COMP 2012 Object-Oriented Programming and Data Structures

Assignment 1 Database Queries

Introduction

Suppose we are given a table of courses that are offered in Spring Semester of 2020. You can find eight courses, each o
are represented as an **entry** in the table. There are four **attributes** in the table, Department, Course Code, Course Nam
Credits. An attribute characterizes information about entries.

Department	Course Code	Course Name	Credits
CSE	COMP2012	Object Oriented Programming and Data Structures	4
CSE	COMP3021	Java Programming	3
CSE	COMP3711	Design and Analysis of Algorithms	3
CSE	COMP4511	System and Kernel Programming in Linux	3
CSE	COMP4641	Social Information Network Analysis and Engineering	3
MATH	MATH2343	Discrete Structures	4
MATH	MATH3033	Real Analysis	4
MATH	MATH4023	Complex Analysis	3

Table 1 - courses

Now we are asked to get the top 3 courses that are offered by CSE, ranked in descending order with their credits, then i
ascending order with their name. This can be done by a SQL query:

```sql
SELECT * FROM courses
WHERE Department = 'CSE'
ORDER BY Credits DESC, `Course Name` ASC
LIMIT 3;
```

But we are taking a C++ course, aren't we? How about writing the query using the *syntaxes* of C++? Imagine we can tra
the above SQL query into a C++ statement:

```
courses.query()
.where("Department", EQ, "CSE")
.orderBy("Course Name", ASCENDING)
.orderBy("Credits", DESCENDING)
.limit(3)
.select();
```

Looks very readable, isn't it? Your tasks in this programming assignment is to implement classes to support database qu
C++.

For those who are unfamiliar with SQL, here is a *brief* introduction to the SQL statements and clauses which are mimicke
this programming assignment. Click here for a full introduction to SQL.

- `SELECT` statement retrieves information from a table in a database. Columns need to be specified explicity by a lis
  their names, or use `*` to denote *all* columns. In this programming assignment, the result of `SELECT` is printed out i
  formatted way.
- `UPDATE` statement modifies rows of a table in a database.

- `WHERE` clause provides a condition such that `SELECT` and `UPDATE` statements return or update rows that satisfy t condition.
- `ORDER BY` clause orders the result of `SELECT` statements according to the values of columns, in ascending or descending order.
- `LIMIT` clause limits the result of `SELECT` and `UPDATE` statements. (MySQL only)

End of Introduction

Download

Download `main.cpp`, `table.h` and `intermediate.h` HERE.

Download expected output HERE.

End of Download

Overview

You are required to implement two classes, `class Table` and `class Intermediate`. The implementations should b written in `table.cpp` and `intermediate.cpp` respectively.

This programming assignment aims to introduce the notion of *classes* and reinforce the concepts of *pointer manipulatio dynamic memory management*. In particular, you will be manipulating the following data structures:

- Dynamically allocated 2D array
- Doubly linked list

You are highly advised to revise the lectures if these concepts sound unfamiliar to you.

End of Overview

`class Table`

Before we talk about queries, we must define a table. For simplicity, in this programming assignment we assume tables c hold `string` data.

A table has zero or more *entries*. The information of an entry is characterized by the *attributes* of a table. In this program assignment, an entry is represented by a 1-D string array of attributes.

Private Members

A table has 4 private members:

```
private:
  string *attrs;
  string **entries;
  int numAttrs;
  int numEntries;
```

- `string *attrs` stores the 1D array of attributes of the table.
- `string **entries` stores the 2D array of entries. `entries[0]` gets the first entry. `entries[0][0]` gets the v
  the first attribute of the first entry.
- `int numAttrs` stores the number of attributes (columns) of the table. For example, in Table 1, `numAttrs` is 4.
- `int numEntries` stores the number of entries (rows) of the table. For example, in Table 1, `numEntries` is 8.

Constructors and Destructors

```
Table()
```

The default constructor creates an empty table with *no* attributes and *no* entries. You should initialize the private membe
accordingly.

```
Table(const Table &another)
```

The copy constructor copies the attributes and entries from the table `another`. Note that deep copy is required.

```
~Table()
```

The destructor deallocates all dynamically allocated memory owned by this instance.

Member Functions

```
bool addAttribute(const string &attr, int index = -1, const string &default_value = "")
```

- Add the attribute `attr` at `index`. Existing attributes are right shifted.
- If the table has existing entries, the 2D array, `entries`, must be expanded. The values of the new attribute is `default_value`.
- If `index == -1`, insert attribute at the end.
- If `index` is out of range, return `false` immediately.
- Otherwise, proceed and return `true`.
- For example, in Table 1, the valid indices are -1, 0, 1, 2, 3, 4. Adding an attribute at index -1 has the same effect of a
  at index 4.

```
bool addEntry(const string *entry, int index = -1)
```

- Add the entry `entry` at `index`. Existing entries are down shifted.
- If `index == -1`, insert entry at the end.
- If `index` is out of range, return `false` immediately.
- Otherwise, proceed and return `true`.
- **Input assertion:** `entry` is guaranteed to have size of `numAttrs`.
- **Input assertion:** Table has at least one attribute (`numAttrs > 0` and `attrs != nullptr`).

```
bool deleteAttribute(int index)
```

- Delete the attribute at `index`.

