School Management And Reporting Tool

Due Tuesday, March 23 at 8 a.m.

CSE 1325 - Spring 2021 - Homework #6 / Sprint 2 - 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 2 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 2 - Windows with Simple Dialogs

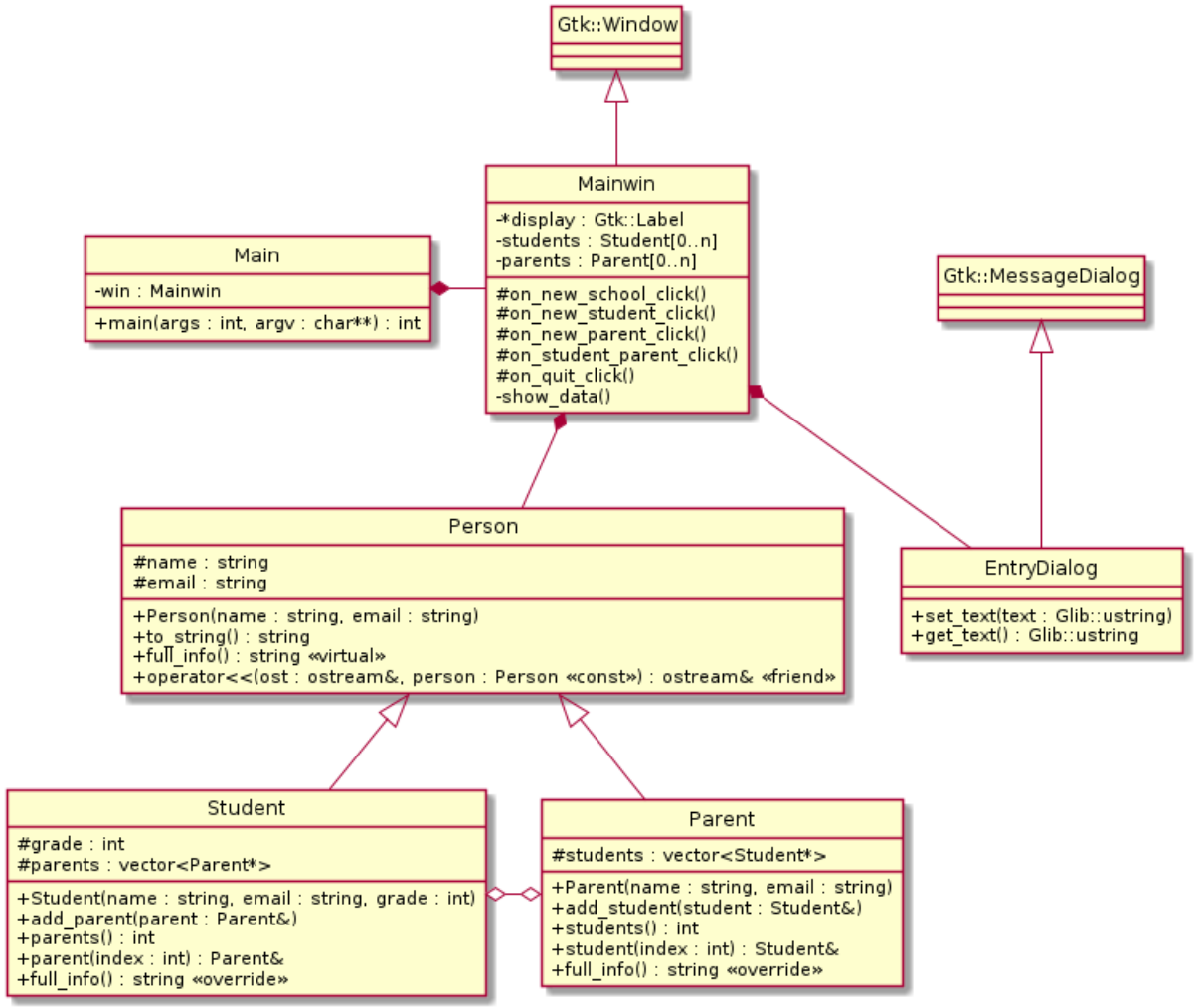
The goal for sprint 2 is to provide the same functionality as in sprint 1, but using only a Graphical User Interface (GUI) - NO console / command line interaction is permitted!

A suggested class diagram for the second sprint is on page 2. **The 3 classes from sprint 1 should not change at all.** You will add a Mainwin class (which inherits from Gtk::Window) and a very simple main function. You are permitted but not required to use the EntryDialog class from Lecture 14.

For this sprint, you **MAY** adopt code from the sprint 1 suggested solution with the following caveats:

- Code from cli.cpp and Makefile may be adapted *without attribution*. You may treat code from these two files as if it is in the public domain.
- Code from the Person, Student, and Parent classes may be adapted under the terms of the Gnu General Public License 3 (GPL3), which requires attribution among other restrictions. An About dialog (as in The Game of Nim) is a good way to fulfill these obligations.

Class Overview



Person, Student, Parent

These classes should not require changes for this sprint.

