School Management And Reporting Tool

Due Tuesday, March 30 at 8 a.m.

CSE 1325 - Spring 2021 - Homework #7 / Sprint 3 - 1 - Rev 0

Assignment Background

The School Management And Reporting Tool (SMART) will assist administrators of elementary and secondary schools with keeping track of students and their parents, teachers, course subjects, sections, and grades, and the many relationships between them.

Your goal is to prove that you can implement the SMART (and possibly smart) specification and thus win the contract to build the full production version along with the associated fame and cash and possible spacecraft tickets for Mars on-site support.

This is sprint 3 of a 5-sprint, 5-week project that you can add to your growing resume, with emphasis on writing graphical user interfaces in gtkmm, file I/O, and polymorphism.

Your implementation is permitted to vary from any class diagram provided with the final project as needed without penalty, as long as you correctly implement both the letter of and the intent of the feature list. If you prefer to build a tool that addresses a different cultural approach to educating children, that would simply be delightful. **If you have questions about the acceptability of changes that you are considering, contact the professor first!**

Sprint 3 - File I/O and Toolbars

The primary goal for sprint 3 is to implement saving and loading of SMART data to and from the filesystem. In addition, you will add a toolbar and an About dialog.

Because mutual aggregation is very tricky to save and restore, *you may omit the vector attributes of Parent and Student* and save only their non-vector data, though you MUST save and open Mainwin's vector as well as scalar attributes. This in effect discards the Student - Parent relationships, but they may be added as another 25-point bonus for this sprint (see Bonus below).

A suggested class diagram for the third sprint is on page 2.

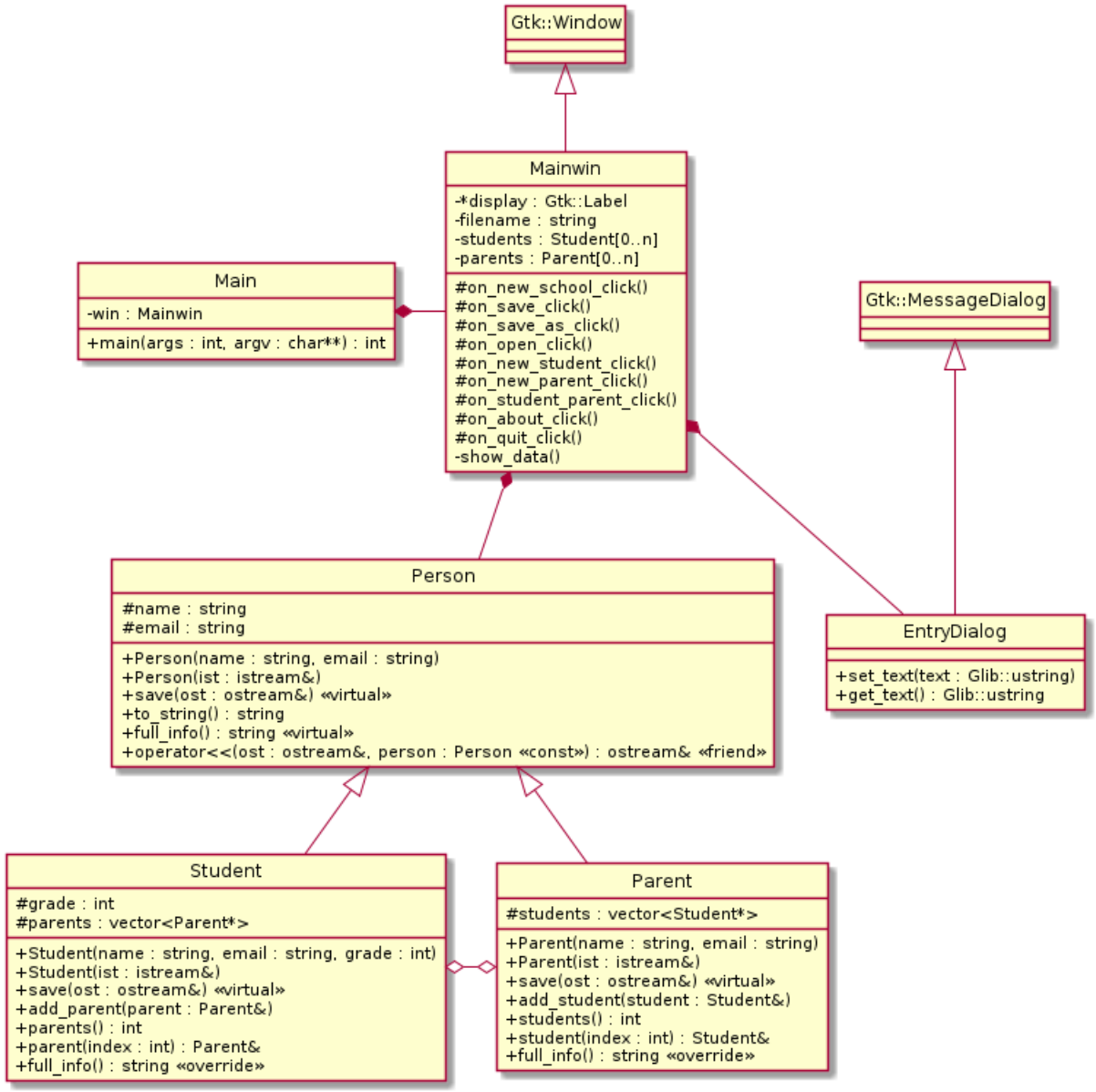
- For saving and loading data to each class, you should carefully consider the recommendations in Lecture 15 and the example at cse1325-prof/15/nim/nim.cpp!
- Good examples for the toolbar and About dialog may be found in cse1325-prof/15/nim/mainwin.cpp.

