2008

TESTING xUML: A STUDY OF IMPLEMENTING AND TESTING MODEL DRIVEN ARCHITECTURE

Dylan O. Flaherty
The University of Montana

TESTING xUML: A STUDY OF IMPLEMENTING AND TESTING MODEL DRIVEN ARCHITECTURE

By

Dylan O Flaherty

Bachelor of Science, Computer Science, University of Montana, Missoula, MT, 2007

Thesis

presented in partial fulfillment of the requirements
for the degree of

Master of Science
in Computer Science

The University of Montana
Missoula, MT

Autumn 2008

Approved by:

Dr. Perry Brown, Dean
Graduate School

Dr. Joel Henry, Chair
Computer Science

Dr. Min Chen
Computer Science

Dr. John Bardsley
Mathematical Science

Flaherty, Dylan, M.S., Autumn 2008 *Computer Science*

Testing xUML: A Study of Implementing and Testing Model Driven Architecture

Chairperson: Dr. Joel Henry

Model Driven Architecture (MDA) is a relatively new and completely different approach to developing software in which diagrams and formal specifications are written and then translated into executable programs. Development using this approach takes longer than using traditional software engineering approaches, but results in products that more correctly meet user requirements and take less maintenance effort. While there is considerable information available regarding development using MDA, the implications for testing have not been fully explored or measured.

In order to investigate this facet of MDA, a traditional C++ program which had previously been tested was rebuilt as an MDA model. Metrics for unit, integration, and system testing on the two systems were then gathered and compared. The MDA model proved to be moderately easier to test than a traditional system; tools are provided to assist testers, allowing testing to potentially be accomplished earlier in the life cycle of a project. Additionally, the MDA approach may make it possible to create a reusable model which could be used for testing many widely different applications with minimal effort required to adapt the model between programs.

Results indicated that developers could expect unit and integration testing of iUML systems to take slightly less time than testing traditional systems. System testing of the MDA model is likely to be more expensive in the short term, but payoff over time. While developing in MDA necessitates overcoming several hurdles and may prove to be too expensive to be practical, ease of testing is not one of MDA's shortcomings.

TABLE OF CONTENTS

TABLE OF FIGURES

ACKNOWLEDGMENTS

I would like to thank the students of Dr. Henry's SQA class for digging through three years' worth of homework to find metrics for use in this thesis: Geddy Tarbell, Chap Alex, and Jonathan Adams.

Above all I would like to thank Dr. Joel Henry for being my mentor throughout my entire academic career, for his help on this work, and for convincing me not to change my major five years ago; it's been fun.

CHAPTER 1 INTRODUCTION

1.1 Introduction

Model Driven Architecture (MDA) is a software development approach in which developers create detailed diagrams and formal specifications which exactly define the states and behaviors of a system. iUML is a tool developed by Kennedy Carter which essentially provides an Integrated Development Environment (IDE) for MDA; within iUML developers can create interconnected diagrams and define formal specifications which can then be translated into code and compiled into an executable program (Raistrick, 2004). Action Specification Language (ASL), an extremely limited and generic programming language, is inserted throughout the diagrams to specify details of how the system works, such as decisions and loops. This system of diagrams and ASL code can be translated into an executable Platform Independent Model (PIM), which can be run within the iUML simulator. The model can later be translated into a Platform Specific Model (PSM) in almost any programming language (commonly C). The iUML tool is extremely complex, with many detailed diagrams that must be interconnected exactly. Consequently, the learning curve for iUML is very steep and the effort to develop a system in iUML is considerably higher than traditional approaches.

The MDA approach has the potential to radically change the way software is developed, especially for safety critical systems. However, the effects of MDA on software quality assurance have not been thoroughly explored. By comparing metrics of testing effort

on an iUML system with testing effort for a traditional C++ system, this thesis provides a starting point for analyzing how to approach testing of iUML models, and a gauge for the expected time to complete testing.

The model which was created for these tests is as similar as possible to a C++ system for which unit, integration, and system testing metrics were available. After testing the iUML model and gathering metrics, the metrics were compared with metrics from testing the original C++ program.

This work draws ideas from NASA's Orion project, which used iUML for development (Henry, 2007). Approaches identified for testing iUML used within the Orion project are partially applied to the model and the usefulness of these approaches is evaluated, including a new approach to system testing.

Some of the unique features of iUML were explored over the course of testing. Many of these features were found to have considerable potential and could save significant testing time. This potential is discussed in the conclusions section.

1.2 About iUML

Tools which support MDA through the use of executable UML models are called Executable UML, or xUML. Kennedy Carter's implementation of xUML, which they call

iUML, gained a considerable amount of attention recently when NASA announced that Lockheed Martin would be using iUML for NASA's Orion project (Lowry, 2008). It is likely that NASA was most attracted by MDA's aptitude for exactly specifying system requirements –a necessity for critical systems used within Orion. Most of MDA's key strengths also target development of critical systems. Orion is likely the largest system for which the MDA approach has been employed, but several smaller, proof of concept applications have been developed with it as well (Customer Success, 2008) .

MDA was created by the Object Management Group (OMG) in 2001. One of the main goals of MDA was to provide exact requirements specification for projects. "Where implementation is to be carried out by an external party, an executable PIM forms an ideal specification to hand over to the implementers since it fully specifies required behavior" (Raistrick et al., 2004). A large proportion of software projects fail because the developers created the wrong product: the developers created a requirements specification which their customers approved, but when the project was complete or nearly complete it was discovered that ambiguities or incompleteness in the requirements document had caused the project to stray from what the customers wanted. By creating a system which forces developers to fully specify every aspect of a system, the OMG hoped to solve the problem of incomplete and incorrect requirements. Moreover, since with MDA the system is the requirements specification, there is no chance of a mistake occurring in the translation from requirements to

implementation (a translation that produces significant defects in traditional software development processes).

A second goal for MDA was to provide a level of abstraction between the system definition and the code implementation (Mellor, Not Dated). New programming languages are created every year, and often new languages are faster, more secure, or otherwise improve on their predecessors. The iUML system allows the system definition to be translated into any language, so a system could be translated into a language which does not even exist at the time the system was originally designed.

A third major goal was to reduce the amount of required experience for developers. Rather than writing code, most developers using iUML would create models of the system using terms understood and widely used within the application area –a process which iUML's creators hoped would be more intuitive, complete, and correct.

1.3 Structure of an iUML Model

The iUML program is divided into two main components, the modeler and the simulator. The modeler is the development environment component of iUML. This is where diagrams are created and code is embedded into the diagrams.

The simulator is the portion of iUML which simulates running the PIM. The simulator environment has several tools included which allow users to monitor the model as it runs; these tools significantly improved test activities.

An iUML model is defined by 6 different types of diagrams. The diagrams are largely interconnected; a change in one diagram often results in the change automatically being reflected in another diagram.

Within the modeler portion of iUML is a “Projects” tree and a “Domains” tree. The projects tree is primarily for configuration management purposes. The projects tree contains a domain model diagram, use case diagrams, and sequence diagrams. The projects tree also contains the code for bridges between domains.

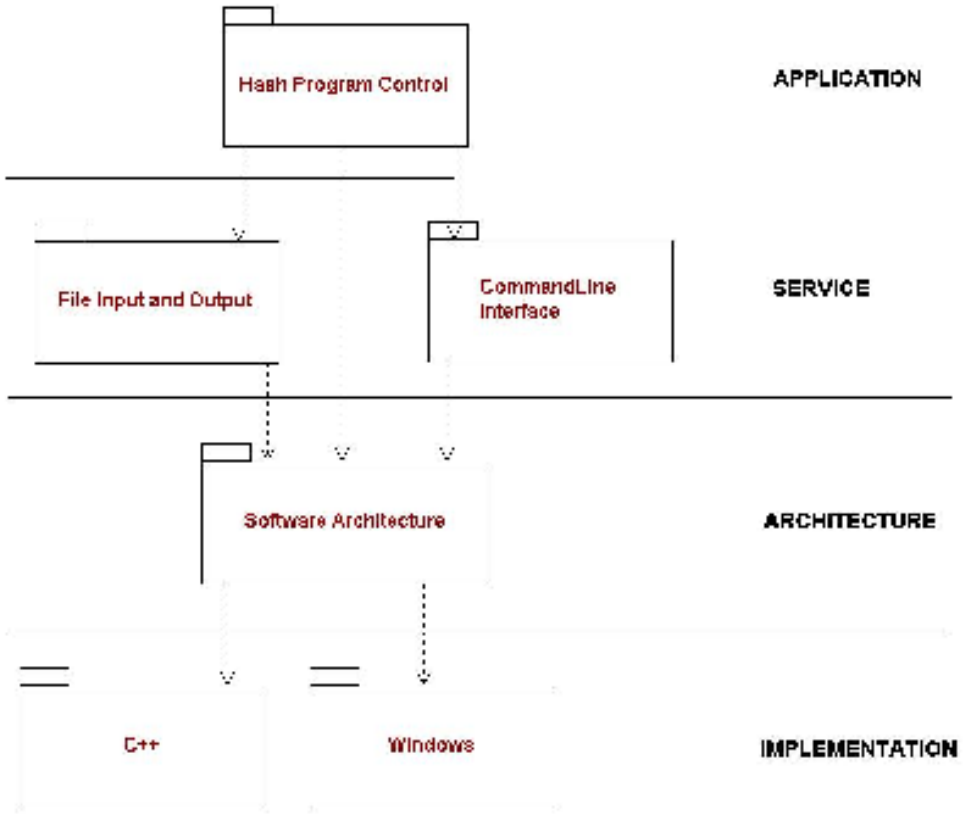

Figure 1: The Domain model for the system.

An iUML model is made up of one or more domains. The domain model (Figure 1) shows the set of domains for a single project. Arrows between domains indicate the client/server relationship between the domains. In an iUML model with multiple domains, domains are connected together via bridges. Bridges encapsulate the interfaces between domains into a single location. Properly implemented domains are not dependant on one another (iUML Modeller User Guide, 2003) because the coupling between them is contained within the bridges. This is done for a number of reasons; by decoupling the domains, domains can be developed in parallel and integration testing can be performed at the bridges. Additionally, keeping the domains well encapsulated and decoupled results in domains which are highly reusable (iUML Tutorial , 2003). The iUML design process naturally encourages proper domain separation by first allowing designers to create a domain model, and then forcing them to consider every interaction between domains when sequence diagrams are created (Luz, Not Dated).

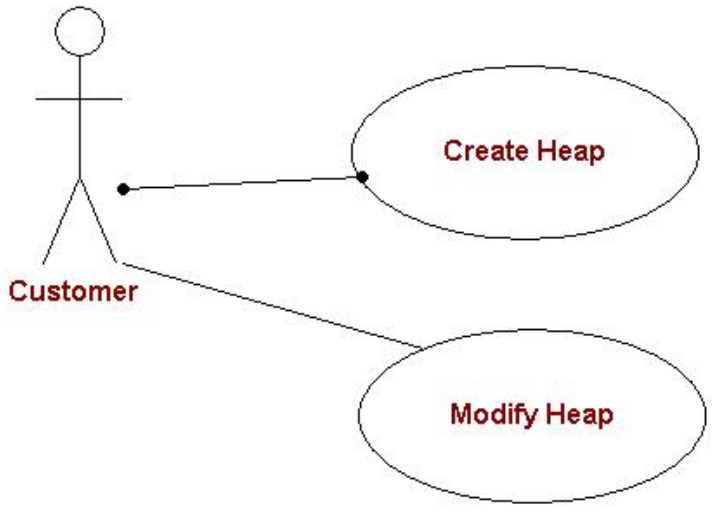

Figure 2: A use case model for a system.

Use case diagrams (Figure 2) and sequence diagrams (Figure 3) are components of the projects tree portion of iUML, they are not translated into executable code. Instead, these diagrams provide more exact specification of the system being developed. The use case diagram specifies what users will need to do with a system, from a black box perspective. The diagram is composed of actors, use cases, and communications which tie an actor to one or more use cases.

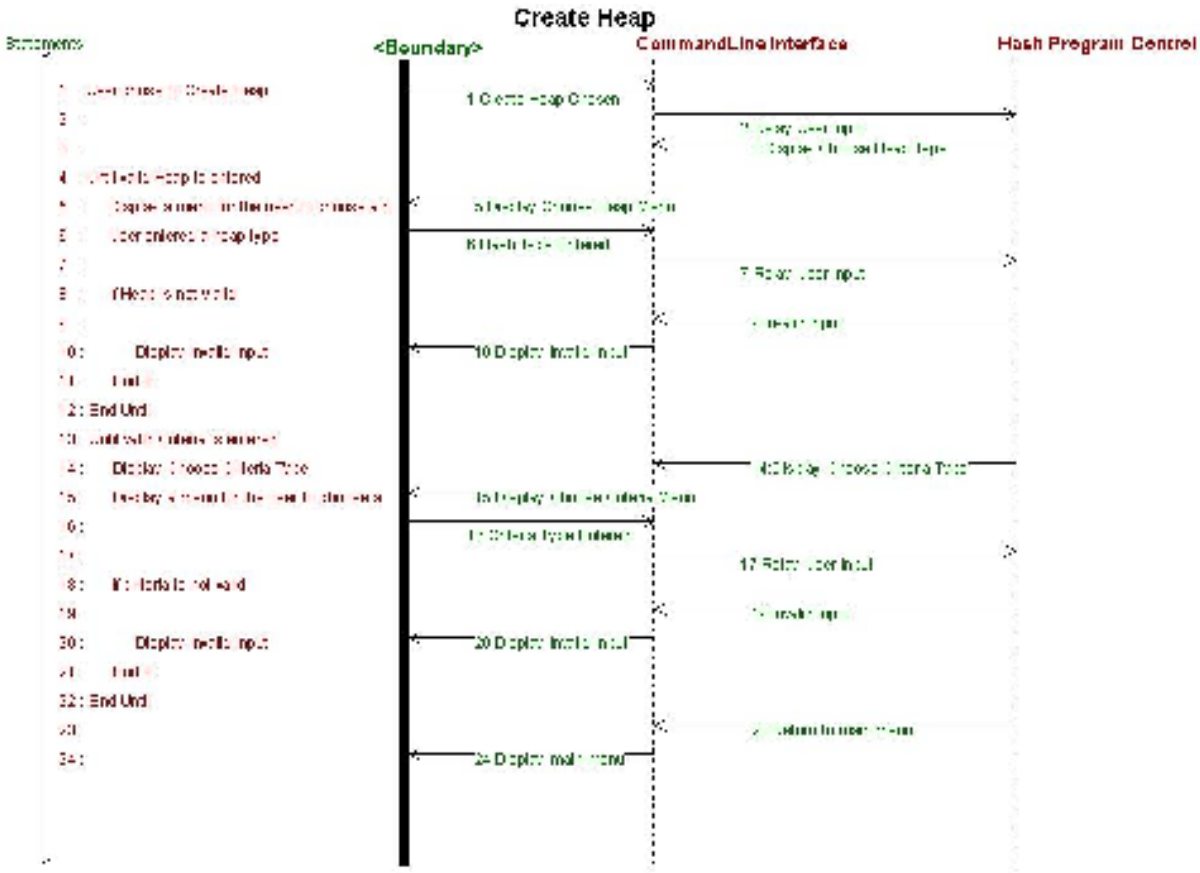

Figure 3: A Sequence diagram for one use case of a model.

Sequence diagrams (Figure 3) are used to describe the interactions between domains. Creating this diagram prior to development of the domains portion of the project helps to ensure that domains are correctly partitioned. This diagram is intended to be a tool and a reference when developing the rest of the model. Additionally, the sequence diagram can be a useful tool when planning integration testing.

The domains tree contains class diagrams, class collaboration diagrams, and state machines. The domains tree is the portion of the program which is compiled to create the executable model. The work in this thesis focuses on the domains portion of iUML.

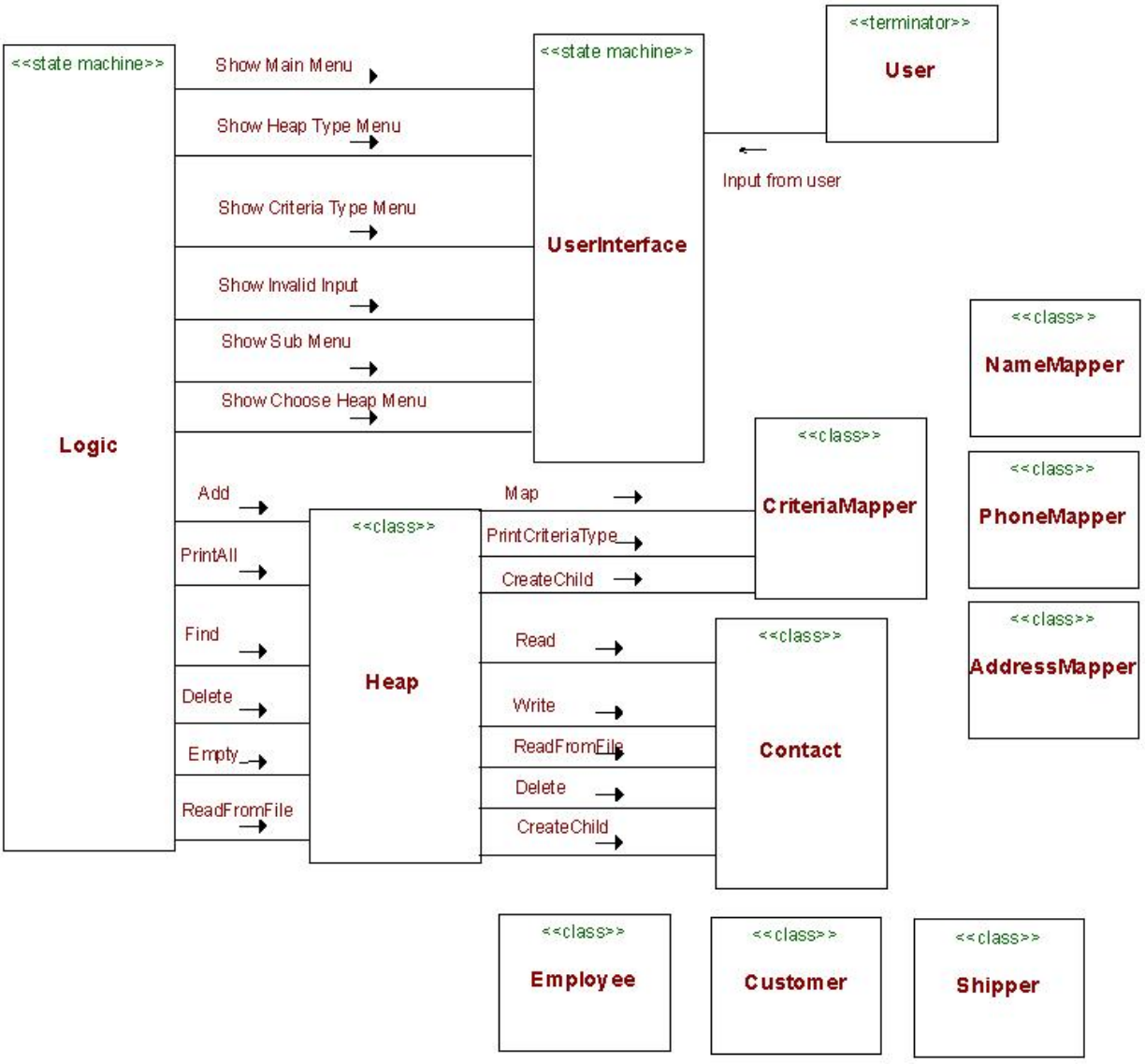

Figure 4: Class Collaboration Diagram

As shown in Figure 4, the class collaboration diagram shows the operation calls and signals that are sent between the classes. Operation calls are solid arrows, signals are outlined arrows. There are both classes and state machines on the class collaboration diagram. For each state machine in the diagram there is also a state machine diagram as shown in Figure 5.

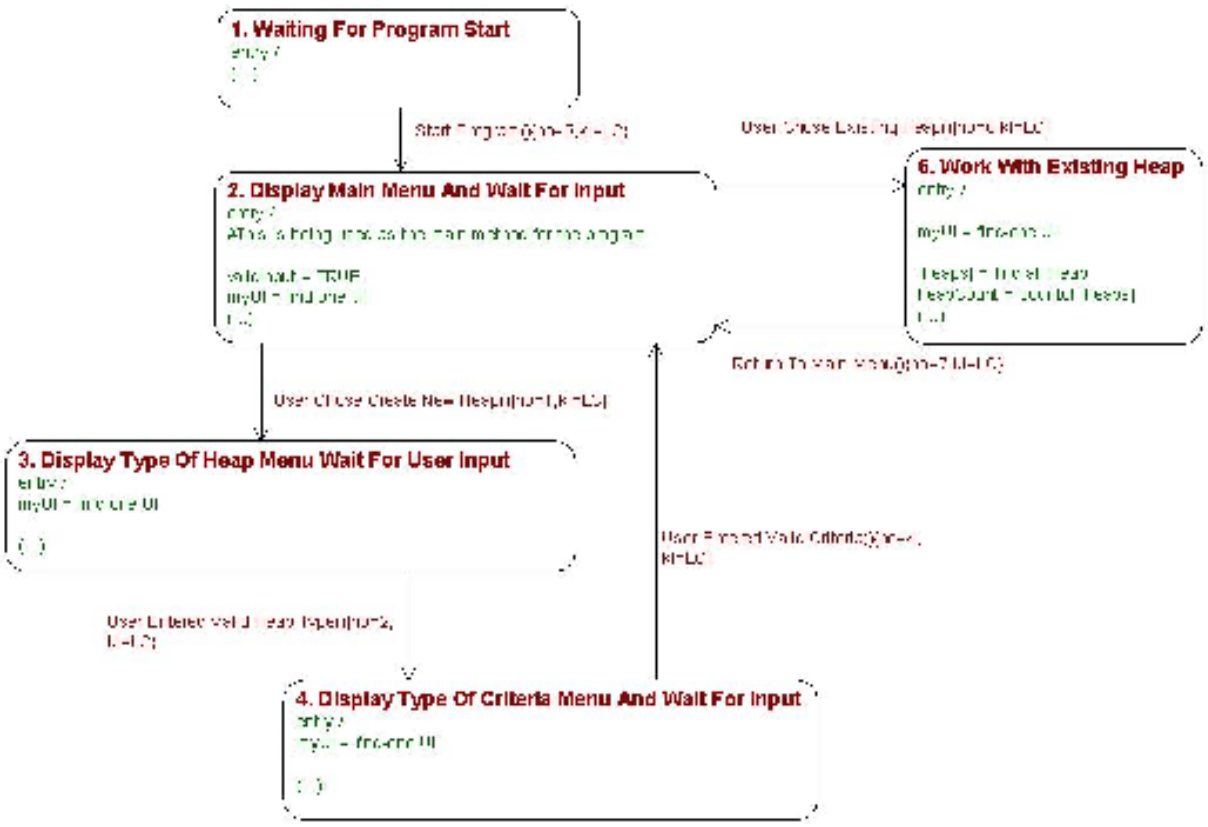

Figure 5: A State Machine from iUML

ASL code is embedded within the diagrams in many different places. Unlike a traditional program, code is not limited to being within operations. Within the model ASL code can be inserted into the bodies of operations, but ASL code can also be inserted into the states of State Machines, dedicated "exception code" areas throughout the model, and other places.

