AQ1:

```c
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

bool containsDuplicate(int* nums, int numsSize) {
    int hashSize = numsSize * 2;
    int* hash = (int*)calloc(hashSize, sizeof(int));

    for (int i = 0; i < numsSize; i++) {
        int key = abs(nums[i]) % hashSize;

        if (hash[key] == nums[i]) {   // Found duplicate
            free(hash);
            return true;
        }
        hash[key] = nums[i];
    }

    free(hash);
    return false;
}

int main() {
    int nums[] = {1, 2, 3, 1};
    int size = sizeof(nums) / sizeof(nums[0]);

    // Print the array initially
    printf("Array: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", nums[i]);
    }
    printf("\n");

    // Check duplicate
    if (containsDuplicate(nums, size))
        printf("Contains duplicate: true");
    else
        printf("Contains duplicate: false");

    return 0;
}
```

Output:

```
PS F:\Work\SEM3\DSA\LAB\10>

                        > cd "f:\Work\SEM3\DSA\LAB\10\" ; if ($?) { g++ AQ1.cpp -o AQ1 } ; if ($?) { .\AQ1 }
Array: 1 2 3 1
Contains duplicate: true
PS F:\Work\SEM3\DSA\LAB\10>
```

AQ2:

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int A[] = {1, 2, 3, 4};
    int B[] = {3, 4, 5, 6};
    int sizeA = sizeof(A) / sizeof(A[0]);
    int sizeB = sizeof(B) / sizeof(B[0]);

    // Print both arrays initially
    printf("Array A: ");
    for (int i = 0; i < sizeA; i++) printf("%d ", A[i]);
    printf("\nArray B: ");
    for (int i = 0; i < sizeB; i++) printf("%d ", B[i]);
    printf("\n");

    int hashSize = sizeA + sizeB;
    int *hash = (int *)calloc(hashSize, sizeof(int));

    for (int i = 0; i < sizeA; i++) {
        int key = abs(A[i]) % hashSize;
        hash[key] = A[i];
    }

    printf("Common Elements: ");
    int found = 0;
    for (int i = 0; i < sizeB; i++) {
        int key = abs(B[i]) % hashSize;
        if (hash[key] == B[i]) {
            printf("%d ", B[i]);
            found = 1;
        }
    }

    if (!found)
        printf("None");

    free(hash);
    return 0;
}
```

Output:

```
PS F:\Work\SEM3\DSA\LAB\10> cd "f:\Work\SEM3\DSA\LAB\10\" ; if ($?) { g++ AQ2.cpp -o AQ2 } ; if ($?) { .\AQ2 }
Array A: 1 2 3 4
Array B: 3 4 5 6
Common Elements: 3 4
PS F:\Work\SEM3\DSA\LAB\10>
```

AQ3:

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int nums[] = {2, 3, 2, 4, 3, 2};
    int size = sizeof(nums) / sizeof(nums[0]);

    // Print array initially
    printf("Array: ");
    for (int i = 0; i < size; i++)
        printf("%d ", nums[i]);
    printf("\n");

    // Hash table (key = number, value = count)
    int hashSize = size * 2;
    int *hashKey = (int *)calloc(hashSize, sizeof(int));
    int *hashVal = (int *)calloc(hashSize, sizeof(int));

    // Insert and count frequency
    for (int i = 0; i < size; i++) {
        int key = abs(nums[i]) % hashSize;

        // If same number already at hashed index → increment count
        if (hashKey[key] == nums[i]) {
            hashVal[key]++;
        }
        else { // store new number + set count to 1
            hashKey[key] = nums[i];
            hashVal[key] = 1;
        }
    }

    // Print frequency result
    printf("\nFrequency of each element:\n");
    for (int i = 0; i < hashSize; i++) {
        if (hashVal[i] > 0) {
            printf("%d : %d times\n", hashKey[i], hashVal[i]);
        }
```

```
    }

    free(hashKey);
    free(hashVal);
    return 0;
}
```

Output:

```
PS F:\Work\SEM3\DSA\LAB\10>

                        > cd "f:\Work\SEM3\DSA\LAB\10\" ; if ($?) { g++ AQ3.cpp -o AQ3 } ; if ($?) { .\AQ3 }
Array: 2 3 2 4 3 2

Frequency of each element:
2 : 3 times
3 : 2 times
4 : 1 times
PS F:\Work\SEM3\DSA\LAB\10>
```