- If `index` is out of range, return `false` immediately.
- Otherwise, proceed and return `true`.
- If the table has existing entries, the 2D array, `entries`, must be shrinked.
- If there are no attributes left, delete all remaining entries.

```
bool deleteEntry(int index)
```

- Delete the entry at `index`.
- If `index` is out of range, return `false` immediately.
- Otherwise, proceed and return `true`.
- Keep the attributes even though no entries remain.

```
bool append(const Table &another)
```

- If the current table and `another` have different attributes, return `false` immediately. The order of attributes has
  the same.
- Otherwise, append entries of `another`, by adding the entries of `another` to the *end* of current table, and return

---

```
Intermediate query() const
```

- **Given**
- Begin a query by making an `Intermediate` object from the current table.

Hint:

- Since table is instantiated as 1D/2D array, all add/delete operations can only be done by allocating new arrays and deallocating the existing ones.

End of class Table

`class Intermediate`

An `Intermediate` object represent the intermediate results at various parts of a query. Most importantly, it has a doub
linked list of pointers to entry of table being queried:

```
struct EntryNode {
  string *entry;

  EntryNode *prev;
  EntryNode *next;
};
```

It should be emphasized that `Intermediate` objects *do not own entries*. They simply *point* to the entries that are owne
tables.

An `Intermediate` object is created from a table when a query starts. At first, it contains *all* entries of the table, *in the* o *they appear in the table*. During a query, entries can be removed, and the order can be changed, which are done by the following 3 operations:

- `where()`: A condition is specified. Entries that do not satisfy the condition are removed from the `Intermediate`
- `orderBy()`: An attribute and an order is specified. The entries are *sorted* by the attribute according to the given c
- `limit()`: An limit `N` is specified. Only the first `N` entries stay in the `Intermediate` object. The rest of the entries removed.

At the end of a query, the `Intermediate` object is passed to one of the following 2 operations:

- `select()`: The remaining entries are printed out. An optional array of attributes spefifies which attributes are prin absent, all attributes are printed.
- `update()`: An attribute and a new value is specified. The attribute of *all* entries are updated to the new value.

Private Members

An intermediate object has 4 private members:

```
private:
  const string *attrs;
  int numAttrs;

  EntryNode *head;
  EntryNode *tail;
```

- `const string *attrs` points to the 1D array of attributes of the table.
- `int numAttrs` stores the number of attributes.
- `EntryNode *head` and `EntryNode *tail` are the head and tail of the doubly linked list.

Constructors and Destructors

```
Intermediate() = delete
```

No default constructor.

```
Intermediate(const Table &table)
```

The only way to construct an intermediate is by converting from a table. When you convert a table to an intermediate, th doubly linked list of `EntryNode` should store the pointer of entries *in the order they appear in the table*. That is, `head->entry` should point to `entries[0]`, and `tail->entry` should point to `entries[numEntries-1]`. `attrs` should j simply points to `table.attrs`. You don't need to copy it.

You will need to access the private members of `Table` objects. A `friend` class declaration at `table.h` allows you to c in this constructor only. Note: The `friend` syntax will be formally introduced in later sections of this course.

You can assume the following:

- `table` has at least one attribute.
- After an intermediate is created, `table` will not be deleted or modified by any member functions of Table until the intermediate is not used.

```
~Intermediate()
```

The destructor should only deallocate dynamically allocated memory that is owned by this object.

Member Functions

```
Intermediate& where(const string &attr, enum compare mode, const string &value)
```

Implements the `WHERE` clause. A *condition* is specified. Only entries whose conditions evaluate to true remain in the intermediate. The entries whose conditions evaluate to false are removed.

An entry is said to *satisfy* a condition if its attribute `attr` satisfy the comparison `mode` with `value`. The comparison mo defined as:

```
enum compare {EQ, CONTAINS}
```

For example, suppose the attribute of an entry has a value `v`. The comparison is satisfied if:

- For `EQ`(Equals), `v` is equal to `value`.
- For `CONTAINS`, `value` is a *substring* of v.

Hints:

- For substring checking, see `std::string::find`.
- You need to remove elements from a doubly linked list.

Exception:

- If the attribute `attr` is not found, do nothing.

