ASSIGNMENT 2 REPORT

CE1007 DATA STRUCTURES

EDWIN CANDINEGARA (FE2)

U1320135K

SEMESTER 2

AY 2013/2014

SCHOOL OF COMPUTER ENGINEERING

NANYANG TECHNOLOGICAL UNIVERISTY

```c
/*
       CE1007 Data Structure Assignment 2
       Name: Edwin Candinegara
       Matric No.: U1320135K
       Lab Group: FE2
*/

/* Preprocessor Instrucions */
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>


/* Structure Declarations */
typedef struct _listnode{
    int item;
    struct _listnode *next;
} ListNode;

typedef struct _linkedlist{
    int size;
    ListNode *head;
    ListNode *tail; /* To point at the very last node */
} LinkedList;


/* Function Prototypes */
void printList(LinkedList *ll);
ListNode * findNode(LinkedList *ll, int index);
int insertNode(LinkedList *ll, int index, int value);
int removeNode(LinkedList *ll, int index);
void createLinkedList(LinkedList *ll);
int insertSorted(LinkedList *ll, int value);
int removeDuplicates(LinkedList *ll);
```

```c
/* Main Program */
int main() {
   LinkedList ll; /* The main linked list for the whole program */
   int choice; /* For choosing which function to run */
   int value; /* Store the value that is going to be inserted*/
   int index; /* Store the index where the value is added after executing
                insertSorted() */

   /* Linked list initialization */
   ll.head = ll.tail = NULL;
   ll.size = 0;

   /* Print menu */
   printf("Menu:\n");
   printf("1. createLinkedList\n");
   printf("2. insertSorted\n");
   printf("3. removeDuplicates\n");
   printf("4. Exit\n\n");

   do {
      /* Choose which function to be run */
      printf("Enter your choice: ");
      scanf("%d", &choice);

      /* Run the chosen function */
      switch (choice) {
            case 1 :
               /* Create the linked list based on the user input */
               createLinkedList(&ll);
               break;

            case 2 :
               /* Take value input from the user */
               printf("Enter the value: ");
               scanf("%d", &value);
```

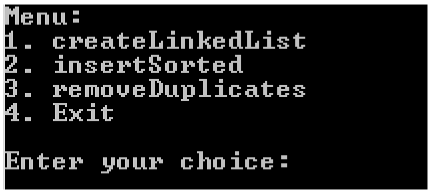
```
                /* Insert the value to the ascending sorted linked list and store
                   the index of the new node */
                index = insertSorted(&ll, value);

                /* Print the index of the new node */
                printf("\nThe value %d was added at index %d\n\n", value, index);
                break;

            case 3 :
                /* Remove any duplicates items in the list and print the number
                   of items removed */
                printf("%d numbers were removed from the list", removeDuplicates(&ll));
                printf("\n\n");
                break;
        }
    } while (choice < 4);
    return 0;
}

/* Q1. Create a linked list with the given value by the user's inputs */
void createLinkedList(LinkedList *ll) {
    int value, count = ll->size;

    /* Take the value input from user */
    printf("Enter a list of number, terminated by the value -1: ");
    scanf("%d", &value);

    /* Inserting the value to the linked list until the value is -1 */
    while (value != -1) {
        /* Inserting a new node with the value in the end of the linked list */
        insertNode(ll, ll->size, value);
        scanf("%d", &value);
    }

    /* Print the created linked list */
    printf("The list: ");
    printList(ll);
```

```c
    printf("\n\n");
}

/* Q2. Add a new node with its value where the list is already in ascending order */
int insertSorted(LinkedList *ll, int value) {
    ListNode *temp = ll->head;
    int index = 0;

    /* If the list is an empty list */
    if (temp == NULL)
        return -1;

    /* If the list is not empty */
    while (temp != NULL) {
        /* Break whenever the new value is less than or equal to the next node */
        if (value <= temp->item)
                break;

        /* Move to the next node */
        temp = temp->next;
        index++;
    }

    /* Insert the new value as a new node at the particular index */
    insertNode(ll, index, value);

    /* Print the created linked list */
    printf("The list: ");
    printList(ll);

    /* Return the index of the new node */
    return index;
}
```

