**CS 162 Computer Science II**

# Programming Assignment Lab 5

The alien digital life form that has taken up residence in the Enterprise data banks is doing well, and the xenomorphology lab is making progress towards communication with them. However, the researchers would like to have the ability to recover any location in heap memory at any time. So for this assignment, you will re-implement the code for assignment 4 to use a linear linked list. So this lab assignment will have the same requirements as lab 4, with a few additional requirements:

* Use a linear linked list to store the fleet information in memory. Take a look at the struct Ship and class Starfleet to the right. This is what your struct and class might look like for assignment 4, with two changes. You will notice that a Ship pointer has been added to the struct, and the array of Ship pointers in the class has been replaced with a pointer called head.

struct Ship

{

char \* name;

int registry;

char \* type;

char \* position;

char \* condition;

char \* captain;

Ship \* next; // Can point to next Ship.

};

class Starfleet

{

public:

void addShip();

void displayShips();

// Other methods go here.

private:

Ship \* head; // Head of the linked list.

int shipCount; // May not be needed.

};

* When a ship is added to the linked list, it must be added in alphabetical order according to the name of the ship.
* You must add a menu item to allow the operator (user) to delete a ship from memory.

## Strategies for using a linear linked list

* When you instantiate your Starfleet object, it is important to set the head pointer to nullptr. Using nullptr is important because if a pointer is set to this value, it means that it doesn’t point to anything, and is therefore the end of the list. So use your constructor for Starfleet to set head to nullptr.
* The head pointer will point to the first ship to be added to the linked list. You will know it is the first ship if head is equal to nullptr. You can go about adding the first ship in two ways: set the head pointer to the new Ship and then fill in the data, or set up a temporary Ship, fill in the data, and then set head to point to the temporary ship. Creating a temporary ship will pay dividends for you in the long run. Take a look at the code in the textbox to the left to see an example. This is only an example and you may have added ship data in other ways for assignment 4. You can modify your code from assignment 4 to work with assignment 5.

void StarFleet::addShip()

{

char input[STR\_SIZE];

cout << "What is the ships name? ";

cin.get(input, STR\_SIZE);

Ship \* temp = new Ship;

temp->name = new char[strlen(input) + 1];

strncpy(temp->name, input, strlen(input));

temp->name[strlen(input)] = '\0';

// Add other data items here.

temp->next = nullptr;

if (head == nullptr)

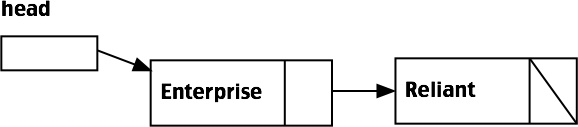
head = temp;

// else code will go here, for adding

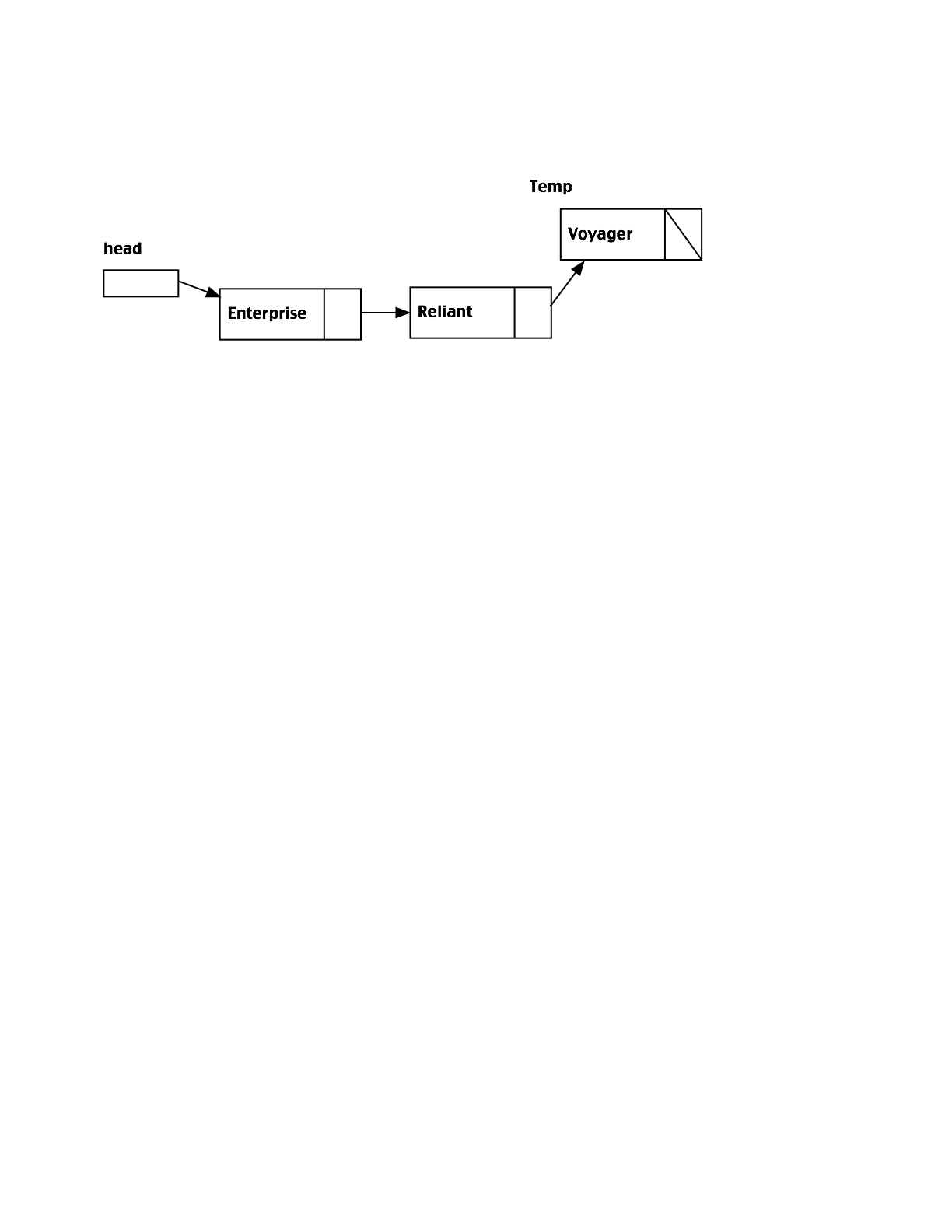
// other ships.

