Data Structures and Algorithms

An Assignment Work Submitted for the 3rd Semester
of Bachelor of Technology

in

Computer Science and Engineering (Data Science)

by

Suman Mondal
Enrollment no.: KNU2200101

Under the guidance of

Dr.Joydeep Dutta

DEPARTMENT OF COMPUTER SCIENCE
KAZI NAZRUL UNIVERSITY
ASANSOL - 713340, WEST BENGAL

Contents

Chapter 1

Array

1.1 Create a Menu driven program using Array and implement the following options:

MENU

---

1. **Array Traversal**
   (I) Forward Traversal
   (II) Backward Traversal
2. **Searching**
3. **Sorting**
4. **Insertion**
5. **Deletion**
6. **Exit**

Source Code :

```
/*
-------------------------------------------------
Author: Suman Mondal <suman.mondal@outlook.in>
Compiler version: gcc 9.4.0
-------------------------------------------------
*/


#include <stdio.h>

//Function for displaying array
void display (int arr[], int arr_size) {
    printf ("Array is: ");
    for (int i = 0; i < arr_size; i++)
        printf (" %d", arr[i]);
}
```

```c
// Function for array traversal
void traversal (int arr[], int arr_size) {
    int n;

    printf ("For Array Traversal, there is: \n");
    printf ("1. Forward Traversal\n");
    printf ("2. Backward Traversal\n");
    printf ("Enter your choice: ");
    scanf ("%d", &n);

    switch (n) {
        case 1:
            printf ("Forward Traversed Array is: ");
            for (int i = 0; i < arr_size; i++)
                printf (" %d", arr[i]);
            break;
        case 2:
            printf ("Backward Traversed Array is: ");
            for (int i = arr_size - 1; i >= 0; i--)
                printf (" %d", arr[i]);
            break;
        default:
            printf ("------------Invalid Input-----------");

    }
}

//Function for element searching
void searching (int arr[], int arr_size) {
    int element, flag = 0, i;
    printf ("Enter the element you want to search: ");
    scanf ("%d", &element);

    for (i = 0; i < arr_size; i++) {
        if (arr[i] == element) {
            flag = 1;
            break;
        }
    }
    (flag == 1) ? printf ("Element is present at position: %d", i+1) : printf
↪   ("Element is NOT FOUND");

}

//Function for element sorting
void sorting (int arr[], int arr_size) {

  // loop to access each arr element
  for (int step = 0; step < arr_size - 1; ++step) {
```

```
    // loop to compare arr elements
    for (int i = 0; i < arr_size - step - 1; ++i) {

      // compare two adjacent elements
      // change > to < to sort in descending order
      if (arr[i] > arr[i+1]) {
        int temp = arr[i];
        arr[i] = arr[i+1];
        arr[i+1] = temp;
      }
    }
  }

  display (arr, arr_size);
}

//Function for element insertion
void insertion (int arr[], int arr_size) {
    int pos, ins_num;

    Enter_the_Position:
        printf ("In Which position you want to insert element?\n");
        scanf ("%d", &pos); //taking actual position not array index

    if (pos > arr_size || pos < 1) {
        printf ("Oops! Array position doesn't exists. Please Choose between 1
↪   to %d\n", arr_size);
        goto Enter_the_Position;
    }

    printf ("Enter the element: ");
    scanf ("%d", &ins_num);

    for (int i = arr_size-1; i >= pos-1; i--)
        arr[i+1] = arr[i];
    arr[pos-1] = ins_num;

    arr_size += 1;

    display (arr, arr_size);
}

//Function for element deletion
void deletion (int arr[], int arr_size) {
    int pos;

    Enter_the_Position:
        printf ("In Which position you want to delete element?\n");
        scanf ("%d", &pos); //taking actual position not array index

    if (pos > arr_size || pos < 1) {
```

```
        printf ("Oops! Array position doesn't exists. Please Choose between 1
↪   to %d\n", arr_size);
        goto Enter_the_Position;
    }

    for (int i = pos-1; i < arr_size; i++)
        arr[i] = arr[i+1];

    arr_size -= 1;

    display (arr, arr_size);
}

//Main Function

int main () {
    int n, arr_size, arr[50];

    //No of elements in the array
    printf ("How many elements you want to enter:");
    scanf ("%d", &arr_size);

    //Input elements in the array
    printf ("Enter the elements into the array: ");
    for (int i = 0; i < arr_size; i++)
        scanf ("%d", &arr[i]);

    display (arr, arr_size);

    while (1) {
        printf ("\n----------MENU----------\n");
        printf ("1.Traversal\n");
        printf ("2.Searching\n");
        printf ("3.Sorting\n");
        printf ("4.Element Insertion\n");
        printf ("5.Element Deletion\n");
        printf ("0.Exit\n");
        printf ("\n------------------------\n");

        printf ("Enter the choice: ");
        scanf ("%d", &n);

        if (n == 0)
            break;
        else {
            switch (n) {
                case 1:
                    traversal (arr, arr_size);
                    break;
                case 2:
                    searching (arr, arr_size);
```

```
                    break;
                case 3:
                    sorting (arr, arr_size);
                    break;
                case 4:
                    insertion (arr, arr_size);
                    break;
                case 5:
                    deletion (arr, arr_size);
                    break;
                default:
                    printf ("------------Invalid Input-----------");
            }
        }
    }
    return 0;
}
```

