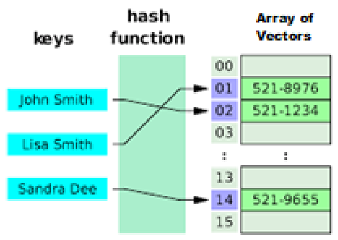
CSCI 2212/6610

Final C++ Program: A Minimal Hash Table

1 Goals



- Write a complex program in C++.
- With your partner, add a hash table to your program for Programming Project Part 1 Class Application.
- To implement this two-dimensional hash table using an array of vectors.
- To use bitwise operators to calculate the index of the vector in hash table::hashIt().
- Make the appearance of your output neat and readable on the console and in the file.

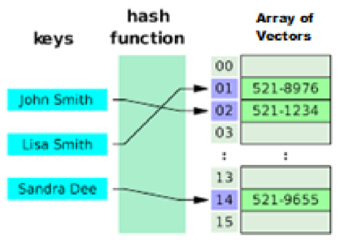
I would like you to use the class project you just did and add a hash table so it can handle large volumes of data. To do this we will create a hashed index to reach a vector that contains a subset of the data. This will make searching large datasets more efficient. The user should be able to request all the objects in the hash table with a specific date from your program (remember I told you to have a date in yourClass's data members).

The important part of the program is HOW YOU DO THIS, not that your program can find the objects I am looking for by date. There are easier ways to do that, in fact, you already did in program 1.

- You must create and use an **array of vectors of your class (the hash table)**. This means you have a two-dimensional array where you have pointers that point at a separate vector for each row. The syntax to create it is:
  ```
  vector<yourClass> hashTable[number of rows];
  ```
- **To access one vector** from the array of vectors you would reference using a square bracket, e.g. `hashTable[index].push_back(myobject);`
- You must use **bitwise operations** to use the date to **calculate the index of the vector** you will search in your hash table (an array of vectors of your class, the data). **You will use a simple encryption algorithm on the date,** put it in in main() and call it hashIt().
- Note that multiple dates will hash to the same index (row/vector) within the hash table. 365 days mapping to no more than 97 vectors, so there will be data with 3-4 different dates in each row/vector in the hash table.
- You must search the row/vector in the hash table at the calculated index and print only the items that have the exact date you input. Use yourClass::isMyDate() to check the date in each object in the vector. To access one object within the hash table you can use double square backets e.g. `hash table[vectorIndex][objectIndex].isMyDate(); //this would call isMyDate().`
- You must **validate all user input** – whether it is a menu choice, data for the object, or the date to find, you need to check the input and **make sure they entered valid data**.
  - Check that the data is a value of the **right type.**
  - Is it **within the range of values that are correct for your program.**
  - **If the user made an error,** give the user feedback, and ask them to re-enter incorrect input (give clear instructions). Put this in a loop that repeats until the input is valid. It is nice to give the user the option to quit the program in your validation loop. Be clear, nobody wants to get stuck in an infinite loop of "`Bad data, re-enter your answer: `".

Final C++ Program: A Minimal Hash Table

- Note: String and char and sstream all have useful functions to help you do this. It takes a lot of time, but it is essential for professional coding.



Division of the work is up to you. It needs to be clearly who did what in this project. If you are unsure and want my ideas, you can use the following division of labor, but your team needs to communicate and make sure it is fair.
One option is:

- **Partner one** can be responsible for the changes to change your vector to a Hash Table and change all the loops to handle an array of vectors instead of one vector (this will require nested loops or using the method hashIt(targetDate) to identify the relevant vector's index.
- **Partner two** can write the hashIt(targetDate) method to use bitwise operations to calculate the hashed vector index to access the hash table and add these changes where needed in main().
- **You should not need to make any changes to your class if you followed the instructions for Program .**

Resources: For help understanding a hash table, here is one resource I found with a quick google search. The vector is replacing the linked list for the hash table. This gives you the concept of a hash table without the implementation difficulties of creating a linked list using structure and pointers, save that for your data structures class. https://yourbasic.org/algorithms/hash-tables-explained/

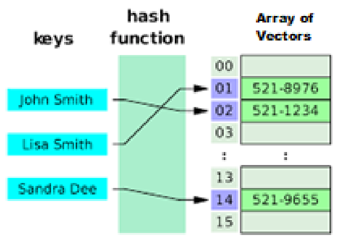
2 The Hash Table

You will build a data structure called a hash table. Data in the hash table is not in sorted order, but it is very fast to put a new item into the table and very fast to find it later. Note: The hashIt(targetDate) method will use an algorithm to map data to the proper vector in the hash table. This algorithm is a very simple encryption. It is too simple for use in data security today but is a good starting place to understand how encryption works.