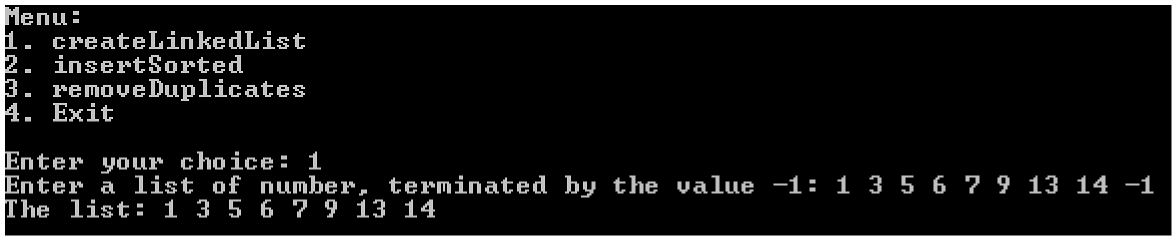
```
/* Q3. Remove any duplicates of a value */
int removeDuplicates(LinkedList *ll) {
   ListNode *pre, *cur;
   int index = 0, count = 0;

   /* Point to the first node */
   pre = ll->head;

   /* Check whether the list is empty */
   if (pre == NULL) {
      /* Print the created linked list */
      printf("The list: ");
      printList(ll);
      printf("\n");
      return 0;
   }

   /* Second pointer to compare the value inside a node with its previous nodes */
   cur = pre->next;

   /* When cur is NULL, it means that we have checked every nodes */
   while (cur != NULL) {

      /* Remove the node with the same value with the next node's value */
      if (pre->item == cur->item) {
            removeNode(ll, index);
            index--; /* After removal, each node’s index will be reduced by one */
            count++;
      }

      /* Move to the next node */
      pre = cur;
      cur = cur->next;
      index++;
   }
```

```c
    /* Print the created linked list */
    printf("The list: ");
    printList(ll);
    printf("\n");

    /* Return the number of nodes removed */
    return count;
}


/* ************************************************************** */
/* Some basic functions (credit to Mr. Mark Yong) */
/* Print the entire items in an existing list */
void printList(LinkedList *ll){
     ListNode *temp = ll->head;

     if (temp == NULL)
         return;

     while (temp != NULL){
         printf("%d ", temp->item);
         temp = temp->next;
     }
}

/* Find the address of a node given the index of the node */
ListNode * findNode(LinkedList *ll, int index){
   ListNode *temp;

   if (ll == NULL || index < 0 || index >= ll->size)
        return NULL;

   temp = ll->head;

   if (temp == NULL || index < 0)
       return NULL;
   while (index > 0){
       temp = temp->next;
       if (temp == NULL)
           return NULL;

       index--;
   }

   return temp;
}

/* Insert a node with the given the index */
int insertNode(LinkedList *ll, int index, int value){
   ListNode *pre, *cur;

   if (ll == NULL || index < 0 || index > ll->size)
       return -1;

   /* If empty list or inserting first node, need to update head pointer */
   if (ll->head == NULL || index == 0){
       cur = ll->head;
       ll->head = malloc(sizeof(ListNode));

       if (ll->size == 0)
           ll->tail = ll->head;

       ll->head->item = value;
       ll->head->next = cur;
       ll->size++;
       return 0;
   }

   /* Inserting as new last node */
   if (index == ll->size){
       pre = ll->tail;
       cur = pre->next;
       pre->next = malloc(sizeof(ListNode));
       ll->tail = pre->next;
       pre->next->item = value;
       pre->next->next = cur;
       ll->size++;
       return 0;
   }

   /* Find the nodes before and at the target position
      Create a new node and reconnect the links */
   if ((pre = findNode(ll, index-1)) != NULL){
       cur = pre->next;
       pre->next = malloc(sizeof(ListNode));
       pre->next->item = value;
       pre->next->next = cur;
       ll->size++;
       return 0;
   }

   return -1;
}

/* Remove a node with the given index */
int removeNode(LinkedList *ll, int index){
   ListNode *pre, *cur;

   /* Highest index we can remove is size - 1 */
   if (ll == NULL || index < 0 || index >= ll->size)
       return -1;

   /* If removing first node, need to update head pointer */
   if (index == 0){
       cur = ll->head->next;
       free(ll->head);
       ll->head = cur;
       ll->size--;

       if (ll->size == 0)
           ll->tail = 0;
       return 0;
   }

   /* Find the nodes before and after the target position
      Free the target node and reconnect the links */
   if ((pre = findNode(ll, index-1)) != NULL){

       /* Removing the last node, update the tail pointer */
       if (index == ll->size - 1){
           ll->tail = pre;
           free(pre->next);
           pre->next = NULL;
       }

       else {
           cur = pre->next->next;
           free(pre->next);
           pre->next = cur;
       }

       ll->size--;
       return 0;
   }

   return -1;
}
```

