LAB 07

IMPLEMENTATION OF LISTS USING ARRAYS

Spring 2024

Submitted by: **HASSAN ZAIB JADOON**

Registration No: **22pwsce2144**

Class Section: **A**

"On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work."

Submitted to:

Engr. Usman Malik

Month Day, Year (28 April, 2024)

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

TITLE:

IMPLEMENTATION OF LISTS USING ARRAYS

OBJECTIVES:

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

• TASK 1:

1. Implement LIST using arrays. It should perform following operations
2. Create (Creates LIST)
3. Add (Adds item to the LIST at given index)
4. Remove (Removes item from the List at provided index)
5. waSize (Determines Size of the LIST)
6. IsEmpty (Determines if LIST is empty or not)
7. Get (Retrieves an Item from the LIST)
8. End (Returns end of the LIST)
9. Start (Return start of the LIST)

CODE:

task1.cpp

```cpp
#include <iostream>
using namespace std;

#define MAX_SIZE 100 // Maximum size of the list

class List {
private:
    int data[MAX_SIZE];
    int size;
public:
    // Constructor
    List() {
        size = 0;
    }

    // Member functions
    void addToList(int item, int index);
    void removeFromList(int index);
    int listSize();
    bool isEmpty();
    int getItem(int index);
    int end();
    int start();
};

// Function to add an item to the list at a specified index
void List::addToList(int item, int index) {
    if (index < 0 || index > size) {
        cout << "Invalid index" << endl;
        return;
    }

    if (size == MAX_SIZE) {
        cout << "List is full" << endl;
        return;
    }

    if (index == size) {
        // If the index is equal to the current size,
        // simply add the item to the end of the list
        data[size++] = item;
    } else {
        // Shift elements to make space for the new item
```

task1.cpp

```cpp

    if (size == MAX_SIZE) {
        cout << "List is full" << endl;
        return;
    }

    if (index == size) {
        // If the index is equal to the current size,
        // simply add the item to the end of the list
        data[size++] = item;
    } else {
        // Shift elements to make space for the new item
        for (int i = size; i > index; i--) {
            data[i] = data[i - 1];
        }
        // Insert the new item at the specified index
        data[index] = item;
        size++;
    }
}

// Function to remove an item from the list at a specified index
void List::removeFromList(int index) {
    if (index < 0 || index >= size) {
        cout << "Invalid index" << endl;
        return;
    }

    for (int i = index; i < size - 1; i++) {
        data[i] = data[i + 1];
    }
    size--;
}

// Function to get the size of the list
int List::listSize() {
    return size;
}

// Function to check if the list is empty
bool List::isEmpty() {
    return size == 0;
}
```

```
71  // Function to check if the list is empty
72  bool List::isEmpty() {
73      return size == 0;
74  }
75
76  // Function to get an item from the list at a specified index
77  int List::getItem(int index) {
78      if (index < 0 || index >= size) {
79          cout << "Invalid index" << endl;
80          return -1; // Return a default value indicating error
81      }
82      return data[index];
83  }
84
85  // Function to get the index of the last element in the list
86  int List::end() {
87      return size;
88  }
89
90  // Function to get the first element of the list
91  int List::start() {
92      if (isEmpty()) {
93          cout << "List is empty" << endl;
94          return -1; // Return a default value indicating error
95      }
96      return data[0];
97  }
98
99  int main() {
100     List myList;
101
102     int choice;
103     int item, index;
104
105     do {
106         cout << "1. Add an item to the list" << endl;
107         cout << "2. Remove an item from the list" << endl;
108         cout << "3. Display list size" << endl;
109         cout << "4. Check if the list is empty" << endl;
110         cout << "5. Get an item from the list" << endl;
111         cout << "6. Get the first element of the list" << endl;
112         cout << "7. Exit" << endl;
113         cout << "Enter your choice: ";
114         cin >> choice;
```

```
        cout << "5. Get an item from the list" << endl;
        cout << "6. Get the first element of the list" << endl;
        cout << "7. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1:
                cout << "Enter the item to add: ";
                cin >> item;
                cout << "Enter the index to add the item at: ";
                cin >> index;
                myList.addToList(item, index);
                break;
            case 2:
                cout << "Enter the index of the item to remove: ";
                cin >> index;
                myList.removeFromList(index);
                break;
            case 3:
                cout << "List size: " << myList.listSize() << endl;
                break;
            case 4:
                cout << "Is the list empty? " << (myList.isEmpty() ? "Yes" : "No") << endl;
                break;
            case 5:
                cout << "Enter the index of the item to get: ";
                cin >> index;
                cout << "Item at index " << index << ": " << myList.getItem(index) << endl;
                break;
            case 6:
                cout << "First element: " << myList.start() << endl;
                break;
            case 7:
                cout << "Exiting..." << endl;
                break;
            default:
                cout << "Invalid choice. Please enter a number between 1 and 7." << endl;
        }
    } while(choice != 7);

    return 0;
}
```

Figure 1: linked list

Output:

```
1. Add an item to the list
2. Remove an item from the list
3. Display list size
4. Check if the list is empty
5. Get an item from the list
6. Get the first element of the list
7. Exit
Enter your choice: 1
Enter the item to add: 6
Enter the index to add the item at: 0
1. Add an item to the list
2. Remove an item from the list
3. Display list size
4. Check if the list is empty
5. Get an item from the list
6. Get the first element of the list
7. Exit
Enter your choice: 1
Enter the item to add: 7
Enter the index to add the item at: 0
1. Add an item to the list
2. Remove an item from the list
3. Display list size
4. Check if the list is empty
5. Get an item from the list
6. Get the first element of the list
7. Exit
Enter your choice: 2
Enter the index of the item to remove: 7
Invalid index
```

```
7. Exit
Enter your choice: 5
Enter the index of the item to get: 0
Item at index 0: 7
1. Add an item to the list
2. Remove an item from the list
3. Display list size
4. Check if the list is empty
5. Get an item from the list
6. Get the first element of the list
7. Exit
Enter your choice:
```

