LAB: 06

Implementations of list using array

Spring 2024

CSE-210L DATA STRUCTURE AND ALGOARITHM

Submitted by:

Name: **Hassan Zaib**

Reg no**: 22PWCSE2144**

Class Section**: A**

Submitted to:

Engr. Usman Malik

April 28, 2024

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Lab Objective:

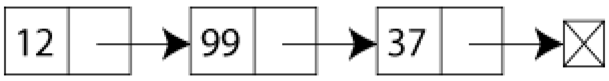
In this lab, we will learn about the Data Structure, lists and their useful operations.

Lists

A list or sequence is a data structure that represents a countable number of ordered values, where the same value may occur more than once.

Lists can be implemented using arrays and linked lists. In this lab we will do arrays implementation. When using arrays we have fixed size list where each element can be accessed directly using the index. As list is an abstract data type, we need to define operations. Some useful operations to be performed on the list are described in the next section.

Example:



As the name implies, lists can be used to store a list of elements. However, unlike in traditional arrays, lists can expand and shrink, and are stored dynamically in memory.

A finite set in the mathematical sense can be realized as a list with additional restrictions; that is, duplicate elements are disallowed and order is irrelevant. Sorting the list speeds up determining if a given item is already in the set, but in order to ensure the order, it requires more time to add new entry to the list. In efficient implementations, however, sets are implemented using self-balancing binary search trees or hash tables, rather than a list.

Lists also form the basis for other abstract data types including the queue, the stack, and their variations.

