

School Management And Reporting Tool

Due Tuesday, April 20 at 8 a.m.

CSE 1325 - Spring 2021 - Homework #8 / Sprint 5 - 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 5 of a 5-sprint, 5-week project that you can add to your growing resume, with emphasis on writing a template and using iterators and (of course) wrapping up the project.

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 5 - Templates, Classes, and Iterators (Oh, My!)

The primary goal for sprint 5 is to add Teachers into the mix (why are we always *last* ■), add a couple of additional classes, and then add a template that creates an auto-populated drop-down list for your more polished unified dialogs with a single command.

A suggested class diagram (in three parts) for the fifth sprint is on pages 2 and 3. The diagram is split into multiple diagrams for easier reading, a very common approach as projects grow in size.

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.

For this sprint, you **MAY** adopt code from the sprint 4 suggested solution with the following caveats:

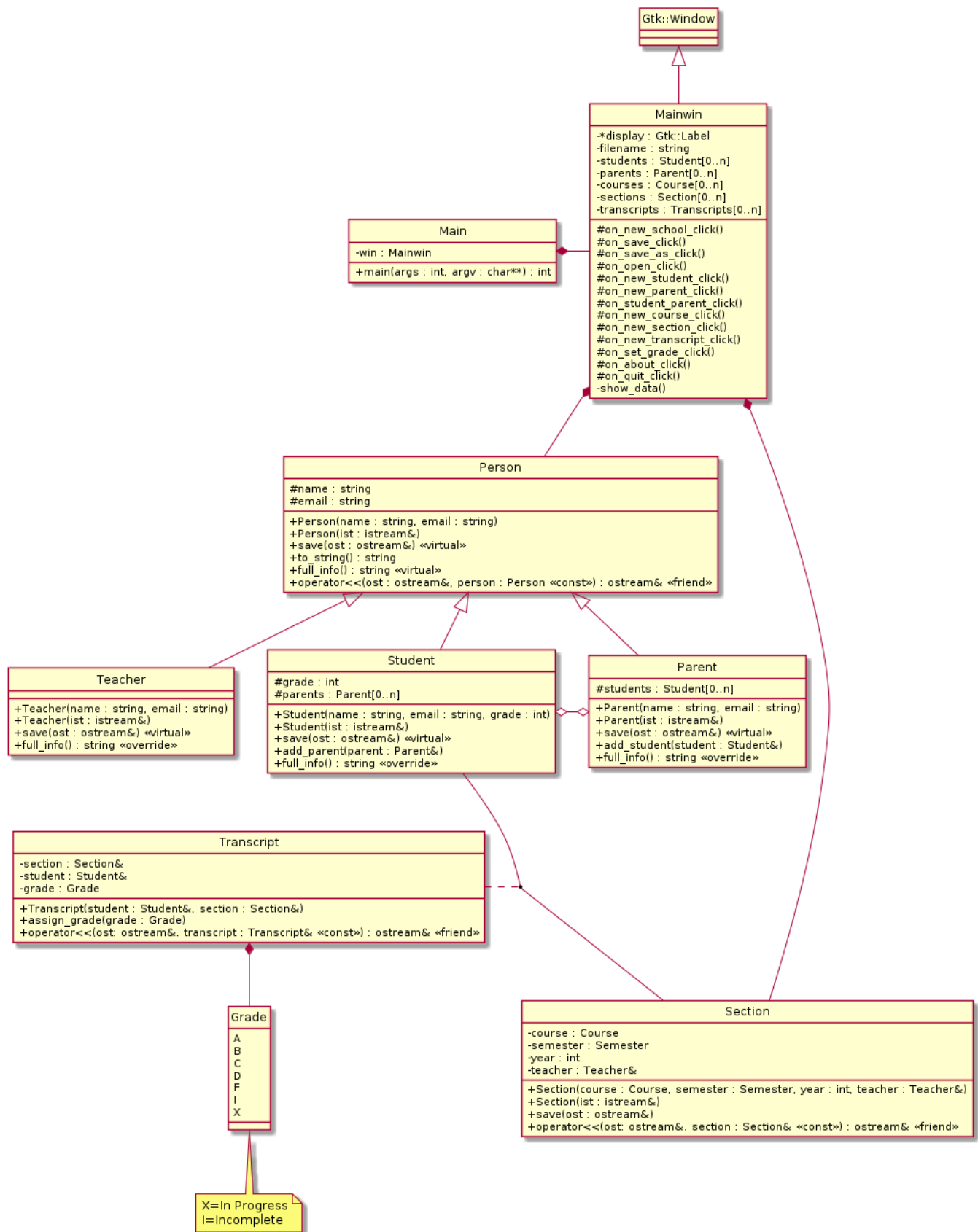
- Code from any class may be adapted under the terms of the Gnu General Public License 3 (GPL3), which requires attribution among other restrictions. The About dialog added in Sprint 3 is a good way to fulfill these obligations. **Remember to observe these license restrictions if you publish your project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

