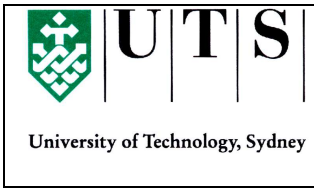


Faculty of Engineering and Information Technology
School of Software

31251 – DATA STRUCTURE AND ALGORITHMS

AUTUMN 2015
ASSIGNMENT 1 SPECIFICATION

DUE DATE – 29th April 2015
FINAL DUE DATE – 5th May 2015

This assignment is worth 25% of the total marks for this subject.

ASSIGNMENT DOCUMNTS

This assignment is split into two documents.

1. Assignment Specification. This document contains the specification for what you will be required to do.
2. Assignment Addendum. This document contains all other details regarding the assignment, including assignment submission and the assessment scheme.

REQUIREMENT

For this assignment, you will write a C++ program that simulates the re-assembly of messages received from a packet-switching network.

Background

A data transmission system typically breaks a large message into smaller packets. In some systems, the packets can arrive at the destination computer out of order, so that before the application requesting the data can process it, the communications program must ensure that the message is assembled in its proper form.

For this program, the data packets will be read from a file. Each packet will contain the following fields, separated by colons:

- Message ID number. This will always contain either 3 digits or the word END.
- Packet sequence number. This will consist of exactly three digits.
- Text. No text line will exceed ninety characters. The text may contain any characters, including colons.

Some examples of packets are:

```
101:021:"Greetings, earthlings, I have come to rule you!"
232:013:Hello, Mother, I can't talk right now,
101:017:Here is a message from an important visitor:
232:015:I am being harangued by a little green thingy.
END
```

All packets with a particular ID number belong to a single message. Sequence numbers may not be contiguous, but the correct ordering for a message will always be in ascending order of sequence numbers.

To simplify the task, it can be assumed:

- All messages will be complete: each message will have at least one data packet.
- Each packet will have a unique sequence number: a sequence number will not be used again for a subsequent packet.
- The END packet will always be the last packet received: this means you do not have to know the number of packets in a message, or wait for pending packets once an END packet has arrived.
- All message IDs and sequence numbers will be correct: you do not have to check messages for valid characters, etc.
- Packets will be contained in a file. Each line of the file will contain one packet, but as indicated above, the packets will be in random order (except for the END packet). There will be no blank lines.

Program Action

1. Your program will get the name of the file containing the packets as a command line argument.
2. It will then read the file and build up all the messages in a linked list of linked lists. For each message ID, you should maintain a linked list of packets received, and each new packet for the message should be inserted into the linked list in the correct place, so the list is always correctly sorted.
3. When the END packet is received, all the messages should be printed, one packet to a line, in correct order, with a header giving the 3 digit message ID. The message ID should be padded with 0's if it is numerically less than 100. A blank line will separate out each message ID.
4. As a final step, the program should then correctly free up all the memory used in the linked lists.

For example, the output of the message above would look like

```
Message 101
Here is a message from an important visitor:
"Greetings, earthlings, I have come to rule you!"

Message 232
Hello, Mother, I can't talk right now,
I am being harangued by a little green thingy.
```

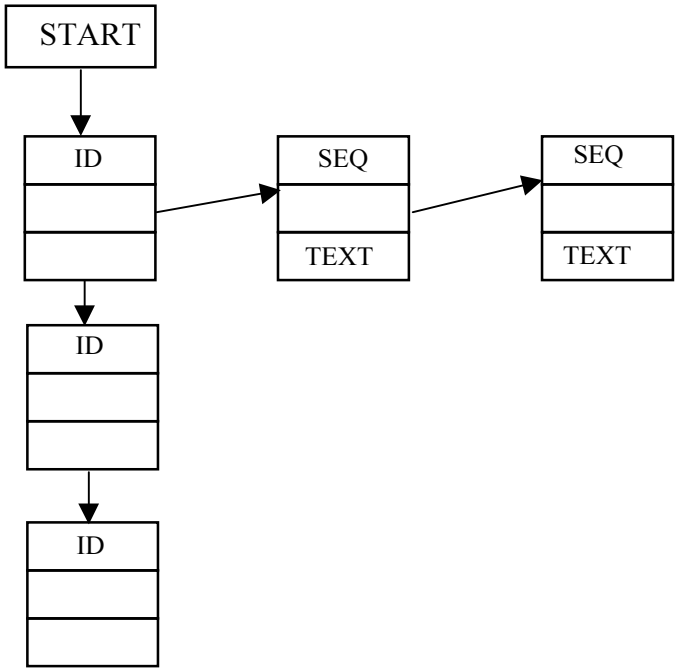
DETAILS

Data Structures

While there are other data structures that would work, you must implement a "linked list of linked lists" using the list template from the Standard Template Library (STL). This means you won't have to write any complex list code yourself.

To do this you will have a linked list of messages. Each message class will contain a linked list of packets. Whenever you need a new element, you should obtain the required space from dynamic memory using the new function. Do not use any other data structures to store the packets.

A diagram of the data structure follows



To help you use the STL list object I have included a file - `exstdlist.cpp` - which you can compile and run. Study the code closely as it demonstrates how to use doubly linked lists in the STL.

Compiling
In order to write your program you will need to create an appropriate set of classes. Each class should be in its own .h/.cpp file. This will probably require creating a `makefile` to handle the compilation steps.

PLEASE NOTE. Your program must compile on the laboratory Linux machines. Programs written on Window's machines sometimes don't compile or run properly on the student server.

HINTS

Separating out the ID and seq numbers is a bit messy if only using the material presented in the lecture notes. You can extract separate sections of a string using the `substr` function of `string`. Check out *http://www.cplusplus.com/reference/string/string/substr.html* for details and an example of its use.

BENCHMARK PROGRAM
To clarify how the program works, a benchmark executable version has been placed on my account. You can run it by typing the following command

```
/home/glingard/packets message_file
```

Where `message_file` is the name of a file containing data packets. You will have to create your own message files to test your program. To give you an idea of the size of this assignment, the source code for the benchmark program, including whitespace and comments, takes about 300 lines of code.

PLEASE NOTE 1. When you run the benchmark it says my name. Do NOT include that. Include yours if you want. If your program includes my name then it will be assumed you are trying to use the benchmark in the lab demonstration. See assessment below in Assignment Addendum.

ASSIGNMENT OBJECTIVES

The purpose of this assignment is to demonstrate competence in the following skills.

- Program design including classes
- Command line arguments
- Handling files
- Manipulation of STL linked lists

These tasks reflect all the subject objectives apart from objective 4.

As part of your subject workload assessment, it is estimated this assignment will take 25 hours to complete.