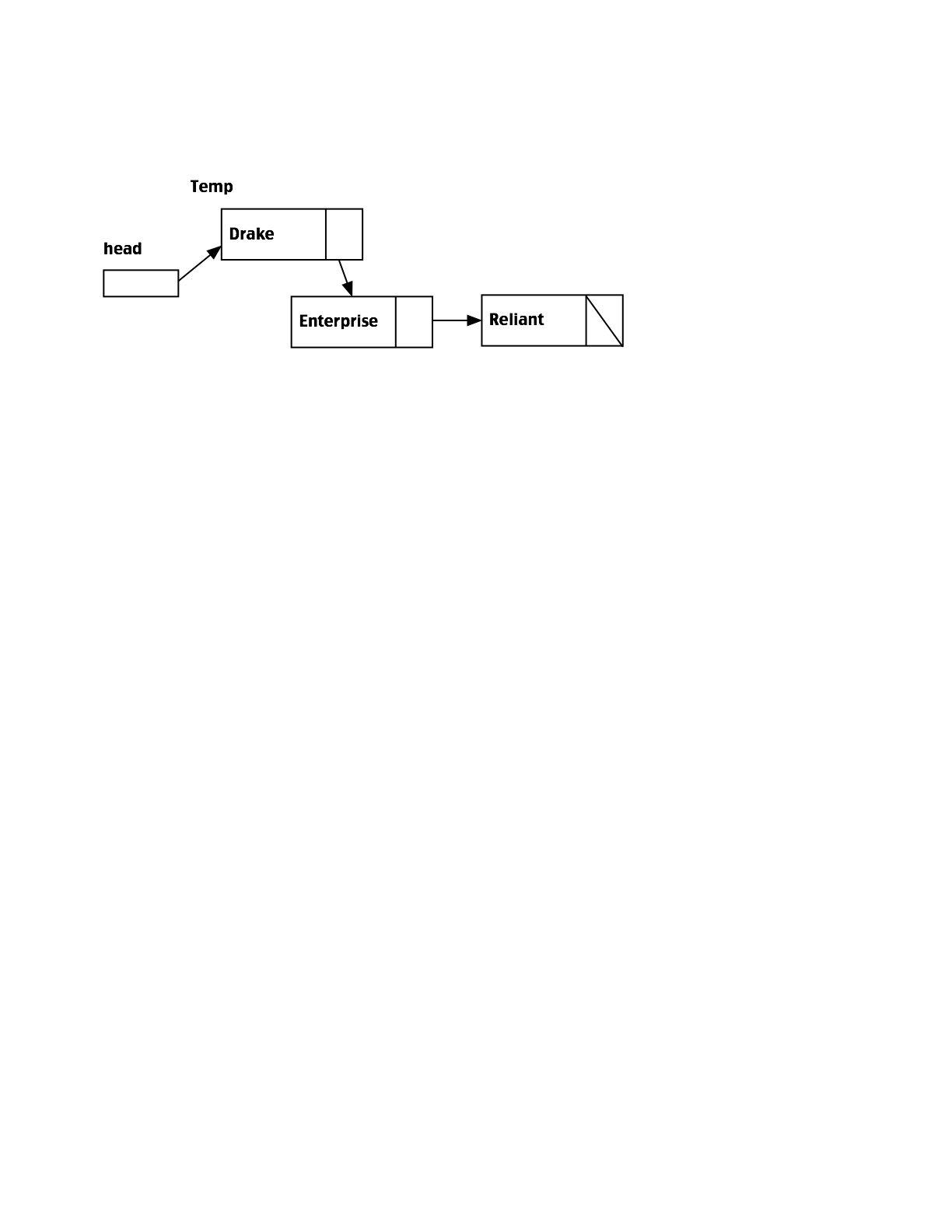
return;

