1.Scenario:

You are implementing a simple hash table using linear probing for a library system. The hash table represents the availability of books in the library. Each book is identified by a unique integer key. The library has a fixed number of shelves, and each shelf corresponds to an index in the hash table.

**Input Format:**

The first line contains an integer M representing the number of shelves in the library.

Subsequent lines contain operations to be performed on the hash table. Each operation is represented by a string ("INSERT" or "SEARCH") followed by an integer key.

**Output Format:**

For each "SEARCH" operation, output "true" if the book is available on a shelf, and "false" otherwise.  
  
CODE:

import java.util.Scanner;

class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int M = scanner.nextInt();

int[] hashTable = new int[M];

while (scanner.hasNext()) {

String operation = scanner.next();

if (scanner.hasNextInt()) {

int x = scanner.nextInt();

if (operation.equals("INSERT")) {

int index = x % M;

while (hashTable[index] != 0) {

index = (index + 1) % M; // Linear probing

}

hashTable[index] = x;

} else if (operation.equals("SEARCH")) {

int index = x % M;

while (hashTable[index] != x && hashTable[index] != 0) {

index = (index + 1) % M; // Linear probing

}

System.out.println(hashTable[index] == x);

}

} else {

// Handle the case where the expected integer is not provided

System.out.println("Invalid input. Expected an integer.");

scanner.next(); // Consume the invalid input to avoid an infinite loop

}

}

}

}

TESTCASES

### Test Case 1:

\*\*Input:\*\*

4

INSERT 8

INSERT 4

SEARCH 12

INSERT 12

SEARCH 12

\*\*Output:\*\*

false

true

### Test Case 2:

\*\*Input:\*\*

6

INSERT 9

INSERT 3

INSERT 15

SEARCH 9

INSERT 21

SEARCH 21

\*\*Output:\*\*

true

true

### Test Case 3:

\*\*Input:\*\*

5

INSERT 5

INSERT 13

INSERT 21

SEARCH 5

SEARCH 21

SEARCH 16

\*\*Output:\*\*

true

true

false

### Test Case 4:

\*\*Input:\*\*

3

INSERT 7

INSERT 10

SEARCH 1

SEARCH 10

\*\*Output:\*\*

false

true

### Test Case 5:

\*\*Input:\*\*

7

INSERT 14

INSERT 21

SEARCH 7

SEARCH 14

INSERT 7

SEARCH 7

\*\*Output:\*\*

false

true

true

### Test Case 6:

\*\*Input:\*\*

8

INSERT 10

INSERT 15

SEARCH 10

SEARCH 12

\*\*Output:\*\*

true

false

2.Question:

You are developing a program that uses a simple hash function to map an integer key to an index in an array. The array is populated with user-input elements. Afterward, the program prompts the user to enter an integer key. If the entered key is present in the array, the program applies the hash function and prints the resulting index. Otherwise, it notifies the user to enter a key that is present in the array.

**Input Format:**

The user is initially prompted to enter the size of the array (an integer).

The program then expects the user to input the array elements (integers) one by one.

Following that, the user is asked to enter an integer key.

**Output Format:**

If the entered key is present in the array, the program prints: "The key [key] is hashed to index [index]."

If the entered key is not in the array, the program prints: "The entered key is not in the array. Please enter a key present in the array."  
  
**CODE :**

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

//System.out.print("Enter the size of the array: ");

int arraySize = scanner.nextInt();

// Create an array of the given size

int[] arr1 = new int[arraySize];

// Populate the array with user input

// System.out.println("Enter " + arraySize + " elements for the array:");

for (int i = 0; i < arraySize; i++) {

//System.out.print("Enter element at index " + i + ": ");

arr1[i] = scanner.nextInt();

}

int key;

while (true) {

// System.out.print("Enter an integer key that is present in the array: ");

key = scanner.nextInt();