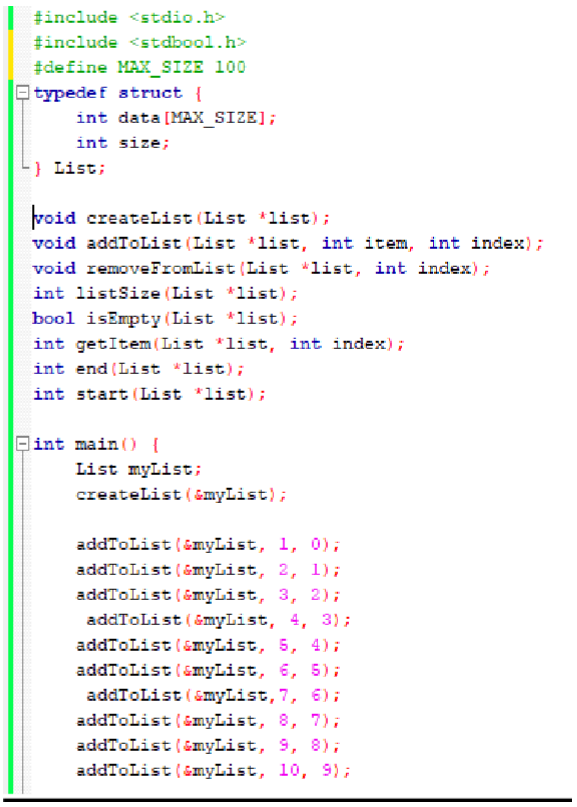
LAB TASKS

Task No: 01

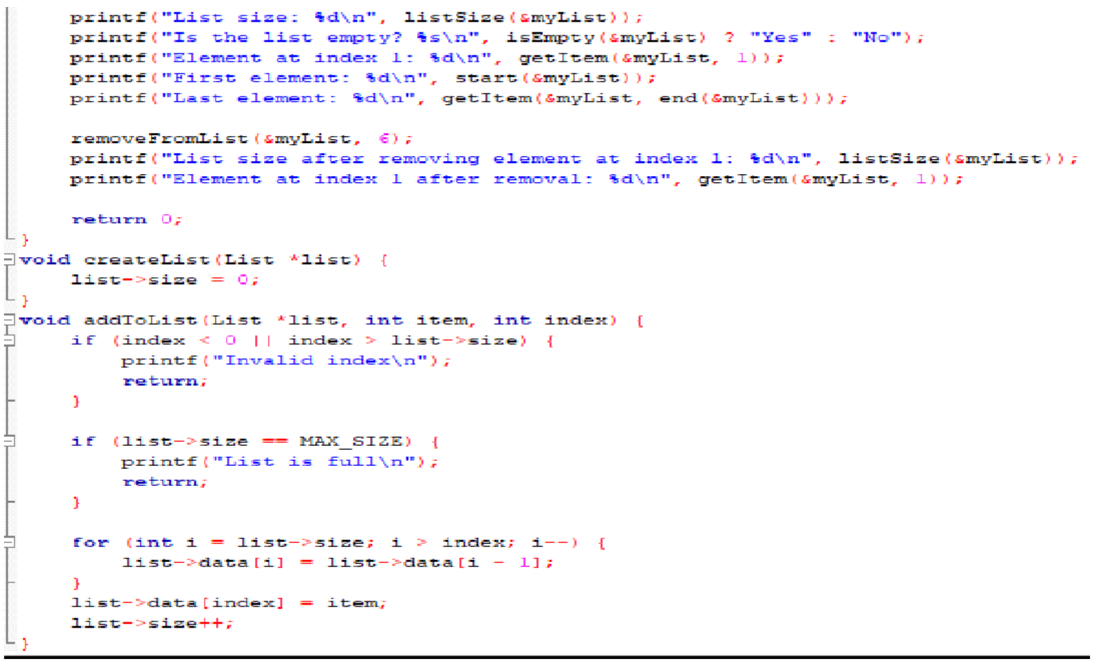
Implement LIST using arrays. It should perform following operations

- Create (Creates LIST)
- Add (Adds item to the LIST at given index)
- Remove (Removes item from the List at provided index)
- waSize (Determines Size of the LIST)
- IsEmpty (Determines if LIST is empty or not)
- Get (Retrieves an Item from the LIST)
- End (Returns end of the LIST)
- Start (Return start of the LIST)

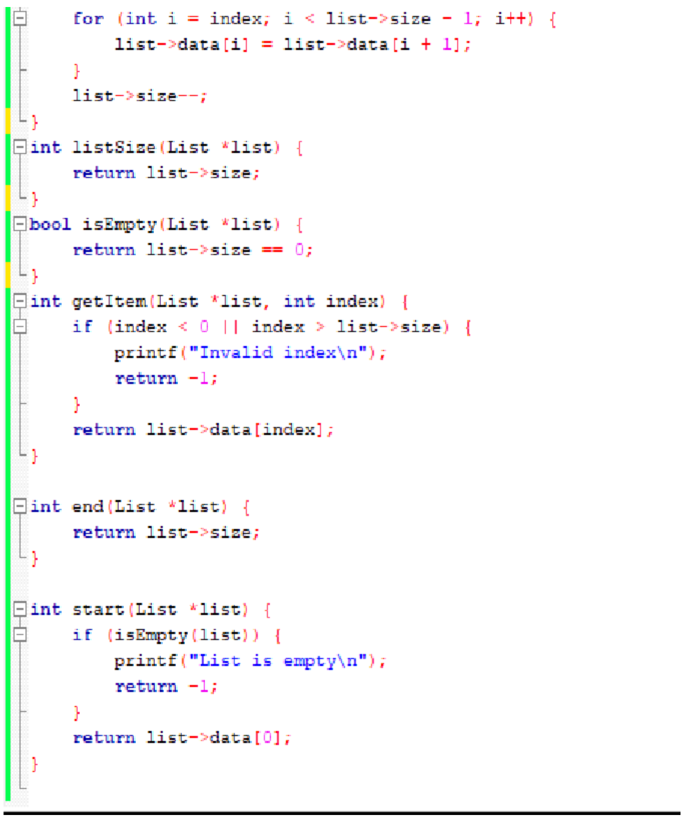
Code:



```c
#include <stdio.h>
#include <stdbool.h>
#define MAX_SIZE 100
typedef struct {
    int data[MAX_SIZE];
    int size;
} List;

void createList(List *list);
void addToList(List *list, int item, int index);
void removeFromList(List *list, int index);
int listSize(List *list);
bool isEmpty(List *list);
int getItem(List *list, int index);
int end(List *list);
int start(List *list);

int main() {
    List myList;
    createList(&myList);

    addToList(&myList, 1, 0);
    addToList(&myList, 2, 1);
    addToList(&myList, 3, 2);
     addToList(&myList, 4, 3);
    addToList(&myList, 5, 4);
    addToList(&myList, 6, 5);
     addToList(&myList,7, 6);
    addToList(&myList, 8, 7);
    addToList(&myList, 9, 8);
    addToList(&myList, 10, 9);
```