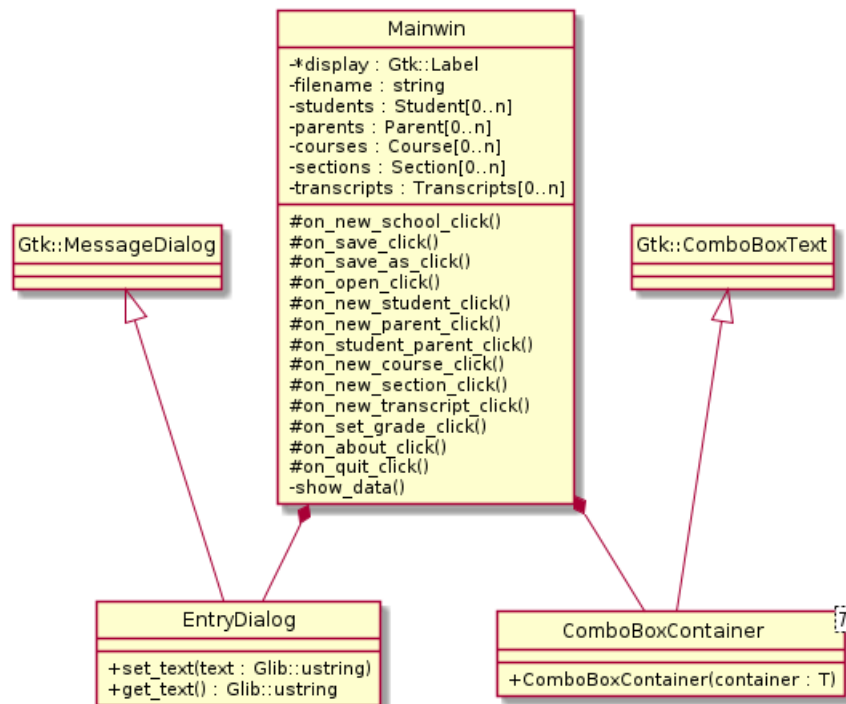
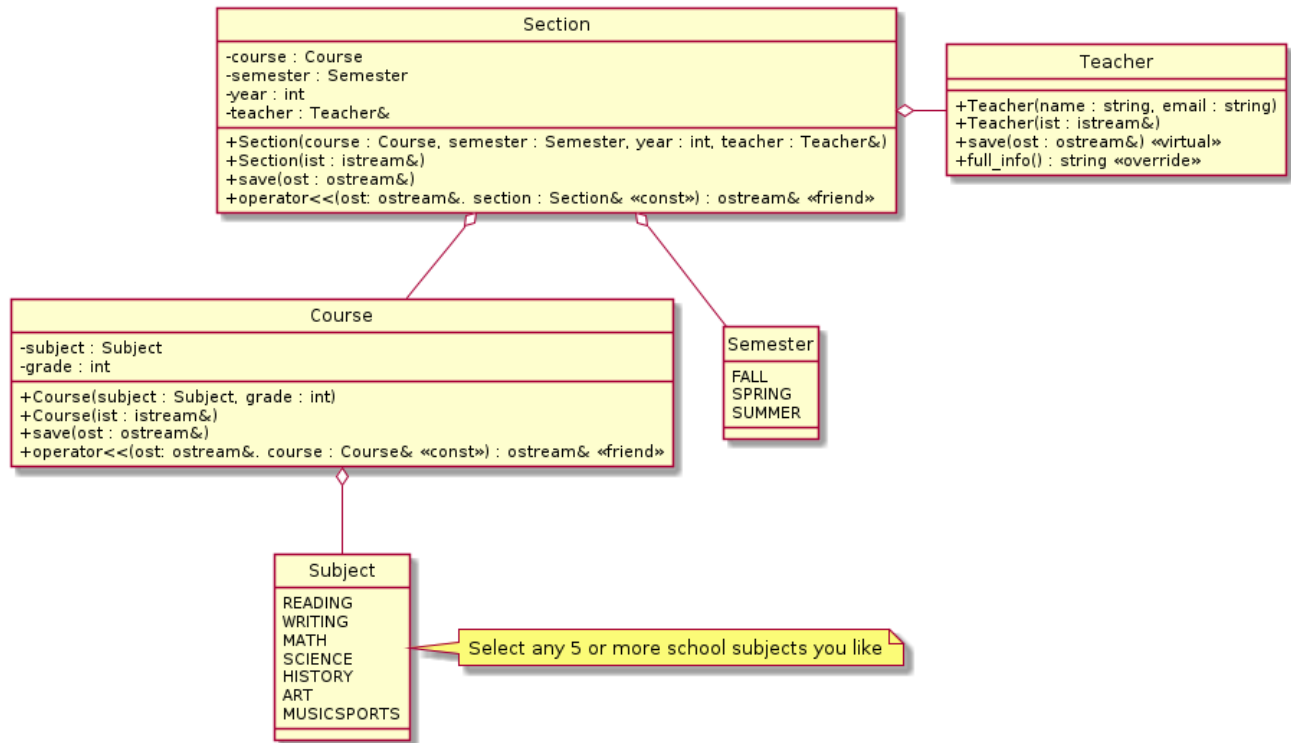
If you seek bonus points, they must be completed THIS SPRINT! On April 20, the project is OVER.