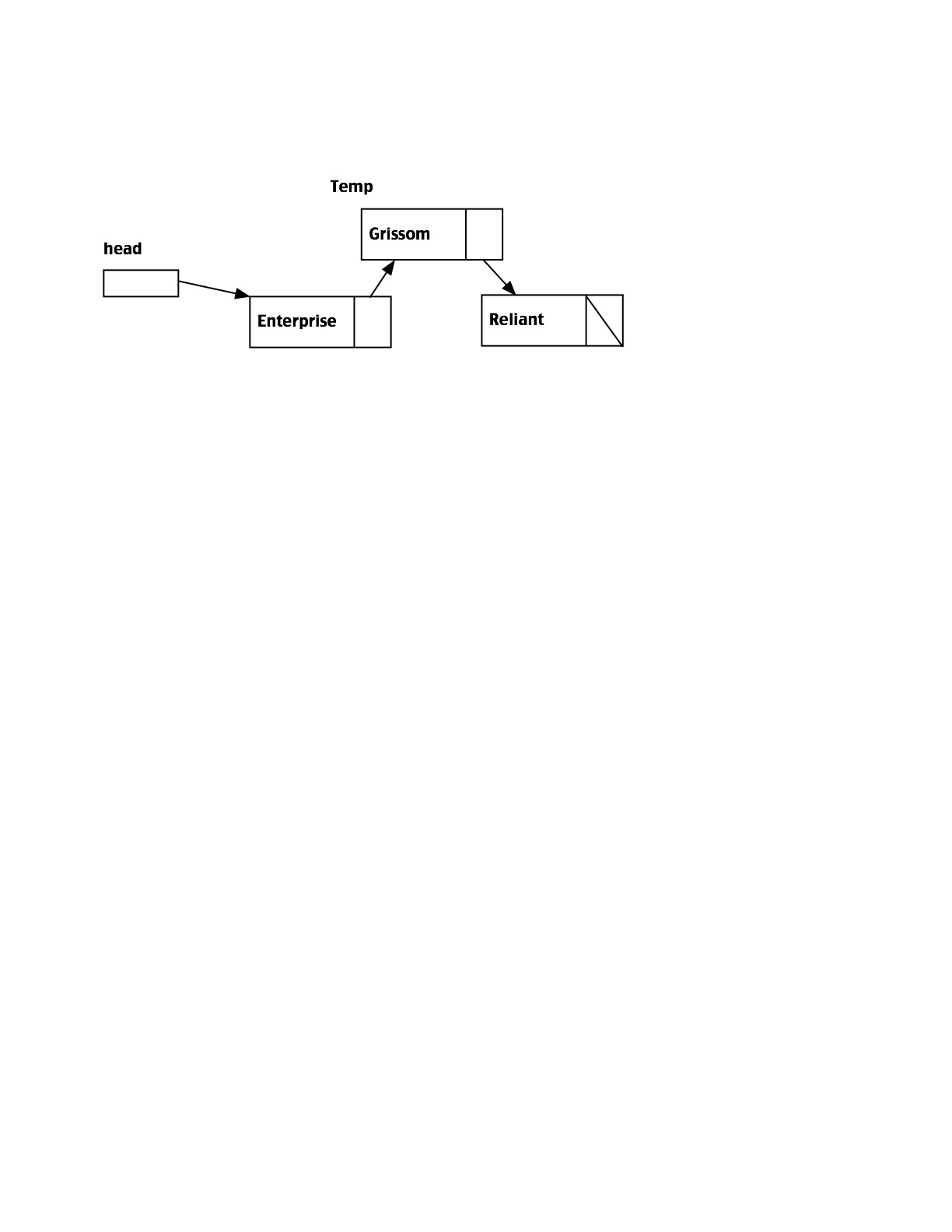
}

* Adding additional ships to the linked list is a bit trickier. Your ship may have a name that places it at the beginning of the list, in which case head will point to the new Ship. If the name is alphabetically at the end, then the new Ship will become the tail of the linked list. Otherwise, it will be inserted between two other ships. This is easier to describe with pictures, so please see left and below. Suppose that the linked list consists of two ships, as pictured to the left.