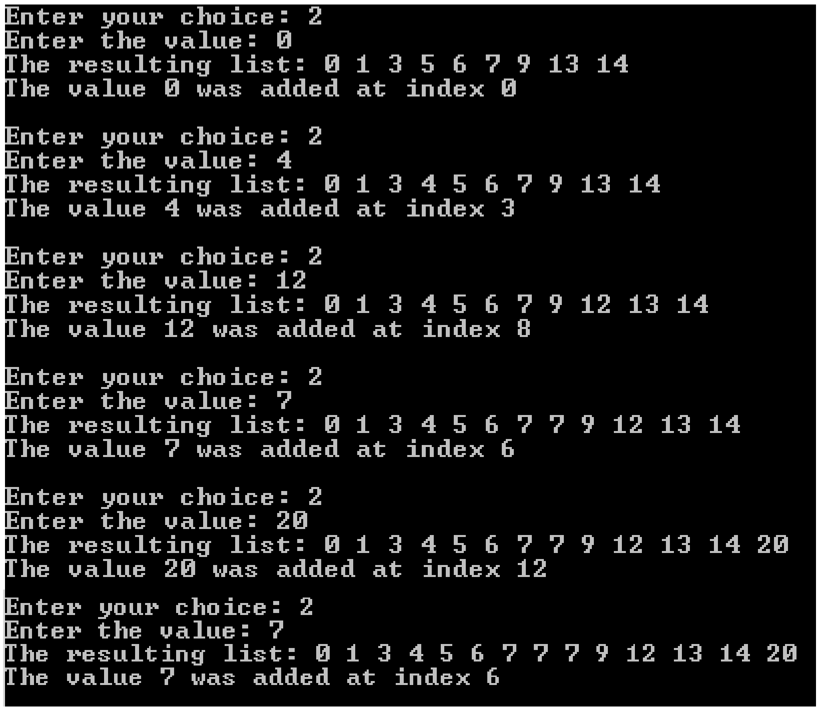
Test Case:



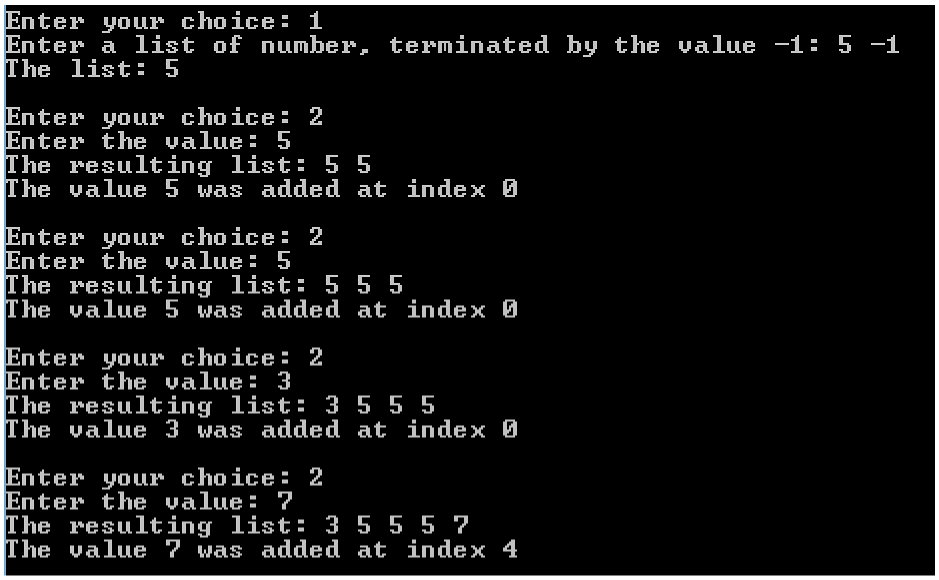
```
Menu:
1. createLinkedList
2. insertSorted
3. removeDuplicates
4. Exit

Enter your choice:
```



