**using the run\_tests.sh script:**

chmod +x run\_tests.sh

./run\_tests.sh ex1\_solution.asm ex1\_tests

./run\_tests.sh ex2\_solution.asm ex1\_tests

./run\_tests.sh ex3\_solution.asm ex1\_tests

./run\_tests.sh ex4\_solution.asm ex1\_tests

./run\_tests.sh ex5\_solution.asm ex5\_tests

**ex1\_tests:**

test01 - Num1 = 0, Num2 = 0 → both even → expect 1

test02 - Num1 = 0xFFFFFFFF, Num2 = 0xFFFFFFFF → both even → expect 1

test03 - Num1 = 0xAAAAAAAA, Num2 = 0x55555555 → both even → expect 1

test04 - Num1 = 0xF0F0F0F0, Num2 = 0x0F0F0F0F → both even → expect 1

test05 - Num1 = 0x0000000F, Num2 = 0x0000000F → both even → expect 1

test06 - Num1 = 0x00000001, Num2 = 0x00000002 → both odd → expect 1

test07 - Num1 = 0xAAAAAAAA, Num2 = 0xFFFFFFFF → both even → expect 1

test08 - Num1 = 0xAAAAAAAA, Num2 = 0x55555555 → both even → expect 1

test09 - Num1 = 0x0000FFFF, Num2 = 0xFFFF0000 → both even → expect 1

test10 - Num1 = 0x0000000F, Num2 = 0x00000001 → even vs odd → expect 0

test11 - Num1 = 0x00000003, Num2 = 0x00000005 → both even → expect 1

test12 - Num1 = 0x80000000, Num2 = 0x00000000 → odd vs even → expect 0

test13 - Num1 = 0xFF00FF00, Num2 = 0x00FF00FF → both even → expect 1

test14 - Num1 = 0x0F0F0F0F, Num2 = 0x0F0F0F0E → even vs odd → expect 0

test15 - Num1 = 0xF0F0F0F0, Num2 = 0x0000FFFF → both even → expect 1

test16 - Num1 = 0x0F0F0000, Num2 = 0x00000F0F → both even → expect 1

test17 - Num1 = 0x12345678, Num2 = 0x9ABCDEF0 → both odd → expect 1

test18 - Num1 = 0x00000001, Num2 = 0x00000000 → odd vs even → expect 0

**ex2\_tests:**

test01 - Empty input ([]) → palindrome → Result = 1

test02 - Single byte input ([5]) → palindrome → Result = 1

test03 - Two identical bytes ([AB, AB]) → palindrome → Result = 1

test04 - Two different bytes ([AA, BB]) → not palindrome → Result = 0

test05 - 3-byte palindrome ([1, 2, 1]) → palindrome → Result = 1

test06 - 3 different bytes ([1, 2, 3]) → not palindrome → Result = 0

test07 - 4-byte palindrome ([12, 34, 34, 12]) → palindrome → Result = 1

test08 - 4 different bytes ([11, 22, 33, 44]) → not palindrome → Result = 0

test09 - 5-byte palindrome ([A0, B1, C2, B1, A0]) → palindrome → Result = 1

test10 - 5-byte palindrome with different middle byte ([A0, B1, C3, B1, A0]) → palindrome → Result = 1

test11 - 6-byte perfect palindrome ([11, 22, 33, 33, 22, 11]) → palindrome → Result = 1

test12 - 6 different bytes ([11, 22, 33, 44, 55, 66]) → not palindrome → Result = 0

test13 - 7-byte palindrome ([1, 2, 3, 4, 3, 2, 1]) → palindrome → Result = 1

test14 - 7-byte modified palindrome ([1, 2, 3, 5, 3, 2, 1]) → palindrome → Result = 1

test15 - 8 different bytes ([10, 20, 30, 40, 50, 60, 70, 80]) → not palindrome → Result = 0

test16 - 8-byte perfect palindrome ([AA, BB, CC, DD, DD, CC, BB, AA]) → palindrome → Result = 1

test17 - 9-byte perfect palindrome ([1, 2, 3, 4, 5, 4, 3, 2, 1]) → palindrome → Result = 1

test18 - 9-byte perfect palindrome ([1, 2, 3, 4, 6, 4, 3, 2, 1]) → palindrome → Result = 1