Inline sections of native code can be inserted within ASL. These sections of native code are just as powerful as they would be in a traditional program that is implemented in that language. This is a potential weakness of iUML, since the potential exists for

developers to abuse the inline code to circumvent the restrictions and complex development of iUML. There is limited ability for data to be transferred between the ASL and inline code; generally only basic, language-standard types can be moved between ASL and inline code.

The Class diagram in iUML (Figure 6) has more in common with a diagram of database tables and foreign key mappings than it does with a traditional UML diagram. This is because the backend of iUML is in fact a database. Composition relationships between classes are not permitted in iUML (it is impossible for an attribute of one class to be an instance of another class). Instead, associations are used to connect classes and queries similar to SQL are used to find related instances.

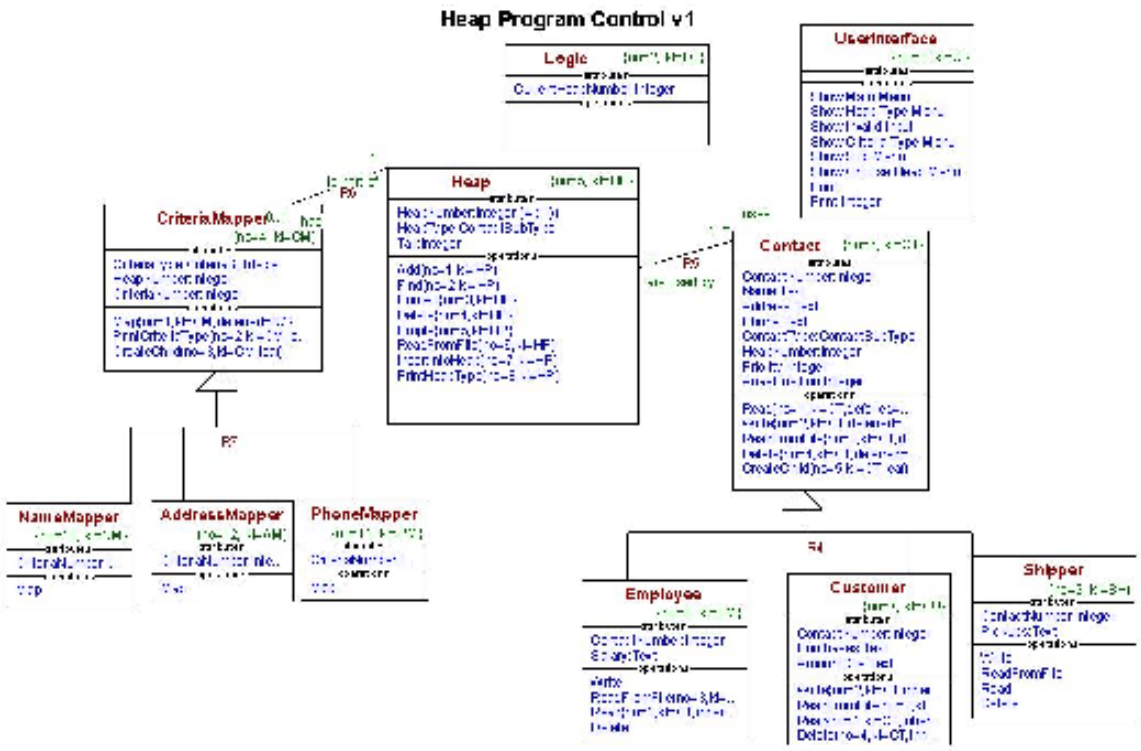

Figure 6: A Class Diagram in iUML

Although it is not documented in any of the iUML user guides, some translation may be necessary to convert values between inline native language code and ASL. This may have been a roadblock encountered during development of Orion. Kennedy Carter provides a code generator to convert the model to native code, but the generator often does not support all of the features of the native language (Hoffman, 2008).

Kennedy Carter's intent was for the Modeller interface to be more intuitive and provide a better understanding of an overall system than traditional code. Several syntax features, such as allowing spaces within operation names, help iUML to be more appealing to non-programmers. In theory, someone with minimal programming and

software engineering experience should be able to design a system simply by creating and connecting the diagrams.

The iUML Simulator allows users to run the PIM and provides functionality to observe the model as it runs. The simulator has some of the same debugging options as normal IDEs (break points, step over, etc). Additionally, the simulator allows monitoring of the instances of classes and tracing of state machines. These functions aid greatly in testing.

CHAPTER 2 METHODS AND TEST SETUP

2.1 Original Program Testing

In Autumn of 2006 Dr. Henry's Software Quality Assurance class was given the source code for a program written in C++ and instructed to perform Unit, Integration and System testing on the code. For each of the three types of testing the class documented the time it took them to write and execute the tests.

The source code which the SQA class performed testing on, "HeapContacts," was a relatively simple program consisting of 12 classes and approximately 3100 lines of code (much of it comments). HeapContacts provided users with a series of command-line menus, from which they could choose to create lists of contacts which were sorted using different criteria. Users could create a new list, add a contact to the list, search for a contact, print all contacts, delete a contact, empty the list, read in contacts from a file, or delete the list. The program demonstrated several object oriented concepts: it utilized two inheritance hierarchies and relied heavily on polymorphism.

2.2 Creating the iUML Model

In order to evaluate testing on an iUML system, a model was created in iUML which was nearly identical to the program tested by the SQA class. This model was subjected to the same tests as the original C++ code, and the same metrics were gathered.

The model was kept as simple as possible while maintaining equivalence with the original program. While the iUML system uses 6 types of diagrams to completely define every aspect of a program, only 3 diagrams are necessary for implementation and code generation; the other diagrams exist for requirements specification and configuration management purposes. Only the Class Diagram, Class Collaboration Diagram, and State Machine Diagram are used to create the iUML PIM implementation of the HeapContacts program.

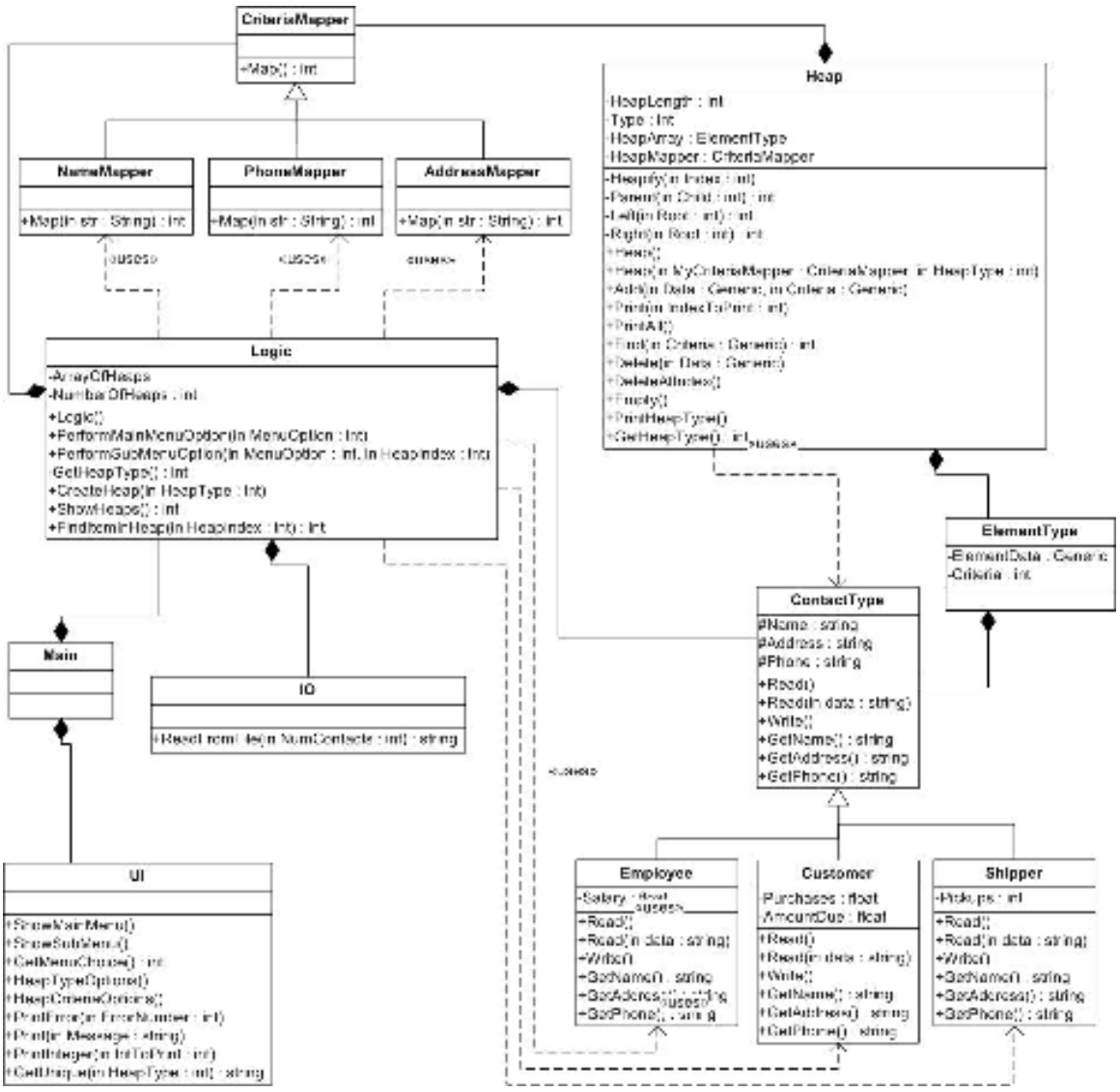

Figure 7: UML Diagram of the original HeapContact implementation in C++

Figure 7 shows the class diagram from the original HeapContact program. Several language-specific features which are impossible to reproduce in iUML can be seen: extensive use of composition (solid black diamonds), generics (within the ElementType and Heap classes), and inheritance hierarchies (triangles).

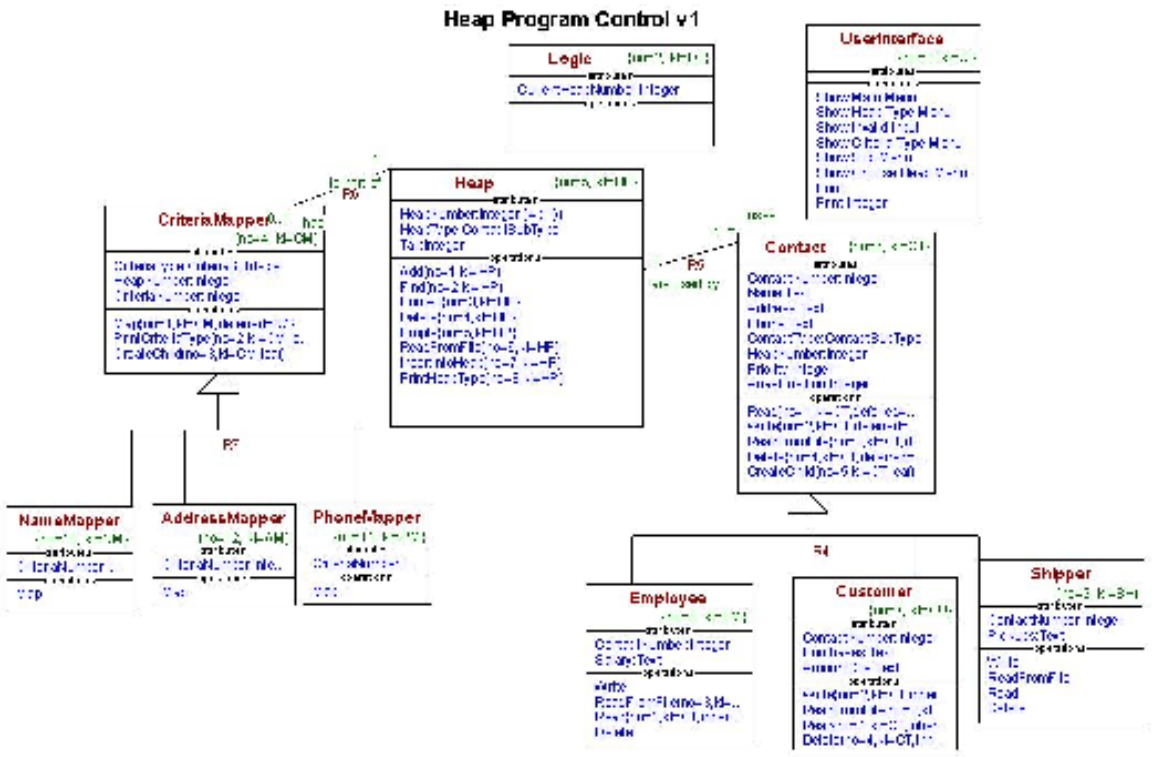

Figure 8: UML Diagram of the HeapContact implementation in iUML

Figure 8 shows the implementation of HeapContact within iUML. This implementation is clearly different from the one in figure 7, but from a black box perspective the programs are identical. The implementations were kept as similar as language differences would allow. The triangles in the diagram (relationships R7 and R4) represent generalization, as opposed to inheritance as shown in Figure 7. According to Wilkie [2003], the difference between traditional inheritance and generalization is: *"UML defines generalization simply as a taxonomic relationship between elements… Inheritance is an implementation mechanism that operates on the generalization hierarchy."* In a generalization relationship, the "child" classes do not inherit the attributes of their parents, instead there is a relationship between a child and a parent, and both are necessary in order to provide the attributes and operations of each. This is

one example of a significant language difference that resulted in a difference in the iUML implementation of HeapContact. Other differences and changes are explained in Section 2.3 below.

2.3 Basis for comparison/effort to keep models comparable

The iUML HeapContact model was kept as similar as possible to the original design. However, minor changes were necessary due to the original program's use of C++ specific features and the limitations of iUML. All changes from the original design are detailed here.

One of these factors which dictated changes from the original program design was restrictions encountered within the iUMLite tool, the free version of iUML which was used for this thesis. Several of these free version restrictions were encountered during development of the model and had to be worked around. The main restrictions which hampered development of the model were that models may contain no more than 10 attributes per class and 8 operations per class. The impact of iUMLite restrictions on this work was that the design needed to be reengineered to have fewer attributes and operations in some classes; this generally meant combining two or more operations together. A complete listing of the restrictions on the free version of iUML can be found in the appendix.

2.3.1 The Logic Class

Logic is a class which is intended to implement the business rules portion of the program. By definition, business rules define the application specific interactions between the classes in an object-oriented design. In a properly designed system with minimal coupling (which is what this example program strives for), most of the "integration" portion of a program should be contained within the business rules portion of the program –in this case, the logic class.

Logic in original HeapContact	Equivalent in iUML Model	Explanation
Logic()	Model Initialization Segment	iUML does not have constructors. Instead, this code can either be placed within the model initialization segment (equivalent to a main method in traditional programming), or within an initialization segment in the state chart.
PerformMainMenuOption	Logic State Chart	Integration code is better suited to the state chart than operations.
PerformSubMenuOption	Logic State Chart	Same as above.
GetHeapType	Logic State Chart	Same as above.
CreateHeap	Logic State Chart	Same as above.
ShowHeaps	Logic State Chart	Same as above.
FindItemInHeap	Logic State Chart	Same as above.

The fact that the iUML version of Logic accomplishes the same tasks as the methods in the C++ Logic implementation, but without the use of any operations, speaks to the capacity of the model to contain operation code in one portion of the model, and integration code in an entirely different portion of the model. This can correctly be viewed as taking encapsulation to a new level.

The implications for testing a system that is designed this way are significant. A developer could easily adopt the standard that most operations must be suitable for unit testing, and integration code must be put within the state charts or initialization segments. The advantage of such a standard is that testers could proceed from the assumption that most operations can be unit tested, unlike the original system, in which none of logic's methods can be unit tested, since they all include integration code.

2.3.2 The UI Class

Minimal changes were needed to the structure of the UI class.

UI	Equivalent in iUML Model	Explanation
ShowMainMenu()	Show Main Menu()	No Change.
ShowSubMenu()	Show Sub Menu()	No Change.
GetMenuChoice()	NA	Included in other menu operations to reduce coupling.
HeapTypeSelection()	Show Heap Type Menu()	No Change.
HeapCriteriaOptions()	Show Criteria Type Menu()	No Change.
PrintError()	Show Invalid Input()	No Change.
Print()	Print()	No Change.
PrintInteger()	Print Integer()	No Change.
GetUnique()	NA	Implemented within Heap.find() because the 8 operations per class limit in iUML was exceeded.

2.3.3 The Heap Class

Significant changes to the Heap class were necessary when the model was recreated in iUML. The original implementation of this class leveraged a number of language features in C++ which were impossible to reproduce in iUML. Among these language

features were the use of generics and arrays. However, the iUML implementation also utilized a number of features which are unique to xUML (SQL queries, default values). The result was two implementations which did not necessarily have direct connections between them, but which were of comparable complexity. Even where methods accomplished roughly the same task, implementation was radically different. This is to be expected when using such different solution approaches and significantly different tools.

Heap	**Equivalent in iUML Model**	**Explanation**
Constructors()	NA	Replaced with default values in iUML
Heapify()	InsertIntoHeap()	No Change.
Parent()	NA	Replaced with SQL Query
Left()	NA	Replaced with Stack Operations
Right()	NA	Replaced with Stack Operations
Delete()	Delete()	No Change.
DeleteAtIndex()	NA	Replaced with SQL Query
Empty()	Empty()	No Change.
Find()	Find()	No Change.
Add()	Add()	No Change.
Print()	Print()	No Change.
PrintAll()	PrintAll()	No Change.
PrintHeapType()	PrintHeapType()	No Change.
GetHeapType()	NA	Not necessary since generics weren't used.
NA	ReadFromFile()	Originally implemented within IO.ReadFromFile, iUML implementation does not have an IO class due to lack of arrays.

Due to difficulties with ASL and embedded C code in the iUML model, some heap operations proved impossible to reproduce. For example, the math.h C library could not be imported, which was necessary for math.floor() and modf() –operations necessary for typical array-based heap implementation (where the location of the parent node is given by: Floor((currentNodeIndex – 1) / 2). The link-list-based implementation of a

heap also could not be implemented due to difficulties with the algorithm needed to find the next empty spot at the base of the heap. Consequently, a stack data structure was implemented instead of the heap. While this did simplify the model somewhat, the differences were minimal (from a testing perspective) since the only methods that were eliminated (left and right), were very small.

Extreme difficulties with file input/output operations in C code embedded within ASL prevented file interactions from being implemented as they were in the original program. Instead, test file contents were hard-coded into the program. Additionally, limitations of ASL (lack of arrays), prevented this code from being implemented somewhere other than within the heap class; the FileIO class was consequently not implemented in iUML. Again, the focus was to reproduce the C++ program where possible but more importantly meet the same functional requirements with both programs (C++ and iUMLite).

2.3.4 The Contact Class

Changes to the Contact class were mostly minor and stemmed from language differences. Some extra operations were included in the iUML version in order to decrease coupling due to differences in how inheritance/generalization are supported in iUML.

Contact	Equivalent in iUML Model	Explanation
Read()	Read()	No change.
Read(String)	ReadFromFile()	There is no overloading in ASL
Write()	Write()	No change.
GetName()	NA	There are no private attributes in iUML, making getters unnecessary.
GetAddress()	NA	Same as above.
GetPhone()	NA	Same as above.
NA	Delete	There is no inheritance in iUML, only generalization, which makes this operation necessary.
NA	CreateChild	The lack of inheritance in iUML means that locating this operation here greatly reduces coupling.

2.3.5 The CriteriaMapper Class

The CriteriaMapper class is essentially the same, with some minor changes to reduce coupling.

CriteriaMapper	Equivalent in iUML Model	Explanation
Map()	Map()	No change.
NA	PrintCriteriaType()	Used in conjunction with Heap.PrintHeapType to reduce coupling.
NA	CreateChild()	Used to reduce coupling.

2.4 Final Comparison of Models

After development of the iUML model was complete and the necessary changes from the original design were implemented, the resulting iUML model was of roughly equivalent complexity to the original C++ version of HeapContact: the original C++ implementation contained 43 methods in 14 classes while the iUML implementation contained 39 operations and 5 states in 11 classes. Changes to the level of coupling meant that a smaller portion of the program would be appropriate for unit testing, and

a larger portion of the program would need to be integration tested. This is detailed in Chapter 4.

CHAPTER 3 MDA IMPLEMENTATION FINDINGS

While this thesis was primarily focused on the processes and effort involved in testing xUML via iUMLite, an enormous amount of time was spent developing the model before it could be tested. The lengthy implementation process revealed many strengths and weaknesses in iUML. The weaknesses listed below were mostly unexpected and in some cases radically delayed implementation of the model. Some of these weaknesses were cases of iUMLite errors, which indicates that the iUML system may not be fully mature.

3.1 Strengths

Throughout the development process the configuration management features of iUML were apparent. Developers must check out the portions of the program they wish to work on, and must check those portions back in with a description of changes when they're done.

The task of developing the diagrams, which must be logically connected, naturally results in a more well-thought-out design process for the system. The iUML process makes it impossible for developers to bypass the design stage and begin coding. No code can be written until at least some part of a model is completed.

3.2 Weaknesses

Operation calls in ASL require both the name and number of an operation (UML ASL Reference Guide, 2003). This means that a small change in a diagram can necessitate sweeping changes in the ASL. This problem is made worse by the fact that ASL code can be inserted in so many different places throughout the model, making it difficult to find the areas that are impacted.

There were numerous user interface quirks throughout iUML which made using the program extremely frustrating. Call operations are auto-detected and easily synchronized between the collaboration diagram and the class diagram. However, there is no facility for dropping a call operation into ASL, which should be easy (and is in other IDEs). Instead, it is necessary to manually check the call operation invocation number and key letters, then return to the ASL code and type in the lengthy call statement. Consequently, developers must constantly switch between diagrams, an operation which should be quick and simple but is not.

The ASL development environment is not user friendly (it is just an area to type text into). There is no auto-detect of any kind and any syntax errors are not found until compile time. Use of external code editors is supported, but was found to be buggy. iUMLite code does not support any ASL math operations (addition, subtraction, floor, etc). This differs from other languages which serve similar purposes, such as IBM's OCL which does provide math operations and is also designed to be embedded within UML

in an MDA project [Catalog of OMG Modeling and Metadata Specifications, 2008]. This extreme limitation of ASL forces INL code to have a greater role in the project, since it becomes the only way to implement much of the functionality.