For this sprint, you **MAY** deviate from the class diagram as long as you fulfill the intent of the requirements, including all features allocated to this sprint in Scrum's Product Backlog. As an example, you may want an `int select_student()` method that pops up a dialog asking for the user to select a student and returning an index to the selected student.

For this sprint, you **MAY** adopt code from the sprint 2 suggested solution with the following caveats:

- Code from any class may be adapted under the terms of the Gnu General Public License 3 (GPL3), which requires attribution among other restrictions. The required About dialog (as in The Game of Nim) is a good way to fulfill these obligations. **Remember to observe these license restrictions if you publish your project code after May 30 as part of your resume.**

Class Overview



EntryDialog, Main

The main function and EntryDialog class should require no changes for this sprint.

Person, Student, Parent

As with Nim, we will handle saving with a save method (passing the ostream& as a parameter), and the open with a *constructor* (passing the istream& as a parameter). Write each attribute to a separate line in the save method, and read each attribute from a separate line in the constructor.

Mainwin

Construct the Mainwin::filename attribute as "untitled.smart" (or whatever default filename you like).

Add the File > Save, Save As, and Open menu items in the usual way.

Save

In Mainwin::on_save_click, you need to save the students and parents vectors.

- Open an output file stream using Mainwin::filename (see Lecture 15)
- Write the size of the students vector to the ostream *as a separate line*
- For each Student in the students vector (e.g., with a for-each loop), call the student's save method so it can write its own data to the stream.
- Write the size of the parents vector to the ostream *as a separate line*
- For each Parent in the parents vector, call the parents's save method so it can write its own data to the stream.

Be sure to use a try / catch construct in case of an error when writing the file!

Save As

Pop up a FileChooserDialog for saving a file (see Lecture 15), with the default filter set to SMART files. If the user clicks "Save", set Mainwin::filename to the filename specified in the dialog *and call Mainwin::on_save_click* (which already knows how to save the data to the file specified by Mainwin::filename). If the user clicks anything other than "Save", silently exit the method.

Open

Pop up a FileChooserDialog for opening a file (see Lecture 15), with the default filter set to SMART files. If the user clicks "Open":

- Set Mainwin::filename to the filename specified in the dialog.
- Open an input file stream using Mainwin::filename (see Lecture 15).
- Clear the current school info (consider on_new_school_click() followed by show_data()).
- Read the size of the students vector from the istream *as a separate line*
- For each Student in the students vector (e.g., with a while loop), instance a Student object from the istream and push it to the back of the students vector.
- Read the size of the parents vector from the istream *as a separate line*
- For each Parent in the parents vector, instance a Parent object from the istream and push it to the back of the parents vector.

