学生管理系统

Introduction

This project is designed for managing (massively) student information and the correlations between the student and other parties(e.g. the courses that the student has choosen and the corresponding learning outcome (grade or score)).

Design requirements

- All data is stored in hard disk with reliable approach, which means that the data is reusable after the process is terminated.
- Users can add, delete, modify and query the information regarding all possiblities, such as "adding student", "adding course", "listing all the courses and learning outcomes of a student", "choosing a specific course for a specific student (and generating a random learning outcome)", "listing all the students who have choosen a specific course" and so on.
- The time complexity of querying should be equal or less than logarithmic ($T(n) = O(\log n)$).
- The volume of data can be massive. At least 100, 000 students and 1, 000 courses should be stored in disk as default.
- Using 3rd party libraries, such as STL, is forbidden.

Practical meaning in the future

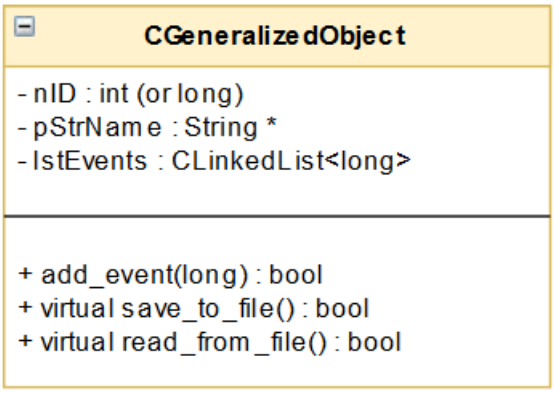
With a solid and proactive software design, a programmer should be able to conveniently reuse this very framework in other similar information management systems, such as Supermarket Purchasing/Reselling/Stocking Management System, Hospital Patient/Doctor/Nurse Information Management System and so on.

Project's core design principle

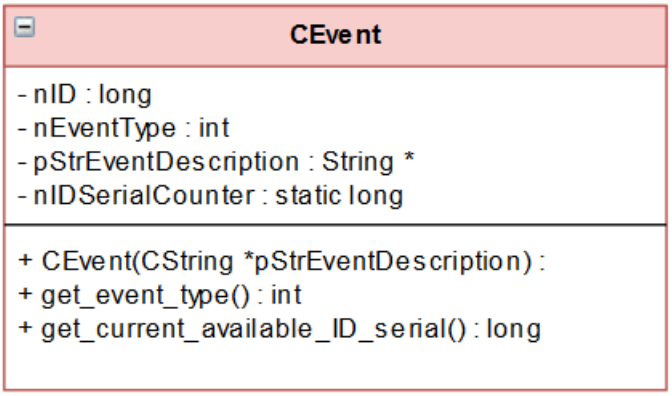
With the aforementioned pratical meaning in the future bearing in mind, the core design principle (core concept) of this project is called "event list".

All the objects (e.g. students, courses, teachers, goods, cashiers, patients and so on) and their information are managed with an independent domain (usually a balanced binary tree for the sake of querying effciency) according to their class property. A event will be generated whenever an object is involved in an event related with other parites, such as a student has choosen a specific course (and gotten the corresponding learning outcome), a good in the supermarket has been sold via a specific cashier's desk, a patient has received the treatments from a specific doctor with the assistance of a specific nurse, and so on. Please note that the number and class property of the "other parties" is not restricted, which meanings, saying a student can establish relationship with other student(s). For instance, a student can create a habbit club with other five students. Moreover, this habbit club can even have other two guiding teachers included.

To realize this design priciple, each object should have a linked list as private data member, which records all the events with which this object was involved. The design of this generalized class is descibed below:



Note that an event is mainly represnted by a single string, as shown in the figure below:



The format of this event string is described in the next section.

Event string design

For the sake of software scalability and flexibity, using a single string to describe the event is a handy option. However, one obvious shortcoming of this approach is that the processing of encoding and decoding the string is extremely prone to the disturbance of delimiter noise. This shortcoming requires the programmer to carefully consider the selection of delimiter, as well as which can be sent to the string as a valid column and which can not. The best practice is using solely integer as the column content, and rarely used symbol as the delimiter. An example protocal is shown blow:

23	#	2	#	10022	#	389	#	67.8	#	\0
event id	delim	event type	delim	student id	delim	course id	delim	learning outcome	delim	end

The above example demonstrations the constrcution of an event (event id: 23, event type 2) that a student (student id: 10022) has choosen a specific course (course id: 389) and gotten a learning outcome of 67.8.