ASL error messages are often cryptic, and the line numbers where errors are claimed to be are never correct because lines are offset by auto-generated code. The amount of auto-generated code is not consistent, so compensating for the offset does not help.

Inheritance is not supported, only generalization (iUML Tutorial, 2003). For example, an object cannot be passed to an operation as its base type. Despite this limitation, Polymorphism is supported through some auto-generated code, but implementation is significantly different from traditional coding and must be carefully managed to prevent excessive coupling. Several other common programming practices are also not supported in iUML, including overloaded operators and typecasting.

A number of obvious bugs were found in the iUML environment, they are documented in the appendix.

3.3 Implementation Results

The model used here is not especially complicated. The 14 classes in the original system likely would have taken less than 3 hours to develop using C++, C#, Java, or another modern object oriented language. Implementation of this model in iUML took an

estimated 350+ hours, a testament to the steep learning curve and difficulties in using iUML.

While iUML attempts to allow non-programmers to design the majority of the system without ever needing to write any code, it is far too complicated for inexperienced developers to work with. If anything, the complexity of iUML requires even more software engineering experience than traditional software design.

According to Hoffman [2008], Lockheed Martin halted the use of iUML as the development tool for NASA's Orion project in August of 2008 and the project began using another tool instead. The specific reasons cited for the switch were difficulties with inter-domain modeling and problems with the C++ code generator, although an extensive list of problems was cited. The team had started training on iUML in February 2008, so this tool change came after a significant investment had been made in iUML.

Having invested so heavily in iUML, Lockheed Martin would only switch to another tool if they believed that it would be more expensive to fix the many problems with iUML than it would be to discard 7 months of work and start over. The benefits of using iUML were significant enough to entice Lockheed to make the initial decision to chose iUML as a development tool, but after development started Lockheed reached the conclusion that iUML was simply not mature enough to be used for their project.

After completing the model for this thesis a conclusion similar to Lockheed's was reached. In order for this relatively simple model to be completed it was necessary to constantly work around bugs and limitations in the ASL code. Bugs in the interface and the fact that the interface was generally very difficult to use made the problems even more frustrating. The iUML system has excellent goals and great potential, but the software itself behaves like an alpha release and seems to be missing many obvious features.

CHAPTER 4 TESTING THE MODEL

In addition to allowing placement of code within diagrams, iUML provides for the creation of test methods and test method sets within their own forms (located within the domains portion of the modeler). These test forms were used exclusively for unit and system testing.

While simulating a model, iUML has a feature which allows testers to monitor the instances of objects. This display reflects the database implementation of the backend of iUML; instances are shown each on one row of a table, with an attribute in each column. Additional columns indicate whether the instances are linked to other object instances in an association relationship. The instance monitoring software is considerably easier to use and more detailed than equivalents in some Integrated Development Environments (i.e. Visual Studio, NetBeans). This feature was used heavily for both unit and integration testing of the HeapContact program to check values as the test scripts progressed.

Having test scripts rely on the instance monitoring feature for output was convenient and reduced the amount of time needed to write tests, since it was not necessary to write output code. Reducing the time needed to write tests may make earlier, more thorough, and more frequent testing easier in iUML. It should be noted that this will not help fully automated tests, but automated testing was not possible either for this implementation of the HeapContact program or the original C++ implementation, due to

the amount of user input required and the poor degree of encapsulation of that user input.

Use of instance monitoring within test scripts was not only convenient; to a certain extent it was necessary. ASL is incapable of outputting values to the user, which means the only other means of outputting values from test cases would have been the heavy use of inline native language code. If native language code was used for this purpose it would not only be extremely time consuming (writing inline NLC and moving values between the ASL and NLC is difficult), it would also break the test scripts if the model switched to a different native language.

The iUML Simulator has a very interesting feature called Record/Playback. This feature allows a tester to record all interactions with the simulator, including command-line input, to a .rec file (essentially a script). Later, the tester can choose to playback the script and every action the tester took previously will be repeated exactly. This feature has the potential to be very beneficial for testing. Test plans often have tests which must be run hundreds of times with slightly different values or combinations of values each time; at the end of these tests the resulting values must be checked against expected values. Other tests only need to be run a few times and are designed to test some unique conditions or paths of execution, where a resulting value is not important. For these unique tests, the record/playback feature could be used to create tests

without ever needing to write a line of code. For these tests, an error during playback will indicate a failure of the test.

The record/playback and instance monitoring tools are both useful, but if these features could be combined and some small modifications included, they could potentially radically decrease the time required to test a system. In order for these features to be useful when used in concert, a logging feature would need to be included and the simulator would need a "log instances" control. With these modifications, it would be possible for a tester to begin recording a test, then walk through the path of the test and log the instance values when needed. This single test could then be modified by a relatively simple script which could go through the recording and change the test values. This would allow creation of tests with any number of combinations of values, without writing a single line of code. This approach could be useful for many different types of testing, with the possible exception of tests that attempt to provide statement, path, or decision coverage, since the time required for a tester to walk through these combinations would likely be much longer than the time required to code the tests.

4.1 Unit Testing

Unit testing on a class within an iUML model can commence as soon as the class is defined in the class diagram and the ASL for the operations within the class is written. A

test method can instantiate that class and test implemented operations without needing to create the link relationships to other classes. Consequently, the use of iUML for development should not impact the placement of unit testing within a development plan.

Due to a slightly higher degree of coupling in the iUML implementation (which could utilize neither generics nor typecasting), a smaller number of methods in the HeapContact model were appropriate for unit testing than in the original C++ implementation. This likely reduced testing time somewhat. Three classes were unit tested in the iUML implementation, while 6 classes were tested by the SQA class. However, the additional 3 classes were all nearly identical (being in the same inheritance hierarchy) and had only one method each, so the additional effort to test these classes would be minimal.

The format of iUML compliments unit testing. Much of the code which is "integration" in nature tends to be within the "states" of state machines; other code, which is more stand-alone and suited to unit testing, is in the operations, making it easier to test. Thus, if the program is logically created, unit-testers should be able to test a greater percentage of operations than they would normally (assuming that placing code within state machines eliminates the need for some operations).

Unit tests were organized with one test method per class and sorted into a “Unit Tests” test method set. Comments were inserted into the script to indicate to testers the places where they should check the instance monitor with the simulator.

It took 120 minutes to write the test scripts and code and 20 minutes to perform the tests.

4.2 Integration Testing

Integration testing on iUML must test not only code within operations, but also code located within state machines. Integration testing on groups of classes can begin as soon as the classes are implemented and connected by relationships. Testing the state machine is somewhat more complicated. Integration testing of the state machine cannot begin until the state machine diagram is complete and portions of the class diagram and class collaboration diagram are complete.

The HeapContact program is fairly small and consequently the entire program is located within a single domain. In more complex systems, sets of domains must be tested together to verify that bridge mappings are working (Raistrick et al., 2004). Consequently, larger systems may incur an additional testing cost beyond what is demonstrated in testing the HeapContact program.

A large portion of the code which was integration tested was code within the logic state machine of the model, as opposed to code within the operations of classes. In order to test the state machine code it was necessary to create an instance of logic and then initialize it to a certain state and send it a signal to begin its process for that state. From the time the signal is sent, the state will continue running asynchronously –unlike a function call, state machine execution will not return to the test function which made the call/sent the signal. This may have adverse effects for testing, since everything the test script needs to do or analyze must be accomplished before the signal is sent to the state machine. This had no effect on testing for the Heap Contact model, but may affect larger systems since after a test method (which tests state machine code) begins, execution cannot return to the test method, making it difficult to capture results and detect test completion.

In large systems, integration tests will need to be located at the bridges which connect the domains. As soon as state machines within a domain have completed, execution will return to the test method at the bridge. It will still not be possible to easily test sub-portions of domains. This can potentially delay integration testing on some portions of systems, but additionally, this places a great deal of importance on the design of the domains. It will be difficult to break a domain up and perform integration tests on portions of the domain if some of the code is located within state machines. Integration testing will be easiest if the domains reflect exactly the modules which should be integration tested.

For integration testing the HeapContact program was divided into logical modules based on dependencies of classes. Integration tests were created with one test method per module. It took 5 hours to write the test scripts and code, 30 minutes to perform the tests, and 30 minutes to document the results.

4.3 Testing the Platform Specific Model

No actual testing of the Platform Specific Model could be performed because iUMLite, the free version of iUML, does not support compiling the model into a PSM. However, work with the PIM revealed some details regarding testing of the PSM.

One interesting byproduct of iUML's ability to convert a platform independent model into a platform specific model is that the test scripts can also be converted. After the model is converted to a native language, the test scripts will also be in native language – and will still work. This ability was recognized during the development of the Independent Verification and Validation (IV&V) plan for Orion. According to Henry [2007], the Orion testing plan required testing of the system with the same test methods on both the PIM and the PSM.

After the test scripts are converted to native code it will no longer be possible to run the test scripts within the iUML simulator, which means that instance monitoring and other

features of the simulator will no longer be usable. At this point any scripts that rely on instance monitoring will need to be reworked with native language code in order to check values. However, despite this rework, it should still be worthwhile to utilize instance monitoring in PIM tests (rather than inline NLC) for a number of reasons.

1) Instance monitoring takes no effort to incorporate. This feature is fully incorporated in the simulator, it is not necessary for a developer to write any code to take advantage of it.
2) It would take more work to incorporate NLC before the transition to PSM than it would after the transition to PSM. The reason for this is that after the transition to PSM no translation would be necessary between ASL and NLC. Additionally, inline tags would not be necessary and include statements would be simpler after the transition to PSM (include statements for inline NLC require additional special tags).

4.4 System Testing

The System Testing of the original C++ HeapContact code was traditional black box system testing. The SQA class wrote test scripts and then ran the program and manually walked through the tests. Automating these tests would not have been possible due to the amount of user input required by the program, and the relatively poor encapsulation of the user interface code.

From a black box perspective, the iUML Heap Contact program is identical to the C++ program. Consequently, black box system testing would have yielded identical results to testing of the C++ program. Rather than take this trivial approach, iUML's potential for more unusual system testing approaches was explored.

An interesting approach to system testing iUML is to create an entire model dedicated to testing (Henry, 2007). This model could have modules for generating test values and computing expected results, as well as other modules for storing the data and outputting test results. The testing model would interface with the model that is being tested through the bridges. These bridges could be used to pass values in to the model and then check the results that come back before passing values on to the next domain. This would provide system testing with no need to make changes to the model that is being tested.

The main advantage of a testing model in iUML is the potential for reuse. This model could potentially be adapted to work on many programs. Many of the classes and domains within the testing model would require rework to adapt the model for another program, but a large portion of the rework would be just data entry (as in the test data and expected values domains shown below). The most significant rework required would be in the bridge terminators which interface the testing model with the system that is being tested.

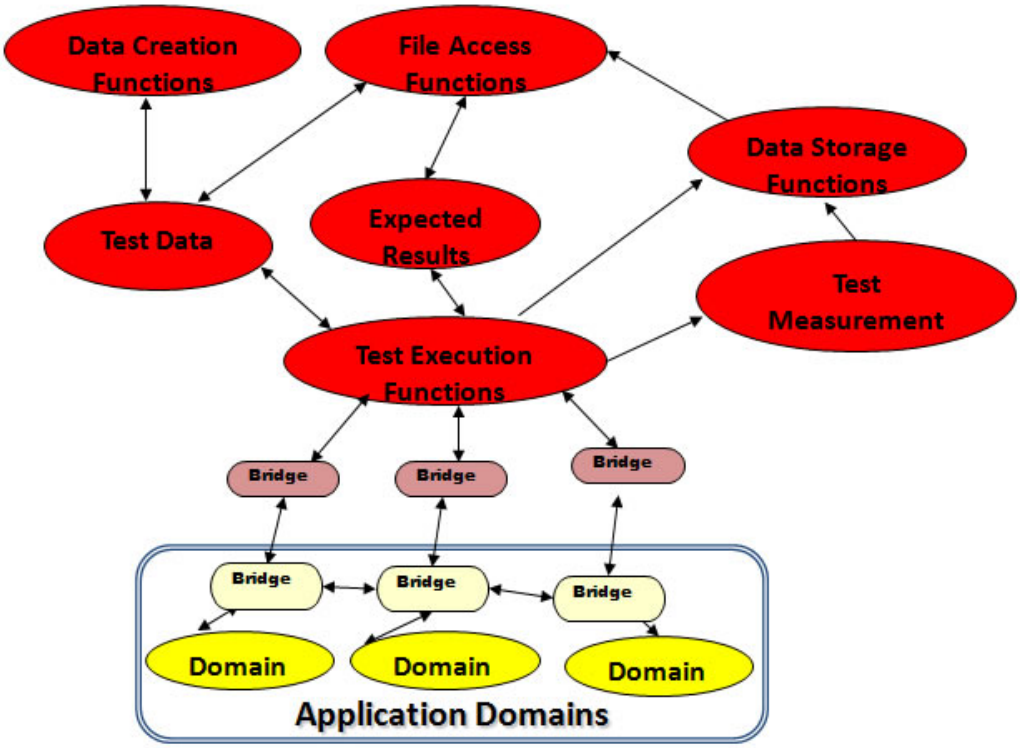

Figure 9: A potentially reusable testing model (modified from Henry [2008]).

The effort to develop a testing model such as this would be large, almost certainly much larger than development of traditional automated system testing. However, the potential for reuse of this testing model means that the model may take less time in the long run; once the initial investment is made and the model is completed, only the bridges and expected values should need to be changed.

We can even envision that it might be possible with iUML for a company to provide "off the shelf" testing models. A developer could simply pick a testing model that approximately serves their needs, purchase it, and make modifications to it, rather than needing to develop it in-house. Similar approaches are already common in industry, a

company might buy a license for a physics engine, graphics library, or other code module and then incorporate that module into their product. This approach is becoming more and more common, so obviously it is cost effective. With the use of iUML, it may be possible to bring that same "off the shelf" approach to testing.

Development of a complete testing model as shown above is beyond the scope of this thesis. Instead, just the bridges on the test model side were developed and made to interact with bridges from the original program. In order to accommodate this, the HeapContact program (which originally had no bridges) was modified. The UI was separated from the rest of the model and put in its own domain. This was a logical way to break up the program and promote testing, since it isolated most of the code requiring input from the user.

Although the use of bridges is supported in the free version of iUML, the iUML documentation does not provide complete information on how to implement bridges. Consequently, the bridges implemented here are believed to be correct, but the program could not be run successfully with bridges in place.

The association terminators shown below form the bridge. This diagram shows how the signals are translated from one side of the bridge to the other.

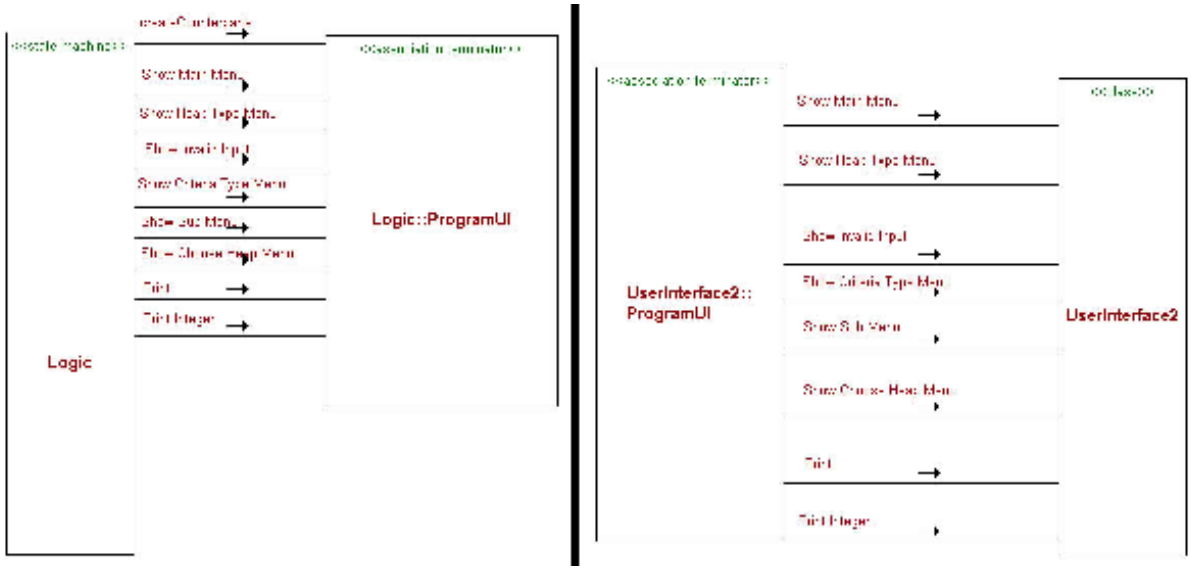

Figure 10: The modified HeapContact program with a bridge between logic and UI.

We cannot directly compare the time to create the testing bridges with the time to system test the C++ program, but this does give us an idea of the time it would take to adapt a testing model to one specific program. Other changes that would be needed in the model are changes to the values generator and checker.

It took 50 minutes to implement the bridges for system testing of the model. Again, this represents only a portion of the rework needed to adapt a testing model to a program.

The IV&V testing plan for the Orion project included plans to create a testing model like the one shown above. Henry [2007] identified a number of advantages for creating a testing model for the Orion project. The main advantages are:

1. Testing domains are created and executed with the xUML environment eliminating all differences between testing and application environments.
2. Testing domains use the same constructs and are subject to the same constraints and consistencies as the application domains.
3. Testing domains do not change the application domains in any way.
4. Testing approaches and methods can be added, extended, or modified in any way as they are independent, and insulated within domains.
5. Testing domains can be translated to Platform Specific Models and testing can be performed on the translated application domains using exactly the same translation methods on both types of domains.
6. Testing domains can be created over time as a library of testing approaches. Bridges can be reused as-is in some cases, with minor modifications in other cases, and made application specific where unique and critical application domain testing is needed.
7. Testing domains can be created in parallel with application domains, making them available as soon as possible during development.

CHAPTER 5 RESULTS

5.1 Results

Metrics for testing of the original C++ program were taken from 2 graduate students. Their results were very similar and were averaged to provide the numbers below.

Unit Testing	Average C++ Testing	iUML Testing
Creating Tests	113	120
Running Tests	72	20
Total	**185 minutes**	**140 minutes**

Integration Testing	Average C++ Testing	iUML Testing
Creating Tests	200	300
Running Tests	168	30
Total	**368 minutes**	**330 minutes**

System Testing	Average C++ Testing	iUML Testing
Creating Tests	98	50 (bridges only)
Running Tests	76	Na
Total	**174 minutes**	**Na**

5.2 Conclusions

Despite the complexities of working with iUML models, testing took less time in the iUML model than in the C++ implementation. Both unit and integration testing took moderately less time in iUML, possibly due to minor simplifications in the model, additional tools available for testing with iUML, or differences in the developers doing

the testing. What this research shows is that the extreme complications of iUML are primarily in the design phase of a project, testing in iUML need not be more complicated or time consuming than in a traditional software project.

With regard to an adaptable testing model, it took about one-half as long to create bridges for system testing in iUML as it did to create complete system tests in C++. The bridges are where the most significant rework is necessary to adapt the testing model to work on a new program (the other work is mainly data entry), so these results imply that the adaptable testing model would be cost-effective in the long term. The HeapContact program was simple enough that determining test values and expected results were not difficult. More complicated programs, which require thousands of test values and combinations of test values, should see even more benefit from the use of the reusable testing model, since it provides the structure to support these tests.

From an implementation standpoint, placement of unit testing within a development schedule should not be affected by the use of MDA, and the dependency on complete class collaboration diagrams should push back integration testing only slightly in the development plan. However, it is important to note that before the implementation phase of the project can even begin (creation of the class, class collaboration, and state machine diagrams), the design phase should first be finished (creation of the domain model, use cases, and sequence diagrams). The extremely lengthy and detailed iUML design process was skipped during development of the HeapContact model, but there's

no doubt that it would have added an enormous amount of time to the project. The time required to complete this design process, and the fact that it is a prerequisite to implementation in iUML, would significantly push back testing.

Unit and integration testing of the iUML system was accelerated by the integrated testing tools within iUML. While these tools were very helpful, some small improvements could make them tremendously helpful and allow testing time to be radically reduced. This is an example of the tremendous potential of the MDA approach, which is only partially realized in the iUML implementation.

Testing of an iUML system does bring complications of its own, and may give designers less flexibility when creating a system. In particular, it proved extremely difficult to perform integration testing on large domains with a high dependency of state machines. This makes it even more imperative that iUML models be well designed and domains not be inappropriately large.

A decision to use iUML for a large project will result in radically more time being required for design and implementation. However, testing of the system should not take longer than it would in a traditional development process, and if a reusable system testing model is already available, testing may take considerably less time in iUML. Future improvements in iUML may even result in a radical reduction in the time required for testing.

5.3 Future Directions

This thesis explored the implications of unit and integration testing in iUML and arrived at some conclusions that will hopefully be useful to teams evaluating whether or not to use iUML in the future. Some remaining unknowns with regard to testing iUML systems are the full cost of developing a reusable testing model and the effectiveness of such a model.

In order to fully evaluate such a model, it would need to be employed on a much larger system than the one tested here. Ideally it would be a critical system with several domains for which formal testing methods can be applied.

There is little doubt that development of such a testing model in iUML would take much longer than creating test methods in a traditional programming language, so the crucial question becomes "how reusable would the model be?" The completed testing model would need to be adapted to test a different system, and the amount of work required to adapt the testing model would need to be measured.

A reusable testing model, if effective, could be one of the greatest advantages of MDA, and might help offset the cost of MDA development.

BIBLIOGRAPHY

"Catalog of OMG Modeling and Metadata Specifications." Object Management Group. OMG. 28 Oct. 2008 <http://www.omg.org/technology/documents/modeling_spec_catalog.htm#ocl>

"Customer Success." Intelligent Solutions for Model Driven Architecture. Kennedy Carter. 28 Oct. 2008 <http://www.kc.com>.

Henry, Joel. "Verification and Validation in Model-Driven Development." 01 May 2008.

Henry, Joel, Michael Hieber, and Craig Schulenberg. ORION iV&V Approach, Procedures, and Results. MRI Corporation. Version 2 ed. 2007.

Hoffman, Mark. "Orion SEPG CASE Tool Change." 28 Aug. 2008.

IUML Tutorial. Kennedy Carter. Manual Revision 2 ed. 2003.

IUML Modeller User Guide. Kennedy Carter. Revision 2 ed. 2003.

IUML Simulator User Guide. Kennedy Carter. Manual Revision 1 ed. 2003.

Lowry, Michael. "Intelligent Software Engineering Tools for NASA's Crew Exploration Vehicle." Lecture Notes in Computer Science. NASA Ames Research Center, Moffett Field, CA. SpringerLink. 10 May 2008. 28 Oct. 2008 < http://www.springerlink.com/content/91j236610552kgmv/ />.