If the user clicks anything other than "Open", silently exit the method.

Be sure to use a try / catch construct in case of an error when reading the file!

Toolbar

Create a toolbar just below the menu bar in the main window. Include the following buttons. You may use stock icons where one exists, or create all custom icons.

- New School (Gtk::Stock::NEW, calls on_new_school_click)
- Open School (Gtk::Stock::OPEN, calls on_open_click)
- Save School (Gtk::Stock::SAVE, calls on_save_click)
- Save School As (Gtk::Stock::SAVE_AS, calls on_save_as_click)
- A spacer is appropriate here
- Insert Student (custom icon, calls on_new_student_click)
- Insert Parent (custom icon, calls on_new_parent_click)
- Relate Student to Parent (custom icon, calls on_student_parent_click)

You may add other buttons or other widgets to the toolbar as you wish.

About Dialog

Add a Help > About menu item which calls on_about_click. A toolbar button is optional (About dialogs are not usually represented on a toolbar).

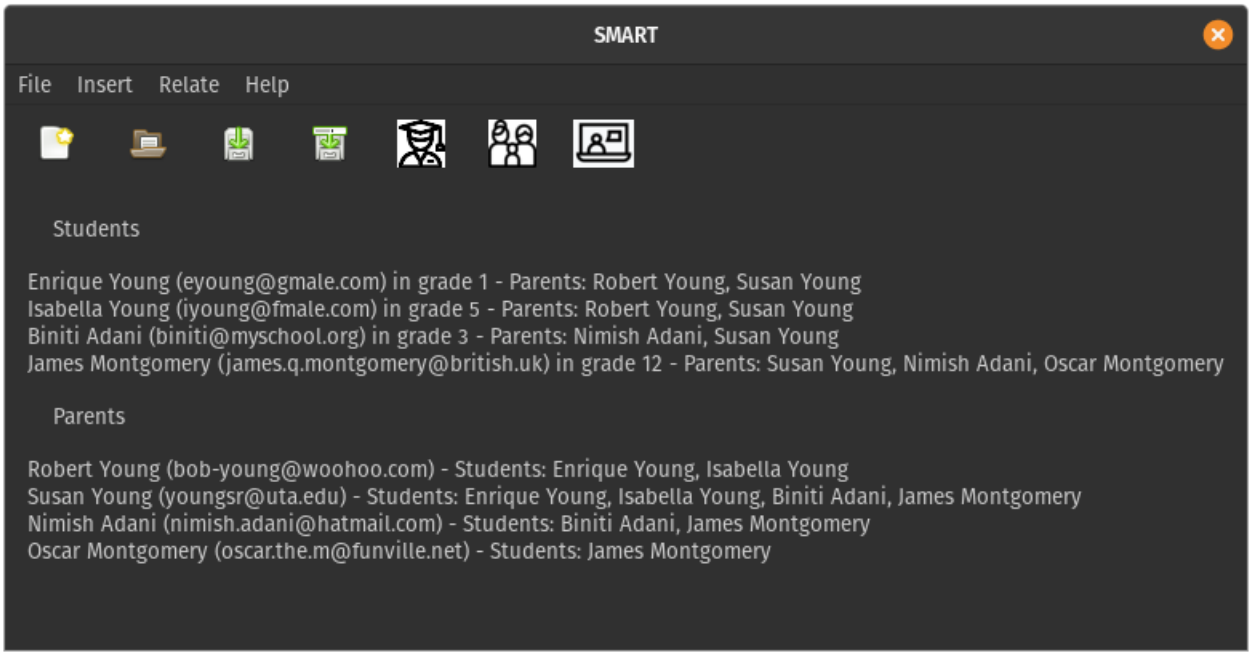
Create a Gtk::AboutDialog.

