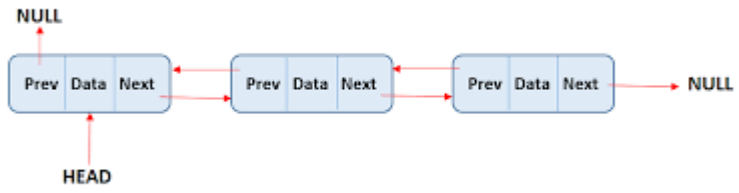
Assignment 3: Doubly Linked List and its applications

Create an ADT for the doubly linked list data structure with the following functions. Each node which consists of integer data, address of left and right nodes

Create a ListADT which has implementations for the following operations
Insert an item in the front of the list
void insertFront(listADT L, int c)
Insert an item at the end of the list
void insertEnd(listADT L, int c)
Insert an item ‘d’ after the first occurrence ‘c’ of the list
void insertMiddle(listADT L, int c, int d)
Display the items from the list
void displayItems(listADT L)
Delete the item present in the list
void deleteItem(listADT L, int c)
Search an element in the list and return the number of occurrences
int searchItem(listADT L, int c)

In addition, do the following operations:
Check whether the list contains duplicates?

DESIGN:



CODE:

main()

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

```
#include<stdlib.h>
#include "arrADT.h"
int main() {
  struct node *header;
  header = (struct node *)malloc(sizeof(struct node));
  header->left = NULL;
  header->right = NULL;
  create(header, 6);
  create(header, 8);
  create(header, 9);
  printf("1.the elements are:\n");
  display(header);
  insertEnd(header, 2);
  printf("\n2.insert End");
  display(header);
  insertAfter(header, 6, 7);
  printf("\n3.insert After some(6) element\n");
  display(header);

  struct node*ele;
  ele=search(header,2);
  if(ele!=NULL){
   printf("\n4.the searched element Found:%d",ele->data);
   }
   else{
          printf("\n4.Not found");
   }
  delete(header, 6);
  printf("\n5.delete the element 6:\n");
  printf("Ather the Deletion:\n");
  display(header);

  int d=findDup(header);
  if(d==1)
  {
     printf("\n6.duplicate found\n");
  }
  else
  {
     printf("\n6.No duplicate\n");
  }

  return 0;
}
```

