# WARP: Windowed Academic Records Program

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

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

The only additions to the command set will involve the capability to edit a course record to process a grade change. In addition, the QCA and alternate QCA must now be recalculated whenever a grade is changed, or a course is added or dropped. The QCA values must also be written to your saved database files.

## Amulet GUI:

You must implement a GUI for this project. Extensive sample code showing how to use Amulet has been posted on the course website and discussed in class during the week of April 3. 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, as is the layout of data in the windows.

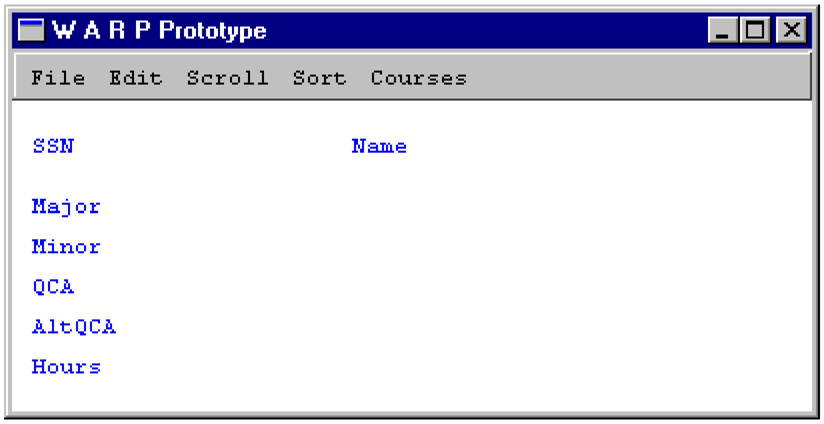
## Prototype GUI:

Here's a possible main window for WARP:

The client area shows labels for the display of information for a single student at a time. On startup, no student data has been loaded, so there is no data displayed.

The layout of the labels and accompanying data fields in the client area is up to you. All the fields shown here must be included.

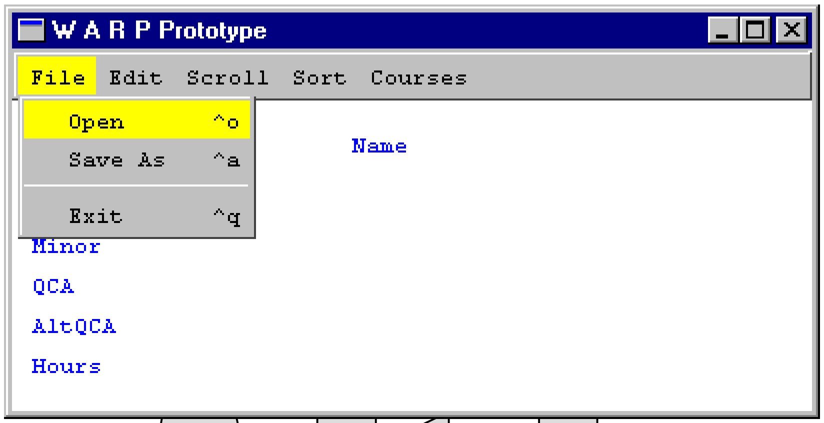
There are five menus in the main WARP window, each listing related choices.



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'.



Name

Minor

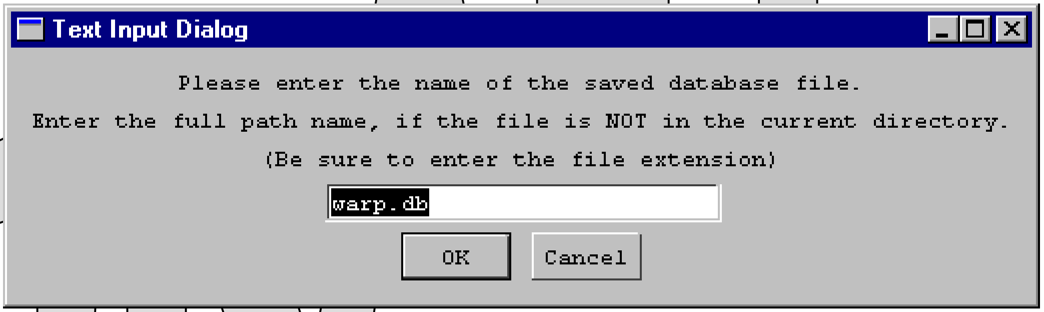
QCA

AltQCA

Hours

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

Note this corresponds to the Load command in LARP.