Potential problems with the current event string design

In practice, a progammer can barely keep the event column content as solely integer numbers. The "learning

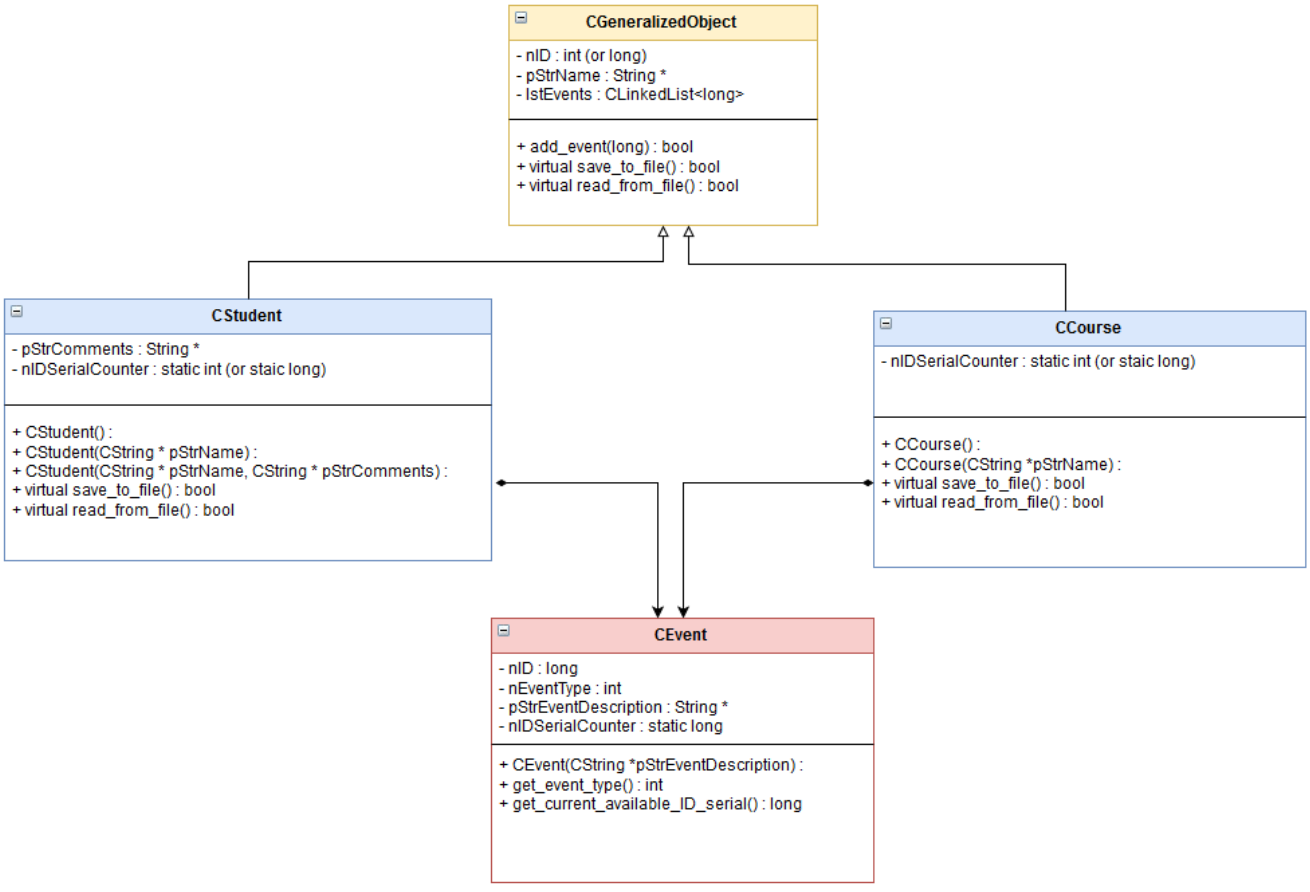
outcome" in the previous string is a very representive example. Considering the event that a customer has left a random comment (could be any legitimate sentence) regarding the supermarket service for the manager, a progammer should always think about how to design the corresponding string protocal carefully.

