The focus of these problems will be working with information extracted from a municipal government data feed containing bids submitted for auction of property. All materials for this lab assignment can be found in the Supporting Materials section below. The data set is provided in two comma-separated files:

1. eBid\_Monthly\_Sales.csv (larger set of 12,023 bids)
2. eBid\_Monthly\_Sales\_Dec\_2016.csv (smaller set of 76 bids)

This assignment is designed to explore linked lists, so you will implement a singly linked list to hold a collection of bids loaded from a CSV file. We provide a starter console program that uses a menu to enable testing of the hash table logic you will complete. It also allows you to pass in the path to the bids CSV file to be loaded, enabling you to try both files. In this version, the following menu is presented when the program is run:

**Menu:**

* 1. **Load Bids**
  2. **Display All Bids**
  3. **Find Bid**
  4. **Remove Bid**
  5. **Exit**

**Enter choice:**

The HashTable.cpp program is partially completed. It contains empty methods representing the programming interface used to interact with a hash table. You will need to add logic to the methods to implement the necessary behavior. Here is the public API for HashTable.cpp that you have to complete:

public:

HashTable();  
virtual ~HashTable();  
void Insert(Bid bid);  
void PrintAll();  
void Remove(string bidId);  
Bid Search(string bidId);

**Prompt**

You will need to perform the following steps to complete this activity:

**Setup:** Begin by creating a new C++ project with a project type of "Hello World C++ Project".

1. Name the project “HashTable”. Remember to pick the correct compiler in **Toolchains** and click **Finish**. This will create a simple HashTable.cpp source file under the **/src** directory.
2. Download the starter program files and copy them to the project’s **/src** directory, replacing the existing auto-generated ones. Remember to right-click on the project in the Project Explorer pane on the left and **Refresh** the project so it adds all the new files to the **src** folder underneath.
3. Because this activity uses C++ 11 features, you may need to add the **-std=c++11** compiler switch to the miscellaneous settings.

**Task 1:** Define structures to hold bids. Hint: You may choose either an array or a vector for storage. Note that you may be able to reuse portions of your code from previous assignments to save you time. Look for places you have implemented vectors for storage or a Node structure for a linked list. Reusing code from these labs may save you time.

**Task 2:** Initialize the structures used to hold bids.

**Task 3:** Implement code to free storage when a class is destroyed.

**Task 4:**Implement code to calculate a hash value using the bid ID as the source for calculating the key.

**Task 5:** Implement code to insert a bid. Be sure to check for key collisions and use the chaining technique with a linked list to store the additional bids.

**Task 6:**Implement code to print all bids.

**Task 7:**Implement code to remove a bid.

**Task 8:**Implement code to search for and return a bid.

Here is sample output from running the completed program:

> ./HashTable ~/Downloads/eBid\_Monthly\_Sales\_Dec\_2016.csv  
> HashTable.exe Downloads\eBid\_Monthly\_Sales\_Dec\_2016.csv

**Load bids from CSV and display the hash table contents:**

|  |  |  |
| --- | --- | --- |
| **Example Input** | **Choice: 1** | **Choice: 2** |
| **Display** | Menu: 1. Load Bids 2. Display All Bids 3. Find Bid 4. Remove Bid 9. Exit Enter choice: 1 | Menu: 1. Load Bids 2. Display All Bids 3. Find Bid 4. Remove Bid 9. Exit Enter choice: 2 |
| **Output** | Loading CSV file eBid\_Monthly\_Sales.csv  76 bids read time: 3069 clock ticks time: 0.3069 seconds | Key 2: 98094 | Credenza | 57 | General Fund 2: 98273 | Nike Tennis Shoes Size: 11.5 | 84 | Enterprise Key 5: 98276 | Nike Tennis Shoes Size: 11.5 | 83.99 | Enterprise Key 8: 98279 | Nike Tennis Shoes Size: 11 | 51.57 | Enterprise Key 10: 98102 | Battery Cart | 42 | Enterprise Key 12: 98104 | 3 Ticket Booths | 395.01 | Enterprise 12: 98283 | Jordan Tennis Shoes Size: 11 | 160 | Enterprise Key 13: 98105 | 2 PS4 Games | 11 | Enterprise 13: 98284 | Jordan Tennis Shoes Size: 11 | 89.01 | Enterprise ... ... Key 176: 98268 | Dayton Pallet Jack | 78.85 | Enterprise Key 177: 98269 | 5 Extron Control Systems | 25 | General Fund |

Note that Keys 2, 12, and 13 highlighted above indicate that key collisions occurred.

