# WIMP: Windowed Inventory Maintenance Program

This assignment will modify and extend LIMP. Almost all of the specified functionality of LIMP will be retained. In WIMP, the actions file has been eliminated in favor of interactions through the user interface that will be added.

Virtually all of the implementation for LIMP can be recycled into WIMP, provided the LIMP design is sufficiently good. The major extension is that WIMP will incorporate a graphical user interface (GUI) implemented via the Amulet library. It is required that all of WIMP's functionality be accessed via the GUI. WIMP will not use command-line arguments of any sort.

The only additions to the command set will involve the capability to search the database for records matching a user-specified SKU or item description. In the latter case, it is logically possible there may be more than one match, but the test data will never create that situation.

## Amulet GUI:

You must implement a GUI for this project. Extensive sample code showing how to use Amulet will be posted on the course website and discussed in class during the week of November 8. Failure to attend those discussions will probably have a severely adverse effect on your ability to complete this assignment (a bit late for that, of course).

You have considerable latitude in your interface design. A prototype is illustrated below. All the specified functionality must be provided (for full credit), but you may vary the design in a number of ways if you like. In particular, you may implement buttons for some obvious operations, like sorting and navigation. You should NOT implement the entire interface in that manner.

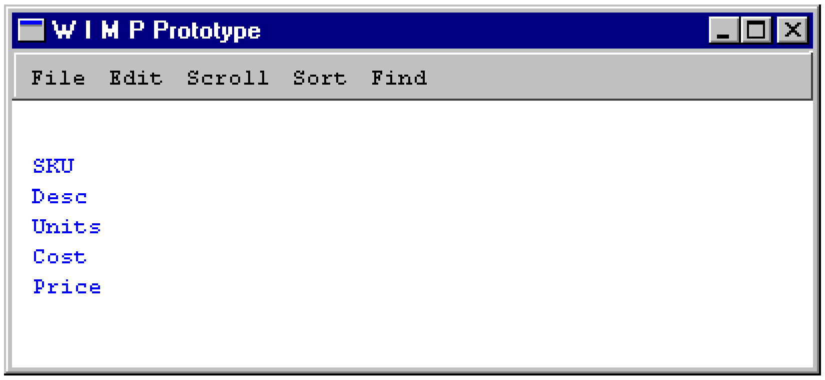
You must implement at least two menus, and at least three dialog interactions triggered by menu selections. The quality of your text labels in the interface is important.

## Prototype GUI:

Here's a possible main window for WIMP:

The client area shows labels for the display of a single inventory item at a time. Currently, no inventory file has been loaded, so there is no data displayed.

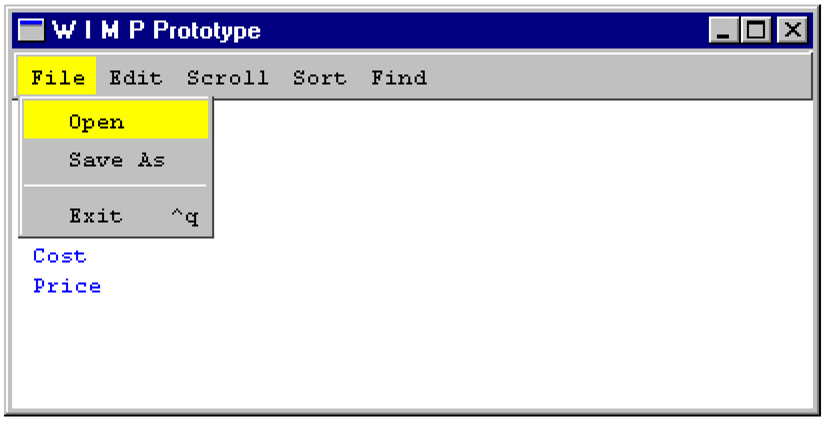
There are five menus, each listing related choices.



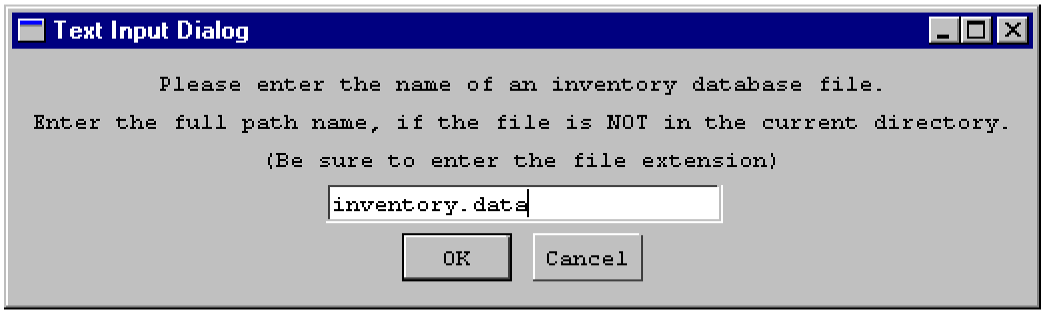
Here's the File menu:

There are choices for opening an existing file, saving the current database list to a file with a user-selected name, and exiting the program.