Class Overview

EntryDialog, Main, Person

These classes should require no changes for this sprint.





Student and Parent

This one is quick and easy. **Modify Student so that iterating over a variable of type Student yields that student's parents. Do the same for Parent, yielding their students.**

Here's more detail. We originally provided access to a Student's parents (that is, the elements in the `_parents` vector) with two methods: `int parents()` to provide the number of parents in the vector, and `Parent& parent(int index)` to provide a reference to each individual parent. So you could print out a student's parents like this:

```
Student student = students[0];
std::cout << student << "'s parents are ";
for(int i=0; i<student.parents(); ++i)
    std::cout << student.parent(i) << ' ';
```

That's the beginner approach - but now you know *iterators*. So *remove* those two methods, and instead delegate Student's iterators to `Student::_parents'` iterators directly.

Doing this requires adding the 4 "magic lines" to `student.h` (add nothing to `student.cpp`). You'll find the pattern in the Lecture 23 slide titled "Defining an Order Book: A Nested Container Example" - you simply typedef iterator and `const_iterator` to match the type of the `_parents` vector's iterator exactly, and then delegate `_parents.begin()` and `_parents.end()` to `Student::begin()` and `Student::end()`, respectively.

Given this, we can do something much more C++ like:

```
Student student = students[0];
std::cout << student << "'s parents are ";
for(Parent* p : student) std::cout << *p << ' ';
```

ComboBoxContainer

As a first step toward unified dialogs, and also to practice writing a template, **derive a new widget named `ComboBoxContainer` from `Gtk::ComboBoxText`**. This will be very similar to our `EntryDialog` class, which we derived from `Gtk::MessageDialog` in Lecture 14. (It works with many, but not every, STL container.)

For `ComboBoxContainer`, *make it a template* (covered in Lecture 21). Assume the template type is a vector of pointers to any of our classes here - `Student`, `Parent`, `Teacher`, `Course`, `Section`, `Transcript` - for which `operator<<` is defined. (This happily matches the 6 vector attributes of `Mainwin` by the time you're done with Sprint 5 - students, parents, teachers, courses, sections, and transcripts.) Only one member need be defined, the constructor, which will accept as a parameter a vector of pointers such as `Mainwin::students` or `Mainwin::parents`.

In the constructor, iterate over the vector provided as a parameter, using an output string stream to convert each object from the vector into a string. Append each resulting string to the `ComboBoxText` widget base class using its `append` method. So, basically, `ComboBoxContainer` is a self-populating combo box - pass a vector of pointers in the constructor, and the vector elements are listed in the drop down automatically!

This template will make creating a drop-down list of students, parents, teachers, courses, sections, or transcripts to add to your custom dialogs quite trivial. As a drop-down list of students, for example, you would only need this:

```
ComboBoxContainer<std::vector<Student*>> cbt_students(students);
dialog->get_vbox()->pack_start(cbt_students);
```

That's it!