**Finding and Removing an existing bid:**

|  |  |  |
| --- | --- | --- |
| **Example Input** | **Choice: 3** | **Choice: 4** |
| **Display** | Menu: 1. Load Bids 2. Display All Bids 3. Find Bid 4. Remove Bid 9. Exit Enter choice: 3 | Menu: 1. Load Bids 2. Display All Bids 3. Find Bid 4. Remove Bid 9. Exit Enter choice: 4 |
| **Output** | 98129: Printer | 52.00 | Enterprise  time: 59 clock ticks time: 5.9e-05 seconds | {no output shown} |

**Finding a bid that no longer exists:**

|  |  |  |
| --- | --- | --- |
| **Example Input** | **Choice: 3** | **Choice: 9** |
| **Display** | Menu: 1. Load Bids 2. Display All Bids 3. Find Bid 4. Remove Bid 9. Exit Enter choice: 3 | Menu: 1. Load Bids 2. Display All Bids 3. Find Bid 4. Remove Bid 9. Exit Enter choice: 9 |
| **Output** | Bid Id 98129 not found.  time: 12 clock ticks time: 1.2e-05 seconds | Good bye. |

Your submission must address the following rubric criteria:

* **Code Reflection:** A brief explanation of the code and its purpose, and a brief discussion of your experience in developing it, including any issues that you encountered while completing the exercise and what approaches you took to solve them
* **Pseudocode or Flowchart:** A pseudocode or flowchart description of the code that is clear and understandable and captures accurate logic to translate to the programming language
* **Specifications and Correctness:** Source code must meet its specifications and behave as desired. Correct code produces the correct output as defined by the data and problem; however, you should also produce fully functioning code (with no errors) that aligns with as many of the specifications as possible. You should write your code in such a way that the submitted file executes, even if it does not produce the correct output. You will be given credit for partially correct output that can be viewed and seen to be partially correct.
* **Annotation / Documentation:** All code should also be well-commented. This is a practiced art that requires striking a balance between commenting everything, which adds a great deal of unneeded noise to the code, and commenting nothing. Well-annotated code requires you to:
  + Explain the purpose of lines or sections of your code, detailing the approach and method you took to achieve a specific task in the code.
  + Document any section of code that is producing errors or incorrect results.
* **Modular and Reusable:** Programmers should develop code that is modular and reusable. If it contains functionality and responsibility in distinct methods, code is more flexible and maintainable. Your code should adhere to the single responsibility principle—classes and methods should do only one job. If you can replace a method with another that uses a different technique or implementation without impacting (having to refactor or rewrite) other parts of your code, then you have succeeded in creating modular methods.
* **Readability:** Code needs to be readable to a knowledgeable programmer. In this course, readable code requires:
  + Consistent, appropriate whitespace (blank lines, spaces) and indentation to separate distinct parts of the code and operations
  + Explicit, consistent variable names, which should clearly indicate the data they hold and be formatted consistently: for example, numOrders (camelCase) or item\_cost (underscored)
  + Organized structure and clear design that separates components with different responsibilities or grouping-related code into blocks

**Guidelines for Submission**

To complete this lab assignment, submit the **CPP code files** and a **code reflection and associated pseudocode or flowchart**. Your written portion should be 1–2 paragraphs in length.

**Supporting Materials**

[**CS 300 Hash Table Assignment Student Files.zip**](https://learn.snhu.edu/content/enforced/1160013-CS-300-H7580-OL-TRAD-UG.22EW1/course_documents/CS%20300%20Hash%20Table%20Assignment%20Student%20Files.zip?_&d2lSessionVal=gZv9xjNPWWmJazBEbhFr3N0yS&ou=1160013)  
Download this zipped file folder to begin your lab assignment. The data sets you will use in this assignment are provided in these comma-separated files:

* eBid\_Monthly\_Sales.csv (larger set of 12,023 bids)
* eBid\_Monthly\_Sales\_Dec\_2016.csv (smaller set of 76 bids)
* HashTable.cpp program, which is a partially completed program that you can use as a starting point for the assignment

| **Module Four Assignment Rubric** | | | | |
| --- | --- | --- | --- | --- |
| **Criteria** | **Proficient (100%)** | **Needs Improvement (70%)** | **Not Evident (0%)** | **Value** |
| **Code Reflection** | Describes purpose of code, techniques implemented to solve problem, challenges encountered, and approaches to overcome the challenges | Lacks details in code purpose, techniques implemented, or challenges encountered | Does not explain purpose of code, techniques used, or challenges encountered | 25 |
| **Pseudocode or Flowchart** | Pseudocode or flowchart is clear and understandable and captures accurate logic to translate to the programming language | Pseudocode or flowchart has errors or omissions that affect its clarity or understandability, or the logic to translate to the programming language is inaccurate or incomplete | Pseudocode or flowchart does not contain the logic to translate to the programming language | 10 |
| **Specifications and Correctness: Algorithm** | All algorithm specifications are met completely and function in all cases | Details of the specifications are violated, or program often exhibits incorrect behavior | Program only functions correctly in very limited cases or not at all | 20 |
| **Specifications and Correctness: Data Structure** | All data structure specifications are met completely and function in all cases | Details of the specifications are violated, or program often exhibits incorrect behavior | Program only functions correctly in very limited cases or not at all | 20 |
| **Annotation / Documentation** | Code annotations explain and facilitate navigation of the code | Comments provide little assistance with understanding the code | Code annotations do not explain the code or do not facilitate navigation of code, or code is not fully or logically annotated | 10 |
| **Modular and Reusable** | Methods are limited in scope and responsibility, and both algorithms and data structures are implemented in such a way that they can be reused in other programs | Methods have errors in scope or responsibility, or algorithms or data structure are overly tied to the specific program | No attempt was made to develop modular or reusable code | 10 |
| **Readability** | Code follows proper syntax and demonstrates deliberate attention spacing, whitespace, and variable naming | Code contains variations from established syntax and conventions | Code contains significant variations from established syntax and conventions | 5 |
| **Total:** | | | | 100% |