Program Output :

```
How many elements you want to enter:6
Enter the elements into the array: 2
3
4
8
9
7
Array is:  2 3 4 8 9 7
----------MENU----------
1.Traversal
2.Searching
3.Sorting
4.Element Insertion
5.Element Deletion
0.Exit

------------------------
Enter the choice: 1
For Array Traversal, there is:
1. Forward Traversal
2. Backward Traversal
Enter your choice: 2
Backward Traversed Array is:  7 9 8 4 3 2
```

Chapter 2

Singly Linked List

2.1 Create a program of Singly Linked List and implement the following options:

MENU

- **Create a node**
- **Insert node**
  - (I) Insert at first
  - (II) Insert at the end
  - (III) Insert between the nodes
- **Delete a Node**
- (I) Delete at first
  (II) Delete at the end
  (III) Delete between the nodes
- **Count Number of Nodes**
- **Print the Linked List**
- **Reverse the Linked List**
- **Exit**

Source Code :

```
// Create a Singly Linked List and Implement Insert, Delete, Print, Count
 ↪  and Reverse Nodes

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

// structure defination for node
struct linkedList {
    int data;
```

```
    struct linkedList *next;
};

typedef struct linkedList node;

node *head = NULL;

// function prototypes

void create (node *);
void print ();
int count ();
void insBeg ();
void insEnd ();
void insBet ();
void delBeg ();
void delBet ();
void delEnd ();
void reverse ();

// main function
int main () {
    int choice;

    while (1) {
        printf ("\n----------MENU----------\n");
        printf ("1.Create Node\n");
        printf ("2.Insert Node\n");
        printf ("3.Delete Node\n");
        printf ("4.Show Total Number of Nodes\n");
        printf ("5.Display the Linked List\n");
        printf ("6.Reverse the Linked List\n");
        printf ("0.Exit\n");
        printf ("\n------------------------------\n");

        printf ("Enter the choice: ");
        scanf ("%d", &choice);

        if (choice == 0)
            break;
        else {
            switch (choice) {
                case 1:
                    head = (node *) malloc (sizeof(node));
                    create (head);
                    break;
                case 2:
                    int ch;
                    printf ("1.Insert Node at Beginning\n");
                    printf ("2.Insert Node at Middle\n");
                    printf ("3.Insert Node at End\n");
```

```
                    printf ("Enter the choice: ");
                    scanf ("%d", &ch);
                    switch (ch) {
                        case 1:
                            insBeg ();
                            break;
                        case 2:
                            insBet ();
                            break;
                        case 3:
                            insEnd ();
                            break;
                        default:
                            printf ("\n------------Invalid
↪  Input-----------\n");
                    }
                    break;
                case 3:
                    int ch2;

                    printf ("1.Delete Node at Beginning\n");
                    printf ("2.Delete Node at Middle\n");
                    printf ("3.Delete Node at End\n");
                    printf ("Enter the choice: ");
                    scanf ("%d", &ch2);
                    switch (ch2) {
                        case 1:
                            delBeg ();
                            break;
                        case 2:
                            delBet ();
                            break;
                        case 3:
                            delEnd ();
                            break;
                        default:
                            printf ("\n------------Invalid
↪  Input-----------\n");
                    }
                    break;
                case 4:
                    printf ("\nTotal Number of node is: %d\n", count ());
                    break;
                case 5:
                    print ();
                    break;
                case 6:
                    reverse ();
                    print ();
                    break;
```

```
                default:
                    printf ("\n------------Invalid Input----------\n");
            }
        }
    }
    return 0;
}

// function for create node
void create (node *item) {
    int choice;

    printf ("Enter value for data part in node: ");
    scanf ("%d", &item->data);

    printf ("----------MENU----------\n");
    printf ("1.I want to create another node\n");
    printf ("2.I don't want to create another node\n");
    printf ("Enter the choice: ");
    scanf ("%d", &choice);

    if (choice == 1) {
        item->next = (node *) malloc (sizeof(node));
        create (item->next);
    }
    else {
        item->next = NULL;
    }
}

// function for display node data
void print () {
    node *item;

    if (head == NULL) {
        printf ("Linked List is Empty.\n");
        exit (1);
    }
    else {
        item = head;
        printf ("\n-------------------------------\n");
        while (item != NULL) {
            printf ("%d->", item->data);
            item = item->next;