Click here for example.
Click here for example where the attribute `attr` is not found.

```
Intermediate& orderBy(const string &attr, enum order order)
```

Implements the `ORDER BY` clause. The entries in the intermediate are sorted by the attribute `attr`, with order specifie `order`. The order is defined as:

```
enum order {ASCENDING, DESCENDING}
```

For `ASCENDING` order, the entries are sorted in ascending order of their attributes `attr`. Similarly for `DESCENDING` ord String comparisons are performed. If two strings are equal, their relative order does not change (see Examples). Since t not a course on algorithms, you are free to any method to sort the entries.

Hints:

- For string comparison, see `std::string::compare`.
- For a simple sorting algorithm, see Bubble Sort.

Exception:

- If the attribute `attr` is not found, do nothing.

Click here for example.

```
Intermediate& limit(unsigned int limit)
```

Implements the `LIMIT` clause. Only the first `limit` entries remain in the intermediate. All following entries are removed

- If `limit == 0`, results an empty intermediate.
- If `limit` is larger than number of entries, results an intermediate with all entries.

Hint:

- You need to *truncate* a doubly linked list.

Click here for example.

```
void update(const string &attr, const string &new_value) const
```

Implements the `UPDATE` statement. The attribute `attr` of *all* entries in the intermediate gets the value `new_value`.

Exception:

- If the attribute `attr` is not found, do nothing.

Click here for example.

```
void select(const string *attrs = nullptr, int numAttrs = 0) const
```

Implements the `SELECT` statement. The entries in the intermediate are printed, only with attributes specified *in the orde* *array* `attrs`, whose size is given by `numAttrs`.

- If `attrs == nullptr`, print all attributes in the order of the table, regardless of `numAttrs`.
- If the table has no attribute, do nothing.
- If the intermediate has no entries, print attributes only.
- **Input assertion:** The attributes in `attrs` are guaranteed to be found and unique.

The output should be *formatted*:

- Each column is separated by three characters: A space, `|`, and another space.
- Each column has the length *equal to the longest value in the column*. Values are *left-padded with spaces*. See the provided helper function `string _left_pad_until(const string &s, int length)` for left-padding.

End of class Intermediate

Click here for example.

Examples

Each of the example below individually demonstrate the expected behaviours of the functions you need to implement. U stated otherwise, we will always start from Table 1.

```
bool result = courses.addAttribute("Semester", 1, "2020S");
```

`result` is `true`. And Table 1 becomes:

Department	Semester	Course Code	Course Name	Credits
CSE	2020S	COMP2012	Object Oriented Programming and Data Structures	4
CSE	2020S	COMP3021	Java Programming	3
CSE	2020S	COMP3711	Design and Analysis of Algorithms	3
CSE	2020S	COMP4511	System and Kernel Programming in Linux	3
CSE	2020S	COMP4641	Social Information Network Analysis and Engineering	3
MATH	2020S	MATH2343	Discrete Structures	4
MATH	2020S	MATH3033	Real Analysis	4
MATH	2020S	MATH4023	Complex Analysis	3

```
bool result = courses.addAttribute("Semester");
```

`result` is `true`. And Table 1 becomes:

Department	Course Code	Course Name	Credits	Semester
CSE	COMP2012	Object Oriented Programming and Data Structures	4	
CSE	COMP3021	Java Programming	3	
CSE	COMP3711	Design and Analysis of Algorithms	3	
CSE	COMP4511	System and Kernel Programming in Linux	3	
CSE	COMP4641	Social Information Network Analysis and Engineering	3	
MATH	MATH2343	Discrete Structures	4	
MATH	MATH3033	Real Analysis	4	
MATH	MATH4023	Complex Analysis	3	

```
bool result = courses.addAttribute("Semester", -2, "2020S");
```

`result` is `false`.

```
bool result = courses.addAttribute("Semester", 6, "2020S");
```

`result` is `false`.

```
string comp3511[] {"CSE", "COMP3511", "Operating Systems", "3"};
bool result = courses.addEntry(comp3511, 2);
```

`result` is `true`. And Table 1 becomes:

Department	Course Code	Course Name	Credits
CSE	COMP2012	Object Oriented Programming and Data Structures	4
CSE	COMP3021	Java Programming	3
CSE	COMP3511	Operating Systems	3
CSE	COMP3711	Design and Analysis of Algorithms	3
CSE	COMP4511	System and Kernel Programming in Linux	3
CSE	COMP4641	Social Information Network Analysis and Engineering	3
MATH	MATH2343	Discrete Structures	4
MATH	MATH3033	Real Analysis	4
MATH	MATH4023	Complex Analysis	3

```
courses.query().select();
```

prints out Table 1.

```
courses.query().where("Department", EQ, "CSE").select();
```

prints out the following:

Department	Course Code	Course Name	Credits
CSE	COMP2012	Object Oriented Programming and Data Structures	4
CSE	COMP3021	Java Programming	3
CSE	COMP3711	Design and Analysis of Algorithms	3
CSE	COMP4511	System and Kernel Programming in Linux	3
CSE	COMP4641	Social Information Network Analysis and Engineering	3

```
courses.query().where("Semester", EQ, "2020S").select();
```

prints out Table 1, since the attribute "Semester" is not found and thus `where()` returns the unmodified intermediate.

```
courses.query().orderBy("Department", DESCENDING).select();
```

prints out the following:

Department	Course Code	Course Name	Credits
MATH	MATH2343	Discrete Structures	4
MATH	MATH3033	Real Analysis	4
MATH	MATH4023	Complex Analysis	3
CSE	COMP2012	Object Oriented Programming and Data Structures	4
CSE	COMP3021	Java Programming	3
CSE	COMP3511	Operating Systems	3
CSE	COMP3711	Design and Analysis of Algorithms	3
CSE	COMP4511	System and Kernel Programming in Linux	3
CSE	COMP4641	Social Information Network Analysis and Engineering	3

Note: This sorts only the department. The order among CSE courses did not change. Same for MATH courses.

```
courses.query().limit(2).select();
```

prints out the following:

Department	Course Code	Course Name	Credits
CSE	COMP2012	Object Oriented Programming and Data Structures	4
CSE	COMP3021	Java Programming	3

```
string selector[] = {"Course Name", "Course Code"};
courses.query().select(selector, 2);
```

prints out the following:

Course Name	Course Code
Object Oriented Programming and Data Structures	COMP2012
Java Programming	COMP3021
Operating Systems	COMP3511
Design and Analysis of Algorithms	COMP3711
System and Kernel Programming in Linux	COMP4511
Social Information Network Analysis and Engineering	COMP4641
Discrete Structures	MATH2343
Real Analysis	MATH3033
Complex Analysis	MATH4023

Note: The order of provided attributes does not have to follow the order of attributes of the table.

```
courses.query().where("Course Name", CONTAINS, "Programming").update("Credits", "5");
courses.query().select();
```

prints out the following:

Department	Course Code	Course Name	Credits
CSE	COMP2012	Object Oriented Programming and Data Structures	5
CSE	COMP3021	Java Programming	5
CSE	COMP3711	Design and Analysis of Algorithms	3
CSE	COMP4511	System and Kernel Programming in Linux	5
CSE	COMP4641	Social Information Network Analysis and Engineering	3
MATH	MATH2343	Discrete Structures	4
MATH	MATH3033	Real Analysis	4
MATH	MATH4023	Complex Analysis	3

Note: For *all* the courses whose name contains "Programming", the credits are updated to 5.

```
courses.query()
.where("Department", EQ, "CSE")
.orderBy("Course Name", ASCENDING)
.orderBy("Credits", DESCENDING)
.limit(3)
.select();
```

prints out the following:

Department	Course Code	Course Name	Credits
CSE	COMP2012	Object Oriented Programming and Data Structures	4
CSE	COMP3711	Design and Analysis of Algorithms	3
CSE	COMP3021	Java Programming	3

End of Examples

Grading

- When we grade your assignment, your submissions will be compiled with `main.cpp`, `table.h` and `intermedia` that are *different* from what you have.
- We will inspect the *private members* of objects that you create and compare them with expected values.
  - For 2D array in table, make sure that `data[m][n]` is the n-th attribute of the m-th entry (index starts with 0) *the other way round*.
  - For the pointers in intermediates, e.g. `Intermediate::attrs`, we will verify its address, whether it points t `Table::attrs`.
  - For the doubly linked list of intermediates, please verify *all* pointers are correctly set. There should be no dan pointers, and forward and backward linked list traversal should work fine.
- For string output in `Intermediate::select()`, we will perform string matching. Therefore, ensure your output i *exactly* the same from the given sample output.
  - Make sure you removed *ALL* debug statements. We may deduct points if you failed to do so.
  - For tools that can do string matching, see diffchecker.
- Do not assume the size of tables. We may use *large* tables in grading.

End of Grading

Memory Leak

`drmemory` (Dr. Memory) is an open-source memory profiler. A feature that are interesting for this course is detection fo memory leaks. To enforce a high coding standard, *your submissions will be checked for memory leaks*.

Download `drmemory`

- For Windows users, we recommend you to download the portable version.
- For macOS users, click here. Double-click in Finder to decompress.
- For Linux users, click here.