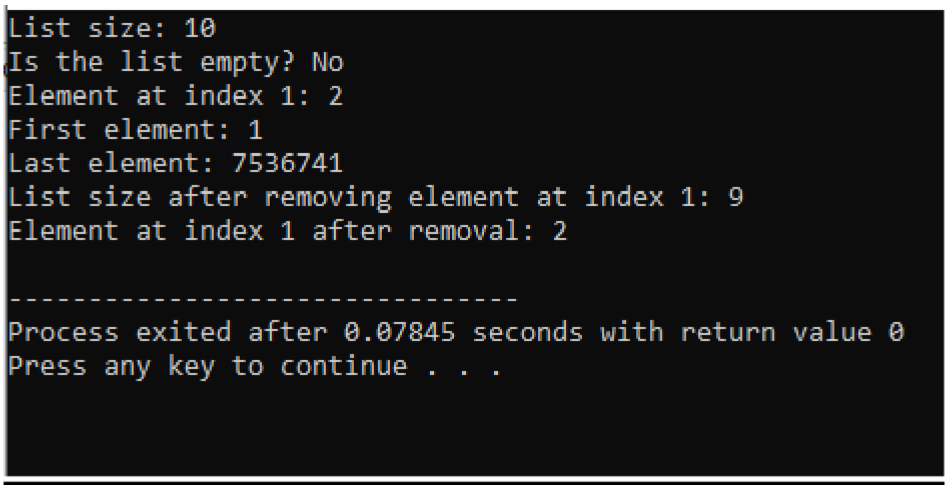
```c
    printf("List size: %d\n", listSize(&myList));
    printf("Is the list empty? %s\n", isEmpty(&myList) ? "Yes" : "No");
    printf("Element at index 1: %d\n", getItem(&myList, 1));
    printf("First element: %d\n", start(&myList));
    printf("Last element: %d\n", getItem(&myList, end(&myList)));

    removeFromList(&myList, 6);
    printf("List size after removing element at index 1: %d\n", listSize(&myList));
    printf("Element at index 1 after removal: %d\n", getItem(&myList, 1));

    return 0;
}
void createList(List *list) {
    list->size = 0;
}
void addToList(List *list, int item, int index) {
    if (index < 0 || index > list->size) {
        printf("Invalid index\n");
        return;
    }

    if (list->size == MAX_SIZE) {
        printf("List is full\n");
        return;
    }

    for (int i = list->size; i > index; i--) {
        list->data[i] = list->data[i - 1];
    }
    list->data[index] = item;
    list->size++;
}
```



```c
    for (int i = index; i < list->size - 1; i++) {
        list->data[i] = list->data[i + 1];
    }
    list->size--;
}
int listSize(List *list) {
    return list->size;
}
bool isEmpty(List *list) {
    return list->size == 0;
}
int getItem(List *list, int index) {
    if (index < 0 || index > list->size) {
        printf("Invalid index\n");
        return -1;
    }
    return list->data[index];
}

int end(List *list) {
    return list->size;
}

int start(List *list) {
    if (isEmpty(list)) {
        printf("List is empty\n");
        return -1;
    }
    return list->data[0];
}
```

Figure 1

Output:

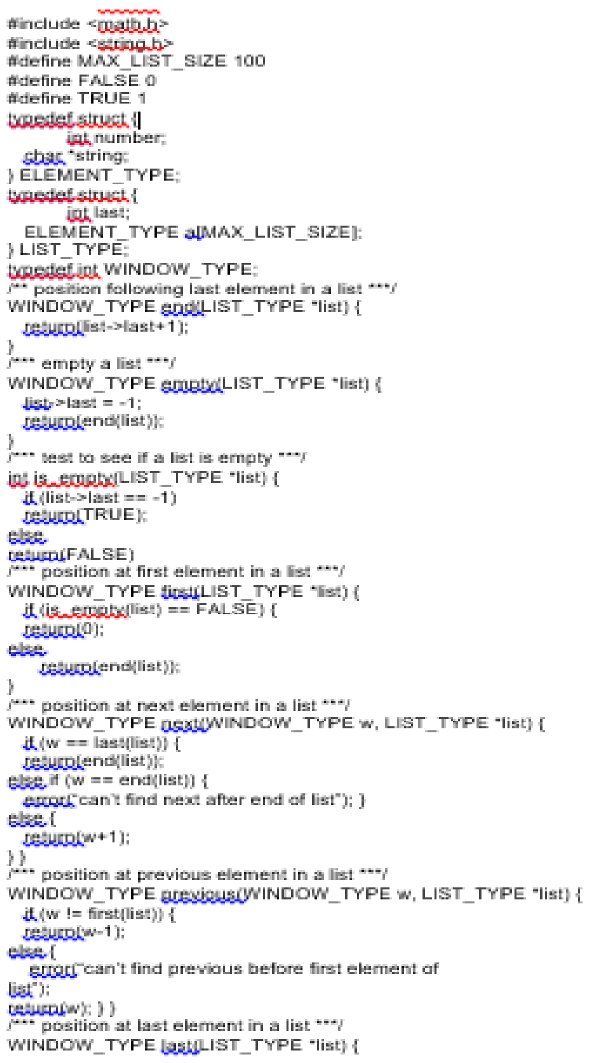


```
List size: 10
Is the list empty? No
Element at index 1: 2
First element: 1
Last element: 7536741
List size after removing element at index 1: 9
Element at index 1 after removal: 2

--------------------------------
Process exited after 0.07845 seconds with return value 0
Press any key to continue . . .
```

