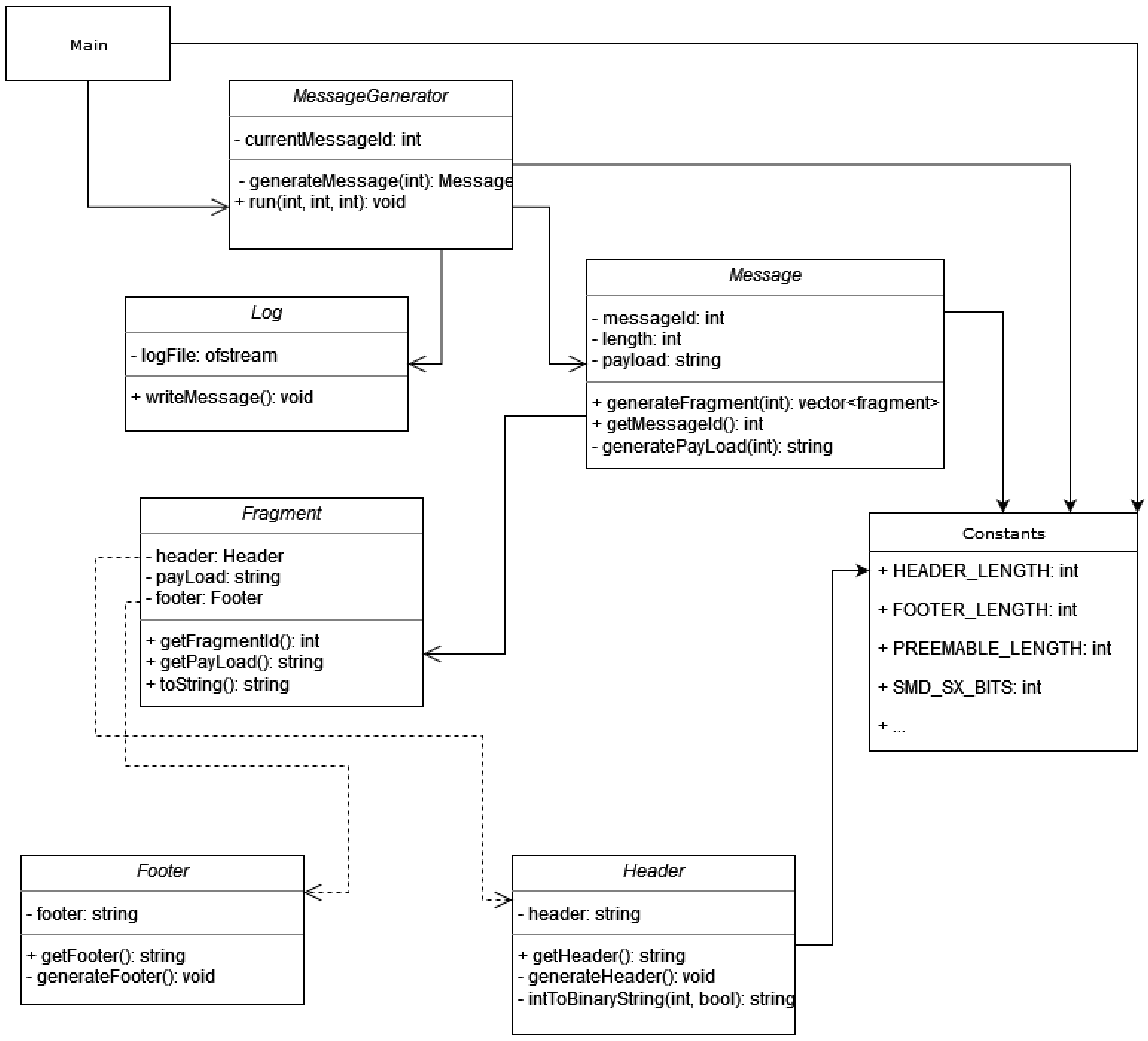
Design and Implementation Overview

Class Structure Diagram

The following diagram demonstrates the relationships and dependencies among the classes in the project:



Code Components

1. main.cpp

The main.cpp file contains the `main()` function, which serves as the entry point of the application. It parses command-line arguments to get the number of messages, message length, and channel width. It then creates an instance of `MessageGenerator` and calls its `run()` method to start the message generation process.