EntryDialog

This simple dialog class (which adds an entry widget to Gtk::MessageDialog) replaces std::getline almost verbatim. You are permitted but *not required* to use this class. It may help, though.

You will find it at cse1325-prof/14/02_entrydialog.

Main

As always, main() is a *function*, not a class. UML simply doesn't support diagramming functions.

The main.cpp file should be identical to the one in The Game of Nim, except for the program name in Gtk::Application::create.

Mainwin

And here's where you'll spend most of the week!

I *strongly* recommend that you baseline mainwin.h and mainwin.cpp from The Game of Nim in Lecture 13. Baseline means (1) copy those files to your P06 working directory, (2) add / commit them as-is to git, then (3) modify them to meet these requirements.

Here is additional information on each Mainwin member.

- Mainwin - The constructor should create the menu, linking each menuitem to a callback. It should also instance the display area (a Gtk::Label is suggested). Feel free to keep the toolbar and status bar code as well - those are coming in future sprints. (You may comment them out if you like, leave them active but unused, or implement them now and get ahead of the project with more time for bonus features.)
- ~Mainwin - The virtual destructor is empty. It exists to ensure the Gtk::Window::~Window destructor is called.
- display - This is a pointer to the Gtk::Label (or whatever) that will display your data in the main window. Construct it in the constructor, and update it using its set_text method whenever data changes. Pro Tip: Once things are working, enable Pango and aim for a much more professional appearance.
- students and parents - Standard UML notation is shown for an array, which in C++ usually means (wait for it) std::vector. You may use a single vector if desired, but separate vectors for Student and Parent are much easier and are recommended. Optionally, you may create a School class (as shown in the sprint 1 "Where We're Going" section) to manage these vectors for Mainwin. IMPORTANT: **The Person class hierarcy requires that their objects never move in memory**, so you must either ensure that the vector size is never expanded (see cli.cpp in the sprint 1 suggested solution for one approach) OR allocate these objects on the heap and store pointers to them instead. (The latter is a better approach, I think, but we'll accept either.)
- on_new_school_click() - This method simply clears the vectors and updates display. It is the callback for File > New.
- on_quit-click() simply exits the program. It is the callback for File > Quit.
- on_new_student_click() - This method should obtain the name, email, and grade for a new student, instance a Student class, and add it to the students vector. It is the callback for Insert > Student.
- on_new_parent_click() - This method should obtain the name and emai for a new parent, instance a Parent class, and add it to the parents vector. It is the callback for Insert > Parent.
- on_student_parent_click() - This method should ask the user to select one student and one parent, then relate them (see code for option 3 in sprint 1's cli.cpp suggested solution). It is the callback for Relate > Student to Parent. You may want to add private methods to Mainwin for selecting an int between a min and max value, and also add private methods for selecting a student or a parent from the associated vectors. One way to do the latter is with an EntryDialog in which the list of students or parents is the prompt. (We'll replace this later with a much more intuitive drop-down selector.)
- show_data() - This optional private method updates display with the current students and parents' full_info results. It should be called when Mainwin constructs and any time those vectors change.

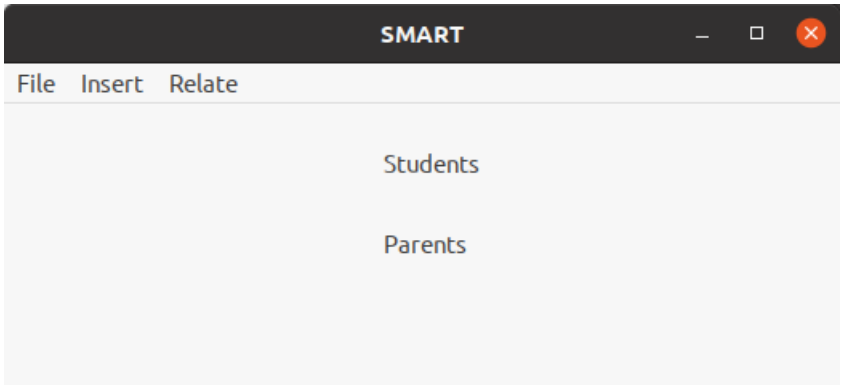
Once you have things working, consider how entering bad data (such as an empty name or email address, an invalid grade, or a non-integer grade), closing each dialog, or clicking Cancel in each dialog affects your program. Remember the Principle of Least Astonishment. Don't astonish your grader!