Figure 2 output

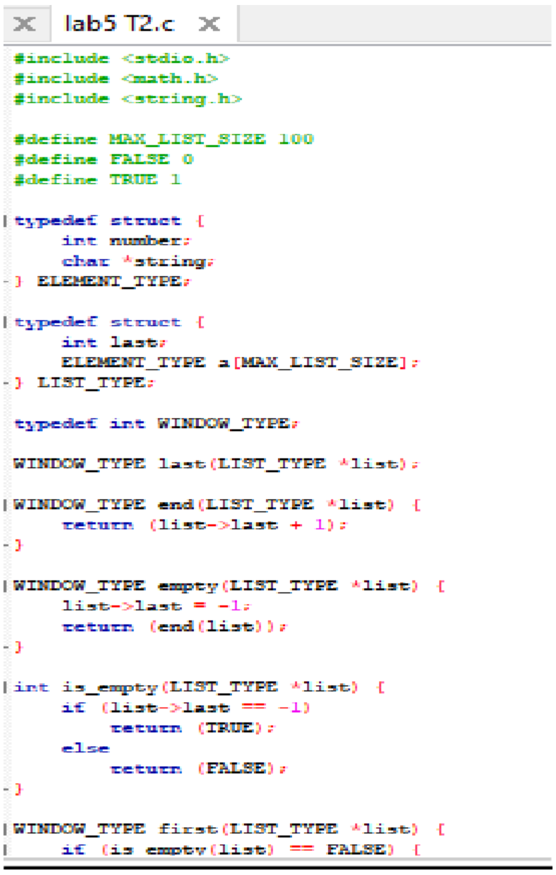
Task No: 02

Debugging of Code:



```
#include <math.h>
#include <string.h>
#define MAX_LIST_SIZE 100
#define FALSE 0
#define TRUE 1
typedef struct {
          int number;
  char *string;
} ELEMENT_TYPE;
typedef struct {
          int last;
  ELEMENT_TYPE a[MAX_LIST_SIZE];
} LIST_TYPE;
typedef int WINDOW_TYPE;
/** position following last element in a list ***/
WINDOW_TYPE end(LIST_TYPE *list) {
  return(list->last+1);
}
/*** empty a list ***/
WINDOW_TYPE empty(LIST_TYPE *list) {
  list->last = -1;
  return(end(list));
}
/*** test to see if a list is empty ***/
int is_empty(LIST_TYPE *list) {
  if (list->last == -1)
  return(TRUE);
else
return(FALSE)
/*** position at first element in a list ***/
WINDOW_TYPE first(LIST_TYPE *list) {
  if (is_empty(list) == FALSE) {
  return(0);
else
    return(end(list));
}
/*** position at next element in a list ***/
WINDOW_TYPE next(WINDOW_TYPE w, LIST_TYPE *list) {
  if (w == last(list)) {
  return(end(list));
else if (w == end(list)) {
  error("can't find next after end of list"); }
else {
  return(w+1);
} }
/*** position at previous element in a list ***/
WINDOW_TYPE previous(WINDOW_TYPE w, LIST_TYPE *list) {
  if (w != first(list)) {
  return(w-1);
else {
   error("can't find previous before first element of
list");
return(w); } }
/*** position at last element in a list ***/
WINDOW_TYPE last(LIST_TYPE *list) {
```

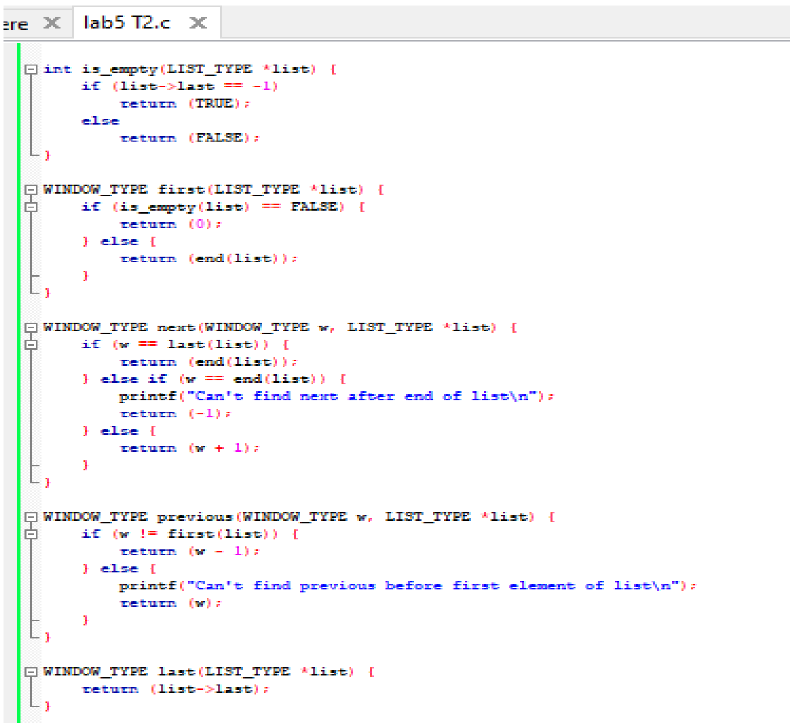
Correct Code:



