For this exercise you are required to implement a command line utility for validating ASCII encoded protocol packets. A header file (*packets*.h) will be provided with a sample of valid and invalid packets (one line per packet). The utility should take the packets from the header file and validate each of them, writing the results to the console. If a packet is invalid, then the reason why it is considered invalid should also be written.

The utility **must be implemented using C**. You will need to submit all your source code and the console output from a successful run of the program.

If you do not have a C development environment ready configured, we suggest that you take a look at https://www.onlinegdb.com/ which is a web based IDE, allowing you to complete the task without installing or configuring any software.

Your solution will be assessed against the following criteria:

- Approach to solving the problem, including design and preparation
- Readability of the code, including pragmatic use of comments
- Writing code using styling and conventions idiomatic for the programming language
- Accuracy of the solution (does it work!)
- Error handling with clear and human readable error messages

A **GUI is not required** and will not be considered part of the assessment.

Parsing command line arguments is **not required**.

You **are permitted** to use the internet to help develop your solution for this task.

This document contains multiple pages. Please make sure to read all pages before starting.

The final section of this document outlines an optional unit testing task for if you have time remaining and are familiar with unit testing.

1. Packet Format

Below describes the format of a valid packet.

1.1. Packet

The minimum length of a packet is 4 characters.

The structure of each packet is as follows:

Item	Size in Chars	Description
Type	1	Any uppercase letter (A-Z). ASCII codes 65-90
SubType	1	Any alphanumeric character (0-9 A-Z). ASCII codes 48-57, 65-90
Data Portion	variable	The data portion is variable length and can be of any size. Some packets have an empty data portion. Section 1.2 describes the format of the data portion.
Wrapper Checksum	2	8bit unsigned integer encoded in hexadecimal as 2 ASCII characters (0-9 A-F). ASCII codes 48-57, 65-70

1.1.1. Calculating the Wrapper Checksum

The wrapper checksum is an 8bit unsigned integer value and is calculated by taking the modulus 256 of the ASCII codes for Type + SubType.

$$\text{Wrapper Checksum} = (\text{Type} + \text{SubType}) \text{ modulus } 256$$

The resultant checksum value is stored in the packet as a hexadecimal number using 2 ASCII characters (in the range 00-FF).

The data portion characters are not included in the wrapper checksum as the data portion has its own checksum(s).

Example

Type = W (ASCII code 87)

SubType = S (ASCII code 83)

Wrapper Checksum = (87 + 83) modulus 256 = 170

Wrapper Checksum (hex) = AA

1.2. Data Portion

Where there is no data to transmit, the length of the data portion is zero.

Where there is data to transmit, the data portion is split into '**chunks**' of up to 34 ASCII characters, consisting of up to 32 characters for the data itself, and is followed by a 2-character checksum. So in total a chunk with the checksum will be between 3 and 34 ASCII characters in length.

Any chunks preceding the final chunk will be of maximum length (34 characters), and the final chunk may be between 3 and 34 characters.

The structure of each chunk is as follows:

Item	Size in Chars	Description
Data	variable	32 characters.

		Final chunk may contain 1-32 characters.
Chunk Checksum	2	8bit unsigned integer encoded in hexadecimal as 2 ASCII characters (0-9 A-F). ASCII codes 48-57, 65-70

1.2.1. Calculating the Chunk Checksum(s)

The chunk checksum is calculated by summing all of the ASCII codes of the chunk characters (except it's checksum) and expressing the answer as modulus 256.

$$\text{Chunk Checksum} = (\sum \text{char}) \text{ modulus } 256$$

The resultant checksum value is stored in the packet as a hexadecimal number using 2 ASCII characters (in the range 00-FF).

Example

Data = ABCD

Checksum = (65 + 66 + 67 + 68) modulus 256 = 10

Checksum (hex) = 0A

1.3. Example Packets

Valid packet with 0 characters of data to transport

A172

Type	SubType	DATA PORTION	Wrapper Checksum
A	1	Empty	72

Valid packet with a single chunk of data to transport

L1001411608032923583423317D

Type	SubType	DATA PORTION	Wrapper Checksum
L	1	00141160803292358342331	7D

	DATA	Chunk Checksum
Chunk 1	00141160803292358342	31

Valid packet with multiple chunks of data to transport

H11E1050110117003D0000004000100000430002000 36+1368079

Type	SubType	DATA PORTION	Wrapper Checksum
H	1	1E1050110117003D0000004000100000430002000 36+13680	79

	DATA	Chunk Checksum
Chunk 1	1E1050110117003D0000004000100000	43
Chunk 2	000200036+136	80

2. Unit Testing

If you have time remaining and are familiar with unit testing then add unit tests.

3. Sample Packets

3.1. Valid Packet

L10014116080329235834233 17D

3.2. Invalid Packet (Incorrect Wrapper Checksum)

Y1ZHRCNHZHUIHBJUXTFWUAPUZKVEBCZOROC9NFXDYZUSMVEOOUPMWIFWSZAFHGGJFXHFC0QSOEWUUGC
CNUPLVAOOGRIUVOFRVSSLMOD2NZXWGLPBNZUQLFQWKGURQQWWXAZEZCOR03PYGPNZTCYEDJZEDZBYHL
QVYXCBYGMKCLBBLDAMZAJKMSSMNNXSWADAUCAU208B

3.3. Invalid Packet (Incorrect Data Portion Checksum)

A1MMBNQOBRXWEQZBOWYYDMJPDGJMHCQJCF92LQNXJZBDKWPBOGACLYDCROZXIVDEGZAO93TYFJNRMZ
WZCNACSDLFQSFANXQQOOWXXKD6VAHPBQOULEF772