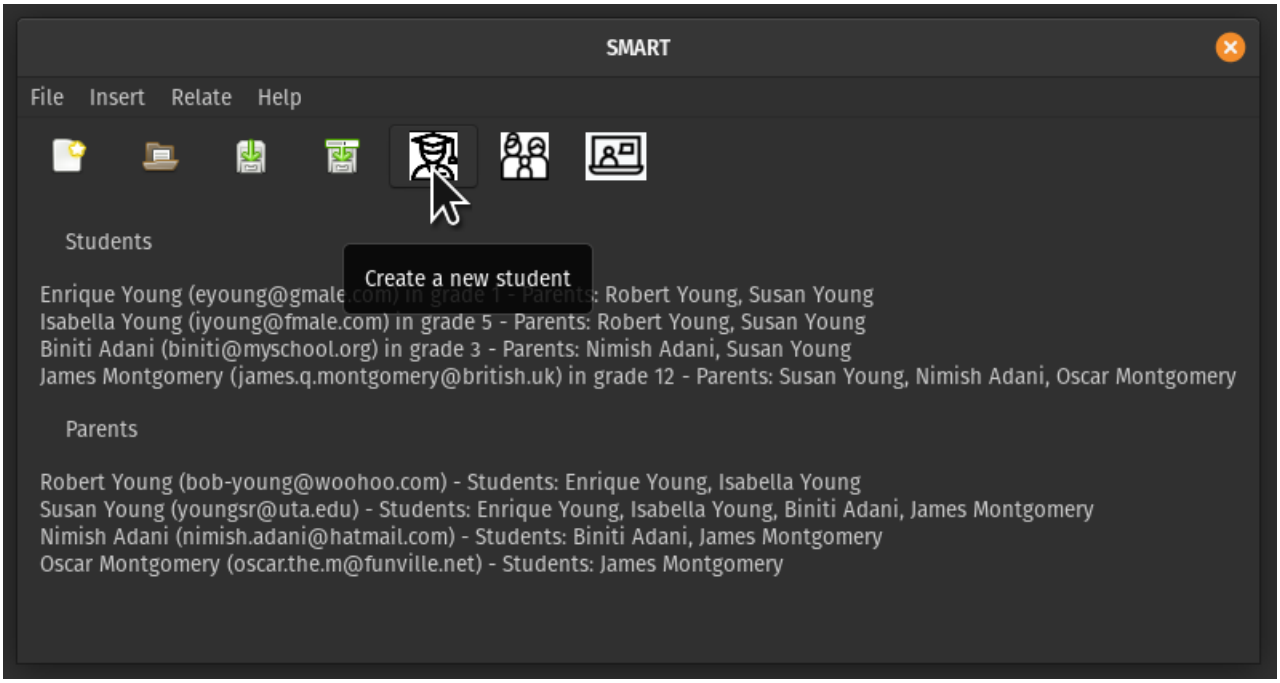
- Avoid the discouraging warning (that is, set a main window as parent of this dialog).
- Add a title via set_program_name.
- Add your version number via set_version.
- Add a custom logo via set_logo.
- Add a copyright notice via set_copyright. **This is required for any license.**
- Set a software license of your choosing (other than public domain).
- Set yourself as author using set_authors. **If you adapted code from a suggested solution where credit is required, add the professor as an author as well.**
- Give credit to any artists whose work you adapted for your logo or button icons, including a link to the original artwork, even if the license doesn't require it. (Your lawyers will thank you later.) Use set_artists. If you created your own artwork or substantially modified existing artwork under a permissive license, credit yourself here as well.
- Optionally add your website (set_website), comments (set_comments), or system information (set_system_information).
- Optionally credit documenters (set_documenters) and translators (set_translator_credits).