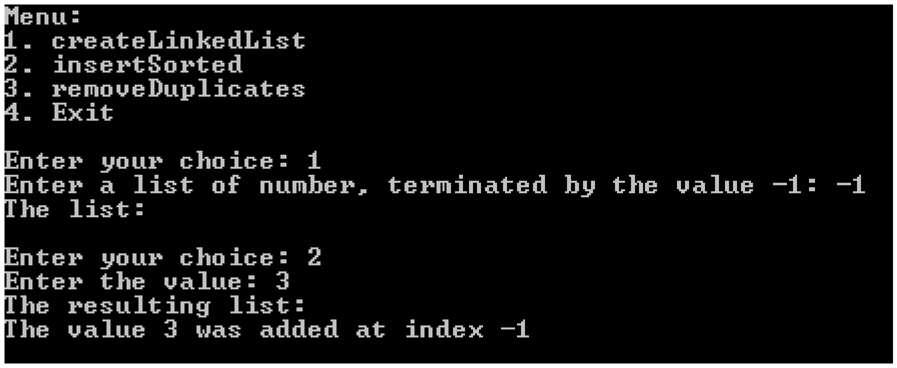
```
Menu:
1. createLinkedList
2. insertSorted
3. removeDuplicates
4. Exit

Enter your choice: 1
Enter a list of number, terminated by the value -1: 1 3 5 6 7 9 13 14 -1
The list: 1 3 5 6 7 9 13 14
```



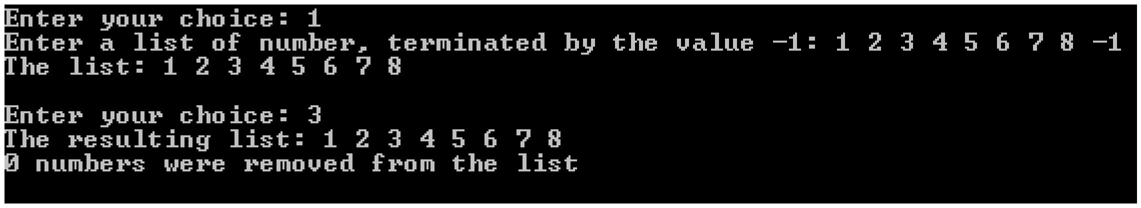
```
Enter your choice: 2
Enter the value: 0
The resulting list: 0 1 3 5 6 7 9 13 14
The value 0 was added at index 0

Enter your choice: 2
Enter the value: 4
The resulting list: 0 1 3 4 5 6 7 9 13 14
The value 4 was added at index 3

Enter your choice: 2
Enter the value: 12
The resulting list: 0 1 3 4 5 6 7 9 12 13 14
The value 12 was added at index 8

Enter your choice: 2
Enter the value: 7
The resulting list: 0 1 3 4 5 6 7 7 9 12 13 14
The value 7 was added at index 6

Enter your choice: 2
Enter the value: 20
The resulting list: 0 1 3 4 5 6 7 7 9 12 13 14 20
The value 20 was added at index 12
Enter your choice: 2
Enter the value: 7
The resulting list: 0 1 3 4 5 6 7 7 7 9 12 13 14 20
The value 7 was added at index 6
```