// Check if the key is present in the array

boolean keyInArray = false;

for (int i = 0; i < arraySize; i++) {

if (arr1[i] == key) {

keyInArray = true;

break;

}

}

if (keyInArray) {

break;

} else {

System.out.println("The entered key is not in the array. Please enter a key present in the array.");

}

}

// Use the simple hash function to find the remainder when dividing the key by the array size

int index = hashFunction(key, arraySize);

System.out.println("The key " + key + " is hashed to index " + index);

scanner.close();

}

private static int hashFunction(int key, int arraySize) {

// Simple hash function: take the remainder when dividing the key by the array size

return key % arraySize;

}

\*\*Test Case 1:\*\*

Input:

4

15

27

42

10

42

Output:

The key 42 is hashed to index 2

\*\*Test Case 2:\*\*

Input:

3

8

16

24

10

Output:

The key 10 is hashed to index 1

\*\*Test Case 3:\*\*

Input:

2

5

10

7

Output:

The key 7 is hashed to index 1

\*\*Test Case 4:\*\*

Input:

6

30

45

60

75

90

105

Output:

The entered key is not in the array. Please enter a key present in the array.

\*\*Test Case 5:\*\*

Input:

5

2

4

8

16

32

64

Output:

The key 64 is hashed to index 4

\*\*Test Case 6:\*\*

Input:

1

7

7

Output:

The key 7 is hashed to index 0  
  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3.Alice works at a library and needs to efficiently catalog a list of books. Each book has a unique ID number. To quickly access information about a specific book, she decides to implement a hash table using linear probing. However, to prevent the library from having to constantly reallocate memory as the collection grows, the hash table's size is fixed at 20.  
  
**INPUT FORMAT :**

Help Alice insert the following book IDs into the hash table: 12, 17, 35, 28, 10, 5, 9, 14, 42, 31. Assume a simple modulo function h(key) = key % 20 for generating the initial index.

After each insertion, print the updated hash table state, highlighting the probing sequence if a collision occurs.

After all insertions, analyze the performance of the hash table based on the number of probing steps per operation.  
  
**Output Format:**

For each insertion, print:

The book ID being inserted.

The initial index calculated using the hash function.

If a collision occurs, print the complete probing sequence until an empty slot is found.

Finally, print the updated hash table with the inserted book in its final position.

After all insertions, calculate and print the average number of probing steps per operation.  
  
CODE:

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// System.out.print("Enter the size of the hash table: ");

int size = scanner.nextInt();

// Creating a hash table with the specified size

int[] hashTable = new int[size];

// Linear probing hash function

// h(key) = key % tableSize

// System.out.println("Hash Function: h(key) = key % " + tableSize);

// Inserting keys dynamically

for (int i = 0; i < 10; i++) {

int key = scanner.nextInt();

linearProbingInsert(hashTable, key,size);

printHashTable(hashTable);

}

scanner.close();

}

private static void linearProbingInsert(int[] hashTable, int key,int size) {

int index = key % size;

while (hashTable[index] != 0) {

// Linear probing: move to the next slot if the current slot is occupied

index = (index + 1) % size;

}

// Insert the key into the found empty slot

hashTable[index] = key;

}

private static void printHashTable(int[] hashTable) {

System.out.print("Hash Table: [ ");

for (int i : hashTable) {

System.out.print(i + " ");

}

System.out.println("]");

}

}

4.Imagine you're developing a basic spell-checking feature for a text editor. You decide to use a hash table to store a set of correctly spelled words. The user can input words, and the program will check if they're present in the hash table. If not, they're considered potentially misspelled.  
  
**Input Format:**

Hash table size: An integer representing the desired size of the hash table.

Words: A sequence of words, one per line, for the program to process. The user can type "exit" to stop entering words.