Note the "^q" on the Exit line. This is an accelerator (or keyboard shortcut). You may choose this menu item by holding down the Control key and pressing 'q'.



Selecting File/Open should raise a dialog box prompting the user to enter the name of the file, for example:

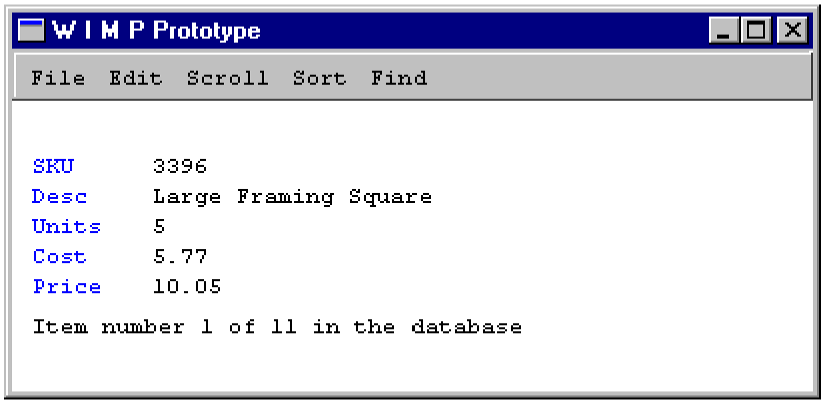


```
Please enter the name of an inventory database file.
Enter the full path name, if the file is NOT in the current directory.
(Be sure to enter the file extension)
inventory.data
```

This may be complicated if you didn't use the same format for your saved database file as I use for the initial inventory file. In that case, you'll need to have some way for the user to indicate whether you'll be loading an initial inventory file or a saved database file. (Hint: good time to repent if you used different formats.)

Once the user enters a file name and clicks OK, the file should be opened and read, and a new database list should be created, just as in LIMP. The window should be updated to display the first item in the database list:

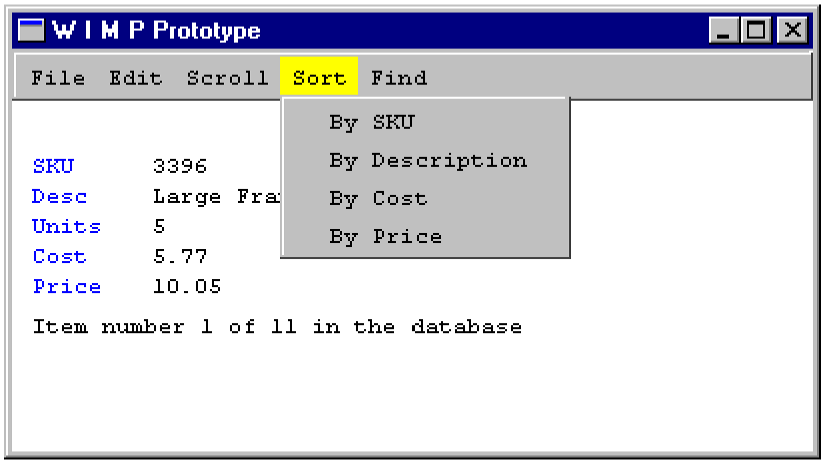
The window also displays a status line indicating which item is displayed (numerical order) and how many items are currently in the list. This is required, although you can be fancier with it if you like.



```
SKU      3396
Desc     Large Framing Square
Units    5
Cost     5.77
Price    10.05

Item number 1 of 11 in the database
```