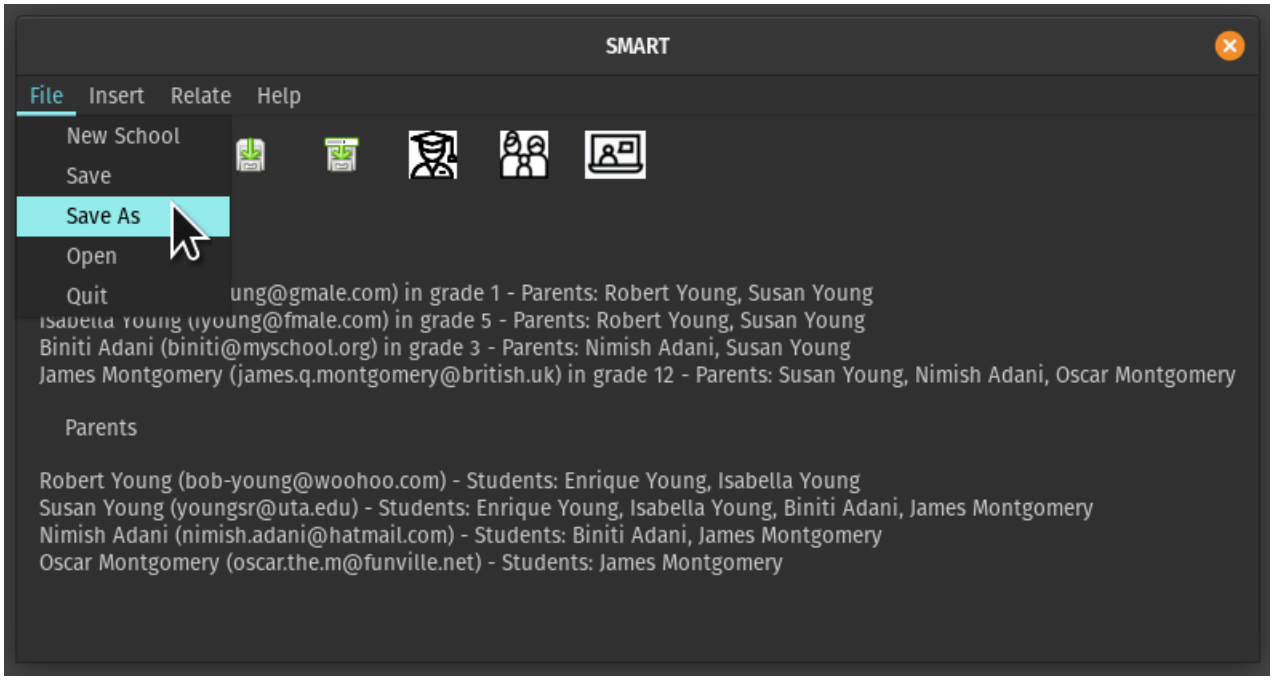
Makefile

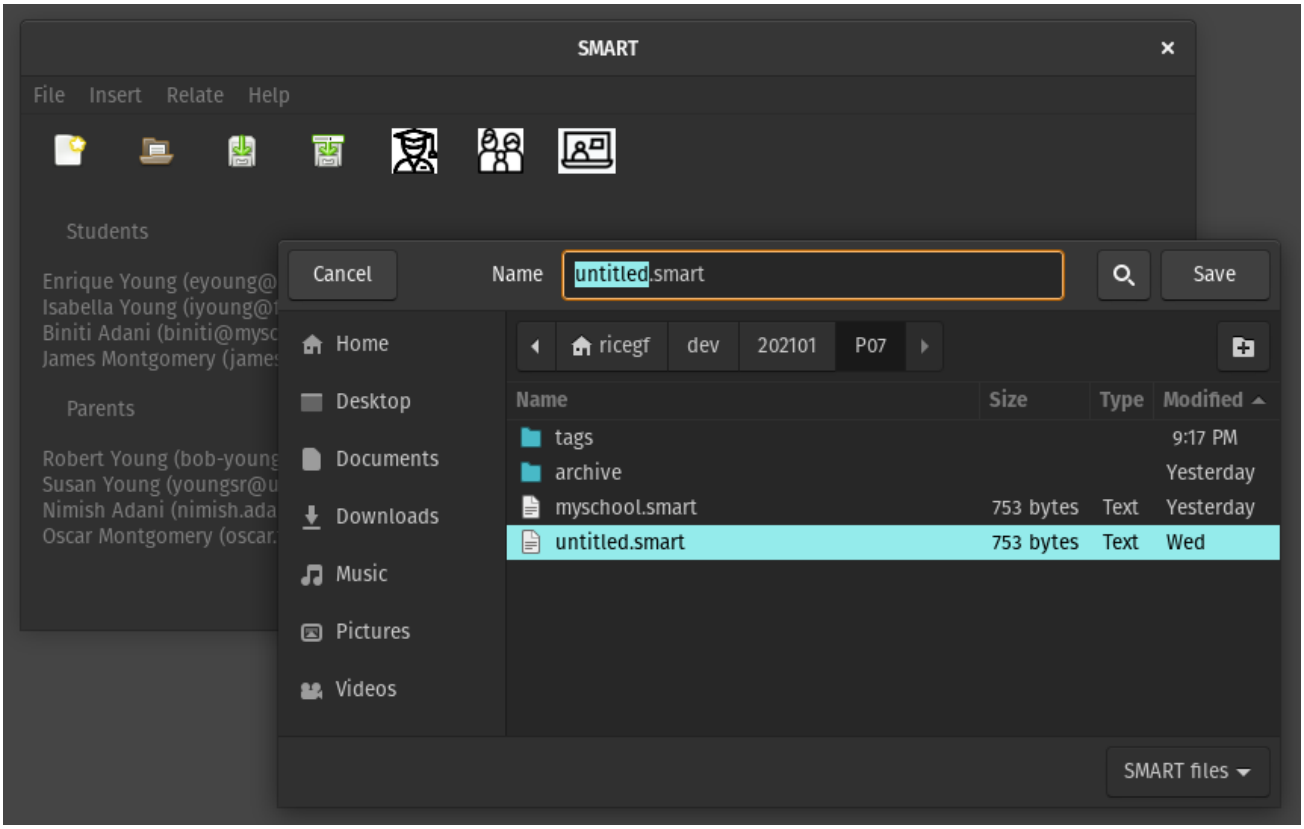
No changes are anticipated to the Makefile for this sprint.