DoubleADT.h

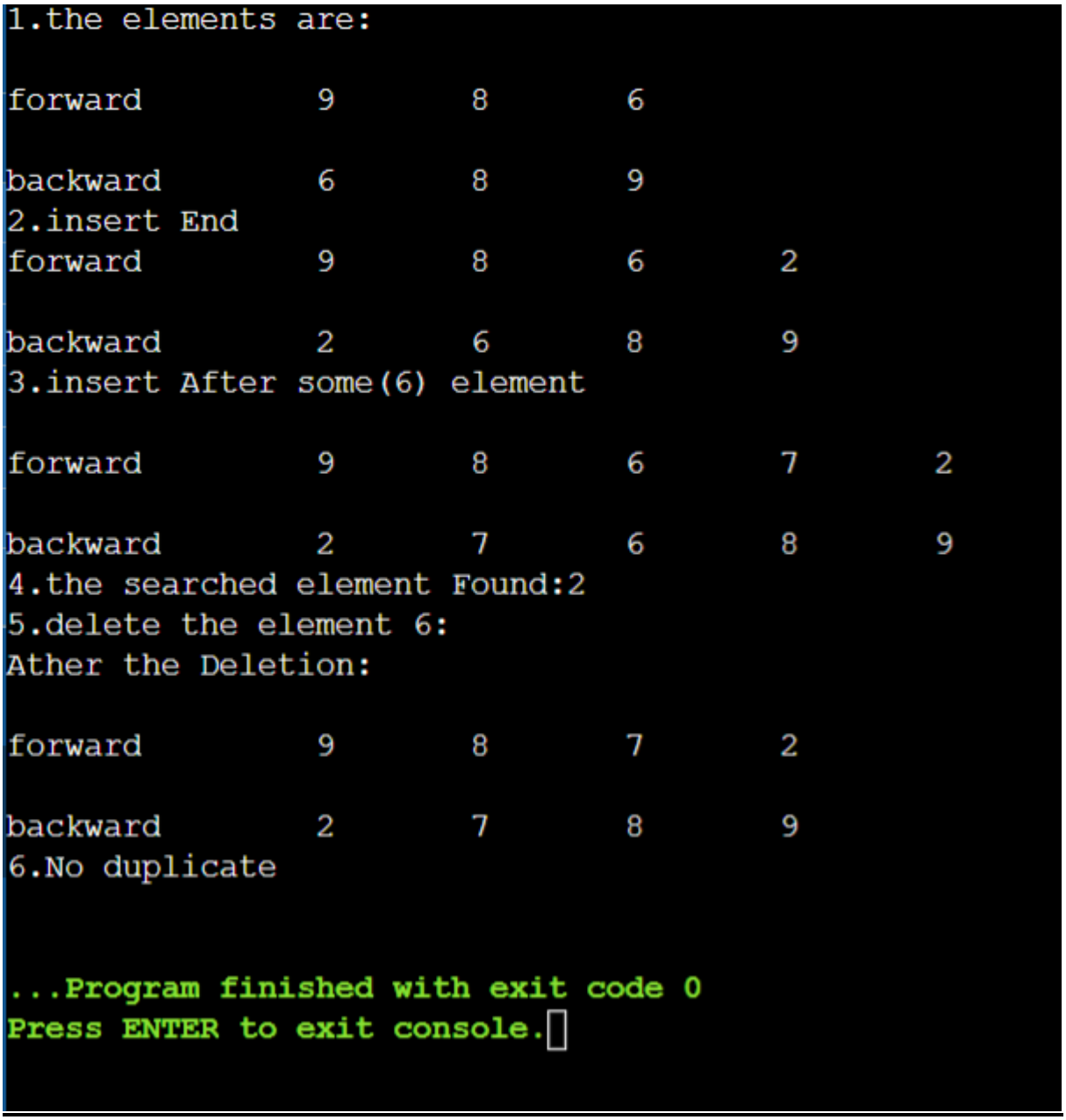
```
#include<stdlib.h>
#include<stdio.h>

//Insert an item in the front of the list

struct node {
  int data;
  struct node *left;
  struct node *right;
};

void create(struct node *header, int key) {
  struct node *temp;
  struct node *ptr;
  temp = (struct node *)malloc(sizeof(struct node));
  temp->data = key;
  if (header->right == NULL) {
          temp->right = header->right;
          temp->left = header;
          header->right = temp;
  } else {
          ptr = header->right;
          temp->left = header;
          header->right = temp;
          temp->right = ptr;
          ptr->left = temp;
  }
}
//Insert an item at the end of the list
void insertEnd(struct node *header, int key) {
  struct node *ptr;
  struct node *temp;
  temp = (struct node *)malloc(sizeof(struct node));
  ptr = header->right;
  temp->data = key;

  if (header->right == NULL) {
          temp->right = NULL;
          header->right = temp;
          temp->left = header;
  } else {
          while (ptr->right != NULL) {
```

```
            ptr = ptr->right;
            }
            temp->left = ptr;
            temp->right = NULL;
            ptr->right = temp;
  }
}
//Insert an item ‘d’ after the first occurrence ‘c’ of the list
void insertAfter(struct node *header, int key, int ele) {
  struct node *ptr, *next, *temp;
  ptr = header->right;
  while (ptr != NULL) {
            if (ptr->data == key) {
            temp = (struct node *)malloc(sizeof(struct node));
            temp->data = ele;
            next = ptr->right;
            ptr->right = temp;
            temp->left = ptr;
            temp->right = next;
            if (next != NULL) {
            next->left = temp;
            }
            return;
            }
            ptr = ptr->right;
  }
  printf("\ncannot be inserted after element %d", key);
}
//Display the items from the list
void display(struct node *header) {
  struct node *ptr;
  struct node *end;
  ptr = header->right;
  printf("\nforward ");
  while (ptr != NULL) {
            printf("\t%d", ptr->data);
            end = ptr;
            ptr = ptr->right;
  }
  printf("\n");
  printf("\nbackward");
  while (end != header) {
            printf("\t%d", end->data);
            end = end->left;
```

```
  }
}


//Delete the item present in the list
void delete(struct node *header, int key) {
  struct node *prev, *next, *ptr;
  ptr = header->right;
  while (ptr != NULL) {
          if (ptr->data == key) {
          prev = ptr->left;
          next = ptr->right;
          prev->right = next;
          if (next != NULL) {
          next->left = prev;
          }
          free(ptr);
          return;
          }
          ptr = ptr->right;
  }
  printf("\ncannot be deleted element %d", key);
}


//Search an element in the list and return the number of occurrences
struct node* search(struct node* header, int key) {
          struct node* ptr;
          ptr = header->right;
          while (ptr != NULL) {
          if (ptr->data == key) {
          return ptr; // Return a pointer to the found node not data
          }
          ptr = ptr->right;
          }
          return NULL;
}


int findDup(struct node *header)
{
          struct node *ptr, *next;
          ptr = header->right;
```

```
        while (ptr != NULL)
        {
        int x = ptr->data;
        next = ptr->right; // Move this line inside the outer loop

        while (next != NULL)
        {
        if (next->data == x)
        {
                return 1; // Found a duplicate
        }

        next = next->right;
        }

        ptr = ptr->right; // Move this line inside the outer loop
        }

        return 0; // No duplicates found
}
```

OUTPUT:



Learning Outcomes:

Understanding doubly linked lists as a data structure.
Performing insertion, deletion, searching, and counting operations on a linked list.