Running `drmemory`

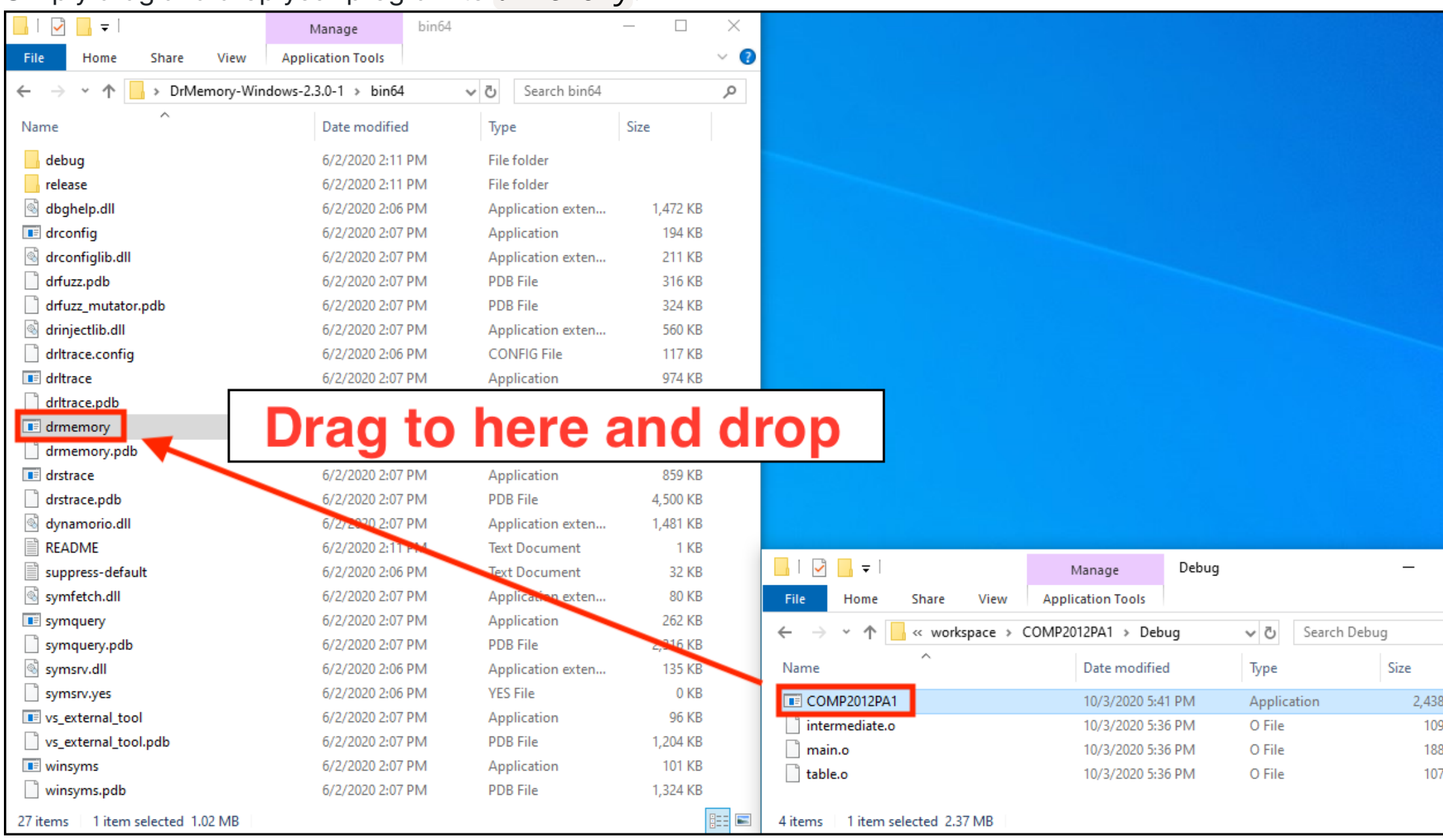
Notes on Eclipse User

If you are using a managed build project in Eclipse (not a Makefile project), you need to use static linking so that you can your program outside Eclipse.

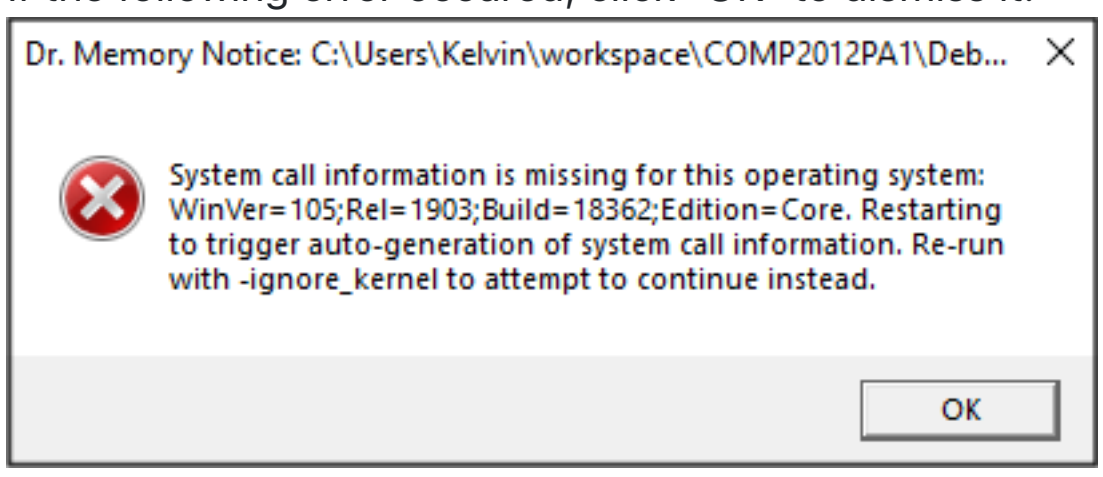
Goto "Project" -> "Properties" -> "C/C++ Build" -> "Settings" -> "Tool Settings" tab -> "MinGW C++ Linker" -> "Miscellaneous" -> type "-static" in the "Linker flags" box.