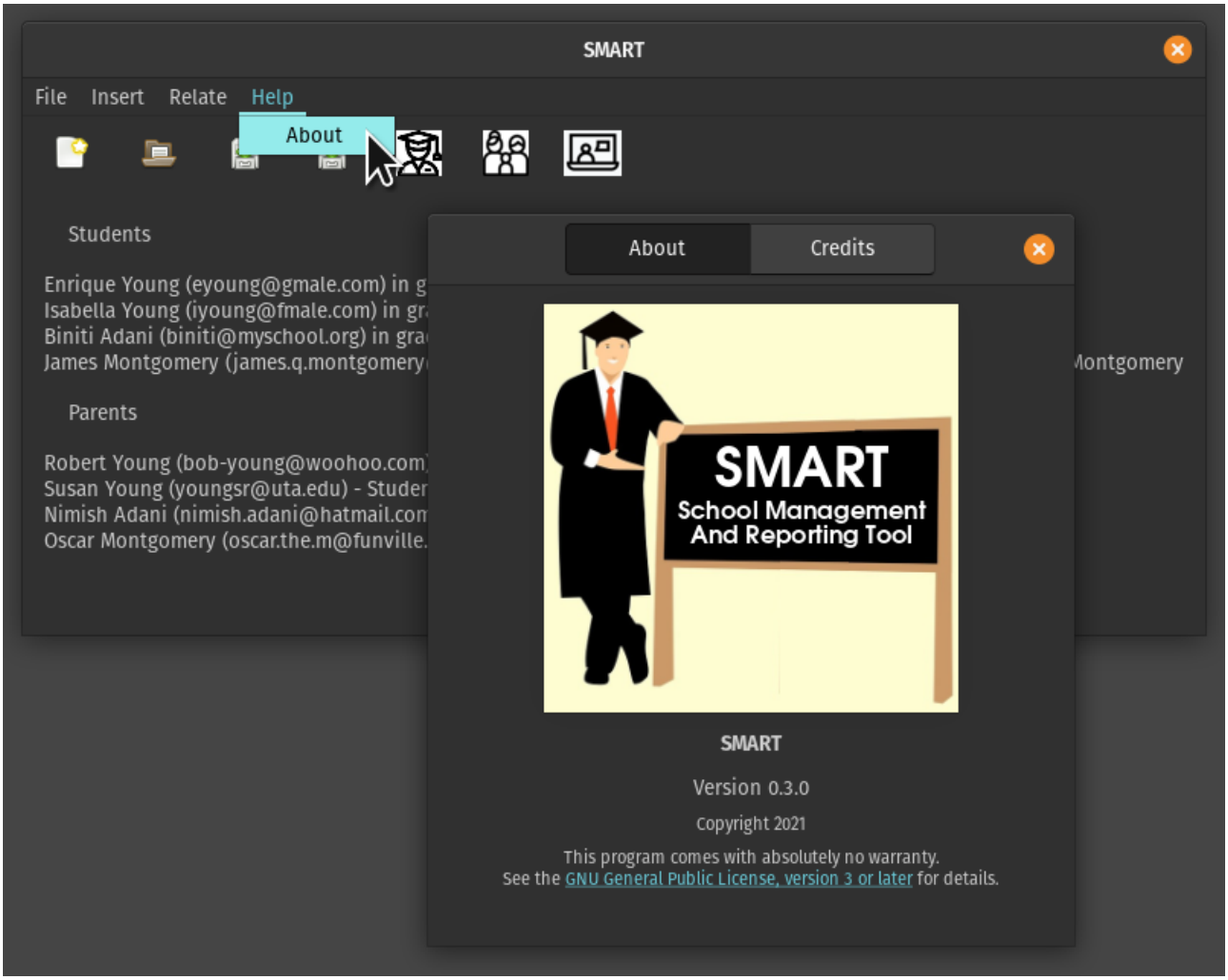
Here are some screenshots from the suggested solution. **Your application should look different than these,** since your logo and some icons should be custom. Your application may in fact look much *better* than these. Just ensure you meet all of the required features for this sprint!






