AOC-2021-16

-- Day 16: Packet Decoder ---

As you leave the cave and reach open waters, you receive a transmission from the Elves back on the ship.

The transmission was sent using the Buoyancy Interchange Transmission System (BITS), a method of packing numeric expressions into a binary sequence. Your submarine's computer has saved the transmission in [hexadecimal](https://en.wikipedia.org/wiki/Hexadecimal) (your puzzle input).

The first step of decoding the message is to convert the hexadecimal representation into binary. Each character of hexadecimal corresponds to four bits of binary data:

0 = 0000

1 = 0001

2 = 0010

3 = 0011

4 = 0100

5 = 0101

6 = 0110

7 = 0111

8 = 1000

9 = 1001

A = 1010

B = 1011

C = 1100

D = 1101

E = 1110

F = 1111

The BITS transmission contains a single packet at its outermost layer which itself contains many other packets. The hexadecimal representation of this packet might encode a few extra 0 bits at the end; these are not part of the transmission and should be ignored.

Every packet begins with a standard header: the first three bits encode the packet version, and the next three bits encode the packet type ID. These two values are numbers; all numbers encoded in any packet are represented as binary with the most significant bit first. For example, a version encoded as the binary sequence 100 represents the number 4.

Packets with type ID 4 represent a literal value. Literal value packets encode a single binary number. To do this, the binary number is padded with leading zeroes until its length is a multiple of four bits, and then it is broken into groups of four bits. Each group is prefixed by a 1 bit except the last group, which is prefixed by a 0 bit. These groups of five bits immediately follow the packet header. For example, the hexadecimal string D2FE28 becomes:

110100101111111000101000

VVVTTTAAAAABBBBBCCCCC

Below each bit is a label indicating its purpose:

* The three bits labeled V (110) are the packet version, 6.
* The three bits labeled T (100) are the packet type ID, 4, which means the packet is a literal value.
* The five bits labeled A (10111) start with a 1 (not the last group, keep reading) and contain the first four bits of the number, 0111.
* The five bits labeled B (11110) start with a 1 (not the last group, keep reading) and contain four more bits of the number, 1110.
* The five bits labeled C (00101) start with a 0 (last group, end of packet) and contain the last four bits of the number, 0101.
* The three unlabeled 0 bits at the end are extra due to the hexadecimal representation and should be ignored.

So, this packet represents a literal value with binary representation 011111100101, which is 2021 in decimal.

Every other type of packet (any packet with a type ID other than 4) represent an operator that performs some calculation on one or more sub-packets contained within. Right now, the specific operations aren't important; focus on parsing the hierarchy of sub-packets.

An operator packet contains one or more packets. To indicate which subsequent binary data represents its sub-packets, an operator packet can use one of two modes indicated by the bit immediately after the packet header; this is called the length type ID:

* If the length type ID is 0, then the next 15 bits are a number that represents the total length in bits of the sub-packets contained by this packet.
* If the length type ID is 1, then the next 11 bits are a number that represents the number of sub-packets immediately contained by this packet.

Finally, after the length type ID bit and the 15-bit or 11-bit field, the sub-packets appear.

For example, here is an operator packet (hexadecimal string 38006F45291200) with length type ID 0 that contains two sub-packets:

00111000000000000110111101000101001010010001001000000000

VVVTTTILLLLLLLLLLLLLLLAAAAAAAAAAABBBBBBBBBBBBBBBB

* The three bits labeled V (001) are the packet version, 1.
* The three bits labeled T (110) are the packet type ID, 6, which means the packet is an operator.
* The bit labeled I (0) is the length type ID, which indicates that the length is a 15-bit number representing the number of bits in the sub-packets.
* The 15 bits labeled L (000000000011011) contain the length of the sub-packets in bits, 27.
* The 11 bits labeled A contain the first sub-packet, a literal value representing the number 10.
* The 16 bits labeled B contain the second sub-packet, a literal value representing the number 20.

After reading 11 and 16 bits of sub-packet data, the total length indicated in L (27) is reached, and so parsing of this packet stops.