lab5 T2.c

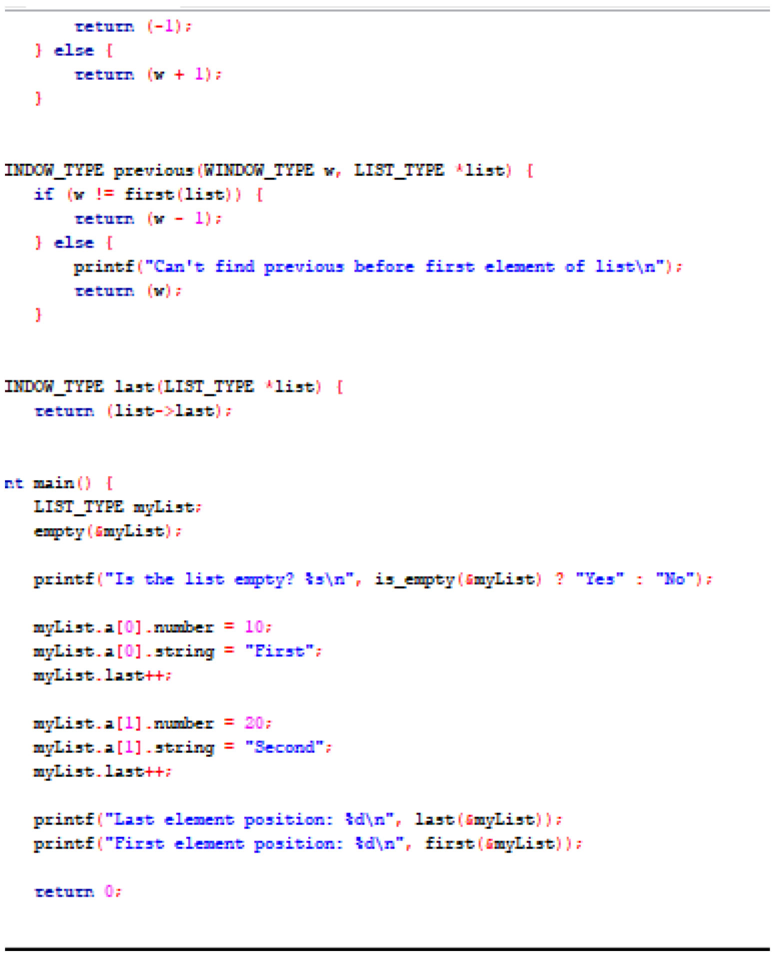
```c
#include <stdio.h>
#include <math.h>
#include <string.h>

#define MAX_LIST_SIZE 100
#define FALSE 0
#define TRUE 1

typedef struct {
    int number;
    char *string;
} ELEMENT_TYPE;

typedef struct {
    int last;
    ELEMENT_TYPE a[MAX_LIST_SIZE];
} LIST_TYPE;

typedef int WINDOW_TYPE;

WINDOW_TYPE last(LIST_TYPE *list);

WINDOW_TYPE end(LIST_TYPE *list) {
    return (list->last + 1);
}

WINDOW_TYPE empty(LIST_TYPE *list) {
    list->last = -1;
    return (end(list));
}

int is_empty(LIST_TYPE *list) {
    if (list->last == -1)
        return (TRUE);
    else
        return (FALSE);
}

WINDOW_TYPE first(LIST_TYPE *list) {
    if (is_empty(list) == FALSE) {
```

