Background

The Peidmont Accident Analysts, Inc. (PAAI) board of directors is really, really, basically, almost completely satisfied with the results of our latest release. However, they are frustrated with all of the separate linked lists in the `IncidentLog` class. They recommended that we maintain the same inheritance hierarchy but figure out a way to store all derived objects on one linked list. We are confident that this polymorphic behavior can be accomplished using a linked list of pointers to base class objects. Virtual functions may be required as well; and we are going to add iterators to the linked list class to speed up processing.

Requirements

Modify the `IncidentLog` class declaration as shown in the updated UML diagram. Associated changes required in the `IncidentLog` class implementation code are discussed below and in the milestones.

The new `IncidentLog` class has only one linked list. This linked list is defined to store a pointer to a `HazMat7k` object. The `loadData` function must be changed to pass the address of a derived object to the `appendObject` member function the `IncidentLog` class. The `summaryReport` function of the `HazMat7k` class still has customized output for derived classes. To achieve such polymorphic behavior, that function must be made virtual.

Storing pointers in the `info` data member of our `Node` objects implies that we should consider a memory deallocation strategy. A few of the PAAI senior software architects have been using template specialization in similar situations. We are going to reuse some of their code to see how it will work for this project. A specialized `Node` class declaration and specialized implementation code are attached to the assignment in Blackboard. This code (both the declaration and implementation) should be added to the file `LL.h`. These are **additions**. Do **not** remove your existing `Node` declaration and implementation code.

Add the class declaration (provided) and your implementation code for class `LL_iterator` to the file `LL.h`. Add functions `begin` and `end` (that return `LL_iterator` objects) to the `LL` class. Modify the `displayReport` function in class `IncidentLog` to use iterators. Test running your code using both iterators and the `at()` function to access Nodes for the report output. Record timing data for these tests in the table provided.

Either an updated UML diagram or a new class declaration will be provided for classes that have changed.

Programming Skills

The programming skills required to complete this assignment include:

- **Polymorphism**
- **Virtual Functions**
- **Abstract Base Classes**
- **Linked List Iterators**
- **Template Specialization (not testable)**
- Object oriented design
- Linked Lists, Self-referential classes
- Class Templates
- Pointers, Dynamic Memory Allocation
- Class composition, class inheritance
- Exception handling
- Function overloading, function overriding
- File I/O
- Enumerated data types
- Information hiding
- Object oriented design
- Operator overloading

Recommended Milestones

For this project several milestones are recommended, however you are NOT required to turn anything in or to meet these milestones. Always make sure that your code compiles and runs before starting the next milestone.

Milestone 1 – NLT November 2nd

- Subscribe to the Project #4 Forum
- Create an empty project
- Add all code from your Project #3 submission
- Edit comments with new project number and due date, compile and run

Milestone 2 – NLT November 7th

- Move all code in `IncidentLog.cpp` to `IncidentLog.h`, then remove `IncicdentLog.cpp` from your project
- Add the declaration for class `ProcessTimer` to the `DateTime.h` file
- Add the implementation code for class `ProcessTimer` to the `DateTime.cpp` file
- Use a `ProcessorTimer` object to test the time required to load the data file and the time required to run reports, observe any difference