Running `drmemory` on Windows

1. Locate your program. If you are using Eclipse, then it is available at `"Eclipse_Workspace"\"Project_Name"\Debug\"Project_Name".exe`.
2. Locate `drmemory`. It is located at `bin64` folder after extraction.
3. Simply drag and drop your program to `drmemory`.



4. If the following error occured, click "OK" to dismiss it.



5. After a while, the output is shown in a text document.

Running `drmemory` on macOS

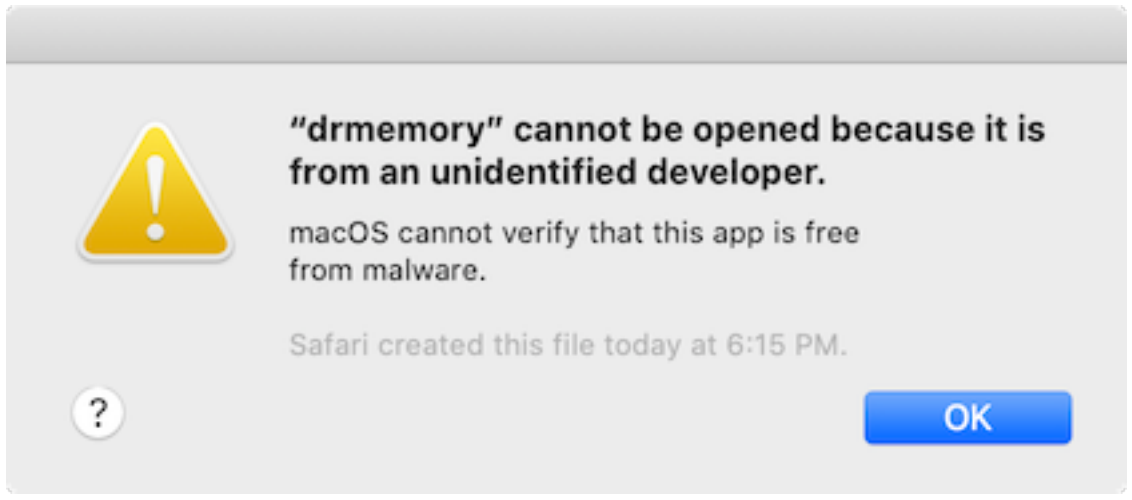
1. Find the path to your program. You can reveal its path by dragging it from Finder onto a terminal.
2. Find the path to `drmemory`.
3. Run the following command in a terminal.

```
path/to/drmemory -- path/to/your/program
```

For example:

```
/Users/kelvin/DrMemory-MacOS-2.3.0-1/bin64/drmemory -- /Users/kelvin/pa1/pa1
```

4. If the following alert pops up, dismiss it.



5. Then go to "System Preferences" > "Security & Privacy" > "General" > Click "Open Anyway". Rerun the command. this for all alerts.

Running `drmemory` on Linux

See the command for macOS. Make sure you run `bin64/drmemory` (NOT `bin/drmemory`!).

Result

At the end of the output, find the result for **leak(s)** and **possible leak(s)**:

```
~~Dr.M~~        0 unique,     0 total,      0 byte(s) of leak(s)
~~Dr.M~~        0 unique,     0 total,      0 byte(s) of possible leak(s)
```

If the number of bytes for *both* are 0, then your code is free from memory leaks. If the number of bytes are not zero, the code is prone to memory leaks. Note: You can ignore the bytes for *still-reachable allocation(s)*. They are not counted as memory leaks.

End of Memory Leak

Hints

Your submissions will be tested on all kinds of input, whether it is valid or not. Being able to produce the expected outpu not imply you will receive full marks in this programming assignment. Make sure you followed the specifications of each function carefully.