**ex3\_tests:**

test01 - Empty list, no elements -> Result = 1

test02 - Single element list -> Result = 1

test03 - 3→6→9 arithmetic sequence -> Result = 1

test04 - 2→6→12 second degree sequence -> Result = 2

test05 - 1→4→9→16 second degree sequence -> Result = 2

**test06** - 10→12→15→19 second degree sequence -> Result = 2

test07 - 5→9→13→17→21 arithmetic sequence -> Result = 1

test08 - 0→1→4→9 second degree sequence -> Result = 2

test09 - 2→5→8→11→14 arithmetic sequence -> Result = 1

test10 - 10→13→18→25 second degree sequence -> Result = 2

test11 - -5→0→5→10 arithmetic sequence -> Result = 1

test12 - 7→15→26→40 second degree sequence -> Result = 2

test13 - 1→1→1 constant sequence -> Result = 1

test14 - 2→5→10 second degree sequence -> Result = 2

test15 - 3→7→12→18 second degree sequence -> Result = 2

test16 - 5→10→17→26 second degree sequence -> Result = 2

test17 - 8→16→24→32 arithmetic sequence -> Result = 1

test18 - 2→3→5→8→13 not a sequence (Fibonacci-like) -> Result = 0

test19 - 0→0→0→0 constant sequence -> Result = 1

**test20** - 1→2→4→8→16 not a sequence -> Result = 0

test21 - 4→9→16→25 second degree sequence -> Result = 2

test22 - 2→6→12→20→30 second degree sequence -> Result = 2

test23 - 7→14→21→28→35→42 arithmetic sequence -> Result = 1

test24 - 1→1→2→3→5 not a sequence (Fibonacci-like) -> Result = 0

test25 - 10→21→36→55→78 second degree sequence -> Result = 2

test26 - Two elements (5,10), constant difference (5) → Result = 1

test27 - Two elements (-7,-2), constant difference (5) → Result = 1

test28 - Three elements (2,6,12), second degree sequence (diffs 4 and 6) → Result = 2

test29 - Three elements (0,3,8), second degree sequence (diffs 3 and 5) → Result = 2

w

**ex4\_tests:**

test01 - Mix of lowercase, uppercase, special characters : abcDE!#aD → ABCDEAD

test02 - Lowercase letters and digits : 123aBzZ! → 123ABZZ

test03 - Only special characters : ##!!\*\* → (empty)

test04 - Alternating letters and digits : a1b2C3 → A1B2C3

test05 - Mixed case and digits : lowerCASE123! → LOWERCASE123

test06 - Upper, lower, digits, symbols : He!@#123llo → HE123LLO

test07 - Repeated lower and upper case letters : AAaaBBbb!! → AAAABBBB

test08 - Only digits : 1234567890 → 1234567890

test09 - Sentence with spaces and numbers : Hello World 2024! → HELLOWORLD2024

test10 - Only special characters : !@#$%^&\*()\_+-=[]{}|;':,.<>/? → (empty)

test11 - Mixed letters and digits with symbols : aA1!bB2@cC3# → AA1BB2CC3

test12 - Title case with numbers : Only\_Capital\_Letters123 → ONLYCAPITALLETTERS123

test13 - Mixed case with numbers : MiXeD CaSe1234 → MIXEDCASE1234

test14 - Lowercase and uppercase together : abcDEFghiJKL → ABCDEFGHIJKL

test15 - Lowercase, numbers and underscore : nothing\_special12345 → NOTHINGSPECIAL12345

test16 - Mixed lowercase and uppercase ending with digit : easyCASE9! → EASYCASE9

test17 - PascalCase with underscore and numbers : Data\_Structures2024 → DATASTRUCTURES2024

test18 - Lowercase word with numbers and special symbols : goodluck2025!! → GOODLUCK2025

**ex5\_tests:**

test01 - "add" appears alone -> Result = 1

test02 - "0x345" appears -> Result = 2

test03 - "imul" and "0b1010" appear -> Result = 3

test04 - No valid words or numbers -> Result = 0

test05 - "add" hidden inside "address" -> Result = 0

test06 - "mul" and "0345" both present -> Result = 3

test07 - "idiv" appears alone -> Result = 1

test08 - "123" (simple number) appears -> Result = 2

test09 - "div" appears alone -> Result = 1

test10 - "0b1101" binary number appears -> Result = 2

test11 - "add" hidden inside "addsubtracted" -> Result = 0

test12 - "sub" and "0x10" both present -> Result = 3

test13 - "0345" octal number appears -> Result = 2

test14 - "0b0001" binary number appears -> Result = 2

test15 - All valid words (mul, div, idiv, sub, etc) present -> Result = 1

test16 - "mul\_div\_idiv" (no separate words) -> Result = 0

test17 - "0b123" binary number -> Result = 2

test18 - "idiv" and "345" present -> Result = 3

test19 - "imul\_add\_sub" (no separated words) -> Result = 0

test20 - "idiv" and "0345" both present -> Result = 3

test21 - No valid words or numbers -> Result = 0

test22 - "mul" and "0x123" both present -> Result = 3

test23 - "sub" and "0b111" both present -> Result = 3

test24 - "imul" alone -> Result = 1

test25 - "0x2345" number only -> Result = 2

test26 - Words "multiply" and "divide" (not valid) -> Result = 0

test27 - Word "subtracted" (not valid) -> Result = 0

test28 - "idiv" present alone -> Result = 1

test29 - "mul", "0b000", and "sub" all present -> Result = 3

test30 - "0456" octal number -> Result = 2