**Output Format:**

Hash table state: After each word is inserted, display the current state of the hash table, indicating the index and the word (or "empty" if the slot is empty)  
  
ANSWER:

import java.util.Scanner;

class HashTable {

private String[] table;

private int size;

public HashTable(int size) {

this.size = size;

this.table = new String[size];

}

public void insert(String word) {

int index = word.length() % size;

int i = 1;

while (table[(index + i) % size] != null) i++;

table[(index + i) % size] = word;

}

public void printTable() {

System.out.println("Hash Table:");

for (int i = 0; i < size; i++)

System.out.println(i + ": " + (table[i] == null ? "empty" : table[i]));

System.out.println();

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// System.out.print("Enter hash table size: ");

HashTable spellChecker = new HashTable(scanner.nextInt());

// System.out.println("Enter words (type 'exit' to stop):");

while (true) {

String word = scanner.next();

if (word.equals("exit")) break;

spellChecker.insert(word);

spellChecker.printTable();

}

scanner.close();

}

}

\*\*Test Case 1:\*\*

Input:

3

apple

banana

exit

\*\*Expected Output:\*\*

Hash Table:

0: empty

1: empty

2: apple

Hash Table:

0: empty

1: banana

2: apple

\*\*Test Case 2:\*\*

Input:

5

apple

banana

orange

exit

\*\*Expected Output:\*\*

Hash Table:

0: empty

1: empty

2: empty

3: apple

4: empty

Hash Table:

0: banana

1: empty

2: empty

3: apple

4: empty

Hash Table:

0: banana

1: empty

2: empty

3: apple

4: orange

\*\*Test Case 3:\*\*

Input:

4

apple

banana

orange

exit

```

\*\*Expected Output:\*\*

```

Hash Table:

0: empty

1: empty

2: apple

3: empty

Hash Table:

0: banana

1: empty

2: apple

3: empty

Hash Table:

0: banana

1: orange

2: apple

3: empty

```

\*\*Test Case 4:\*\*

```

Input:

2

apple

banana

exit

```

\*\*Expected Output:\*\*

```

Hash Table:

0: empty

1: apple

Hash Table:

0: banana

1: apple

```

\*\*Test Case 5:\*\*

```

Input:

6

apple

banana

orange

grape

exit

```

\*\*Expected Output:\*\*

```

Hash Table:

0: empty

1: empty

2: empty

3: apple

4: empty

5: empty

Hash Table:

0: banana

1: empty

2: empty

3: apple

4: empty

5: empty

Hash Table:

0: banana

1: empty

2: grape

3: apple

4: empty

5: empty

Hash Table:

0: banana

1: empty

2: grape

3: apple

4: orange

5: empty

\*\*Test Case 6:\*

Input:

1

apple

banana

exit

\*\*Expected Output:\*\*

Hash Table:

0: banana

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
4. Imagine you're developing a simple inventory management system for a small bookstore. You decide to use a Hashtable to store book information, where the keys are book ISBNs (integers) and the values are book titles (strings).

**Input Format:**

Key-value pairs: A single line of text containing key-value pairs separated by spaces, where each key is an integer representing a book ISBN and each value is a string representing the corresponding book title.

**Output Format:**

Final Hashtable: The contents of the Hashtable after all key-value pairs have been added, displayed in a clear format that shows the ISBN-title associations.  
  
ANSWER:

import java.util.\*;

class Main{

public static void main(String args[]) {

Scanner scanner = new Scanner(System.in);

// Create a Hashtable to store book titles corresponding to ISBNs

Hashtable<Integer, String> inventory = new Hashtable<>();

// Input the values as key-value pairs

// System.out.println("Enter book ISBN-title pairs for the Hashtable (e.g., 123456789 Book1 987654321 Book2):");

String input = scanner.nextLine();

String[] keyValuePairs = input.split("\\s+");

for (int i = 0; i < keyValuePairs.length; i += 2) {

int isbn = Integer.parseInt(keyValuePairs[i]);

String title = keyValuePairs[i + 1];

inventory.put(isbn, title);