AQ4:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int nums[] = {4, 5, 1, 2, 0, 4};
    int size = sizeof(nums) / sizeof(nums[0]);

    // Print array initially
    printf("Array: ");
    for (int i = 0; i < size; i++)
        printf("%d ", nums[i]);
    printf("\n");

    int hashSize = size * 2;
    int *hashKey = (int *)calloc(hashSize, sizeof(int));
    int *hashVal = (int *)calloc(hashSize, sizeof(int));

    for (int i = 0; i < size; i++) {
        int key = abs(nums[i]) % hashSize;
        if (hashKey[key] == nums[i]) {
            hashVal[key]++;
        } else {
            hashKey[key] = nums[i];
            hashVal[key] = 1;
        }
    }

    int found = 0;
    for (int i = 0; i < size; i++) {
```

```
        int key = abs(nums[i]) % hashSize;
        if (hashVal[key] == 1) {
            printf("\nFirst non-repeating element: %d\n", nums[i]);
            found = 1;
            break;
        }
    }

    if (!found)
        printf("\nNo non-repeating element found\n");

    free(hashKey);
    free(hashVal);
    return 0;
}
```

Output:

```
PS F:\Work\SEM3\DSA\LAB\10> cd "f:\Work\SEM3\DSA\LAB\10\" ; if ($?) { g++ AQ4.cpp -o AQ4 } ; if ($?) { .\AQ4 }
Array: 4 5 1 2 0 4

First non-repeating element: 5
PS F:\Work\SEM3\DSA\LAB\10>
```

AQ5:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

struct Node {
    int data;
    struct Node *next;
};
bool hasCycle(struct Node *head) {
    struct Node **visited = (struct Node **)calloc(100, sizeof(struct Node
*));
    int size = 100;

    while (head != NULL) {
        int key = ((unsigned long)head) % size;

        if (visited[key] == head) {
            free(visited);
            return true;
        }

        visited[key] = head;
        head = head->next;
    }
```

```
    free(visited);
    return false;
}
int main() {
    // Creating linked list: 1 → 2 → 3 → 4 → 2
    struct Node *n1 = (struct Node *)malloc(sizeof(struct Node));
    struct Node *n2 = (struct Node *)malloc(sizeof(struct Node));
    struct Node *n3 = (struct Node *)malloc(sizeof(struct Node));
    struct Node *n4 = (struct Node *)malloc(sizeof(struct Node));
    n1->data = 1; n1->next = n2;
    n2->data = 2; n2->next = n3;
    n3->data = 3; n3->next = n4;
    n4->data = 4; n4->next = n2; // Loop back to node 2
    // Check loop
    if (hasCycle(n1))
        printf("true\n");
    else
        printf("false\n");

    return 0;
}
```

Output:

```
PS F:\Work\SEM3\DSA\LAB\10> cd "f:\Work\SEM3\DSA\LAB\10\" ; if ($?) { g++ AQ5.cpp -o AQ5 } ; if ($?) { .\AQ5 }
true
PS F:\Work\SEM3\DSA\LAB\10>
```

AQ6:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

struct Node {
    int data;
    struct Node *left, *right;
};

#define HASH_SIZE 100
int hashSet[HASH_SIZE] = {0};
int hashVal[HASH_SIZE] = {0};

bool checkDuplicate(struct Node *root) {
    if (root == NULL)
        return false;

    int key = abs(root->data) % HASH_SIZE;

    if (hashSet[key] == root->data && hashVal[key] == 1)
```

```
        return true;

    hashSet[key] = root->data;
    hashVal[key] = 1;

    if (checkDuplicate(root->left)) return true;
    if (checkDuplicate(root->right)) return true;

    return false;
}

// Create new node
struct Node *newNode(int data) {
    struct Node *node = (struct Node *)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

int main() {

    struct Node *root = newNode(5);
    root->left = newNode(3);
    root->right = newNode(8);
    root->left->left = newNode(2);
    root->left->right = newNode(3);
    root->right->right = newNode(9);

    if (checkDuplicate(root))
        printf("Duplicates Found\n");
    else
        printf("No Duplicates\n");

    return 0;
}
```

Output:

```
PS F:\Work\SEM3\DSA\LAB\10> cd "f:\Work\SEM3\DSA\LAB\10\" ; if ($?) { g++ AQ6.cpp -o AQ6 } ; if ($?) { .\AQ6 }
Duplicates Found
PS F:\Work\SEM3\DSA\LAB\10>
```