Luz, Miguel P., and Alberto R. Silva. Executing UML Models. Tech.No. Instituto Superior Técnico. Not Dated.

Mellor, Stephen J., and Steve Tockey. Software-Platform-Independant Precise Action Specifications for UML. Tech.No. Project Technology Inc, Rockwell Collins Inc. Not Dated.

Raistrick, Chris, Paul Francis, and John Wright. Model Driven Architecture with Executable UML. New York: Cambridge UP, 2004.

Wilkie, Ian, Adrian King, Mike Clarke, Chas Weaver, Chris Raistrick, and Paul Francis. UML ASL Reference Guide. Kennedy Carter. Manual Revision D. Kennedy Carter, 2003.

APPENDIX

Appendix A – Additional Bugs found in iUML

- Floating point values cannot be moved from within NLC to the ASL, this should be supported but causes the program to crash.
- It is impossible for the user to change the numbers assigned to the states in the state machine.
- Program sometimes inexplicably does not run, or will not allow user to step through while checking values. This problem is erratic –it works sometimes and not others, with no changes in-between.
- Transferring any values from NLC to ASL can cause issues, sometimes there will not be issues in immediate ASL operation, but there will be problems when the values are passed or returned to other ASL operations. The most common problem in these cases is that the value passed will be null or zero. This implies a problem with addressing values in ASL/NLC.

Appendix B - Restrictions on iUMLite

(From "iUML – Read Me First!" document):

- 5 Domains per database
- 3 Versions per Domain
- 15 Classes per Domain version
- 40 Classes in total per database
- 15 States per state model
- 10 Attributes per class
- 8 Operations per class
- 2 Projects per database
- 3 Versions per project
- 10 Use Case diagrams per database
- 5 Use Cases per database
- 5 Actors per database
- 10 Sequence Diagrams per database
- 100 Interactions per database

Appendix C – Test Cases

Unit Test Cases

Class Name	Function Name	Test Type	Result	Severity	Proposed Solution
Employee	Write	Call function, Check result	Pass	NA	NA
	ReadFromFile	Input values / Check result	Pass	NA	NA
	Read	Input values / Check result	Pass	NA	NA
	Delete	Call function, Check result	Pass	NA	NA
Customer	Write	Call function, Check result	Pass	NA	NA
	ReadFromFile	Input values / Check result	Pass	NA	NA
	Read	Input values / Check result	Pass	NA	NA
	Delete	Call function, Check result	Pass	NA	NA
Shipper	Write	Call function, Check result	Pass	NA	NA
	ReadFromFile	Input values / Check result	Pass	NA	NA
	Read	Input values / Check result	Pass	NA	NA
	Delete	Call function, Check result	Pass	NA	NA

Integration Test Cases

Module	Class	Function	Test	Result	Reason
Criteria/Contact	NameMapper	Map	Send in Contact object, check return value	Pass	
	AddressMapper	Map	Send in Contact object, check return value	Pass	
	PhoneMapper	Map	Send2 in Contact object, check return value	Pass	
Criteria/Heap	Heap	PrintHeapType	Call function, verify that heap type and criteria type are both printed.	Pass	
Heap/Contact	Heap	Find	Call function, input contact name, check that contact info is written out.	Pass	
	Heap	Delete	Call function, input contact name, verify that contact instance is deleted.	Pass	
	Heap	Empty	Call function, verify that all contact instances for that heap are deleted and all their child classes are deleted.	Pass	
	Heap	ReadFromFile	Call function, verify that contact instances and contact child instances are created for that heap	Pass	
Heap/Contact/Criteria	Heap	Add	Call function, input contact info, check that contact is created and within heap	Pass	
	Heap	InsertIntoHeap	Call add function, input contact info. Check resulting ArrayPosition property value to verify that it is correct.	Pass	

	Heap	PrintAll	Call function, check output against listing of instances of contacts for that heap. Verify that contacts are printed in the same order as their arrayPosition values.	Pass	
Logic/Heap/UI	Logic	State Machine	Set state to Create New Heap. Input null, alpha, too large a number, too small a number, then valid number. Verify that heap was created. Verify that state advanced.	Fail	Program assumes an integer will be input. It can handle an illegal integer, but not nulls and chars.
Logic/Criteria/UI	Logic	State Machine	Set state to User Entered Valid Heap. Input null, alpha, too large a number, too small a number, then valid number. Verify that criteria instance was created. Verify that state advanced.	Fail	Program assumes an integer will be input. It can handle an illegal integer, but not nulls and chars.
Logic/UI	Logic	State Machine	Set state to Start Program. Input null, alpha, too large a number, too small a number, then valid number. Verify that state advanced.	Fail	Program assumes an integer will be input. It can handle an illegal integer, but not nulls and chars.

Appendix D – C++HeapContact Code

Logic Class

```
//-------------------------Logic.h-------------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//---------------------------------------------------------------
// Purpose - This file contains variable definitions and
//                                      function declarations for Logic.cpp.
//---------------------------------------------------------------

// If not defined, define
#ifndef INC_LOGIC_H
#define INC_LOGIC_H

// Include files
#include "ContactType.h"
#include "Heap.h"

// Constants
const static int MAX_HEAPS = 10;

class Logic
{       // Begin Logic

        private:

        // Member data
                Heap <ContactType *, char *> *ArrayOfHeaps[MAX_HEAPS];
                int NumberOfHeaps;

        // Private member functions
                int GetHeapType();
```

```
                        void CreateHeap(int HeapType);
                        int ShowHeaps();
                        int FindItemInHeap(int HeapIndex);

        public:

        // Default constructor
                        Logic();

        // Member functions
                        void PerformMainMenuOption(int MenuOption);
                        void PerformSubMenuOption(int MenuOption, int HeapIndex);

};              // End Logic

#endif;
```

```
//-------------------------Logic.cpp----------------------------
// James Fishbaugh  CS 441-01
// 10/11/2004
// Last Modified:  10/13/04
//---------------------------------------------------------------
// Purpose - Implements Logic class which defines all the
//          logic and control for the other classes
//---------------------------------------------------------------

// Include files
#include <stdlib.h>     // For exit
#include "Logic.h"
#include "UI.h"
#include "IO.h"
#include "Heap.h"

#include "CriteriaMapper.h"
#include "AddressMapper.h"
#include "NameMapper.h"
#include "PhoneMapper.h"

#include "ContactType.h"
#include "Employee.h"
#include "Customer.h"
#include "Shipper.h"

// Creates a global UI object
UI *ScreenOutput = new UI();
// Creates a global IO object
IO *FileIO = new IO();

//---------------------------Logic-------------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//---------------------------------------------------------------
// Calls to:  n/a
// Called by: main
//---------------------------------------------------------------
// Arguments:  void
//---------------------------------------------------------------
// Description:  Default constructor for Logic
```

```
//---------------------------------------------------------------
Logic::Logic()
{		// Begin Logic

		NumberOfHeaps = 0;

}		// End Logic

//-------------------PerformMainMenuOption---------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/13/05
//---------------------------------------------------------------
// Calls to:  Logic->GetHeapType()
//                                      Logic->CreateHeap()
//                                      Logic->ShowHeaps()
//                                      Logic->PerformMainMenuOption()
//                                      Logic->PerformSubMenuOption()
//                                      UI->PrintError()
//                                      UI->ShowSubMenu()
//                                      UI->ShowMainMenu()
//                                      UI->GetMenuChoice()
// Called by: Logic->PerformMainMenuOption()
//                                      Logic->PerformSubMenuOption()
//                                      Logic->CreateHeap()
//                                      main()
//---------------------------------------------------------------
// Arguments:  MenuOption - the menu option the user selected
//---------------------------------------------------------------
// Description:  This function calls the appropriate function
//            to perform the desired main menu option
//---------------------------------------------------------------
void Logic::PerformMainMenuOption(int MenuOption)
{		// Begin PerformMainMenuOption

		// The type of heap
		int HeapType = 0;
		// The users new menu choice
		int MenuChoice = 0;
		// What heap we are working with
		int HeapIndex = 0;

		switch (MenuOption)
		{		// Begin switch

				// User wants to create a new heap
				case 1:

						// If the user has already the maximum allowed heaps
						if (NumberOfHeaps == MAX_HEAPS)
						{		// Begin if

								ScreenOutput->PrintError(4);

						}		// End if
						else
						{		// Begin else

								HeapType = GetHeapType();
								CreateHeap(HeapType);

						}		// End else

						break;

				// User wants to work with an existing heap
```

```
                case 2:

                        // Finds which heap they want to work with
                        HeapIndex = ShowHeaps();

                        // If HeapIndex = -1 there are no heaps created
                        if (HeapIndex != -1)
                        {       // Begin if

                                ScreenOutput->ShowSubMenu();
                                MenuChoice = ScreenOutput->GetMenuChoice();
                                PerformSubMenuOption(MenuChoice, HeapIndex);

                        }       // End if
                        // Else there is at least one heap created
                        else
                        {       // Begin else

                                ScreenOutput->ShowMainMenu();
                                MenuChoice = ScreenOutput->GetMenuChoice();
                                PerformMainMenuOption(MenuChoice);

                        }       // End else

                        break;

                // User wants to quit
                case 3:

                        // Exits the program
                        exit(0);

                        break;

                // User entered an invalid number
                default:

                        // Show the error message
                        ScreenOutput->PrintError(1);
                        // Show the main menu again
                        ScreenOutput->ShowMainMenu();
                        // Get the users choice
                        MenuChoice = ScreenOutput->GetMenuChoice();
                        // Call itself to perform the menu option
                        PerformMainMenuOption(MenuChoice);

                        break;

        }       // End switch

}       // End PerformMainMenuOption

//-------------------PerformSubMenuOption----------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/12/05
//-----------------------------------------------------------
// Calls to:  Logic->GetHeapType()
//                      Logic->CreateHeap()
//                      Logic->ShowHeaps()
//                      Logic->PerformMainMenuOption()
//                      Logic->PerformSubMenuOption()
//                      Logic->FindItemInHeap()
//                      UI->Print()
//                      UI->PrintInteger()
//                      UI->PrintError()
```

```
//                                      UI->ShowSubMenu()
//                                      UI->ShowMainMenu()
//                                      UI->GetMenuChoice()
//                                      Employee->Employee()
//                                      Customer->Customer()
//                                      Shipper->Shipper()
//                                      ContactType->Read()
//                                      ContactType->GetName()
//                                      ContactType->GetAddress()
//                                      ContactType->GetPhone()
//                                      Heap->Add()
//                                      Heap->Empty()
//                              Heap->DeleteAtIndex()
//                                      Heap->Print()
//                                      Heap->PrintAll()
//                                      IO->ReadFromFile()
// Called by: Logic->PerformSubMenuOption()
//                                      Logic->PerformMainMenuOption()
//---------------------------------------------------------------
// Arguments:  MenuOption - the menu option the user selected
//---------------------------------------------------------------
// Description:  This function calls the appropriate function
//           to perform the desired sub menu option
//---------------------------------------------------------------
void Logic::PerformSubMenuOption(int MenuOption, int HeapIndex)
{               // Begin PerformSubMenuOption

                int MenuChoice = 0;
                int HeapType = 0;
                int FoundIndex = 0;
                int i;
                char ** Data;
                int NumberOfContacts = 0;

                switch (MenuOption)
                {               // Begin switch

                                // User wants to Add
                                case 1:

                                                HeapType = ArrayOfHeaps[HeapIndex]->GetHeapType();

                                                // Add an Employee with Name criteria
                                                if (HeapType == 1)
                                                {               // Begin if

                                                                Employee * NewEmployee = new Employee;

                                                                NewEmployee->Read();

                                                                ArrayOfHeaps[HeapIndex]->
                                                                                Add((ContactType *) NewEmployee, NewEmployee->GetName());
                                                }               // End if
                                                // Add an Employee with Address criteria
                                                else if (HeapType == 2)
                                                {               // Begin else if

                                                                Employee * NewEmployee = new Employee;

                                                                NewEmployee->Read();

                                                                ArrayOfHeaps[HeapIndex]->
                                                                                                Add((ContactType *) NewEmployee, NewEmployee-
>GetAddress());

                                                }               // End else if
```

```
                    // Add an Employee with Phone criteria
                    else if (HeapType == 3)
                    {           // Begin else if

                                Employee * NewEmployee = new Employee;

                                NewEmployee->Read();

                                ArrayOfHeaps[HeapIndex]->
                                                    Add((ContactType *) NewEmployee, NewEmployee-
>GetPhone());
                    }           // End else if
                    // Add a Customer with Name criteria
                    else if (HeapType == 4)
                    {           // Begin else if

                                Customer * NewCustomer = new Customer;

                                NewCustomer->Read();

                                ArrayOfHeaps[HeapIndex]->
                                                    Add((ContactType *) NewCustomer, NewCustomer-
>GetName());

                    }           // End else if
                    // Add a Customer with Address criteria
                    else if (HeapType == 5)
                    {           // Begin else if

                                Customer * NewCustomer = new Customer;

                                NewCustomer->Read();

                                ArrayOfHeaps[HeapIndex]->
                                                    Add((ContactType *) NewCustomer, NewCustomer-
>GetAddress());

                    }           // End else if
                    // Add a Customer with Phone criteria
                    else if (HeapType == 6)
                    {           // Begin else if

                                Customer * NewCustomer = new Customer;

                                NewCustomer->Read();

                                ArrayOfHeaps[HeapIndex]->
                                                    Add((ContactType *) NewCustomer, NewCustomer-
>GetPhone());

                    }           // End else if
                    // Add a Shipper with Name criteria
                    else if (HeapType == 7)
                    {           // Begin else if

                                Shipper * NewShipper = new Shipper;

                                NewShipper->Read();

                                ArrayOfHeaps[HeapIndex]->
                                                    Add((ContactType *) NewShipper, NewShipper-
>GetName());

                    }           // End else if
                    // Add a Shipper with Address criteria
                    else if (HeapType == 8)
```

```
                    {       // Begin else if

                            Shipper * NewShipper = new Shipper;

                            NewShipper->Read();

                            ArrayOfHeaps[HeapIndex]->
                                                    Add((ContactType *) NewShipper, NewShipper-
>GetAddress());

                    }       // End else if
                    // Add a Shipper with Phone criteria
                    else
                    {       // Begin else

                            Shipper * NewShipper = new Shipper;

                            NewShipper->Read();

                            ArrayOfHeaps[HeapIndex]->
                                                    Add((ContactType *) NewShipper, NewShipper-
>GetPhone());

                    }       // End else

                    ScreenOutput->ShowSubMenu();
                    MenuChoice = ScreenOutput->GetMenuChoice();
                    PerformSubMenuOption(MenuChoice, HeapIndex);

                    break;

            // User wants to Find
            case 2:

                    FoundIndex = FindItemInHeap(HeapIndex);

                    if (FoundIndex == -1)
                    {       // Begin if

                            ScreenOutput->PrintError(3);

                    }       // End if
                    else
                    {       // Begin else

                            ArrayOfHeaps[HeapIndex]->Print(FoundIndex);

                    }       // End else

                    ScreenOutput->ShowSubMenu();
                    MenuChoice = ScreenOutput->GetMenuChoice();
                    PerformSubMenuOption(MenuChoice, HeapIndex);

                    break;

            // User wants to Print All
            case 3:

                    // Print all the information in heap at HeapIndex
                    ArrayOfHeaps[HeapIndex]->PrintAll();

                    ScreenOutput->ShowSubMenu();
                    MenuChoice = ScreenOutput->GetMenuChoice();
                    PerformSubMenuOption(MenuChoice, HeapIndex);

                    break;
```

```
// User wants to Delete
case 4:

		FoundIndex = FindItemInHeap(HeapIndex);

		if (FoundIndex == -1)
		{		// Begin if

				ScreenOutput->PrintError(3);

		}		// End if
		else
		{		// Begin else

				ArrayOfHeaps[HeapIndex]->DeleteAtIndex(FoundIndex);

		}		// End else

		ScreenOutput->ShowSubMenu();
		MenuChoice = ScreenOutput->GetMenuChoice();
		PerformSubMenuOption(MenuChoice, HeapIndex);

		break;

// User wants to Empty
case 5:

		// Empty all the information in heap at HeapIndex
		ArrayOfHeaps[HeapIndex]->Empty();

		ScreenOutput->ShowSubMenu();
		MenuChoice = ScreenOutput->GetMenuChoice();
		PerformSubMenuOption(MenuChoice, HeapIndex);

		break;

// User wants to Read
case 6:

		Data = FileIO->ReadFromFile(NumberOfContacts);

		HeapType = ArrayOfHeaps[HeapIndex]->GetHeapType();

		// Calls appropriate functions in UI class to
		// print to the screen
		ScreenOutput->Print("\n");
		ScreenOutput->PrintInteger(NumberOfContacts);
		ScreenOutput->Print(" new elements added to the heap\n");

		// Loop adds all the new contacts the the correct heap
		for (i = 0; i < NumberOfContacts; i++)
		{		// Begin for

				// Add an Employee with Name criteria
				if (HeapType == 1)
				{		// Begin if

						Employee * NewEmployee = new Employee;

						NewEmployee->Read(Data[i]);

						ArrayOfHeaps[HeapIndex]->
							Add((ContactType *) NewEmployee, NewEmployee->GetName());

				}		// End if
```

```
                    // Add an Employee with Address criteria
                    else if (HeapType == 2)
                    {           // Begin else if

                                Employee * NewEmployee = new Employee;

                                NewEmployee->Read(Data[i]);

                                ArrayOfHeaps[HeapIndex]->
                                                    Add((ContactType *) NewEmployee,
NewEmployee->GetAddress());

                    }           // End else if
                    // Add an Employee with Phone criteria
                    else if (HeapType == 3)
                    {           // Begin else if

                                Employee * NewEmployee = new Employee;

                                NewEmployee->Read(Data[i]);

                                ArrayOfHeaps[HeapIndex]->
                                                    Add((ContactType *) NewEmployee,
NewEmployee->GetPhone());
                    }           // End else if
                    // Add a Customer with Name criteria
                    else if (HeapType == 4)
                    {           // Begin else if

                                Customer * NewCustomer = new Customer;

                                NewCustomer->Read(Data[i]);

                                ArrayOfHeaps[HeapIndex]->
                                                    Add((ContactType *) NewCustomer,
NewCustomer->GetName());

                    }           // End else if
                    // Add a Customer with Address criteria
                    else if (HeapType == 5)
                    {           // Begin else if

                                Customer * NewCustomer = new Customer;

                                NewCustomer->Read(Data[i]);

                                ArrayOfHeaps[HeapIndex]->
                                                    Add((ContactType *) NewCustomer,
NewCustomer->GetAddress());

                    }           // End else if
                    // Add a Customer with Phone criteria
                    else if (HeapType == 6)
                    {           // Begin else if

                                Customer * NewCustomer = new Customer;

                                NewCustomer->Read(Data[i]);

                                ArrayOfHeaps[HeapIndex]->
                                                    Add((ContactType *) NewCustomer,
NewCustomer->GetPhone());

                    }           // End else if
                    // Add a Shipper with Name criteria
                    else if (HeapType == 7)
```

```
                                        {       // Begin else if

                                                Shipper * NewShipper = new Shipper;

                                                NewShipper->Read(Data[i]);

                                                ArrayOfHeaps[HeapIndex]->
                                                                        Add((ContactType *) NewShipper,
NewShipper->GetName());

                                        }       // End else if
                                        // Add a Shipper with Address criteria
                                        else if (HeapType == 8)
                                        {       // Begin else if

                                                Shipper * NewShipper = new Shipper;

                                                NewShipper->Read(Data[i]);

                                                ArrayOfHeaps[HeapIndex]->
                                                                        Add((ContactType *) NewShipper,
NewShipper->GetAddress());

                                        }       // End else if
                                        // Add a Shipper with Phone criteria
                                        else
                                        {       // Begin else

                                                Shipper * NewShipper = new Shipper;

                                                NewShipper->Read(Data[i]);

                                                ArrayOfHeaps[HeapIndex]->
                                                                        Add((ContactType *) NewShipper,
NewShipper->GetPhone());

                                        }       // End else

                                }

                                ScreenOutput->ShowSubMenu();
                                MenuChoice = ScreenOutput->GetMenuChoice();
                                PerformSubMenuOption(MenuChoice, HeapIndex);

                                break;

                        // User wants to Delete the entire heap
                        case 7:

                                // Shift everything to the left
                                for (i=HeapIndex; i<NumberOfHeaps-1; i++)
                                {       // Begin for

                                        ArrayOfHeaps[i] = ArrayOfHeaps[i+1];

                                }       // End for

                                // Decrement the number of heaps
                                NumberOfHeaps--;

                                // Return to the main menu
                                ScreenOutput->ShowMainMenu();
                                MenuChoice = ScreenOutput->GetMenuChoice();
                                PerformMainMenuOption(MenuChoice);

                                break;
```

```
                        // User wants to return to main menu
                        case 8:

                                // Return to the main menu
                                ScreenOutput->ShowMainMenu();
                                MenuChoice = ScreenOutput->GetMenuChoice();
                                PerformMainMenuOption(MenuChoice);

                                break;

                        // User wants to Quit
                        case 9:

                                exit(0);

                                break;

                        // User entered an invalid number
                        default:

                                // Show the error message
                                ScreenOutput->PrintError(1);
                                // Show the sub menu again
                                ScreenOutput->ShowSubMenu();
                                // Get the users choice
                                MenuChoice = ScreenOutput->GetMenuChoice();
                                // Call itself to perform the menu option
                                PerformSubMenuOption(MenuChoice, HeapIndex);
                                break;

        }       // End switch

}       // End PerformSubMenuOption

//------------------------GetHeapType--------------------------
// James Fishbaugh  CS 441-01
// 10/12/2005
// Last Modified:  10/12/05
//---------------------------------------------------------------
// Calls to:  UI->HeapTypeOptions()
//                                      UI->GetMenuChoice()
//                                      UI->PrintError()
// Called by: Logic->PerformMainMenuOption()
//                                      Logic->PerformSubMenuOption()
//---------------------------------------------------------------
// Arguments:  void
//---------------------------------------------------------------
// Description:  Finds the type of heap the user wants
//              to create and returns that value
//---------------------------------------------------------------
int Logic::GetHeapType()
{       // Begin GetHeapType

        int HeapType = 0;
        int MenuChoice = 0;

        ScreenOutput->HeapTypeOptions();
        MenuChoice = ScreenOutput->GetMenuChoice();

        do
        {       // Begin do

                if (MenuChoice == 1)
                {       // Begin if
```

```
                        HeapType = 1;

                }       // End if
                else if (MenuChoice == 2)
                {       // Begin else if

                        HeapType = 2;

                }       // End else if
                else if (MenuChoice == 3)
                {       // Begin else if

                        HeapType = 3;

                }       // End else if
                else
                {       // Begin else

                        ScreenOutput->PrintError(1);
                        ScreenOutput->HeapTypeOptions();
                        MenuChoice = ScreenOutput->GetMenuChoice();

                }       // End else

        }while ((MenuChoice < 1) || (MenuChoice > 3));          // End do while

        return HeapType;

}       // End GetHeapType

//-------------------------CreateHeap--------------------------
// James Fishbaugh  CS 441-01
// 10/12/2005
// Last Modified:  10/12/05
//---------------------------------------------------------------
// Calls to:  UI->HeapCriteriaOptions()
//                                      UI->GetMenuChoice()
//                                      UI->PrintError()
//                                      Heap->Heap()
//                                      Employee->Employee()
//                                      Customer->Customer()
//                              Shipper->Shipper()
//                                      Logic->PerformMainMenuOption()
// Called by: Logic->PerformMainMenuOption()
//---------------------------------------------------------------
// Arguments:  HeapType - Integer which denotes what type
//                  of heap to create.  1 denotes
//                          Employee, 2 denotes Customer, and
//                  3 denotes Shipper.
//---------------------------------------------------------------
// Description:  Finds the criteria type and then creates
//           the heap
//---------------------------------------------------------------
void Logic::CreateHeap(int HeapType)
{       // Begin CreateHeap

        // Integers to hold the menu choice and criteria type
        int MenuChoice = 0;
        int CriteriaChoice = 0;
        // A pointer to a criteria mapper object
        CriteriaMapper * MyMapper;

        // Print out the choices
        ScreenOutput->HeapCriteriaOptions();
        // Get the users selection
        MenuChoice = ScreenOutput->GetMenuChoice();
```

```
// Finds which criteria the user wants
do
{		// Begin do

		// User wants Name criteria
		if (MenuChoice == 1)
		{		// Begin if

				CriteriaChoice = 1;

		}		// End if
		// User wants Address criteria
		else if (MenuChoice == 2)
		{		// Begin else if

				CriteriaChoice = 2;

		}		// End else if
		// User wants Phone criteria
		else if (MenuChoice == 3)
		{		// Begin else if

				CriteriaChoice = 3;

		}		// End else if
		// User entered an invalid criteria
		else
		{		// Begin else

				ScreenOutput->PrintError(1);
				ScreenOutput->HeapCriteriaOptions();
				MenuChoice = ScreenOutput->GetMenuChoice();

		}		// End else

}while ((MenuChoice < 1) || (MenuChoice > 3));		// End do

// Create the employee heap
if (HeapType == 1)
{		// Begin if

		Heap <ContactType *, char *> *EmployeeHeap;

		Employee * NewEmployee = new Employee;

		// The criteria is Name
		if (CriteriaChoice == 1)
		{		// Begin if

				MyMapper = (CriteriaMapper *) new NameMapper;
				EmployeeHeap = new Heap <ContactType *, char *> (MyMapper, 1);

		}		// End if
		// The criteria is Address
		else if (CriteriaChoice == 2)
		{		// Begin else if

				MyMapper = (CriteriaMapper *) new AddressMapper;
				EmployeeHeap = new Heap <ContactType *, char *> (MyMapper, 2);

		}		// End else if
		// Then criteria is Phone
		else
		{		// Begin else
```

```
                    MyMapper = (CriteriaMapper *) new PhoneMapper;
                    EmployeeHeap = new Heap <ContactType *, char *> (MyMapper, 3);

          }         // End else

          ArrayOfHeaps[NumberOfHeaps] = EmployeeHeap;

}         // End if
// Create the customer heap
else if (HeapType == 2)
{         // Begin else if

          Heap <ContactType *, char *> *CustomerHeap;

          Customer * NewCustomer = new Customer;

          // The criteria is Name
          if (CriteriaChoice == 1)
          {         // Begin if

                    MyMapper = (CriteriaMapper *) new NameMapper;
                    CustomerHeap = new Heap <ContactType *, char *> (MyMapper, 4);

          }         // End if
          // The criteria is Address
          else if (CriteriaChoice == 2)
          {         // Begin else if

                    MyMapper = (CriteriaMapper *) new AddressMapper;
                    CustomerHeap = new Heap <ContactType *, char *> (MyMapper, 5);

          }         // End else if
          // Then criteria is Phone
          else
          {         // Begin else

                    MyMapper = (CriteriaMapper *) new PhoneMapper;
                    CustomerHeap = new Heap <ContactType *, char *> (MyMapper, 6);

          }         // End else

          ArrayOfHeaps[NumberOfHeaps] = CustomerHeap;

}         // End else if
// Create the shipper heap
else
{         // Begin else

          Heap <ContactType *, char *> *ShipperHeap;

          Shipper * NewShipper = new Shipper;

          // The criteria is Name
          if (CriteriaChoice == 1)
          {         // Begin if

                    MyMapper = (CriteriaMapper *) new NameMapper;
                    ShipperHeap = new Heap <ContactType *, char *> (MyMapper, 7);

          }         // End if
          // The criteria is Address
          else if (CriteriaChoice == 2)
          {         // Begin else if

                    MyMapper = (CriteriaMapper *) new AddressMapper;
                    ShipperHeap = new Heap <ContactType *, char *> (MyMapper, 8);
```

```
                        }               // End else if
                        // Then criteria is Phone
                        else
                        {               // Begin else

                                        MyMapper = (CriteriaMapper *) new PhoneMapper;
                                        ShipperHeap = new Heap <ContactType *, char *> (MyMapper, 9);

                        }               // End else

                        ArrayOfHeaps[NumberOfHeaps] = ShipperHeap;

                }               // End else

                // Increment the number of heaps
                NumberOfHeaps++;

                // Return to the main menu
                ScreenOutput->ShowMainMenu();
                MenuChoice = ScreenOutput->GetMenuChoice();
                PerformMainMenuOption(MenuChoice);

}               // End CreateHeap

//-------------------------ShowHeaps----------------------------
// James Fishbaugh  CS 441-01
// 10/12/2005
// Last Modified:  10/12/05
//---------------------------------------------------------------
// Calls to:  UI->PrintError()
//                                      UI->Print()
//                                      UI->PrintInteger()
//                                      UI->GetMenuChoice()
// Called by: Logic->PerformMainMenuOption()
//                                      Logic->PerformSubMenuOption()
//---------------------------------------------------------------
// Arguments: void
//---------------------------------------------------------------
// Description:  Prints out the heaps that currently exist
//              and returns the index of the heap array.
//---------------------------------------------------------------
int Logic::ShowHeaps()
{               // Begin ShowHeaps

                int HeapType = 0;

                // If there are no heaps
                if (NumberOfHeaps == 0)
                {               // Begin if

                                ScreenOutput->PrintError(2);
                                return -1;

                }               // End if

                ScreenOutput->Print("\nChoose one of the following heaps\n");
                ScreenOutput->Print("--------------------------------\n");

                for (int i=0; i < NumberOfHeaps; i++)
                {               // Begin for

                                // Necessary output for logic class
                                ScreenOutput->PrintInteger(i+1);
                                ScreenOutput->Print(") ");
```

```
                ArrayOfHeaps[i]->PrintHeapType();

        }       // End for

        do
        {       // Begin do

                ScreenOutput->Print("\nEnter your choice: ");
                HeapType = ScreenOutput->GetMenuChoice();

                if ((HeapType < 1) || (HeapType > NumberOfHeaps))
                {       // Begin if

                        ScreenOutput->PrintError(1);

                }       // End if

        } while ((HeapType < 1) || (HeapType > NumberOfHeaps));         // End do

        // Returns the index of the heap the user wants to work with
        return (HeapType-1);

}       // End ShowHeaps

//----------------------FindItemInHeap-------------------------
// James Fishbaugh  CS 441-01
// 10/12/2005
// Last Modified:  10/12/05
//--------------------------------------------------------------
// Calls to:  Heap->GetHeapType()
//                                      Heap->Find()
//                                      UI->GetUnique()
// Called by: Logic->PerformSubMenuOption()
//--------------------------------------------------------------
// Arguments:   HeapIndex - the index of the heap array
//--------------------------------------------------------------
// Description:  Searches the heap for given data
//--------------------------------------------------------------
int Logic::FindItemInHeap(int HeapIndex)
{       // Begin FindItemInHeap

        int HeapType = 0;
        int IndexFound = -1;

        char * SearchCriteria = new char[20];

        HeapType = ArrayOfHeaps[HeapIndex]->GetHeapType();

        SearchCriteria = ScreenOutput->GetUnique(HeapType);

        IndexFound = ArrayOfHeaps[HeapIndex]->Find(SearchCriteria);

        return IndexFound;

}       // End FindItemInHeap
```