}

// Printing the final Hashtable

System.out.println("Final Inventory:");

for (Map.Entry<Integer, String> entry : inventory.entrySet()) {

System.out.println("ISBN: " + entry.getKey() + ", Title: " + entry.getValue());

}

scanner.close();

}

}

\*\*Test Case 1:\*\*

123456789 Book1 987654321 Book2 111111111 Book3

\*\*Output:\*\*

Final Inventory:

ISBN: 123456789, Title: Book1

ISBN: 987654321, Title: Book2

ISBN: 111111111, Title: Book3

```

\*\*Test Case 2:\*\*

555555555 JavaProgramming 888888888 DataStructures 999999999 Algorithms

```

\*\*Output:\*\*

```

Final Inventory:

ISBN: 555555555, Title: JavaProgramming

ISBN: 888888888, Title: DataStructures

ISBN: 999999999, Title: Algorithms

```

\*\*Test Case 3:\*\*

111111111 BookA 333333333 BookC 444444444 BookD

```

\*\*Output:\*\*

```

Final Inventory:

ISBN: 111111111, Title: BookA

ISBN: 333333333, Title: BookC

ISBN: 444444444, Title: BookD

```

\*\*Test Case 4:\*\*

123 Book1 456 Book2 789 Book3

```

\*\*Output:\*\*

```

Final Inventory:

ISBN: 123, Title: Book1

ISBN: 456, Title: Book2

ISBN: 789, Title: Book3

```

\*\*Test Case 5:\*\*

111 BookA 333 BookC 222 BookD

```

\*\*Output:\*\*

```

Final Inventory:

ISBN: 111, Title: BookA

ISBN: 333, Title: BookC

\*\*Test Case 6:\*\*

555 JavaProgramming 555 JavaProgramming 555 JavaProgramming

\*\*Output:\*\*

Final Inventory:

ISBN: 555, Title: JavaProgramming  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Imagine you are developing a program to manage a user's favorite colors. The user can add new colors to their collection, view the current collection, and remove specific colors. Write a Java program that uses a HashSet to store the user's favorite colors. Provide a scenario where the user adds, removes, and views their favorite colors.

INPUT FORMAT:

Add your favorite colors (comma-separated): <color1>, <color2>, ...

Remove colors from your favorites (comma-separated): <colorToRemove1>, <colorToRemove2>, ...

OUTPUT FORMAT:

Your Favorite Colors: [<currentFavoriteColor1>, <currentFavoriteColor2>, ...]

Removed Colors: [<colorToRemove1>, <colorToRemove2>, ...]

Your Favorite Colors Final Output: [<finalFavoriteColor1>, <finalFavoriteColor2>, ...]

ANSWER:

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Create an empty hash set

HashSet<String> favoriteColors = new HashSet<String>();

// Scenario: User adds favorite colors

// System.out.print("Add your favorite colors (comma-separated): ");

String addInput = scanner.nextLine();

String[] colorsToAdd = addInput.split(",\\s\*");

favoriteColors.addAll(Arrays.asList(colorsToAdd));

// Print the current favorite colors

System.out.println("Your Favorite Colors: " + favoriteColors);

// Scenario: User removes specific colors

// System.out.print("Remove colors from your favorites (comma-separated): ");

String removeInput = scanner.nextLine();

String[] colorsToRemove = removeInput.split(",\\s\*");

favoriteColors.removeAll(Arrays.asList(colorsToRemove));

// Print the final favorite colors

System.out.println("Removed Colors: " + Arrays.asList(colorsToRemove));

System.out.println("Your Favorite Colors Final Output: " + favoriteColors);

scanner.close();

}

}

\*\*Test Case 1:\*\*

Red, Blue, Green, Yellow

Blue, Yellow

\*\*Output:\*\*

Your Favorite Colors: [Red, Blue, Green, Yellow]

Removed Colors: [Blue, Yellow]

Your Favorite Colors Final Output: [Red, Green]

```

\*\*Test Case 2:\*\*

Orange, Purple, Pink, Brown

Purple, Brown