You will need to use your implementation of these things from Program 1:

- Your class. (the .hpp and .cpp), you should not need any changes to this code, but you can improve it (refactor) if you want to. **I will refer to the class you created in Program 1 as yourClass in these requirements.**
- Your main() program (main.cpp or whatever you called it in Program 1) with changes to use the hash table. You will need to add or modify the following in your main() to make it work:
    - Add the array of vectors as described above.
    - Add a hashIt(targetDate) function to use the date to calculate the vector index (a number between zero and the number of rows in your vector) using the algorithm below).

Final C++ Program: A Minimal Hash Table



- Update the part that reads the data from the file and saves it in your hash table to use hashIt(targetDate) to calculate the vector's index/row in the hash table for the object.
- Modify the loop/function that prints all the data from the hash table to the console. This will need to be a nested loop to go through all the row/vectors and print their data.
- Similar changes are needed to print the data from the hash table to a file.
- You can add other functions if you want, such as a function to display the menu and take user input.

3 Implementation Details.

Using your class that stores the data you created of various types (string(s), number(s) and a date).

1. Use your class constructor with arguments to create objects of your class to put in the vector at the calculated index.
2. Using yourClass::isMyDate(targetDate) method with a Boolean return type. Return true if the object's date matches the argument targetDate and false if it does not.
3. Use yourClass::toString() method with a string return type. Returns a string with the data formatted for output: nice, clear and lined-up so it looks good.
4. Use yourClass::toFile() method to format the data to store it in a file and read it back into the array of vectors later on.

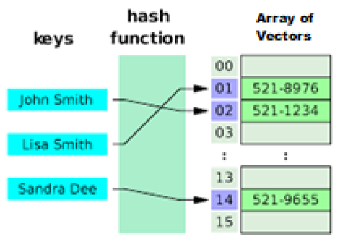
The Main Program.

Make these changes to add the hash table, which is a 2D array of vectors of yourClass from program 1 (the data). Do the following things in your main program:

- Make the changes above to add the hash table, which is a 2D array of vectors of yourClass from program 1 (the data).
- Read data from the file into the hash table.
- Enter a processing loop:
  - Show a menu with the same options as program 1.
  - Change the implementation of the options to use the data and functions of the new hash table.
  - Provide a simple and convenient way for the user to end the program, such as a quit option on a menu or simply entering quit.
- Save the data to the file.
- Say bye before your quit.

1. **Create a hash table in main()** as an array of vectors that can store yourClass objects and the length of the vector array (Hint: you can declare a constant with #define to hold the number of rows). *Choose an array length that is a **prime number between 53 and 97.***
2. **Read the file** one line at a time and store the data in your hash table, this works best in a function.

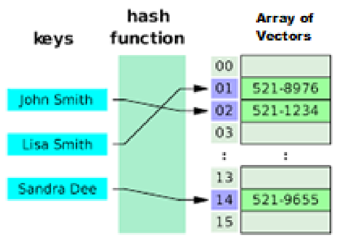
Final C++ Program: A Minimal Hash Table



- Open the data file (data.txt) in the constructor, do this in a professional way and close the file when you are finished.
- Read the data as you did in Program 1, for each object do the following:
    - Use the hashIt(targetDate) function with the object's date as its argument to calculate the index of the vector in the hash table to add this object to.
    - Add the object of your class to the correct hash table vector using the Vector::push_back() function. E.g `table[index].push_back(yourClass(1,2,”string”));`
        - Create objects of your class using the yourClass::yourClass(…) constructor with arguments containing the data (Hint: You did this in program 1).
- Close the file when you are done.

3. **Add a new object** of your class in the hash table, this also works best as a function.
    - Collect the date for the object the same way you did in Program 1. The object will be stored in the hash table.
    - Use the hashIt(targetDate) function with the object's date as the argument to calculate the **index** of the vector in the hash table (row).
    - Use the vector's index to select one vector from the hash table and push the yourClass object onto it. e.g. `hash table[index].push_back(myobject);`
    - All the objects with dates that translate to that index (more than one date will result in the same hash) will go into the same vector in the hash table (2D array).
4. **Find all the objects with the same date**.
    - Use the hashIt(targetDate) function to get the index of the vector for the targetDate.
    - Since it is possible for different dates to end up in the same vector, you need to *look at every object in the vector and find out if its date matches the one you are looking for*.
        - Go to that vector at the correct row. Loop through this vector's columns (these are the objects) and check the date for each object.
            - Check that object's date by calling yourClass::isMyDate(targetDate) method with the object and the date as an argument. Hint: You did this in Program 1. Use double square brackets e.g. `hash table[vectorIndex][objectIndex].isMyDate(targetDate); //this would call isMyDate(targetDate).`
            - If the date matches, print out the object's info using yourClass::toString() method.
5. **Create a hashIt(targetDate) function** that takes a date as the argument and returns a random-looking integer between 0 and (number of rows in the array of vectors -1). **Use this algorithm with your object's date when adding an object to the hash table and**