Milestone 3 – NLT November 9th

- Correct any issues identified from Project #3 grading

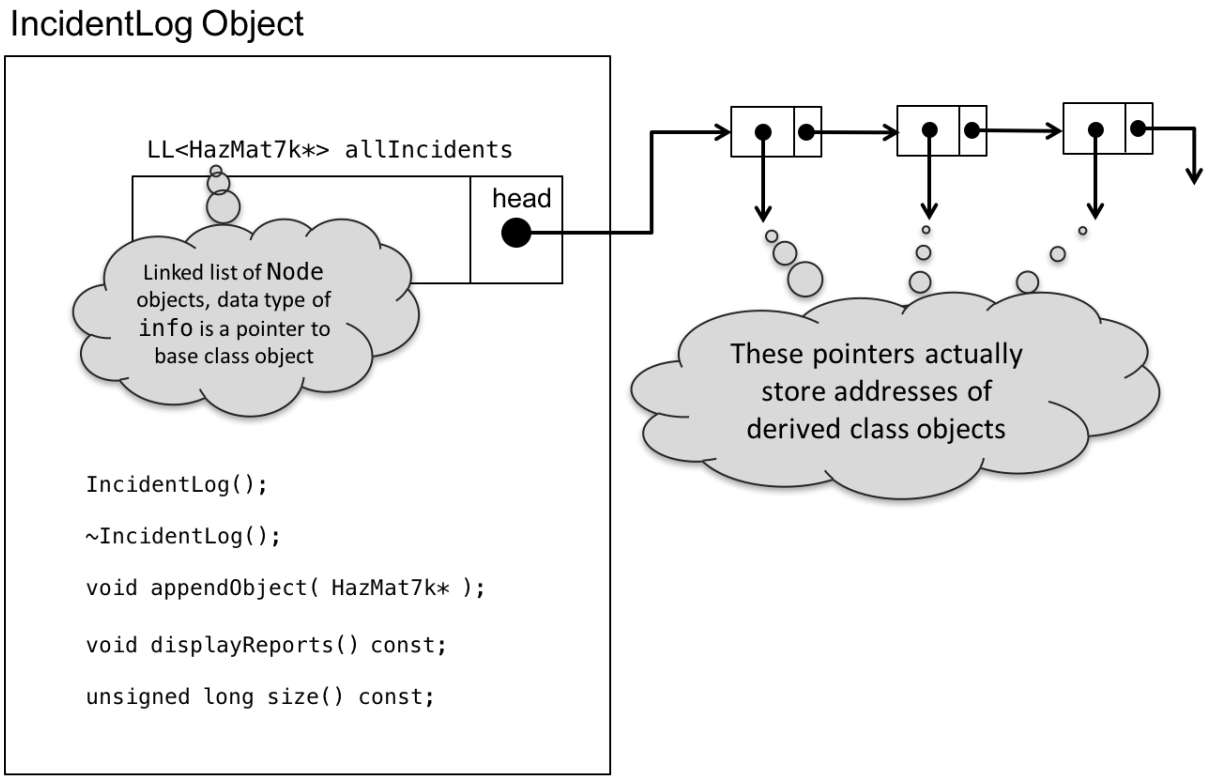
Milestone 4 – NLT November 14th (Read carefully and make changes precisely)

- Modify class `HazMat7k` add new members listed in the UML diagram
- Add provided code for Node specialization to `LL.h`; both specialized class declaration and implementation code go in the `LL.h` file
- Modify class `IncidentLog` (see UML diagram):
    - replace two `LL` objects with one `LL` object that stores a pointer to the `HazMat7k` base class
    - since there is now only one linked list, only one `size()` function is required, it may be implemented in-line and it returns the number of objects that are stored on the one linked list
    - since there is now only one linked list, only one `appendObject` function is required, it may be implemented in-line and may call either the `push_back` or `insert` member function of the linked list class (since ordering of pointer values is not of interest)
    - modify the report display function, it only needs to traverse one linked list now
- Modify the `loadData` function in your driver program, `loadData` still needs to instantiate derived class objects depending on the type of incident; however, now it should pass the address of the derived object to the `appendObject` method of the `IncidentLog` class
- Modify the `loadData` function in your driver program to call `setIncidentCode` with the appropriate argument, before the address of the object is passed to the `appendObject` method (the update should be done here since this is the last time we can determine the correct value for the incident code)
- Compile and run
- Make the `summaryReport` function of the `HazMat7k` base class so that it is a virtual function
- Compile and run, note any difference in output after making the report function virtual

Milestone 5 – NLT November 16th

- Add class `LL_iterator` to `LL.h`, put both the class declaration and the member function implementation code in `LL.h` (Note: The `LL_iterator` declaration must be placed prior to the `LL` class declaration)
- Add functions `begin()` and `end()` to the `LL` class, the functions return `LL_iterator` objects
- Run reports and record timing data in the table provided
- Modify the report display function of class `IncidentLog` so that it uses `LL_iterator` objects, instead of the `at()` functions to control the traversal of the linked list (keep the `at()` function code for later testing)
- Record the time it takes to run reports using the `at()` function and iterators

Graphical Representation of the Modified `IncidentLog` class



Submission Details

What to submit:
One compressed file containing all source code and any other files associated with this project (see list below). The file name should be `<netID>P4.zip`.

```
DateTime.h, DateTime.cpp
PHMSA7000.h, PHMSA7000.cpp
IncidentLog.h
LL.h
<netID>P4.cpp
P4Timing.pdf
```

The driver program should include code to demonstrate the program functionality. The file `P4Timing.pdf` should contain your timing data in a table such as the one below (also posted to Blackboard as word file).