```

\*\*Output:\*\*

Your Favorite Colors: [Orange, Purple, Pink, Brown]

Removed Colors: [Purple, Brown]

Your Favorite Colors Final Output: [Orange, Pink]

```

\*\*Test Case 3:\*\*

```

Black, White, Gray, Silver

Red, Blue

```

\*\*Output:\*\*

```

Your Favorite Colors: [Black, White, Gray, Silver]

Removed Colors: [Red, Blue]

Your Favorite Colors Final Output: [Black, White, Gray, Silver]

```

\*\*Test Case 4:\*\*

Yellow, Green, Blue, Purple

Red, Orange

\*\*Output:\*\*

```

Your Favorite Colors: [Yellow, Green, Blue, Purple]

Removed Colors: [Red, Orange]

Your Favorite Colors Final Output: [Yellow, Green, Blue, Purple]

```

\*\*Test Case 5:\*\*

Pink, Red, Orange, Yellow

Purple, Blue, Green

```

\*\*Output:\*\*

```

Your Favorite Colors: [Pink, Red, Orange, Yellow]

Removed Colors: [Purple, Blue, Green]

Your Favorite Colors Final Output: [Pink, Red, Orange, Yellow]

\*\*Test Case 6:\*\*

Black, White, Gray, Silver

Black, White, Gray, Silver

\*\*Output:\*\*

Your Favorite Colors: [Black, White, Gray, Silver]

Removed Colors: [Black, White, Gray, Silver]

Your Favorite Colors Final Output: []  
  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Imagine you are developing a program to manage a vegetable market's inventory. The inventory is represented by a HashSet, where each element is a vegetable. Users can dynamically add new vegetables to the inventory and view the current inventory size. Write a Java program based on this scenario.

INPUT FORMAT :

Add vegetables to the inventory (comma-separated): <vegetable1>, <vegetable2>, ...

OUTPUT FORMAT:

Current Vegetable Inventory: [<currentVegetable1>, <currentVegetable2>, ...]

Size of the Vegetable Inventory: <currentSize>

ANSWER:

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Create an empty hash set for vegetable inventory

HashSet<String> vegetableInventory = new HashSet<String>();

// Scenario: User adds vegetables to the inventory

// System.out.print("Add vegetables to the inventory (comma-separated): ");

String addInput = scanner.nextLine();

String[] vegetablesToAdd = addInput.split(",\\s\*");

vegetableInventory.addAll(Arrays.asList(vegetablesToAdd));

// Display the current vegetable inventory

System.out.println("Current Vegetable Inventory: " + vegetableInventory);

// Display the size of the vegetable inventory

System.out.println("Size of the Vegetable Inventory: " + vegetableInventory.size());

scanner.close();

}

}

\*\*Test Case 1:\*\*

Carrot, Tomato, Spinach, Potato

```

\*\*Output:\*\*

```

Current Vegetable Inventory: [Carrot, Tomato, Spinach, Potato]

Size of the Vegetable Inventory: 4

```

\*\*Test Case 2:\*\*

```

Broccoli, Cabbage, Onion, Garlic

```

\*\*Output:\*\*

```

Current Vegetable Inventory: [Broccoli, Cabbage, Onion, Garlic]

Size of the Vegetable Inventory: 4

```

\*\*Test Case 3:\*\*

```

Bell Pepper, Eggplant, Zucchini, Mushroom

```

\*\*Output:\*\*

```

Current Vegetable Inventory: [Bell Pepper, Eggplant, Zucchini, Mushroom]

Size of the Vegetable Inventory: 4

```

\*\*Test Case 4:\*\*

```

Spinach, Potato, Onion, Garlic

```

\*\*Output:\*\*

```

Current Vegetable Inventory: [Spinach, Potato, Onion, Garlic]

Size of the Vegetable Inventory: 4