The two ships in the database so far are the Enterprise and the Reliant. The Reliant is the last ship in the linked list, so it’s Ship \* next is set to nullptr. That is signified with a slashing line in the next pointer field. Now let’s add Voyager.

Notice that Voyager comes after Reliant alphabetically, so Voyager becomes the tail of the linked list. The next pointer for Reliant is made to point at Voyager, Voyager’s next is set to nullptr, and the list is updated. Now let’s add Drake to the original list.

Drake comes before Enterprise alphabetically, and Enterprise is at the head of the linked list. So the head pointer is made to point at Drake, and Drake’s next pointer is set to point at Enterprise, and the list is updated.

The last possibility is to add a ship somewhere in the middle, so lets add Grissom to the original list. Grissom is between Enterprise and Reliant alphabetically. So we need to have the Enterprise next pointer point to Grissom, and have Grissom point to Reliant. The list is then updated.

Deleting a Ship is the opposite of adding a Ship. Suppose we wanted to delete the Enterprise from the list to the right. Then the head pointer would point at Grissom, and the Enterprise can be deleted (remember to delete all the c-strings in the Ship struct, and then delete the Ship). To delete Reliant, just set Grissom’s next pointer to the nullptr and delete the Reliant struct. To delete Grissom, you would set Enterprise’ pointer to Reliant, and then delete the Grissom struct.

**Design Considerations**

Please follow the specifications below and do not deviate from them. Failure to follow the specifications will result in deduction of points.

1. Make sure your program doesn’t have any memory leaks. Use valgrind on the Linux systems, or some other leak checking program or utility to make sure you deleted all dynamically allocated memory before program termination.
2. Use multiple source files and a header file for this project. Name your main file lab5main.cpp, and your additional files appropriately (lab5ClassImp.cpp and lab5Header.h for example). Please upload your data file as well.
3. Please be sure the source file includes your name, assignment description and number, and date, as a program comment. Also include a Sources line.
4. You must use a class to model the collection of starships (class Starfleet, for example).
5. Write methods for your Starfleet class to do the tasks (load the data, add a new ship, search for a ship, etc.). You may write any other methods that you think you need. You may create other nonmember functions, but if you do, they must not have any interaction with data items internal to a class.
6. Use a struct or a class named Ship, or something similar, to model the data for a starship.
7. You must use a linked list to implement the above class (Starfleet). The linked list of structs must be internal to the class, and must be dynamically allocated.
8. You must use char pointers and dynamically allocated c-strings for all string-type items in a Ship struct (or class). You may optionally use a dynamically allocated c-string or an integer for a ship’s registration.
9. You must create a constructor for the Starfleet class that will set the head pointer to the nullptr. Your constructor may optionally take care of other initialization needs, and you may optionally create other constructors with arguments if you wish.
10. You must use a destructor for the Starfleet class that will deallocate all dynamically allocated memory. You may optionally create a destructor for the Ship struct (or class).
11. When using a class, please make sure you encapsulate the data, which means make all the instance data members private and provide accessor methods and mutator methods to access and manipulate the data.
12. Make sure to have a delimiter written between each item in the file when you write the text file, such as a newline or a semicolon. This will be important when you read the information back from the file.

### “Do-Not” List for All Labs in CS162:

* No Global Variables (you can have global constants)
* No use of the stdio library (use iostream and fstream)
* Instead of the string class, you will be using arrays of characters and the cstring library
* No goto statements.

### Things You Should Do:

* Follow the style guide for this class
* Your programs should always guard against bad data being entered by mistake. Bad data means anything that could cause a stream to go into input failure.

### Goals for This Lab:

* Using class to model Abstract Data Type
* OOP-Data Encapsulation
* Breaking tasks down into methods
* Dynamically allocated linked lists.
* File input/output

### How to Submit Your Work:

You may submit your files separately or zipped to the dropbox. Do not use .rar compression. You may upload as many versions as you wish prior to the due date. I will grade the last one unless you tell me otherwise.