Please enter the name of the saved database file.

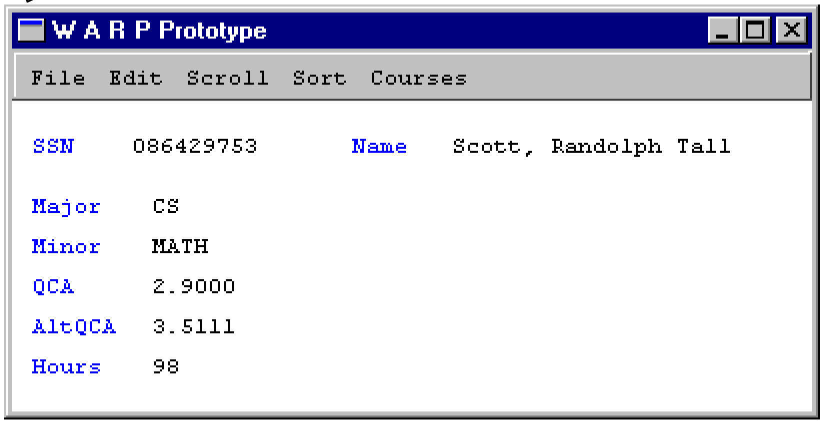
Enter the full path name, if the file is NOT in the current directory.

(Be sure to enter the file extension)

warp.db

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 LARP. The window should be updated to display the first item in the database list:

The fields should be clearly labeled, and the display should include all the information shown here. The precise layout is up to you.