Heap Class

```
//---------------------------Heap.h----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/13/05
```

```
//---------------------------------------------------------------
// Purpose - This file contains variable definitions and
//                                      function declarations for Heap.cpp
//        Implementation provided below.  Defines a heap
//        data structure and the methods that provide
//        heap functionality.
//---------------------------------------------------------------

// Compiler directive
#pragma once

// Include files
#include <iostream>
#include <string.h>    // For strcmp
#include "CriteriaMapper.h"
using namespace std;

template <class DataType, class CriteriaType> class Heap
{            // Begin Heap

            private:

            // Member Data

                        // Small class for data storage
                        class ElementType
                        {            // Begin ElementType

                                    public:

                                                DataType ElementData;
                                                long int Criteria;

                        };           // End ElementType

                        long int HeapLength;
                        int Type;
                        ElementType HeapArray[100];
                        CriteriaMapper *HeapMapper;

            // Private member functions
                        void Heapify(long int Index);
                        long int Parent(long int Child);
                        long int Left(long int Root);
                        long int Right(long int Root);

            public:

            // Default constructor
                        Heap();

            // Init constructor
                        Heap(CriteriaMapper *MyCriteriaMapper, int HeapType);

            // Member functions
                        void Add(DataType Data, CriteriaType Criteria);
                        void Print(int IndexToPrint);
                        void PrintAll();
                        long int Find(CriteriaType Criteria);
                        void Delete(DataType Data);
                        void DeleteAtIndex(long int IndexToDelete);
                        void Empty();
                        void PrintHeapType();
                        int GetHeapType();

};           // End Heap
```

```
// This is the implementation of Heap.h

//---------------------------Heap-----------------------------
// James Fishbaugh  CS 441-01
// 10/12/2005
// Last Modified:  10/12/05
//------------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->CreateHeap()
//------------------------------------------------------------
// Arguments: void
//------------------------------------------------------------
// Description: The default constructor
//------------------------------------------------------------
template <class DataType, class CriteriaType>
Heap<DataType, CriteriaType>::Heap()
{               // Begin Heap

                HeapLength = 0;
                Type = 0;

}               // End Heap

//---------------------------Heap-----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/12/05
//------------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->CreateHeap()
//------------------------------------------------------------
// Arguments: *MyCriteriaMapper - A pointer to a
//                        CriteriaMapper obect
//------------------------------------------------------------
// Description: The initializing constructor
//------------------------------------------------------------
template <class DataType, class CriteriaType>
Heap<DataType, CriteriaType>::Heap(CriteriaMapper *MyCriteriaMapper, int HeapType)
{               // Begin Heap()

                HeapMapper = MyCriteriaMapper;
                HeapLength = 0;
                Type = HeapType;

}               // End Heap()

//---------------------------Heapify--------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//------------------------------------------------------------
// Calls to:  Heapify()
// Called by: Heap->Heapify()
//          Heap->Add()
//          Heap->Delete()
//------------------------------------------------------------
// Arguments:  Index - The index of the root being looked at
//------------------------------------------------------------
// Description:  Heapify takes a partial ordered heap and
//             rearranges elements to make it a heap
//------------------------------------------------------------
template <class DataType, class CriteriaType>
void Heap<DataType, CriteriaType>::Heapify(long int Index)
{               // Begin Heapify
```

```
		long int Child = Left(Index);

		if ((HeapArray[Child].Criteria < HeapArray[Child+1].Criteria) &&
								(Child < (HeapLength-1)))
		{		// Begin if

			Child++;

		}		// End if

		if (HeapArray[Index].Criteria >= HeapArray[Child].Criteria)
		{		// Begin if

			return;

		}		// End if

		// Make the swap of data and criteria
		DataType tempData = HeapArray[Index].ElementData;
		long int tempCriteria = HeapArray[Index].Criteria;
		HeapArray[Index].ElementData = HeapArray[Child].ElementData;
		HeapArray[Index].Criteria = HeapArray[Child].Criteria;
		HeapArray[Child].ElementData = tempData;
		HeapArray[Child].Criteria = tempCriteria;

		// Make a recursive call
		Heapify(Child);

}		// End Heapify

//-------------------------Parent-----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//----------------------------------------------------------------
// Calls to:  n/a
// Called by: Heap->Add()
//----------------------------------------------------------------
// Arguments: Child - the index of the child
//----------------------------------------------------------------
// Description:  Returns the index of the parent
//----------------------------------------------------------------
template <class DataType, class CriteriaType>
long int Heap<DataType, CriteriaType>::Parent(long int Child)
{		// Begin Parent

		return (Child - 1) / 2;

}		// End Parent

//-------------------------Left-----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//----------------------------------------------------------------
// Calls to:  n/a
// Called by: Heap->Heapify()
//----------------------------------------------------------------
// Arguments: Root - The index of the root
//----------------------------------------------------------------
// Description: Returns the left child of a parent
//----------------------------------------------------------------
template <class DataType, class CriteriaType>
long int Heap<DataType, CriteriaType>::Left(long int Root)
{		// Begin Left
```

```
		return (Root * 2) + 1;

}		// End Left

//----------------------------Right-----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//---------------------------------------------------------------
// Calls to:  n/a
// Called by: n/a
//---------------------------------------------------------------
// Arguments: Root - The index of the root
//---------------------------------------------------------------
// Description:  Returns the right child of a parent
//---------------------------------------------------------------
template <class DataType, class CriteriaType>
long int Heap<DataType, CriteriaType>::Right(long int Root)
{		// Begin Right

		return (Root * 2) + 2;

}		// End Right

//---------------------------Delete-----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//---------------------------------------------------------------
// Calls to:  Heap->Heapify()
// Called by: n/a
//---------------------------------------------------------------
// Arguments: Data - The object to be deleted
//---------------------------------------------------------------
// Description: If the object exists, deltes that object
//		from the heap
//---------------------------------------------------------------
template <class DataType, class CriteriaType>
void Heap<DataType, CriteriaType>::Delete(DataType Data)
{		// Begin Delete

		// Search for the information
		long int index = Find(Data);

		// If not found
		if (index == -1)
		{		// Begin if

				cout<<endl<<"That information wasnt found in the heap."<<endl;
				return;

		}		// End if
		// Else if was found
		else
		{		// Begin else

				// Shift everything over one to the left
				for(int i=index; i<HeapLength; i++)
				{		// Begin for

						HeapArray[i].ElementData = HeapArray[i+1].ElementData;
						HeapArray[i].Criteria = HeapArray[i+1].Criteria;

				}		// End for

				// Effectively removes the last item which is now duplicated
```

```
                        HeapLength--;

                        cout<<endl<<"Information successfully deleted."<<endl;

                        // Calls Heapify to maintain the heap structure
                        Heapify(HeapLength);

                }               // End else

}               // End Delete

//-----------------------DeleteAtIndex-------------------------
// James Fishbaugh  CS 441-01
// 11/12/2005
// Last Modified:  11/12/05
//-------------------------------------------------------------
// Calls to:  Heap->Heapify()
// Called by: Logic->PerformSubMenuOption()
//-------------------------------------------------------------
// Arguments: IndexToDelete - the index of the data to
//                      delete
//-------------------------------------------------------------
// Description: Deletes the data at the index specifed
//           if the index exists.
//-------------------------------------------------------------
template <class DataType, class CriteriaType>
void Heap<DataType, CriteriaType>::DeleteAtIndex(long int IndexToDelete)
{               // Begin DeleteAtIndex

                // Shift everything over one to the left
                for(int i=IndexToDelete; i<HeapLength-1; i++)
                {               // Begin for

                        HeapArray[i].ElementData = HeapArray[i+1].ElementData;
                        HeapArray[i].Criteria = HeapArray[i+1].Criteria;

                }               // End for

                // Effectively removes the last item which is now duplicated
                HeapLength--;

                cout<<endl<<"Information successfully deleted."<<endl;

                // Calls Heapify to maintain the heap structure
                Heapify(HeapLength);

}               // End DeleteAtIndex

//---------------------------Empty-----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/12/05
//-------------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->PerformSubMenuOption()
//-------------------------------------------------------------
// Arguments: void
//-------------------------------------------------------------
// Description: Deletes the heap
//-------------------------------------------------------------
template <class DataType, class CriteriaType>
void Heap<DataType, CriteriaType>::Empty()
{               // Begin Empty

                if (HeapLength == 0)
                {               // Begin if
```

```
                cout<<endl<<"Heap is already empty"<<endl;

        }       // End if
        else
        {       // Begin else

                HeapLength=0;
                cout<<endl<<"Heap successfully emptied"<<endl;

        }       // End else

}       // End Empty

//----------------------------Find----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/12/05
//------------------------------------------------------------
// Calls to:  strcmp()
//                                      ContactType->GetName();
// Called by: Heap->Delete()
//                                      Logic->FindItemInHeap()
//------------------------------------------------------------
// Arguments: Data - The object of information
//------------------------------------------------------------
// Description: Searches for an object with that name in the
//          heap, case sensitive.
//------------------------------------------------------------
template <class DataType, class CriteriaType>
long int Heap<DataType, CriteriaType>::Find(CriteriaType Criteria)
{       // Begin Find

        // The index where the data is found
        // Negitive 1 implies it was not found
        long int IndexFound = -1;

        for(int i=0; i < HeapLength; i++)
        {       // Begin for

                // If the criteria is Name
                if ((Type == 1) || (Type == 4) || (Type == 7))
                {       // Begin if

                        if ( strcmp(((DataType)HeapArray[i].ElementData)->GetName(),
                                                        Criteria) == 0)
                        {       // Begin if

                                IndexFound = i;

                        }       // End if

                }       // End if
                // If the criteria is Address
                else if ((Type == 2) || (Type == 5) || (Type == 8))
                {       // Begin else if

                        if ( strcmp(((DataType)HeapArray[i].ElementData)->GetAddress(),
                                                        Criteria) == 0)
                        {       // Begin if

                                IndexFound = i;

                        }       // End if

                }       // End else if
```

```
                        // If the criteria is Phone
                        else
                        {               // Begin else if

                                        if ( strcmp(((DataType)HeapArray[i].ElementData)->GetPhone(),
                                                                        Criteria) == 0)
                                        {               // Begin if

                                                        IndexFound = i;

                                        }               // End if

                        }               // End else

                }               // End for

                return IndexFound;

}               // End Find

//-----------------------------Add-------------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//---------------------------------------------------------------
// Calls to:  Heap->Heapify()
//                                      Heap->Parent()
//          CriteriaMapper->Map()
// Called by: Logic->PerformSubMenuOption()
//---------------------------------------------------------------
// Arguments: Data - The oject of information
//          Criteria - A string to be sent to the mapping
//                  function
//---------------------------------------------------------------
// Description: Takes the information and criteria sent in
//           and correctly places it in the heap based
//           on the hash function
//---------------------------------------------------------------
template <class DataType, class CriteriaType>
void Heap<DataType, CriteriaType>::Add(DataType Data, CriteriaType Criteria)
{               // Begin Add

                HeapArray[HeapLength].Criteria = HeapMapper->Map(Criteria);
                HeapArray[HeapLength].ElementData = Data;

                long int new_pos = HeapLength;
                HeapLength++;

                while ((new_pos != 0) && (HeapArray[new_pos].Criteria > HeapArray[Parent(new_pos)].Criteria))
                {               // Begin while

                                DataType tempData = HeapArray[new_pos].ElementData;
                                long int tempCriteria = HeapArray[new_pos].Criteria;
                                HeapArray[new_pos].ElementData = HeapArray[Parent(new_pos)].ElementData;
                                HeapArray[new_pos].Criteria = HeapArray[Parent(new_pos)].Criteria;
                                HeapArray[Parent(new_pos)].ElementData = tempData;
                                HeapArray[Parent(new_pos)].Criteria = tempCriteria;

                                new_pos = Parent(new_pos);

                }               // End while

                // Call Heapify
                Heapify(HeapLength);

}               // End Add
```

```
//-------------------------Print-------------------------------
// James Fishbaugh  CS 441-01
// 10/12/2005
// Last Modified:  10/12/05
//-------------------------------------------------------------
// Calls to:  ContactType->Write()
// Called by: Logic->PerformSubMenuOption()
//-------------------------------------------------------------
// Arguments: IndexToPrint - the index to print
//-------------------------------------------------------------
// Description: Prints a specific item from the heap
//-------------------------------------------------------------
template <class DataType, class CriteriaType>
void Heap<DataType, CriteriaType>::Print(int IndexToPrint)
{		// Begin PrintAll

		// Make sure the index exists so we are not reaching
		// for memory that isnt there
		if (HeapLength >= IndexToPrint)
		{		// Begin if

				HeapArray[IndexToPrint].ElementData->Write();

		}		// End if

}		// End PrintAll()

//-------------------------PrintAll----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//-------------------------------------------------------------
// Calls to:  cout<<
//                                      ContactType->Write()
// Called by: Logic->PerformSubMenuOption()
//-------------------------------------------------------------
// Arguments: void
//-------------------------------------------------------------
// Description: Prints all the information in the heap
//-------------------------------------------------------------
template <class DataType, class CriteriaType>
void Heap<DataType, CriteriaType>::PrintAll()
{		// Begin PrintAll

		if (HeapLength > 0)
		{		// Begin if

				for (int i=0; i < HeapLength; i++)
				{		// Begin for

						HeapArray[i].ElementData->Write();

				}		// End for

		}		// End if
		else
		{		// Begin else

				cout<<endl<<"The heap is empty"<<endl;

		}		// End else

}		// End PrintAll()

//---------------------PrintHeapType-------------------------
```

```
// James Fishbaugh  CS 441-01
// 10/12/2005
// Last Modified:  10/12/05
//---------------------------------------------------------------
// Calls to:  cout<<
// Called by: Logic->ShowHeaps()
//---------------------------------------------------------------
// Arguments: void
//---------------------------------------------------------------
// Description:  Prints out the types of heaps
//---------------------------------------------------------------
template <class DataType, class CriteriaType>
void Heap<DataType, CriteriaType>::PrintHeapType()
{		// Begin PrintHeapType

		switch (Type)
		{		// Begin switch

			case 1:

				cout<<"Employee heap with name criteria"<<endl;
				break;

			case 2:

				cout<<"Employee heap with address criteria"<<endl;
				break;

			case 3:

				cout<<"Employee heap with phone criteria"<<endl;
				break;

			case 4:

				cout<<"Customer heap with name criteria"<<endl;
				break;

			case 5:

				cout<<"Customer heap with address criteria"<<endl;
				break;

			case 6:

				cout<<"Customer heap with phone criteria"<<endl;
				break;

			case 7:

				cout<<"Shipper heap with name criteria"<<endl;
				break;

			case 8:

				cout<<"Shipper heap with address criteria"<<endl;
				break;

			case 9:

				cout<<"Shipper heap with phone criteria"<<endl;
				break;

		}		// End switch

}		// End PrintHeapType
```

```
//----------------------GetHeapType--------------------------
// James Fishbaugh  CS 441-01
// 10/12/2005
// Last Modified:  10/12/05
//-----------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->FindItemInHeap()
//-----------------------------------------------------------
// Arguments: void
//-----------------------------------------------------------
// Description:  Returns the type of heap
//-----------------------------------------------------------
template <class DataType, class CriteriaType>
int Heap<DataType, CriteriaType>::GetHeapType()
{               // Begin GetHeapType

                return Type;

}               // End GetHeapType
```