To put our new widget to work, **improve `Mainwin::on_student_parent_click()` by replacing the sequence of text menu dialogs** for selecting a student and parent to relate ****** with a single custom dialog*, with a student and a parent drop-down list and "Relate" and "Cancel" buttons. See the example dialog below. Feel free to use this template for other custom dialogs as part of your bonus work for Sprint 5.

Teacher

Back to the more mundane. Write the Teacher class, which has no attributes beyond a Person. Enable the user to create Teacher objects, store them in a vector of pointers in Mainwin, and provide a way to view all Teacher object in the main window. Ensure they are saved and opened with the rest of the data.

Section

Each section will now have a specified teacher. Add the attribute and a constructor parameter to `Section::Section` to construct it. Save it, open it, and add it to the listing for each Section in the main window display.

Grade

Write the Grade enum class and similar utility functions (NOT methods - an enum class *cannot* have methods in C++). This is needed for the Transcript.

Transcript

Transcript is an *association class* (see Lecture 7) - that is, it associates two classes, in this case, a Student with a specific Section of a course. It also carries the Grade the student earned for the course (an 'X' means that section is still in progress).

You'll add a `Mainwin::transcripts` vector in which to store all Transcript instances. Enable the user to create Transcript objects, store them in a vector of pointers in Mainwin, and provide a way to view all Transcript objects in the main window. Ensure they are saved and opened with the rest of the data.

Mainwin

Mainwin adds teachers and transcripts attributes, most likely vectors, and `on_new_teacher_click()`, `on_new_transcript_click()`, and `on_set_grade_click()` methods in which the data is obtained from the user. (`on_set_grade_click()` let's the user specify a transcript and a new grade, and uses that transcript's `set_grade(Grade grade)` method to change it.) Make creating these objects available from both the menu bar and the toolbar (3 more icons!). If a data type (now we have 6!) is successfully created, it would be convenient to switch the view to display all of that data type. It's fine to use `EntryDialog` for creating the new objects in the callbacks, but of course unified dialogs with drop-downs (consider your new `ComboBoxContainer` template - quick and easy!) and spinners look nicer.

Extend your view mechanism to display teachers or transcripts in addition to students, parents, sections or courses in the main window display area.

Also update `Mainwin::on_save_click` and `Mainwin::on_open_click` to handle the new vectors, of course. Since we're adding new data types to the file format, here's another (optional) opportunity to provide backward compatibility with the older format. Gets easier with practice, no?