You can safely assume the following when you complete this programming assignment:

- No two attributes in a table are the same.
- In `Table::addEntry()`, the entry being added has size equal to the number of attributes in the table.
- In `Table::select()`, the given attributes exist in the table.
- When `Intermediate::Intermediate(const Table &table)` is called, `table` has at least one attribute.
- After an intermediate is created, the table will NOT be modified by ANY member functions of Table (including dest until the intermediate is not used.

Complete `class Table` before `class Intermediate`. You can implement `Table::print()` function to print out th table.

Here are some tips to check for memory leak.

- Limit the number of classes constructed and functions called, by commenting out the examples in `main()`. Then `valgrind` again. This will help you to pinpoint which classes and functions are the sources of memory leak.
- Make sure a `delete` follows every `new`.
- If you are going to modify pointers, make sure the reference to previous block of memory is not lost.

Double deletion refers to calling `delete` multiple times on the same block of dynamic memory. Here are some tips to av this.

- Be clear about the ownership of dynamic memory. Only the class that owns a particular block of dynamic memory call `delete`.
- Pay attention when using pointer variables in functions. Do not call `delete` unless there is a reason to.

End of Hints

Submission and Deadline

Deadline: 23:59:00 March 28, 2020

Canvas Submission

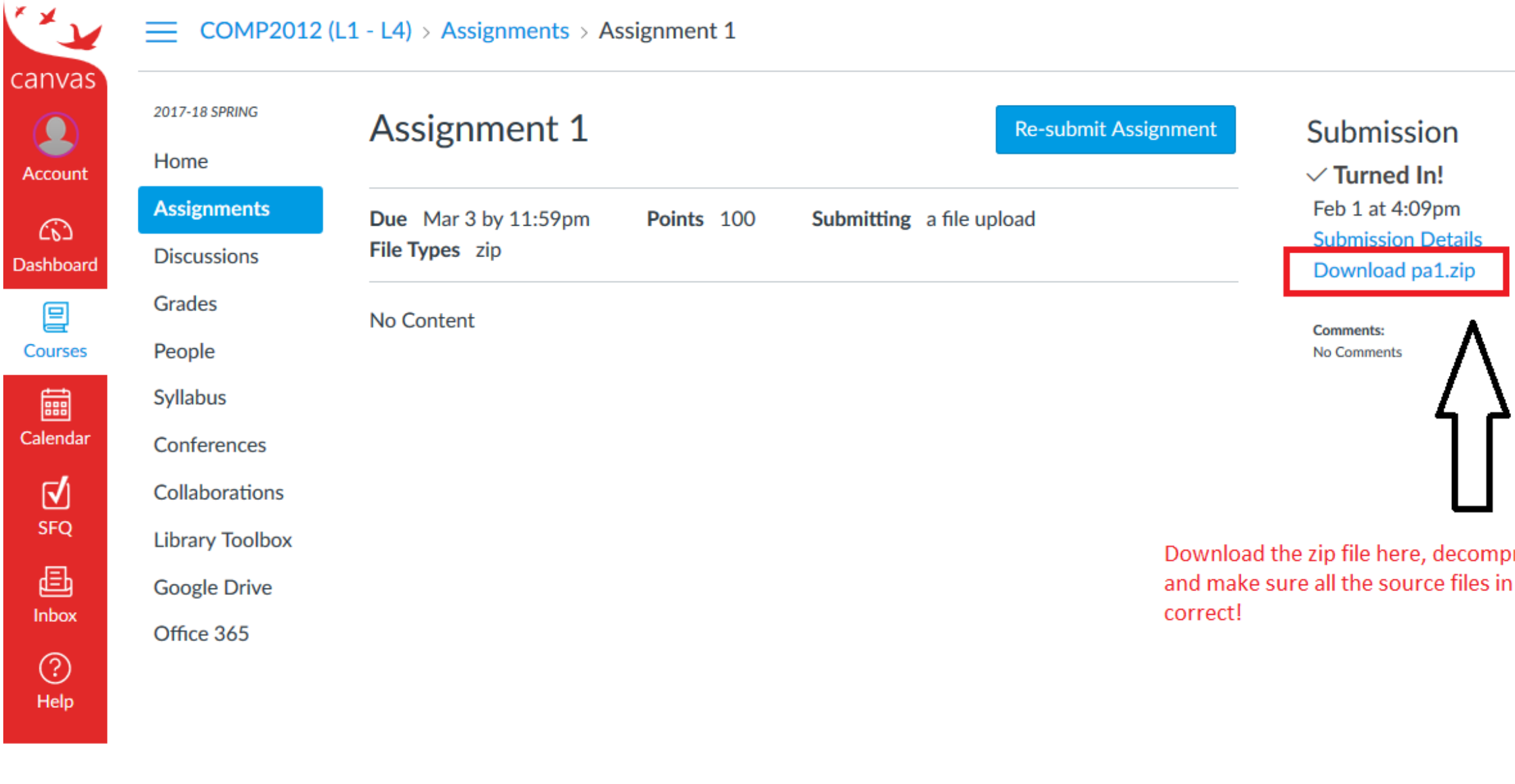
Create a zip file that contains the **2 .cpp files**: `table.cpp` and `intermediate.cpp`. The zip file should be named as **p**
Submit only **pa1.zip** through the **Canvas Submission Page**:

- Submission Page for L1.
- Submission Page for L2.
- Submission Page for L3.
- Submission Page for L4.