SSN 086429753 Name Scott, Randolph Tall

Major CS

Minor MATH

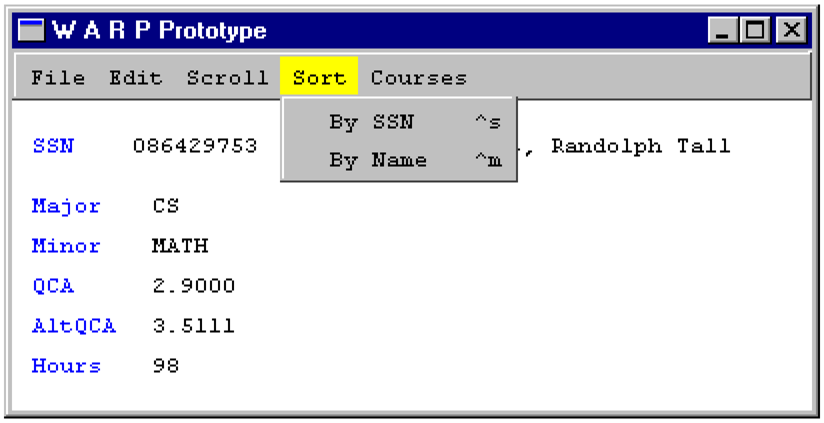
QCA 2.9000

AltQCA 3.5111

Hours 98

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

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



SSN 086429753 ., Randolph Tall

Major CS

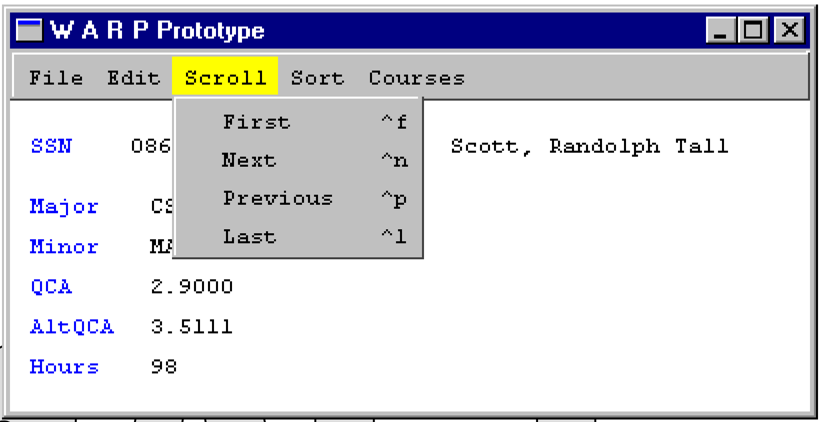
Minor MATH

QCA 2.9000

AltQCA 3.5111

Hours 98

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.)

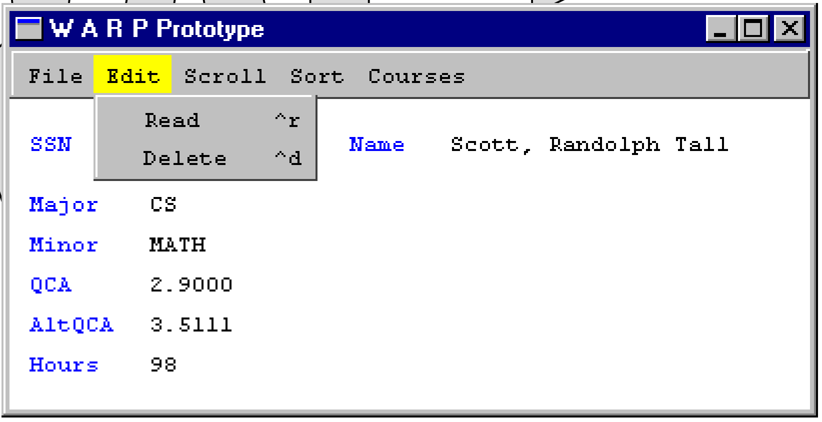


SSN 086 Scott, Randolph Tall
Major CS
Minor MA
QCA 2.9000
AltQCA 3.5111
Hours 98

The Edit menu should provide choices for Read and Delete. Choosing Read should pop up a dialog box requesting the name of a student information file (one student as in LARP), and then cause the relevant LARP functions to be called to carry out the action.

Choosing Delete should cause the current student record to be deleted from the database.

Once each action is carried out, the window should be updated in some sensible way (your choice as long as it makes sense).

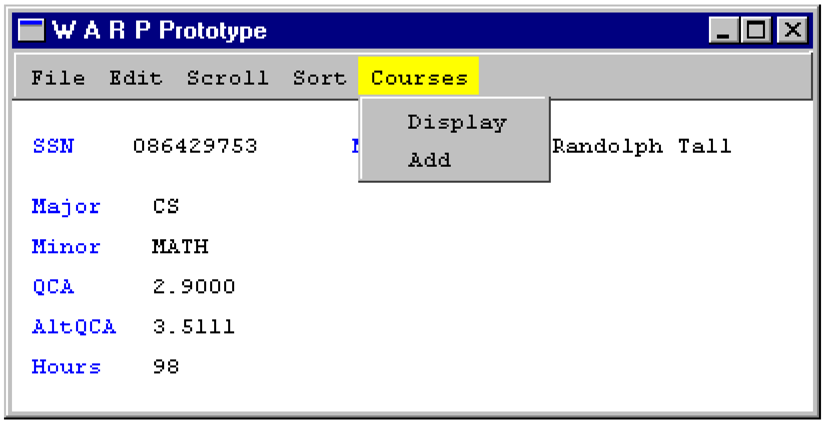


SSN Name Scott, Randolph Tall
Major CS
Minor MATH
QCA 2.9000
AltQCA 3.5111
Hours 98

The Courses menu should list choices for displaying the course list for the current student, and for adding a new course for the current student (dropping a course is handled elsewhere).