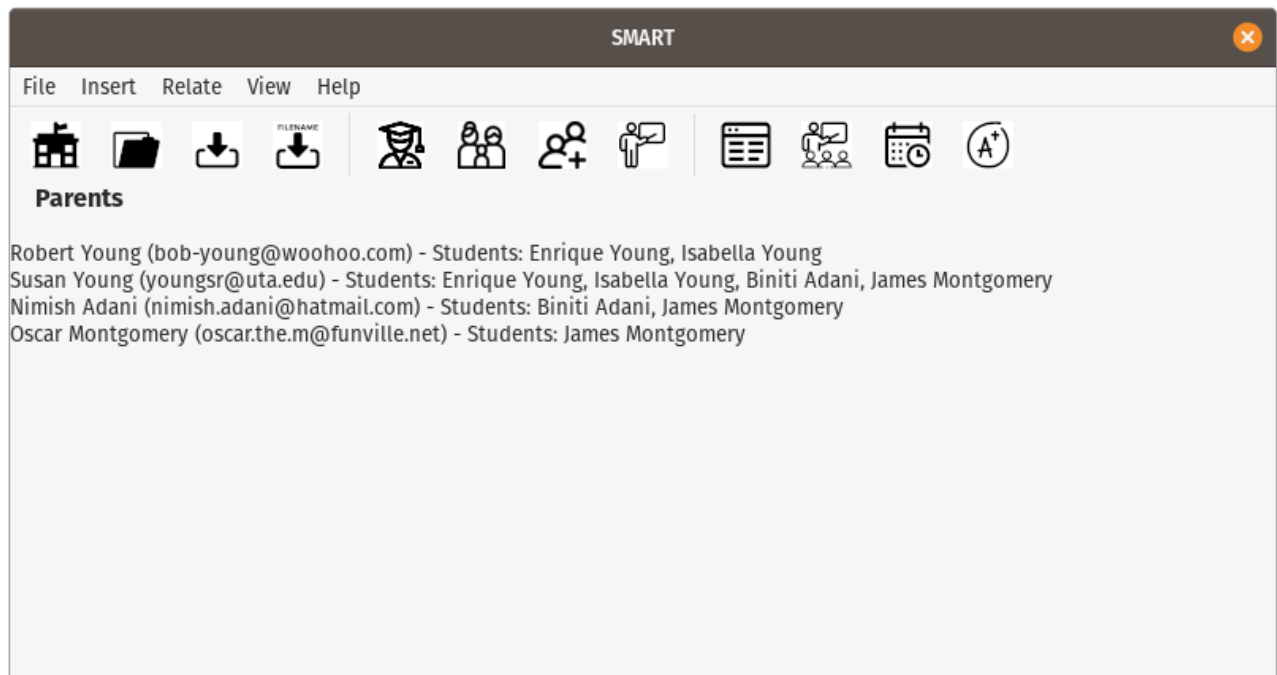
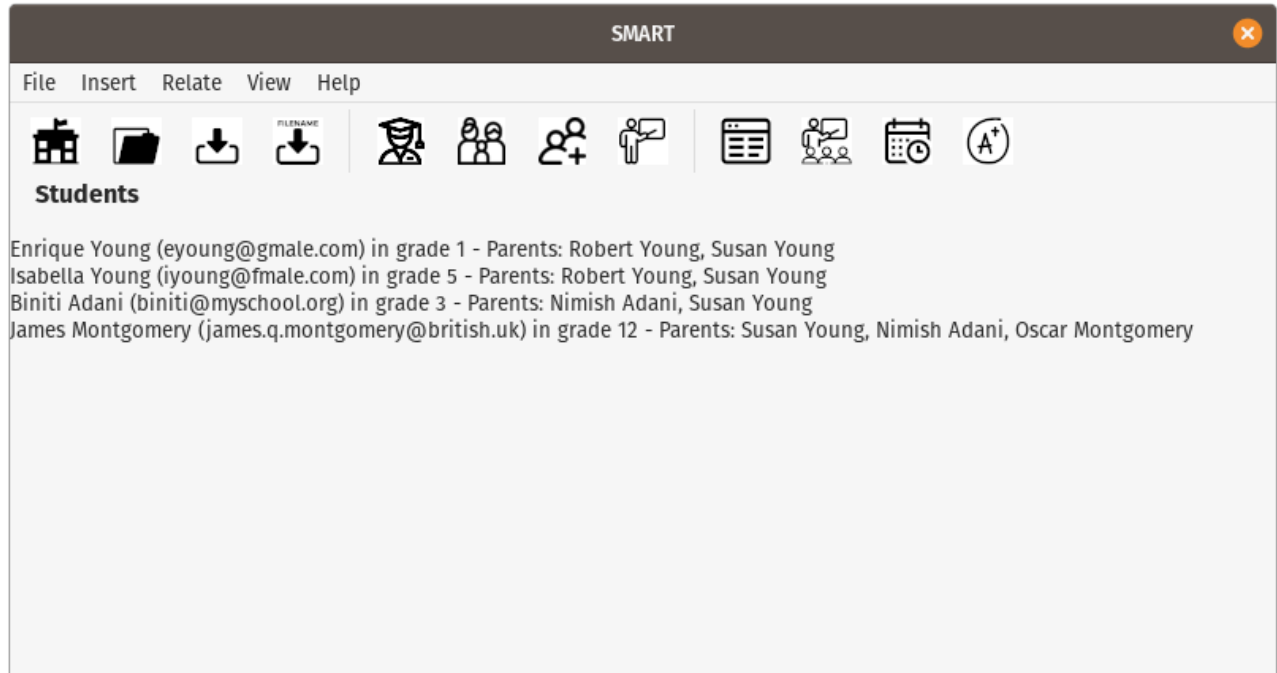
Makefile

You will need to accommodate building the 2 new classes and 1 new enum class this week. Since the template is only in a header file, it is not represented in the Makefile at all.

Screenshots





Below 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!

Here are the six views, selected (in the suggested solution - *your implementation may vary*) from the View menu.