The filenames have to be exactly the same. It must be a zip file, not rar, not 7z, not tar, not gz, etc. Inside the zip file, t
cpp files need to have correct names. If your submissions cannot be uncompressed, they will not be graded.

Make sure your source files can be compiled. If we cannot compile your submissions, they will not be graded. Therefore
should at least put in dummy implementations to the parts that you cannot finish so that there will be no compilation err

Make sure you actually upload the correct version of your source files - we only grade what you upload. Some stu
in the past submitted an empty file or a wrong file or an exe file which is worth zero mark. So **you must download and c
check the files you have submitted**.



You may submit your file multiple times, but only the latest version will be graded.

Submit early to avoid any last-minute problem. Only canvas submissions will be accepted.

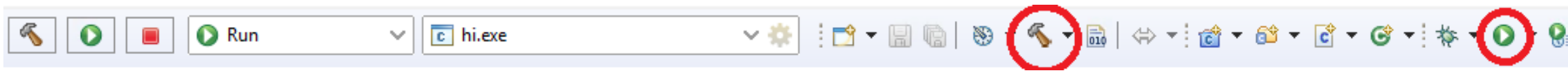
Note 1: If you have no idea how to create a zip file, you may see How to create a zip file in Windows 10 or How to create
file in Mac OS X . Try to zip just the 2 cpp files, not a folder containing the 2 files.

Note 2: Canvas may append a number to the filename of the file you have submitted. e.g. pa1-1.zip. It is OK as long as yo
named your file as pa1.zip when you submit it.

Using Virtual Barn

It is **required** that your submissions can be compiled and run successfully in our official Windows Eclipse which can be
downloaded from the "Using Eclipse at home (Windows)" section here. You need to unzip the zip file first, and then run t
Eclipse program extracted from it. Do not just double-click the zip file. If you have used other IDE/compiler/OS (including
macOS Eclipse) to work out your solution, you should test your program in the aforementioned official environment befo
submission. This version of Eclipse is also installed on our lab machines.

If you have no access to a standard Windows machine, you may remote control a Windows machine in [HKUST virtual ba
Choose the "Programming Software" server, and you will find Eclipse shortcut on the desktop. This is a newer version of
Eclipse with a slightly different interface. However, its compiler has the same version as ours and can be used to verify it
program can be compiled by us for grading. In particular, to create a new C++ project, you can do so from "File" menu -
"New" -> "Project..." -> "C++ project" -> Type a project name -> Choose MinGW compiler -> "Finish". Also, to build the
project, save your files, then use the Hammer and Run buttons, circled in the following screenshot (NOT the ones on the



Late submission policy

There will be a penalty of -1 point (out of a maximum 100 points) for every minute you are late. For instance, since the de
of assignment 1 is 23:59:00 on Mar 28, if you submit your solution at 1:00:00 on Mar 29, there will be a penalty of -61 po
your assignment. However, the lowest grade you may get from an assignment is zero: any negative score after the dedu
due to late penalty (and any other penalties) will be reset to zero.

End of Submission and Deadline

FAQ

- Q: What do I need to do when I call `addEntry()` when the table has no attributes (`numAttrs == 0` and `attrs == nullptr`)?
- A: You don't need to consider this. It is not tested.

- Q: Can I use extra libraries?
- A: No. You just need to use `iostream`.

- Q: Can I create helper functions?
- A: Yes. Just put them along your code in `table.cpp` and `intermediate.cpp`.

End of FAQ

About Plagiarism

If you get caught cheating, both you and the other person will get zero for the assignment. It does not matter how much
copied. According to HKUST rules, both of you have the same penalty. There may also be additional penalties other than
getting zero for the assignment. You are responsible for keeping your own work safe from being accessed or "reference
others in any way. However, discussing with others is fine as that's different from direct copying. Also be reminded that
plagiarism detection software is very robust nowadays. On top of that we will also examine your submissions manually.

End of Plagiarism

Menu

- Introduction
- Download
- Overview
  - class Table
  - class Intermediate
- Examples
- Grading
- Memory Leak
- Hints
- Submission & Deadline
- FAQ
- About Plagiarism

Page maintained by

Kelvin Chiu
Email: kelvinchiu@cse.ust.hk

Last Modified: 03/20/2020 16:01:0403/19/2020 12:19:56

Homepage

Course Homepage