SC1008 Assignment 1 of C++: C++ Basics and Linked List

1. **(Merge Arrays) [10 Marks]** Write a C++ **function template** called `mergeArrays` to concatenate two arrays of different type of data (i.e., int, double, float, string, long). Below are the basic requirements for the function template `mergeArrays`:
    - The two arrays will always be arrays of the same data type;
    - The array can be empty, i.e., `nullptr`;
    - You are **NOT allowed to use any STL containers or algorithms** in your implementation. Otherwise, 0 marks will be given;
    - If you do not implement the solution as a function template, 0 marks will be given.

    Also, you are requested to write another C++ function template `printAndDeleteArray` to 1) print each element of the merged array, and 2) then free the dynamically allocated memory for merged array.

Below is the starting code with two test cases.

```cpp
#include <iostream>
#include <string>

////// To-do: Write Your Code Here//////////////
// Template function mergeArrays()  to merge two arrays
//
//



////// To-do: Write Your Code Here//////////////
// Template function printAndDeallocate() to print and deallocate the merged array
//
//


int main() {
    // Case 1: Integers
    int* arr1 = new int[3];
    arr1[0] = 1;
    arr1[1] = 2;
    arr1[2] = 3;
    int size1 = 3;
    int arr2[] = {4, 5, 6};
    int size2 = sizeof(arr2) / sizeof(arr2[0]);
    int* mergedArray = nullptr;
    // Merging arrays
    mergeArrays(arr1, size1, arr2, size2, mergedArray);
    // Printing and deallocating the merged array
    printAndDeallocate(mergedArray, size1 + size2);
```

```
    delete[] arr1;

    // Case 2: doubles
    double arr3[] = {1.1, 2.2, 3.3};
    int size3 = sizeof(arr3) / sizeof(arr3[0]);
    double arr4[] = {4.4, 5.5};
    int size4 = sizeof(arr4) / sizeof(arr4[0]);
    double* mergedArray2 = nullptr;
    mergeArrays(arr3, size3, arr4, size4, mergedArray2);
    printAndDeallocate(mergedArray2, size3 + size4);

    return 0;
}
```

The sample outputs should be:

```
Merged Array: 1 2 3 4 5 6
Merged Array: 1.1 2.2 3.3 4.4 5.5
```

2. **(Remove Duplicate Names from An Unsorted Linked List) [12 Marks]** A linked list is used to store students' name list. However, due to mistaking operations, there may be duplicated names in the linked list. You are asked to write a function to remove duplicated names from a given linked list. The structure of each node of the linked list is as follows:

```
struct StringNode {
    string name;
    StringNode* next;
};
```

The function prototype is as follows:

```
void removeDuplicatedNames(const StringNode*& head);
```

You are requested to follow the given requirements:

- When there are duplicated names, the corresponding first name should be kept;
- After removing the duplicated names, the relative order of the remaining names should be the same as the original linked list;
- The names are case sensitive;
- You are **NOT allowed to use any STL containers or algorithms** in your implementation. Otherwise, 0 marks will be given.

Below is the starting code for you.

```
#include <iostream>
#include <string>

struct StringNode {
```

```
    std::string name;
    StringNode* next;
};


void printList(const StringNode* head) {
    const StringNode* temp = head;
    while (temp) {
        std::cout << temp->name << " -> ";
        temp = temp->next;
    }
    std::cout << "NULL" << std::endl;
}

void append(StringNode*& head, const std::string& name) {
    StringNode* newNode = new StringNode;
    newNode->name = name;
    newNode->next = nullptr;
    if (!head) {
        head = newNode;
        return;
    }
    StringNode* temp = head;
    while (temp->next) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void freeList(StringNode*& head) {
    while (head) {
        StringNode* temp = head;
        head = head->next;
        delete temp;
    }
}

// Remove duplicate names from the linked list
void removeDuplicatedNames(StringNode*& head) {
    // TO-DO: Write Your Code Here
    //
    //
    //



}
```

```
int main() {
    StringNode* head = nullptr;
    append(head, "Alice");
    append(head, "Alice");
    append(head, "Bob");
    append(head, "Charlie");
    append(head, "David");
    printList(head);
    removeDuplicatedNames(head);
    printList(head);
    freeList(head);

    return 0;
}
```

Sample output should be:

```
Alice -> Alice -> Bob -> Charlie -> David -> NULL
Alice -> Bob -> Charlie -> David -> NULL
```