Figure 3 correct Code



lab5 T2.c

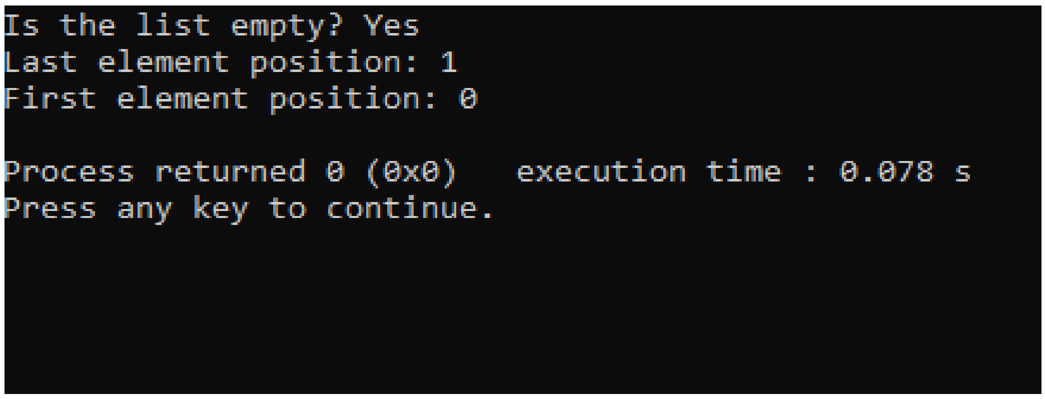
```c
int is_empty(LIST_TYPE *list) {
    if (list->last == -1)
        return (TRUE);
    else
        return (FALSE);
}

WINDOW_TYPE first(LIST_TYPE *list) {
    if (is_empty(list) == FALSE) {
        return (0);
    } else {
        return (end(list));
    }
}

WINDOW_TYPE next(WINDOW_TYPE w, LIST_TYPE *list) {
    if (w == last(list)) {
        return (end(list));
    } else if (w == end(list)) {
        printf("Can't find next after end of list\n");
        return (-1);
    } else {
        return (w + 1);
    }
}

WINDOW_TYPE previous(WINDOW_TYPE w, LIST_TYPE *list) {
    if (w != first(list)) {
        return (w - 1);
    } else {
        printf("Can't find previous before first element of list\n");
        return (w);
    }
}

WINDOW_TYPE last(LIST_TYPE *list) {
    return (list->last);
}
```



```
        return (-1);
    } else {
        return (w + 1);
    }


INDOW_TYPE previous(WINDOW_TYPE w, LIST_TYPE *list) {
    if (w != first(list)) {
        return (w - 1);
    } else {
        printf("Can't find previous before first element of list\n");
        return (w);
    }


INDOW_TYPE last(LIST_TYPE *list) {
    return (list->last);


nt main() {
    LIST_TYPE myList;
    empty(&myList);

    printf("Is the list empty? %s\n", is_empty(&myList) ? "Yes" : "No");

    myList.a[0].number = 10;
    myList.a[0].string = "First";
    myList.last++;

    myList.a[1].number = 20;
    myList.a[1].string = "Second";
    myList.last++;

    printf("Last element position: %d\n", last(&myList));
    printf("First element position: %d\n", first(&myList));

    return 0;
```

Figure 4 correct code

Output:



```
Is the list empty? Yes
Last element position: 1
First element position: 0

Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

Figure 5 output

LAB RUBRICS: (CSE-210L DATA STRUCTURE AND ALGORITHM)

Criteria & Point Assigned	Outstanding 4	Acceptable 3	Considerable 2	Below Expectations 1
Attendance and Attentiveness in Lab PLO10	Attended in proper Time and attentive in Lab	Attended in proper Time but not attentive in Lab	Attended late but attentive in Lab	Attended late not attentive in Lab
Equipment / Instruments Selection and Operation PLO1, PLO2, PLO3, PLO5,	Right selection and operation of appropriate equipment and instruments to perform experiment.	Right selection of appropriate equipment and instruments to perform experiment but with minor issues in operation	Needs guidance for right selection of appropriate equipment and instruments to perform experiment and to overcome errors in operation	Cannot appropriately select and operate equipment and instruments to perform experiment.
Result or Output/ Completion of target in Lab PLO9,	100% target has been completed and well formatted.	75% target has been completed and well formatted.	50% target has been completed but not well formatted.	None of the outputs are correct
Overall, Knowledge PLO10,	Demonstrates excellent knowledge of lab	Demonstrates good knowledge of lab	Has partial idea about the Lab and procedure followed	Has poor idea about the Lab and procedure followed
Attention to Lab Report PLO4,	Submission of Lab Report in Proper Time i.e. in next day of lab.,	Submission of Lab Report in proper time but not with proper documentation.	Late Submission with proper documentation.	Late Submission Very poor documentation

	with proper documentation.			