The safest approach is to define a dedicated class that inherits from class "Generalized Object" to represent the "learning outcome" or "customer comment", taking those troublesome issues as objects, assigning an unique id to each of them. Think of the event that a student has created a habbit club. By applying this approach, we will make the number of involved objects from 2 (student and course) into 3 (student, course and learning outcome).

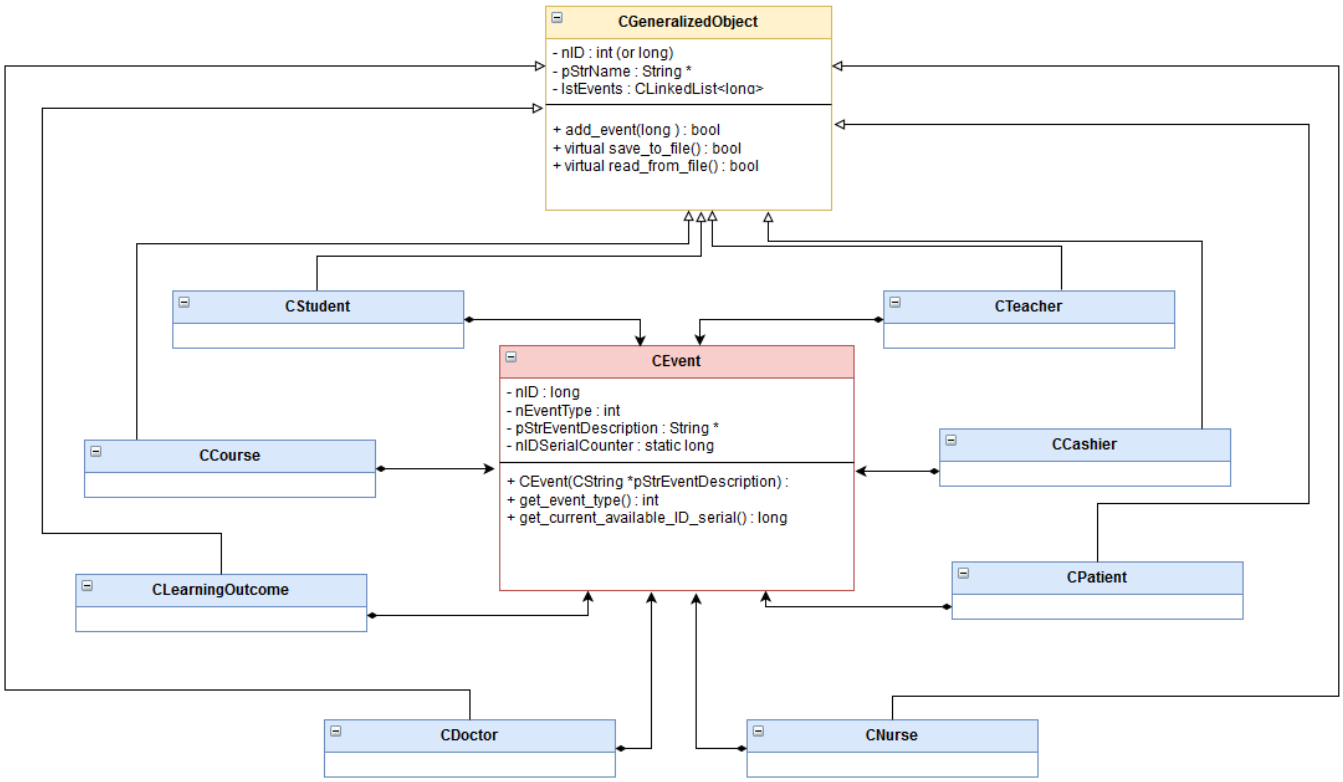
Nevertheless, considering the current project's design requirements (the attributes of student and course are not too complicated), we did not implement the aforementioned approach. Instead, the simple attributes, just like the learning outcome, are just inserted into event description during the process of event description generation without extra consideration.