SMART
File
Insert
Relate
Help
New School
Save
Save As
Open
Quit
ung@gmale.com) in grade 1 - Parents: Robert Young, Susan Young
Isabella Young (iyoung@fmale.com) in grade 5 - Parents: Robert Young, Susan Young
Biniti Adani (biniti@myschool.org) in grade 3 - Parents: Nimish Adani, Susan Young
James Montgomery (james.q.montgomery@british.uk) in grade 12 - Parents: Susan Young, Nimish Adani, Oscar Montgomery
Parents
Robert Young (bob-young@woohoo.com) - Students: Enrique Young, Isabella Young
Susan Young (youngsr@uta.edu) - Students: Enrique Young, Isabella Young, Biniti Adani, James Montgomery
Nimish Adani (nimish.adani@hatmail.com) - Students: Biniti Adani, James Montgomery
Oscar Montgomery (oscar.the.m@funville.net) - Students: James Montgomery


SMART
File
Insert
Relate
Help
Students
Parents
Cancel
Name
untitled.smart
Save
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
ricegf
dev
202101
P07
Name
Size
Type
Modified
tags
9:17 PM
archive
Yesterday
myschool.smart
753 bytes
Text
Yesterday
untitled.smart
753 bytes
Text
Wed
SMART files



If you have questions about any part of this project, contact me via email or Teams. Ask questions. I **love** questions!

Bonus

This is more of an *extreme* bonus, but it's worth 25 points if you can implement it well.

The challenge with saving class aggregations is that aggregations are (remember?) pointers or references in C++. Mutual aggregation is even harder - a Student must point to their Parent, but the Parent must also point back to the Student. But they can't be instanced at the same time.