SMART

FileInsertRelateViewHelp



FILENAME











Teachers





Elizabeth Crabtree (crabtree@hallsferry.edu)
Saint Bingham (bingham@hallsferry.edu)
Kermit Harness (kharness@vwsd.org)

SMART

FileInsertRelateViewHelp







Courses

reading (grade 1)

writing (grade 2)

math (grade 3)

science (grade 1)

history (grade 2)

art (grade 3)

music (grade 1)

sports (grade 2)

reading (grade 3)

writing (grade 1)

math (grade 2)

science (grade 3)

history (grade 1)

art (grade 2)

music (grade 3)

sports (grade 1)

reading (grade 2)

writing (grade 3)

math (grade 1)

science (grade 2)

history (grade 3)

art (grade 1)

music (grade 2)

sports (grade 3)

reading (grade 1)

writing (grade 2)

math (grade 3)

science (grade 1)

history (grade 2)

art (grade 3)

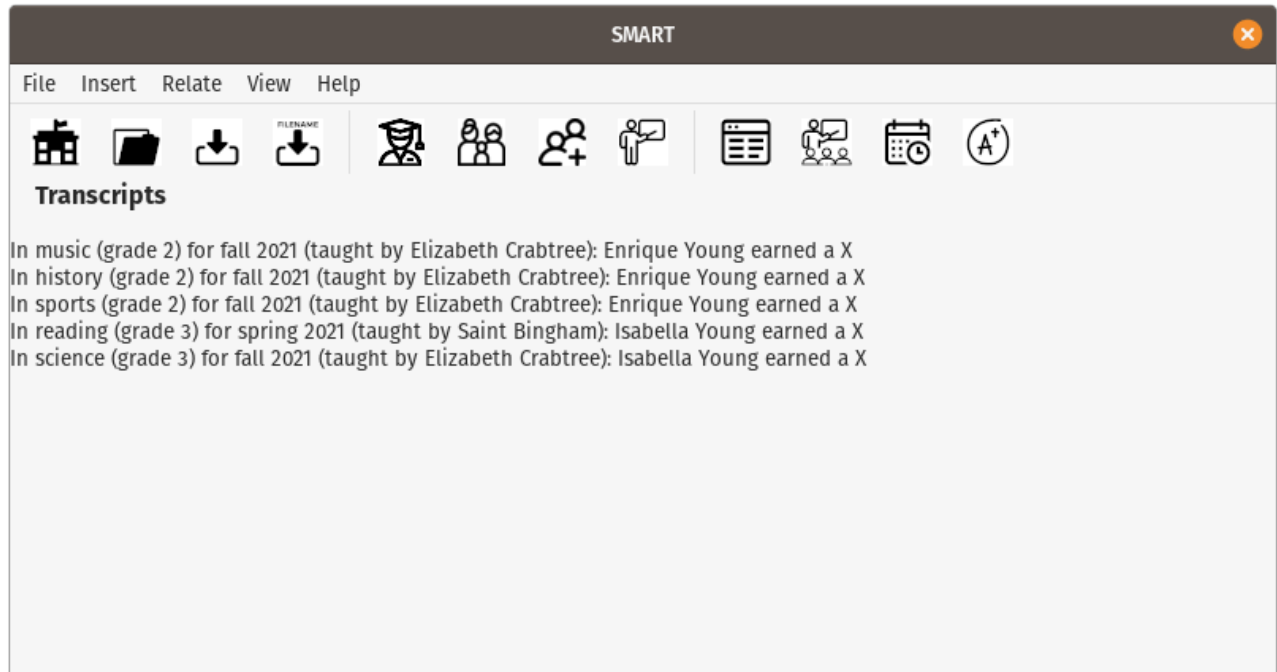
music (grade 1)

sports (grade 2)



