Ethan Peralta

Professor Zhao

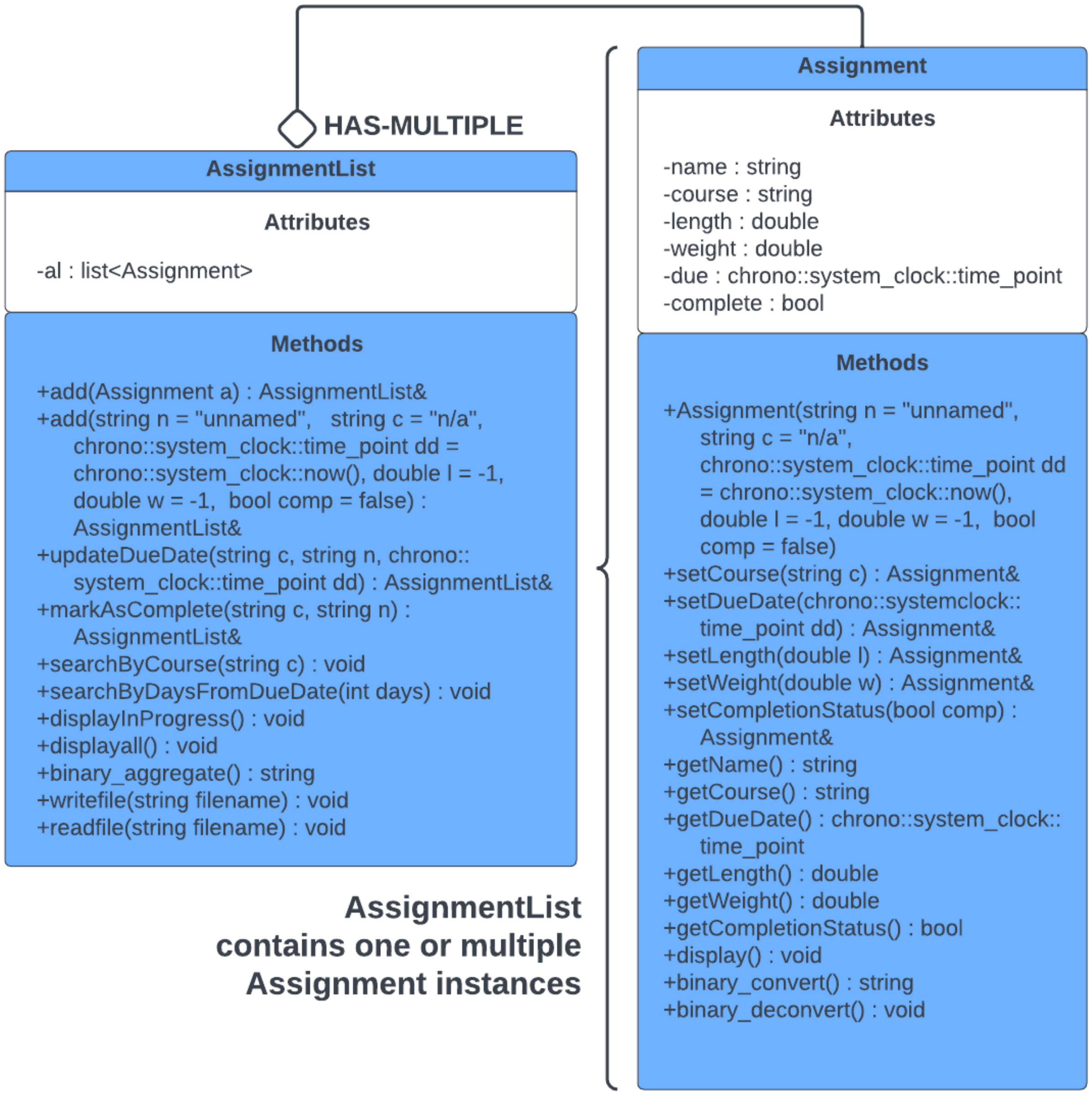
BCS370

5/3/2023

College Student Homework Management System

a. ***Team member(s):*** Ethan Peralta [individual project]