Core design principle summary

The following diagram describes the core design principle:



In the future, when this framework is applied to other programs that have more kinds of objects, the abstract design principle can be described as below:

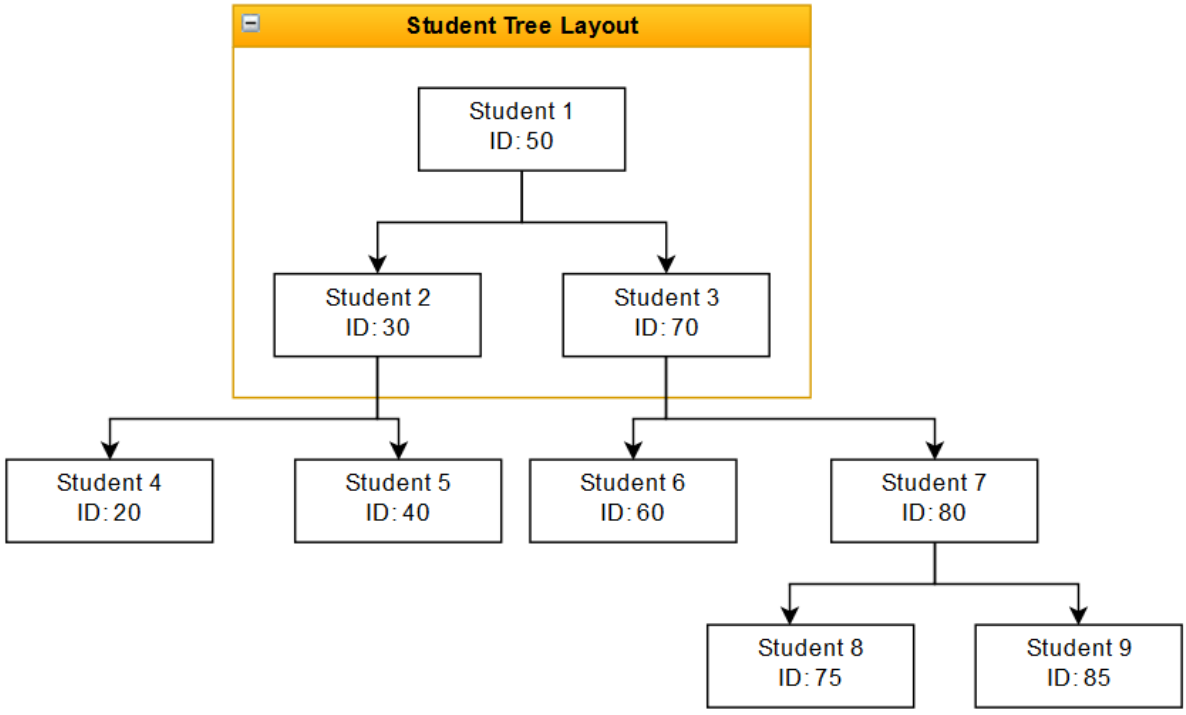


Data storage

Data structure in memory

General objects (student and course)

According to the class they belong to, the information of general objects is stored in independent domains, organized by balanced binary tree (AVL tree). For example, all the student informtion is stored in the tree, sorted by the student ID, described in the following diagram:



In assistance with this balanced binary tree, a hash table will be built to realize fast-querying via name.

We use a string-to-hash function to map each name into a fix-sized array, and each slot of this array contains a linked list of the objects whose names share the same hash value. Taking student information as example, when querying a specific student's name, we first calculate the hash value of this name, and then we can locate the specific slot that contains the linked list of one or more students. To find the specific student(s), we traverse the linked list and compare the student names (using strcmp() function). Finally, after the entire linked list is traversed, we get a list of student(s) who has the name we queried. The hash table is described in the diagram below:

Hash table

0	1	...	899	900	901	...	end
			student ID: 123 student name: Wang student ID: 7210 student name: Wang student ID: 8899 student name: Wang student ID: 8521 student name: Wang	student ID: 2789 student name: Qwera	student ID: 2210 student name: Zhao student ID: 3221 student name: Zhao student ID: 8702 student name: Wu		

As shown in slot 901, different student names can share the same hash table slot, due to the particular hash value distribution. A random integer seed must be set with the string-to-hash function, and this seed should remain the same during both the construction and querying period of a hash table.