TASK 2:

Debug following code.

```
/* array implementation of list adt */
#include &lt;stdio.h&gt;
#include &lt;math.h&gt;
#include &lt;string.h&gt;
#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-
&gt;last+1); }
/*** empty a list ***/ window_type
empty(list_type *list) { list-&gt;last
= -1; return(end(list));
}
```

```
/*** test to see if a list is empty ***/
int is_empty(list_type *list) { if
(list-&gt;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) { return(list-
&gt;last);}
```

CODE:

task2.cpp

```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;

/* Forward declaration */
WINDOW_TYPE last(LIST_TYPE *list);

/* 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);
}
```

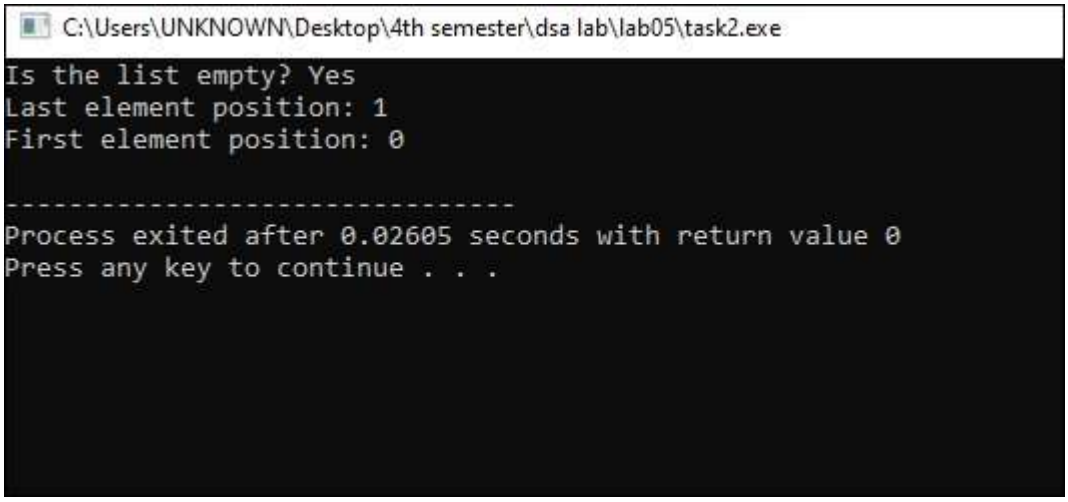
task2.cpp

```
27  }
28
29  /* empty a list */
30  WINDOW_TYPE empty(LIST_TYPE *list) {
31      list->last = -1;
32      return (end(list));
33  }
34
35  /* test to see if a list is empty */
36  int is_empty(LIST_TYPE *list) {
37      if (list->last == -1)
38          return (TRUE);
39      else
40          return (FALSE);
41  }
42
43  /* position at first element in a list */
44  WINDOW_TYPE first(LIST_TYPE *list) {
45      if (is_empty(list) == FALSE) {
46          return (0);
47      } else {
48          return (end(list));
49      }
50  }
51
52  /* position at next element in a list */
53  WINDOW_TYPE next(WINDOW_TYPE w, LIST_TYPE *list) {
54      if (w == last(list)) {
55          return (end(list));
56      } else if (w == end(list)) {
57          printf("Can't find next after end of list\n");
58          return (-1); // Return an error value
59      } else {
60          return (w + 1);
61      }
62  }
63
64  /* position at previous element in a list */
65  WINDOW_TYPE previous(WINDOW_TYPE w, LIST_TYPE *list) {
66      if (w != first(list)) {
67          return (w - 1);
68      } else {
```

task2.cpp

```
58          return (-1); // Return an error value
59      } else {
60          return (w + 1);
61      }
62  }
63
64  /* position at previous element in a list */
65  WINDOW_TYPE previous(WINDOW_TYPE w, LIST_TYPE *list) {
66      if (w != first(list)) {
67          return (w - 1);
68      } else {
69          printf("Can't find previous before first element of list\n");
70          return (w); // Return current window position
71      }
72  }
73
74  /* position at last element in a list */
75  WINDOW_TYPE last(LIST_TYPE *list) {
76      return (list->last);
77  }
78
79  int main() {
80      // Example usage
81      LIST_TYPE myList;
82      empty(&myList);
83
84      printf("Is the list empty? %s\n", is_empty(&myList) ? "Yes" : "No");
85
86      // Add elements to the list
87      myList.a[0].number = 10;
88      myList.a[0].string = "First";
89      myList.last++;
90
91      myList.a[1].number = 20;
92      myList.a[1].string = "Second";
93      myList.last++;
94
95      printf("Last element position: %d\n", last(&myList));
96      printf("First element position: %d\n", first(&myList));
97
98      return 0;
99  }
```

OUTPUT:


C:\Users\UNKNOWN\Desktop\4th semester\dsa lab\lab05\task2.exe
Is the list empty? Yes
Last element position: 1
First element position: 0

Process exited after 0.02605 seconds with return value 0
Press any key to continue . . .