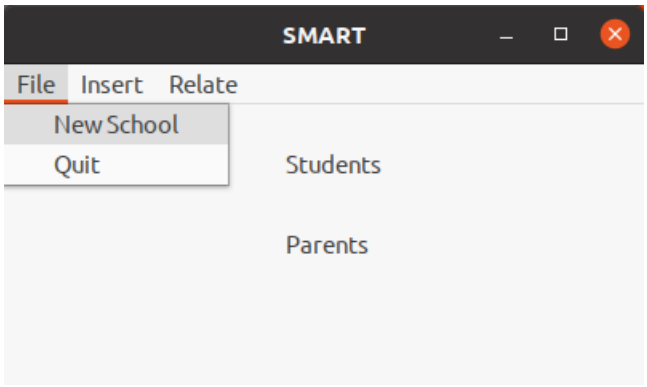
Makefile

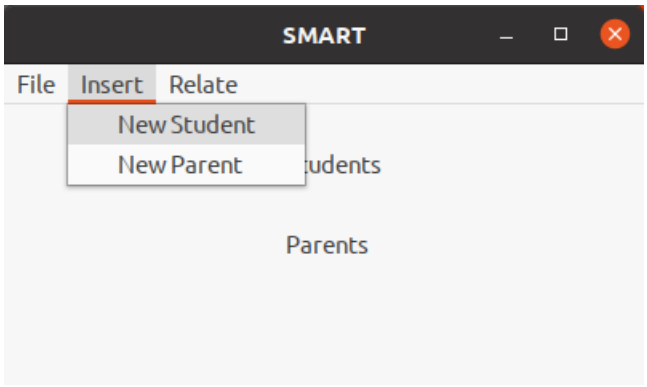
Keep the same targets as in sprint 1, including target 'smart'. You will need to add the GTKFLAGS variable (as in Nim), and add `$(GTKFLAGS)` to any compile or link that includes gtkmm references - that is, main.cpp, mainwin.cpp, entrydialog.cpp (if you are using it), and the link.

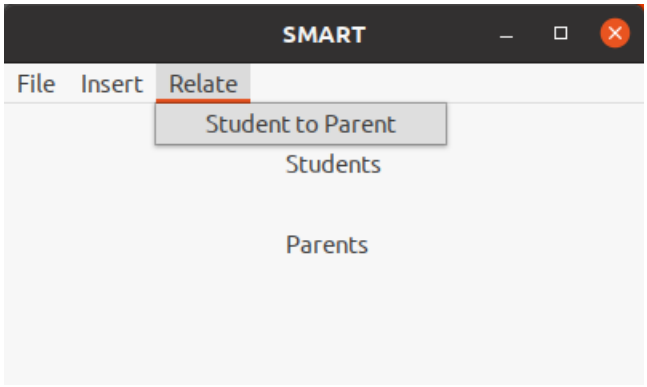
GTKFLAGS is what tells the compiler where your gtkmm libraries reside. If you forget it, you will get a "gtkmm.h not found" compiler error.

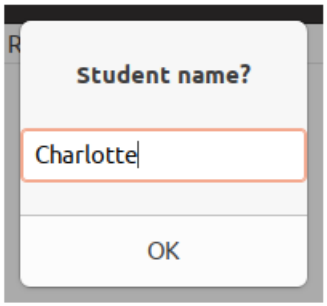
Here are some screenshots from the suggested solution. **Your application may look different than these.** Your application may look much *better* than these. Just ensure you meet all of the required features for this sprint!

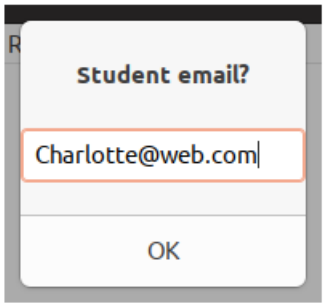


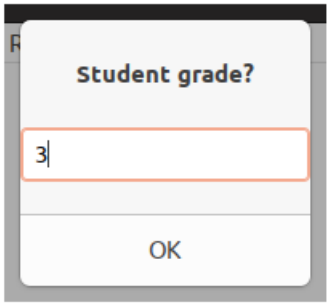


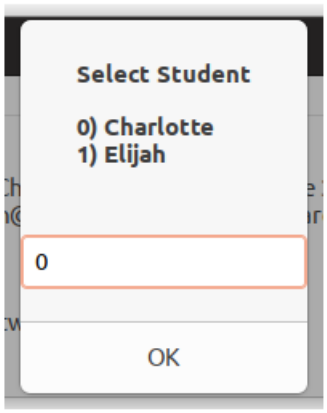


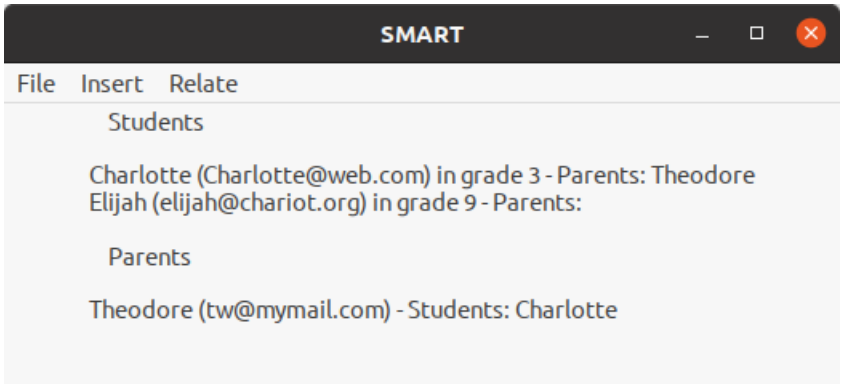












If you have questions about any part of this project, contact me via email or Teams. Ask questions. I **love** questions!