The same balanced binary tree and hash table design applies to course information as well. As a matter of fact, this pattern should apply to all other general object's information in the future development. In summary, constructing one balanced binary tree is guaranteed for each class; one hash table should be built for each attribute that needs

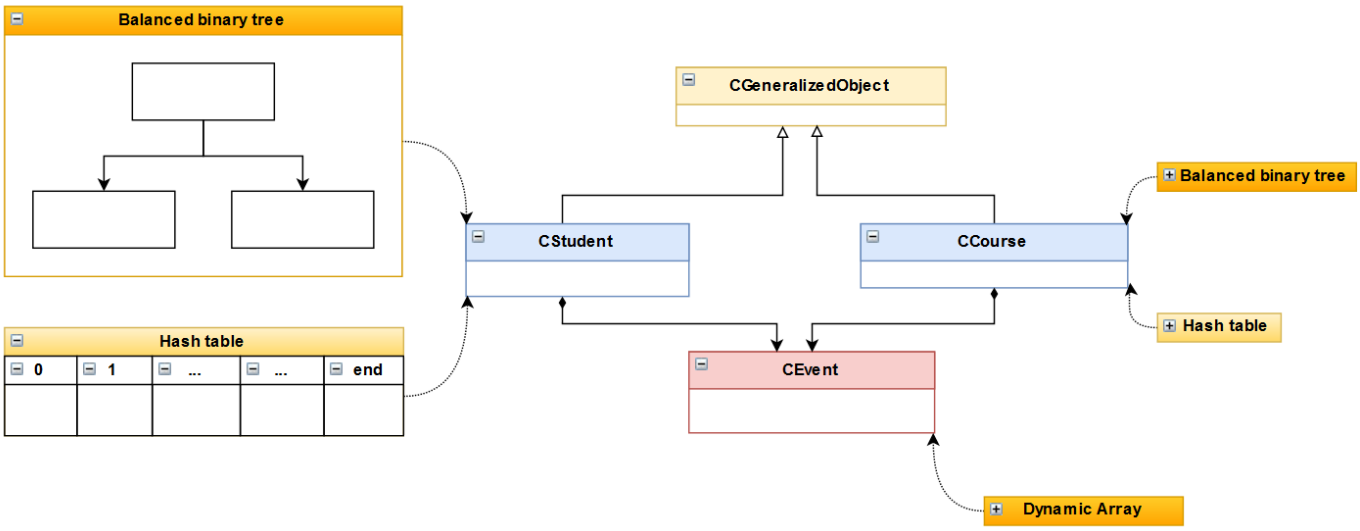
fast-querying.

Event objects

Since event objects are produced in a incrementally manner, it is not necessary to implement complex data structure. A dynamic array which scalability is good enough to achieve efficient querying for events. The dynamic array's index is exactly the ID of events.

Overall status

The overall status of data structure in memory is described in the following diagram:



Data format in file

In order to reliably read data from file and write data to file, a certain data format (file format)should be clarified in advance.

General objects (student and course)

For general objects, two files are assigned to one class: "roster" and "data". Taking student as example, file "student_roster" acts as an index, and file "student_data" stores the detailed data of each student.

The data format in "roster" is described in the following diagram:

content	T	ABCDE123	000020	\0
property	char	hex	int	char
size	1 char	8 char	6 char	1 char
meaing	is_valid	offset	length	end

> Reminder: the "offset" column in roster means this object's data offset in the "data" file. It is NOT the offset in the "roster" file.

The data format in "data" varies according to specific class definition. Taking student as example, the data format in "student_data" is described as below:

content	10000000	#	wang	#	love study	#	11688	#	11689	#	...	\0
property	int	char	string	char	string	char	long	char	long	char	--	char
size	--	1 char	--	1 char	--	1 char	--	1 char	--	1 char	--	1 char
meaing	ID	delim	name	delim	comments	delim	event id	delim	event id	delim	more event id and delim	end

Event objects

Smilar as general objects, two files are assigned: "event_roster" and "event_data".

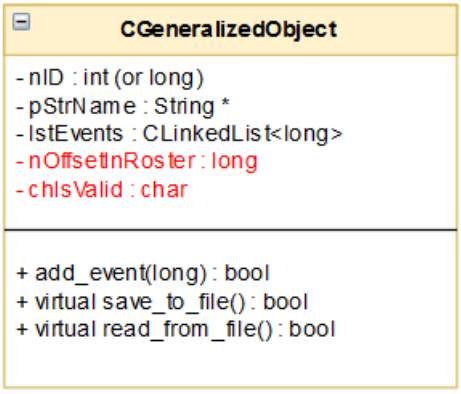
The data format in "event_roster" is exactly same as the "student_roster" file. In other words, the data format of all "roster" files of this project are identical within the whole project.