ContactType Class

```
//----------------------ContactType.h-------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/13/05
//-----------------------------------------------------------
// Purpose - This file contains variable definitions and
//                                      function declarations for ContactType.cpp
//        which is the abstract base class in an
//                                      inheritance hierarchy.
//-----------------------------------------------------------

// If not defined, define
#ifndef INC_CONTACTTYPE_H
#define INC_CONTACTTYPE_H

class ContactType
{               // Begin ContactType


                protected:

                // Member data
                                char * Name;
                                char ** Address;
                                char * Phone;

                public:

                // Pure virtual functions to be implemented
                // by the base classes
                                virtual void Read() = 0;
                                virtual void Read(char * Data) = 0;
                                virtual void Write() = 0;
                                virtual char * GetName() = 0;
                                virtual char * GetAddress() = 0;
                                virtual char * GetPhone() = 0;

};              // End ContactType
```

```
#endif;
```

Employee Class

```
//-------------------------Employee.h--------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/13/05
//---------------------------------------------------------------
// Purpose - This file contains variable definitions and
//                                      function declarations for Employee.cpp which
//                                 is inherited from ContactType.h
//---------------------------------------------------------------

// If not defined, define
#ifndef INC_EMPLOYEE_H
#define INC_EMPLOYEE_H

//  Include files
#include "ContactType.h"

class Employee:public ContactType
{               // Begin Employee

                private:

                // Member data
                                float Salary;

                public:

                // Default constructor
                                Employee();

                // Member functions
                                void Read();
                                void Read(char * Data);
                                void Write();
                                char * GetName();
                                char * GetAddress();
                                char * GetPhone();

};              // End Employee

#endif;

//-------------------------Employee.cpp-------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//---------------------------------------------------------------
// Purpose - Implements Employee class which stores and
//                                      retrives Employee information.
//---------------------------------------------------------------

// Include files
#include <iostream>
#include <stdio.h>      // For sscanf
#include "Employee.h"
using namespace std;
// Constants
const int LENGTH_OF_NAME = 20;
const int LENGTH_OF_ADDRESS = 20;
const int ADDRESS_FIELDS = 3;
```

```
const int LENGTH_OF_PHONE = 10;

//-------------------------Employee-----------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//--------------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->PerformSubMenuOption()
//                                      Logic->CreateHeap()
//--------------------------------------------------------------
// Arguments: void
//--------------------------------------------------------------
// Description: Default constructor for Employee
//--------------------------------------------------------------
Employee::Employee()
{               // Begin Employee

                // Allocates space for Name
                Name = new char[LENGTH_OF_NAME + 1];

                // Allocates space for Address
                Address = new char * [ADDRESS_FIELDS];

                for (int i = 0; i < ADDRESS_FIELDS ; i++)
                {               // Begin for

                                // Also allocates space for Address
                                Address[i] = new char[LENGTH_OF_ADDRESS + 1];

                }               // End for

                // Allocates space for Phone
                Phone = new char[LENGTH_OF_PHONE + 1];

                // Sets the salary to the default value of zero
                Salary = 0.0;

}               // End Employee

//----------------------------Read----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/12/05
//--------------------------------------------------------------
// Calls to:  cout<<
//          cin>>
//                                  getline()
// Called by: Logic->PerformSubMenuOption()
//--------------------------------------------------------------
// Arguments: void
//--------------------------------------------------------------
// Description: Reads in all Employee information
//--------------------------------------------------------------
void Employee::Read()
{               // Begin Read

                // Used to hold unwanted characters
                char Dummy[256];

                // Outputs a new line
                cout<<endl;

                // Basically flushes the input buffer so we
                // can get input
                cin.getline(Dummy, 256);
```

```
		// Gets the employee name from the user
		cout<<"Enter employee name ("<<LENGTH_OF_NAME<<" chars max): ";
		cin.getline(Name, LENGTH_OF_NAME+1);

		// Loop that gets the address from the user
		for (int i=0; i < ADDRESS_FIELDS; i++)
		{		// Begin for

			cout<<"Enter address line "<<i+1<<" ("<<LENGTH_OF_ADDRESS<<" chars max): ";
			cin.getline(Address[i], LENGTH_OF_ADDRESS+1);

		}		// End for

		// Get the phone number from the user
		cout<<"Enter phone number ("<<LENGTH_OF_PHONE<<" chars max): ";
		cin.getline(Phone, LENGTH_OF_PHONE+1);

		/// Get the salary from the user
		cout<<"Enter salary: ";
		cin>>Salary;

}		// End Read

//----------------------------Read-----------------------------
// James Fishbaugh  CS 441-01
// 10/13/2005
// Last Modified:  10/13/05
//---------------------------------------------------------------
// Calls to:  sscanf()
// Called by: Logic->PerformSubMenuOption()
//---------------------------------------------------------------
// Arguments: Data - a character string with the different
//                fields seperated by spaces
//---------------------------------------------------------------
// Description: Reads in all Employee information as sent
//           in as a character array
//---------------------------------------------------------------
void Employee::Read(char * Data)
{		// Begin Read

		// Parses Data into the data space of Employee
		sscanf(Data, "%s %s %s %s %s %f", Name, Address[0],
						Address[1], Address[2], Phone, &Salary);

}		// End Read

//----------------------------Write----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//---------------------------------------------------------------
// Calls to:  cout<<
// Called by: Heap->Print()
//                                      Heap->PrintAll()
//---------------------------------------------------------------
// Arguments: void
//---------------------------------------------------------------
// Description: Writes all Employee information
//---------------------------------------------------------------
void Employee::Write()
{		// Begin Write

		// Prints the name
		cout<<endl<<"Employee name:   "<<Name<<endl;
```

```
		// Loop that prints the address
		for (int i=0; i < ADDRESS_FIELDS; i++)
		{		// Begin for

			cout<<"Address line "<<i+1<<":  "<<Address[i]<<endl;

		}		// End for

		// Prints the phone number
		cout<<"Phone number:    "<<Phone<<endl;

		// Prints the salary
		cout<<"Salary:        $"<<Salary<<endl;

}		// End Write

//---------------------------GetName---------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//---------------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->PerformSubMenuOption()
//---------------------------------------------------------------
// Arguments: void
//---------------------------------------------------------------
// Description: Returns the name
//---------------------------------------------------------------
char * Employee::GetName()
{		// Begin GetName

		return Name;

}		// End GetName

//-------------------------GetAddress---------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//---------------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->PerformSubMenuOption()
//---------------------------------------------------------------
// Arguments: void
//---------------------------------------------------------------
// Description: Returns the first address field
//---------------------------------------------------------------
char * Employee::GetAddress()
{		// Begin GetAddress

		return Address[0];

}		// End GetAddress

//---------------------------GetPhone---------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//---------------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->PerformSubMenuOption()
//---------------------------------------------------------------
// Arguments: void
//---------------------------------------------------------------
// Description: Returns the phone number
//---------------------------------------------------------------
```

```
char * Employee::GetPhone()
{		// Begin GetPhone

		return Phone;

}		// End GetPhone
```

Customer Class

```
//-------------------------Customer.h--------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/13/05
//-------------------------------------------------------------
// Purpose - This file contains variable definitions and
//                                      function declarations for Customer.cpp which
//        is inherited from ContactType.h
//-------------------------------------------------------------

// If not defined, define
#ifndef INC_CUSTOMER_H
#define INC_CUSTOMER_H

// Include files
#include "ContactType.h"

class Customer:public ContactType
{		// Begin Customer

		private:

		// Member data
			float Purchases;
			float AmountDue;

		public:

		// Default constructor
			Customer();

		// Member functions
			void Read();
			void Read(char * Data);
			void Write();
			char * GetName();
			char * GetAddress();
			char * GetPhone();

};		// End Customer

#endif;


//-------------------------Customer.cpp-------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/13/05
//-------------------------------------------------------------
// Purpose - Implements Customer class which stores and
//                                      retrives Customer information.
//-------------------------------------------------------------
```

```
// Include files
#include <iostream>
#include <stdio.h>      // For sscanf
#include "Customer.h"
using namespace std;
// Constants
const int LENGTH_OF_NAME = 20;
const int LENGTH_OF_ADDRESS = 20;
const int ADDRESS_FIELDS = 4;
const int LENGTH_OF_PHONE = 10;

//-------------------------Customer-----------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//--------------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->PerformSubMenuOption()
//                                      Logic->CreateHeap()
//--------------------------------------------------------------
// Arguments: void
//--------------------------------------------------------------
// Description: Default constructor for Customer
//--------------------------------------------------------------
Customer::Customer()
{               // Begin Customer

                // Allocates space for Name
                Name = new char[LENGTH_OF_NAME + 1];

                // Allocates space for Address
                Address = new char * [ADDRESS_FIELDS];

                for (int i = 0; i < ADDRESS_FIELDS ; i++)
                {               // Begin for

                                // Also allocates space for Address
                                Address[i] = new char[LENGTH_OF_ADDRESS + 1];

                }               // End for

                // Allocates space for Phone
                Phone = new char[LENGTH_OF_PHONE + 1];

                // Sets Purchases and AmountDue to the default value of zero
                Purchases = 0.0;
                AmountDue = 0.0;

}               // End Customer

//---------------------------Read-----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/12/05
//--------------------------------------------------------------
// Calls to:  cout<<
//          cin>>
//                                      getline()
// Called by: Logic->PerformSubMenuOption()
//--------------------------------------------------------------
// Arguments: void
//--------------------------------------------------------------
// Description: Reads in all Customer information
//--------------------------------------------------------------
void Customer::Read()
```

```
{               // Begin Read

                // Used to hold unwanted characters
                char Dummy[256];

                // Outputs a new line
                cout<<endl;

                // Basically flushes the input buffer so we
                // can get input
                cin.getline(Dummy, 256);

                // Gets the customer name from the user
                cout<<"Enter customer name ("<<LENGTH_OF_NAME<<" chars max): ";
                cin.getline(Name, LENGTH_OF_NAME+1);

                // Loop that gets the address from the user
                for (int i=0; i < ADDRESS_FIELDS; i++)
                {               // Begin for

                                cout<<"Enter address line "<<i+1<<" ("<<LENGTH_OF_ADDRESS<<" chars max): ";
                                cin.getline(Address[i], LENGTH_OF_ADDRESS+1);

                }               // End for

                // Gets the phone number from the user
                cout<<"Enter phone number ("<<LENGTH_OF_PHONE<<" chars max): ";
                cin.getline(Phone, LENGTH_OF_PHONE+1);

                // Gets the purchases from the user
                cout<<"Enter purchases: ";
                cin>>Purchases;

                // Gets the amount due from the user
                cout<<"Enter amount due: ";
                cin>>AmountDue;

}               // End Read

//-----------------------------Read-----------------------------
// James Fishbaugh  CS 441-01
// 10/13/2005
// Last Modified:  10/13/05
//--------------------------------------------------------------
// Calls to:  sscanf()
// Called by: Logic->PerformSubMenuOption()
//--------------------------------------------------------------
// Arguments: Data - a character string with the different
//                   fields seperated by spaces
//--------------------------------------------------------------
// Description: Reads in all Customer information as sent
//              in as a character array
//--------------------------------------------------------------
void Customer::Read(char * Data)
{               // Begin Read

                // Parses Data into the data space of Customer
                sscanf(Data, "%s %s %s %s %s %s %f %f", Name, Address[0],
                                                        Address[1], Address[2], Address[3],
                                                        Phone, &Purchases, &AmountDue);

}               // End Read

//----------------------------Write-----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
```

```
// Last Modified:  10/11/05
//--------------------------------------------------------
// Calls to:  cout<<
// Called by: Heap->Print()
//                                      Heap->PrintAll()
//--------------------------------------------------------
// Arguments: void
//--------------------------------------------------------
// Description: Writes all Customer information
//--------------------------------------------------------
void Customer::Write()
{               //Begin Write

                // Prints name
                cout<<endl<<"Customer name: "<<Name<<endl;

                // Loop that prints address
                for (int i=0; i < ADDRESS_FIELDS; i++)
                {               // Begin for

                                cout<<"Address line "<<i+1<<": "<<Address[i]<<endl;

                }               // End for

                // Prints phone number
                cout<<"Phone number: "<<Phone<<endl;

                // Prints purchases
                cout<<"Purchases: "<<Purchases<<endl;
                // Prints amount due
                cout<<"Amount due: $"<<AmountDue<<endl;

}               // End Write

//----------------------------GetName----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//--------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->PerformSubMenuOption()
//--------------------------------------------------------
// Arguments: void
//--------------------------------------------------------
// Description: Returns the name
//--------------------------------------------------------
char * Customer::GetName()
{               // Begin GetName

                return Name;

}               // End GetName

//-------------------------GetAddress----------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//--------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->PerformSubMenuOption()
//--------------------------------------------------------
// Arguments: void
//--------------------------------------------------------
// Description: Returns the first address field
//--------------------------------------------------------
char * Customer::GetAddress()
```

```
{               // Begin GetAddress

                return Address[0];

}               // End GetAddress

//-------------------------GetPhone--------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//-----------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->PerformSubMenuOption()
//-----------------------------------------------------------
// Arguments: void
//-----------------------------------------------------------
// Description: Returns the phone number
//-----------------------------------------------------------
char * Customer::GetPhone()
{               // Begin GetPhone

                return Phone;

}               // End GetPhone
```

Shipper Class

```
//-------------------------Shipper.h-------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/11/05
//-----------------------------------------------------------
// Purpose - This file contains variable definitions and
//                                      function declarations for Shipper.cpp which
//        is inherited from ContactType.h
//-----------------------------------------------------------

// If not defined, define
#ifndef INC_SHIPPER_H
#define INC_SHIPPER_H

// Include files
#include "ContactType.h"

class Shipper:public ContactType
{               // Begin Shipper

                private:

                // Member data
                                int Pickups;

                public:

                // Default constructor
                                Shipper();

                // Member functions
                                void Read();
                                void Read(char * Data);
                                void Write();
                                char * GetName();
```

```
                        char * GetAddress();
                        char * GetPhone();

};              // End Shipper

#endif;



//------------------------Shipper.cpp--------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/13/05
//---------------------------------------------------------------
// Purpose - Implements Shipper class which stores and
//                                      retrives Shipper information.
//---------------------------------------------------------------

// Include files
#include <iostream>
#include <stdio.h>      // For sscanf
#include "Shipper.h"
using namespace std;

// Constants
const int LENGTH_OF_NAME = 30;
const int LENGTH_OF_ADDRESS = 20;
const int ADDRESS_FIELDS = 3;
const int LENGTH_OF_PHONE = 10;

//-------------------------Shipper-----------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//---------------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->PerformSubMenuOption()
//                                      Logic->CreateHeap()
//---------------------------------------------------------------
// Arguments: void
//---------------------------------------------------------------
// Description: Default constructor for Shipper
//---------------------------------------------------------------
Shipper::Shipper()
{               // Begin Shipper

                // Allocates space for Name
                Name = new char[LENGTH_OF_NAME + 1];

                // Allocates space for Address
                Address = new char * [ADDRESS_FIELDS];

                for (int i = 0; i < ADDRESS_FIELDS ; i++)
                {               // Begin for

                        // Also allocates space for Address
                        Address[i] = new char[LENGTH_OF_ADDRESS + 1];

                }               // End for

                // Allocates space for Phone
                Phone = new char[LENGTH_OF_PHONE + 1];

                // Sets the pickups to the default value of zero
                Pickups = 0;
```

```
}               // End Shipper

//----------------------------Read----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/11/05
//------------------------------------------------------------
// Calls to:  cout<<
//          cin>>
//                                          getline()
// Called by: Logic->PerformSubMenuOption()
//------------------------------------------------------------
// Arguments: void
//------------------------------------------------------------
// Description: Reads all shipper information
//------------------------------------------------------------
void Shipper::Read()
{               // Begin Read

                // Used to hold unwanted characters
                char Dummy[256];

                // Outputs a new line
                cout<<endl;

                // Basically flushes the input buffer so we
                // can get input
                cin.getline(Dummy, 256);

                // Gets the shipper name from the user
                cout<<"Enter shipper name ("<<LENGTH_OF_NAME<<" chars max): ";
                cin.getline(Name, LENGTH_OF_NAME+1);

                // Loop that gets the address from the user
                for (int i=0; i < ADDRESS_FIELDS; i++)
                {               // Begin for

                                cout<<"Enter address line "<<i+1<<" ("<<LENGTH_OF_ADDRESS<<" chars max): ";
                                cin.getline(Address[i], LENGTH_OF_ADDRESS+1);

                }               // End for

                // Gets the phone number from the user
                cout<<"Enter phone number ("<<LENGTH_OF_PHONE<<" chars max): ";
                cin.getline(Phone, LENGTH_OF_PHONE+1);

                // Gets the pickups from the user
                cout<<"Enter pickups: ";
                cin>>Pickups;

}               // End Read

//----------------------------Read----------------------------
// James Fishbaugh  CS 441-01
// 10/13/2005
// Last Modified:  10/13/05
//------------------------------------------------------------
// Calls to:  sscanf()
// Called by: Logic->PerformSubMenuOption()
//------------------------------------------------------------
// Arguments: Data - a character string with the different
//                   fields seperated by spaces
//------------------------------------------------------------
// Description: Reads in all Shipper information as sent
//            in as a character array
//------------------------------------------------------------
```

```
void Shipper::Read(char * Data)
{		// Begin Read

		// Parses Data into the data space of Shipper
		sscanf(Data, "%s %s %s %s %s %d", Name, Address[0],
							Address[1], Address[2], Phone, &Pickups);

}		// End read

//-------------------------Write-----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//-----------------------------------------------------------
// Calls to:  cout<<
// Called by: Heap->Print()
//                                      Heap->PrintAll()
//-----------------------------------------------------------
// Arguments: void
//-----------------------------------------------------------
// Description: Prints all shipper information
//-----------------------------------------------------------
void Shipper::Write()
{		// Begin Write

		// Prints the shipper name
		cout<<endl<<"Shipper name: "<<Name<<endl;

		// Loop that prints the address
		for (int i=0; i < ADDRESS_FIELDS; i++)
		{		// Begin for

			cout<<"Address line "<<i+1<<": "<<Address[i]<<endl;

		}		// End for

		// Prints the phone number
		cout<<"Phone number: "<<Phone<<endl;

		// Prints the pickups
		cout<<"Pickups: "<<Pickups<<endl;

}		// End Write


//-------------------------GetName---------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/11/05
//-----------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->PerformSubMenuOption()
//-----------------------------------------------------------
// Arguments: void
//-----------------------------------------------------------
// Description: Returns the name
//-----------------------------------------------------------
char * Shipper::GetName()
{		// Begin GetName

		return Name;

}		// End GetName

//-----------------------GetAddress---------------------------
// James Fishbaugh  CS 441-01
```

```
// 10/11/2005
// Last Modified:  10/11/05
//---------------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->PerformSubMenuOption()
//---------------------------------------------------------------
// Arguments: void
//---------------------------------------------------------------
// Description: Returns the first address field
//---------------------------------------------------------------
char * Shipper::GetAddress()
{		// Begin GetAddress

		return Address[0];

}		// End GetAddress

//-------------------------GetPhone--------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//---------------------------------------------------------------
// Calls to:  n/a
// Called by: Logic->PerformSubMenuOption()
//---------------------------------------------------------------
// Arguments: void
//---------------------------------------------------------------
// Description: Returns the phone number
//---------------------------------------------------------------
char * Shipper::GetPhone()
{		// Begin GetPhone

		return Phone;

}		// End GetPhone
```