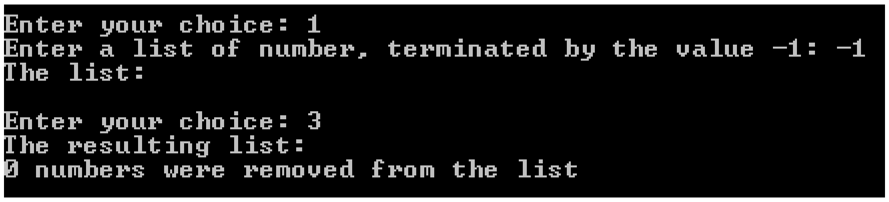
```
Enter your choice: 1
Enter a list of number, terminated by the value -1: 5 -1
The list: 5

Enter your choice: 2
Enter the value: 5
The resulting list: 5 5
The value 5 was added at index 0

Enter your choice: 2
Enter the value: 5
The resulting list: 5 5 5
The value 5 was added at index 0

Enter your choice: 2
Enter the value: 3
The resulting list: 3 5 5 5
The value 3 was added at index 0

Enter your choice: 2
Enter the value: 7
The resulting list: 3 5 5 5 7
The value 7 was added at index 4
```



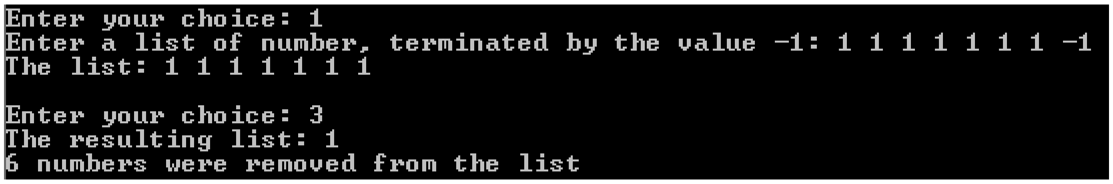
```
Menu:
1. createLinkedList
2. insertSorted
3. removeDuplicates
4. Exit

Enter your choice: 1
Enter a list of number, terminated by the value -1: -1
The list:

Enter your choice: 2
Enter the value: 3
The resulting list:
The value 3 was added at index -1
```



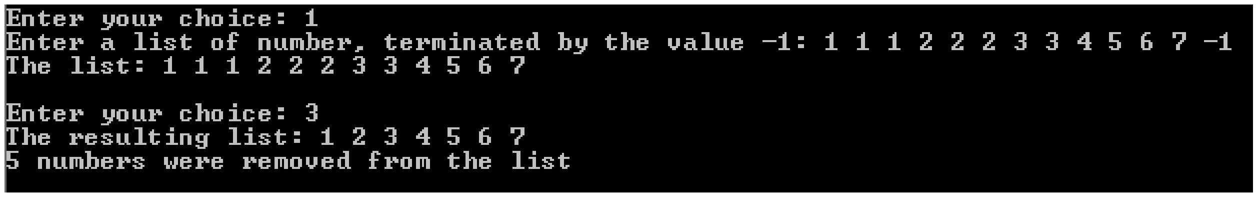
```
Enter your choice: 1
Enter a list of number, terminated by the value -1: 1 2 3 4 5 6 7 8 -1
The list: 1 2 3 4 5 6 7 8

Enter your choice: 3
The resulting list: 1 2 3 4 5 6 7 8
0 numbers were removed from the list
```



```
Enter your choice: 1
Enter a list of number, terminated by the value -1: -1
The list:

Enter your choice: 3
The resulting list:
0 numbers were removed from the list
```



```
Enter your choice: 1
Enter a list of number, terminated by the value -1: 1 1 1 1 1 1 1 -1
The list: 1 1 1 1 1 1 1

Enter your choice: 3
The resulting list: 1
6 numbers were removed from the list
```



```
Enter your choice: 1
Enter a list of number, terminated by the value -1: 1 1 1 2 2 2 3 3 4 5 6 7 -1
The list: 1 1 1 2 2 2 3 3 4 5 6 7

Enter your choice: 3
The resulting list: 1 2 3 4 5 6 7
5 numbers were removed from the list
```