As another example, here is an operator packet (hexadecimal string EE00D40C823060) with length type ID 1 that contains three sub-packets:

11101110000000001101010000001100100000100011000001100000

VVVTTTILLLLLLLLLLLAAAAAAAAAAABBBBBBBBBBBCCCCCCCCCCC

* The three bits labeled V (111) are the packet version, 7.
* The three bits labeled T (011) are the packet type ID, 3, which means the packet is an operator.
* The bit labeled I (1) is the length type ID, which indicates that the length is a 11-bit number representing the number of sub-packets.
* The 11 bits labeled L (00000000011) contain the number of sub-packets, 3.
* The 11 bits labeled A contain the first sub-packet, a literal value representing the number 1.
* The 11 bits labeled B contain the second sub-packet, a literal value representing the number 2.
* The 11 bits labeled C contain the third sub-packet, a literal value representing the number 3.

After reading 3 complete sub-packets, the number of sub-packets indicated in L (3) is reached, and so parsing of this packet stops.

For now, parse the hierarchy of the packets throughout the transmission and add up all of the version numbers.

Here are a few more examples of hexadecimal-encoded transmissions:

* 8A004A801A8002F478 represents an operator packet (version 4) which contains an operator packet (version 1) which contains an operator packet (version 5) which contains a literal value (version 6); this packet has a version sum of 16.
* 620080001611562C8802118E34 represents an operator packet (version 3) which contains two sub-packets; each sub-packet is an operator packet that contains two literal values. This packet has a version sum of 12.
* C0015000016115A2E0802F182340 has the same structure as the previous example, but the outermost packet uses a different length type ID. This packet has a version sum of 23.
* A0016C880162017C3686B18A3D4780 is an operator packet that contains an operator packet that contains an operator packet that contains five literal values; it has a version sum of 31.

Decode the structure of your hexadecimal-encoded BITS transmission; what do you get if you add up the version numbers in all packets?

To begin, [get your puzzle input](https://adventofcode.com/2021/day/16/input).

양식의 맨 위

Answer:

양식의 맨 아래

You can also [Share] this puzzle.

Your puzzle answer was 963.

The first half of this puzzle is complete! It provides one gold star: \*

## --- Part Two ---

Now that you have the structure of your transmission decoded, you can calculate the value of the expression it represents.

Literal values (type ID 4) represent a single number as described above. The remaining type IDs are more interesting:

* Packets with type ID 0 are sum packets - their value is the sum of the values of their sub-packets. If they only have a single sub-packet, their value is the value of the sub-packet.
* Packets with type ID 1 are product packets - their value is the result of multiplying together the values of their sub-packets. If they only have a single sub-packet, their value is the value of the sub-packet.
* Packets with type ID 2 are minimum packets - their value is the minimum of the values of their sub-packets.
* Packets with type ID 3 are maximum packets - their value is the maximum of the values of their sub-packets.
* Packets with type ID 5 are greater than packets - their value is 1 if the value of the first sub-packet is greater than the value of the second sub-packet; otherwise, their value is 0. These packets always have exactly two sub-packets.
* Packets with type ID 6 are less than packets - their value is 1 if the value of the first sub-packet is less than the value of the second sub-packet; otherwise, their value is 0. These packets always have exactly two sub-packets.
* Packets with type ID 7 are equal to packets - their value is 1 if the value of the first sub-packet is equal to the value of the second sub-packet; otherwise, their value is 0. These packets always have exactly two sub-packets.

Using these rules, you can now work out the value of the outermost packet in your BITS transmission.

For example:

* C200B40A82 finds the sum of 1 and 2, resulting in the value 3.
* 04005AC33890 finds the product of 6 and 9, resulting in the value 54.
* 880086C3E88112 finds the minimum of 7, 8, and 9, resulting in the value 7.
* CE00C43D881120 finds the maximum of 7, 8, and 9, resulting in the value 9.
* D8005AC2A8F0 produces 1, because 5 is less than 15.
* F600BC2D8F produces 0, because 5 is not greater than 15.
* 9C005AC2F8F0 produces 0, because 5 is not equal to 15.
* 9C0141080250320F1802104A08 produces 1, because 1 + 3 = 2 \* 2.

What do you get if you evaluate the expression represented by your hexadecimal-encoded BITS transmission?

양식의 맨 위

Answer:

양식의 맨 아래

Although it hasn't changed, you can still [get your puzzle input](https://adventofcode.com/2021/day/16/input).

You can also [Share] this puzzle.