Node Access Technique	Screen Output
`.at() function`	`0.2490`
`iterators`	`0.2002`

Due date/time:
Thursday, 17 NOV 2016, no later than end-of-day (11:59pm). Late submissions will be penalized 2.5 points for each 15 minutes late. If you are over 10 hours late you may turn in the project to receive feedback but the grade will be zero. In general requests for extensions will not be considered. This project is worth 100 points.

Academic Integrity

This is an individual project and all work must be your own. Refer to the guidelines specified in the *Academic Honesty* section of this course syllabus or contact me if you have any questions.
Include the following comments at the start of your program:

```
/*
 * <FileName>.<file extension>
 *
 *  COSC 052 Fall 2016
 *  Project #4
 *
 *  Due on: NOV 17, 2016
 *  Author: <your name>
 *
 *
 *  In accordance with the class policies and Georgetown's
 *  Honor Code, I certify that, with the exception of the
 *  class resources and those items noted below, I have neither
 *  given nor received any assistance on this project.
 *
 *  References not otherwise commented within the program source code.
 *  Note that you should not mention any help from the TAs, the professor,
 *  or any code taken from the class textbooks.
 */
```

Grading

This graded assignment is worth 100 points and will be counted as part of the *Programming Projects* category for the course. A grade rubric will be provided separately.

UML Diagrams

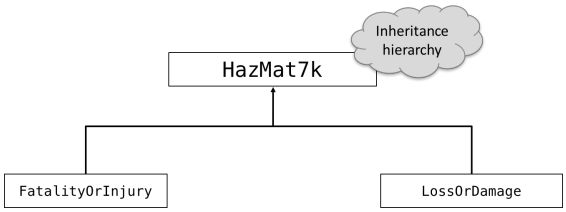
Node
+ info : T + next : Node<T>*
+ Node(T = T(), Node<T>* = NULL)

LL
- count : unsigned long - head : Node<T>*
+ LL() + ~LL() //destructor, must ensure memory is deallocated **//copyList is a private function called by** **//both copy constructor and overloaded = operator (must ensure deep copy)** - copyList(listToCopy : const LL<T> &) : void + LL(otherLL : const LL<T> &) **//copy constructor** + operator=(rhsObj : const LL<T> &) : LL<T> **//overloaded assignment operator** **//member function push_back results in a Node<T> object being added** **//to the end of the linked list requires dynamic allocation of memory** + push_back(T) : void **//member function insert results in a Node<T> object being inserted ~~by~~** ~~**//ascending order (smallest to largest) by incident number**~~ + insert(T) : void **//member function pop_back removes the last list object and deallocates memory** + pop_back() : void **//member function clear removes all objects from the list and deallocates memory** + clear() : void **//member function size returns number of objects on the linked list** + size() const : unsigned long **//member function at, same as at function of vector class** + at(ndx : int) const : T& **//friend function, overloaded stream insertion operator mainly for testing** **//NOTE: The template parameter has a different identifier, this is on purpose** + operator <<(os : ostream&, rhsObj : const LL<F> &) : ostream& + begin() const : LL_iterator<T> + end() const : LL_iterator<T>

Date
- yyyy : int - mm : int - dd : int
+ Date(int = 1923, int = 1, int = 1) //yyyy, mm, dd + Date(const Date&) + setDate(int, int, int) : void //yyyy, mm, dd + setDate(const Date&) : void + getYear() const : int + getMonth() const : int + getDay() const : int + operator <<(ostream&, const Date&) : ostream& + operator >>(istream&, Date&) : istream& + operator ==(const Date&) const : bool + operator !=(const Date&) const : bool + operator <=(const Date&) const : bool + operator <(const Date&) const : bool + operator >=(const Date&) const : bool + operator >(const Date&) const : bool + operator =(const Date&) : Date