b. ***UML diagram***


HAS-MULTIPLE
AssignmentList
Attributes
-al : list<Assignment>
Methods
+add(Assignment a) : AssignmentList&
+add(string n = "unnamed", string c = "n/a", chrono::system_clock::time_point dd = chrono::system_clock::now(), double l = -1, double w = -1, bool comp = false) : AssignmentList&
+updateDueDate(string c, string n, chrono::system_clock::time_point dd) : AssignmentList&
+markAsComplete(string c, string n) : AssignmentList&
+searchByCourse(string c) : void
+searchByDaysFromDueDate(int days) : void
+displayInProgress() : void
+displayall() : void
+binary_aggregate() : string
+writefile(string filename) : void
+readfile(string filename) : void
AssignmentList contains one or multiple Assignment instances
Assignment
Attributes
-name : string
-course : string
-length : double
-weight : double
-due : chrono::system_clock::time_point
-complete : bool
Methods
+Assignment(string n = "unnamed", string c = "n/a", chrono::system_clock::time_point dd = chrono::system_clock::now(), double l = -1, double w = -1, bool comp = false)
+setCourse(string c) : Assignment&
+setDueDate(chrono::systemclock::time_point dd) : Assignment&
+setLength(double l) : Assignment&
+setWeight(double w) : Assignment&
+setCompletionStatus(bool comp) : Assignment&
+getName() : string
+getCourse() : string
+getDueDate() : chrono::system_clock::time_point
+getLength() : double
+getWeight() : double
+getCompletionStatus() : bool
+display() : void
+binary_convert() : string
+binary_deconvert() : void

c. *Key technical implementation descriptions & explanations*

i. ***Data structure***

For the data structure, I opted to use the standard STL implementation of linked list. Since I was storing an unordered collection of objects and wanted low runtime complexity, it was the obvious choice. Plus, I had a similar project in a Java data structures class where I had a collection of game objects. I also modeled that with an unordered linked list.

ii. ***Recursive function(s)***

```
string binary_aggregate() {
		string agg_str;
		for (auto &a : this->al) {
			agg_str += a.binary_convert();
			if (!this->al.empty()) { agg_str += ","; }
		}
		return ("[" + agg_str + "]") ;
	}
```

I honestly understand if you don't give me credit for recursion here. This function is very lazily recursive; I added in the recursion as an afterthought, because I had forgotten to implement any other member functions recursively and they would have been a headache to rewrite. However, in my own defense, *it does call itself within the function body…* ***recursively***. (no matter how loosely recursive :p)

iii. ***Complexity of* searchByDaysFromDueDate() *and* searchByCourse()**

```
void searchByDaysFromDueDate(int days) {
	list<Assignment> results; // O(1)
	for (auto& a : this->al) { // O(N)
		tm dtm; // O(1)
		time_t dt = chrono::system_clock::to_time_t(a.getDueDate()); // O(1)
		if (gmtime_s(&dtm, &dt) != 0) { return; } // O(1)
		time_t cur = time(NULL); // O(1)
		time_t conv = mktime(&dtm); // O(1)
		if (difftime(conv, cur) / 86400 <= days) { results.push_back(a); } // O(1)
	}
	if (results.empty()) { cout << "No results found!\n"; } // O(1)
	else { // O(1)
		cout << "List of assignments due in the next " << days << " days: " << endl << endl; // O(1)
		printf("%-18s%-30s%-27s%-19s%-21s%-20s\n", "Course title", "Name of assignment", "Due date &
			time", "Weight (0 - 1)", "Est. length (hrs)", "Status"); // O(1)
		cout << "-----------------------------------------------------------------------------------
			-----------------------------------------" << endl; // O(1)
		for (auto& a : results) { a.display(); } // O(N)
		cout << endl << endl; // O(1)
	}
}
```

As you can see, all of the lines of code in the snippet above are of constant runtime complexity O(1) with the exception of the two for-loops. Fortunately, neither of the for-loops interact with one another; they are not nested. The runtime is O(N) since constants are irrelevant and this is the dominating term among all present. I'm glad I was able to make this function run efficiently while piggybacking off the normal STL linked list implementation.

```
void searchByCourse(string c) {
	list<Assignment> results; // O(1)
	for (auto& a : this->al) { // O(N)
		if (a.getCourse().find(c) != -1) { results.push_back(a); } // // O(1)
	}
	if (results.empty()) { cout << "No results found!\n"; } // O(1)
	else { // O(1)
		cout << "List of assignments with '" << c << "' in course title: " << endl << endl; // O(1)
		printf("%-18s%-30s%-27s%-19s%-21s%-20s\n", "Course title", "Name of assignment", "Due date &
			time", "Weight (0 - 1)", "Est. length (hrs)", "Status"); // O(1)
		cout << "-----------------------------------------------------------------------------------
			-----------------------------------------" << endl; // O(1)
		for (auto& a : results) { a.display(); } // O(N)
		cout << endl << endl; // O(1)
	}
}
```

This function was built using the searchByDaysFromDueDate() as a reference, so its runtime complexity is the same: O(N). There are two for-loops present once again, but neither of them interact with one another.