2. MessageGenerator

The `MessageGenerator` class is responsible for creating and managing messages and their fragments. Its primary function is `run()`, which performs the following tasks:

- **Loop Through Messages**: Iterates over the number of messages to be generated.
- **Create Message Objects**: For each message, it creates an instance of the Message class.
- **Generate Fragments**: Calls the `generateFragments()` method of the Message object.

3. Message

The Message class handles the creation of message fragments. Its `generateFragments()` method does the following:

- **Calculate Payload Size**: Determines the size of each fragment based on the channel width and message length.
- **Create Fragment Objects**: Loops over the message payload and creates fragments. Each fragment is sized to fit within the channel width, considering headers and footers.
- **Handle Padding**: If a fragment is shorter than the channel width, it is padded with zeros.

4. Fragment

The Fragment class represents a single fragment of a message. It has the following member variables:

- **Header**: An instance of the Header class, which is created based on whether the fragment is the first or last in the message.
- **Payload**: The actual data of the fragment.
- **Footer**: An instance of the Footer class, which is determined based on whether the fragment is the last one in the message, padded with 0s if the fragment length is less than the channel width.

5. Header

The Header class handles the header for each fragment. It takes a boolean `isFirstFragment` to determine the appropriate header format. The header contains:

- **Preamble**: Fixed 6 bytes of 0x07.
- **SMD-Sx**: Message ID in a 2-bit field.
- **SFD**: Fixed value 0xD5 for the first fragment, otherwise, the fragment count 0xXX.

6. Footer

The Footer class represents the footer for each fragment. It takes a boolean `isLastFragment` to determine whether the footer should be:

- **Last Fragment**: Fixed 0xAABBCCDD.
- **Other Fragments**: Fixed 0x11223344.

7. Logger

The Logger class is responsible for writing the generated fragments to a file. It logs each fragment along with its message ID and fragment ID to message_log.txt in the docs/ directory.

How the Code Works

1. **Initialization**: The `main()` function initializes the `MessageGenerator` with the command-line arguments.
2. **Message Generation**: The `MessageGenerator` creates Message objects and calls `generateFragments()` for each message.
3. **Fragment Creation**: Each Message object generates fragments based on the payload and channel width.
4. **Header and Footer Handling**: Each fragment has a header and footer created according to its position (first, last, or intermediate).
5. **Padding**: Fragments shorter than the channel width are padded with zeros.
6. **Logging**: The Logger writes the details of each fragment to message_log.txt.

Makefile and Running the Program

Ensure that you have a `Makefile` in the project root to build and run the program. The `Makefile` should handle compiling, linking, and cleaning tasks. Use the make command to build the project and `make run ARGS="arg1 arg2 arg3"` to execute it with command-line arguments, where:

arg1: number of messages.

arg2: length of the message payload (in bytes).

arg3: channel width (in bytes).

Sample Output

Here are examples of how the output is logged:

```
make run ARGS="2 6 20"
```

1. **Non-fragmented Messages**:

2 messages with length 6 and channel width of 20:

```
1, 1, 07 07 07 07 07 07 00 D5 FF FF FF FF FF FF AA BB CC DD 00 00

2, 1, 07 07 07 07 07 07 01 D5 FF FF FF FF FF FF AA BB CC DD 00 00
```

2 messages with length 64 and channel width of 100

```
1, 1, 07 07 07 07 07 07 00 D5 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF AA BB CC DD 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

2, 1, 07 07 07 07 07 07 01 D5 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF AA BB CC DD 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
make run ARGS="2 5 15"
```

2. **Fragmented Messages**:

2 messages with length 5 and channel width of 15:

```
1, 1, 07 07 07 07 07 07 00 D5 FF FF FF 11 22 33 44

1, 2, 07 07 07 07 07 07 00 00 FF FF AA BB CC DD 00

2, 1, 07 07 07 07 07 07 01 D5 FF FF FF 11 22 33 44

2, 2, 07 07 07 07 07 07 01 00 FF FF AA BB CC DD 00
```