UI Class

```
//-------------------------UI.h-----------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/12/05
//---------------------------------------------------------------
// Purpose - This file contains variable definitions and
//                                        function declarations for UI.cpp
//---------------------------------------------------------------

// If not defined, define
#ifndef INC_UI_H
#define INC_UI_H

class UI
{		// Begin UI

		public:

		// Member functions
			void ShowMainMenu();
			void ShowSubMenu();
			int GetMenuChoice();
			void HeapTypeOptions();
			void HeapCriteriaOptions();
			void PrintError(int ErrorNumber);
			void Print(char * Message);
```

```
                        void PrintInteger(int IntToPrint);
                        char * GetUnique(int HeapType);

};              // End UI

#endif;



//------------------------UI.cpp-----------------------------
// James Fishbaugh  CS 441-01
// 10/10/2005
// Last Modified:  10/13/05
//---------------------------------------------------------------
// Purpose - The user interface which handles mostly all
//          input and output.
//---------------------------------------------------------------

// Include files
#include <iostream>
#include "UI.h"
using namespace std;

//-----------------------ShowMainMenu--------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//---------------------------------------------------------------
// Calls to:  cout<<
// Called by: main
//                                      Logic->PerformMainMenuOption
//                                      Logic->PerformSubMenuOption
//---------------------------------------------------------------
// Arguments: void
//---------------------------------------------------------------
// Description: Displays the main menu
//---------------------------------------------------------------
void UI::ShowMainMenu()
{               // Begin ShowMainMenu

                cout<<endl<<"Main Menu"<<endl;
                cout<<"---------"<<endl;
                cout<<"1) Create a new heap"<<endl;
                cout<<"2) Work with an existing heap"<<endl;
                cout<<"3) Quit"<<endl<<endl;

                cout<<"Enter your choice: ";

}               // End ShowMainMenu

//-----------------------ShowSubMenu--------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//---------------------------------------------------------------
// Calls to:  cout<<
// Called by: Logic->PerformMainMenuOption
//                                      Logic->PerformSubMenuOption
//---------------------------------------------------------------
// Arguments: void
//---------------------------------------------------------------
// Description: Displays the sub menu
//---------------------------------------------------------------
void UI::ShowSubMenu()
{               // Begin ShowSubMenu
```

```
		cout<<endl<<"Sub Menu"<<endl;
		cout<<"--------"<<endl;
		cout<<"1) Add"<<endl;
		cout<<"2) Find"<<endl;
		cout<<"3) Print all"<<endl;
		cout<<"4) Delete"<<endl;
		cout<<"5) Empty"<<endl;
		cout<<"6) Read"<<endl;
		cout<<"7) Delete entire heap"<<endl;
		cout<<"8) Back to main menu"<<endl;
		cout<<"9) Quit"<<endl<<endl;

		cout<<"Enter your choice: ";

}		// End ShowSubMenu

//-----------------------GetMenuChoice-------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//-------------------------------------------------------------
// Calls to:  cin>>
// Called by: Logic->PerformMainMenuOption
//                                      Logic->PerformSubMenuOption
//-------------------------------------------------------------
// Arguments: void
//-------------------------------------------------------------
// Description: Gets the menu choice from the user
//-------------------------------------------------------------
int UI::GetMenuChoice()
{		// Begin GetMenuChoice

		int choice;

		cin>>choice;

		return choice;

}		// End GetMenuChoice

//-----------------------HeapTypeSelection---------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//-------------------------------------------------------------
// Calls to:  cout<<
// Called by: Logic->PerformSubMenuOption
//-------------------------------------------------------------
// Arguments: void
//-------------------------------------------------------------
// Description: Prints out heap type options
//-------------------------------------------------------------
void UI::HeapTypeOptions()
{		// Begin HeapTypeSelection

		cout<<endl<<"What type of heap do you want to create?"<<endl;
		cout<<"---------------------------------------"<<endl;
		cout<<"1) Employee"<<endl;
		cout<<"2) Customer"<<endl;
		cout<<"3) Shipper"<<endl<<endl;

		cout<<"Enter your choice: ";

}		// End HeapTypeSelection
```

```
//--------------------HeapCriteriaOptions---------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//---------------------------------------------------------------
// Calls to:  cout<<
// Called by: Logic->CreateHeap()
//---------------------------------------------------------------
// Arguments: void
//---------------------------------------------------------------
// Description: Prints out criteria options
//---------------------------------------------------------------
void UI::HeapCriteriaOptions()
{		// Begin HeapCriteriaOptions

		cout<<endl<<"What type of criteria?"<<endl;
		cout<<"---------------------"<<endl;
		cout<<"1) Name"<<endl;
		cout<<"2) Address"<<endl;
		cout<<"3) Phone"<<endl<<endl;

		cout<<"Enter your choice: ";

}		// End HeapCriteriaOptions

//-------------------------PrintError--------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/11/05
//---------------------------------------------------------------
// Calls to:  cout<<
// Called by: Logic->PerformMainMenuOption
//                                      Logic->PerformSubMenuOption
//                                      Logic->ShowHeaps()
//                                      Logic->CreateHeap()
//                                Logic->GetHeapType()
//---------------------------------------------------------------
// Arguments: ErrorNumber - the error number to print
//---------------------------------------------------------------
// Description: Prints out error messages
//---------------------------------------------------------------
void UI::PrintError(int ErrorNumber)
{		// Begin PrintError

		switch (ErrorNumber)
		{		// Begin switch

				// Error code 1
				case 1:

						cout<<endl<<"*** Error *** Not a valid menu choice"<<endl;
						break;

				// Error code 2
				case 2:

						cout<<endl<<"*** Error *** No heaps exist"<<endl;
						break;

				// Error code 3
				case 3:

						cout<<endl<<"*** Error *** Data not found in heap"<<endl;
						break;

				// Error code 4
```

```
                        case 4:

                                cout<<endl<<"*** Error *** Already created the max amount of heaps"<<endl;
                                break;

                        // Should never get to default
                        default:

                                cout<<endl<<"*** Error ***"<<endl;
                                break;

                }               // End switch

}               // End PrintError

//-----------------------------Print-----------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/12/05
//---------------------------------------------------------------
// Calls to:  cout<<
// Called by: Logic->PerformSubMenuOption
//---------------------------------------------------------------
// Arguments: Message - the message to print
//---------------------------------------------------------------
// Description: Prints out Message
//---------------------------------------------------------------
void UI::Print(char * Message)
{               // Begin Print

                cout<<Message;

}               // End Print

//------------------------PrintInteger-----------------------------
// James Fishbaugh  CS 441-01
// 10/13/2005
// Last Modified:  10/13/05
//---------------------------------------------------------------
// Calls to:  cout<<
// Called by: Logic->PerformSubMenuOption
//                                      Logic->ShowHeaps()
//---------------------------------------------------------------
// Arguments: IntToPrint - the integer to print
//---------------------------------------------------------------
// Description: Prints out an integer
//---------------------------------------------------------------
void UI::PrintInteger(int IntToPrint)
{               // Begin PrintInteger

                cout<<IntToPrint;

}               // End PrintInteger

//-------------------------GetUnique-----------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/12/05
//---------------------------------------------------------------
// Calls to:  cout<<
//                                      cin>>
//                                      getline()
// Called by: Logic->FindItemInHeap()
//---------------------------------------------------------------
// Arguments: HeapType - the type of heap
//---------------------------------------------------------------
```

```
// Description: Prints out the correct prompt,gets
//                                                 the user input, and then returns it.
//-------------------------------------------------------------
char * UI::GetUnique(int HeapType)
{               // Begin GetUnique

                // To get the newline character
                char * Dummy = new char[256];
                char * UniqueString = new char[20];

                cout<<endl;

                if (HeapType == 1)
                {               // Begin if

                                cout<<"Enter employee name (max 20 chars): ";

                }               // End if
                // Find an Employee with Address criteria
                else if (HeapType == 2)
                {               // Begin else if

                                cout<<"Enter employee address line 1 (max 20 chars): ";

                }               // End else if
                // Find an Employee with Phone criteria
                else if (HeapType == 3)
                {               // Begin else if

                                cout<<"Enter employee phone number (max 9 chars): ";

                }               // End else if
                // Find a Customer with Name criteria
                else if (HeapType == 4)
                {               // Begin else if

                                cout<<"Enter customer name (max 30 chars): ";

                }               // End else if
                // Find a Customer with Address criteria
                else if (HeapType == 5)
                {               // Begin else if

                                cout<<"Enter customer address line 1 (max 20 chars): ";

                }               // End else if
                // Find a Customer with Phone criteria
                else if (HeapType == 6)
                {               // Begin else if

                                cout<<"Enter customer phone number (max 9 chars): ";


                }               // End else if
                // Find a Shipper with Name criteria
                else if (HeapType == 7)
                {               // Begin else if

                                cout<<"Enter shipper name (max 20 chars): ";

                }               // End else if
                // Find a Shipper with Address criteria
                else if (HeapType == 8)
                {               // Begin else if

                                cout<<"Enter shipper address line 1 (max 20 chars): ";
```

```
		}		// End else if
		// Add a Shipper with Phone criteria
		else
		{		// Begin else

				cout<<"Enter shipper phone number (max 9 chars): ";

		}		// End else

		cin.getline(Dummy, 256);
		cin.getline(UniqueString, 20+1);

		return UniqueString;

}		// End GetUnique
```

CriteriaMapper Class

```
//----------------------CriteriaMapper.h-----------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/12/05
//---------------------------------------------------------------
// Purpose - This file contains variable definitions and
//                                      function declarations for CriteriaMapper.cpp
//                                      This is an abstract class which provides
//                                  polymorphic functionality.
//---------------------------------------------------------------

// If not defined, define
#ifndef INC_CRITERIAMAPPER_H
#define INC_CRITERIAMAPPER_H

class CriteriaMapper
{  // Begin CriteriaMapper

		public:

		// Pure virtual function
		// Makes this an abstract class
			virtual long int Map(char * String) = 0;

};		// End CriteriaMapper

#endif;
```

NameMapper Class

```
//-------------------------NameMapper.h-------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//---------------------------------------------------------------
// Purpose - This file contains variable definitions and
//                                      function declarations for NameMapper.cpp,
//                                      is inherited from CriteriaMapper.h
//---------------------------------------------------------------
```

```
// If not defined, define
#ifndef INC_NAMEMAPPER_H
#define INC_NAMEMAPPER_H

// Include files
#include "CriteriaMapper.h"

class NameMapper:public CriteriaMapper
{  // Begin NameMapper

        public:

        // Member functions
                long int Map(char * String);

};      // End NameMapper

#endif;
```

```
//----------------------NameMapper.cpp-------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/12/05
//---------------------------------------------------------------
// Purpose - Implements NameMapper class provides a hash
//         function for heap storage
//---------------------------------------------------------------

// Include files
#include "NameMapper.h"

//-------------------------NameMapper-------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//---------------------------------------------------------------
// Calls to:  n/a
// Called by: Heap->Add()
//---------------------------------------------------------------
// Arguments: *String - a char string
//---------------------------------------------------------------
// Description:  Returns a hash value
//---------------------------------------------------------------
long int NameMapper::Map(char * String)
{       // Begin NameMapper

        // A long int to store the hash value
        long int HashValue;

        // The hash function
        HashValue = ((int)String[0] + (int)String[1] +
                                                (int)String[2]) % 1000;

        // Return the hash value
        return HashValue;

}       // End NameMapper
```

AddressMapper Class

```
//----------------------AddressMapper.h-------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/12/05
//---------------------------------------------------------------
// Purpose - This file contains variable definitions and
//                                      function declarations for AddressMapper.cpp,
//                                      is inherited from CriteriaMapper.h
//---------------------------------------------------------------

// If not defined, define
#ifndef INC_ADDRESSMAPPER_H
#define INC_ADDRESSMAPPER_H

// Include files
#include "CriteriaMapper.h"

class AddressMapper:public CriteriaMapper
{  // Begin AddressMapper

        public:

        // Member functions
                long int Map(char * String);

};      // End AddressMapper

#endif;
```

```
//----------------------AddressMapper.cpp-------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/12/05
//---------------------------------------------------------------
// Purpose - Implements AddressMapper class provides a hash
//      function for heap storage
//---------------------------------------------------------------

// Include files
#include "AddressMapper.h"

//-----------------------AddressMapper-------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/12/05
//---------------------------------------------------------------
// Calls to:  n/a
// Called by: Heap->Add()
//---------------------------------------------------------------
// Arguments: *String - a char string
//---------------------------------------------------------------
// Description: Returns a hash value
//---------------------------------------------------------------
long int AddressMapper::Map(char * String)
{       // Begin Map

        // A long int to store the hash value
        long int HashValue;

        // The hash function
```

```
        HashValue = ((int)String[0] + (int)String[1] +
                                                (int)String[2]) % 1000;

        // Return the hash value
        return HashValue;

}       // End Map
```

PhoneMapper Class

```
//------------------------PhoneMapper.h-------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//---------------------------------------------------------------
// Purpose - This file contains variable definitions and
//                                  function declarations for PhoneMapper.cpp,
//                                  is inherited from CriteriaMapper.h
//---------------------------------------------------------------

// If not defined, define
#ifndef INC_PHONEMAPPER_H
#define INC_PHONEMAPPER_H

// Include files
#include "CriteriaMapper.h"

class PhoneMapper:public CriteriaMapper
{  // Begin PhoneMapper

        public:

        // Member functions
                long int Map(char * String);

};      // End PhoneMapper

#endif;



//-----------------------PhoneMapper.cpp-------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  10/12/05
//---------------------------------------------------------------
// Purpose - Implements PhoneMapper class provides a hash
//         function for heap storage
//---------------------------------------------------------------

// Include files
#include "PhoneMapper.h"

//------------------------PhoneMapper----------------------------
// James Fishbaugh  CS 441-01
// 09/18/2005
// Last Modified:  09/20/05
//---------------------------------------------------------------
// Calls to:  n/a
// Called by: Heap->Add()
//---------------------------------------------------------------
// Arguments: *String - a char string
//---------------------------------------------------------------
// Description: Returns a hash value
//---------------------------------------------------------------
```

```
long int PhoneMapper::Map(char * String)
{		// Begin PhoneMapper

		// A long int to store the hash value
		long int HashValue;

		// The hash function
		HashValue = ((int)String[0] + (int)String[1] +
								(int)String[2]) % 1000;

		// Return the hash value
		return HashValue;

}		// End PhoneMapper
```

IO Class

```
//----------------------------IO.h-------------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/13/05
//---------------------------------------------------------------
// Purpose - This file contains variable definitions and
//                                      function declarations for IO.cpp
//---------------------------------------------------------------

// If not defined, define
#ifndef INC_IO_H
#define INC_IO_H

class IO
{		// Begin IO

		public:

		// Member functions
			char ** ReadFromFile(int & NumContacts);

};		// End IO

#endif;
```

```
//------------------------UI.cpp-------------------------------
// James Fishbaugh  CS 441-01
// 10/12/2005
// Last Modified:  10/12/05
//---------------------------------------------------------------
// Purpose - Handles the file input.
//---------------------------------------------------------------

// Include files
#include <stdio.h>
#include <fstream>
#include <iostream>
#include <stdlib.h>
#include "IO.h"

// The standard namespace
```

```
using namespace std;

// Constants
// The maximum number of information that can be read
const static int MAX_DATA_READ = 25;
const static int TOTAL_SIZE = 150;

//-----------------------ReadFromFile-------------------------
// James Fishbaugh  CS 441-01
// 10/11/2005
// Last Modified:  10/13/05
//-----------------------------------------------------------------
// Calls to:  is_open()
//                                      eof()
//                                      exit()
//                                      getline()
// Called by: Logic->PerformSubMenuOption()
//-----------------------------------------------------------------
// Arguments: NumContacts - the number of contacts
//-----------------------------------------------------------------
// Description: Reads info from a file and returns the
//           contents in a character array.
//-----------------------------------------------------------------
char ** IO::ReadFromFile(int & NumContacts)
{               // Begin ReadFromFile

                char * NameOfFile = new char[30];
                char * TypeOfContact = new char[10];
                char * Dummy = new char[256];

                // Allocated below
                char ** TempData;
                char ** ReturnData;

                int NumberOfContacts = 0;

                // Allocates space for TempData
                TempData = new char * [MAX_DATA_READ];

                for (int k = 0; k < MAX_DATA_READ; k++)
                {               // Begin for

                                // Also allocates space for Address
                                TempData[k] = new char[TOTAL_SIZE];

                }               // End for

                // Get the name of the file to open from the user
                cout<<endl<<"What is the name of the file (max 30 chars): ";
                cin>>NameOfFile;

                // Creates a new input stream object
                ifstream InputFile (NameOfFile);

                // Checks to make sure everything went ok opening the file
                if (!InputFile.is_open())
                {               // Begin if

                                cout<<endl<<"*** ERROR *** There was a problem opening ";
                                cout<<NameOfFile<<", exiting..."<<endl<<endl;

                                // If there was a problem, exit gracefully
                                exit(1);

                }               // End if
```

```
		// Get the type of contact which is the first
		// line of the file
		InputFile>>TypeOfContact;
		// Ignore the rest of the line
		InputFile.getline(Dummy, 256);

		// Keep reading info until the end of file is reached
		while (!InputFile.eof())
		{		// Begin while

				// Get the current line and store it in TempData[NumberOfContacts]
				InputFile.getline(TempData[NumberOfContacts], 150);
				// Increment the number of contacts
				NumberOfContacts++;

		}		// End while

		// Allocate up the ReturnData char string array so
		// it is the perfect size before it is returned
		ReturnData = new char * [MAX_DATA_READ];

		for (int k = 0; k < NumberOfContacts; k++)
		{		// Begin for

				// Also allocates space for Address
				ReturnData[k] = new char[TOTAL_SIZE];

		}		// End for

		// Copy TempData to ReturnData
		for (int i=0; i < NumberOfContacts; i++)
		{		// Begin for

				for (int j=0; j < TOTAL_SIZE; j++)
				{		// Begin for

						ReturnData[i][j] = TempData[i][j];

				}		// End for

		}		// End for

		// Sets NumContacts (passed by reference) to the NumberOfContacts
		NumContacts = NumberOfContacts;

		// Return the perfectly sized and formated data
		return ReturnData;

}		// End ReadFromFile

end
```

Appendix E - iUML model code

Logic Class

State 1

```
validInput = TRUE
myUI = find-one UI
[menuSelection] = UI1:Show_Main_Menu[] on myUI

#$INLINE
#printf("The number received back from UI is: %d \n", menuSelection);
#$ENDINLINE

loop
   validInput = TRUE
   switch menuSelection
      case 1
         generate LC1:User_Chose_Create_New_Heap() to this
         break
      case 2
         generate LC6:User_Chose_Existing_Heap() to this
         break
      case 3
          #Need something here
          break
      default
          validInput = FALSE
          [] = UI3:Show_Invalid_Input[] on myUI
          [menuSelection] = UI1:Show_Main_Menu[] on myUI
   endswitch

   if(validInput) then
       break
   endif
endloop
```

State 2

```
myUI = find-one UI

validInput = TRUE

loop
   validInput = TRUE
   [menuSelection] = UI2:Show_Heap_Type_Menu[] on myUI

   switch menuSelection
      case 1
         myHeap= create unique Heap with HeapType = 'employee'
       case 2
         myHeap= create unique Heap with HeapType = 'customer'
      case 3
         myHeap= create unique Heap with HeapType = 'shipper'

      default
          validInput = FALSE
          [] = UI3:Show_Invalid_Input[] on myUI
   endswitch

   if validInput then
       break
   endif
endloop

tempHeapNum = myHeap.HeapNumber

this.CurrentHeapNumber = tempHeapNum
```

```
generate LC2:User_Entered_Valid_Heap_Type() to this
```

State 3

```
myUI = find-one UI

myHeap = find-one Heap where HeapNumber = this.CurrentHeapNumber
tempHeapNum = this.CurrentHeapNumber

myCriteria = create unique CriteriaMapper with HeapNumber = tempHeapNum
link myHeap R6 myCriteria

loop
   validInput = TRUE
   [menuSelection] = UI4:Show_Criteria_Type_Menu[] on myUI

   switch menuSelection
      case 1
         myCriteria.CriteriaType = 'name'
         break
      case 2
         myCriteria.CriteriaType = 'address'
         break
      case 3
         myCriteria.CriteriaType = 'phone'
          break
      default
          validInput = FALSE
          [] = UI3:Show_Invalid_Input[] on myUI
   endswitch

   if(validInput) then
       break
   endif
endloop

[] = CM3:CreateChild[] on myCriteria
generate LC4:User_Entered_Valid_Criteria_Type() to this
```

State 4

```
myUI = find-one UI

{heaps} = find-all Heap
heapCount = countof {heaps}

for h in {heaps} do
   []=HP8:PrintHeapType[] on h
endfor

tempHeapNumber = 0
[tempHeapNumber] = UI6:Show_Choose_Heap_Menu[heapCount] on myUI

this.CurrentHeapNumber = tempHeapNumber
myHeap = find-one Heap where HeapNumber = tempHeapNumber

loop
```

```
  [menuSelection] = UI5:Show_Sub_Menu[] on myUI

  switch menuSelection
    case 1
      [] = HP1:Add[] on myHeap
      break
    case 2
      [] = HP2:Find[] on myHeap
      break
    case 3
       [] = HP3:PrintAll[] on myHeap
       break
    case 4
      [] = HP4:Delete[] on myHeap
      break
    case 5
       [] = HP5:Empty[] on myHeap
       break
    case 6
       [] = HP6:ReadFromFile[] on myHeap
       break
    case 7
       [] = HP5:Empty[] on myHeap
       delete myHeap
       generate LC7:Return_To_Main_Menu() to this
       break
    case 8
       generate LC7:Return_To_Main_Menu() to this
       break
    default
       #Do Nothing
  endswitch
endloop
```

Heap Class

Heap - Add

```
tempHeapNumber = this.HeapNumber
{allContactsInHeap} = find Contact where HeapNumber = tempHeapNumber
tempHeapLength = countof {allContactsInHeap}



#----------------------------------------BEGIN FACTORY
myContact = create unique Contact with HeapNumber = tempHeapNumber & ContactType = this.HeapType
link this R5 myContact

[] = CT5:CreateChild[] on myContact
[] = CT1:Read[] on myContact

#----------------------------------------END FACTORY

[] = HP7:InsertIntoHeap[myContact] on this
```

Heap - Find

```
myUI = find-one UserInterface
```

```
name = ""

$INLINE
printf("Enter the name of the person you would like to find: \n");
scanf("%s", name);
$ENDINLINE

person = find-one Contact where Name = name and HeapNumber = this.HeapNumber
[temp] = CT2:Write[] on person

[] = UI7:Print[temp] on myUI
```

Heap – PrintAll

```
myUI = find-one UserInterface

tempHeapNumber = this.HeapNumber

{contacts} = find Contact where HeapNumber = tempHeapNumber ordered by ArrayPosition

for con in {contacts} do
[temp] = CT2:Write[] on con
[] = UI7:Print[temp] on myUI
#$INLINE
#printf(temp);
#printf("\n");
#$ENDINLINE
Endfor
```

Heap - Delete

```
name = ""

$INLINE
printf("Enter the name of the person you would like to delete: \n");
scanf("%s", name);
$ENDINLINE

myContact = find-one Contact where Name = name & HeapNumber = this.HeapNumber

#Delete and unlink the child class
[] = CT4:Delete[] on myContact

unlink this R5 myContact
delete myContact
```

Heap - Empty

```
{myContacts} = find Contact where HeapNumber = this.HeapNumber

for con in {myContacts} do
```

```
  #Delete and unlink the child class
  [] = CT4:Delete[] on con

  unlink this R5 con
  delete con
endfor
```

Heap - ReadFromFile

```
myContact = create unique Contact with HeapNumber = this.HeapNumber & ContactType = 'employee'
link this R5 myContact
[] = CT5:CreateChild[] on myContact
[] = CT3:ReadFromFile["bob 4619Drive 4062430000 11111.00"] on myContact
[] = HP7:InsertIntoHeap[myContact] on this

myContact = create unique Contact with HeapNumber = this.HeapNumber & ContactType = 'employee'
link this R5 myContact
[] = CT5:CreateChild[] on myContact
[] = CT3:ReadFromFile["joe 4619Drive 4062430000 11111.00"] on myContact
[] = HP7:InsertIntoHeap[myContact] on this

myContact = create unique Contact with HeapNumber = this.HeapNumber & ContactType = 'employee'
link this R5 myContact
[] = CT5:CreateChild[] on myContact
[] = CT3:ReadFromFile["ted 4619Drive 4062430000 11111.00"] on myContact
[] = HP7:InsertIntoHeap[myContact] on this
```