Choosing to display the course list should pop up a second window (modal) that will be described shortly.

Choosing to add a course should pop up a dialog to receive the relevant course data. Once the course data is entered and the user chooses OK, LARP functions should be invoked (via the translation layer of course) to create a course record and add it to the student's course list.



SSN 086429753 N Randolph Tall
Major CS
Minor MATH
QCA 2.9000
AltQCA 3.5111
Hours 98

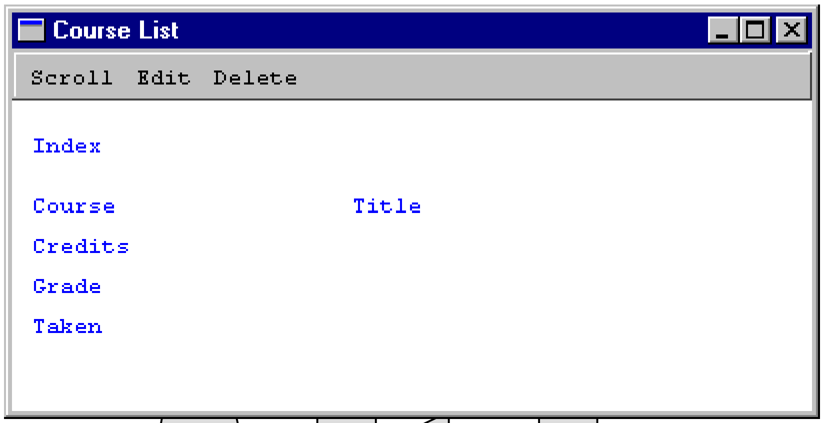
You should provide keyboard shortcuts for most, if not all, of the menu items. This is trivial in Amulet and will make it much easier to test your program. Warning: be very careful not to use the same shortcut for two different menu items.

We will not award extra credit for creating exceptionally convoluted (or elegant) GUIs.

The course list should be displayed in a separate window, with fields as shown.

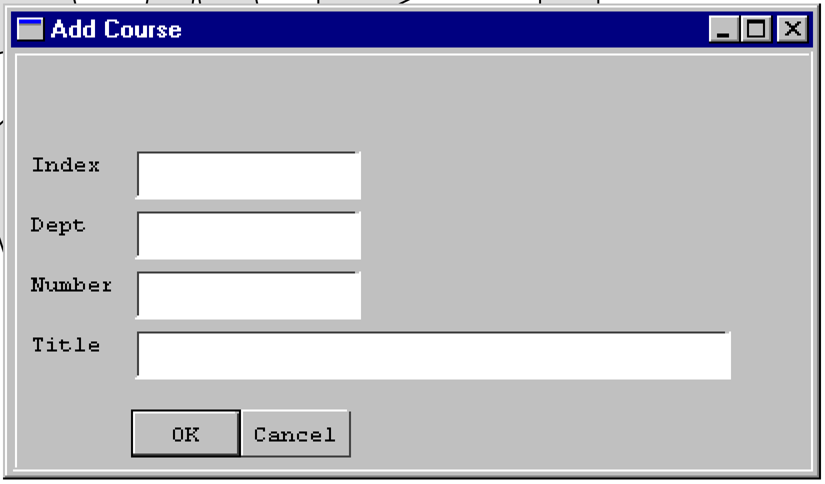
Layout is up to you.

The course list window should provide access to scrolling (menu list is obvious), deleting a course (only item on the Delete menu here), changing a course grade (only item under the Edit menu here), and sorting by either Index of Course, as in LARP (menu not shown here).



Choosing add course should pop up a dialog to collect the necessary information, somewhat as shown here.