Final C++ Program: A Minimal Hash Table



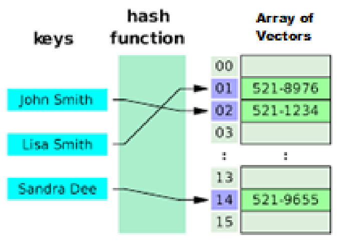
the targetDate when you are trying to find objects to print. It returns the index of the hash table's row (a vector):

a. Create a long integer named `key` and initialize it to the **year squared** plus **day**. (Hint: x squared is x^2). Also create a `temp` variable of the same type initialized to zero.
b. Then **enter a loop**.
   i. Use a mask to get the 5 high-order bits of key (5 leftmost bits) and save them into the `temp` variable.
   ii. Shift these bits to the five low order bits (5 rightmost bits) of the `temp` variable.
   iii. Shift the `key's` 5 bits toward the left. This will drop those bits off the end and make the 5 low order bits all zero.
   iv. Use bitwise OR of `temp` with `key` to put the 5 bits that you saved in the `temp` variable back into the `key` in the low order 5 bits.
   v. Get the name of the month from the date and save it in a string. Then cast the last letter of the month to an int and xor (^) it with `key` and save it as the `key`. *Hint: there are built in functions to **find the day of the week from a date** in C++.*
   vi. **Repeat steps i. to v. in the loop until** you have **used all the letters from the month** (going backward from the last letter to the first letter). E.g. For January, start with `y`, then `r`, then `a`, then `u`, then `n`, then `a`, and `J` is last.
c. After the loop, calculate `index = key % the length of your array of vectors` (prime number you chose as the number of rows). You should get a number between 0 and (that prime number -1).
d. Return the `index` from this method.

6. **Write all the objects' data to the file** from each row and column of the hash table.
   - Open the data file (data.txt) for output, do this in a professional way.
   - Write the data to the file from each object in the hash table. Put the data from one object on one row of the file like you did before. *Hint: Do not try to "save" the location of the object in the hash table, just calculate it again when you read the file.*
   - *Use a loop* to walk through the array's rows and print each object in each vector (columns). Note: `table[row]` is one vector in the hash table.
     - You can use double square backets with the hash table (because it's a 2D, an array of vectors) to access each yourClass object at a specific row and column. E.g. `table[row][col]` is one object of yourClass.
     - Some vectors will be empty, the vector::size() method tells you how many objects are in the vector, zero means it is empty. Use `table[row].size();`
     - Use the yourClass::toFile() method to format your data for the file. E.g. `table[row][col].toFile();//returns a string`
     - Add each object to the file as you did in Program 1.
   - Close the file when you are done.
7. **End your program gracefully**, and don't forget to say bye.

Final C++ Program: A Minimal Hash Table

4 Code Demo and Presentation Video



Create a 5 to 8-minute video with your partner of your presentation of your project classes and code and demo showing your code running. You can record to the cloud using zoom. In this video each of you should walk us through the part of the program you wrote that you are most proud of, show the code and explain how you figured out how to solve the problem. Then show it in action by running a test case that shows what you explained.

Submit any documents or slides you used for your presentation. This is a major grade, and all work should be your best work and done at the using the skills you learned in Technical Communication for Computing and your Academic Research and Writing courses. I shared links on making your presentation great in Canvas.

Optional challenge for extra credit (up to 4 points in total, depending on elegance and quality of implementation) is to Make hash table into a class, this requires significant changes to your Program 1, **see my separate requirements**.

5 Testing and submission.

If you understand the directions and follow them carefully, you will learn about a useful data structure in computing. **Everything should be submitted to your group project in Canvas except the peer evaluation which must be done individually.**

- Write a test plan and follow each case in separate runs of your program. Put the runs in a text file with each one clearly labeled followed by its console output. Your test must include April 10, 2021 as a find(targetDate). Call the file testPlan.txt. You can modify your test plan from Program 1 appropriately for this project. Make sure you test all code and options and every requirement.
- Turn in your code .hpp or .h, and .cpp files and your data.txt file with the final version of the data.
- **Give me the link to your Code Demo Presentation Video** (*do not upload a video into canvas*). Upload a narrative or slides you used to make your presentation. This is a major grade, and all work should be your best work and properly done.
- In addition, each of you will evaluate your own and your partner's performance on the project on taking a fair share of the work, meeting their commitments, doing a good job, communication, and teamwork. The questions and criteria that will be used are in Canvas, you will each do this alone. I will take your feedback seriously and combine it with my own observations as the teamwork part of your project grade. Your coding evaluation is based on your own work.