Sections

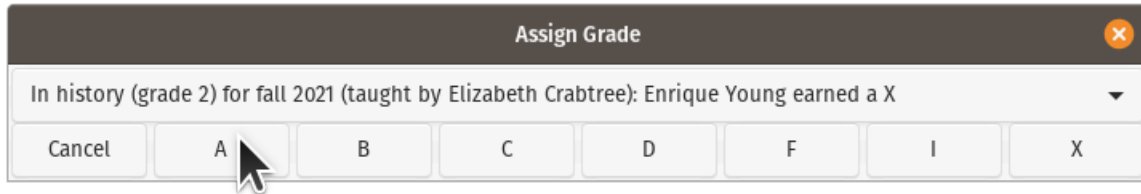
sports (grade 1) for fall 2021 (taught by Elizabeth Crabtree)
music (grade 2) for fall 2021 (taught by Elizabeth Crabtree)
history (grade 2) for fall 2021 (taught by Elizabeth Crabtree)
art (grade 2) for fall 2021 (taught by Elizabeth Crabtree)
math (grade 1) for fall 2021 (taught by Elizabeth Crabtree)
music (grade 1) for fall 2021 (taught by Elizabeth Crabtree)
reading (grade 3) for fall 2021 (taught by Elizabeth Crabtree)
history (grade 1) for fall 2021 (taught by Elizabeth Crabtree)
sports (grade 2) for fall 2021 (taught by Elizabeth Crabtree)
sports (grade 3) for fall 2021 (taught by Elizabeth Crabtree)
science (grade 3) for fall 2021 (taught by Elizabeth Crabtree)
writing (grade 2) for fall 2021 (taught by Elizabeth Crabtree)
writing (grade 2) for fall 2021 (taught by Elizabeth Crabtree)
history (grade 2) for fall 2021 (taught by Elizabeth Crabtree)
music (grade 2) for fall 2021 (taught by Elizabeth Crabtree)
art (grade 3) for fall 2021 (taught by Elizabeth Crabtree)
reading (grade 3) for spring 2021 (taught by Saint Bingham)
writing (grade 1) for spring 2021 (taught by Saint Bingham)
sports (grade 2) for spring 2021 (taught by Saint Bingham)
writing (grade 2) for spring 2021 (taught by Saint Bingham)
math (grade 3) for spring 2021 (taught by Saint Bingham)
reading (grade 1) for spring 2021 (taught by Saint Bingham)
science (grade 2) for spring 2021 (taught by Saint Bingham)
art (grade 2) for spring 2021 (taught by Saint Bingham)
reading (grade 3) for spring 2021 (taught by Saint Bingham)
science (grade 1) for spring 2021 (taught by Saint Bingham)
history (grade 1) for spring 2021 (taught by Saint Bingham)
music (grade 1) for spring 2021 (taught by Saint Bingham)
art (grade 3) for spring 2021 (taught by Saint Bingham)
sports (grade 3) for spring 2021 (taught by Saint Bingham)
math (grade 3) for spring 2021 (taught by Saint Bingham)
history (grade 1) for spring 2021 (taught by Saint Bingham)
art (grade 2) for summer 2021 (taught by Kermit Harness)
music (grade 2) for summer 2021 (taught by Kermit Harness)
writing (grade 2) for summer 2021 (taught by Kermit Harness)
sports (grade 2) for summer 2021 (taught by Kermit Harness)
history (grade 3) for summer 2021 (taught by Kermit Harness)
writing (grade 2) for summer 2021 (taught by Kermit Harness)
science (grade 3) for summer 2021 (taught by Kermit Harness)
history (grade 2) for summer 2021 (taught by Kermit Harness)
writing (grade 2) for summer 2021 (taught by Kermit Harness)
music (grade 2) for summer 2021 (taught by Kermit Harness)
art (grade 1) for summer 2021 (taught by Kermit Harness)
math (grade 1) for summer 2021 (taught by Kermit Harness)
science (grade 1) for summer 2021 (taught by Kermit Harness)
science (grade 3) for summer 2021 (taught by Kermit Harness)
reading (grade 2) for summer 2021 (taught by Kermit Harness)
science (grade 1) for summer 2021 (taught by Kermit Harness)



This is the Relate Student / Parent dialog, reworked with your custom ComboBoxContainer template to provide two auto-populating combo boxes and two buttons. Feel free to label the student and parent for greater clarity.

The image shows a dialog box titled "Relate Student to Parent" with a close button in the top right corner. It features two vertically stacked dropdown menus. The first dropdown menu displays "Isabella Young" and the second displays "Susan Young". Below the dropdowns are two buttons: "Cancel" and "Relate".The image shows the same "Relate Student to Parent" dialog box, but with a list of names displayed instead of dropdown menus. The names are "Robert Young", "Susan Young", "Nimish Adani", and "Oscar Montgomery". "Susan Young" is highlighted with a blue background, indicating she is the selected option.

This is a custom dialog for assigning a grade to a transcript of a section for a given student. The drop-down is our template ComboBoxContainer, and the grades are on buttons created in a loop by iterating over all of the Grade enum class values (this requires a vector that *you* create - C++ cannot iterate over enum values directly, unlike most languages).



