU to use a much lower level of abstraction.
   It is very inconvenient for humans to write in assembly language, due to it being so low-level and difficult to write in without making a huge number of annoying mistakes.
   A compiler converts high level programming language code into CPU readable machine code. More on that in a later week.

6. Fill in the blanks on this table:

Hexadecimal (Radix = 16)	Binary (Radix = 2)	Decimal (Radix = 10)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

7. Explain what a debugger is and how we use them in this class.
   A program that is used to test and debug other programs.
   The debuggers we will be looking at are low-level debuggers or machine-language debuggers it shows the line in the disassembly (unless it also has online access to the original source code and can display the appropriate section of code from the assembly or compilation).
   The same functionality which makes a debugger useful for eliminating bugs allows it to be used as a software cracking tool to evade copy protection, digital rights management, and other software protection features. It often also makes it useful as a general verification tool, fault coverage, an performance analyzer, especially if instruction path lengths are shown.
   GDB, Radare2, IDA are examples of these.

8. Draw the compilation process.


Source program
Lexical analyzer
Lexical units
Syntax analyzer
Parse trees
Symbol table
Intermediate code generator (and semantic analyzer)
Optimization
(optic
Intermediate code
Code generator
Machine language
Input data
Computer
Results

process.png

9. Compiler optimization can be targeted towards two variables. What are they and why is that relevant to reverse engineering?
   Optimization can be targeted towards code size or code speed. A non-optimizing compiler is faster and produces more understandable (albeit verbose) code, whereas an optimizing compiler is slower and tries to produce code that runs faster (but is not necessarily more compact).
   Reverse engineers can encounter either version, simply because some developers turn on the compilers optimization flags and others do not. We cannot take one compilers output to be the verbatim output of source to machine code, it just mirrors the function of the source.

   BONUS if they have any of the below.
   In addition to optimization levels, a compiler can include some debug information in the resulting file, producing code that is easy to debug.
   One of the important features of the debug code is that it might contain links between each line of the source code and its respective machine code address.
   Optimizing compilers, on the other hand, tend to produce output where entire lines of source code can be optimized away and thus not even be present in the resulting machine code.