The data format in "event_data" is shown as below:

content	12300000	#	45	#	021	#	45678	#	11223	#	...	\0
property	long	char	int	char	--	char	--	char	--	char	--	char
size	--	1 char	--	1 char	--	1 char	--	1 char	--	1 char	--	1 char
meaing	ID	delim	event type	delim	event column	delim	event column	delim	event column	delim	more event column and delim	end

Class property modification to cooperate with file operation

As shown above, writing data to file and reading data from file have strict regulations. To cooperate with the regulations, we need to add corresponding data members into class. The updated "CGenerlizedObject" class now should be:



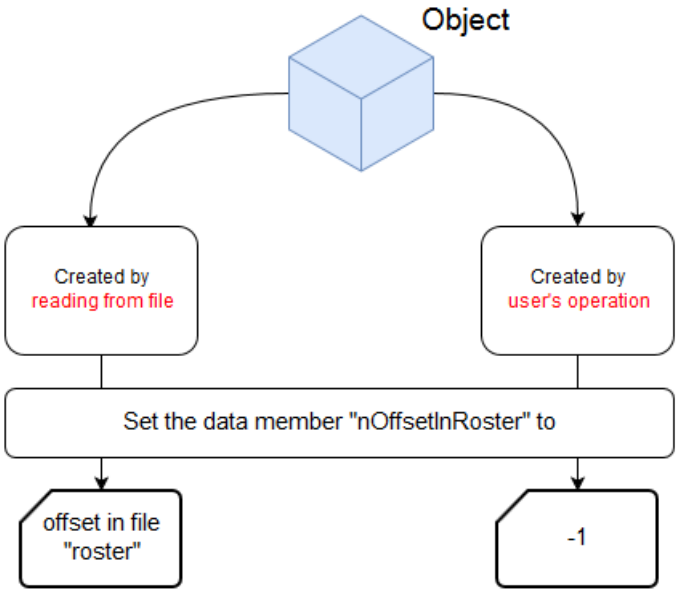
The data member "nOffsetInRoster" is set when reading the "roster" file, and this data member can help when we updating the "roster" file and "data" files after user has made changes to object's property.

When an object is deleted, we simply set the data member "chIsValid" to 'F'. The next time user opens this project, the object's with a 'F' property will not be read into memory from file.