        }
        printf ("\n-------------------------------\n");
    }
}

//function for count number of Nodes
int count () {
```

```
    node *item;

    int cnt = 0;
    if (head != NULL) {
        item = head;
        while (item != NULL) {
            cnt++;
            item = item->next;
        }
    }
    return cnt;
}

// function for insert node at beginning
void insBeg () {
    node *ptr;

    if (head == NULL) {
        head = (node *) malloc (sizeof(node));
        printf ("\nEnter value for data part in node: ");
        scanf ("%d", &head->data);
        head->next = NULL;
    }
    else {
        ptr = (node *) malloc (sizeof(node));
        printf ("\nEnter value for data part in node: ");
        scanf ("%d", &ptr->data);
        ptr->next = head;
        head = ptr;
    }
}

// function for insert node at the end
void insEnd () {
    node *ptr, *item;

    if (head == NULL) {
        head = (node *) malloc (sizeof(node));
        printf ("\nEnter value for data part in node ");
        scanf ("%d", &head->data);
        head->next = NULL;
    }
    else {
        ptr = (node *) malloc (sizeof(node));
        printf ("\nEnter value for data part in node ");
        scanf ("%d", &ptr->data);
        item = head;
        while (item->next != NULL) {
            item = item->next;
        }
        item->next = ptr;
```

```
        ptr->next = NULL;
    }
}

// function for insert node in between of Nodes
void insBet () {
    node *prev, *item, *ptr;
    int nodePosition, totalNode, i;

    if (head == NULL) {
        head = (node *) malloc (sizeof(node));
        printf ("\nEnter value for data part in node: ");
        scanf ("%d", &head->data);
        head->next = NULL;
    }
    else {
        printf ("\nEnter position you want to insert node: ");
        scanf ("%d", &nodePosition);

        totalNode = count ();

        if (nodePosition > 1 && nodePosition < totalNode) {
            ptr = (node *) malloc (sizeof(node));
            printf ("\nEnter value for data part in node: ");
            scanf ("%d", &ptr->data);

            item = head;
            i = 1;
            while (i < nodePosition) {
                prev = item;
                item = item->next;
                i++;
            }
            prev->next = ptr;
            ptr->next = item;
        }
        else
            printf ("\nInvalid Positon or\n Try to insert 2nd to %d Position",
↪   count());
    }
}

//function for delete node at the beginning
void delBeg () {
    node *item;

    if (head == NULL)
        printf ("\nLinked List is Empty. Nothing to Delete");
    else {
        item = head;
        head = head->next;
```

```
        item->next = NULL;
        free (item);
    }
}

//function for delete node at the end
void delEnd () {
    node *item, *prev;

    if (head == NULL)
        printf ("\nLinked List is Empty. Nothing to Delete");
    else {
        item = head;
        while (item->next != NULL) {
            prev = item;
            item = item->next;
        }
        prev->next = NULL;
        free (item);
    }
}

//function for delete node in between of Nodes
void delBet () {
    node *item, *prev;
    int nodePosition, totalNode, i;

    if (head == NULL)
        printf ("\nLinked List is Empty. Nothing to Delete");
    else {
        printf ("\nEnter Node Position.");
        scanf ("%d", &nodePosition);
        totalNode = count ();
        if (nodePosition >= 1 && nodePosition <= totalNode) {
            item = head;
            i = 1;
            while (i < nodePosition) {
                prev = item;
                item = item->next;
                i++;
            }
            prev->next = item->next;
            item->next = NULL;
            free (item);
        }
        else {
            printf ("\nInvalid Position");
        }
    }
}
```

```
// function for reverse a linked List
void reverse () {
    node *item, *prev = NULL;
    if (head == NULL)
        printf ("\nLinked List is Empty. Nothing to Delete");
    else {
        item = head;
        while (head != NULL) {
            item = item->next;
            head->next = prev;
            prev = head;
            head = item;
        }
        head = prev;
    }
}
```

Program Output :

```
➜ gcc singly_linked_list.c && ./a.out

----------MENU----------
1.Create Node
2.Insert Node
3.Delete Node
4.Show Total Number of Nodes
5.Display the Linked List
6.Reverse the Linked List
0.Exit

------------------------------
Enter the choice: 1
Enter value for data part in node: 10
----------MENU----------
1.I want to create another node
2.I don't want to create another node
Enter the choice: 1
Enter value for data part in node: 60
----------MENU----------
1.I want to create another node
2.I don't want to create another node
Enter the choice: 2

----------MENU----------
1.Create Node
2.Insert Node
3.Delete Node
4.Show Total Number of Nodes
5.Display the Linked List
6.Reverse the Linked List
0.Exit

------------------------------
Enter the choice: 5

------------------------------
10->60->
------------------------------
```

Chapter 3

Doubly Linked List

3.1 Work In Progress :)