```

\*\*Test Case 5:\*\*

```

Tomato, Bell Pepper, Mushroom, Zucchini

```

\*\*Output:\*\*

```

Current Vegetable Inventory: [Tomato, Bell Pepper, Mushroom, Zucchini]

Size of the Vegetable Inventory: 4

```

\*\*Test Case 6:\*\*

```

Carrot, Broccoli, Garlic, Onion

```

\*\*Output:\*\*

```

Current Vegetable Inventory: [Carrot, Broccoli, Garlic, Onion]

Size of the Vegetable Inventory: 4

```

7. Imagine you are developing a program to manage student names in a classroom. Students' names are initially stored in a HashSet. Write a Java program that allows the user to dynamically add student names to the list and then displays both the original student name list and the array of student names.

INPUT FORMAT :

Add student names to the list (comma-separated): <name1>, <name2>, ...

OUTPUT FORMAT :

Original Student Name List: [<originalName1>, <originalName2>, ...]

Array elements (Student Names):

<name1>

<name2>

...

ANSWER:

import java.util.\*;

public class StudentManagement {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Create an empty hash set for student names

HashSet<String> studentNames = new HashSet<String>();

// Scenario: User adds student names to the list

System.out.print("Add student names to the list (comma-separated): ");

String addInput = scanner.nextLine();

String[] namesToAdd = addInput.split(",\\s\*");

studentNames.addAll(Arrays.asList(namesToAdd));

// Display the original student name list

System.out.println("Original Student Name List: " + studentNames);

// Convert the hash set to an array

String[] studentArray = new String[studentNames.size()];

studentNames.toArray(studentArray);

// Display the array elements (student names)

System.out.println("Array elements (Student Names): ");

for (String name : studentArray) {

System.out.println(name);

}

scanner.close();

}

}

\*\*Input Format:\*\*

```

Add student names to the list (comma-separated): <name1>, <name2>, ...

```

\*\*Output Format:\*\*

```

Original Student Name List: [<originalName1>, <originalName2>, ...]

Array elements (Student Names):

<name1>

<name2>

...

```

\*\*Example:\*\*

```

Add student names to the list (comma-separated): John, Alice, Bob, Sarah

```

\*\*Output:\*\*

```

Original Student Name List: [John, Alice, Bob, Sarah]

Array elements (Student Names):

John

Alice

Bob

Sarah

```

\*\*Sample Test Cases:\*\*

\*\*Test Case 1:\*\*

Emma, Michael, Olivia, Daniel

\*\*Output:\*\*

Original Student Name List: [Emma, Michael, Olivia, Daniel]

Array elements (Student Names):

Emma

Michael

Olivia

Daniel

\*\*Test Case 2:\*\*

Sophia, Ethan, Ava, Logan

\*\*Output:\*\*

Original Student Name List: [Sophia, Ethan, Ava, Logan]

Array elements (Student Names):

Sophia

Ethan

Ava

Logan

\*\*Test Case 3:\*\*

Mia, Liam, Isabella, Benjamin

\*\*Output:\*\*

Original Student Name List: [Mia, Liam, Isabella, Benjamin]

Array elements (Student Names):

Mia

Liam

Isabella

Benjamin

\*\*Test Case 4:\*\*

Noah, Emily, Jackson, Abigail

\*\*Output:\*\*

Original Student Name List: [Noah, Emily, Jackson, Abigail]

Array elements (Student Names):

Noah

Emily

Jackson

Abigail

\*\*Test Case 5:\*\*

Harper, Aiden, Addison, Elijah

\*\*Output:\*\*

Original Student Name List: [Harper, Aiden, Addison, Elijah]

Array elements (Student Names):

Harper

Aiden

Addison

Elijah

\*\*Test Case 6:\*\*

Chloe, Henry, Amelia, Oliver

\*\*Output:\*\*

Original Student Name List: [Chloe, Henry, Amelia, Oliver]

Array elements (Student Names):

Chloe

Henry

Amelia

Oliver  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_