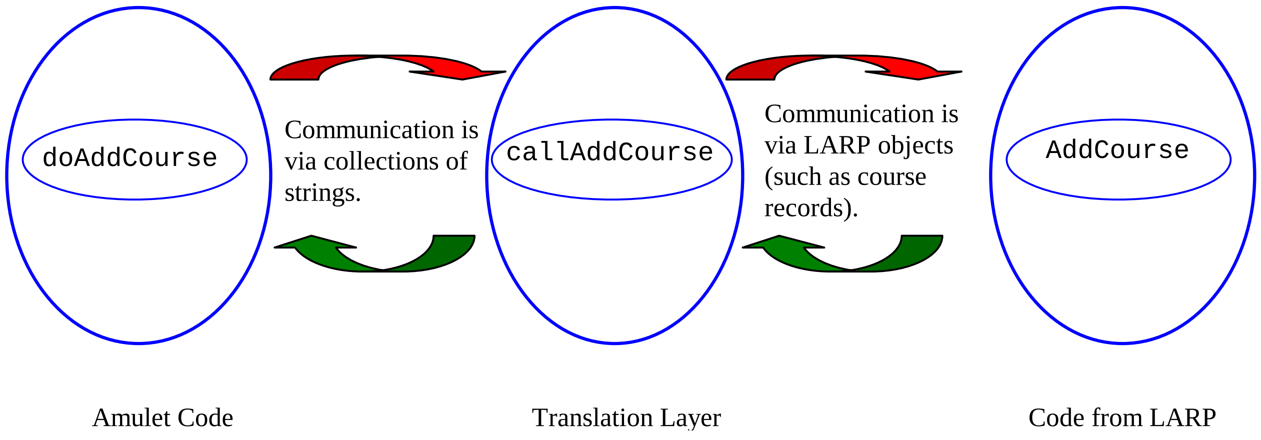
Unfortunately, Amulet does not provide a widget (built-in type) for this purpose, so you'll have to implement your own. This one is just a regular window, similar to the course list window above, except that the Am_Text fields have been replaced with Am_Text_Input_Widget objects. These allow the user to type in data (as in the dialog box) and you to read that data. There's an example in the source examples on the course website that uses an approach like this, and some additional helpful code will be posted (and probably discussed briefly in class).



Make sure you include ALL the necessary fields for adding a course (the example here does not).

## Integrating your GUI with LARP:

The correct (and required) organization of your program has been discussed in class. The basic logical idea is that your program structure should be something like:


doAddCourse
Communication is via collections of strings.
callAddCourse
Communication is via LARP objects (such as course records).
AddCourse
Amulet Code
Translation Layer
Code from LARP

## Input file descriptions:

The initial student data files have exactly the same syntax as for LARP. The sample files posted on the website for LARP may be used to test WARP, emulating the processing of the actions file via the graphical interface.

## 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, presumably you will use the same format as for the Save/Load commands in LARP. There is no longer a dump command.

## Linked List:

As with LARP: you are **required** to use a linked list to store the student 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 student and course data records must be encapsulated (as a class, or a struct) 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 LARP 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 WARP, since the design is essentially the same as for LARP, aside from the Amulet components. The final design document should be in the same format as for LARP, and should include the translation layer and the Amulet components (as well as possible).

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

As usual, you will submit your project to the Curator System 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 may delay its evaluation and **will** result in a substantial loss of points. In particular, if you omit a source file necessary to compile your program, you **will** be allowed to supply that file; however, we will then apply a late penalty corresponding to the date that you have provided a complete copy for evaluation.

**Also note** that including unnecessary files is also a classic error. Visual C++ users: do not zip up the **debug** subdirectory!

## Evaluation:

WARP will be subjected to runtime testing by the TAs, 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 LARP (e.g., adding and dropping courses, sorting, load and save). It is therefore important that you fix as many problems in your LARP 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 course additions, drops of nonexistent courses, 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 WARP, as you did for LARP. Un-demoed projects will not be graded. Demos must be completed by 5:00 on Thursday, May 4. Since the TAs will also be taking exams, some TAs may require that demos be completed earlier than that, so sign up early.

## Testing:

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

## Pledge:

Each of your project submissions to the Curator System 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.**