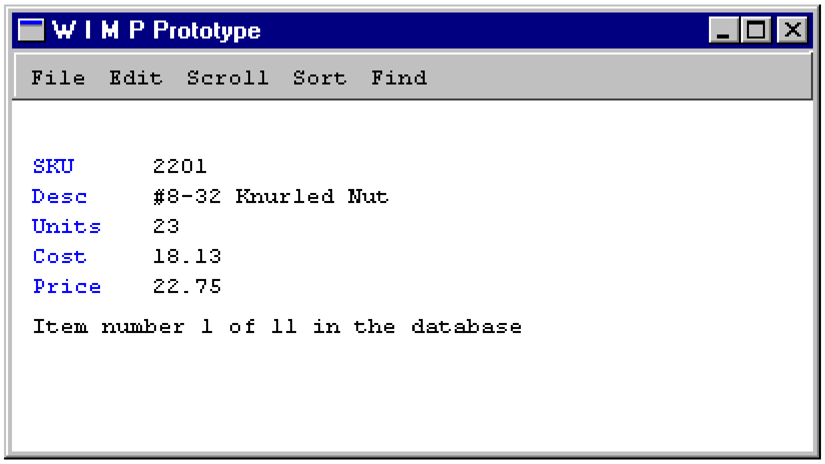
The Sort menu should provide choices for each field on which a sort is supported:



```
SKU      3396
Desc     Large Fra
Units    5
Cost     5.77
Price    10.05

Item number 1 of 11 in the database
```

Selecting one of the options should cause the database list to be sorted on that key, and the window updated to show the first element in the newly sorted list:



```
SKU      2201
Desc     #8-32 Knurled Nut
Units    23
Cost     18.13
Price    22.75

Item number 1 of 11 in the database
```

The Scroll menu should provide the ability to navigate the database list item-by-item in either direction and also to jump directly to either end, updating the window display accordingly. (This may be implemented as buttons if you want to learn more Amulet than is strictly required.)

The Edit menu should provide choices for Add Item, Delete Item, Buy and Sell. Choosing one of these should pop up a dialog box requesting the necessary information from the user, and then cause the relevant LIMP functions to be called to carry out the action. Once the action is carried out, the window should be updated in some sensible way (your choice as long as it makes sense).

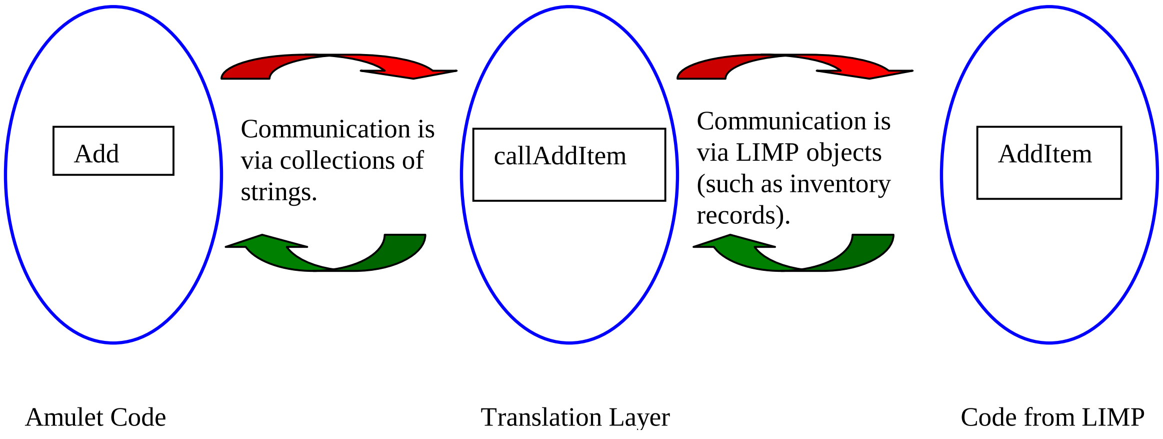
The Find menu should list choices for SKU and Description. Again, each should pop up a suitable dialog box and then call the appropriate functions to search the database list for a matching record. If a match is found, the window should be updated to display that item. If no match is found, a suitable error message should be displayed.

You should provide keyboard shortcuts for most, if not all, of the menu items. This is trivial in Amulet and will make it much easier to test your program. Warning: be very careful not to use the same shortcut for two different menu items.

There will not be extra credit for creating exceptionally convoluted (or elegant) GUIs.

## Integrating your GUI with LIMP:

First of all, read the short discussion of this on the website. I will discuss this further in class. The basic logical idea is that your program structure should be something like:


Add
Communication is via collections of strings.
callAddItem
Communication is via LIMP objects (such as inventory records).
AddItem
Amulet Code
Translation Layer
Code from LIMP

## Input file descriptions:

The initial inventory file has exactly the same syntax as for LIMP. The sample initial inventory files posted on the website for LIMP may be used to test WIMP.