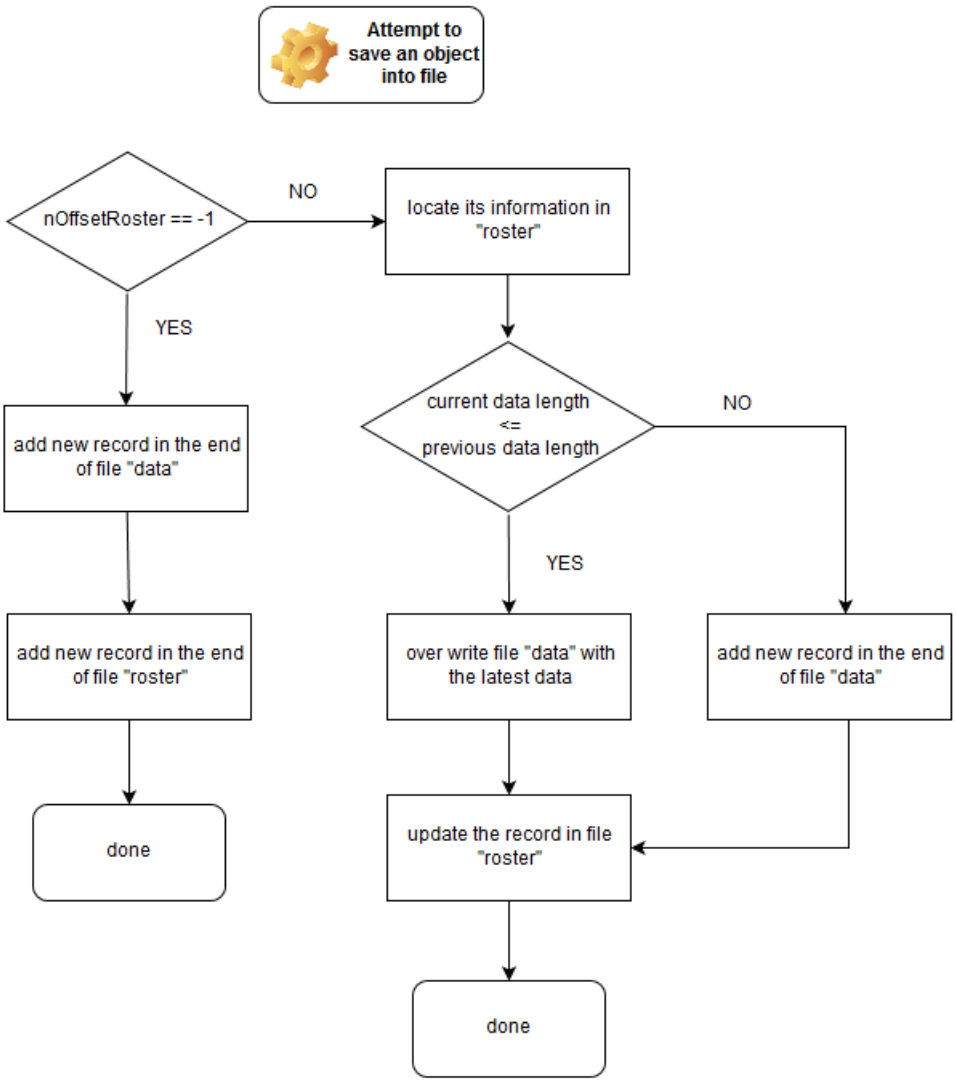
file operation

Save data to file

If an object is read from file, it's offset in "roster" will be recorded in data member "nOffsetRoster". If an objext is just created by user, the object's data member "nOffsetRoster" will be set to -1 via the construction function. The process is described as below:



When attempting to save an object into file, we first check if it has a valid data member "nOffsetRoster". If not, we add a new record in the end of file "data", and add a new record in the end of file "roster". If yes, we locate its information in "roster", and we check if the latest data's length is smaller or equal to the previous one. If not, we add a new record in the end of file "data". If yes, we overwrite the former data in "data". No matter what, we must update the object's record in "roster" with the latest information as well. The process is described as below:



Read data from file

Reading data from file is quite straightforward with the combination of both file "roster" and file "data". The program reads file "roster" from beginning to the end, traversing the objects. Unless the object is with a 'F' property regarding is_valid, the program will locate this object's offset in file "data" and map the file section into memory. With the content in memory, the program constructs the object.

File size limit

- File "roster" does not have a size limitation.
- With size of the "offset" column of file "roster" setting to eight charactor, the maximum size of file "data" is 4GB. Assuming each data entry has a size of 40 byte, the approximately maximum number of data entry will be 107, 000, 000.

File size optimization

After a certain period of usage, file "roster" may contain many objects that is not valid any more (deleted by user and is_valid set to 'F'). Meanwhile, file "data" could also contain data sections that are no longer needed.

To optimize the file size, the program follows these steps:

1. Create two files named "roster_new" and "data_new".
2. Traverse file "roster" from beginning to the end, and check if an object is still valid (the is_valid) column. Copy the object to the end of file "roster_new" if it is valid.
3. Delete file "roster". Rename file "roster_new" as "roster".
4. Traverse file "roster" from beginning to the end, and copy every data section the "offset" column points to the end of file "data_new". After each moving operation, update the corresponding "offset" column in file "roster".
5. Delete file "data". Rename file "data_new" as "data".

Other design details

String object factory

Since

ID recorder files

qw

Command list

Here lists the full command table:

- -g num
  Generate *num* random students and added them into storage.
- -a "string1" ["string2"]
  Add a student, setting name as *string1*, and comments as *string2* (optional).
- -d num
  Delete a student, whose ID is *num*.
- -m num "string1" ["string2"]
  Modify a student's information whse ID is *num*, setting name to *"string1"*, and comments to *"string2"* (optional).
- -l [num | "string"]
  List the student whose ID is *num*, or list the students whose names are *"string"*. If no argument is given, list all.
- -G num
  Generate *num* random courses and added them into storage.
- -A "string"
  Add a course, setting name as *string*.
- -D num
  Delete a course, whose ID is *num*.
- -M num "string"
  Modify a course's information whse ID is *num*, setting name to *"string"*.
- -L [num | "string"]
  List the course whose ID is *num*, or list the courses whose names are *"string"*. If no argument is given, list all.
- -c num1 num2
  Make the student whose ID is *num1* to randomly choose *num2* courses.
- -C num1 num2
  Make the student whose ID is *num1* to choose the course whose id is *num2*.