ECE 150: Fundamentals of Programming

(Sections 001 and 002)

Project 3

Deadline: 11:59pm Monday December 3, 2018

Problem 0: Packet ordering

Communication over the Internet happens in the form of transferring messages from a source to a destination. These messages can be of arbitrary size and to make the communication efficient, they are broken down into smaller sizes called packets. Each packet consists of a unique packet identifier, and a fragment of the message within it. Due to the nature of the communication, although these packets are transmitted in an ascending order of packet identifiers, they are often received by the destination out of order (not in ascending order). Consequently, a key function of the software stack in receiving these packets is in reconstructing the original order of the message. This is the objective of this project.

Consider that a source wishes to send the following message to a destination:

```
"1A is almost over."
```

This message would be broken down into packets such that each packet has a unique identifier, and stores one word as a fragment of the message within the packet. Therefore, the above message would result in the source generating the following packets. As we wish for you to use the `std::cin` functionality, we will restrict each *packet* to a string not containing any white space.

Packet identifier	Message fragment
`0`	`1A`
`1`	`is`
`2`	`almost`
`3`	`over.`
`4`	`EOT`

Notice that a complete message transmission terminates with a special end-of-transmission message `EOT`.

You will write a function that receives packets by reading from the console. Your program will perform actions based on the integer:

1. If that integer is `-2`, you will exit your program.
2. If that integer is `-1`, you will print the message as it is received so far.
3. If the integer is a positive integer (`0` or greater), that number indicates the packet identifier and your program will then read a string (the next word) from the console.

You are guaranteed to only receive one complete message; therefore, only one EOT will be transmitted. You are guaranteed that all packet identifiers will be less than the packet identifier of the EOT packet.

The order in which your program receives the packet will likely not be in the correct order. For example, you may receive the packets in the order:

Packet identifier	Message fragment
`4`	`EOT`
`2`	`almost`
`3`	`over.`
`1`	`is`
`0`	`1A`

The print function that is called to print the string will concatenate the strings together with a single blank character between the message fragments. If a message fragment is missing, print three periods; i.e., the string `"..."`. Do not print the EOT packet and do not print a blank character after the last message. If you have not yet received the EOT packet, you should terminate your string by printing three questions marks; i.e., `"???"`. When the print function is finished, it will print an end-of-line character.

Example

Recalling that white space is ignored, suppose the input file is given by

```
4    EOT
-1
2 almost
3 over.
-1
1 is
0 1A
-1
-2
```

Your output will be

```
... ... ... ...
... ... almost over.
1A is almost over.
```

Suppose, instead, that the input file is

```
-1
1 is
-1
3 over.
-1
0 1A
2 almost
4    EOT
-1
-2
```

Your output will be

```
???
... is ???
... is ... over. ???
1A is almost over.
```

Remember, there is no whitespace before the first character and no white space after the last printed character in your output.

`int main()` function

Your main function will instantiate an instance of the node class and then begin to read from console input. You may assume that the input will always be valid. If the input is an integer greater than or equal to zero, that integer represents the packet identifier and you will then read in a string. Both of these are then inserted into the linked list class. If the integer is `-1`, the `print_message()` member function will be called on the linked list class, and if the integer is `-2`, your function returns `0`.

Linked list class

In order to implement a linked list this project, you will need to implement two classes: `Node` and `Message`. Recall that a message chains together multiple nodes that consist of the identifiers and the message fragments.

The Node class

The Node class will store a message:

```
unsigned int identifier;
```

Stores the packet identifier.

```
std::string fragment;
```

Stores the packet message that corresponds to one string.

```
Node *p_next;
```

A pointer to the next node in the linked list of type `Node`.

The `Node` class should have the following public member functions:

```
Node( unsigned int id, std::string frag );
```

Constructor for a new node that contains these two

```
std::string get_fragment();
```

Returns the string representing the fragment stored in this packet.

```
Node *get_next();
```

Returns the pointer to the next node.

The `Message` class should be declared a friend of this node class, by including the following in the class definition:

```
friend class Message;
```

This will allow member functions to access and manipulate the member variables of the `Node` class.

The Message class

You will implement a class called `Message` that will store the packets of a message in ascending order of packet identifiers using a linked list. This list will be ordered when a packet is inserted into the list.

The `Message` class will have the following private member variable:

`Node *head;`
: A pointer to the first packet of the message.

The `Message` class will have the following public member functions:

`Message();`
: This is the default constructor. It will set the head to the null pointer.

`~Message();`
: This is the destructor. We must ensure that the dynamically allocated nodes are deallocated properly.

`void insert( unsigned int id, std::string fragment);`
: Inserts a node in the linked list that contains the packet information provided as parameters.

`void print_message();`
: Prints to the console output stream the stored message as described above.

Marmoset Instructions

The Marmoset submission filename is `Message.cpp`.

You are required to submit an `int main()` function according to the specifications above. You must enclose it within the preprocessor directives:

```
#ifndef MARMOSET_TESTING
int main();
#endif
```

Header File

You are provided with a header file (`Message.h`) that includes the class declarations. For testing, copy this file into the same folder as your code. Add the following statement to the top of your code file:

```
#include “Message.h”
```

You do not need to submit this header file to Marmoset, as Marmoset will provide its own copy of this file.

Modifications (optional)

The following options are possible, although you are not required to do either of these:

1. Your linked list class has a head pointer. In general, however, packets are delivered approximately in order, so if you included a tail pointer (always pointing to the last entry in the linked list), then you can check whether or not your new packet should be inserted at the back of the linked list (if it is after the packet currently at the end of the linked list) or if you have to iterate through the linked list to find the correct position of the node.

Policy 71 Warning

Warning: If you implement a variation of this project where you do not use the `Node` and `Message` classes as described in this project description, while you may receive 100% from the Marmoset test cases, you will never-the-less subsequently receive `0` on this project and receive -5% off of your final grade for misrepresentation.

Warning: You are reminded that your code may be submitted to plagiarism-detection software.