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

Bonus Features

The bonus features suggested in Sprint 4 may be implemented for additional points listed in the Bonus column *after* the required features have been implemented. (If you missed an earlier sprint and baselined the suggested solution, you are NOT disqualified - you just need to have completed the features required since your latest baseline.) You may implement the bonus features in any order you please, but minimal implementation of all of them won't earn as much as a few well-crafted gems.

If you implement one or more bonus features to a significant extent, **be sure to mark its Status (column G of the Product Backlog tab in your Scrum spreadsheet) as 'In Work', 'In Test', or 'Finished in Sprint n'** (where n is the sprint in which you actually finished it). The graders will only grade bonus features that you claim to have extensively implemented or completed. (Please don't mark bonus features as complete if you didn't implement them. I take a dim view of those who waste my graders' time. Defend your integrity!)

Yes, if you implement them all, you could get a homework grade of 322 for Sprint 5. Full bonus points for each feature will require a good, working, non-astonishing implementation, but as always we give partial credit for the old college try at our sole discretion.

UNIFIED

Bid EntryDialog adieu! Replace any callbacks that still use a sequence of two or more dialogs (such as EntryDialog) with a single, unified dialog with one or more widgets. We discussed this in Lecture 14.

After you complete a few, they go faster. Remember that the unified dialog for `on_student_parent_click()` was required in Sprint 5 to demonstrate your `ComboBoxContainer` template, so it doesn't count as bonus points.

Each unified dialog you create from those listed earns 6 bonus points.

EDIT

Five of the major data types are immutable and cannot be edited (the Grade in a Transcript can be changed as often as desired). But you can improve this! Add an Edit menu (for example) listing the data types. Once clicked, allow the user to select from a `ComboBoxContainer` of the elements of the associated vector. Then display the same unified dialog as was used to originally create it and allow the user to change the selections.

But how to actually change them? Either add a method for this purpose, or just create a new one and replace the pointer.

If you haven't been diligent to sustain aggregation, the old data may hang around when part of another object, e.g., if your `Student` class has separate copies of each `Parent` (aggregation) rather than a pointer to a single copy in `Mainwin` (composition). We won't count off for this, given the time crunch.

SCROLL

If you create enough data, you eventually won't be able to see it all. That's why *scroll bars* were invented!

`Gtkmm` includes a `Gtk::ScrolledWindow` widget that can contain a single widget. If you put the `Gtk::Label` display into that, you can configure scroll bars.

STATUS++

Noticably missing in the feature list has been a status bar with appropriate updates (e.g., "Written to untitled.smart" after File > Save). For these bonus points, implement a status bar using `Gtk::Statusbar` and display messages where appropriate (after each successful command plus error messages sounds appropriate to me). That's about half of the points - the remaining points can be won by a simple demonstration of stacked status messages (that is, after a temporary message such as an error message, the original message returns).

MULTI

The largest value bonus feature takes the most work, which seems fair. Move the vector attributes from `Mainwin` to a separate class called `School`, and provide an interface to manipulate them. (I would be happy to help you find something workable.) From your current `Mainwin`, separate all of your *user interface* code into `Mainwin`, and all of your *data management* code into `School`.

File > New School then instances a new `School` object and places it onto a vector in `Mainwin`. You'll have an int or a pointer named `active_school` on which all commands are executed. Provide a means to switch to a different loaded school, e.g., Window > [school name] or File > Switch School and a dialog.

This is the way Microsoft Office (as a well-known example) works, although it now tends to put each of its documents into separate windows rather than switching within a single window.

PHOTO

One of the nicer features of Canvas for professors is the tiny photos of students that can be enlarged to a visible size. Add an attribute to `Student` that stores the filename of a photo. Provide a unified dialog with a `Gtk::Image`, `Gtk::ComboBoxContainer`, and Close button such that when the user selects a student from the combo box, their photo (if available) is immediately loaded into the `Gtk::Image` widget.

GPA

This is an example of a report, which are very common in database-like apps (again, we would use a database for this project if we didn't need to practice our object-oriented programming skills). In this case, provide a combo box for the user to select a student. Then present a table (you can use a "real" `gtkmm` table, or use the teletype monospaced font in the `Gtk::Label` to fake it) of all of a student's Transcript objects. At the bottom, show their GPA.

Other

If you finish all of these (!), feel free to suggest or ask for more. Numerous other reports similar to GPA make sense, and you may have some additional features that you'd like to implement for fun and bonus points. Don't be shy!