Heap - InsertIntoHeap

```
#Simplified to highest-priority first stack, since can't use math.h so can't use floor()

myCriteria = find-one CriteriaMapper where HeapNumber = this.HeapNumber
[priority] = CM1:Map[myContact] on myCriteria
myContact.Priority = priority

myContact.ArrayPosition = this.Tail

tail = this.Tail
$INLINE
tail = tail + 1;
$ENDINLINE
this.Tail = tail

parentPos = 0
pos = myContact.ArrayPosition
{contacts} = find Contact where HeapNumber = this.HeapNumber
count = countof {contacts}

if count > 1 then
loop
  pos = myContact.ArrayPosition
  $INLINE
  parentPos = pos - 1;
  $ENDINLINE

  parent = find-one Contact where HeapNumber = this.HeapNumber & ArrayPosition = parentPos

  if(parent.Priority < myContact.Priority) then
    myContact.ArrayPosition = parentPos
    parent.ArrayPosition = pos
```

```
    if(parentPos = 0) then
      break
    endif
  else
    break
  endif
endloop
endif

#{contacts}## = find Contact where HeapNumber = this.HeapNumber
#count = countof {contacts}

#if count > 1 then
#   loop
#      pos = myContact.ArrayPosition
#      parentPos = 0
#     $INLINE
#        parentPos = pos - 1;
#     $ENDINLINE
#      parent = find-one Contact where HeapNumber = this.HeapNumber & ArrayPosition = parentPos

#      if myContact.Priority > parent.Priority then
#           myContact.ArrayPosition = parentPos
#           parent.ArrayPosition = pos
#      else
#          break
#      endif
#   endloop
#endif

#pos = myContact.ArrayPosition


#printf("POS: ");
#printf(pos);
#printf("\n");
#$INLINE
#printf("PRI: ");
#printf(priority);
#printf("\n");
#$ENDINLINE

test = 0
```

Heap - PrintHeapType

```
myCriteria = find-one CriteriaMapper where HeapNumber = this.HeapNumber

switch this.HeapType
   case 'employee'
      $INLINE
         printf("Employee   ");
      $ENDINLINE
   case 'customer'
      $INLINE
         printf("Customer   ");
      $ENDINLINE
   case 'shipper'
      $INLINE
         printf("Shipper   ");
      $ENDINLINE
endswitch
```

[] = CM2:PrintCriteriaType[] on myCriteria

Contact Class

Contact - CreateChild

```
switch this.ContactType
   case 'employee'
      myEmployee = create Employee with ContactNumber = this.ContactNumber
      link this R4 myEmployee
   case 'customer'
      myCustomer = create Customer with ContactNumber = this.ContactNumber
      link this R4 myCustomer
   case 'shipper'
      myShipper = create Shipper with ContactNumber = this.ContactNumber
      link this R4 myShipper
endswitch
```

Employee Class

Employee - Write

```
base = find-one Contact where ContactNumber = this.ContactNumber

name = base.Name
address = base.Address
phone = base.Phone
salary= this.Salary

data = name

#$INCLUDE_HEADER "string.h"
#$INLINE
#strcat(str,"   ");
#strcat(str, address);
#strcat(str,"   ");
#strcat(str, phone);
#printf("WRITE: ");
#printf(str);
#printf("\n");
#data = str;
#$ENDINLINE
```

Employee - ReadFromFile

```
base = find-one Contact where ContactNumber = this.ContactNumber

name = ""
address = ""
phone = ""
salary = ""

$INCLUDE_HEADER "stdio.h"
$INLINE

sscanf(data, "%s %s %s %s", name, address, phone, salary);
```

```
printf(name);
printf("\n");
printf(address);
printf("\n");
printf(phone);
printf("\n");
printf(salary);
printf("\n");

$ENDINLINE

test = 3

base.Name = name
base.Address = address
base.Phone = phone
this.Salary = salary
```

Employee - Read

```
base = find-one Contact where ContactNumber = this.ContactNumber

name = ""
address = ""
phone = ""
salary = ""

$INLINE
printf("Enter Name:\n");
scanf("%s", name);
printf("Enter Address:\n");
scanf("%s", address);
printf("Enter Phone:\n");
scanf("%s", phone);
printf("Enter Salary:\n");
scanf("%s", salary);
$ENDINLINE

base.Name = name
base.Address = address
base.Phone = phone
this.Salary = salary
```

Employee - Delete

```
myContact = find-one Contact where ContactNumber = this.ContactNumber

unlink this R4 myContact

delete this
```

Customer Class

Customer – Write

```
base = find-one Contact where ContactNumber = this.ContactNumber

name = base.Name
address = base.Address
phone = base.Address
purchases = this.Purchases
amountdue = this.AmountDue

data = name

#$INLINE
#include <string.h>
#strcat(data,name);
#strcat(data,"   ");
#strcat(data,address);
#strcat(data,"   ");
#strcat(data,phone);
#$ENDINLINE
```

Customer – ReadFromFile

```
base = find-one Contact where ContactNumber = this.ContactNumber

name = ""
address = ""
phone = ""
purchases = ""
amountdue = ""

$INCLUDE_HEADER "stdio.h"
$INLINE

sscanf(data, "%s %s %s %s %s", name, address, phone, purchases, amountdue);

printf(name);
printf("\n");
printf(address);
printf("\n");
printf(phone);
printf("\n");
printf(purchases);
printf("\n");
printf(amountdue);
printf("\n");

$ENDINLINE

test = 3

base.Name = name
base.Address = address
base.Phone = phone
this.Purchases = purchases
this.AmountDue = amountdue
```

Customer - Read

```
base = find-one Contact where ContactNumber = this.ContactNumber
```

```
name = ""
address = ""
phone = ""
purchases = ""
amountdue = ""

$INLINE
printf("Enter Name:\n");
scanf("%s", name);
printf("Enter Address:\n");
scanf("%s", address);
printf("Enter Phone:\n");
scanf("%s", phone);
printf("Enter Purchases:\n");
scanf("%s", purchases);
printf("Enter Amount Due:\n");
scanf("%s", amountdue);
$ENDINLINE

base.Name = name
base.Address = address
base.Phone = phone
this.Purchases = purchases
this.AmountDue = amountdue
```

Customer –Delete

```
myContact = find-one Contact where ContactNumber = this.ContactNumber
unlink this R4 myContact
delete this
```

Shipper Class

Shipper – Write

```
base = find-one Contact where ContactNumber = this.ContactNumber

name = base.Name
address = base.Address
phone = base.Phone
pickups= this.Pickups

data = name
```

Shipper – ReadFromFile

```
base = find-one Contact where ContactNumber = this.ContactNumber

name = ""
address = ""
phone = ""
pickups = ""

$INCLUDE_HEADER "stdio.h"
```

```
$INLINE

sscanf(data, "%s %s %s %s", name, address, phone, pickups);

printf(name);
printf("\n");
printf(address);
printf("\n");
printf(phone);
printf("\n");
printf(pickups);
printf("\n");

$ENDINLINE

test = 3

base.Name = name
base.Address = address
base.Phone = phone
this.Pickups = pickups
```

Shipper – Read

```
base = find-one Contact where ContactNumber = this.ContactNumber

name = ""
address = ""
phone = ""
pickups = ""

$INLINE
printf("Enter Name:\n");
scanf("%s", name);
printf("Enter Address:\n");
scanf("%s", address);
printf("Enter Phone:\n");
scanf("%s", phone);
printf("Enter Pickups:\n");
scanf("%s", pickups);
$ENDINLINE

base.Name = name
base.Address = address
base.Phone = phone
this.Pickups = pickups
```

Shipper - Delete

```
myContact = find-one Contact where ContactNumber = this.ContactNumber

unlink this R4 myContact

delete this
```

UserInterface Class

UI – Show Main Menu

```
menuSelection = 0

$INLINE
printf("\nMAIN MENU \n");
printf("1) Create a new heap \n");
printf("2) Work with an existing heap \n");
printf("3) Quit \n");
scanf("%d", menuSelection);
$ENDINLINE
```

UI – Show Heap Type Menu

```
menuSelection = 0

$INLINE
printf("What type of heap do you want to create? \n");
printf("1) Employee \n");
printf("2) Customer \n");
printf("3) Shipper \n");
scanf("%d", menuSelection);
$ENDINLINE
```

UI – Show Invalid Input

```
$INLINE
printf("Invalid input. \n");
$ENDINLINE
```

UI – Show Criteria Type Menu

```
menuSelection = 0

$INLINE
printf("What type of criteria? \n");
printf("1) Name\n");
printf("2) Address\n");
printf("3) Phone\n");
scanf("%d", menuSelection);
$ENDINLINE
```

UI – Show Sub Menu

```
menuSelection = 0

$INLINE
printf("\nSUB MENU \n");
printf("1) Add \n");
printf("2) Find \n");
printf("3) Print All \n");
printf("4) Delete \n");
printf("5) Empty \n");
printf("6) Read \n");
printf("7) Delete Entire Heap \n");
printf("8) Back to Main Menu \n");
printf("9) Quit \n");
scanf("%d", menuSelection);
$ENDINLINE
```

UI – Show Choose Heap Menu

```
menuSelection = 0

$INLINE
printf("\nChoose one of the following heaps: ");
scanf("%d", menuSelection);
$ENDINLINE
```

UI - Print

```
$INLINE
printf(str);
printf("\n");
$ENDINLINE
```

UI – Print Integer

```
$INLINE
printf(num);
printf("\n");
$ENDINLINE
```

CriteriaMapper Class

CriteriaMapper – Map

Pure virtual

CriteriaMapper – PrintCriteriaType

```
switch this.CriteriaType
   case 'name'
      $INLINE
         printf("Name");
         printf("\n");
      $ENDINLINE
   case 'address'
      $INLINE
         printf("Address");
         printf("\n");
      $ENDINLINE
   case 'phone'
      $INLINE
         printf("Phone");
         printf("\n");
      $ENDINLINE
endswitch
```

CriteriaMapper – CreateChild

```
switch this.CriteriaType
```

```
   case 'name'
      myName = create NameMapper with CriteriaNumber = this.CriteriaNumber
      link this R7 myName
   case 'address'
      myAddress = create AddressMapper with CriteriaNumber = this.CriteriaNumber
      link this R7 myAddress
   case 'phone'
      myPhone = create PhoneMapper with CriteriaNumber = this.CriteriaNumber
      link this R7 myPhone
endswitch
```

NameMapperClass

NameMapper – Map

```
name = con.Name
switch con.ContactNumber
#1 bob
#2 joe
#3 ted

case 1
  value = 307
case 2
  value = 318
case 3
  value = 317
default
  value = 555
endswitch


#The following code absolutely should work.  It does successfully get the value,
#but when it returns it returns a null value for some reason I cannot understand.
#Other sections of code return ints with no problem, but those are always values
#that are inputted by the user.  (e.g. UI.mainMenu).
#name = con.Name
#$INCLUDE_HEADER "stdio.h"
#$INCLUDE_HEADER "string.h"
#$INLINE
#int temp;
#temp = (int)((int)name[0] + (int)name[1] + (int)name[2]);
#value = temp;
#$ENDINLINE
#$INLINE
#printf("VALUE: ");
#printf("%d", value);
#printf("\n");
#$ENDINLINE
```

AddressMapper Class

AddressMapper – Map

```
switch con.ContactNumber
#1 bob
```

```
#2 joe
#3 ted

case 1
  value = 267
case 2
  value = 974
case 3
  value = 395
default
  value = 555
endswitch
```

PhoneMapper Class

PhoneMapper – Map

```
switch con.ContactNumber
#1 bob
#2 joe
#3 ted

case 1
  value = 847
case 2
  value = 224
case 3
  value = 943

endswitch
```

Initialization Segment

```
# This is for initialization and linking purposes.
# This cannot be used as a main method.

myLogic = create Logic with Current_State = 'Waiting_For_Program_Start'
myUI = create UI with Current_State = 'Waiting_For_Command'

#link myLogic R1 myUI

generate LC5:Start_Program() to myLogic
```

Test Methods

Employee Tests

#This Test leverages iUMLs built-in functionality to easily monitor the instances of records in the table

```
#This test should be run with instance windows open for Contact and Employee tables
#Places where the instance values should be checked are indicated with a comment.

myContact = create unique Contact with HeapNumber = 1 & ContactType = 'employee'

#Check that myContact instance is created

[] = CT5:CreateChild[] on myContact

#Check that Employee instance is created

[] = CT3:ReadFromFile["bob 4619Drive 4062430000 11111.00"] on myContact

#Check values read in to employee instance

[result] = CT2:Write[] on myContact

$INLINE
printf("\nTEST RESULTS ----------\n");
printf(result);
printf("\n---------------------------------------\n");
$ENDINLINE

[] = CT4:Delete[] on myContact

#Verify that Employee record is deleted

delete myContact

#Verify that Contact record is deleted


myContact = create unique Contact with HeapNumber = 1 & ContactType = 'employee'
[] = CT5:CreateChild[] on myContact

$INLINE
printf("ENTER TEST DATA: dan 1234Drive 4062430000 222222.00 \n");
$ENDINLINE

[] = CT1:Read[] on myContact

#Verify employee has appropriate values

[result] = CT2:Write[] on myContact
$INLINE
printf("\nTEST RESULTS ----------\n");
printf(result);
printf("\n---------------------------------------\n");
$ENDINLINE

#Verify data written out.

[] = CT4:Delete[] on myContact
delete myContact

#END OF TEST
```

Customer Tests

```
#This Test leverages iUMLs built-in functionality to easily monitor the instances of records in the table
#This test should be run with instance windows open for Contact and Customer tables
#Places where the instance values should be checked are indicated with a comment.

myContact = create unique Contact with HeapNumber = 1 & ContactType = 'customer'

#Check that myContact instance is created
```

```
[] = CT5:CreateChild[] on myContact

#Check that Customer instance is created

[] = CT3:ReadFromFile["bob 4619Drive 4062430000 111 222"] on myContact

#Check values read in to customer instance

[result] = CT2:Write[] on myContact

$INLINE
printf("\nTEST RESULTS ----------\n");
printf(result);
printf("\n---------------------------------------\n");
$ENDINLINE

[] = CT4:Delete[] on myContact

#Verify that Customer record is deleted

delete myContact

#Verify that Contact record is deleted


myContact = create unique Contact with HeapNumber = 1 & ContactType = 'customer'
[] = CT5:CreateChild[] on myContact

$INLINE
printf("ENTER TEST DATA: dan 1234Drive 4062430000 222 333 \n");
$ENDINLINE

[] = CT1:Read[] on myContact

#Verify customer has appropriate values

[result] = CT2:Write[] on myContact
$INLINE
printf("\nTEST RESULTS ----------\n");
printf(result);
printf("\n---------------------------------------\n");
$ENDINLINE

#Verify data written out.

[] = CT4:Delete[] on myContact
delete myContact

#END OF TEST
```

ShipperTests

```
#This Test leverages iUMLs built-in functionality to easily monitor the instances of records in the table
#This test should be run with instance windows open for Contact and Shipper tables
#Places where the instance values should be checked are indicated with a comment.

myContact = create unique Contact with HeapNumber = 1 & ContactType = 'shipper'

#Check that myContact instance is created

[] = CT5:CreateChild[] on myContact

#Check that Shipper instance is created

[] = CT3:ReadFromFile["bob 4619Drive 4062430000 123"] on myContact
```

```
#Check values read in to shipper instance

[result] = CT2:Write[] on myContact

$INLINE
printf("\nTEST RESULTS ----------\n");
printf(result);
printf("\n---------------------------------------\n");
$ENDINLINE

[] = CT4:Delete[] on myContact

#Verify that Shipper record is deleted

delete myContact

#Verify that Contact record is deleted


myContact = create unique Contact with HeapNumber = 1 & ContactType = 'shipper'
[] = CT5:CreateChild[] on myContact

$INLINE
printf("ENTER TEST DATA: dan 1234Drive 4062430000 234 \n");
$ENDINLINE

[] = CT1:Read[] on myContact

#Verify shipper has appropriate values

[result] = CT2:Write[] on myContact
$INLINE
printf("\nTEST RESULTS ----------\n");
printf(result);
printf("\n---------------------------------------\n");
$ENDINLINE

#Verify data written out.

[] = CT4:Delete[] on myContact
delete myContact

#END OF TEST
```

Criteria-Contact-Module

```
#This test script should be run with a local varibles window open in order to monitor the "value"
#variable.

myContact = create unique Contact with ContactType = 'employee'
[] = CT5:CreateChild[] on myContact
[] = CT3:ReadFromFile["bob 4619Drive 4062430000 11111.00"] on myContact

# NAME

myNameMapper = create unique CriteriaMapper with CriteriaType = 'name'
[] = CM3:CreateChild[] on myNameMapper
[value] = CM1:Map[myContact] on myNameMapper

$INLINE
printf("\nName value should be: 307\n");
$ENDINLINE
```

```
# ADDRESS

myAddressMapper = create unique CriteriaMapper with CriteriaType = 'address'
[] = CM3:CreateChild[] on myAddressMapper
[value] = CM1:Map[myContact] on myAddressMapper

$INLINE
printf("\nAddress value should be: 267 \n");
$ENDINLINE

# PHONE

myPhoneMapper = create unique CriteriaMapper with CriteriaType = 'phone'
[] = CM3:CreateChild[] on myPhoneMapper
[value] = CM1:Map[myContact] on myPhoneMapper

$INLINE
printf("\nPhone value should be: 847\n");
$ENDINLINE

#END OF TEST
```

Criteria-Heap-Module

```
heap1 = create unique Heap with HeapType = 'employee'

nC = create unique CriteriaMapper with HeapNumber = heap1.HeapNumber & CriteriaType = 'name'
[] = CM3:CreateChild[] on nC
link heap1 R6 nC

[] = HP8:PrintHeapType[] on heap1

$INLINE
printf("\n");
$ENDINLINE

nC.CriteriaType = 'address'
[] = HP8:PrintHeapType[] on heap1

$INLINE
printf("\n");
$ENDINLINE

nC.CriteriaType = 'phone'
[] = HP8:PrintHeapType[] on heap1

$INLINE
printf("\n");
$ENDINLINE

heap1.HeapType = 'customer'

nC.CriteriaType = 'name'
[] = HP8:PrintHeapType[] on heap1

$INLINE
printf("\n");
$ENDINLINE

nC.CriteriaType = 'address'
[] = HP8:PrintHeapType[] on heap1


$INLINE
printf("\n");
```

```
$ENDINLINE

nC.CriteriaType = 'phone'
[] = HP8:PrintHeapType[] on heap1

$INLINE
printf("\n");
$ENDINLINE

heap1.HeapType = 'shipper'

nC.CriteriaType = 'name'
[] = HP8:PrintHeapType[] on heap1

$INLINE
printf("\n");
$ENDINLINE

nC.CriteriaType = 'address'
[] = HP8:PrintHeapType[] on heap1

$INLINE
printf("\n");
$ENDINLINE

nC.CriteriaType = 'phone'
[] = HP8:PrintHeapType[] on heap1

$INLINE
printf("\n");
$ENDINLINE


# You should see 9 different combinations of heap and criteria type printed to the console

#End test
```

Heap-Contact-Criteria-Module

```
myHeap = create unique Heap with HeapType = 'employee'
myCriteria = create unique CriteriaMapper with HeapNumber = myHeap.HeapNumber & CriteriaType = 'name'
[] = CM3:CreateChild[] on myCriteria
link myHeap R6 myCriteria

#SEED WITH VALUES
myContact = create unique Contact with HeapNumber = myHeap.HeapNumber & ContactType = 'employee'
link myHeap R5 myContact
[] = CT5:CreateChild[] on myContact
[] = CT3:ReadFromFile["bob 4619Drive 4062430000 11111.00"] on myContact
[] = HP7:InsertIntoHeap[myContact] on myHeap

myContact = create unique Contact with HeapNumber = myHeap.HeapNumber & ContactType = 'employee'
link myHeap R5 myContact
[] = CT5:CreateChild[] on myContact
[] = CT3:ReadFromFile["joe 4619Drive 4062430000 11111.00"] on myContact
[] = HP7:InsertIntoHeap[myContact] on myHeap


#ADD
#INPUT THE FOLLOWING INFO: ted 4619Drive 4062430000 11111.00
[] = HP1:Add[] on myHeap
#Check that new Contact and Employee exists.


#INSERT INTO HEAP
#Verify that contact bob is in arrayposition # 2
```

#PRINT ALL
[] = HP3:PrintAll[] on myHeap

#Verify that they are printed in this order: joe, ted, bob

#END OF TEST

Heap-Contact-Module

myHeap = create unique Heap with HeapType = 'employee'
myCriteria = create unique CriteriaMapper with HeapNumber = myHeap.HeapNumber & CriteriaType = 'name'
[] = CM3:CreateChild[] on myCriteria

#READ FROM FILE
[] = HP6:ReadFromFile[] on myHeap
#Verify that there are 3 contact instances and 3 employee instances, with all fields populated

#DELETE
# Input: bob
[] = HP4:Delete[] on myHeap
#Verify that contact bob, and accompanying child class, have been deleted.

#FIND
# Input: ted
[] = HP2:Find[] on myHeap
#Verify that program output "ted"

#Empty
[] = HP5:Empty[] on myHeap
#Verify that all contact and child instances have been deleted.

#END OF TEST.

Logic-Heap-UI-Module

#Set state to Create New Heap.
#Input null, alpha, too large a number, too small a number, then valid number.
#Verify that heap was created.
#Verify that state advanced.

#myLogic = create Logic with Current_State = 'Waiting_For_Program_Start'
myLogic = create Logic with Current_State = 'Display_Main_Menu_And_Wait_For_Input'
#myLogic = create Logic with Current_State = 'Display_Type_Of_Heap_Menu_Wait_For_User_Input'
myUI = create UI with Current_State = 'Waiting_For_Command'

#generate LC5:Start_Program() to myLogic
generate LC1:User_Chose_Create_New_Heap() to myLogic

Logic-Criteria-UI-Module

#Set state to User Entered Valid Heap.
#Input null, alpha, too large a number, too small a number, then valid number.
#Verify that criteria instance was created. Verify that state advanced.

myLogic = create Logic with Current_State = 'Display_Type_Of_Heap_Menu_Wait_For_User_Input'
myUI = create UI with Current_State = 'Waiting_For_Command'

myHeap = create unique Heap with HeapType = 'employee'
myLogic.CurrentHeapNumber = 1

```
generate LC2:User_Entered_Valid_Heap_Type() to myLogic


#myLogic = create Logic with Current_State = 'Waiting_For_Program_Start'
myLogic = create Logic with Current_State = 'Display_Main_Menu_And_Wait_For_Input'
#myLogic = create Logic with Current_State = 'Display_Type_Of_Heap_Menu_Wait_For_User_Input'
myUI = create UI with Current_State = 'Waiting_For_Command'

#generate LC5:Start_Program() to myLogic
generate LC1:User_Chose_Create_New_Heap() to myLogic
```

Logic-UI-Module

```
# This is for initialization and linking purposes.
# This cannot be used as a main method.

myLogic = create Logic with Current_State = 'Waiting_For_Program_Start'
myUI = create UI with Current_State = 'Waiting_For_Command'

#link myLogic R1 myUI

generate LC5:Start_Program() to myLogic
```