## Output description and sample:

Output data resulting from a File/Save As command must be written to a file with the user-specified name. The format of this output is up to you, but the one used for the initial inventory file is a particularly good choice.

## Linked List:

As with LIMP: you are **required** to use a linked list to store the inventory information. Your implementation of this list will be examined. In order to receive full credit, you must implement the nodes using a well-designed node class and the linked list itself must be encapsulated using a well-designed list class. In addition, the inventory records must be encapsulated so that the data they store is logically separated from the node pointers.

Note this mimics the behavior of a C++ vector object. You are specifically forbidden to use any C++ vector objects in this program, or any other sort of predefined dynamic list type. You may base your implementation on the linked list implementations shown in the textbook, or those shown in the notes and lectures. You may not use linked list code from any other source. Violating that restriction would remove one of the major points of this assignment and will certainly result in a major deduction.

## Programming Standards:

You'll be expected to observe good programming/documentation standards. All the requirements for documentation and coding given in the LIMP specification are still in effect. In addition your user interface code should be documented as well as you can manage.

## Deliverables:

There will be no interim design document for WIMP, since the design is essentially the same as for LIMP, aside from the Amulet components. The final design document should be in the same format as for LIMP.

Your final project submission must include the following (and absolutely no other files):

- all source code (`*.cpp` and `*.h` files) comprising your project
- a final design document reflecting the final design of your project, at the time of submission, including the Amulet components; this must either be in a format that can be viewed in MS Word or be a PDF file.
- a brief ASCII text readme file, named `readme.txt`, with any special execution instructions
- either the MS Visual C++ `.dsp` and `.dsw` files, or a UNIX makefile, as appropriate

Submissions will be archived, but not scored, by the EAGS. You will submit your project as an archive file in one of two formats. For Windows users, submit a zipped archive containing the items listed above. (The program WinZip is very easy to use and is available from the University Computing Services website: `http://www.ucs.vt.edu/`) For UNIX users, submit a gzipped tar file containing the items listed above.

Note that omitting files from your archive is a classic error. Once you've created your project archive, copy the file to a new location, unzip it, attempt a build and then test the resulting executable. Submitting an incomplete copy of your project will delay evaluation and result in a loss of points.

## Evaluation:

`WIMP` will be subjected to runtime testing by the GTAs, who will also score your implementation for adherence to the specified programming standards. Your score will depend on the quality of your design documentation, the quality of your coding and internal documentation, and the correctness and completeness of the functionality of your program during runtime testing. The relative weights of these factors may be announced later.

Note that considerable weight WILL be given to the correctness of the functionality specified for LIMP (e.g., sorting, load and save). It is therefore important that you fix as many problems in your LIMP implementation as you can. Simply providing a graphical user interface that does not accomplish much of anything will result in a poor score.

Amulet makes it very easy to disable menu choices. In the event that an option does not work, you should disable the choice but leave it on the menu. An acceptable alternative is to pop up a dialog box indicating that the corresponding feature is not functional at this time. It is much better to acknowledge these shortcomings openly that to knowingly run the risk that your program will crash if we select the wrong action.

The input values we enter during testing will be syntactically correct. You should still watch for duplicate additions, deletions of nonexistent items, and so forth. However, you may safely assume we will enter numerical data where it's expected.

You will sign up for a demo of `WIMP`, as you did for `LIMP`.

## Testing:

At minimum, you should test your program on **all** the posted input/output examples given for `LIMP`. Note that you can use the posted actions files to test the correctness of your implementation by entering the transactions manually through your GUI.

## Submitting your project:

You will be allowed to make up to five submissions of `WIMP` to the EAGS. Instructions for submitting your project archive are available in the *Student Guide* at the EAGS Homepage:

`http://ei.cs.vt.edu/~eags/EAGS.html`

Read the instructions carefully. Note well: **your last submission will be graded.**

## Pledge:

Each of your project submissions to the EAGS must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment in the main source file:

```
//    On my honor:
//
//    - I have not discussed the C++ language code in my program with
//      anyone other than my instructor or the teaching assistants
//      assigned to this course.
//
//    - I have not used C++ language code obtained from another student,
//      or any other unauthorized source, either modified or unmodified.
//
//    - If any C++ language code or documentation used in my program
//      was obtained from another source, such as a text book or course
//      notes, that has been clearly noted with a proper citation in
//      the comments of my program.
//
```

**Failure to include this pledge in a submission is a violation of the Honor Code.**