Time
- hh : int - mm : int
+ operator <<(ostream&, const Time&) : ostream& + operator >>(istream&, Time&) : istream& + Time(int = 0, int = 0) + setTime(int, int) : void + getHour() const : int + getMinute() const : int + operator =(const Time&) : Time

IncidentLog
- allIncidents : LL<HazMat7k*>
+ IncidentLog() + ~IncidentLog() + displayReports() const : void + appendObject(HazMat7k*) : void + size() const : unsigned long



HazMat7k

```
-  reportReceivedDate : Date
-  reportNumber : string
-  supplementalNumber : string
-  reportType : string
-  operatorID : string
-  name : string
-  operatorStreetAddress : string
-  operatorCityName : string
-  operatorStateAbbreviation : string
-  operatorPostalCode : string
-  localDate : Date
-  localTime : Time
-  commodityReleasedType : string
-  unintentionalReleaseBbls : double
-  intentionalReleaseBbls : double
-  recoveredBbls : double
-  igniteInd : string
-  explodeInd : string
-  preparedDate : Date
-  authorizerName : string
-  authorizerEmail : string
-  incidentCode : char        // New for P4, valid values are:
                              // f – fatality or injury
                              // l – loss or damage
```

Add `char` data member to the `HazMat7k` base class to store a code for the incident type. Also add accessor and mutator functions. This data member will be assigned a value in the `loadData` function.

```
+ ~HazMat7k()

//default constructor
+ HazMat7k()

//convert constructor
+ HazMat7k(const HazMatData&)

+ summaryReport() const : void

+ getIncidentCode() const : char       // New for P4

+ setIncidentCode( char ) : void       // New for P4
```

NOTE: Because of space limitations, accessor and mutator functions implemented in-line are not shown here. The functions are fairly simple, so to save typing the C++ code for these functions will be provided for download from Blackboard.

```
//friend function, NOTE: the output generated is different than that of
//the summaryReport function, the stream insertion operator should output
//all data members on one line
+ operator<<( os : ostream&, rhsObj : const HazMat7k& ) : ostream&
```

LossOrDamage

```
//this class does not contain any non-inherited data members
```

```
//default constructor
+ LossOrDamage()

//convert constructor
+ LossOrDamage(const HazMatData&)
```

FatalityOrInjury

```
- fatal : int
- injure : int
- narrative : string
```

```
//default constructor
+ FatalityOrInjury()

//constructor with parameters
+ FatalityOrInjury(const HazMatData&)

//overridden summary report function
+ summaryReport() const : void

NOTE: Because of space limitations, simple accessor and mutator functions
   implemented in-line are not shown here.

//friend function, NOTE: the output generated is different than that of
//the summaryReport function, the stream insertion operator should output
//all data members on one line
+ operator<<( os : ostream&, rhsObj : const FatalityOrInjury& ) : ostream&
```

Class definition is provided for `LL_iterator` class

LL_iterator
- current : Node<T>*
+ LL_iterator() + LL_iterator(ptr : Node<T>*) + operator*() : T& + operator++() : LL_iterator + operator==(rhsObj : const LL_iterator&) const : bool + operator!=(rhsObj : const LL_iterator&) const : bool

Template specialization for class `Node` (specification and implementation code provided)

Node<HazMat7k*>
+ info : HazMat7k* + next : Node< Hazmat7k* > *
+ Node(Hazmat7k *, n = NULL : Node< Hazmat7k* >*) + ~Node()

Class specification and implementation code provided for `ProcessTimer` class

ProcessTimer
- timeStart : clock_t - timeEnd : clock_t
+ ProcessTimer() + ProcessTimer(clock_t) + ~ProcessTimer() + setTimeStart(clock_t) : void + setTimeEnd(clock_t) : void + getTimeStart() : clock_t + getTimeEnd() : clock_t + getTimeElapsed() : clock_t + operator<<(ostream&, const ProcessTimer&) : ostream& //friend function

(This page intentionally left blank)