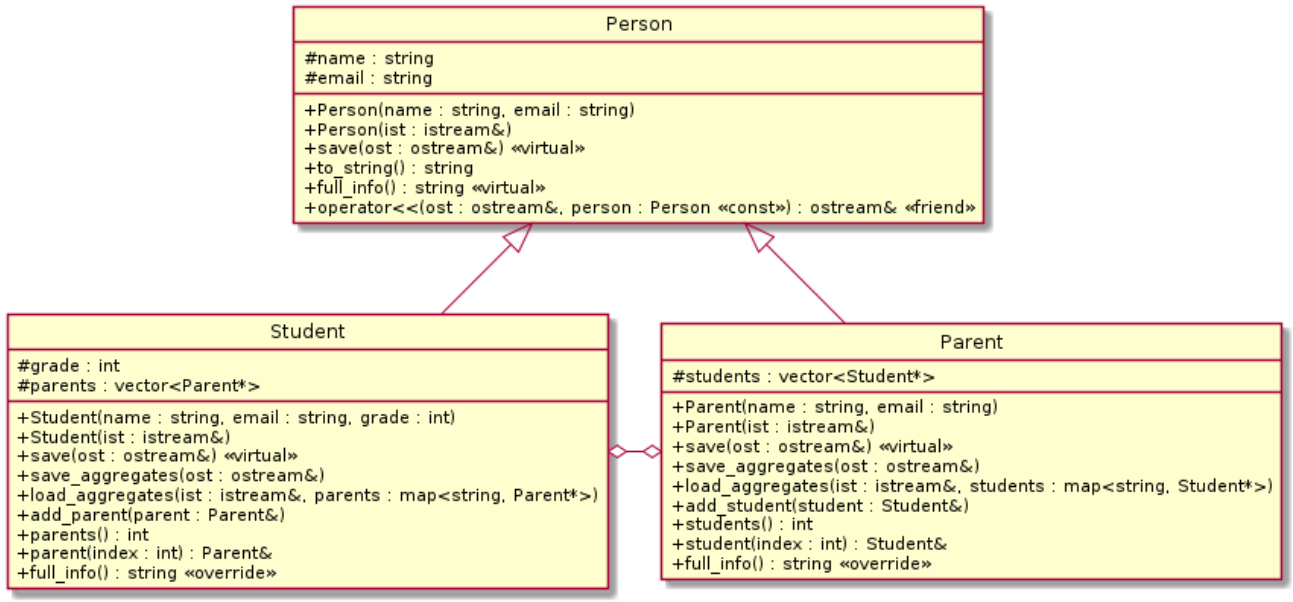
Saving the memory address (the literal pointer) from the aggregate vectors to the file won't work, of course, because the loaded data won't be at the same address.

So which comes first, the Students or the Parents? Neither. The *composites* must all come before the *aggregates*!

We must load all of the composite objects from the disk file first, filling Mainwin::students and Mainwin::parents. Only then can we add the aggregates from the disk file - Student::parents and Parent::students - by providing a way for each Student and Parent object to recreate its pointers to the (new) Parent and Student objects in Mainwin that each aggregates.

We handled loading the composites in the non-bonus portion of this sprint. If you'd like bonus credit, try to handle the latter.

Here's the approach from the suggested solution, along with a partial updated class diagram.



In addition to the istream constructor and ostream save method, add a save_aggregates and load_aggregates method to class Parent and class Student.

- Parent::save_aggregates should write the size of Parent::students to the ostream *on a separate line*, then iterate over that vector, writing a unique data item on distinct lines for each entry. Streaming out the Student object itself should stream out the name of each student; or you can add a getter for the email address (arguably a more unique key); or even add a unique ID to each Person for this purpose.
- Parent::load_aggregates accepts as a second parameter a std::map, associating the key written above with a pointer to the object *after it has been created by Parent::Parent(std::istream&)*. This is the "Rosetta stone" that allows recreation of the vector of pointers to the student(s) of each parent.

Do the same for class Student.

Now, update Mainwin::on_save_click to add a second phase right after each student's and parent's save method has been called. In the second pass, call each student's and each parent's save_aggregates method, thus writing out a unique key for each element pointed to by their vector elements to the ostream.

Finally, the tricky part. In Mainwin::on_open_click, right after loading the composite students and composite parents, we need to load their aggregates.

- For the Parent objects, first create the std::map<std::string, Student*> map, and populate it by iterating over the newly created students vector. For each element, store the key you selected above for that Student with its newly created address in memory. Then iterate over the parents vector, calling load_aggregates for each, which will use your map to recreate the aggregates. That is, load_aggregates will read each key from the istream, and push the associated pointer from the map onto its students vector.
- Do the same for the Student objects, creating a map and passing it to each student's load_aggregates method to recreate the parent aggregates.

Easy? No. But it works. Feel free to implement a simpler algorithm if you can identify one, though.